Прикарпатський національний університет імені Василя Стефаника
Фізико-технічний факультет
Кафедра комп'ютерної інженерії та електроніки

Гаврилюк Олександр Ігорович
Oleksandr Havrylyuk

УДК 004:42

Спеціальність 123 «Комп'ютерна інженерія»

Кваліфікаційна робота
на здобуття освітнього ступеня бакалавра

Розроблення web-платформи для соціальної мережі
Development of a web platform for a social network

Науковий керівник:
доцент кафедри комп'ютерної інженерії та електроніки, Віктор ГОЛОТА
Рецензент:
д.ф.-м.н., проф. каф. матер. і новітніх технологій,Іван ЯРЕМІЙ.

Івано-Франківськ
2024

Формат	Поз.	Позначення	Найменування	К-ть	Прим.
А4			Пояснювальна записка	133	
А4					
А4					

Змн.	Арк.	№ докум.	Підпис	Дата	123.КІ-41.02		
Розробив		Гаврилюк О.І.			Специфікація	Літ.	Арк. 2 / Аркушів 1
Перевірив		Голота В.І.					
Н. Контр.							
Затвердив							

АНОТАЦІЯ

В роботі розглянуто процес розробки web-платформи для соціальної мережі. Проаналізовано існуючі рішення, визначено їх переваги та недоліки. Обґрунтовано вибір технологій для реалізації проекту: мови програмування, фреймворків, бібліотек, засобів розробки.

Описано архітектуру системи, яка складається з серверної та клієнтської частин. Серверна частина реалізована за допомогою Node.js та Express і забезпечує функціонал реєстрації/авторизації користувачів, публікації дописів, коментування, а також взаємодію з базою даних. Клієнтська частина виконана на React з використанням Redux для управління станом застосунку.

У роботі наведено діаграми класів, послідовності, компонентів та технологічну архітектуру додатку. Описано процес розгортання платформи та забезпечення її безпеки. Проведено тестування функціональних можливостей системи та проаналізовано результати.

Розроблена платформа може бути використана як базова основа для створення власної соціальної мережі із подальшим розширенням можливостей.

Ключові слова: соціальна мережа, web-платформа, Node.js, React, Redux, безпека, тестування.

ABSTRACT

The paper examines the process of developing a web platform for a social network. The existing solutions were analyzed, their advantages and disadvantages were determined. The choice of technologies for the implementation of the project is substantiated: programming languages, frameworks, libraries, development tools.

The architecture of the system, which consists of server and client parts, is described. The server part is implemented using Node.js and Express and provides the functionality of user registration/authorization, publication of posts, commenting, as well as interaction with the database. The client part is made on React using Redux to manage the state of the application.

The work provides diagrams of classes, sequences, components and technological architecture of the application. The process of deploying the platform and ensuring its security is described. Functional capabilities of the system were tested and the results were analyzed.

The developed platform can be used as a basic basis for creating your own social network with further expansion of possibilities.

Keywords: social network, web platform, Node.js, React, Redux, security, testing.

ПЕРЕЛІК ОСНОВНИХ СКОРОЧЕНЬ

API - Application Programming Interface

SPA - Single Page Application

REST - Representational State Transfer

MVC - Model-View-Controller

CRUD - Create, Read, Update, Delete

UI - User Interface

UX - User Experience

ORM - Object-Relational Mapping

JWT - JSON Web Token

HTTPS - Hypertext Transfer Protocol Secure

DBMS - Database Management System

SQL - Structured Query Language

NoSQL - Non-relational Database

CDN - Content Delivery Network

CI/CD - Continuous Integration/Continuous Deployment

Пояснювальна записка

до кваліфікаційної роботи

на тему:

«Розроблення web-платформи для соціальної мережі»

					123.КІ-41.02		
Змн.	*Арк.*	*№ докум.*	*Підпис*	*Дата*			
Розробив		Гаврилюк О.І.			. Пояснювальна записка	*Літ.*	*Арк.* 7 / *Аркушів* 133
Перевірив		Голота В.І.					
Н. Контр.							
Затвердив							

ЗМІСТ

ВСТУП

Соціальні мережі стали невід'ємною складовою сучасного суспільства, забезпечуючи користувачам широкі можливості для спілкування, обміну інформацією та розваг. За статистикою, понад 4 мільярди людей активно користуються цими платформами, що відображає їхню важливість у нашому світі. Таким чином, розробка власної соціальної мережі стає актуальною та перспективною темою.

Наразі існує велика множина соціальних мереж з різноманітними функціями та перевагами. Наприклад, Facebook налічує понад 2,8 мільярди активних користувачів, веб-сайт дозволяє створювати профілі, обмінюватися повідомленнями та публікувати власні статті та блоги. Instagram спеціалізується на візуальному змісті, а LinkedIn служить для пошуку професійних контактів та роботи. Також існують спеціалізовані платформи, як Twitter для коротких повідомлень, TikTok для поширення короткого креативного відео та Pinterest для обміну зображеннями. Більшість з цих мереж також підтримують інтеграцію з іншими сервісами, що дозволяє користувачам зручно використовувати різноманітні сервіси в одному місці.

Проте, багато соціальних мереж стикаються з проблемами безпеки та конфіденційності. Наприклад, Facebook була критикована через скандал з приватною англійською компанією, коли персональні дані користувачів були зловживані для політичної маніпуляції. Це спричинило зростання інтересу до децентралізованих соціальних мереж, таких як Mastodon та IndieWeb, які пропонують більшу приватність та безпеку, оскільки дані зберігаються розподілено на окремих серверах учасників мережі, а не на централізованих платформах. Однак, наразі ці мережі ще не набрали такої популярності та функціоналу, як централізовані аналоги.

Аналіз існуючих рішень виявив, що хоча найбільш популярні сучасні соціальні мережі, маючи мільйони користувачів та широкий спектр функцій, не завжди гарантують належний рівень конфіденційності та безпеки даних. Їх складна архітектура ускладнює впровадження новітніх технологій та інтеграцію з

іншими сервісами. Тому розробка нової платформи для соціальної мережі, яка б відповідала потребам користувачів та забезпечувала надійність, конфіденційність та зручність спілкування, є важливим кроком та актуальною задачею.

Метою цієї бакалаврської роботи є розробка платформи для соціальної мережі, що надасть користувачам можливість створювати профілі, спілкуватися, ділитися власною інформацією та взаємодіяти, зокрема, забезпечуючи високий рівень безпеки та конфіденційності даних.

Цей проект визначається головними пріоритетами, серед яких забезпечення безпеки й приватності користувачів, максимальна швидкість та зручність використання, а також ефективна інтеграція з іншими сервісами. Для досягнення цих цілей визначені наступні завдання:

- провести дсолідження потреб і функціоналу платформи соціальної мережі.
- розробити архітектуру платформи, включаючи організацію бази даних та компонент сервера.
- створити інтерфейс користувача, охоплюючи сторінки профілю, стрічки, чатів та повідомлень.
- розробити можливості для збереження безпеки й конфіденційності даних користувачів.
- реалізувати механізми безпеки й конфіденційності даних, такі як шифрування та контроль доступу до даних.
- провести тестування й оптимізацію платформи для забезпечення її ефективної роботи та покращення функціональності.

Під час розробки платформи для соціальної мережі потрібно враховувати сучані напрями та вимоги до соціальних мереж, такі як:

- продуктивність;
- швидкість;
- комфорт у використанні;
- можливість інтеграції з іншими сервісами;
- забезпечення безпеки та конфіденційності даних.

Важливо, щоб інтерфейс соціальної мережі був інтуїтивно зрозумілим та зручним для користувачів, подібно до популярних соціальних платформ. Це дає змогу швидко знаходити необхідну інформацію. Розробка зручного і простого інтерфейсу - ключовий аспект створення соціальної мережі. Крім того, інтеграція з іншими сервісами є важливою, оскільки багато користувачів бажають поєднати свої соціальні мережі з іншими платформами, такими як магазини або інші соціальні медіа. Це забезпечує зручне та ефективне використання соціальної мережі, що може сприяти її популярності серед користувачів.

Для досягнення цих цілей була розроблена архітектура платформи, яка використовує розподілену систему зберігання даних та асинхронну комунікацію між її компонентами. База даних платформи забезпечує зберігання персональних даних користувачів та їх взаємодій з можливістю шифрування та резервного копіювання даних в безпечному місці. Сервіс також пропонує функціонал для забезпечення конфіденційності та безпеки даних користувачів, зокрема контроль доступу до особистих даних, шифрування взаємодій між користувачами та можливість видалення особистої інформації з бази даних. Функціонал для взаємодії користувачів включає можливість створення профілів, додавання друзів, обмін повідомленнями, публікацію записів та можливість їхнього коментування. Інтерфейс користувача є простим та зручним у використанні з можливістю персоналізації та налаштування.

Результатом цієї роботи стала розробка платформи для соціальної мережі, яка забезпечує високий рівень приватності та безпеки даних користувачів, має розширений функціонал для взаємодії між ними та привабливий інтерфейс. Створена платформа може бути використана як засіб для спілкування, обміну досвідом, а також для вирішення різноманітних завдань, пов'язаних з обміном інформацією.

Таким чином, дана робота має велике значення у вирішенні проблеми забезпечення безпеки та конфіденційності користувачів у соціальних мережах, що є одним із найважливіших аспектів в сфері IT-технологій. Крім того, розробка цієї

платформи сприятиме розширенню вибору користувачів у соціальних мережах і підвищить конкуренцію на цьому ринку послуг.

У даному дослідженні розглянуті сучасні стратегії та інноваційні технології, які використовуються для створення соціальних мереж. Особлива увага приділялась методам розподіленого забезпечення, будові програмного забезпечення, базам даних, протоколам передання даних, технологіям шифрування та інтерфейсам для користувачів. Також детально вивчені новаторські підходи до забезпечення конфіденційності та безпеки даних у соціальних мережах, а також методи для боротьби з шахрайством та надмірною активністю.

На основі розробленої будови програми було створена веб-платформа для соціальної мережі. Здійснено наступні кроки:

1. Втілення функціоналу платформи: створені модулі для керування користувачами, прийняття та видалення друзів, створення та редагування публікацій, обмін повідомленнями та інші функції.

2. Тестування системи: розроблені функціональні тести для перевірки працездатності роботи модулів. Проведено тестування прийняття та видалення користувачів, методу дружби, створення публікацій та коментарів, надсилання повідомлень.

3. Оцінка продуктивності: проведено повноцінне тестування продуктивності мережі з багатьма видами налаштувань.

4. Аналіз економічної вигоди: здійснено аналіз витрат на створення, підтримку та розширення потужностей системи, а також потенційних переваг від використання платформи, таких як залучення людей, поширення активності та реклама.

Виконані кроки під час аналізу та створення веб-платформи для соціальної мережі дозволили досягти поставлених цілей та висвітлити ефективне вирішення проблем схожих програмних забезпечень.

Отже, цей документ описує структуру та функції соціальної мережі, а також аналізується процес її створення з використанням оновлених технологій. В

дослідженні розглядаються вимоги до проекту та його архітектури, вибір технологічних засобів розробки, опис реалізації і випробування. Також розглядаються можливості збільшення проекту та його масштабування. Узагальнюючи, дослідження містить всі необхідні аспекти для створення ефективної соціальної мережі, яка здатна привернути увагу користувачів та задовольнити їхні потреби у спілкуванні та поширенні інформації.

РОЗДІЛ 1. АНАЛІЗ ТЕМАТИКИ БАКАЛАВРСЬКОЇ РОБОТИ.

1.1. Історія виникнення соціальних мереж.

З самого початку історії розвитку соціальних мереж передбачалася їхня комунікативно-інформаційна роль. У 1954 році термін "соціальна мережа" з'явився за далеко до настання епохи Інтернету і походив від поняття тісних взаємин між двома або більше особами. Це поняття було введено британським науковцем Джеймсом Барнсом для опису соціальних зв'язків, що відрізнялися від звичних соціологічних концепцій, таких як обмежені спільноти (племена, родини) або соціальні групи (стать, етнічна приналежність). Новітнє значення соціальної мережі зводиться до того, що вона представляє собою структуру, побудовану на зв'язках між людьми та їх взаємних бажань. Як інтернет-сервіс, соціальна мережа є програмою, що допомагає індивідуумам спілкуватися та класифікуватися за конкретними інтересами. Основна ціль таких платформ полягає в наданні користувачам різних можливостей взаємодії, таких як відеоконференції, чати, фотографії, музика, блоги тощо. Всесвітня соціальна мережа є мережею, що сполучає велику кількість особистостей та груп, незалежно від їхніх характеристик, уподобань чи місця проживання [1].

Теорія спільнот у мережах аналізує взаємодії між особами та групами, використовуючи поняття вузлів та зв'язків. Вузли відображають окремих учасників у мережі, а зв'язки відтворюють відносини між ними. Дослідження соціальних мереж є критичним, оскільки забезпечує значну інформацію про учасників соціальних груп та їх міжособисті взаємини. Соціометрія займається аналізом цих даних та оцінює соціальні зв'язки. Наприклад, дослідження свідчать, що у організаціях вплив мають особи з великою кількістю соціальних зв'язків, а не тільки особи з вищим статусом. Соціальні мережі впливають на продуктивність роботи, успішність компаній та прийняття кадрових рішень. Раніше створення такої системи було неможливим у фізичному плані через відсутність необхідних технічних можливостей, але з появою комп'ютерів та глобальної мережі Інтернет соціальні мережі отримали відповідність [1].

Першими онлайн-сервісами, що відкривали можливості для спільної діяльності у соціальних мережах, були clasmates.com (1994) і SixDegrees.com (1998). Починаючи з 2001 року, з'явилися ресурси, які оперували принципом "Кола друзів". Цей формат соціальних мереж набув популярності у 2001 році. На сьогоднішній день існує понад 250 веб-платформ для створення соціальних спільнот [1].

4 лютого 2004 року Марк Цукерберг запустив "The facebook", початково розташований на facebook.com. Перший доступ до платформи був обмежений лише для студентів Гарвардського університету, і протягом першого місяця зареєструвалися більше половини студентів. Згодом до команди Цукерберга приєдналися Едуард Саверін (бізнес-аналітик), Джастін Московіц (розробник), Ендрю Мак-Колум (дизайнер) і Кріс Хюз для підтримки розвитку проекту. Вже в березні 2004 року Facebook став популярним серед студентів інших університетів, таких як Стенфорд, Колумбія та Єльський. Згодом він став доступним і для студентів інших університетів, таких як Массачусетський технологічний інститут, а потім і для багатьох інших вузів у США та Канаді. Компанію "Facebook" заснували влітку 2004 року, а очолив її підприємець Шон Паркер, який був консультантом Цукерберга. У червні 2004 року компанія Facebook переїхала до Пал-Альто в Каліфорнії. 26 вересня 2006 року Facebook став доступний для всіх користувачів віком від 13 років і старше з дійсною електронною адресою. На сьогоднішній день майже третина світового населення, включаючи користувачів мобільних пристроїв, використовує цю платформу.

У процесі, веб-сайти набували все більшого розповсюдження, і в 2005 році My Space мав більше переглядів сторінок, ніж Google. Крім того, Google випустив веб-сайт із функціями соціальних мереж, orkut, який був представлений у 2004 році. У той самий період соціальні мережі стали важливою частиною інтернет-стратегій: у березні 2005 року Yahoo запустив Yahoo! 360°, а в липні того ж року News Corporation вивела на ринок My Space. 1 жовтня 2006 року була випущена відома російська соціальна мережа ВКонтакте, яка здобула велику популярність в Україні, Білорусі та Казахстані [1].

Однією з найвідоміших українських соціальних мереж є Connect.ua, випущена у кінці 2007 року. Ця мережа дозволяє користувачам спілкуватися, обмінюватися вмістом, створювати чати та знаходити нових користувачів. Пізніше Connect.ua характеризувалася як хороший, відомий сервіс в нашій онлайн-спільноті [1].

У сучасному світі система запрошень активно використовується у соціальних мережах, дозволяючи початковій групі користувачів розширювати своє коло знайомств шляхом відправлення запрошень своїм друзям. Цей підхід сприяє зростанню кількості учасників та зв'язків у мережі. Додатково, багато платформ пропонують функцію автоматичного оновлення списків контактів, що дозволяє користувачам легко визначати знайомих і додавати їх до свого соціального кола. Користувачі також можуть переглядати особисті дані одне одного, встановлювати нові контакти через різні сервіси, включаючи сайти знайомств та інші засоби соціальної взаємодії. Соціальні мережі можуть мати різні спрямування, наприклад, Linked In зосереджений на професійних контактах, тоді як інші платформи, такі як My Space та Facebook, підтримують спільноти, що базуються на музичних інтересах, святкових подіях або академічних зв'язках. Також існують соціальні мережі, що спеціалізуються на певних захопленнях або професійних сферах, таких як мистецтво, спорт, автомобілі або навіть естетична хірургія. Соціальні мережі грають ключову роль у сучасному спілкуванні, забезпечуючи платформу для обміну інформацією та ідеями, як у масштабі великих груп, так і для особистого спілкування між індивідами [1].

1.2. Загальні відомості про соціальні мережі та їх роль у сучасному світі.

Соціальні мережі представляють собою цифрові платформи, які надають можливість користувачам встановлювати зв'язок, обмінюватися інформацією та взаємодіяти в онлайн-просторі. Вони радикально змінили методи комунікації та взаємодії між людьми, ставши ключовим елементом сучасного суспільства.

Основна функція соціальних мереж полягає у стимулюванні соціальної активності та співробітництва серед користувачів. Ці платформи спрощують процес знайомства з новими особами, підтримки зв'язку з друзями та колегами,

обміну думками та інформацією, обговорення спільних інтересів та ознайомлення з різними культурами.

Важливою перевагою соціальних мереж є їх здатність підвищувати обізнаність щодо соціальних проблем та актуальних подій. Користувачі мають можливість ділитися новинами та інформацією про події, що відбуваються у світі, а також приєднуватися до спільнот, які підтримують конкретні соціальні ініціативи.

Крім того, соціальні мережі відіграють значну роль у сфері бізнесу та маркетингу. Вони використовуються компаніями для просування продукції та послуг, приваблення нових клієнтів та взаємодії з існуючою клієнтурою та бізнес-партнерами.

Загалом, соціальні мережі є невід'ємною складовою нашого повсякденного життя, пропонуючи величезний потенціал для розвитку нових технологій та інноваційних рішень. Вони сприяють створенню нових сервісів та продуктів, які ведуть до покращення якості життя населення.

Платформи соціальних медіа можуть слугувати потужним засобом для адресації соціальних викликів, включаючи сфери освіти, медицини та захисту навколишнього середовища. Вони здатні надавати підтримку у створенні та розповсюдженні електронних освітніх програм та матеріалів, які будуть доступні широкому колу зацікавлених осіб. Крім того, ці платформи можуть служити місцем для розробки веб-сервісів, що надають консультації та підтримку особам, які стикаються з медичними проблемами.

Одним із ключових переваг соціальних медіа є їхня здатність сприяти комунікації та колаборації між особами з різних частин планети. Це дозволяє користувачам формувати спільноти та групи для обговорення різноманітних тем, спільно працювати над проектами, а також обмінюватися ідеями та знаннями.

Таким чином, соціальні медіа відіграють важливу роль у багатьох аспектах сучасного життя, від особистісних взаємин до професійної діяльності та вирішення глобальних соціальних викликів. Створення та розвиток соціальних

медіа платформ може стати ключовим елементом у прогресі цих сфер та у вирішенні нагальних проблем.

1.3. Класифікація соціальних мереж.

Для повного розуміння соціальних мереж та їх впливу на суспільство, важливим є аналіз їх різновидів. Класифікація соціальних мереж сприяє упорядкуванню їх за певними критеріями, що допомагає краще зрозуміти їх функції та цілі. Це, в свою чергу, дозволяє нам оцінити їх вплив на різні аспекти нашого життя.

У цьому огляді розглянемо основні типи соціальних мереж, такі як універсальні соціальні платформи, спеціалізовані професійні мережі, платформи для обміну візуальним контентом, системи мікроблогінгу, медіа-орієнтовані соціальні мережі, закриті соціальні мережі та інші. Кожен з цих типів має унікальні характеристики та призначення, що впливає на їх популярність та способи використання користувачами.

Розподіл соціальних платформ за категоріями:

1. Універсальні соціальні платформи:
    - Опис: Ці платформи надають можливість користувачам створювати особисті акаунти, знаходити нових друзів і ділитися різноманітним контентом, включаючи текст, зображення та відео.
    - Приклад: TikTok, Intsagram, Linked In.
2. Спеціалізовані професійні платформи:
    - Опис: Ці платформи спрямовані на спілкування серед професіоналів і дозволяють створювати професійні акаунти, розміщувати резюме, шукати вакансії та обговорювати спеціалізовані теми.
    - Приклад: LinkedIn.
3. Платформи для обміну візуальним контентом:
    - Опис: Ці платформи зосереджені на обміні зображеннями та відео.
    - Приклади: Tumblr, Reddit.
4. Платформи мікроблогінгуі:

- Опис: Ці платформи дозволяють користувачам публікувати короткі записи або "tweet", обмінюватися новинами, думками та посиланнями.
- Приклад: Twitter.

5. Медіа-орієнтовані соціальні платформи:
    - Опис: Ці платформи спеціалізуються на розповсюдженні медійного контенту, такого як відео, музика, подкасти та фільми.
    - Приклад: You Tube, Vimeo.
6. Закриті соціальні платформи:
    - Опис: Ці платформи забезпечують приватне обмін текстовими повідомленнями, зображеннями та іншим контентом, доступним лише обмеженому колу осіб.
    - Приклад: Wedex, Wickr.
7. Соціальні мережі на базі блокчейну:

Опис: Використання блокчейн-технології у цих соціальних мережах гарантує високий рівень захищеності, прозорість дій та можливість користувачам самостійно контролювати свої дані. Такі платформи створюють унікальні простори для обміну думками та досвідом між фахівцями з різних галузей.

Приклади:

1. Aleo: Платформа, що базується на технології блокчейну та розроблена для програмістів. Вона сприяє спільній роботі над кодом, контролю версій та обговоренню технічних аспектів проектів.
2. Farcaster: Мережа на блокчейні, орієнтована на дизайнерів, де вони можуть демонструвати свої творчі роботи, отримувати відгуки та взаємодіяти з колегами.
3. Meta: Спеціалізована блокчейн-мережа для взаємодії між стартапами та інвесторами, що допомагає стартапам презентувати свої проекти та знаходити фінансування, а інвесторам - знаходити перспективні інвестиційні можливості.

Блокчейн-базовані соціальні платформи створюють неповторні простори для спільної роботи, обміну знаннями та взаємодії серед професіоналів різних сфер або індустрій. Вони відіграють ключову роль у покращенні якості професійної діяльності, сприяють спільному прогресу та відкривають нові перспективи у відповідних областях.

1.4. Розгляд існуючих соцмереж та аналіз їхніх плюсів та мінусів.

Соціальні платформи є невід'ємною частиною сучасного життя. Їх існує безліч форм, проте деякі з них вирізняються більшою популярністю.

1.4.1. Дослідження відомих соціальних платформ та вивчення їхніх характеристик.

Соціальні платформи активно використовуються у сучасному суспільстві. Вони представлені у різноманітних формах, але деякі з них є більш відомими порівняно з іншими.

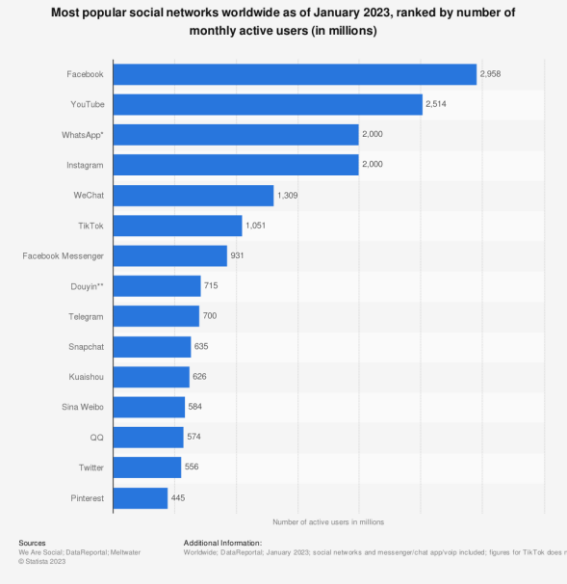

Рис.1.1. Дані найпопулярніших соціальних мереж[4].

1. **Facebook** - 2,95 мільярда активних користувачів.

Платформа є лідером серед соціальних мереж, залучаючи близько 36,9% населення землі щомісяця. Ця платформа є домом для понад семи мільйонів рекламодавців, які активно рекламують свої товари та послуги. Найбільшу увагу на Facebook привертають новини з різних галузей, короткі відеоролики та візуально привабливий контент [4].

2.**Youtube -** 2,51 мільярда активних користувачів.

YouTube - виступає не лише як платформа для обміну відео, де щодня переглядається мільярд годин контенту, але й як друга за популярністю пошукова система після Google. Ця платформа користується популярністю серед усіх вікових груп, з легким переважанням чоловічої аудиторії над жіночою.

3. **Whats App -** 2 мільярда активних користувачів.

Whats App зі своїми 2 мільярдами активних користувачів, залишається на вершині серед додатків для обміну повідомленнями, об'єднуючи людей з понад 180 країн. Платформа неустанно розширює свої функції, спрощуючи комунікацію між користувачами та бізнесом.[4].

4. **Instagram -** 1 мільярд активних користувачів.

Instagram - налічує понад 1 мільярд активних користувачів, продовжує швидко зростати, особливо серед молоді, завдяки своєму візуальному формату подання контенту, включаючи відео та зображення. Платформа стала значущим каналом для B2B-брендів завдяки останнім оновленням [4].

