Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра комп'ютерних наук
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Розробка програмної платформи для розгортання соціальних мереж на основі стеку MERN

Виконав(ла): студент(ка) 4 курсу, групи СНс-41
спеціальності 122 Комп'ютерні науки
(шифр і назва спеціальності)

(підпис) Яцишин В.А. (прізвище та ініціали)

Керівник (підпис) Ясній О.П. (прізвище та ініціали)

Нормоконтроль (підпис) (прізвище та ініціали)

Завідувач кафедри (підпис) Боднарчук І.О. (прізвище та ініціали)

Рецензент (підпис) (прізвище та ініціали)

Тернопіль
2024

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра комп'ютерних наук
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Боднарчук І.О.
(підпис) (прізвище та ініціали)
«25» червня 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавр
(назва освітнього ступеня)

за спеціальністю 122 Комп'ютерні науки
(шифр і назва спеціальності)

студенту Яцишин Владислав Андрійович
(прізвище, ім'я, по батькові)

1. Тема роботи **Розробка програмної платформи для розгортання соціальних мереж на основі стеку MERN**

Керівник роботи **д.т.н., проф. Ясній О. П.**
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «*29*» *квітня* 2024 року № *4/7-472*

2. Термін подання студентом завершеної роботи 25 червня 2024 р.

3. Вихідні дані до роботи Літературні джерела з тематики роботи

4. Зміст роботи (перелік питань, які потрібно розробити)
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКРОЧЕНЬ І ТЕРМІНІВ; ВСТУП 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ 1.1 Поняття про соціальні мережі як засіб обміну цифровими даними 1.2 Особливості проєктування мобільних соціальних мереж 1.3 Популярні комерційні платформи СМ 1.4 Висновки до розділу 1;2 ОСОБЛИВОСТІ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ СОЦІАЛЬНИХ МЕРЕЖ 2.1 Принципи проєктування та розробки соціальних мереж 2.2 Програмні архітектури соціальних мереж 2.3 Висновки до розділу 2; 3 РОЗРОБКА ПРОГРАМНОЇ ПЛАТФОРМИ ДЛЯ СОЦІАЛЬНОЇ МЕРЕЖІ 3.1 Основні технології front-end розробки 3.2 Аналіз стеку технологій MERN для побудови платформи для соціальних мереж 3.3 Установка компонентів стеку MERN 4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ 4.1 Охорона праці та її актуальність в ІТ-сфері 4.2 Шкідлива дія шуту та вібрації і захист від неї; ВИСНОВКИ; СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ; ДОДАТКИ

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
1.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці	Сенчишин В. С., доцент кафедри МТ		

7. Дата видачі завдання 29 січня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	30.01.2024	*Виконано*
2.	Підбір джерел про плагіни та шаблони WordPress	31.01.2024-03.02.2024	*Виконано*
3.	Опрацювання джерел по темі кваліфікаційної роботи	04.02.2024-06.02.2024	*Виконано*
4.	Виконання дослідження щодо шаблонів та плагінів WordPress	07.02.2024-11.02.2024	*Виконано*
5.	Розроблення сайту на WordPress	21.05.2024-02.06.2024	*Виконано*
6.	Оформлення розділу «Аналіз предметної області та постановка завдання»	03.06.2024-05.06.2024	*Виконано*
7.	Оформлення розділу «Проєктна частина»	06.06.2024-08.06.2024	*Виконано*
8.	Оформлення розділу «Практична частина»	09.06.2024-11.06.2024	*Виконано*
9.	Виконання завдання до підрозділу «Безпека життєдіяльності»	12.06.2024-14.06.2024	*Виконано*
10.	Виконання завдання до підрозділу «Основи охорони праці»	14.06.2024-16.06.2024	*Виконано*
11.	Оформлення кваліфікаційної роботи	16.06.2024-17.06.2024	*Виконано*
12.	Нормоконтроль	18.06.2024-19.06.2024	*Виконано*
13.	Перевірка на плагіат	20.06.2024	*Виконано*
14.	Попередній захист кваліфікаційної роботи	21.06.2024	*Виконано*
15.	Захист кваліфікаційної роботи	26.06.2024	

Студент _____ Яцишин В. А.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Ясній О. П.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

"Розробка веб-платформи для соціальної мережі з використанням технологій MERN." // Кваліфікаційна робота освітнього рівня «Бакалавр» // Яцишин Владислав Андрійович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра комп’ютерних наук, група СНс-41 // Тернопіль, 2024 // с. – 72 , рис. – 16, табл. – 3 , джерел – 26, додатків – 5, сторінок додатків – 38.