5. **TikTok –** 1,51 мільярда активних користувачів.

За останні роки TikTok швидко завоював популярність у всьому світі завдяки своєму розважальному та креативному контенту. Платформа, що дозволяє користувачам створювати короткі відеоролики, стала однією з найвідоміших соціальних мереж [4].

1.4.2. Переваги соціальних мереж.

1. Соціальні мережі розкривають можливості для міжнародного спілкування. Основною перевагою цих платформ є можливість встановлення контактів з особами з різних куточків світу. Додавання у друзі або підписка на користувача дозволяє підтримувати зв'язок у будь-який момент, за умови доступу до Інтернету та активного профілю. Люди використовують такі платформи, як Facebook, для спілкування з родиною, відновлення знайомств і вираження своїх думок. Twitter стає місцем для обміну жартами, актуальних новин та особистих спостережень, дозволяючи взаємодіяти навіть з тими, хто знаходиться далеко за межами звичного оточення.[2].

2. Миттєвий та зручний обмін інформацією.

Соціальні мережі дозволяють нам залишатися на зв'язку незалежно від місцезнаходження. Ці платформи дозволяють спілкуватися без необхідності в традиційних засобах зв'язку, таких як стаціонарний телефон. Відправлення листів поштою тепер здається застарілим; досить лише залишити коментар на сторінці користувача, щоб він миттєво отримав повідомлення [2].

3. Соціальні мережі як інструмент для просування бізнесу.

Компанії мають можливість взаємодіяти зі своєю цільовою аудиторією, використовуючи соціальні мережі для спілкування з поточними та потенційними клієнтами. Багато платформ дозволяють прямий продаж товарів або розміщення посилань на онлайн-магазини, спрощуючи процес привертання нових клієнтів. Це особливо корисно для малих підприємств та індивідуальних підприємців, які спираються на контент, створений у соціальних мережах, для розвитку свого бізнесу [2].

4. Зменшує вартість маркетингу.

Зниження витрат на маркетинг. Використання соціальних мереж для маркетингових кампаній може значно зменшити витрати на рекламу. Платна реклама на таких платформах, як Facebook, часто виявляється більш вигідною з економічної точки зору порівняно з традиційними медіа, такими як радіо або телебачення, при цьому охоплюючи більшу аудиторію. Для нових малих підприємств з обмеженим бюджетом на маркетинг соціальні мережі стають невід'ємним інструментом, дозволяючи зменшити або навіть повністю відмовитися від платної реклами, акцентуючи увагу на створенні та поширенні контенту.

Поліпшення цільової реклами у соціальних мережах дозволяє підприємствам ефективно звертатися до потенційних клієнтів. Наприклад, виробник дитячих іграшок може спрямовувати свої просувальні компанії на молодих батьків. Це особливо корисно для нових проектів та малих бізнесів з обмеженими маркетинговими ресурсами [2].

5. Соціальні мережі також є відмінним освітнім ресурсом.

Вони спрощують процес навчання, з'єднуючи вчителів, фахівців та учнів різного віку. Вони дозволяють студентам взаємодіяти з викладачами та продовжувати освіту у комфортному темпі, що робить навчальний процес більш гнучким.

6. Крім того, соціальні мережі є ідеальним місцем для обміну різноманітною інформацією.

Вони стають платформою для професіоналів та митців, які бажають поділитися своїми творами, такими як музика, поезія, художні твори та кулінарні шедеври, з широким загалом [2].

Соціальні мережі допомагають розширити творчі горизонти та збудувати бренд, надаючи можливість привертати увагу мільйонів потенційних клієнтів безпосередньо. Це одна з ключових переваг для бізнесу, оскільки ви можете збільшити зацікавленість у своїх творчих проектах, зміцнюючи ваш творчий вплив [2].

7. Соціальні платформи сприяють інтеграції людей похилого віку у суспільство, надаючи їм можливість відчувати себе залученими та активними.

Згідно з аналізом PawResearch Center у 2014 році, особи віком 60 років і старше, які активно використовують соціальні мережі, виявили значне задоволення від онлайн-спілкування. Це дало їм можливість залишатися на зв'язку з родиною, отримувати фотографії та відео від онуків, а також бути в курсі подій у своїй церковній громаді. У 2007 році лише менше 3% людей похилого віку використовували соціальні мережі, але до 2014 року ця цифра зросла до 34%, що свідчить про швидке прийняття цієї технології серед старших груп населення.[2].

1.4.3. Недоліки соціальних мереж.

1. Залежність.

Активне використання соціальних мереж може стати причиною залежності серед користувачів, зокрема серед молоді. Ця залежність від онлайн-взаємодії та постійного оновлення новин може відволікати від реального життя, змушуючи людей проводити занадто багато часу в Інтернеті. У деяких випадках така поведінка може призвести до негативних наслідків [2].

2. Вплив на здоров'я.

Крім того, надмірне використання соціальних мереж може викликати психологічні проблеми, такі як тривожність, стрес, депресія та інші емоційні розлади. Дорослі можуть відчувати погіршення свого емоційного стану через залежність від соціальних мереж. Крім того, тривале користування електронними пристроями може негативно впливати на здоров'я очей та порушувати нормальний сон через вплив штучного світла. Більше того, тривале сидіння перед екраном може мати негативний вплив на фізичне здоров'я, зменшуючи активність та сприяючи сидячому способу життя. [2].

3. Відсутність фізичних вправ.

Часте використання онлайн-медіа може призвести до зменшення бажання вести особисті розмови. Створення профілю в Інтернеті та висловлення думок онлайн стає простішим, ніж реальні зустрічі з друзями або родиною. Це може змінювати традиційні способи спілкування та мати негативний вплив на відносини з близькими [2].

4. Зростання кіберзлочинності.

Багато соціальних мереж стають ареною для кіберзлочинності, включаючи харасмент, шантаж, незаконне стеження та інші види злочинів. Це створює ризики для безпеки користувачів, особливо для дітей, які можуть стати легкими мішенями для зловмисників. Анонімність в Інтернеті та легкість створення фальшивих профілів ускладнюють ідентифікацію та притягнення до відповідальності злочинців. [2].

5. Хакінг.

Зловмисники часто використовують соціальні мережі як канали для проникнення в комп'ютерні системи користувачів з метою крадіжки конфіденційної інформації для використання у шахрайських схемах. Користувачі, які активно управляють своїми онлайн-профілями, часто не усвідомлюють ризики, пов'язані з розміщенням особистих даних, фотографій та інформації про своє життя. Ця інформація може стати простою мішенню для зловмисників, які намагаються взламати їхні акаунти. Важливо бути обережними та застосовувати

заходи безпеки при керуванні своїми онлайн-акаунтами, щоб уникнути таких загроз [2].

6. Безпека.

Проблеми з безпекою виникають через те, що багато платформ для особистого спілкування дозволяють користувачам розміщувати інформацію публічно. Це знижує рівень захисту даних, оскільки користувачі не завжди можуть контролювати, хто переглядає їхні профілі та отримує доступ до особистої інформації. Хоча соціальні мережі стараються забезпечити захист даних своїх користувачів, повна безпека не може бути гарантована.

7. Впливати на якість сну.

Крім того, соціальні мережі можуть негативно впливати на якість сну. Використання технологій перед сном може змусити мозок віддавати перевагу активності в соціальних мережах замість відпочинку. Це може призвести до збільшення тривожності через постійне занепокоєння про діяльність інших у соціальних мережах або стан власного профілю. Також синє світло від екранів може порушувати природний цикл сну, роблячи засинання складнішим.

8. Неправдива або помилкова інформація має тенденцію швидко поширюватися в соціальних мережах.

Згідно з дослідженням серед журналістів традиційних ЗМІ, понад 78% з них використовують соціальні мережі для пошуку останніх новин, що вимагає критичного переосмислення, оскільки важливо перевіряти факти перед їх поширенням. Це особливо актуально в умовах, коли інформація, яка підтверджує існуючі упередження, поширюється швидше, ніж об'єктивні дані. У Twitter, наприклад, неправдиві новини розповсюджуються в шість разів швидше, ніж достовірні. Тому критичне мислення та перевірка інформації є ключовими для запобігання поширенню неправдивих новин [2].

9. Проблеми з психікою.

Дослідження Університету Піттсбурга виявило, що використання соціальних мереж може бути пов'язане з психологічними проблемами, такими як депресія, особливо серед молодих людей віком від 18 до 33 років. Соціальні мережі

також можуть сприяти розвитку нарцисизму та інших особистісних розладів, а також викликати проблеми з увагою та гіперактивність. [2].

Отже, хоча соціальні мережі відіграють важливу роль у підтримці зв'язків між людьми та пропонують численні переваги, важливо пам'ятати про їх потенційні недоліки. Модерація та обережне використання соціальних мереж можуть допомогти збалансувати їх позитивний вплив з можливими ризиками. Використання соціальних мереж вимагає від нас відповідальності та усвідомлення, щоб забезпечити, що вони служать нашому благу, а не стають джерелом проблем.

1.6. Огляд останніх досягнень у сфері розробки соціальних мереж підкреслює важливу роль веб-програмування.

Цей підхід дозволяє створювати веб-сайти, які можна відвідати через різні веб-переглядачі, сприяючи зручності для користувачів.

У процесі розробки соціальних мереж застосовуються різні мови програмування, такі як PHP, Java Script, Python, а також різноманітні системи управління базами даних, такі як My SQL, Postgre SQL, Mongo DB.

Серед інноваційних підходів у розробці соціальних мереж варто відзначити використання API (інтерфейсу програмування додатків), що дає можливість інтегрувати соціальні мережі з іншими онлайн-сервісами та додатками, розширюючи їх функціональність. Крім того, у розробці соціальних мереж все частіше залучаються технології штучного інтелекту, включаючи машинне навчання та нейронні мережі, що підвищує точність рекомендацій та прогнозування поведінки користувачів.

Розробка соціальних мереж потребує використання широкого спектру технологій та інструментів для створення масштабованих та безпечних платформ. Основні використовувані технології включають::

1. Фронтенд: HTML, CSS і JavaScript використовуються для створення інтерфейсу користувача, що дозволяє спілкуватися з соціальною мережею.
2. Бекенд: Мови програмування, такі як PHP, Ruby, Python, та Node.js, використовуються для розробки логіки на сервері та обробки даних користувачів.

3. Бази даних: MySQL, PostgreSQL, MongoDB використовуються для ефективного зберігання та управління даними користувачів.
4. Хмарні технології: Для забезпечення високої продуктивності та доступності соціальних мереж використовуються хмарні платформи, такі як Amazon Web Services (AWS), Microsoft Azure, та Google Cloud Platform (GCP).
5. Машинне навчання та аналітика даних: Вони застосовуються для оптимізації рекомендацій та цільової реклами, використовуючи мови програмування Python та R для обробки та аналізу даних.
6. АПІ: Інтерфейси програмування додатків є важливими для створення додатків, які інтегруються з соціальними мережами, надаючи доступ до їх даних та функціоналу.

Штучний інтелект набуває ключового значення у розробці соціальних мереж, дозволяючи аналізувати об'ємні набори даних, зібрані з платформ, для виявлення цінної інформації, такої як тенденції, спільні інтереси користувачів та моделі поведінки. За допомогою цієї технології, аналітика даних стає більш точною та оперативною, сприяючи ефективнішій взаємодії з користувачами.

Розробники соціальних мереж також використовують різні програмні фреймворки та інструменти для підвищення швидкості та надійності платформ. Наприклад, фреймворки, засновані на Java Script, такі як React та Angular, дозволяють розробникам ефективно створювати багатофункціональні та інтерактивні інтерфейси, поліпшуючи користувацький досвід. Інструменти, такі як Redis та Cassandra, використовуються для зберігання та обробки великих обсягів даних, що є критично важливим для соціальних мереж з великою кількістю користувачів.

Веб-розробка лишається однією з основних технологій у створенні соціальних мереж, використовуючи HTML, CSS та Java Script для створення користувацького інтерфейсу, а також різні бібліотеки та фреймворки, такі як React або Angular, для реалізації функціоналу. Ця технологія має переваги, такі як широке застосування та доступність готових рішень, але може мати й недоліки, включаючи потенційні затримки в роботі та залежності.

Мобільні платформи, такі як iOS та Android, або розробка хмарних додатків, пропонують альтернативні підходи до створення соціальних мереж, забезпечуючи вищу швидкість та надійність, а також доступ до спеціалізованих бібліотек та інструментів. Ці технології можуть вимагати більших інвестицій та бути складнішими у розробці.

React, бібліотека JavaScript від Facebook, є популярним вибором для створення односторінкових додатків, забезпечуючи швидкий та ефективний інтерфейс користувача. За допомогою різноманітних бібліотек та інструментів, таких як Redux для управління станом додатку та Axios для виконання запитів на сервер, React допомагає створювати високопродуктивні веб-додатки.

Хоча React безперечно вважається однією з найпопулярніших бібліотек для розробки веб-додатків, він також має свої недоліки, які важливо враховувати. Один з таких недоліків полягає в складності у вивченні та розумінні для новачків. Це пояснюється тим, що React пропонує компонентний підхід до розробки, що може виявитися неочевидним без попереднього досвіду роботи з подібними концепціями.

1.7. Аналіз головних функцій, які має бути в платформі соціальної мережі.

Платформа соціальної мережі є цифровим простором, де користувачі можуть взаємодіяти та обмінюватися інформацією в онлайн-режимі. Головна мета такої платформи полягає у сприянні комунікації між користувачами та підтримці соціальної активності в мережі.

Для досягнення цієї мети, платформа повинна включати наступні ключові функції та можливості:

1. Управління профілями користувачів: Можливість створювати та налаштовувати особисті профілі з розгорнутою інформацією про себе, включаючи фотографії та інші особисті дані.

2. Засоби комунікації: Надання засобів для обміну повідомленнями, коментарів, вираження вподобань та інших форм взаємодії.

3. Публікація контенту: Можливість користувачів ділитися різноманітним контентом, таким як фотографії, відео, статті тощо.

4. Монетизація та реклама: Забезпечення інструментів для розміщення реклами та монетизації створеного користувачами контенту.

5. Пошук та рекомендації: Функції пошуку користувачів, контенту та автоматичного рекомендування контенту на основі інтересів користувача.

6. Захист даних: Забезпечення конфіденційності та безпеки персональних даних користувачів відповідно до вимог законодавства та норм приватності.

7. Доступність на різних пристроях: Підтримка мобільних версій та додатків для забезпечення доступу до платформи з будь-якого пристрою.

8. Аналітика та статистика: Надання інструментів для вимірювання активності користувачів, аналізу поведінки та ефективності контенту.

9. Інтеграція з іншими сервісами: Можливість інтеграції платформи з іншими онлайн-сервісами, такими як месенджери чи електронна пошта, для зручності користувачів.

10. Спеціалізація та фокус: Можливість фокусуватися на певній тематиці або цільовій аудиторії для створення унікального співтовариства з глибшою взаємодією.

Ці ключові функції та можливості є основою для створення успішної та залученої соціальної мережі, яка може задовольнити потреби та інтереси своїх користувачів.

1.8. Висновоки до розділу.

У цьому розділі описано ключові елементи та властивості, необхідні для соціальної мережевої платформи. Серед них - створення та налаштування користувацьких акаунтів, взаємодія між учасниками, публікація та обмін контентом, системи пошуку та персоналізованих рекомендацій, захист інформації про користувачів, адаптація для мобільних пристроїв, аналіз даних та статистика, а також націлення на конкретні теми або аудиторію.

Ці компоненти критично важливі для ефективності та популярності соціальної мережі, а також для задоволення потреб її користувачів. В залежності

від унікальної концепції та цілей, які стоять перед соціальною мережею, можливе додавання додаткових унікальних функцій, що розширюють стандартний набір можливостей.

Розробка та впровадження цих основоположних елементів є ключовим аспектом створення моєї соціальної мережі. Вдумливе впровадження зазначених властивостей сприятиме створенню зручної для користувачів платформи, яка сприяє активному спілкуванню та обміну інформацією, приваблюватиме нових учасників та сприятиме їх відданості створеній соціальній мережі

РОЗДІЛ 2. ВИБІР ЗАСОБІВ ДЛЯ РЕАЛІЗАЦІЇ ПРОЕКТУ.

2.1 Розробка математичної моделі платформи та алгоритмів, що використовуються для реалізації функцій.

Створення математичної моделі та алгоритмів є ключовим аспектом під час проектування будь-якої програмної системи, оскільки це допомагає формувати адекватну архітектуру та забезпечує коректну функціональність системи.

Математична модель платформи представляє собою формалізований опис системи, заснований на математичних правилах та поняттях. Цей опис охоплює функціональні та структурні аспекти системи, їх взаємодію та поведінку під час різних сценаріїв.

Алгоритм визначається як набір інструкцій, призначених для вирішення певної задачі. Створення алгоритмів полягає у виборі найбільш ефективних методів для виконання задач, поставлених перед системою.

Процес розробки математичної моделі та алгоритмів включає наступні кроки:

- Аналіз вимог до системи: збір інформації про потреби користувачів та умови використання системи.

- Проектування архітектури: визначення структури системи, включаючи її компоненти, зв'язки між ними та інтерфейси.

- Формування математичної моделі: створення детального математичного опису системи, що відображає її структуру, функції та поведінку.

- Розробка алгоритмів: вибір та реалізація оптимальних методів для вирішення завдань системи.

- Тестування та налагодження: перевірка системи на відповідність вимогам та виправлення помилок.

Під час розробки важливо враховувати такі аспекти:

- Відповідність функціональним вимогам: система має виконувати всі задані функції.

- Ефективність: забезпечення швидкої та ефективної роботи системи.

- Надійність: забезпечення стабільності роботи системи та мінімізація помилок.

- Сумісність: забезпечення роботи системи з іншими програмами та обладнанням.

- Простота управління: забезпечення зручності використання системи користувачами.

Після створення математичної моделі та алгоритмів, їх перевіряють та вдосконалюють, щоб гарантувати коректну роботу системи. Під час експлуатації системи можуть виникати нові вимоги, які вимагатимуть модифікації алгоритмів. Таким чином, розробка математичної моделі та алгоритмів є постійним процесом, який потребує систематичного оновлення та оптимізації.

2.2 Архітектурні та технічні рішення, що були обрані для розробки платформи соціальної мережі, ґрунтуються на сучасному наборі технологій та методиках.

Вони включають в себе використання HTML5 та CSS для створення інтерфейсу, JavaScript та Node. js для програмного забезпечення, і також React із React-Redux для створення динамічного відображення.

Головна концепція полягає в розподіленні системи на фронтенд та бекенд з метою спрощення процесу розробки та подальшої підтримки.

Щодо фронтенду, використовуються:

- HTML5 та CSS3 для створення структури та стилізації веб-сторінок.

- JavaScript як основний засіб для втілення інтерактивності та динамічних елементів.

- React для розробки інтерфейсу як набору незалежних компонентів, що забезпечує швидке оновлення та високу продуктивність.

- React-Redux і Redux для управління станом додатку та спрощення роботи з даними в масштабованих проектах.

Бекенд ґрунтується на:

- Node.js як середовище виконання для JavaScript, що дозволяє створювати швидкі та масштабовані мережеві додатки.
- Type Script для додавання статичної типізації до JavaScript, що підвищує надійність коду та спрощує розробку.

Взаємодія між фронтендом та бекендом відбувається через HTTP та Веб-сокет протоколи, де HTTP використовується для стандартних запитів, а Веб-сокет забезпечує двосторонній обмін даними в реальному часі.

У сфері баз даних використовуються як реляційні (наприклад, PostgreSQL, MySQL), так і нереляційні системи для зберігання даних, а також ORM бібліотеки для полегшення роботи з ними.

Для забезпечення безпеки та продуктивності застосовуються методи шифрування, двофакторної автентифікації, захисту від XSS та CSRF атак, а також кешування з використанням Redis або Memcached для підвищення продуктивності.

Інтеграція з зовнішніми сервісами реалізується через API, такі як Google Maps API, Facebook API, Twitter API, для розширення функціоналу та привертання користувачів.

Отже, архітектура платформи соціальної мережі є складною та багатошаровою, вимагаючи інтеграції різноманітних технологій та підходів для створення ефективної, безпечної та масштабованої системи.

Розробка сучасної соціальної мережі потребує використання передових технологій та методів, таких як HTML5, CSS, JavaScript, Node.js, React, Redux та TypeScript для створення користувацького інтерфейсу та серверної логіки. Цей підхід дозволяє об'єднати розробку клієнтської та серверної частин за допомогою єдиного стеку технологій.

Клієнтська частина відповідає за візуальне відображення та інтерактивність, використовуючи React для побудови інтерфейсу. Серверна частина, заснована на Node js, забезпечує обробку запитів, авторизацію користувачів та взаємодію з базою даних.

Для підвищення продуктивності використовується кешування даних для зменшення навантаження на базу даних та CDN для розподілення статичного контенту. Масштабованість системи є ключовою для забезпечення стабільності та продуктивності під час збільшення обсягів використання. Для цього застосовуються горизонтальне та вертикальне масштабування, а також розподілення баз даних та кешувальних систем, таких як Redis або Memcached.

У підсумку, успішне створення соціальної мережі вимагає розуміння та вправного застосування сучасних технологій. Важливо забезпечити технічну ефективність та стабільність системи, а також зрозуміти потреби користувачів та створити зручний та привабливий інтерфейс.

2.2.1 Найпопулярніші мови програмування та їх фреймворки для веб-розробок.

Веб-розробка використовує широкий спектр програмних мов, серед яких деякі виділяються своєю популярністю та універсальністю:

Рис.2.1. Найпопулярніші мови програмування для веб-розробок [5].

Веб-розробка використовує широкий спектр програмних мов, серед яких деякі виділяються своєю популярністю та універсальністю:

- JavaScript займає перше місце завдяки своїй унікальній можливості працювати як на клієнтській, так і на серверній стороні, що робить його незамінним інструментом для створення інтерактивних веб-сайтів.
- Java відома своєю стабільністю та масштабованістю, що робить її ідеальним вибором для розробки складних корпоративних веб-додатків, особливо з використанням Java EE.
- Python здобув популярність завдяки своїй читабельності та ефективності, ставши вибором для багатьох розробників, які цінують швидкість розробки та гнучкість, особливо з фреймворками Django та Flask.
- C# використовується для створення веб-додатків на платформі .NET, забезпечуючи потужні можливості та підтримку від Microsoft, що робить його популярним серед розробників, які працюють з цією платформою.
- TypeScript, як розширення JavaScript, пропонує додаткові переваги статичної типізації, що сприяє розробці більш стабільного та легкого для підтримки коду, особливо у великих проектах, з використанням фреймворків, таких як Angular та React.
- JavaScript і TypeScript часто використовуються разом для розробки фронтенду веб-додатків, надаючи розробникам гнучкість та потужні інструменти для створення сучасних веб-додатків.

Ці мови програмування мають велику підтримку від спільноти розробників та багато інструментів для розробки веб-додатків, що робить їх дуже популярними серед розробників веб-додатків.

Щодо фреймворків, їх є дуже велика кількість та всі вони розрізняються своїм синтаксисом написання коду та функціями. Так як в даній роботі я буду використовувати JavaScript та TypeScript, я наведу самі популярні фреймворки для цих мов програмування.

Основні фреймворки для JavaScript, які полегшують створення швидких, масштабованих та ефективних веб-додатків:

- *React*: Популярний фреймворк для фронтенду на JavaScript, розроблений Facebook. React спрощує створення компонентної структури веб-додатків і дозволяє ефективно керувати DOM, що робить його ідеальним для великих проектів.
- *Angular*: Фреймворк для фронтенду на JavaScript від Google. Angular дозволяє створювати веб-додатки зі складною бізнес-логікою та підтримує статичну типізацію за допомогою TypeScript.
- *Vue.js*: Прогресивний фреймворк для фронтенду на JavaScript, що дозволяє швидко та ефективно розробляти веб-додатки та компоненти. Він має простий синтаксис та легко інтегрується з іншими бібліотеками та фреймворками.
- *Ember.js*: Фреймворк для фронтенду на JavaScript, призначений для розробки складних веб-додатків. Ember.js надає розробникам широкий набір інструментів та функцій для створення масштабованих та легко підтримуваних додатків.
- *Node.js*: Середовище виконання JavaScript на сервері, що дозволяє створювати бекенд-додатки. Node.js дозволяє використовувати одну мову програмування для розробки фронтенду та бекенду, спрощуючи процес розробки та підтримки веб-додатків.

У цьому проекті я збираюся використовувати фреймворки React і Node.js, оскільки вони забезпечують достатній функціонал для створення соціальної мережі.

2.2.2. JavaScript.

JavaScript є однією з найпопулярніших та широко використовуваних мов програмування з великим впливом на сферу веб-розробки. Його веб-орієнтованість і можливість взаємодії з HTML та CSS роблять його ключовим інструментом для створення веб-додатків прямо у браузері.

Вплив JavaScript на розвиток веб-технологій та веб-додатків дуже значний:

- Динамічність веб-сторінок: JavaScript дозволяє створювати динамічні веб-сторінки, що реагують на дії користувача без перезавантаження. Це робить веб-додатки більш інтерактивними та зручними.
- Валідація даних: JavaScript може використовуватися для перевірки даних, введених користувачем, перед їхньою відправкою на сервер, що забезпечує їхню правильність та цілісність.
- Розробка веб-додатків: JavaScript є основою для багатьох сучасних веб-додатків і фреймворків, таких як Angular, React і Vue.js, що дозволяють створювати складні та ефективні додатки.
- Взаємодія з API: JavaScript дозволяє взаємодіяти з різними API, що розширює можливості веб-додатків та інтегрує їх зі сторонніми сервісами.
- Анімація та візуалізація: JavaScript використовується для створення анімацій, переходів та візуалізації даних, що робить веб-сторінки привабливішими та зрозумілішими для користувачів.

JavaScript продовжує активно розвиватися, стаючи необхідним інструментом для багатьох веб-розробників та постійно розширюючи можливості веб-програмування.

2.2.3. React.

React є одним з найпопулярніших фреймворків для створення веб-додатків на JavaScript, розроблений Facebook у 2013 році. Його популярність пояснюється кількома причинами.

По-перше, React відзначається високою продуктивністю та швидкістю роботи веб-додатків. Завдяки спеціальним алгоритмам, React здатний швидко та ефективно оновлювати лише ті частини додатку, які зазнали змін.

По-друге, React має велику спільноту розробників, яка постійно доповнює та вдосконалює цей фреймворк. Це призводить до наявності безлічі сторонніх бібліотек, компонентів та плагінів, що полегшують процес розробки.

По-третє, React надає розробникам можливість створювати компоненти з нуля, що дозволяє створювати унікальні та інноваційні інтерфейси додатків. Крім того, React має різноманітні інструменти для тестування та відлагодження, що спрощує процес розробки.

Узагальнюючи, React є потужним та ефективним фреймворком для створення веб-додатків на JavaScript, який дозволяє розробникам створювати високоякісні та інноваційні продукти. Його можливості забезпечують плавну обробку подій, переходів станів та підготовку даних відображення, відхиляючись від стандартної практики зберігання розмітки та логіки у різних файлах.

[12].

Плюси:

- Економія часу під час перепрофілювання компонентів.
- Бібліотека з відкритим кодом, що включає широкий спектр інструментів
- Одностороннє переміщення даних забезпечує стабільний код.
- Віртуальний DOM покращує як досвід роботи користувача, так і працю розробника.
- Його багаторазові компоненти полегшують розробку та обслуговування додатків.
- Він регулярно оновлює та випускає нові версії фреймворку. Ви вчасно отримаєте виправлення помилок та імпровізації.

Мінуси:

- Крива навчання досить стрімка.
- Складність JSX важко зрозуміти розробникам.
- Через швидкі темпи розвитку бракує документації.

- Ви можете втратити «компоненти потоку та даних» у міру розширення проекту.

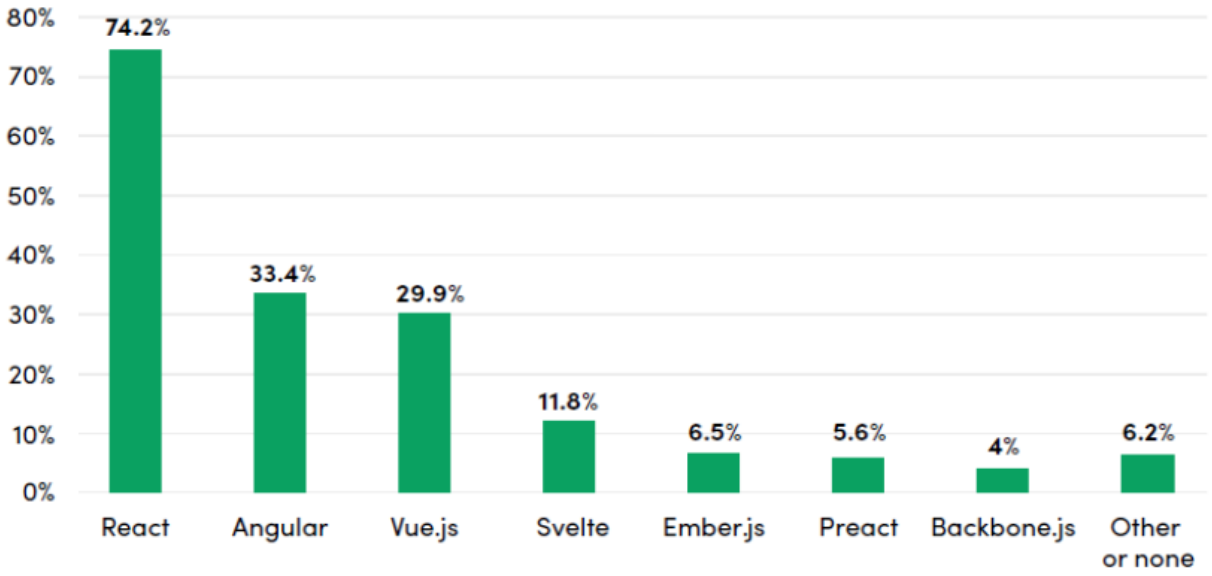

Рис. 2.2. Популярність React в порівнянні з іншими відомими фреймворками JavaScript [6].

2.2.4. Node.js.

Node.js - це середовище виконання JavaScript на сервері, яке дозволяє розробникам використовувати JavaScript для створення бекенду веб-додатків. Він заснований на движку V8, який розробляється Google для виконання JavaScript у браузері Google Chrome. Node.js дозволяє розробникам використовувати одну мову програмування (JavaScript) для розробки як фронтенду, так і бекенду, що полегшує розробку та підтримку веб-додатків [7].

Однією з основних переваг Node.js є те, що він дозволяє створювати швидкі та масштабовані веб-додатки з високою продуктивністю. Node.js дозволяє розробникам створювати бекенд-додатки зі складною бізнес-логікою, що можуть взаємодіяти з базами даних, мережами, іншими веб-серверами та API.

Node.js також має велику екосистему модулів та бібліотек, що дозволяє розробникам швидко та ефективно створювати веб-додатки. Наприклад, Express.js - це один з найбільш популярних фреймворків для Node.js, який дозволяє розробникам створювати API та веб-сервери з використанням Node.js.

2.2.5. TypeScript.

TypeScript - це мова програмування, яка є розширенням JavaScript і надає статичну типізацію. Вона розроблена компанією Microsoft і стала досить популярною серед розробників веб-додатків.

Однією з ключових особливостей TypeScript є підтримка статичного аналізу типів, що дозволяє виявляти помилки та неправильне використання типів під час розробки. Це сприяє підвищенню надійності коду, полегшує рефакторинг та підтримку великих проектів [17].

TypeScript компілюється до звичайного JavaScript, що означає, що ви можете використовувати його в будь-якому середовищі, яке підтримує JavaScript. Це також дозволяє поступово впроваджувати TypeScript у існуючі проекти, починаючи з окремих файлів чи модулів.

Окрім статичної типізації, TypeScript також надає інші корисні можливості, такі як підтримка класів, інтерфейсів, модулів, узагальнених типів і багато іншого. Він також інтегрується з популярними фреймворками та бібліотеками, такими як React, Angular і Node.js, що робить TypeScript ще більш привабливим для розробників.

TypeScript допомагає знизити кількість помилок, полегшує співпрацю між командами розробників і покращує продуктивність. Він використовується для розробки різноманітних веб-додатків, від невеликих сайтів до великих корпоративних систем [17].

2.2.6. Redux.

Redux - це бібліотека для управління станом додатків у JavaScript. Вона призначена для роботи з даними та станом у складних веб-додатках з багатьма компонентами. Redux дозволяє розробникам зберігати та змінювати стан додатку в єдиному місці, у так званому "store" (сховищі). Кожен компонент може отримувати доступ до стану через спеціальні "actions" (дії), що сприяє кращій організації та керуванню даними.

Redux має безліч переваг для розробників. Наприклад, вона полегшує управління станом додатку, сприяє взаємодії між його компонентами та має добре

розроблену систему відлагодження. Крім того, Redux дозволяє розробникам комбінувати її з різними бібліотеками та фреймворками, що підвищує продуктивність та ефективність розробки.

Redux є невід'ємною частиною екосистеми React, одного з найпопулярніших фреймворків для створення веб-додатків на JavaScript. Використовуючи Redux з React, розробники можуть ефективно керувати станом додатку та забезпечити оптимальну роботу з великим обсягом даних та компонентів.[10].

2.3 Огляд процесу розробки та тестування платформи соціальної мережі.

Розробка та тестування платформи соціальної мережі - це складний процес, що вимагає багато часу, ресурсів та зусиль. У цьому процесі можуть брати участь різні команди розробників, які працюють над різними аспектами розробки, такими як фронтенд, бекенд, дизайн та тестування.

Процес розробки може складатися з наступних етапів:

1. Аналіз та збір вимог - це перший етап розробки, під час якого розробники зустрічаються з клієнтом, щоб з'ясувати вимоги до платформи соціальної мережі та збір вимог від користувачів.
2. Проектування та розробка бази даних - це етап, на якому розробники проектують та розробляють базу даних, яка буде використовуватися для зберігання інформації користувачів, дописів, коментарів та іншої інформації, що використовується в платформі.
3. Розробка бекенду - це етап, на якому розробники створюють серверну частину платформи соціальної мережі. Для розробки бекенду можна використовувати Node.js та NoSQL базу даних, такі як MongoDB або CouchDB.
4. Розробка фронтенду - це етап, на якому розробники створюють клієнтську частину платформи соціальної мережі. Для розробки фронтенду можна використовувати HTML5, CSS3, JavaScript, React, React-Redux, Redux, TypeScript, Ant-design.

5. Тестування та налагодження - це етап, на якому розробники тестують платформу соціальної мережі, щоб переконатися, що вона працює належним чином та відповідає вимогам клієнта та користувачів
6. Розгортання та налаштування - це етап, на якому розробники розгортають платформу соціальної мережі на серверах та налаштовують її для роботи в режимі виробництва.
7. Підтримка та розвиток - це етап, на якому розробники забезпечують підтримку та розвиток платформи соціальної мережі після її запуску. Це може включати в себе розробку нових функцій, виправлення помилок та забезпечення безпеки.

Після завершення розробки платформа соціальної мережі проходить процес тестування, який допомагає переконатися, що вона відповідає вимогам та працює належним чином. Тестування може бути автоматичним та ручним. Автоматичне тестування виконується за допомогою спеціальних програмних засобів, які перевіряють функціональність платформи та виявляють помилки. Ручне тестування проводиться тестувальниками, які тестують різні аспекти платформи та виявляють помилки, які можуть бути пропущені автоматичним тестуванням.

У процесі тестування можуть бути використані різні методи тестування, такі як модульне тестування, інтеграційне тестування, системне тестування та інші. Модульне тестування дозволяє перевірити окремі модулі платформи, інтеграційне тестування - взаємодію між різними модулями, системне тестування - роботу всієї платформи в цілому.

Після завершення тестування та налаштування платформа готова до запуску та використання користувачами. Проте, після запуску необхідно продовжувати її розвиток та підтримку.

Це може включати в себе наступні дії:

1. Розвиток нових функцій та покращення існуючих - з часом можуть з'являтися нові потреби та вимоги користувачів, тому розробники повинні продовжувати розвивати платформу, додавати нові функції та покращувати існуючі.

2. Виправлення помилок - під час використання платформи користувачі можуть виявляти помилки та недоліки, тому розробники повинні оперативно їх виправляти.
3. Забезпечення безпеки - платформа соціальної мережі повинна бути захищена від зловмисників та забезпечувати приватність користувачів, тому розробники повинні вдосконалювати систему безпеки та вчасно виявляти можливі загрози.
4. Моніторинг та аналітика - розробники повинні відстежувати роботу платформи, збирати дані та аналізувати їх, щоб зрозуміти, як користувачі взаємодіють з платформою та як можна покращити її функціональність.

Усі ці дії вимагають від розробників знань та досвіду в галузі розробки програмного забезпечення та технологій, які використовуються для розробки платформи соціальної мережі, таких як HTML5, CSS3, JavaScript, Node.js, React, React-Redux, Redux, TypeScript, axios, Ant-design. Крім того, необхідно мати розуміння бізнес-потреб та вимог користувачів, щоб створювати функціональності, які будуть корисні та зручні для користувачів.

2.4. Висновки до розділу.

У цьому розділі було розглянуто React як бібліотеку для керування станом додатку в JavaScript. Вона дозволяє зберігати та змінювати стан додатку в одному місці, що сприяє кращій організації та керуванню даними в складних веб-додатках. Redux також має безліч переваг для розробників, забезпечуючи легке керування станом додатку, взаємодію між компонентами та систему відлагодження.

Також було розглянуто загальний процес розробки та тестування платформи соціальної мережі. Цей процес включає аналіз вимог, проектування та розробку бази даних, розробку бекенду та фронтенду, тестування та налагодження, розгортання та налаштування, а також підтримку та розвиток після запуску. Під час тестування можуть використовуватися різні методи, такі як модульне тестування, інтеграційне тестування та системне тестування.

Цей процес вимагає від розробників знань та досвіду в галузі розробки програмного забезпечення та використовуваних технологій, а також розуміння бізнес-потреб та вимог користувачів.

Усе це разом допомагає створити ефективну та функціональну платформу соціальної мережі, яка задовольняє потреби користувачів та пропонує їм зручний інтерфейс для взаємодії та обміну інформацією

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ПРОЕКТУ.

3.1 Архітектура веб-платформи для соціальної мережі

Архітектура веб-платформи для соціальної мережі організована за допомогою клієнт-серверної моделі, де клієнти взаємодіють з сервером для отримання інформації та виконання дій. Основні компоненти архітектури включають:

1. Клієнтська частина:
    - Користувацький інтерфейс: забезпечує взаємодію користувача з платформою, включаючи реєстрацію, авторизацію, перегляд новин, діалоги, профілі користувачів тощо.
    - Функціональні модулі: включають реалізацію функцій соціальної мережі, таких як створення та редагування профілю, додавання друзів, публікація та коментування записів, пошук користувачів тощо.
    - Клієнтська сторона взаємодії з сервером: відповідає за передачу запитів серверу та обробку отриманих відповідей.
2. Серверна частина:
    - Серверний додаток: приймає та обробляє запити від клієнтів, виконує логіку бізнес-процесів, взаємодіє з базою даних та іншими зовнішніми сервісами.
    - База даних: зберігає дані користувачів, їх профілі, дружби, повідомлення, записи тощо. Може використовувати реляційну базу даних або NoSQL рішення, залежно від потреб проекту.
    - Сервіси: можуть включати сервіси для аутентифікації та авторизації користувачів, обробки медіафайлів (фотографій, відео), рекомендаційних систем тощо.
3. API (Application Programming Interface):
    - Визначає набір доступних запитів та формат обміну даними між клієнтською та серверною частинами.
    - Може бути реалізований, наприклад, за допомогою REST або GraphQL протоколів.

4. Інфраструктура:
    - Хостинг: забезпечує розміщення серверної частини платформи в Інтернеті.
    - Масштабованість: розробка архітектури, яка дозволяє масштабувати платформу, наприклад, за допомогою горизонтального масштабування (додавання додаткових серверів) або використання хмарних сервісів.

Це загальна структура архітектури веб-платформи для соціальної мережі, і конкретні деталі можуть змінюватися залежно від вимог проекту, обраної технології та доступних ресурсів.

Структурна схема архітектури проекту наведена у (додатку А).

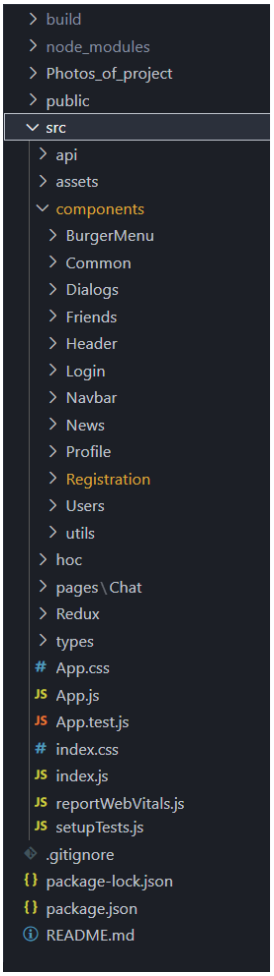

Рис. 3.1. Загальна структура папок проекту.

3.1.1 Клієнтська частина.

Index.js файл є точкою входу для React-додатка. Він імпортує необхідні залежності, налаштовує віртуальний DOM та рендерить компоненту **SocNetworkApp** в елементі з ідентифікатором 'root'.

Опис основних функцій та їх призначення:

1. **ReactDOM.createRoot(document.getElementById('root'))**: Ця функція створює віртуальний DOM кореневий вузол, який буде використовуватися для рендерингу React-компонент. Вона отримує елемент DOM з ідентифікатором 'root' за допомогою **document.getElementById('root')** та створює віртуальний корінь за допомогою **ReactDOM.createRoot()**.
2. **root.render(<React.StrictMode><SocNetworkApp /></React.StrictMode>)**: Ця функція виконує рендеринг React-компоненти **<SocNetworkApp />** віртуальним коренем. Компонента оточена **<React.StrictMode>**, що дозволяє виявляти потенційні проблеми в додатку та накладати додаткові обмеження на деякі функції.
3. **reportWebVitals()**: Ця функція відправляє результати вимірювання продуктивності додатка на аналітичний сервер або виводить їх у консоль. Вона викликається для початку вимірювання продуктивності у додатку.

Цей файл є основним файлом додатка, який ініціалізує його і встановлює точку входу для віртуального DOM. Рендеринг компоненти **SocNetworkApp** виконується у віртуальному корені з ідентифікатором 'root'.

App.js файл є головним компонентом додатка і містить основну логіку маршрутизації та рендерингу компонент.

Приклад коду наведений у (додатку Г).

Опис даного файлу та вкладених функцій:

1. **App**: Класовий компонент, який відповідає за загальну структуру додатка. Він містить методи життєвого циклу та рендерить інші компоненти залежно від поточного шляху URL.

- **componentDidMount()**: Метод життєвого циклу, який викликається після монтажу компонента в дерево DOM. Використовується для ініціалізації додатка шляхом виклику **this.props.initializeApp()**.
- **render()**: Метод, який відповідає за рендеринг компоненти. Якщо **initialized** (значення, що показує, чи завершена ініціалізація додатка) є **false**, то відображається компонента **Preloader** (попередній завантажувач). У протилежному випадку рендеряться інші компоненти, які залежать від поточного шляху URL.

2. **mapStateToProps**: Функція, яка використовується для вибору необхідних даних зі стану Redux і передачі їх як властивостей у компоненту **App**. В даному випадку, властивість **initialized** має значення **state.app.initialized** [8].
3. **AppContainer**: Контейнерний компонент, який з'єднує компонент **App** з Redux Store та використовує функцію **withRouter** для доступу до об'єкта **history**. Використовується функція **compose** для композиції декількох функцій підключення [15].
4. **SocNetworkApp**: Функціональний компонент, який обгортає **AppContainer** у **HashRouter** (компонент маршрутизації з хешем) та **Provider** (компонент Redux), щоб забезпечити правильну роботу маршрутизації та доступ до Redux Store [9].

В цьому файлі також є імпорти компонентів, які використовуються в **App**:

- **Preloader**: Компонента попереднього завантажувача, яка відображається під час завантаження даних або ресурсів.
- **withRouter**: HOC (High Order Component) для надання компонентам доступу до об'єкта **history** з React Router.
- **HeaderContainer**: Контейнерний компонент для заголовка додатка.
- **Navbar**: Компонента бічного меню навігації.
- **UsersContainer**: Контейнерний компонент для списку користувачів.

- **DialogsContainer**: Контейнерний компонент для списку діалогів.
- **ProfileContainer**: Контейнерний компонент для профілю користувача.
- **Login**: Компонента сторінки входу.
- **NewsList**: Компонента списку новин.
- **Registration**: Компонента сторінки реєстрації.
- **Friends**: Компонента сторінки друзів.
- **ChatPage**: Компонента сторінки чату.
- **SportList**: Компонента списку спортивних новин.
- **UkraineList**: Компонента списку новин про Україну.
- **USANewsList**: Компонента списку новин про США.
- **AnimalsNewsList**: Компонента списку новин про тварин.
- **ScienceNewsList**: Компонента списку наукових новин.

Крім того, у цьому файлі є динамічний імпорт компонент **DialogsContainer** та **ProfileContainer** за допомогою **React.lazy()**. Це дозволяє затримувати завантаження компонентів до того моменту, коли вони дійсно потрібні.
UML-діаграма клієнтської частини веб-платформи соціальної мережі наведена у (додатку Б).

Компонента **Navbar** відповідає за відображення бічного меню навігації в додатку.

Приклад коду наведений у (додатку Г).

Опис основних фрагментів коду:

1. Функція **setActive**:

```
const setActive = ({ isActive }) => isActive ? `${n.activeclass}` : '';
```

Ця функція приймає об'єкт з властивістю **isActive** і повертає клас **activeclass**, якщо властивість **isActive** істинна, або порожній рядок в протилежному випадку. Наприклад, якщо передати **{ isActive: true }**, функція поверне **"activeclass"**.

2. Компонента **Navbar**:

Основні характеристики цієї компоненти:

- Використовується елемент **<nav>** для огортання навігаційних елементів.

- Кожен пункт меню представлений елементом ** ** з класом **item**, який містить посилання **<NavLink>**.
- Атрибут **to** у **<NavLink>** вказує на шлях URL, до якого посилання повинно вести.
- Для визначення активного стану посилання використовується функція **setActive**.
- Функція **setActive** перевіряє властивість **isActive** і повертає клас **activeclass**, якщо **isActive** істинна, або порожній рядок в протилежному випадку.

Ця компонента є частиною бічного меню навігації в додатку і дозволяє користувачам перехідати до різних сторінок, таких як "Profile" (Профіль), "Messages" (Повідомлення), "Chat" (Чат), "My friends" (Мої друзі), "Users" (Користувачі), "News" (Новини) та інші.

Наприклад, пункт меню "My friends" виглядає так:

```
const Navbar = () => {
  return (
    <nav className={n.nav}>
{/* Фрагмент коду для пункту My friends */}
      <div className={n.item}>
        <NavLink to='/friends' className={setActive}>
          My friends
        </NavLink>
      </div>
</nav>
  );
};
export default Navbar;
```

Цей код створює посилання, яке веде до сторінки "/myfriends" і має клас **item**. Клас **setActive** використовується для визначення активного стану посилання. Аналогічним чином створюються інші пункти меню, які дозволяють користувачам навігувати по різних сторінках додатку.

Ця компонента також може містити додаткові елементи, наприклад, компоненту меню-бургера для мобільної версії додатку.

Для реалізації компоненти **Profile** яка відповідає за взаємодію з сторінкою користувача було розроблено та використано такі основні файли:

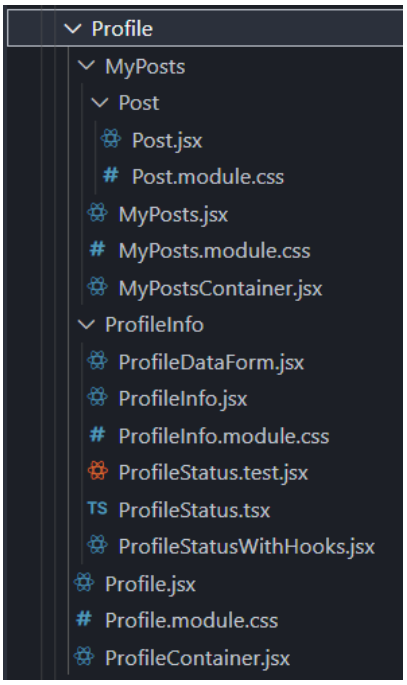

Рис. 3.2. Структура файлів компоненти Profile.

Компонента **Profile** містить у собі основні папки **MyPosts** та **ProfileInfo**, які відповідають за додавання та відображення постів користувача, а також за додавання та редагування інформації про користувача, його фото та статус, відповідно.

Компонента **MyPost** містить в собі компоненту Post яка в свою чергу відповідає за логіку, відображення та стилізацію тільки одного поста.

Нижче я детальне опишу та поясню логіку основних функцій які використовував у компоненті **Profile**.

Компонента **Post** є простим компонентом, який відображає пост з інформацією про користувача, повідомленням і кнопками для вподобання, несподобання та видалення поста. Для здійснення HTTP-запитів для отримання інформації про користувача та взаємодії з сервером використовується бібліотека Axios.

Приклад коду наведений у (додатку Г).

Основні функції, які використовуються в даній компоненті:

1. **useState**: Хук **useState** використовується для оголошення стану компонента. У даному випадку, використовуються стани **userPhoto** (фото користувача) та **userId** (ідентифікатор користувача).
2. **useEffect**: Хук **useEffect** використовується для виконання побічного ефекту, такого як отримання даних з сервера. У компоненті **Post** використовуються два ефекти **fetchUserPhoto** та **fetchUserId**.
3. **handleDelete**: Ця функція викликається при натисканні кнопки "Видалити" і викликає функцію **deletePost** з передачею ідентифікатора поста **props.id**.
4. **handleLike** та **handleDislike**: Ці функції викликаються при натисканні відповідних кнопок "Вподобати" та "Несподобати" і викликають відповідні функції **likePost** та **dislikePost** з передачею ідентифікатора поста **props.id**.

Компонента **MyPosts** є React-компонентою, яка відповідає за відображення постів на сторінці.

Приклад коду наведений у (додатку Г).

Основний код компоненти включає наступні елементи:

1. Оголошення та ініціалізація стану **searchQuery** за допомогою хука **useState**.
2. Обробник події **handleSearch** для пошуку постів.
3. Фільтрація постів згідно з пошуковим запитом.
4. Створення компонентів постів на основі відфільтрованих даних за допомогою метода map().
4. Обробник події **onAddPost** для додавання нового поста.
5. Валідатор **maxLength10** для обмеження довжини тексту поста, максимальна кількість символів у пості 50.
6. Форма додавання нового поста, створена за допомогою **reduxForm**.

Ця компонента має властивості **postsData**, **addPost**, **deletePost**, **likePost** та **dislikePost**, які передаються в якості пропсів для відображення та взаємодії з постами на сторінці.