Ключові слова: веб-розробка, динамічний сайт, стек технологій розробки, програмна архітектура

Для створення соціальної мережі кінцевий програмний продукт має відповідати властивостям гнучкості, ефективності та можливості масштабування. Інструменти для цього повинні забезпечити ефективне зберігання даних користувачів та контенту, обробку запитів сервера, а також швидке та інтерактивне оновлення інтерфейсу користувача.

В роботі проаналізовано основні підходи, інструменти, проблеми побудови соціальних мереж, як динамічних інформаційних систем, що мають задовольняти потреби їх користувачів в обміні інформацією в різних спільнотах та групах.

Центральним елементом розробки, як і будь-якої іншої інформаційної системи є база даних, яка забезпечить зберігання різноманітної інформації, від профілів користувачів до повідомлень, зображень та багато іншого. Потрібна гнучкість схеми даних для забезпечення масштабованості та легкості управління інформацією.

На стороні сервера знадобиться засіб обробки запитів, маршрутизації та обробки даних, які надходять від користувачів. Це важливий аспект управління даними та захисту від зовнішніх запитів.

Щоб забезпечити інтерактивний та привабливий інтерфейс для користувачів, потрібен інструмент, який дозволить легко оновлювати та відображати інформацію на сторінках. Це означає використання швидких та реактивних компонент, які забезпечать зручність користування.

І, нарешті, знадобиться середовище виконання, яке дозволить запускати серверний код та забезпечить інтеракцію між базою даних, сервером та користувацьким інтерфейсом.

Ці засоби у поєднанні надають можливість створити соціальну мережу з високим функціоналом, забезпечуючи зручність користування та ефективне управління даними. Всі вони наявні у стекові технологій MERN, який буде використано в цій роботі для розробки платформи для соціально мережі.

ANNOTATION

"Development of a Software Platform Based on the MERN Stack for the Deployment of Social Networks" // Qualification work of the educational level "Bachelor" // Yatsyshyn Vladyslav Andriiovytch// Ternopil Ivan Puluj National Technical University, Faculty of Computer Information Systems and Software Engineering, Department of Computer Science, Group CHc-41 // Ternopil, 2024 // p. – 72, fig. – 16, tables – 3, references – 26, annexes – 5, pages for annexes – 38.

Keywords: web development, dynamic site, development technology stack, software architecture

When it comes to building a social network, you seek flexibility, efficiency, and scalability. Tools for this purpose should ensure effective storage of user data and content, server-side request handling, as well as quick and responsive user interface updates.

The work analyzes the main approaches, tools, and problems of building social networks as dynamic information systems that should satisfy the needs of their users to share information in various communities and groups.

The central element of the development, like any other information system, is the database, which will ensure the storage of various information, from user profiles to messages, images and much more. Data schema flexibility is required to ensure scalability and ease of information management.

On the server-side, you'll need a means to handle requests, route them, and manage data coming from users. This is a crucial aspect of data management and security against external queries.

To provide an interactive and appealing interface for users, you'll need a tool that allows easy updates and displays information on pages swiftly. This means having fast and reactive components that ensure user convenience.

Finally, you'll require a runtime environment that executes server-side code and facilitates interaction between the database, server, and user interface.

These tools, when combined, enable the creation of a social network with extensive functionality, ensuring user-friendliness and efficient data management. All of them are available in the MERN technology stack, which will be used in this work to develop a platform for a social network.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКРОЧЕНЬ І ТЕРМІНІВ

API – Application Program Interface

CAC – Customer Acquisition Cost

CLI – Command Line Interface

CMS – Content Management System

CND – Content Delivery Network

CSS – Cascading Style Sheets

DOM – Document Object Model

GPS – Global Positioning System

HTML – Hyper Text Markup Lenguage

HTTP – Hyper Text Transmission Protocol

JS – JavaScript

JSON – JavaScript Object Notation

MERN – MongoDB, Express, React, Node

MQL – MongoDB Query Language

MVP – Minimum Viable Product

PWA – Progressive Web App

SMS – Short Message System

SOA – Service Oriented Architecture

SPA – Single Page Application

БД – база даних

ОС – операційна система

СКБД/СУБД – система керування/управління базами даних

СМ – соціальна мережа

ВСТУП

Розробка платформ для соціальних мереж є надзвичайно актуальною у сучасному цифровому світі. Технологічний прогрес та зростання використання Інтернету спричинили підвищений інтерес до соціальних мереж як засобу спілкування, обміну інформацією, розваги та реклами. З огляду на це, розробка нових платформ для соціальних мереж і постійне удосконалення існуючих є важливим завданням в галузі інформаційних технологій [1].