Компонента **ProfileDataForm** використовується для відображення форми редагування профілю користувача.

Приклад коду наведений у (додатку Г).

Вона має наступну структуру:

1. У формі є кнопка "Save" для збереження внесених змін.
2. Якщо виникає помилка при валідації форми, вона відображається у блоку props.error.
3. У формі відображається ідентифікатор користувача з поля props.profile.userId.
4. Компонента **createField** використовується для створення полів вводу або чекбоксів для редагування різних полів профілю. Вона отримує параметри, такі як назва поля, назва поля у формі, валідатори, тип поля та інші налаштування.
5. Поля для редагування повного імені, чи користувач шукає роботу, професійні навички, інформацію про користувача та контакти відображаються за допомогою компоненти **createField**.
6. Компонента **ProfileDataForm** обгортається у HOC reduxForm, де форма отримує назву 'edit-profile'.

Компонента **ProfileDataForm** дозволяє редагувати і зберігати дані профілю користувача, використовуючи поля вводу та відображаючи їх у форматі, специфічному для профілю користувача.

Приклад коду наведений у (додатку Г).

Основні елементи клієнтської частини компоненти **ProfileInfo**:

1. Компонента **ProfileInfo** використовується для відображення інформації про профіль користувача.
2. Компонента **ProfileData** використовується для відображення даних профілю користувача.

```
const ProfileData = (props) => {
  return (
    <>
```

```
        {/* Відображення ID користувача */}
        {/* Відображення повного імені */}
        {/* Відображення, чи користувач шукає роботу */}
        {/* Відображення професійних навичок, якщо користувач шукає
      роботу */}
        {/* Відображення інформації про користувача */}
        {/* Відображення контактів */}
        {/* Кнопка редагування профілю, якщо користувач є власником
      профілю */}
        </>
      );
    };
```

3. Компонента **Contact** використовується для відображення окремого контакту профілю користувача.
4.

Компонента **ProfileInfo** включає в себе компоненту **ProfileData** для відображення даних профілю, а також використовує стан **editMode** для визначення режиму редагування профілю. Компонента також містить функції **onMainPhotoSelected** для обробки вибору фото та **onSubmit** для збереження відредагованих даних профілю.

Компонента **Profile** використовується для відображення профілю користувача цілком, де зібрані всі компоненти які були задіяні вище.

Компонента **Profile** включає в себе компоненту **ProfileInfo**, яка відображає інформацію про профіль користувача, та компоненту **MyPostsContainer**, яка відображає пости користувача.

Приклад коду наведений у (додатку Г).

Компонента **Profile** приймає наступні пропси:

- **savePhoto**: функція для збереження фото профілю;
- **isOwner**: логічне значення, чи є користувач власником профілю;
- **profile**: дані про профіль користувача;

- **status**: статус користувача;
- **saveProfile**: функція для збереження відредагованих даних профілю;
- **updateStatus**: функція для оновлення статусу користувача.

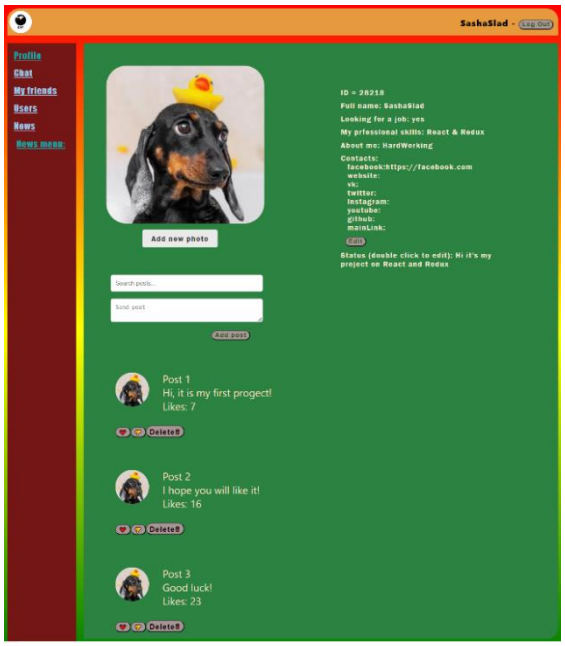

Рис. 3.3. Приклад реалізованої компоненти Profile.

Для реалізації компоненти **Chat** яка відповідає за реалізацію отримання, написання та відправлення повідомлень було розроблено та використано наступні функції та компоненти :

Приклад коду наведений у (додатку Г).

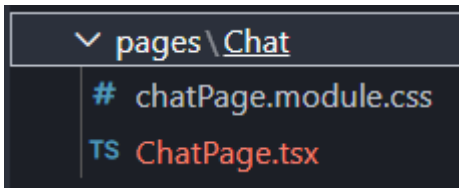

Рис. 3.4. Структура файлів компоненти Chat.

Клієнтська частина складається з компонент ChatPage, Chat, Messages, Message і AddMessageForm.

Приклад коду наведений у (додатку Г).

1. Компонента ChatPage:
    - Відображає основний контейнер для чату.
    - Включає компоненту Chat.

2. Компонента Chat:
    - Отримує доступ до даних з Redux store за допомогою useSelector і useDispatch.
    - Викликає actions для початку та зупинки прослуховування повідомлень.
    - Включає компоненти Messages і AddMessageForm для відображення повідомлень і форми додавання нового повідомлення.
3. Компонента Messages:
    - Отримує повідомлення з Redux store за допомогою useSelector.
    - Використовує useRef для отримання посилання на останнє повідомлення.
    - Встановлює стан isAutoIsScroll для визначення, чи необхідно прокручувати список повідомлень автоматично.
    - Додає обробник прокручування для визначення, чи має бути включений режим автоматичної прокрутки.
    - Відображає повідомлення та мітку, яка служить як якорна точка для автоматичної прокрутки.
4. Компонента Message:
    - Отримує окреме повідомлення як пропс.
    - Відображає інформацію про користувача та текст повідомлення.
5. Компонента AddMessageForm:
    - Використовує useState для збереження значення тексту повідомлення.
    - Отримує доступ до статусу з Redux store за допомогою useSelector.
    - Викликає action для відправлення повідомлення при натисканні на кнопку.
    - Відображає форму для введення тексту повідомлення та кнопку для надсилання.

Це описує основний функціонал та приклади коду для кожної з компонент в ChatPage.

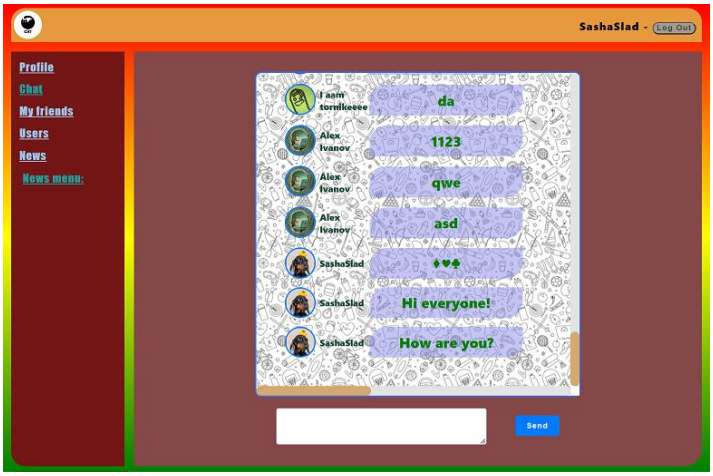

Рис. 3.5. Приклад реалізованої компоненти Chat.

Компонента **Friends**, яка відображає список друзів. При завантаженні компоненти відбувається запит до API для отримання списку користувачів, які є друзями. Компонента також містить можливість пошуку друзів за їх іменем.

Приклад коду наведений у (додатку Г).

1. Використання **useState** для зберігання стану списку користувачів та пошукового запиту:
2. Використання **useEffect** для виконання запиту до API при завантаженні компоненти:
3. Виклик функції **handleFollow** при натисканні кнопки "Follow":
4. Виклик функції **handleUnfollow** при натисканні кнопки "Unfollow":
5. Оновлення значення пошукового запиту у функції **handleSearch**:
6. Фільтрація списку користувачів залежно від пошукового запиту:

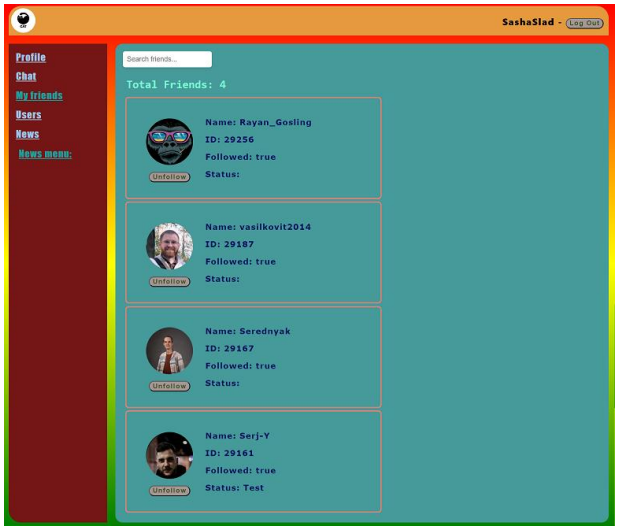

Рис. 3.6. Приклад реалізованої компоненти Friends.

Клієнтська частина компоненти **Users** відповідає за відображення списку користувачів та пагінацію.

Приклад коду наведений у (додатку Г).

Основні елементи коду включають наступне:

1. Компонента пагінації:

У цій частині компонента **Paginator** відображається для забезпечення пагінації списку користувачів. Вона приймає пропси, такі як **currentPage** (поточна сторінка), **onPageChanged** (функція для зміни сторінки), **totalItemsCount** (загальна кількість елементів) та **pageSize** (розмір сторінки).

2. Список користувачів:

У цій частині коду за допомогою **map()** мапується масив **props.users**, щоб створити компоненту **User** для кожного користувача. Кожна компонента **User** отримує властивості, такі як **key** (уникальний ідентифікатор користувача), **user** (дані про користувача), **followingInProgress** (масив ідентифікаторів користувачів, за якими ведеться стеження), **follow** (функція для початку стеження) та **unfollow** (функція для припинення стеження).

Ця клієнтська частина компоненти відповідає за відображення списку користувачів та пагінацію, а також передає необхідні пропси і функції до вкладених компонент.

Компонента **User**, яка відповідає за відображення окремого користувача у списку користувачів.

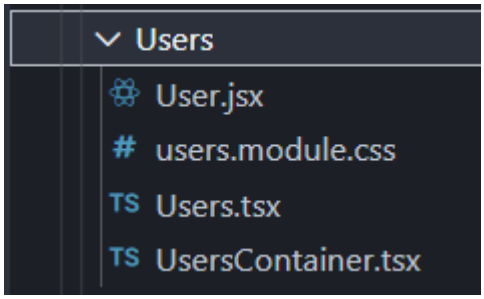

Рис. 3.7. Структура файлів компоненти Users.

Приклад коду наведений у (додатку Г).

Основні елементи коду включають наступне:

1. Відображення фотографії користувача та кнопки "Follow"/"Unfollow":

У цій частині коду відображається фотографія користувача та кнопка "Follow" або "Unfollow", залежно від значення властивості **u.followed**. Кнопка має обробники подій **onClick**, які викликають функції **props.follow(u.id)** або **props.unfollow(u.id)**, передаючи ідентифікатор користувача.

2. Відображення інформації про користувача:

У цій частині коду відображається ім'я користувача, його статус, а також країна та місто за місцезнаходженням.

Ця компонента використовується в компоненті **Users** для відображення кожного окремого користувача у списку.

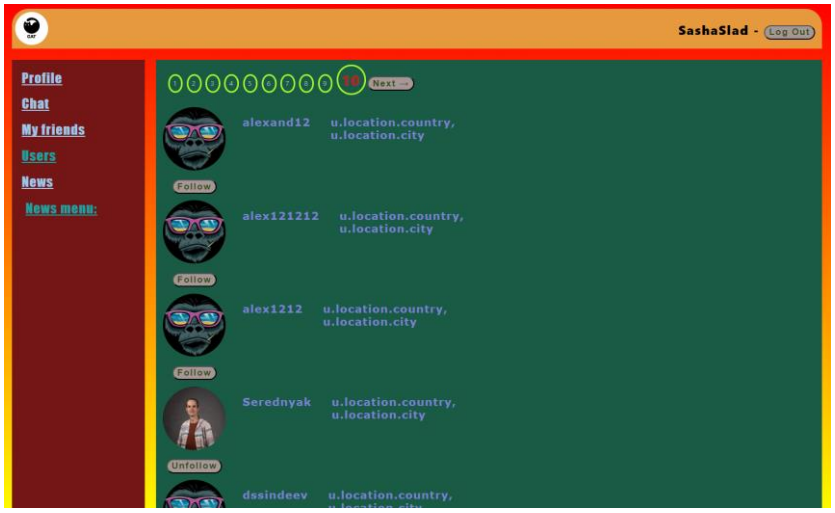

Рис. 3.8. Приклад реалізованої компоненти Users.

Компонента **News** містить в собі файли NewsItems nf NewsList, які відповідають за зв'язок з сервером, обробки та відображення масиву новин, нижче наведений приклад файлової структури та елементів які туди входять:

Компонента **News** відповідає за відображення загальних новин, **UkraineNews**, **USANews**, **ScienceNews**, **SportNews**, **AnimalsNews**.

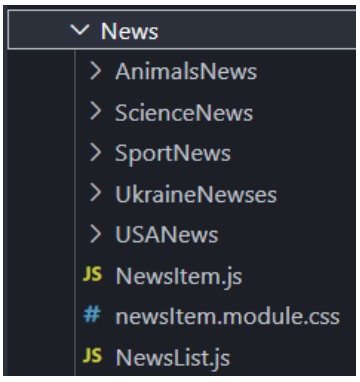

Рис. 3.9. Структура файлів компоненти News.

Компонента **NewsList**, яка відповідає за відображення списку новин.

Приклад коду наведений у (додатку Г).

Давайте розглянемо основні елементи коду:

1. Використання **useState** для збереження стану новин:

У цій частині коду створюється змінна **articles** за допомогою **useState**, яка ініціалізується пустим масивом. Змінна **setArticles** використовуватиметься для оновлення значення **articles**.

2. Використання **useEffect** для отримання новин при завантаженні компоненти:

У цій частині коду **useEffect** використовується для виклику асинхронної функції **getArticles** при завантаженні компоненти. Функція **getArticles** відправляє запит до API за допомогою бібліотеки Axios і отримує список новин. Отримані новини зберігаються в стані за допомогою **setArticles**.

3. Відображення кожної новини у компоненті **NewsItem**:

У цій частині коду використовується метод **map** для перебору кожної новини в масиві **articles**. Для кожної новини створюється компонента **NewsItem**, якій передаються властивості **title**, **description**, **url** та **urlToImage** для відображення відповідної інформації.

Отже, компонента **NewsList** використовується для отримання новин з API і відображення списку новин за допомогою компоненти **NewsItem**.

Компонента **NewsItem**, яка відповідає за відображення окремої новини.

Приклад коду наведений у (додатку Г).

Розглянемо основні елементи коду:

1. Використання пропсів для передачі даних новини:

У цій частині коду визначаються пропси **title**, **description**, **url** та **urlToImage**, які будуть передані з компоненти **NewsList**.

2. Відображення даних новини:

У цій частині коду відбувається відображення даних новини. Компонента **NewsItem** містить блок з класом **news_item**, в якому розміщені зображення, заголовок, опис та посилання на повну новину. Зображення відображається за

допомогою тегу **<img>** з використанням URL-адреси **urlToImage**. Заголовок (**title**) та опис (**description**) відображаються відповідно у тегах **<h3>** та **<p>**. Посилання на повну новину представлене тегом **<a>** з атрибутом **href**, що містить значення **url**.

Таким чином, компонента **NewsItem** відповідає за відображення окремої новини з використанням переданих пропсів, таких як заголовок, опис, URL зображення та URL повної новини.

Компоненти **UkraineNews**, **USANews**, **ScienceNews**, **SportNews**, **AnimalsNews** працють за відповідним принципом який був наведений вище.

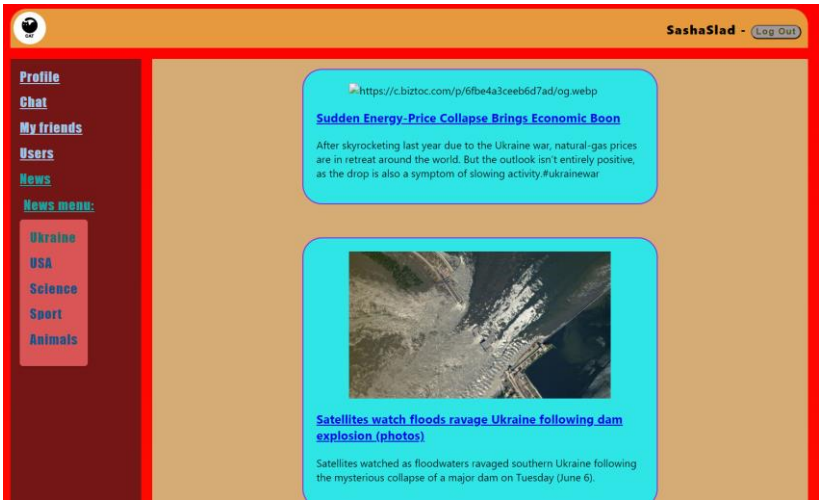

Рис. 3.10. Приклад реалізованої компоненти News.

Компонента **Login** відповідає за форму логінізації користувача. Нижче наведений приклад файлової структури та елементів які туди входять:

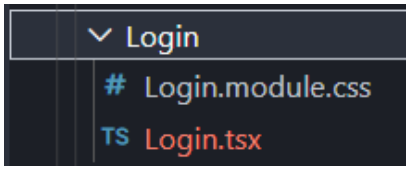

Рис. 3.11. Структура файлів компоненти Login.

Приклад коду наведений у (додатку Г).

Розглянемо основні елементи коду:

У цій частині коду імпортуються необхідні залежності та стилі для компоненти.

1. Оголошення типів та пропсів компоненти:

Тут визначаються типи пропсів для компоненти **LoginForm**, включаючи **LoginFormOwnProps** та **LoginFormValuesType**.

2. Форма для входу:

Ця частина коду відповідає за відображення форми входу користувача. Знаходиться всередині контейнера з класом **log_container**. Форма має обробник подання **onSubmit={props.handleSubmit}**.

3. Відображення полів форми:

У цій частині коду використовується функція **createField**, яка генерує поля форми на основі переданих параметрів. Поля форми включають елементи вводу **Input** для електронної пошти та пароля. Також є поле для позначення "Запам'ятати мене" за допомогою чекбоксу.

4. Обробка помилок та відображення капчі:

Цей код відповідає за відображення капчі (якщо вона є) та повідомлення про помилки під час входу.

5. Кнопки форми:

У цій частині коду відображаються кнопки форми входу та посилання на сторінку реєстрації.

6. Обробка подання форми та перенаправлення:

У цій частині коду відбувається обробка подання форми **onSubmit**. Якщо користувач аутентифікований (**props.isAuth** дорівнює **true**), відбувається перенаправлення на сторінку профілю за допомогою функції **navigate("/profile")**.

7. Підключення Redux та експорт компоненти:

У цій частині коду використовується функція **connect** з бібліотеки Redux для підключення компоненти до Redux store та передачі необхідних даних та дій (**mapStateToProps**, **login**). Компонента експортується за допомогою **export default**.

Компонента **Login** відображає поля для введення електронної пошти та пароля, кнопки входу та реєстрації, обробку помилок та відображення капчі.

Приклад коду наведений у (додатку Г).

Рис. 3.12. Приклад реалізованої компоненти Login.

Компонента **Registration** відповідає за сторінку реєстрації. Код містить станові змінні за допомогою хука **useState**, які зберігають значення полів форми. При зміні полів, змінні оновлюються за допомогою відповідних обробників подій.

Приклад коду наведений у (додатку Г).

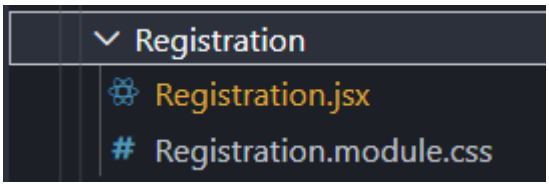

Рис. 3.13. Структура файлів компоненти Registration.

Основні елементи коду:

1. Визначення станових змінних.
2. Обробники подій для зміни значень полів.
3. Обробник події подачі форми.
4. Рендеринг компоненти.

Цей код відповідає за створення форми реєстрації з наступними полями:

- Ім'я (input з типом "text")
- Електронна пошта (input з типом "text")
- Пароль (input з типом "password")
- Підтвердження паролю (input з типом "password")
- Погодження з умовами використання (checkbox)

Також включено кнопку для відправки форми та посилання для входу вже зареєстрованих користувачів.

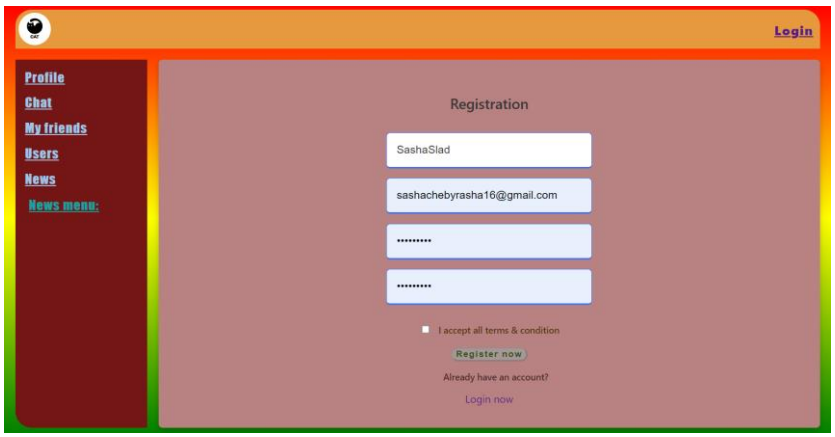

Рис. 3.14. Приклад реалізованої компоненти Registration.

3.1.2 Серверна частина та бізнес-логіка компонент.

В React та Redux, reducer - це чиста функція, яка використовується для зміни стану додатку. Reducer приймає поточний стан і дію, і повертає новий стан. В Redux, стани додатку зберігаються в одному об'єкті, відомому як "store". Reducer відповідає за зміну цього стану відповідно до вхідної дії.

Основні принципи reducer:

1. Чистота: Reducer повинен бути чистою функцією, що означає, що вона повинна завжди повертати однаковий результат для тих самих вхідних даних, без побічних ефектів. Вона не повинна змінювати вхідний стан або викликати зовнішніх сервісів.
2. Незалежність від порядку: Порядок застосування дій до reducer не має значення. Редюсери можуть бути виконані в будь-якому порядку, і результат буде однаковим.
3. Розбиття на декілька редюсерів: Складні додатки можуть мати кілька редюсерів, які відповідають за різні частини стану. Кожен редюсер працює зі своєю власною частиною стану і обробляє відповідну дію.
4. Використання неімутабельності: Редюсери повинні створювати нові об'єкти стану або їхні копії замість зміни поточного стану. Це допомагає забезпечити незмінність та можливість відслідковувати зміни.

Нижче наведена структура папки Redux:

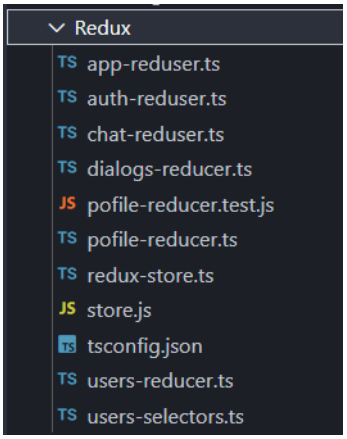

Рис. 3.15. Структура файлів папки Redux.

Нижче наведена структура кодного з файлів, опис основних функцій та реалізацій логіки.

Редюсер **appReduser** відповідає за ініціалізацію додатку та зміну стану **initialized**. Редюсер обробляє дії, які мають тип **INITIALIZED_SUCCESS** і змінює стан, встановлюючи **initialized** на **true**.

Приклад коду наведений у (додатку Г).

Основні елементи коду:

1. Ініціалізація початкового стану.
2. Визначення редюсера.

Коли дія має тип **INITIALIZED_SUCCESS**, редюсер повертає новий стан, в якому поле **initialized** встановлено на **true**. У всіх інших випадках, редюсер повертає поточний стан без змін.

3. Визначення типу дії **INITIALIZED_SUCCESS**.
4. Створення дії **initializedSuccess**, яка повертає об'єкт з типом дії **INITIALIZED_SUCCESS**.
5. Визначення функції **initializeApp**, яка є асинхронною і викликається для ініціалізації додатку.

Ця функція відправляє дію **getAuthUserData()**, яка, ймовірно, запитує дані аутентифікації користувача, а потім за допомогою **Promise.all()** чекає, коли всі асинхронні дії завершаться. Після цього викликається дія **initializedSuccess()**, що спричинить зміну стану на **initialized: true**.

Редюсер **authReduser** відповідає за автентифікацію користувача і зміну стану пов'язаного з автентифікацією.

Приклад коду наведений у (додатку Г).

Основні елементи коду:

1. Ініціалізація початкового стану.
2. Визначення редюсера.

У цьому прикладі, коли дії мають тип **SET_USER_DATA** або **GET_CAPTCHA_URL_SUCCESS**, редюсер повертає новий стан, розпаковуючи поле **payload** дії і змінюючи відповідні поля стану.

3. Визначення типу дії **GET_CAPTCHA_URL_SUCCESS**.
4. Створення дії **getCaptchaUrlSuccess**, яка повертає об'єкт з типом дії **GET_CAPTCHA_URL_SUCCESS**.
5. Визначення типу дії **SET_USER_DATA**.
6. Створення дії **setAuthUserData**, яка повертає об'єкт з типом дії **SET_USER_DATA**.
7. Визначення функції **getAuthUserData**, яка асинхронно виконує запит для отримання даних аутентифікації користувача.

У цьому прикладі, якщо запит на отримання даних **me()** успішний (з кодом **resultCode === ResultCodesEnum.Success**), дія **setAuthUserData** викликається з отриманими даними користувача для зміни стану.

8. Визначення функції **login**, яка асинхронно виконує запит для автентифікації користувача.

У цьому прикладі, якщо запит на автентифікацію login() успішний (з кодом resultCode === ResultCodesEnum.Success), викликається дія getAuthUserData для отримання даних автентифікованого користувача. В іншому випадку, якщо потрібна капча для автентифікації, викликається дія getCaptchaUrl для отримання URL капчі. Також, у випадку невдачної автентифікації, повідомлення про помилку передається у дію stopSubmit для відображення в формі.

9. Визначення функції **getCaptchaUrl**, яка асинхронно отримує URL капчі.

У цьому прикладі, викликається API запит **getCaptchaUrl()** для отримання URL капчі. Отриманий URL передається у дію **getCaptchaUrlSuccess** для зміни стану.

10. Визначення функції **logout**, яка асинхронно виконує вихід з автентифікації.

У цьому прикладі, викликається API запит **logOut()** для виходу з автентифікації. Якщо вихід успішний (з кодом **resultCode === 0**), викликається дія **setAuthUserData** для зміни стану.

Цей редюсер відповідає за зміну стану, пов'язаного з автентифікацією користувача. Він обробляє різні дії, такі як встановлення даних користувача, отримання URL капчі, автентифікація користувача та вихід з автентифікації.

Редюсер **ChatReducer** є частиною системи керування чатом і відповідає за обробку дій, пов'язаних з чатом.

Приклад коду наведений у (додатку Г).

Давайте розглянемо кожну частину коду окремо:

1. Визначення початкового стану редюсера.

- Початковий стан містить поле **messages**, яке є масивом повідомлень типу **ChatMessage[]**. Поки що цей масив є порожнім (**[]**).
- Також визначається поле **status**, яке приймає значення **'pending'** за допомогою ключового слова **as**.

2. Визначення функції редюсера **chatReducer**, яка приймає поточний стан і дію:

- Функція **chatReducer** обробляє різні дії залежно від типу **action.type**, використовуючи оператор **switch**.
- У разі, коли тип дії дорівнює **'SN/chat/MESSAGES_RECEIVED'**, створюється новий стан з полем **messages**, яке містить початкові повідомлення **state.messages**, а також нові повідомлення з масиву **action.payload.messages**. Кожне повідомлення отримує унікальний ідентифікатор за допомогою **uuidv4()**. Масив повідомлень також обмежується до 100 останніх повідомлень за допомогою **filter**.

- У разі, коли тип дії дорівнює **'SN/chat/STATUS_CHANGED'**, створюється новий стан, у якому поле **status** приймає значення **action.payload.status**.
- У всіх інших випадках повертається поточний стан без змін.

3. Експорт дій, які можна виконувати над редюсером:

- Функції **messagesReceived** і **statusChanged** є діями, які можна виконувати над редюсером.
- **messagesReceived** створює дію з типом **'SN/chat/MESSAGES_RECEIVED'** і передає масив повідомлень як параметр.
- **statusChanged** створює дію з типом **'SN/chat/STATUS_CHANGED'** і передає значення статусу як параметр.

У даному редюсері здійснюється керування станом чату. Початковий стан містить порожній масив повідомлень (**messages**) і статус **'pending'**. Редюсер обробляє дві дії: отримання нових повідомлень (**MESSAGES_RECEIVED**) і зміна статусу (**STATUS_CHANGED**).

При отриманні дії **MESSAGES_RECEIVED**, редюсер додає нові повідомлення до масиву **messages**, присвоюючи кожному повідомленню унікальний ідентифікатор. Для обмеження кількості повідомлень зберігається лише останні 100 повідомлень.

При отриманні дії **STATUS_CHANGED**, редюсер змінює поле **status** на нове значення.

Екшени **messagesReceived** і **statusChanged** використовуються для створення відповідних дій і передачі їх до редюсера.

Усі інші дії повертають поточний стан без змін.

Цей редюсер може використовуватись в спільності Redux для керування станом чатової системи, дозволяючи отримувати повідомлення, змінювати статус і взаємодіяти з API чату.

Редюсер **profileReducer** використовується для керування станом профілю користувача в додатку.

Приклад коду наведений у (додатку Г).

Давайте розглянемо кожну частину коду окремо:

1. **ADD_POST** - додає новий пост до списку постів користувача.
2. **SET_USER_PROFILE** - встановлює профіль користувача.
3. **SET_STATUS** - встановлює статус користувача.
4. **LIKE_POST** - збільшує кількість лайків у пості.
5. **DISLIKE_POST** - зменшує кількість лайків у пості.
6. **DELETE_POST** - видаляє пост зі списку постів користувача.
7. **SAVE_PHOTO_SUCCESS** - оновлює фотографію профілю користувача.

У редюсері також є декілька інших функцій, таких як **addPostActionCreator**, **setUserProfile**, **setStatus**, які створюють відповідні дії для виклику з компонентів і передають необхідні дані до редюсера.

Функція **addPostActionCreator** створює дію для додавання нового поста. В компоненті можна викликати цю функцію та передати текст нового поста, щоб додати його до списку постів.

Файл **store.ts** відповідає за створення та конфігурацію Redux store.

Приклад коду наведений у (додатку Г).

Опис даної компоненти:

1. Комбінування редюсерів в кореневий редюсер за допомогою **combineReducers**.
2. Визначення типів для кореневого редюсера та стану Redux store.
3. Визначення типів для екшенів.
4. Конфігурація Redux DevTools за допомогою **composeEnhancers**.
5. Створення та налаштування Redux store з використанням **createStore**, **applyMiddleware** та **composeEnhancers**.
6. Експорт створеного Redux store.

Даний код створює та налаштовує Redux store з використанням комбінованого кореневого редюсера, додає middleware **redux-thunk** для обробки асинхронних дій, а також налаштовує підтримку Redux DevTools для розширеного зневадження та моніторингу.

Редюсер **usersReducer** відповідає за обробку дій, пов'язаних з користувачами.

Він керує станом користувачів, забезпечуючи оновлення їх даних і виконання дій, пов'язаних з підпискою та відпискою.

Приклад коду наведений у (додатку Г).

Основні елементи коду:

1. Ініціалізація початкового стану **initialState**.
2. Опис редюсера **usersReducer**, який обробляє дії користувачів.
3. Опис типів для екшенів.
4. Опис функцій-екшенів з використанням Thunk.

Даний редюсер (usersReducer) відповідає за обробку дій, пов'язаних з користувачами. Він виконує наступні дії:

1. Оновлення стану користувачів: При отриманні дії "SET_USERS", редюсер оновлює список користувачів в стані, замінюючи його на переданий список користувачів.
2. Оновлення сторінки: При отриманні дії "SET_CURRENT_PAGE", редюсер оновлює значення поточної сторінки користувачів в стані.
3. Оновлення загальної кількості користувачів: При отриманні дії "SET_TOTAL_USERS_COUNT", редюсер оновлює значення загальної кількості користувачів в стані.
4. Зміна статусу завантаження: При отриманні дії "TOGGLE_IS_FETCHING", редюсер оновлює прапор isFetching, який вказує на статус завантаження списку користувачів.
5. Виконання підписки: При отриманні дії "FOLLOW", редюсер оновлює статус підписки на користувача. За допомогою допоміжної функції

updateObjectInArray, редюсер змінює об'єкт користувача в масиві користувачів, встановлюючи значення followed на true.

6. Виконання відписки: При отриманні дії "UNFOLLOW", редюсер оновлює статус відписки від користувача. Знову за допомогою функції updateObjectInArray, редюсер змінює об'єкт користувача в масиві користувачів, встановлюючи значення followed на false.

Таким чином, редюсер usersReducer керує станом користувачів, забезпечуючи оновлення їх даних і виконання дій, пов'язаних з підпискою та відпискою.

У файлі **Users-Selectors** описаний функціонал селекторів.

Даний код використовує бібліотеку Reselect для створення селекторів, які дозволяють отримувати підмножини даних зі стану Redux. Селектори дозволяють кешувати результати обчислень та забезпечують ефективність у випадках, коли велика кількість компонентів використовує однакові дані зі стану.

Приклад коду наведений у (додатку Г).

Опис функцій селекторів:

1. **getUsersSelector**: Селектор, який повертає масив користувачів зі стану (**state.usersPage.users**).
2. **getUsers**: Селектор, який використовує **createSelector** для створення мемоізованої функції. Він отримує дані з селектора **getUsersSelector** і фільтрує користувачів, застосовуючи умову **u => true**. Цей селектор може використовуватись для отримання відфільтрованого списку користувачів.
3. **getPageSize**: Селектор, який повертає розмір сторінки для відображення списку користувачів (**state.usersPage.pageSize**).
4. **getTotalUsersCount**: Селектор, який повертає загальну кількість користувачів (**state.usersPage.totalUsersCount**).
5. **getCurrentPage**: Селектор, який повертає поточну сторінку списку користувачів (**state.usersPage.currentPage**).

6. **getIsFetching**: Селектор, який повертає статус завантаження списку користувачів (**state.usersPage.isFetching**).
7. **getFollowingInProgress**: Селектор, який повертає масив ідентифікаторів користувачів, що перебувають у процесі підписки або відписки (**state.usersPage.followingInProgress**).

Приклад коду наведений у (додатку Г).

Приклади використання:

```
import { useSelector } from 'react-redux';
import { getUsers } from './selectors';
// Використання селектора getUsers для отримання відфільтрованого списку користувачів
const users = useSelector(getUsers);

// Використання інших селекторів
const pageSize = useSelector(getPageSize);
const totalUsersCount = useSelector(getTotalUsersCount);
const currentPage = useSelector(getCurrentPage);
const isFetching = useSelector(getIsFetching);
const followingInProgress = useSelector(getFollowingInProgress);
```

У цих прикладах **useSelector** - це хук React Redux, який дозволяє отримати дані зі стану Redux у компоненті. Вказані селектори допомагають вибирати потрібні дані зі стану, забезпечуючи оптимізацію та перевикористання результатів обчислень.

3.1.3 API (Application Programming Interface).

API (Application Programming Interface) - це набір правил та механізмів, які дозволяють різним програмам та сервісам взаємодіяти між собою. API визначає, як програми можуть запитувати інформацію, надсилати дані та виконувати різні дії в інших програмах або сервісах.

API може бути використаний для доступу до різних ресурсів та функціональності, таких як отримання даних з веб-сервера, взаємодія з базою даних, робота з зовнішніми сервісами (наприклад, соціальні мережі, платіжні системи) та багато іншого.

За допомогою API розробники можуть інтегрувати свої програми з іншими сервісами, створювати розширення, реалізовувати функціональність та взаємодію з різними додатками та платформами.

API може бути представлений у різних форматах, таких як REST (Representational State Transfer), SOAP (Simple Object Access Protocol), GraphQL та інші. Кожен формат має свої особливості та протоколи комунікації [16].

Розробники можуть використовувати публічні API, які надаються різними сервісами, або створювати власні API для використання власними програмами або для надання іншим розробникам доступу до своїх сервісів і даних.

Використання API дозволяє розширювати функціональність програм, покращувати їх взаємодію та забезпечувати інтеграцію з різними сервісами та платформами.

В даній API папці зберігаються файли в яких описана вся основна логіка та функції для взаємодії з северною частиною.

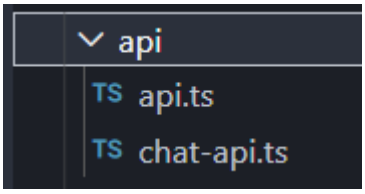

Рис. 3.16. Структура файлів папки API.

UML-діаграма серверної частини веб-платформи соціальної мережі наведена у (додатку В).

У цій діаграмі показано взаємозв'язок між різними частинами серверної частини мого проекту. Кожен модуль API має свої методи, які виконують певні операції, такі як отримання користувачів, робота з профілем, автентифікація, безпека та чат. Деякі типи інтерфейсів також вказані для більшої ясності.

Основні елементи коду:

Приклад коду наведений у (додатку Г).

1. Запит на отримання списку користувачів:

Цей метод **getUsers** використовується для отримання списку користувачів з API. Він приймає два параметри: **currentPage** - номер поточної сторінки (за замовчуванням 1) і **pageSize** - кількість користувачів на сторінці (за замовчуванням 7). Виконується GET-запит до ендпоінту **users**, передаючи параметри **page** і **count**. Результат запиту, який містить список користувачів, повертається у форматі об'єкту **data**.

2. Запит на підписку на користувача:

Цей метод follow використовується для здійснення підписки на користувача з вказаним userId. Виконується POST-запит до ендпоінту follow/{userId}, де {userId} - ідентифікатор користувача, на якого ви хочете підписатися.

3. Запит на відписку від користувача:

Цей метод unfollow використовується для відписки від користувача з вказаним userId. Виконується DELETE-запит до ендпоінту follow/{userId}, де {userId} - ідентифікатор користувача, від якого ви хочете відписатися.

Ці методи використовують бібліотеку Axios для взаємодії з API. instance є екземпляром Axios, створеним з вказаними конфігураціями, такими як baseURL (базовий URL API) та headers (заголовки запиту, включаючи ключ API). Кожен метод виконує відповідний HTTP-запит (GET, POST, DELETE) з використанням instance і повертає проміс, який резолвиться з отриманими даними від сервера.

instance є екземпляром клієнта Axios, який використовується для виконання HTTP-запитів до сервера. При створенні цього екземпляра задаються параметри, такі як наявність креденшалів (withCredentials), базова URL-адреса сервера (baseURL) і заголовки, включаючи API-ключ (API-KEY).

Цей instance використовується для створення об'єктів API, таких як usersAPI, profileAPI, authAPI, securityAPI, які містять методи для взаємодії зі відповідними ендпоінтами сервера.

Приклад коду інтерфейса **instance**:

```
const instance = axios.create({
  withCredentials: true,
  baseURL: 'https://social-network.com/api/1.0/',
  headers: {
    "API-KEY": "bfd52358-f556-49fe-b856-3044468355c0"
  }});
```

Наприклад, для виконання GET-запиту до ендпоінту /users з використанням instance, можна використовувати наступний код:

```
instance.get('/users')
 .then(response => {
   // Обробка відповіді сервера
   console.log(response.data);
 })
 .catch(error => {
   // Обробка помилки
   console.error(error);
 });
```

Такий підхід дозволяє використовувати спільні налаштування (такі як baseURL, headers) для всіх запитів, які виконуються за допомогою instance.

3.1.4. Інфраструктура: Хостинг проекта.

Хостинг проекта на GitHub Pages та Heroku - це дві популярні платформи, які надають можливість розміщення та публікації вашого веб-проекту в Інтернеті. Дозволяючи доступ до вашого проекту через веб-адресу, ці хостинг-платформи допомагають вам зробити ваш проект доступним для широкого кола користувачів.

GitHub Pages - це безкоштовна платформа хостингу, яка дозволяє розмістити статичний веб-сайт, що складається з HTML, CSS та JavaScript файлів, на серверах GitHub. Щоб опублікувати ваш проект на GitHub Pages, вам потрібно створити спеціальний репозиторій на GitHub та завантажити ваші файли проекту до цього репозиторію. GitHub Pages автоматично збирає ваші файли та створює доступний веб-сайт за адресою **https://yourusername.github.io/repositoryname**. Ви також можете налаштувати доменне ім'я для вашого проекту. GitHub Pages дуже зручний для розміщення невеликих проектів, статичних сайтів та документації.

Heroku - це платформа хостингу, яка надає можливість розміщення та розгортання різноманітних веб-додатків, включаючи додатки, що використовують серверний код. Heroku підтримує різні мови програмування та фреймворки, такі як Node.js, Ruby, Python, PHP та інші. Щоб розмістити ваш проект на Heroku, вам потрібно створити обліковий запис та налаштувати вашу локальну розробку для з'єднання з Heroku. Після цього ви можете використовувати Heroku CLI або інші інструменти, щоб розгорнути вашу програму на серверах Heroku. Ваш проект буде доступний за допомогою URL-адреси, яку надає Heroku.

GitHub Pages простий у використанні та безкоштовний. У той же час, Heroku надає більші можливості для розгортання складних веб-додатків, які вимагають серверного коду та баз даних. Обидва варіанти мають свої переваги та обмеження, тому вибір платформи залежить від потреб вашого проекту та вашого рівня експертизи.

Розгортання проекту на GitHub Pages можуть включати наступні кроки:

1. Створюю репозиторій: Створюю новий репозиторій на GitHub, який буде використовуватись для розміщення мого проекту.
2. Клоную репозиторій: Використовую Git для клонування репозиторію на мій комп'ютер.

```
C:\Users\super>git clone https://github.com/sashasa/Social-Network.git
```

3. Прехожу до каталогу проекту: Відкриваю командний рядок або термінал та перехожу до каталогу мого проекту.

```
C:\Users\super>cd soc-netw-repo
```

4. Додаю файли проекту: Копіюю файли мого проекту до клонованого репозиторію.

```
C:\Users\super>xcopy /s /e /i /y "C:\шлях\до\мого\проекту\*" .
```

5. Сворюю гілку **gh-pages**: Виконую наступну команду для створення гілки **gh-pages** у мому репозиторії.

```
C:\Users\super>git checkout -b gh-pages
```

6. Зберігаю та зафіксовую зміни: Виконую команду **git add** для додавання змінених або нових файлів, а потім виконую команду **git commit** для зафіксування змін.

```
C:\Users\super>git add .
```

```
C:\Users\super>git commit -m "Initial commit"
```

7. Завантажую гілку **gh-pages** на GitHub: Вуконую команду **git push** для завантаження гілки **gh-pages** на мій репозиторій на GitHub.

```
C:\Users\super>git push origin gh-pages
```

8. Налаштовую налаштування репозиторію: У репозиторії на GitHub перехожу до вкладки "Settings" (Налаштування) і внизу у розділі "GitHub Pages", вибираю гілку **gh-pages** як джерело для GitHub Pages та зберігаю налаштування.
9. Перевіряю розгорнутий сайт: Мій проект тепер буде доступний за адресою https://sashasa.github.io/Social-Network/.

Це загальний опис процесу розгортання на GitHub Pages. GitHub Pages підтримує різні конфігурації та налаштування, такі як використання власного доменного імені, використання статичного сайту або React-додатку тощо.

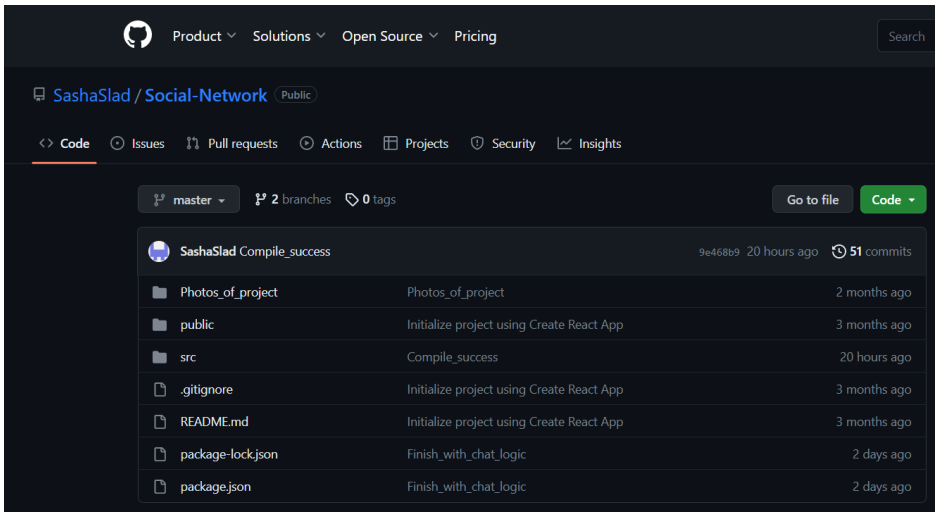

Рис. 3.17. Розміщений репозиторій на GitHub.

```
ПРОБЛЕМЫ   ВЫХОДНЫЕ ДАННЫЕ   ТЕРМИНАЛ   КОНСОЛЬ ОТЛАДКИ

Creating an optimized production build...
Compiled successfully.

File sizes after gzip:

  107.74 kB (+4.03 kB)  build\static\js\main.e7ab6f12.js
  3.03 kB               build\static\js\780.8e255421.chunk.js
  2.71 kB (+1.52 kB)    build\static\css\main.4ba17303.css
  1.78 kB               build\static\js\787.1ef69db0.chunk.js
  926 B (-4 B)          build\static\js\672.01a22042.chunk.js
  780 B                 build\static\css\780.c3b11a4a.chunk.css
  230 B                 build\static\css\672.85d284a1.chunk.css

The project was built assuming it is hosted at /Social-Network/.
You can control this with the homepage field in your package.json.

The build folder is ready to be deployed.

Find out more about deployment here:

  https://cra.link/deployment


> soc-network@0.1.0 deploy
> gh-pages -d build

Published
PS C:\Users\sasha\Desktop\soc-network>
```

Рис. 3.18. Створення оптимузованого білда для gh-pages та хостинг.

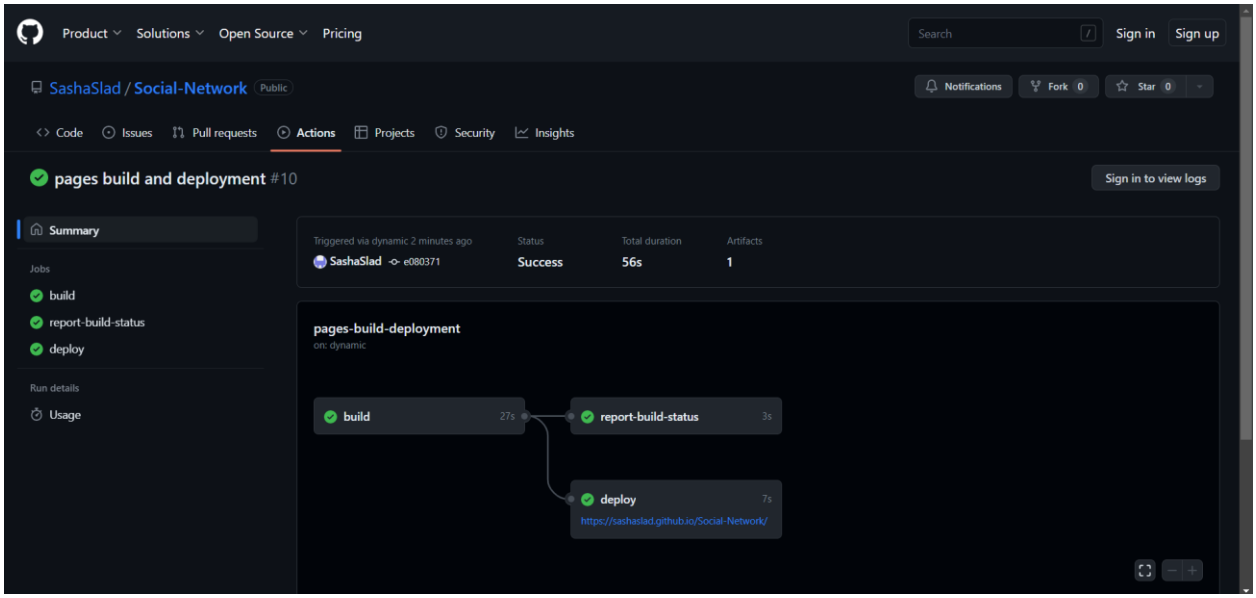

Рис. 3.19. Статус хостингу на GitHub.

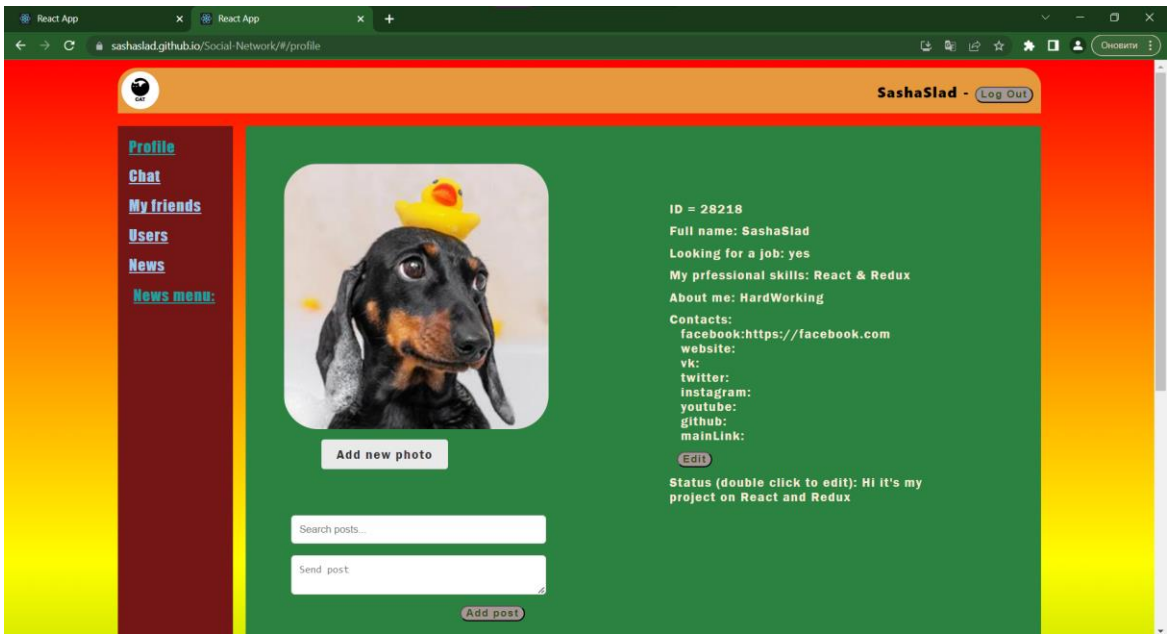

Рис. 3.20. Приклад захощеного сайта.