Однією з ключових причин актуальності розробки платформ для соціальних мереж є постійна зміна та розвиток потреб користувачів. Люди очікують від соціальних мереж не лише можливості спілкування, а й інструменти для вирішення різноманітних завдань, таких як робота, освіта, реклама, покупки тощо [2]. Розробники платформ повинні враховувати ці потреби та постійно вдосконалювати функціонал, щоб задовольнити очікування користувачів.

Додатково, зростання кількості користувачів соціальних мереж вимагає створення платформ, які можуть ефективно масштабуватись та забезпечувати стабільну роботу при великому обсязі даних та великій кількості користувачів одночасно [3], [4].

Вказані та інші джерела можуть надати глибоке розуміння тенденцій розвитку соціальних мереж та важливі аспекти їх платформенної реалізації.

Програмна архітектура для соціальних мереж відіграє важливу роль у створенні та функціонуванні цих платформ. Ця архітектура визначає структуру, компоненти та взаємодії програмного забезпечення, які забезпечують роботу соціальної мережі, забезпечуючи швидкий доступ, безпеку та зручність для користувачів.

Однією з типових архітектур для соціальних мереж є клієнт-серверна архітектура. Клієнти (приклади: веб-браузери або мобільні додатки) взаємодіють із серверами, де зберігається та обробляється інформація. Це забезпечує постійний доступ до даних із будь-якого пристрою.

Крім того, масштабованість є ключовою властивістю архітектури соціальних мереж. Такі платформи мають велику кількість користувачів та оброблюють величезний обсяг даних. Тому важливою є розробка архітектури, яка може масштабуватись, щоб витримувати великі навантаження.

Наприклад, Facebook використовує складну архітектуру, орієнтовану на послуги. Вони використовують мікросервіси, що дозволяє розділяти функції на окремі сервіси для підтримки швидкої розробки та масштабування системи [5]. Крім того, вони використовують технології, такі як Cassandra для зберігання даних та Apache Hive для аналізу великих обсягів інформації.

Іншим прикладом є Twitter, який базується на архітектурі, що складається з тисяч мікросервісів, де кожен сервіс відповідає за певну функціональність [6].

Архітектура соціальних мереж має бути не лише масштабованою, а й безпечною, забезпечуючи захист особистої інформації користувачів [7].

Таким чином розробка програмної платформи для соціальних мереж є актуальною задачею.

Мета роботи: розробити проєкт програмної платформи з використанням стеку MERN, котра б дозволила створювати власні соціальні мережі з клієнтською частиною, що виконується у браузері.

Завдання:

1. Виконати огляд предметної області соціальних мереж.
2. Проаналізувати основні наявні платформи для побудови соціальних мереж.
3. Виконати огляд програмних архітектур для побудови програмного забезпечення для соціальних мереж.
4. Виконати аналіз основних технологій розробки програмного забезпечення, що використовуються для розробки соціальних мереж.
5. Розробити прототип програмної платформи для побудови соціальних мереж.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Поняття про соціальні мережі як засіб обміну цифровими даними

Соціальні мережі (СМ) є одними з найпопулярніших сервісів в Інтернеті, які мають мільярди користувачів. Проте СМ не просто надзвичайно популярні; вони також змінили спосіб нашої взаємодії з Інтернетом та його ресурсами: OSN дозволяють користувачам, які не мають досвіду, створювати та підтримувати особистий простір у Інтернеті простим способом. СМ еволюціонували з часом, забезпечуючи кілька засобів зв’язку та обміну, які змушують користувачів ділитися через них величезною кількістю особистої інформації [16].

Тим не менш, користувачі сучасних популярних СМ страждають від небажаних побічних ефектів, що є наслідком централізованої архітектури СМ, у якій один постачальник має повноваження, які супроводжуються оператором системи. Ці побічні ефекти включають: необхідність високого рівня довіри до постачальника СМ, проблеми цензури та конфіденційності.

Існування потужного системного оператора в поєднанні зі стимулами монетизації викликає занепокоєння щодо конфіденційності. Крім того, у сучасних СМ зустрічаються різні типи цензури: цензура, пов’язана з контентом, з огляду на різні правила та традиції в різних країнах, а також цензура, пов’язана з конкретними особами, що означає заборону певним групам населення доступ до мережі, наприклад доступ до СМ від Google в Китаї.