3.2. Тестування веб-платформи.

Тестування веб-платформи соціальної мережі є невід'ємною частиною процесу розробки і розгортання проекту. Воно допомагає забезпечити якість та надійність платформи, а також впевнитися в її відповідності функціональним вимогам і очікуванням користувачів. Основна мета тестування полягає в перевірці працездатності, безпеки, швидкодії та коректності роботи всіх її компонентів і функціональних можливостей.

При тестуванні веб-платформи соціальної мережі рекомендується використовувати різні підходи і методики, такі як одиничні тести для перевірки окремих компонентів, інтеграційні тести для перевірки взаємодії між компонентами, функціональні тести для перевірки функцій, тести навантаження для перевірки продуктивності та масштабованості, тести безпеки для перевірки захищеності даних тощо.

Окрім цього, важливо враховувати специфіку соціальної мережі під час тестування. Наприклад, перевіряти можливість створення профілів користувачів, додавання друзів, обмін повідомленнями, публікацію постів, відображення новин тощо. Також слід забезпечити тестування на різних платформах, браузерах та пристроях, щоб переконатися, що платформа працює належним чином у різних середовищах.

Важливо також вести систематичне тестування під час розробки, а також планувати регулярні тестові цикли після релізів для виявлення та виправлення можливих проблем. Застосування автоматизованих тестів може спростити процес тестування та забезпечити швидку зворотну інформацію щодо стану платформи.

Узагальнюючи, тестування веб-платформи соціальної мережі є важливою складовою для забезпечення якості та надійності продукту. Воно допомагає виявити і усунути проблеми перед їх впливом на користувачів і забезпечує задоволення їх потреб.

Приклад коду наведений у (додатку Г).

Перейдемо до тестування платформи:

App.test.js

```
import { render, screen } from '@testing-library/react';
import SocNetworkApp from './App';
test('renders learn react link', () => {
  render(<SocNetworkApp />);
});
```

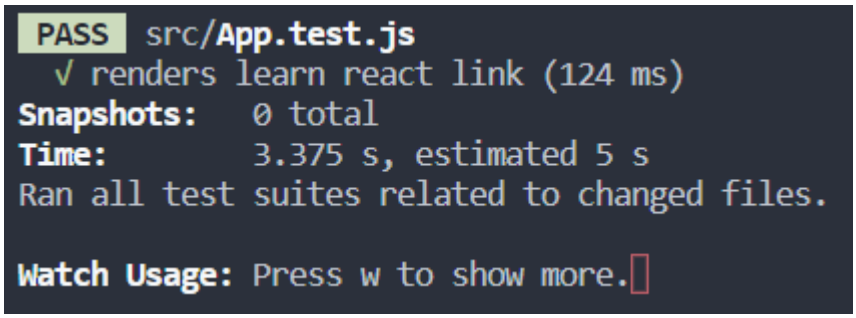

Рис. 3.21. Успішність виконання тесту **App.test.js.**

У наведеному коді використовується тестувальна функція для перевірки, чи компонента **SocNetworkApp** може бути успішно відрендерена без помилок.

Цей код використовується як початковий тест для перевірки, чи компонента **SocNetworkApp** може бути успішно відрендерена без виникнення помилок.

ProfileStatus.test.js

Даний файл містить набір тестів для компоненти **ProfileStatus**, де перевіряються різні аспекти функціональності даної компоненти.

Приклад коду наведений у (додатку Г).

1. Тест "status from props should be in the state": В цьому тесті перевіряється, чи правильно переданий статус через пропси встановлюється в стані компоненти. Створюється екземпляр компоненти ProfileStatus зі статусом "hellow", і потім перевіряється, чи відповідає значення статусу в стані переданому значенню з пропсів.
2. Тест "after creation should be displayed": Цей тест перевіряє, чи після створення компоненти присутній елемент . Створюється екземпляр компоненти ProfileStatus зі статусом "hellow", і потім перевіряється, чи знайдено елемент в структурі компоненти.
3. Тест "after creation <input/> shouldn't be displayed": У цьому тесті перевіряється, чи після створення компоненти відсутній елемент **<input>**. Створюється екземпляр компоненти **ProfileStatus** зі статусом "hellow", і потім виконується перевірка на відсутність елементу **<input>** в структурі компоненти.
4. Тест "after creation should contains correct status": У цьому тесті перевіряється, чи елемент **** містить правильне значення статусу. Створюється екземпляр компоненти **ProfileStatus** зі статусом "hellow", і потім перевіряється, чи значення тексту в елементі **** відповідає переданому статусу.
5. Тест "input should be displayed in EditMode instead of span": В цьому тесті перевіряється, чи елемент **<input>** відображається після подвійного кліку на елемент ****. Створюється екземпляр компоненти **ProfileStatus** зі статусом "hellow", потім емулюється подвійний клік на елемент ****, і перевіряється наявність елементу **<input>** з правильним значенням статусу.
6. Тест "callback should be called": У цьому тесті перевіряється, чи викликається передана функція зворотного виклику при виконанні певної дії. Створюється екземпляр компоненти **ProfileStatus** зі статусом "hellow" і переданою функцією **updateStatus** (mockCallback). Потім емулюється виклик методу **deactivateEditMode** у екземплярі

компоненти, і перевіряється, чи функція **mockCallback** була викликана один раз.

Ці тести перевіряють різні аспекти функціональності компоненти **ProfileStatus** і допомагають переконатися, що вона працює правильно.

Paginator.test.jsx

Даний файл містить набір тестів для компоненти **Paginator**, яка використовується для відображення сторінок у пагінації. Тести перевіряють різні аспекти поведінки компоненти:

Приклад коду наведений у (додатку Г).

1. Pages count is 11 but should be showed only 10 - перевіряє, чи правильно відображається кількість сторінок. У цьому випадку передається загальна кількість елементів **11**, розмір сторінки **1** та розмір порції **10**, і очікується, що буде відображено тільки **10** сторінок.
2. If pages count is more than 10 button NEXT should be present - перевіряє, чи присутня кнопка "NEXT" при наявності більше **10** сторінок. В даному тесті також передається загальна кількість елементів **11**, розмір сторінки **1** та розмір порції **10**, і очікується, що присутня буде лише одна кнопка.

Кожен тест використовує бібліотеку **react-test-renderer** для створення віртуального дерева компонентів та перевірки різних аспектів поведінки компоненти.

Profile-reducer.tets.js

У цьому файлі містяться тести для перевірки роботи редюсера профілю. Редюсер відповідає за зміну стану профілю в залежності від дій користувача.

Приклад коду наведений у (додатку Г).

1. new post should be added, length of posts should be incremented - перевіряє, чи додається новий пост, тобто перевіряє збільшення довжини масиву postsData після додавання поста.
2. new post message should be correct - перевіряє, чи правильно встановлюється повідомлення нового поста, тобто перевіряє, чи відповідний елемент масиву postsData має правильне повідомлення.

3. after deleting length of messages should be decrement - перевіряє, чи відбувається зменшення довжини масиву postsData після видалення поста.
4. after deleting length of messages should be decrement if id is incorrect - перевіряє, чи залишається довжина масиву postsData незмінною, якщо переданий неправильний ідентифікатор для видалення поста.

Ці тести допомагають переконатися, що редюсер працює належним чином, зберігаючи правильний стан профілю та виконуючи очікувані дії при додаванні та видаленні постів.

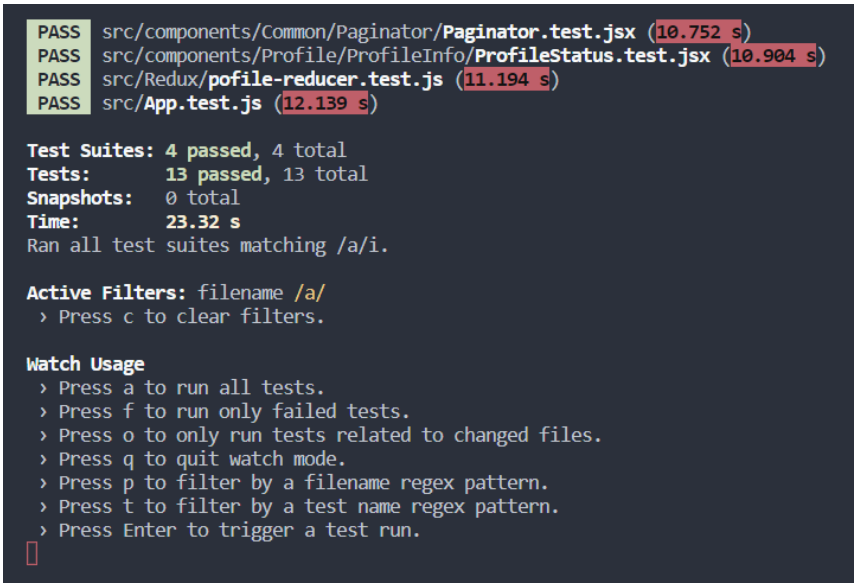

Рис. 3.22. Успішність виконання всіх тестів.

3.3. Висновки до розділу.

У даному розділі було розглянуто архітектуру веб-платформи для соціальної мережі. За допомогою клієнт-серверної моделі, користувачі взаємодіють з сервером для отримання інформації та виконання різних дій. Основні компоненти архітектури включають клієнтську та серверну частини, API та інфраструктуру.

Клієнтська частина включає користувацький інтерфейс, функціональні модулі та клієнтську сторону взаємодії з сервером. Користувацький інтерфейс забезпечує взаємодію користувача з платформою, а функціональні модулі реалізують основні функції соціальної мережі. Клієнтська сторона взаємодії з сервером відповідає за передачу запитів серверу та обробку отриманих відповідей.

Серверна частина включає серверний додаток, базу даних та сервіси. Серверний додаток приймає та обробляє запити від клієнтів, виконує бізнес-логіку та взаємодіє з базою даних та іншими сервісами. База даних зберігає дані користувачів, профілі, підписки, повідомлення та інші важливі дані. Сервіси включають функціонал для аутентифікації, авторизації, обробки данних.

API визначає набір доступних запитів та формат обміну даними між клієнтською та серверною частинами. Це може бути реалізовано за допомогою REST протоколів.

Інфраструктура включає хостинг, що забезпечує розміщення серверної частини платформи в Інтернеті, та масштабованість, яка розробляється з урахуванням можливостей масштабування платформи, таких як горизонтальне масштабування та використання хмарних сервісів.

В ході реалізації проекту було успішно протестовано функціональність кожного компонента, включаючи роботу редюсерів, запити на сервер. Всі ці компоненти разом допомогли створити функціональну та надійну веб-платформу для соціальної мережі, що задовольняє потреби користувачів та забезпечує їм зручний інтерфейс для спілкування та обміну інформацією

РОЗДІЛ 4. ЕКОНОМІЧНЕ ОБҐРУНТУВАННЯ РОЗРОБКИ СИСТЕМИ

4.1. Сутність проектного рішення веб-платформи для соціальної мережі.

Даний розділ бакалаврської кваліфікаційної роботи націлений на проведення економічного обґрунтування доцільності розробки веб-платформи для соціальної мережі. Під час написання даного розділу будуть здійснені розрахунки витрат на розробку програмного забезпечення, веб-інтерфейсу, серверної інфраструктури та інших необхідних компонентів. Важливим етапом буде складання бюджету проектного рішення, де враховуватимуться витрати на розробку, тестування, підтримку та просування веб-платформи.

Для оцінки економічної ефективності проектування веб-платформи будуть проведені дослідження ринку та аналіз конкурентних продуктів. Окрім того, будуть визначені ключові показники доцільності проекту, такі як термін окупності, чистий приведений дохід, рентабельність та інші. Застосовуватимуться методи фінансового аналізу, такі як чистий теперішній дохід, внутрішня норма повернення, окупність інвестицій та інші, для оцінки ризиків та прийняття обґрунтованих рішень щодо продовження проекту.

У результаті проведення аналізу та оцінки економічної ефективності веб-платформи для соціальної мережі будуть отримані дані, які допоможуть прийняти обґрунтоване рішення щодо продовження розробки та запуску продукту на ринку.

4.2. Розрахунок витрат на виконання дослідження за БКР.

До цієї статті належать витрати на основну та додаткову заробітну плату науковим керівникам, інженерно-технічним працівникам, лаборантам, робітникам, студентам, аспірантам, операторам ЕОМ та іншим працівникам, безпосередньо зайнятим виконанням даної БКР, обчислені за посадовими окладами та тарифними ставками, відрядними розцінками для робітників, включаючи преміальні виплати. Вихідні дані наводяться в табл. 4.1.

Середньоденна ставка зарплати ($Сд_i$) для кожного з виконавців (i) дорівнює:

$$Сді = ЗПі/Fр \qquad (4.1)$$

де $F_р$ - місячний фонд робочого часу (24 дні).

Таблиця 4.1

Вихідні дані для розрахунку заробітної плати

№ з/п	Посада виконавців	Місячний оклад, грн. (основ. ЗП)	Місячна тарифна ставка, грн. (основ. ЗП)	Додаткова зарплата, грн.	Премія, грн.	Середньо-місячна ЗП, грн. (сума 3-6)	Середньо-денна ставка, грн./дні
1	Керівник роботи (КР)	12500,00	–	4500,00	–	17000,00	708,33
2	Консультант з економ. частини (КЕ)	11500,00	–	4140,00	–	15640,00	651,67
3	Студент-дипломник	3500,00	–	–	–	3500,00	145,83

Для розрахунку витрат на оплату праці виконавців даної НДДКР (табл. 4.2) визначається трудомісткість роботи кожного з працівників ($Т_і$).

До додаткової зарплати керівника дипломної роботи (КР) та консультанта з економічних питань (КЕ) входять надбавки за наукову ступінь (для кандидата наук 15%, доктора наук – 20%), за наукове звання (для доцента 25%, для професора 30%) та за науково-педагогічний стаж.

Таблиця 4.2

Розрахунок витрат на оплату праці виконавців

№ з/п	Посади виконавців	$Сд_і$ грн./дні	$Т_і$ люд./дні	Витрати на оплату праці ($В_{оп}$), грн.
1	КР	708,33	7	4958,33
2	КЕ	651,67	1	651,67
3	Студ.-дипл.	145,83	55	8020,65
			Разом:	13 630,65

Відрахування на соціальні заходи

Єдиний соціальний внесок у частині нарахувань установлюється в розмірі 22%:

$$B_{сз}=(B_{оп}-B_{оп\ студ.})\cdot 0{,}22=5611\cdot 0{,}23 = 1290{,}5\ (грн.)$$

Розрахунок амортизаційних відрахувань та витрат на енергію для наукових цілей

Вартість однієї машино-години $Ц_м$ обраховується за формулою:

$$Ц_м = B\cdot H_а /(100\cdot N)+ P_к\cdot Ц_е \qquad (4.2)$$

де N − кількість годин роботи комп'ютера (принтера); B − вартість комп'ютера (принтера); $H_а$ − амортизаційні відрахування; $P_к$ − потужність кВт/год; $Ц_е$ – ціна, грн./кВт/год.

Порахуємо для одного комп'ютера:

$$Ц_{м.к} = 22000\cdot(30/(100\cdot 2740)) + 0{,}12\cdot 1{,}67 = 2{,}96\ (грн/год).$$

Порахуємо для одного принтера:

$$Ц_{м.к} = 5300\cdot(19/(100\cdot 1000)) + 0{,}36\cdot 1{,}67 = 1{,}71\ (грн/год).$$

Результати розрахунків витрат на амортизацію та енергію для наукових цілей занесено в табл. 4.3, а розрахунку витрат на носії інформації – в табл. 4.4.

Таблиця 4.3

Розрахунок витрат на амортизацію та енергію для наукових цілей ($B_{ен}$)

№ з/п	Назва устаткування	$Ц_м$, грн./кВтгод.	N, маш/год	$B_{ен}$, грн.
1	Комп'ютер	2,96	16	47,36
2	Принтер	1.71	1	1,71
			Разом:	49,07

Таблиця 4.4

Розрахунок витрат на носії інформації

№ з/п	Найменування (вид) матеріалу	Одиниця виміру	H_i, од.	*Ці*, грн./од.	$B_{мі}$, грн.
1	Папір	пачка	1	210	210
2	Флеш-пам'ять	штука	1	200	200
3	Ручки	штука	1	12	12
				Разом:	422

Накладні витрати

До складу накладних витрат ($B_н$) відносяться:

- витрати, пов'язані з управлінням організацією, де проводиться НДДКР;
- витрати на науково-технічну інформацію;
- витрати на забезпечення нормальних умов праці і техніки безпеки;
- витрати на інші загальногосподарські потреби тощо.

Накладні витрати розраховуються у відсотках до витрат на оплату праці ($B_{оп}$):

$$B_н = B_{оп} га/100 \qquad (4.3)$$

де *а* - середньостатистичний відсоток накладних витрат в організації (155%).

$$B_н = 13530{,}64 \times 154/100 = 21021{,}52 \text{ (грн.)}$$

Розрахунок калькуляції кошторисної вартості розробки

Результати розрахунку по всіх статтях наводяться в табл. 4.5 і складають кошторисну вартість виконання НДДКР (*К*):

$$K = B_{оп} + B_{сз} + B_{су} + B_і + B_н \qquad (4.4)$$

Розрахунок витрат на експлуатацію проектного рішення

Експлуатаційні одноразові витрати (*Е*) для проектного рішення включають вартість підготовки даних (*Е*1) і вартість машино-годин роботи ПЕОМ (*Е*2) (грн.):

$$E=E1+E2 \quad (4.5)$$

Таблиця 4.5

Калькуляція кошторисної вартості проектного рішення

№ з/п	Статті витрат	Сума, грн.
1.	Витрати на оплату праці	13530,64
2.	Відрахування на соціальні заходи	1290,5
3.	Інші витрати, в т.ч. :	1997,62
3.1	Амортизаційні відрахування і енергія для наукових цілей	78,06
3.2	Витрати на носії інформації	421,00
4.	Накладні витрати	21021,51
	Всього:	38587,07

Річні експлуатаційні витрати

$$E_P = E \cdot N \quad (4.6)$$

де N - періодичність експлуатації проектного рішення, разів (N = 320).

Вартість підготовки даних для роботи на ПЕОМ ($E1$) визначають за формулою:

$$E1 = n \cdot t \cdot c \qquad (4.7)$$

де n – чисельність співробітників, які підготовлюють дані, осіб; t – час роботи співробітників із підготовки даних, год.; c – середньо-годинна ставка співробітника, грн./год. (c = 144,82/8 = 18,23 грн./год. – див. табл. 4.1).

Витрати на експлуатацію ПЕОМ ($E2$) визначається за формулою:

$$E2 = t \cdot S_{м\text{-}г} \qquad (4.7)$$

де t – машино-години роботи ПЕОМ для одноразової експлуатації проектного рішення, год.; $S_{м\text{-}г}$ – вартість 1 машино-години роботи ПЕОМ конкретного типу, грн./год. ($S_{м\text{-}г}$ = 3,62+ 0,39 = 4,01 грн./год.)

В нашому випадку: n = 1; t = 7 год. – максимальний час необхідний для підготовки даних.

Отже,

$$E1 = 1 \cdot 7 \cdot 18,23 = 126,22 \text{ (грн.)},$$

$$E2 = 7 \cdot 4,01 = 20,05 \text{ (грн.)}.$$

Тоді

$$E = 127{,}23 + 20{,}05 = 146{,}27 \text{ (грн.)},$$

$$E_P = 147{,}28 \cdot 55 = 8090{,}4 \text{ (грн.)}.$$

4.3. Оцінка економічної ефективності проектного рішення.

Розрахунок ціни споживання проектного рішення

Ціна споживання ($Ц_с$) – це витрати на придбання і експлуатацію проект-ного рішення за весь строк його служби:

$$Ц_{с(п)} = Ц_п + B_{(е)п\ pv} \qquad (4.9)$$

де $Ц_п$ – ціна придбання проектного рішення, грн.:

$$Ц_п = K \cdot (1 + П_р/100) + K_о + K_к \qquad (4.10)$$

де K – витрати на розробку і впровадження проектного рішення; $П_р$ – норма рентабельності (23%); $K_о$ – витрати на прив'язку та освоєння проектного рішення на конкретному об'єкті.

K (витрати на розробку і впровадження проектного рішення) = 37984,99,

$K_о$ включає:

- інсталяцію програмного продукту на ПК (600 грн.);
- документацію на програмний продукт (1200 грн.);
- демонстрацію роботи програмного продукту (400грн.).

Загальна сума витрат на прив’язку та освоєння проектного рішення на конкретному об’єкті:

$$K_о = 600 + 1200 + 400 = 2200 \text{ (грн.)}.$$

де $K_к$ – витрати на доукомплектування технічних засобів на об'єкті (дорівнює нулю).

Таким чином, ціна придбання проектного рішення дорівнює:

$$Ц_п = 38587{,}37 \cdot (1 + 0{,}23) + 2200 = 49662{,}54 \text{ (грн.)}.$$

Теперішня вартість витрат на експлуатацію проектного рішення (за весь час його експлуатації ($B_{(е)п\ pv}$), грн.:

$$B_{(е)п\ pv} = \sum_{t=0}^{T} \frac{B_{(е)п\_t}}{(1+R)^t} \qquad (4.11)$$

де $B_{(е)п\_t}$ - річні експлуатаційні витрати в *t*-ому році, грн.; *T* - строк служби проектного рішення, 7 років; *R* - річна облікова ставка НБУ (тут приймається рівним 0,17 і залишається незмінною протягом експлуатації).

Для даних *R* і *T* отримаємо наступне значення коефіцієнта дисконту:

pv=1/(1+0,17)^1+1/(1+0,17)^2+1/(1+0,17)^3+1/(1+0,17)^4+1/(1+0,17)^5+1/(1+0,17)^6+1/(1+0,17)^7= 3,912

Якщо впродовж всього строку експлуатації $B_{(е)п\_t}$ = const, то:

$$B_{(е)п}{}^{pv} = B_{(е)п} \cdot \sum_{t=0}^{T} \frac{1}{(1+R)^t} = pv \cdot Ер \quad (4.12)$$

де *pv* – коефіцієнт дисконту на період *T*, який визначається залежно від процентної ставки *R* і періоду експлуатації *T* (для нашого випадку *T* =7 років).

$$B_{(е)п}{}^{pv} = 3,912 \cdot 8090,4 = 31236,8 \text{ (грн.)}$$

Тоді ціна споживання проектного рішення дорівнюватиме:

$$Ц_{с(п)} = 49662,54 + 31236,8 = 8085,34 \text{ (грн.)}.$$

4.4. Оцінка науково-технічної ефективності досліджень у БКР.

В залежності від виду виконуваних в дипломній роботі досліджень та поставлених цілей, для підсумкової оцінки результатів вибирається один з перелічених ефектів, а інші використовуються як додаткові характеристики. Як правило, в межах виконуваних БКР характерним є отримання наукового та науково-технічного ефекту, який оцінюється коефіцієнтом науково-технічної ефективності:

$$K_{НТЕ} = \frac{\sum_{j=1}^{n} \alpha_j^{н} \overline{Б}_{jk}}{\sum_{j=1}^{n} \alpha_j^{н} Б_{jk\,\max}} \quad \begin{matrix} j = 1,2,...,n \\ k = 1,2,...,l, \end{matrix} \quad (4.13)$$

де $a_j^{н}$ - нормована величина коефіцієнта вагомості *j*-го фактору науково-технічної ефективності;

Бjk− середнє значення бальної оцінки, яка виставляється експертами *k*-ій якості *j*-го фактору; $Б_{jkmax}$ − максимально можлива величина бальної оцінки; *l* − кількість якостей, які характеризують *j*-ий фактор; *n* − кількість факторів.

Чим більше значення розрахованого $К_{НТЕ}$, тим вищий рівень науково-технічної ефективності НДДКР. Оцінка $К_{НТЕ}$ здійснюється експертним шляхом не менш як трьома експертами за десятибальною шкалою. Бjk визначається як середнє арифметичне.

Нормовані значення коефіцієнтів вагомості для факторів науково-технічної ефективності наведено в табл. 4.6, а результати оцінки науково-технічної ефективності НДДКР – в табл. 4.7.

Таблиця 4.6

Нормовані значення коефіцієнтів вагомості для факторів науково-технічної ефективності пропонованого проектного рішення

№ з/п	Фактор (*j*)	$a_j^н$
1	Новизна очікуваних або одержаних результатів	0,14
2	Глибина наукової проробки	0,13
3	Ступінь ймовірності успіху	0,22
4	Перспективність використання результатів	0,08
5	Масштаб можливої реалізації результатів	0,21
6	Завершеність одержаних результатів	0,22
	Разом:	1,0

Таблиця 4.7

Результати оцінки науково-технічної ефективності пропонованого проектного рішення

№ з/п	Фактори науково-технічної ефективності	Якість фактора	*Експертні оцінки*			$\overline{Б}_{jk}$	$Б_{jk\ max}$
			1	2	3		
1	Новизна очікуваних або одержаних результатів	недостатня	3	3	3	3	3
2	Глибина наукової проробки	середня	6	5	6	5,66	6
3	Ступінь ймовірності успіху	значна	7	9	8	8	9
4	Перспективність використання результатів	важлива	8	7	7	7,33	8
5	Масштаб можливої реалізації результатів	організаційний	4	4	4	4	4
6	Завершеність одержаних результатів	середня	7	6	7	6,66	7
							$K_{НТЕ} = 0,937$

$$K_{НТЕ} = \frac{0,14*3+0,13*5,66+0,22*8+0,08*7,33+0,21*4+0,22*6,66}{0,14*3+0,13*6+0,22*9+0,08*8+0,21*4+0,22*7} = 0,947$$

4.5. Висновки до розділу.