Однак важливість СМ для щоденного міжособистісного спілкування ставить постачальників цих послуг у позицію сторожів у частинах соціального життя своїх користувачів. Через цю залежність користувачі дуже схильні приймати згадані побічні ефекти та навіть невигідні умови використання, оскільки постачальники СМ можуть виключати користувачів, а згодом і з частини їхніх соціальних контактів. Автори підходів до децентралізованих СМ (ДСМ) прагнуть скасувати постачальників СМ і побічні ефекти централізованих

соціальних мереж шляхом створення децентралізованих систем, які забезпечують функціональність соціальних мереж.

Наведемо значення декількох основних термінів, котрі мають значення для цієї роботи.

Соціальна мережа – це «онлайн-платформа, яка (1) надає користувачеві послуги для створення загальнодоступного профілю та явного оголошення зв'язку між своїм профілем і профілем інших користувачів; (2) дозволяє користувачеві ділитися інформацією та контентом з обраними користувачами або громадськістю».

Персональні ідентифікаційні дані згідно Директиви про захист даних 95/46/ЕС Європейського парламенту – це «будь-яка інформація, що стосується [...] фізичної особи [...], яку можна ідентифікувати, прямо чи опосередковано, зокрема за посиланням [...] на один або більше факторів, характерних для її фізичних, фізіологічних, ментальних, економічних, культурних чи соціальних рис ідентичності». Ідентифікаційна інформація може складатися з вмісту, наприклад зображень і повідомлень, а також усіх типів випадкових даних, які можуть бути отримані, наприклад, з технічних властивостей (наприклад, розміру) вмісту або параметрів зв'язку.

Авторизація – це механізм прийняття рішення про легітимізацію на основі попередньо визначених правил. Контроль доступу означає дії, спрямовані на те, щоб дозволити узаконеним суб'єктам отримати доступ до вмісту та заборонити неавторизований доступ. Відповідно до «конфіденційності як контролю» в [17] , конфіденційність у нашому контексті означає ефективність користувачів, щоб мати можливість обмежити доступ до інформації, за яку користувач відповідає (як виробник повідомлення). На ефективність можуть вплинути технічні та соціальні (наприклад, соціальна інженерія) перешкоди.

Оскільки кожна соціальна мережа надає різну функціональність, ми вирішили розрізнити базову функціональність, яка має бути частиною системи, щоб вважатися її власне СМ, і розширену функціональність, яка не відповідає вимогам, але розширює послугу певним чином.

Виходячи з означення СМ, основні функції складаються з таких сценарії використання.

1. Керування профілем: створення, підтримка та видалення профілів користувачів, що згодом включає механізми авторизації атрибутів профілю.

2. Обробка зв'язків: встановлення та видалення нових декларацій про зв'язки.

3. Взаємодія: прямі взаємодії (внутрішня система обміну повідомленнями – (1:1) і непрямі взаємодії шляхом обміну вмістом (1:n).

Розширена функціональність – це набір функцій, які надають деякі з сучасних СМ. Причина згадки про них у цьому визначенні полягає в тому, що ми вважаємо їх важливими елементами, які сприяють привабливості та популярності платформ СМ. Виділимо наступне:

1. API, що дозволяє стороннім програмам працювати на платформі СМ.

2. Функція пошуку для пошуку інших користувачів СМ.

3. Система рекомендацій, яка рекомендує користувачам стати друзями або вмістом для споживання.

4. Конектор для соціальних мереж, що об’єднує різні соціальні мережі.

Компоненти соціальних мереж коротко визначені нижче для подальшого використання в цій роботі.

Користувач: особа або організація, що володіє ідентифікатором і профілем користувача в СМ.

Профіль користувача: цифрове представлення користувача в СМ, що містить усі елементи даних користувача.

Ідентифікатор користувача: унікальний ідентифікатор для кожного користувача, який є частиною системи СМ.

Зміст (Контент): Елемент даних, який зберігається або спільно використовується в СМ.

Підключення: декларована приналежність або знайомство між користувачами (наприклад, дружба).

Вузол: мережевий пристрій, який використовується користувачами для підключення до СМ.

Сервер: мережевий пристрій, який надійно підтримує надання послуг.