1. Для визначення витрат на проведення реалізації веб-платформи соціальної мережі, складена калькуляція кошторисної вартості робіт за статтями витрат на оплату праці, відрахування на соціальні заходи, накладні витрати. Кошторисна вартість досліджень складає 38686,06 грн.

2. Річні витрати на експлуатацію проектного рішення становлять 31236,8 грн., а ціна його споживання дорівнює 81853,34 (грн.).

3. Також було здійснено оцінку науково-технічної ефективності проведених робіт. Отримання наукового і науково-технічного ефекту визначено за допомогою коефіцієнта науково-технічної ефективності, окремі фактори якого оцінені

експертним шляхом. Значення вище згаданого коефіцієнта становить 0,937, що свідчить про високий рівень науково-технічної ефективності проведених робіт.

ВИСНОВКИ

У даній бакалаврській кваліфікаційній роботі була проведена розробка веб-платформи для соціальної мережі. Робота складається з наступних розділів: вступ, аналіз існуючих рішень, постановка завдання, реалізація пректу, економічне обґрунтування розробки.

В ході аналізу існуючих рішень було виявлено, що на ринку існує попит на соціальні мережі, які надають розширені функціональні можливості та забезпечують зручний користувацький досвід. З метою задоволення цього попиту була поставлена мета розробити веб-платформу, яка буде відповідати потребам користувачів та матиме конкурентні переваги.

У розділі з постановки завдання були визначені основні функціональні та нефункціональні вимоги до системи. Ці вимоги слугували основою для подальшої розробки архітектури системи.

Розділ, присвячений реалізації проекту, описує клієнт-серверну модель, ключові компоненти системи та способи взаємодії між ними. Були розглянуті клієнтська та серверна частини, API та інфраструктура системи. Завдяки цим компонентам користувачі мають можливість взаємодіяти з платформою та виконувати різні дії.

У розділі економічного обґрунтування було проведено розрахунок витрат на розробку, тестування та підтримку веб-платформи. Також було проведено оцінку економічної ефективності проекту, зокрема рентабельності та терміну окупності. Отримані дані допомагають прийняти обґрунтоване рішення щодо продовження розробки та запуску продукту на ринку.

Розроблена веб-платформа для соціальної мережі є перспективним проектом з економічною обґрунтованістю. Вона відповідає вимогам ринку та має конкурентні переваги. Проект має високий рівень науково-технічної ефективності, а його реалізація передбачає прибутковість і окупність. Результати дослідження та аналізу свідчать про доцільність продовження розробки та введення веб-платформи для соціальної мережі на ринок

СПИСОК ЛІТЕРАТУРИ

1. Балабанов, А. В. "Основи програмування мовою PHP." Видавничий дім "Києво-Могилянська академія", 2010. - 356 с.
2. Сурин, О. І. "Розробка веб-додатків з використанням JavaScript." К.: Вид-во "Либідь", 2015. - 310 с.
3. Скуратовський, В. Л. "Сучасні веб-технології: HTML, CSS, JavaScript, PHP, MySQL." К.: Кондор, 2013. - 512 с.
4. Лисенко, М. М. "Основи веб-дизайну та розробки веб-сайтів." К.: Вид-во "Знання", 2014. - 290 с.
5. Петров, В. С. "Програмування для Інтернету: створення динамічних веб-сторінок." Харків: Фоліо, 2012. - 420 с.
6. Коваленко, І. О. "Архітектура та проектування інформаційних систем." К.: Вид-во "Наукова думка", 2011. - 384 с.
7. Григорович, О. П. "Веб-програмування: підручник." Львів: Видавництво Львівської політехніки, 2016. - 450 с.
8. Ткаченко, В. А. "Безпека веб-додатків." К.: Кондор, 2015. - 298 с.
9. Кравець, І. В. "Веб-дизайн: теорія і практика." Одеса: Астропринт, 2014. - 320 с.
10. Бондаренко, А. В. "Розробка і адміністрування веб-сайтів." Харків: Ранок, 2012. - 275 с.
11. Михайлов, С. І. "Вступ до веб-технологій." К.: Академвидав, 2013. - 210 с.
12. Чорний, Д. П. "Веб-програмування на мові Python." Львів: ЛНУ ім. І. Франка, 2014. - 290 с.
13. Вишневський, В. В. "Основи веб-розробки з використанням ASP.NET." Харків: Основа, 2011. - 305 с.
14. Костюк, В. Г. "Сучасні методи розробки веб-додатків." К.: Видавництво КНТ, 2016. - 278 с.
15. Назаренко, О. А. "Основи проектування інформаційних систем." К.: Вид-во КПІ ім. Ігоря Сікорського, 2015. - 340 с.

16. Мельник, П. В. "Технології веб-програмування." Харків: ХНУ ім. В. Н. Каразіна, 2013. - 310 с.
17. Іванов, В. І. "Веб-дизайн та розробка мультимедійних додатків." К.: МАУП, 2014. - 295 с.
18. Шевченко, О. М. "Основи програмування на JavaScript." Дніпро: Вид-во ДНУ, 2016. - 288 с.
19. Чернов, А. В. "Проектування та розробка баз даних для веб-додатків." Львів: ЛНТУ, 2015. - 365 с.
20. Гончаренко, В. В. "Інтернет-програмування: теорія та практика." К.: Політехніка, 2013. - 400 с.
21. Печенізький, О. М. "Адміністрування серверів для веб-додатків." Одеса: ВМВ, 2012. - 315 с.
22. Воробйов, О. Г. "Основи веб-аналітики." Харків: ХНУРЕ, 2014. - 275 с.
23. Білоус, Ю. І. "Створення веб-додатків з використанням Ruby on Rails." Львів: БаК, 2015. - 280 с.
24. Савченко, І. М. "Розробка клієнт-серверних додатків." К.: Знання, 2011. - 305 с.
25. Клименко, С. В. "Професійна розробка веб-сайтів на платформі .NET." К.: Видавництво КМУ, 2014. - 320 с.

ДОДАТОК

Додаток А. Скетч для пристрою.

Index.js

```
import React from 'react';
import * as ReactDOM from 'react-dom/client';
import './index.css';
import reportWebVitals from './reportWebVitals';
import SocNetworkApp from './App';
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
   <React.StrictMode>
      <SocNetworkApp />
   </React.StrictMode>
);
reportWebVitals();
```

App.js

```
const DialogsContainer = React.lazy(() =>
import('./components/Dialogs/DialogsContainer'));
const ProfileContainer = React.lazy(() =>
import('./components/Profile/ProfileContainer'));
// const CahtPage = React.lazy(() => import('./pages/Chat/ChatPage.tsx'));
class App extends Component {
   componentDidMount() {
      this.props.initializeApp();
   }
   render() {
      if (!this.props.initialized) {
         return <Preloader />
      }
      return (
         <div className="app-wrapper" >
            <HeaderContainer />
            <Navbar />
            <div className='app-wrapper-content'>
               <Routes>
                  <Route exact path='/profile/:userId' element={<Suspense
fallback={<div>Loading...</div>} ><ProfileContainer /></Suspense>} />
```

```
                    <Route exact path='*' element={<Suspense
fallback={<div>Loading...</div>} ><ProfileContainer /></Suspense>} />
                    <Route exact path='/dialogs' element={<Suspense
fallback={<div>Loading...</div>} > <DialogsContainer /> </Suspense>} />
                    <Route exact path='/chat' element={<ChatPage />} />
                    <Route exact path='/users' element={<UsersContainer />}
/>
                    <Route exact path='/login' element={<Login />} />
                    <Route exact path='/news' element={<NewsList />} />
                    <Route exact path='/registration' element={<Registration
/>} />
                    <Route exact path='/friends' element={<Friends />} />
                    <Route exact path='/news/sport' element={<SportList />}
/>
                    <Route exact path='/news/ukraine' element={<UkraineList
/>} />
                    <Route exact path='/news/usa' element={<USANewsList />}
/>
                    <Route exact path='/news/animals'
element={<AnimalsNewsList />} />
                    <Route exact path='/news/science'
element={<ScienceNewsList />} />
                </Routes>
            </div>
        </div>
    )
  }
}
const mapStateToProps = (state) => ({
  initialized: state.app.initialized
})
let AppContainer = compose(
  withRouter,
  connect(mapStateToProps, { initializeApp }))(App);
const SocNetworkApp = (props) => {
  return (
    <HashRouter>
      <Provider store={store}>
```

```
            <AppContainer />
         </Provider>
      </HashRouter>
   )}
export default SocNetworkApp;

Post.jsx
const Post = (props) => {
   const [userPhoto, setUserPhoto] = useState('');
   const [userId, setUserId] = useState('');
   const handleDelete = () => {
      props.deletePost(props.id);
   }

   const handleLike = () => {
      props.likePost(props.id);
   };
   const handleDislike = () => {
      props.dislikePost(props.id);
   };
   useEffect(() => {
      const fetchUserPhoto = async () => {
         try {
            const response = await axios.get(`https://social-
network.samuraijs.com/api/1.0/profile/${userId}`, {
               headers: {
                  "API-KEY": "bfd52358-f556-49fe-b856-3044468355c0"
               }
            });
            const user = response.data;
            setUserPhoto(user.photos.small || ''); // Set user's photo or
empty string if photo is not available
         } catch (error) {
            console.log(error);
         }
      };
      fetchUserPhoto();
   }, [userId]);
```

```
   useEffect(() => {
      const fetchUserId = async () => {
         try {
            const response = await axios.get('https://social-
network.samuraijs.com/api/1.0/auth/me', {
               withCredentials: true,
               headers: {
                  "API-KEY": "bfd52358-f556-49fe-b856-3044468355c0"
               }
            });
            const { data } = response;
            if (data.resultCode === 0) {
               const { id } = data.data; // Assuming the response data has
the 'id' property
               setUserId(id);
            }
         } catch (error) {
            console.log(error);
         }
      };
      fetchUserId();
   }, []);
   return (
      <div className={s.item}>
         <div className={s.items}>
            <div className={s.photo}>
               <img alt='' src={userPhoto} />
            </div>

            <div className={s.postinfo}>
               <br />
               <span>Post {props.id}</span>
               <br />
               <span>{props.message}</span>
               <br />
               <span>Likes: {props.likes}</span>
            </div>
         </div>
```

```
            <div className={s.btns}>
               <button onClick={handleLike}>❤</button>
               <button onClick={handleDislike}>👎</button>
               <button className={s.trashbtn}
onClick={handleDelete}>Delete&#128465;</button>
            </div>
         </div >

   );
}
export default Post;
```

MyPosts.jsx

```
const MyPosts = React.memo(props => {
   const [searchQuery, setSearchQuery] = useState('');
   const handleSearch = (event) => {
      setSearchQuery(event.target.value);
   }
   const filteredPosts = props.postsData.filter(post =>
      post.message.toLowerCase().includes(searchQuery.toLowerCase())
   );
   let postsElements = filteredPosts.map(post => (
      <Post
         key={post.id}
         message={post.message}
         likes={post.likes}
         id={post.id}
         // userId={userId}
         deletePost={props.deletePost}
         likePost={props.likePost}
         dislikePost={props.dislikePost}
      />
   ));
   let onAddPost = (values) => {
      props.addPost(values.newPostText);
   }
   const maxLength10 = maxLengthCreator(50);
   const addNewPostForm = (props) => {
```

```
      return (
         <form onSubmit={props.handleSubmit}>
            <Field
               component={Textarea}
               name="newPostText"
               placeholder="Send post"
               validate={[required, maxLength10]}
               className={s.textarea}
            />
            <button className={s.addpostbtn}>Add post</button>
         </form>
      )
   }

   const AddPostFormRedux = reduxForm({ form: 'profileAddNewPostForm'
})(addNewPostForm);
   return (
      <div className={s.myPosts}>
         <div className={s.search}>
            <input
               type="text"
               placeholder="Search posts..."
               value={searchQuery}
               onChange={handleSearch}
               className={s.textarea}
            />
         </div>
         <div className={s.textArea}>
            <AddPostFormRedux onSubmit={onAddPost} />
         </div>
         <div className={s.posts}>
            {postsElements}
         </div>
      </div>
   )
});

export default MyPosts;
```

ProfileDataForm.jsx

```
const ProfileDataForm = (props, { initialValues }) => {
   return (
      <form onSubmit={props.handleSubmit} >
         <div><button>Save:</button></div>
         {
            props.error && <div className={style.formSummaryError}>
               {props.error}
            </div>
         }
         <div> ID = {props.profile.userId} </div>
         <div>
            <div>
               <b>Full name:</b> {createField("Full name", "fullName", [],
Input)}
            </div>
            <div>
               <b>Looking for a job:</b> {createField("",
"lookingForAJob", [], Input, { type: "checkbox" })}
            </div>
            <div>
               <b>My prfessional skills:</b> {createField("My prfessional
skills", "lookingForAJobDescription", [], Textarea)}
            </div>
            <div>
               <b>About me:</b> {createField("About me", "aboutMe", [],
Textarea)}
            </div>
            <div>

<b>Contacts:</b>{Object.keys(props.profile.contacts).map(key => {
                  return (
                     <div key={key} className={s.contacts}>
                        <b>{key}:</b>{createField(key, "contacts." + key,
[], Input)}
                     </div>
                  )
```

```
                })}
            </div>
        </div>
    </form >
  )
}
const ProfileDataFormReduxForm = reduxForm({ form: 'edit-profile'
})(ProfileDataForm)

export default ProfileDataFormReduxForm;

ProfileInfo.jsx
const ProfileInfo = (props) => {

  let [editMode, setEditMode] = useState(false);

  if (!props.profile) {
    return (
      <Preloader />
    )
  };
  const onMainPhotoSelected = (e) => {
    if (e.target.files.length) {
      props.savePhoto(e.target.files[0]);
    }
  };
  const onSubmit = (formData) => {
    props.saveProfile(formData).then(
      () => {
        setEditMode(false);
      }
    )
  }
  return (
    <div>
      <div className={s.profile}>
        {/* <div>
```

```
                <img src={props.profile.photos.large || userPhoto} alt="#"
/>
                {props.isOwner && <input type={"file"}
onChange={onMainPhotoSelected} className={s.uploadPhoto} />}
            </div> */}
            <div>
                <img src={props.profile.photos.large || userPhoto} alt="#"
/>
                {props.isOwner && (
                   <label htmlFor="photo-input"
className={s.uploadPhotoLabel}>
                      <span className={s.uploadButtonLabel}>Add new
photo</span>
                      <input
                         type="file"
                         id="photo-input"
                         onChange={onMainPhotoSelected}
                         className={s.uploadPhoto}
                      />
                   </label>
                )}
            </div>
            <div className={s.info}>
                {/* <div>Hi! It's my first project on React! I really hope
you like it! ♥ </div> */}
                {editMode
                   ? <ProfileDataForm initialValues={props.profile}
profile={props.profile} onSubmit={onSubmit} />
                   : <ProfileData goToEditMode={() => { setEditMode(true)
}} profile={props.profile} isOwner={props.isOwner} />}
                <ProfileStatusWithHooks status={props.status}
updateStatus={props.updateStatus} />
            </div>
         </div>
      </div>
   )
}
export default ProfileInfo;
```

```
Profile.jsx
const Profile = (props) => {
   return (
      <div className={p.content}>
         <ProfileInfo
            savePhoto={props.savePhoto}
            isOwner={props.isOwner}
            profile={props.profile}
            status={props.status}
            saveProfile={props.saveProfile}
            updateStatus={props.updateStatus} />
         <MyPostsContainer />
      </div>
   )
}
export default Profile;
ChatPage.tsx
const ChatPage: React.FC = () => {
   return (
      <div><Chat /></div>
   )
}
const Chat: React.FC = () => {
   const dispatch = useDispatch()
   const status = useSelector((state: any) => state.chat.status)
   useEffect(() => {
      dispatch(startMessagesListening())
      return () => {
         dispatch(stopMessagesListening())
      }
   }, [dispatch]) // не впевнений на рахунок цього діспатч
   return (
      <div className={s.chatItems}>
         {status === 'error' && <div>Some Error occured. Please refresh
the page </div>}
         <div className={s.chat}>
            <Messages />
```

```
            <AddMessageForm />
         </div>

      </div>
   )
}
const Messages: React.FC = () => {
   const messages = useSelector((state: any) => state.chat.messages)
   const messagesAnchorRef = useRef<HTMLDivElement>(null);
   const [isAutoIsScroll, setIsAutoScroll] = useState(true)
   const scrollHandler = (e: React.UIEvent<HTMLDivElement>) => {
      const element = e.currentTarget;
      if (Math.abs((element.scrollHeight - element.scrollTop) -
element.clientHeight) < 300) {
         !isAutoIsScroll && setIsAutoScroll(true)
      } else {
         !isAutoIsScroll && setIsAutoScroll(false)
      }
   }
   useEffect(() => {
      if (isAutoIsScroll) {
         messagesAnchorRef.current?.scrollIntoView({ behavior: 'smooth' })
      }
   }, [messages,isAutoIsScroll])

   return (
      <div className={s.messagesitems} style={{ height: '500px',
overflowY: 'auto' }} onScroll={scrollHandler}>
         {messages.map((m, index) => <Message key={m.id} message={m} />)}
         <div ref={messagesAnchorRef}></div>
      </div>
   )
}
const Message: React.FC<{ message: ChatMessageType }> = React.memo(({
message }) => {
   return (
      <div className={s.messageWrapper}>
         <div className={s.userInfo}>
```

```
            <img alt="#" src={message.photo} className={s.imgLogo} />
            <div className={s.messUser}>{message.userName}</div>
         </div>
         <div className={s.messageText}>
            {message.message}
         </div>
         <hr />
      </div >
   )
})

const AddMessageForm: React.FC = () => {
   const [message, setMessage] = useState('')
   // const [readyStatus, setReadyStatus] = useState<'pending' |
'ready'>('pending')
   const dispatch = useDispatch()
   const status = useSelector((state: any) => state.chat.status)

   const sendMessageHandler = () => {
      if (!message) {
         return;
      }
      dispatch(sendMessage(message))
      setMessage('')
   }

   return (
      <div className={s.addmessageForm}>
         <div>
            <textarea className={s.sendmes} onChange={(e) =>
setMessage(e.currentTarget.value)} value={message}></textarea>
         </div>
         <div>
            <button className={s.sendbtn} disabled={status !== 'ready'}
onClick={sendMessageHandler}>Send</button>
         </div>
      </div>
   )
```

```
}

export default ChatPage;

Friends.jsx
const Friends = (props) => {
   const [users, setUsers] = useState([]);
   const [searchQuery, setSearchQuery] = useState('');
   useEffect(() => {
      const fetchData = async () => {
         try {
            const response = await axios.get(
               'https://social-network.samuraijs.com/api/1.0/users',
               {
                  withCredentials: true,
                  headers: {
                     'API-KEY': 'bfd52358-f556-49fe-b856-3044468355c0',
                  },
                  params: {
                     page: 1,
                     count: 100,
                     followed: true,
                  },
               }
            );
            setUsers(response.data.items.filter((item) => item.followed
=== true));
         } catch (error) {
            console.error('Error:', error);
         }
      };

      fetchData();
   }, []);
   const handleFollow = (userId) => {
      props.follow(userId);
   };
   const handleUnfollow = (userId) => {
```

```
      props.unfollow(userId);
      setUsers(users.filter((user) => user.id !== userId));
   };
   const handleSearch = (event) => {
      const searchValue = event.target.value.toLowerCase();
      setSearchQuery(searchValue);
   };
   const filteredUsers = users.filter((user) =>
      user.name.toLowerCase().startsWith(searchQuery)
   );
   return (
      <div className={s.wrapper}>
         <div className={s.search}>
            <input
               type="text"
               placeholder="Search friends..."
               value={searchQuery}
               onChange={handleSearch}
            />
         </div>
         <div className={s.totalfrnds}>Total Friends:
{filteredUsers.length}</div>
         {filteredUsers.map((user) => (
            <div className={s.content} key={user.id}>
               <div className={s.photobutton}>
                  <div className={s.photoinfo}>
                     <NavLink to={'/profile/' + user.id}>
                        <img
                           src={user.photos.small != null ?
user.photos.small : userPhoto}
                           className={styles.userPhoto}
                           alt=""
                        />
                     </NavLink>
                  </div>
                  <div className={styles.buttoncenter}>
                     {user.followed ? (
```

```
                        <button onClick={() =>
handleUnfollow(user.id)}>Unfollow</button>
                     ) : (
                        <button onClick={() =>
handleFollow(user.id)}>Follow</button>
                     )}
                  </div>
               </div>
               <div className={s.info}>
                  <p>Name: {user.name}</p>
                  <p>ID: {user.id}</p>
                  <p>Followed: {user.followed.toString()}</p>
                  <p>Status: {user.status}</p>
               </div>
            </div>
         ))}
      </div>
   );
};
const mapStateToProps = (state) => ({
   users: state.users.users,
   currentPage: state.users.currentPage,
   pageSize: state.users.pageSize,
});
export default connect(mapStateToProps, { follow, unfollow, requestUsers
})(Friends);

User.jsx
let User = (props) => {
   let u = props.user;
   return (
      <div>
         <div className={styles.wflex}>
            <div className={styles.w1}>
               <div>
                  <NavLink to={'/profile/' + u.id}>
                     <img src={u.photos.small != null ? u.photos.small :
userPhoto} className={styles.userPhoto} alt="" />
```

```
                </NavLink>
            </div>
            <div className={styles.buttoncenter}>
                {u.followed
                    ? <button disabled={props.followingInProgress.some(id
=> id === u.id)}
                        onClick={() => { props.unfollow(u.id)
}}>Unfollow</button>

                    : <button disabled={props.followingInProgress.some(id
=> id === u.id)}
                        onClick={() => { props.follow(u.id)
}}>Follow</button>}
            </div>
        </div>
        <div className={styles.w2}>
            <div>{u.name}</div>
            <div>{u.status}</div>
        </div>
        <div className={styles.w3}>
            <div>{"u.location.country,"}</div>
            <div>{"u.location.city"}</div>
        </div>
    </div>
</div>
)
export default User;

Users.tsx
let Users: React.FC<PropsType> = (props) => {
   return <div className={styles.wrapper}>
      <div className={styles.paginator}>
         <Paginator currentPage={props.currentPage}
onPageChanged={props.onPageChanged}
            totalItemsCount={props.totalUsersCount}
pageSize={props.pageSize} />
      </div>
      <div>
```

```
         {
            props.users.map(u =>
               <User key={u.id} user={u}
                  followingInProgress={props.followingInProgress}
                  follow={props.follow}
                  unfollow={props.unfollow} />)
         }
      </div>
   </div >
}
export default Users;
```

NewsList.js

```
const NewsList = () => {
   const [articles, setArticles] = useState([])
   useEffect(() => {
      const getArticles = async () => {
         const response = await
axios.get(`https://newsapi.org/v2/everything?q=all&apiKey=04eab81cc1474689
9bd31f7ea301319f`)
         setArticles(response.data.articles)
         // console.log(response)
      }
      getArticles()
   }, [])
   return (
      <div>
         {articles.map(article => {
            return (
               <NewsItem
                  title={article.title}
                  description={article.description}
                  url={article.url}
                  urlToImage={article.urlToImage}
               />
            )
         })}
      </div>
```

```
   )
}
export default NewsList

NewsItem.js
const NewsItem = ({ title, description, url, urlToImage }) => {
   return (
      <div className={styles.news_app}>
         <div className={styles.news_item}>
            <img className={styles.news_img} src={urlToImage}
alt={urlToImage} />
            <h3><a href={url}>{title}</a></h3>
            <p>{description}</p>
         </div>
      </div>
   )
}
export default NewsItem

Login.tsx
const LoginForm: React.FC<InjectedFormProps<LoginFormValuesType,
LoginFormOwnProps> & LoginFormOwnProps> = (props) => {
   let maxLength30 = maxLengthCreator(30);
   return (
      <div className={s.log_container}>
         <form onSubmit={props.handleSubmit}>
            {createField<LoginFormValuesTypeKeys>("Email", "email",
[required, maxLength30], Input)}
            {createField<LoginFormValuesTypeKeys>("Passwosd", "password",
[required, maxLength30], Input, { type: "password" })}
            <div className={s.remme}>
               {createField<LoginFormValuesTypeKeys>(undefined,
"rememberMe", [], Input, { type: "checkbox" }, "")} <b>Remember me</b>
            </div>
            {/*   <div><Field placeholder="Email" name={"email"}
component={Input} validate={[required, maxLength30]} /></div>
```

```
         <div><Field placeholder="Passwosd" name={"password"}
component={Input} validate={[required, maxLength30]} type={"password"}
/></div>
         <div className={s.remme}><Field type={"Checkbox"}
name={"rememberMe"} component={Input} />Remember me</div> */}

            {props.captchaUrl && <img alt="4" src={props.captchaUrl} />}
            {props.captchaUrl &&
createField<LoginFormValuesTypeKeys>("Symbols from captcha", "captcha",
[required], Input, {})}
            {props.error &&
               <div className={style.formSummaryError}>
                  {props.error}
               </div>
            }
            <div className={s.btns}>
               <div className={s.btn}><button>Login</button></div>
               <div className={s.btn}><button><NavLink to='/registration'
className={s.btnReg}>Registration</NavLink></button></div>
            </div>
         </form>
         <div className={s.log}>
            <p>
               Email: sashachebyrasha16@gmail.com
               <br />
               Password: sasha1604
            </p>
         </div>

      </div>
   )
}
const LoginReduxForm = reduxForm<LoginFormValuesType, LoginFormOwnProps>({
form: 'login' })(LoginForm)
type MapStatePropsType = {
   captchaUrl: string | null
   isAuth: boolean
}
```

```
type MapDispatchPropsType = {
   login: (email: string, password: string, rememberMe: boolean, captcha:
string) => void
}
export type LoginFormValuesType = {
   email: string
   password: string
   rememberMe: boolean
   captcha: string
}
type LoginFormValuesTypeKeys = Extract<keyof LoginFormValuesType, string>
const Login: React.FC<MapStatePropsType & MapDispatchPropsType> = (props)
=> {
   const onSubmit = (formData: LoginFormValuesType) => {
      props.login(formData.email, formData.password, formData.rememberMe,
formData.captcha)
      // console.log(props.login(formData.email, formData.password,
formData.rememberMe))
   }
   let navigate = useNavigate();
   if (props.isAuth) {
      return (
         // redirect("/profile")
         navigate("/profile")
         // <Link to={"/profile"} />
      )
   }
   return (
      <div className={s.log}>
         <h1>Please Login</h1>
         <LoginReduxForm onSubmit={onSubmit} captchaUrl={props.captchaUrl}
/>
      </div >
   )
}
const mapStateToProps = (state: AppStateType): MapStatePropsType => ({
   isAuth: state.auth.isAuth,
   captchaUrl: state.auth.captchaUrl
```

```
})
export default connect(mapStateToProps, { login })(Login);

Registratin.jsx
const Registration = () => {
   const [name, setName] = useState("");
   const [email, setEmail] = useState("");
   const [password, setPassword] = useState("");
   const [confirmPassword, setConfirmPassword] = useState("");
   const [acceptTerms, setAcceptTerms] = useState(false);
   const handleNameChange = (e) => {
      setName(e.target.value);
   };
   const handleEmailChange = (e) => {
      setEmail(e.target.value);
   };
   const handlePasswordChange = (e) => {
      setPassword(e.target.value);
   };
   const handleConfirmPasswordChange = (e) => {
      setConfirmPassword(e.target.value);
   };
   const handleAcceptTermsChange = (e) => {
      setAcceptTerms(e.target.checked);
   };
   const handleSubmit = (e) => {
      e.preventDefault();
      setName("");
      setEmail("");
      setPassword("");
      setConfirmPassword("");
      setAcceptTerms(false);
   };
   return (
      <div className={s.wrapper}>
         <h2>Registration</h2>
         <form onSubmit={handleSubmit}>
            <div className={s.input_box}>
```

```
                <input/>
            </div>
            <div className={s.input_box}>
                <input/>
            </div>
            <div className={s.input_box}>
                <input/>
            </div>
            <div className={s.input_box}>
                <input/>
            </div>
            <div className={s.policy}>
                <input/>
                <h3>I accept all terms & condition</h3>
            </div>
            <div className={s.inputbutton}>
                <button type="submit" disabled={!acceptTerms}>
                    Register now
                </button>
            </div>
            <div className={s.text}>
                <h3>Already have an account? </h3>
                <div className={s.block}>
                    <NavLink to='/login' className={s.btnReg}>Login
now</NavLink>
                </div>
            </div>
        </form>
    </div>
  );
}
export default Registration;
```

app-reducer.ts

```
const INITIALIZED_SUCCESS = 'INITIALIZED_SUCCESS';
export type InitialStateType = {
```

```
   initialized: boolean
}
let initialState: InitialStateType = {
   initialized: false
};
const appReduser = (state = initialState, action: any): InitialStateType
=> {
   switch (action.type) {
      case INITIALIZED_SUCCESS:
         return {
            ...state,
            initialized: true
         }
      default:
         return state;
   }
}
type InitializedSuccessActionType = {
   type: typeof INITIALIZED_SUCCESS
}
export const initializedSuccess = (): InitializedSuccessActionType => ({
type: INITIALIZED_SUCCESS });
export const initializeApp = () => (dispatch: any) => {
   let promise = dispatch(getAuthUserData());
   Promise.all([promise]).then(() => {
      dispatch(initializedSuccess())
   });
}
export default appReduser;
```

auth-reducer.ts

```
const authReduser = (state = initialState, action: any): InitialStateType
=> {
   switch (action.type) {
      case SET_USER_DATA:
      case GET_CAPTCHA_URL_SUCCESS:
         return {
            ...state,
```

```
            ...action.payload,
         }
      default:
         return state;
   }
}
type GetCaptchaUrlSuccessActionType = {
   type: typeof GET_CAPTCHA_URL_SUCCESS
   payload: { captchaUrl: string }
}
export const getCaptchaUrlSuccess = (captchaUrl: string):
GetCaptchaUrlSuccessActionType => ({
   type: GET_CAPTCHA_URL_SUCCESS,
   payload: { captchaUrl }
});
type SetAuthUserDataActionPayloadType = {
   userId: number | null
   email: string | null
   login: string | null
   isAuth: boolean
}
type SetAuthUserDataActionType = {
   type: typeof SET_USER_DATA
   payload: SetAuthUserDataActionPayloadType
}
export const setAuthUserData = (userId: number | null, email: string |
null, login: string | null, isAuth: boolean): SetAuthUserDataActionType =>
({
   type: SET_USER_DATA,
   payload: { userId, email, login, isAuth }
});
export const getAuthUserData = () => async (dispatch: any) => {
   let meData = await authAPI.me()

   if (meData.resultCode === ResultCodesEnum.Success) {
      let { id, login, email } = meData.data;
      dispatch(setAuthUserData(id, email, login, true))
   }
```

```
}
export const login = (email: string, password: string, rememberMe:
boolean, captcha: any) => async (dispatch: any) => { //captcha:string
   let loginData = await authAPI.login(email, password, rememberMe,
captcha);
   if (loginData.resultCode === ResultCodesEnum.Success) {
      dispatch(getAuthUserData())
   } else {
      if (loginData.resultCode === ResultCodeForCaptcha.CaptchaIsRequired)
{
         dispatch(getCaptchaUrl());
      }
      let message = loginData.messages.length > 0 ? loginData.messages[0]
: "Some error";
      dispatch(stopSubmit("login", { _error: message }));
   }
}
export const getCaptchaUrl = () => async (dispatch: any) => {
   let response = await securityAPI.getCaptchaUrl();
   const captchaUrl = response.data.url;

   dispatch(getCaptchaUrlSuccess(captchaUrl));
}
export const logout = () => async (dispatch: any) => {
   let response = await authAPI.logOut();

   if (response.data.resultCode === 0) {
      dispatch(setAuthUserData(null, null, null, false))
   }
}
export default authReduser;

ChatReducer.ts
const chatReducer = (state = initialState, action: ActionsType) => {
   switch (action.type) {
      case 'SN/chat/MESSAGES_RECEIVED':
         return {
            ...state,
```

```
            messages: [...state.messages, ...action.payload.messages.map(m
=> ({ ...m, id: uuidv4() }))]
                .filter((m, index, array) => index >= array.length - 100)
         };
      case 'SN/chat/STATUS_CHANGED':
         return {
            ...state,
            status: action.payload.status
         };
      default:
         return state;
   }
};

export const actions = {
   messagesReceived: (messages: ChatMessageType[]) => ({
      type: 'SN/chat/MESSAGES_RECEIVED',
      payload: { messages }
   } as const),
   statusChanged: (status: StatusType) => ({
      type: 'SN/chat/STATUS_CHANGED',
      payload: { status }
   } as const)
};
let _newMessageHandler: ((messages: ChatMessageType[]) => void) | null =
null;
const newMessageHandlerCreator = (dispatch: Dispatch) => {
   if (_newMessageHandler === null) {
      _newMessageHandler = (messages) => {
         dispatch(actions.messagesReceived(messages));
      };
   }
   return _newMessageHandler;
};
let _statusChangedHandler: ((status: StatusType) => void) | null = null;
const statusChangedHandlerCreator = (dispatch: Dispatch) => {
   if (_statusChangedHandler === null) {
      _statusChangedHandler = (status) => {
```

```
            dispatch(actions.statusChanged(status));
        };
    }
    return _statusChangedHandler;
};
export const startMessagesListening = (): ThunkType => async (dispatch) =>
{
    chatApi.start();
    chatApi.subscribe('messages-received',
newMessageHandlerCreator(dispatch));
    chatApi.subscribe('status-changed',
statusChangedHandlerCreator(dispatch));
};
export const stopMessagesListening = (): ThunkType => async (dispatch) =>
{
    chatApi.unsubscribe('messages-received',
newMessageHandlerCreator(dispatch));
    chatApi.unsubscribe('status-changed',
statusChangedHandlerCreator(dispatch));
    chatApi.stop();
};
export const sendMessage = (message: string): ThunkType => async
(dispatch) => {
    chatApi.sendMessage(message);
};
export default chatReducer;
```

profile-reducer.ts

```
const ADD_POST = 'ADD-POST';
const LIKE_POST = 'LIKE_POST';
const DISLIKE_POST = 'DISLIKE_POST';
const DELETE_POST = 'DELETE_POST';
const SET_USER_PROFILE = 'SET_USER_PROFILE';
const SET_STATUS = 'SET_STATUS';
const SAVE_PHOTO_SUCCESS = 'SAVE_PHOTO_SUCCESS';
let initialState = {
    postsData: [
        { id: 1, message: "Hi, it is my first progect!", likes: 7 },
```

```
      { id: 2, message: "I hope you will like it!", likes: 16 },
      { id: 3, message: "Good luck!", likes: 23 },
   ] as Array<PostType>,
   profile: null as ProfileType | null,
   status: "",
   newPostText: ''
};

export type InitialStateType = typeof initialState;
const profileReducer = (state = initialState, action: any):
InitialStateType => {
   switch (action.type) {
      case ADD_POST:
         return {
            ...state,
            postsData: [
               ...state.postsData,
               {
                  id: state.postsData.length + 1,
                  message: action.newPostText,
                  likes: 7
               }
            ]
         };
      case SET_USER_PROFILE: {
         return {
            ...state,
            profile: action.profile
         };
      }
      case SET_STATUS: {
         return {
            ...state,
            status: action.status
         };
      }
      case LIKE_POST:
         return {
```

```
            ...state,
            postsData: state.postsData.map((post) =>
               post.id === action.postId ? { ...post, likes: post.likes +
1 } : post
            )
         };
      case DISLIKE_POST:
         return {
            ...state,
            postsData: state.postsData.map((post) =>
               post.id === action.postId ? { ...post, likes: post.likes -
1 } : post
            )
         };
      case DELETE_POST:
         return {
            ...state,
            postsData: state.postsData.filter((post) => post.id !==
action.postId)
         };
      case SAVE_PHOTO_SUCCESS: {
         return {
            ...state,
            profile: { ...state.profile, photos: action.photos } as
ProfileType
         };
      }
      default:
         return state;
   }
}
export const addPostActionCreator = (newPostText: string):
AddPostActionCreatorActionType => ({ type: ADD_POST, newPostText });
export const setUserProfile = (profile: ProfileType):
SetUserProfileActionType => ({ type: SET_USER_PROFILE, profile });
export const setStatus = (status: string): SetStatusActionType => ({ type:
SET_STATUS, status });
```

```
export const deletePost = (postId: number): DeletePostActionType => ({
type: DELETE_POST, postId });
export const savePhotoSuccess = (photos: PhotosType):
SavePhotoSuccessActionType => ({ type: SAVE_PHOTO_SUCCESS, photos });
export const likePostActionCreator = (postId) => ({
   type: LIKE_POST,
   postId
});
export const dislikePostActionCreator = (postId) => ({
   type: DISLIKE_POST,
   postId
});
export const deletePostActionCreator = (postId) => ({
   type: DELETE_POST,
   postId
});
export const getUserProfile = (userId: number) => async (dispatch: any) =>
{
   let response = await usersAPI.getProfile(userId);

   dispatch(setUserProfile(response.data));
};
export const getStatus = (userId: number) => async (dispatch: any) => {
   let response = await profileAPI.getStatus(userId);

   dispatch(setStatus(response.data));
};
export const updateStatus = (status: string) => async (dispatch: any) => {
   let response = await profileAPI.updateStatus(status);
   if (response.data.resultCode === 0) {
      dispatch(setStatus(status));
   }
};
export const savePhoto = (file: any) => async (dispatch: any) => {
   let response = await profileAPI.savePhoto(file);
   if (response.data.resultCode === 0) {
      dispatch(savePhotoSuccess(response.data.data.photos));
   }
```

```
};
export const saveProfile = (profile: ProfileType) => async (dispatch: any,
getState: any) => {
   const userId = getState().auth.userId;
   const response = await profileAPI.saveProfile(profile);
   if (response.data.resultCode === 0) {
      dispatch(getUserProfile(userId));
   } else {
      dispatch(stopSubmit("edit-profile", { _error:
response.data.messages[0] }));
      return Promise.reject(response.data.messages[0]);
   }
};
export default profileReducer;

redux-store.ts
let rootReduser = combineReducers({
   profilePage: profileReducer,
   dialogsPage: dialogsReducer,
   usersPage: usersReducer,
   auth: authReduser,
   form: formReducer,
   app: appReduser,
   users: usersReducer,
   chat: chatReducer
});

type RootReduserType = typeof rootReduser; // (globalstate: AppStateType)
=> AppStateType
export type AppStateType = ReturnType<RootReduserType>;
type PropertiesTypes<T> = T extends { [key: string]: infer U } ? U : never
export type InferActionsTypes<T extends { [key: string]: (...args: any[])
=> any }> = ReturnType<PropertiesTypes<T>>
const composeEnhancers = window.__REDUX_DEVTOOLS_EXTENSION_COMPOSE__ ||
compose;
const store = createStore(rootReduser,
composeEnhancers(applyMiddleware(thunkMiddleware)));
window.__store__ = store;
```

```
export default store;

users-reducer.ts
let initialState = {
   users: [] as Array<UserType>,
   pageSize: 10,
   totalUsersCount: 0,
   currentPage: 1,
   isFetching: false,
   followingInProgress: [] as Array<number> // Array of users id
};
type InitialStateType = typeof initialState;
const usersReducer = (state = initialState, action: ActionsTypes):
InitialStateType => {
   switch (action.type) {
      case 'FOLLOW':
         return {
            ...state,
            users: updateObjectInArray(state.users, action.userId, "id", {
followed: true })
         }
      case 'UNFOLLOW':
         return {
            ...state,
            users: updateObjectInArray(state.users, action.userId, "id", {
followed: false })
         }
      case 'SET_USERS': {
         return { ...state, users: action.users }
      }
      case 'SET_CARRENT_PAGE': {
         return { ...state, currentPage: action.currentPage }
      }
      case 'SET_TOTAL_USERS_COUNT': {
         return { ...state, totalUsersCount: action.count }
      }
      case 'TOGGLE_IS_FETCHING': {
         return { ...state, isFetching: action.isFetching }
```

```
      }
      case 'TOGGLE_IS_FOLLOWING_PROGRESS': {
         return {
            ...state, followingInProgress: action.isFetching
               ? [...state.followingInProgress, action.userId]
               : state.followingInProgress.filter(id => id !==
action.userId)
         }
      }
      default:
         return state;
   }
};
type ActionsTypes = InferActionsTypes<typeof actions>
export const actions = {
   followSuccess: (userId: number) => ({ type: 'FOLLOW', userId } as
const),
   unfollowSuccess: (userId: number) => ({ type: 'UNFOLLOW', userId } as
const),
   setUsers: (users: Array<UserType>) => ({ type: 'SET_USERS', users } as
const),

   setCurrentPage: (currentPage: number) => ({ type: 'SET_CARRENT_PAGE',
currentPage } as const),
   setTotalUsersCount: (totalUsersCount: number) => ({ type:
'SET_TOTAL_USERS_COUNT', count: totalUsersCount } as const),
   toggleIsFetching: (isFetching: boolean) => ({ type:
'TOGGLE_IS_FETCHING', isFetching } as const),
   toggleFollowingProgress: (isFetching: boolean, userId: number) => ({
type: 'TOGGLE_IS_FOLLOWING_PROGRESS', isFetching, userId } as const)
}
type DispatchType = () => AppStateType;
type ThunkType = ThunkAction<Promise<void>, AppStateType, unknown,
ActionsTypes>;
export const requestUsers = (currentPage: number, pageSize: number):
ThunkType => {
   return async (dispatch, getState) => {
      dispatch(actions.toggleIsFetching(true));
```

```
        dispatch(actions.setCurrentPage(currentPage))
        let data = await usersAPI.getUsers(currentPage, pageSize);
        dispatch(actions.toggleIsFetching(false));
        dispatch(actions.setUsers(data.items));
        dispatch(actions.setTotalUsersCount(data.totalCount));
    }
}
const _followUnfollowFlow = async (dispatch: DispatchType, userId: number,
apimethod: any, ActionCreator: (userId: number) => ActionsTypes) => {
    dispatch(actions.toggleFollowingProgress(true, userId));
    let response = await apimethod(userId);
    if (response.data.resultCode === 0) {
        dispatch(ActionCreator(userId));
    }
    dispatch(actions.toggleFollowingProgress(false, userId));
}
export const follow = (userId: number): ThunkType => {
    return async (dispatch) => {
        _followUnfollowFlow(dispatch, userId,
usersAPI.follow.bind(usersAPI), actions.followSuccess);
    }
}
export const unfollow = (userId: number): ThunkType => {
    return async (dispatch) => {
        _followUnfollowFlow(dispatch, userId,
usersAPI.unfollow.bind(usersAPI), actions.unfollowSuccess);
    }
}
export default usersReducer;

api.ts
const instance = axios.create({
    withCredentials: true,
    baseURL: 'https://social-network.samuraijs.com/api/1.0/',
    headers: {
        "API-KEY": "bfd52358-f556-49fe-b856-3044468355c0"
    }
});
```

```
export const usersAPI = {
   getUsers(currentPage = 1, pageSize = 7) {
      return (
         instance.get(`users?page=${currentPage}&count=${pageSize}`,
         ).then(response => { return response.data })
      )
   },
   follow(userId: number) {
      return (
         instance.post(`follow/${userId}`)
      )},
   unfollow(userId: number) {
      return (
         instance.delete(`follow/${userId}`)
      ) },

   getProfile(userId: number) {
      console.warn('Obsolete method. Please use profileAPI object!');
      return (
         profileAPI.getProfile(userId)
      ) }
}
export const profileAPI = {
   getProfile(userId: number) {
      return (
         instance.get(`profile/` + userId)
      )  },
   getStatus(userId: number) {
      return (
         instance.get(`profile/status/` + userId)
      ) },
   updateStatus(status: string) {
      return (
         instance.put(`profile/status`, { status: status })
      )
   },
   savePhoto(photoFile: any) {
      const formData = new FormData();
```

```
      formData.append("image", photoFile);
      return (
         instance.put(`profile/photo`, formData, {
            headers: {
               'Content-type': 'multipart/form-data'
            } })
      )
   },
   saveProfile(profile: ProfileType) {
      return (
         instance.put(`profile`, profile)
      ) },
}
export enum ResultCodesEnum {
   Success = 0,
   Error = 1,
}
export enum ResultCodeForCaptcha {
   CaptchaIsRequired = 10
}
type MeResponseType = {
   data: { id: number, email: string, login: string }
   resultCode: ResultCodesEnum
   messages: Array<string>
}
type LoginResponseType = {
   data: { userId: number }
   resultCode: ResultCodesEnum | ResultCodeForCaptcha
   userId: Array<string>
}
export const authAPI = {
   me() {
      return (
         instance.get<MeResponseType>(`auth/me`).then(res => res.data)
      ) },
   login(email: string, password: string, rememberMe = false, captcha:
null | string = null) {
      return (
```

```
         instance.post<LoginResponseType>(`auth/login`, { email, password,
rememberMe, captcha }).then(res => res.data)
      )  },
   logOut() {
      return (
         instance.delete(`auth/login`)
      )  },
}
export const securityAPI = {
   getCaptchaUrl() {
      return (
         instance.get(`security/get-captcha-url`)
      ) }
}

Chat-api.ts
const subscribers = {
   'messages-received': [] as SubscriberType[],
   'status-changed': [] as StatusChangedSubscriberType[]
};
let ws: WebSocket | null = null;
const closeHandler = () => {
   notifySubscribersAboutStatus('pending');
   setTimeout(createChannel, 3000);
};
const messageHandler = (e: MessageEvent) => {
   const newMessages = JSON.parse(e.data);
   subscribers['messages-received'].forEach((s) => s(newMessages));
};
const openHandler = () => {
   notifySubscribersAboutStatus('ready');
};
const errorHandler = () => {
   notifySubscribersAboutStatus('error');
   console.error('REFRESH PAGE');
};
const cleanup = () => {
   ws?.removeEventListener('close', closeHandler);
```

```
    ws?.removeEventListener('message', messageHandler);
    ws?.removeEventListener('open', openHandler);
    ws?.removeEventListener('error', errorHandler);
};
const notifySubscribersAboutStatus = (status: StatusType) => {
    subscribers['status-changed'].forEach((s) => s(status));
};
function createChannel() {
    cleanup();
    ws?.close();
    ws = new WebSocket('wss://social-
network.samuraijs.com/handlers/ChatHandler.ashx');
    notifySubscribersAboutStatus('pending');
    ws.addEventListener('close', closeHandler);
    ws.addEventListener('message', messageHandler);
    ws.addEventListener('open', openHandler);
    ws.addEventListener('error', errorHandler);
}
export const chatApi = {
    start() {
        createChannel();
    },
    stop() {
        subscribers['messages-received'] = [];
        subscribers['status-changed'] = [];
        cleanup();
        ws?.close();
    },
    subscribe(eventName: EventNamesType, callback: SubscriberType |
StatusChangedSubscriberType) {
        if (subscribers.hasOwnProperty(eventName)) {
            subscribers[eventName].push(callback);
        }
        return () => {
            subscribers[eventName] = subscribers[eventName].filter((s) => s
!== callback);
        };
    },
```

```
   unsubscribe(eventName: EventNamesType, callback: SubscriberType |
StatusChangedSubscriberType) {
      if (subscribers.hasOwnProperty(eventName)) {
         subscribers[eventName] = subscribers[eventName].filter((s) => s
!== callback);
      }
   },
   sendMessage(message: string) {
      ws?.send(message);
   }
};

ProfileStatus.test.jsx
describe("ProfileStatus component", () => {
   test("status from props should be in the state", () => {
      const component = create((<ProfileStatus status="hellow" />));
      const instance = component.getInstance();
      expect(instance.state.status).toBe("hellow");
   });
   test("after creation <span/> should be displayed", () => {
      const component = create((<ProfileStatus status="hellow" />));
      const root = component.root;
      let span = root.findByType("span");
      expect(span).not.toBeNull();
   });
   test("after creation <input/> shouldn't be displayed", () => {
      const component = create((<ProfileStatus status="hellow" />));
      const root = component.root;
      expect(() => {
         let input = root.findByType("input");
      }).toThrow();
   });
   test("after creation <span/> should contains correct status", () => {
      const component = create((<ProfileStatus status="hellow" />));
      const root = component.root;
      let span = root.findByType("span");
      expect(span.children[0]).toBe("hellow");
   });
```

```
   test("input should be displayed in EditMode instead of span", () => {
      const component = create((<ProfileStatus status="hellow" />));
      const root = component.root;
      let span = root.findByType("span");
      span.props.onDoubleClick();
      let input = root.findByType("input");
      expect(input.props.value).toBe("hellow");
   });
   test("callback should be called", () => {
      let mockCallback = jest.fn();
      const component = create((<ProfileStatus status="hellow"
updateStatus={mockCallback} />));
      const instance = component.getInstance();
      instance.deactivateEditMode();
      expect(mockCallback.mock.calls.length).toBe(1);
   });
})
```

Paginator.test.jsx

```
describe("Paginator component tests", () => {
   test("Pages count is 11 byt should be showed only 10", () => {
      const component = create(<Paginator totalItemsCount={11}
pageSize={1} portionSize={10} />);
      const root = component.root;
      let spans = root.findAllByType("span");
      expect(spans.length).toBe(10);
   });
   test("If pages count is more then 10 button NEXT should be present", ()
=> {
      const component = create(<Paginator totalItemsCount={11}
pageSize={1} portionSize={10} />);
      const root = component.root;
      let button = root.findAllByType("button");
      expect(button.length).toBe(1);
   });
})
```

Profile-reducer.test.js

```
let state = {
   postsData: [
      { id: "1", message: "Hi, it is my first progect!", likes: "7" },
      { id: "2", message: "I hope you will like it!", likes: "16" },
      { id: "3", message: "Good luck!", likes: "23" },
   ]
};
test('new post should be added, length of posts should be incremented', ()
=> {
   let action = addPostActionCreator("first project");
   let newState = profileReducer(state, action);
      expect(newState.postsData.length).toBe(4);
});
test('new post message should be correct', () => {
   let action = addPostActionCreator("first project");
   let newState = profileReducer(state, action);
   expect(newState.postsData[3].message).toBe("first project");
});
test('after deleting length of messages should be decrement', () => {
   let action = deletePost(1);
   let newState = profileReducer(state, action);
expect(newState.postsData.length).toBe(3);
});
test('after deleting length of messages should be decrement if id is
incorrect', () => {
   let action = deletePost(1000);
   let newState = profileReducer(state, action);
   expect(newState.postsData.length).toBe(3);
});
>
```