1.2 Особливості проєктування мобільних соціальних мереж

На відміну від традиційних соціальних мереж, які орієнтовані на окремих людей, мобільні соціальні мережі можуть скористатися додатковими можливостями сучасних мобільних пристроїв, таких як смартфони. Ці можливості, як-от приймач глобальної системи позиціонування (GPS), сенсорні модулі (камери, акселерометр, давачі гравітації тощо) і кілька радіостанцій (стільниковий зв'язок другого/третього/четвертого покоління, WiFi, Bluetooth, WiFi Direct тощо), дозволяють мобільним соціальним мережам покращувати звичайні соціальні мережі додатковими функціями, такими як визначення місця розташування, здатність асинхронної взаємодії, здатність захоплювати та позначати медіа, а також здатність автоматично обробляти дані давачів.

На відміну від звичайних соціальних мереж, у яких люди взаємодіють через Інтернет, численні радіостанції в мобільних пристроях дозволяють також працювати в опортуністичних мережах, де кожен вузол може діяти як хост, маршрутизатор або шлюз і з'єднуватися з іншими вузлами в спеціальним способом, без володіння або надбання будь-яких знань про топологію мережі. Отже, мобільні мережі є потенційно привабливими для підтримки взаємодії та співпраці між людьми в ряді мобільних середовищ, оскільки вони можуть використовувати переваги як бездротових мереж різного типу.

Багато існуючих платформ і рішень уже підтримують мобільні соціальні мережі. Такі комерційні платформи, як Facebook і Twitter, дуже популярні та широко використовуються на смартфонах і мобільних пристроях. Вони в основному зосереджені на кінцевих користувачах, але також надають деякі програмні інтерфейси (API), які можна використовувати для розробки нових функцій і програм на основі цих платформ.

Крім того, дослідницьке співтовариство розробляє багато експериментальних рішень, наприклад, для підтримки розробки додатків і опортуністичних соціальних зв'язків, які могли б дозволити додаткам працювати в децентралізованих і централізованих мобільних системах.

1.3 Популярні комерційні платформи СМ

На відміну від соціальних мереж минулого часу, які базуються на використанні веб-браузерів як клієнтів для доступу до соціальних веб-сайтів через Інтернет, поточні комерційні рішення для СМ також надають спеціальні мобільні програми, які працюють на мобільних пристроях. Такі додатки зазвичай мають спеціальні інтерфейси користувача, які дозволяють користувачам легко отримувати доступ до відповідних соціальних мереж, одночасно використовуючи переваги мобільних розподілених обчислень. Наприклад, програми можуть розподіляти деякі обчислювальні завдання, такі як попереднє зберігання та попередня обробка даних соціального контексту на мобільних пристроях. Ці схеми можуть допомогти зменшити мережеві накладні витрати та час затримки, коли користувачі мережі отримують доступ і/або публікують дані з/у соціальних мережах через свої мобільні пристрої.

1) Facebook – це сервіс соціальної мережі. Це цікавий ресурс для дослідників, які цікавляться функціями та послугами соціальних мереж, оскільки він містить різноманітні моделі використання та технологічні можливості, які комбінують онлайн- і офлайн-з'єднання. Крім того, попередні дослідження показали, що користувачі Facebook схильні шукати людей, з якими вони мають офлайн-зв'язок, а не шукати абсолютно незнайомих людей для зустрічей [18], і вони зазвичай зацікавлені в тому, про що думають їхні друзі. Поточна мобільна програма Facebook підтримує більшість оригінальних послуг і функцій, які надає соціальний веб-сайт Facebook. Мобільна версія Facebook також надає відкриту платформу з API, яку можуть використовувати сторонні постачальники для створення програм, які додають більше функціональних можливостей

оригінальній мобільній програмі Facebook, таким чином дозволяючи користувачам насолоджуватися більш багатим досвідом.

2) Twitter (тепер X): на відміну від більшості соціальних мереж, Twitter зосереджується на послугах мікроблогів, і існує багато розширень, які підтримують твіти зображень і текстів, довжина яких перевищує 140 слів. Будь-який користувач Twitter може стежити за іншими або за ним можна стежити, а також отримувати інформацію про те, що ви робите чи думаєте, від своїх друзів Twitter у режимі реального часу. Загалом Facebook допомагає користувачам взаємодіяти з друзями та родиною в реальному світі, а Twitter допомагає користувачам спілкуватися з людьми, які цікавляться подібними речами. Подібно до Facebook, Twitter також надає відповідну мобільну програму та пов'язані API для програмістів додатків, які дозволяють їм розробляти нові функції та послуги для Twitter. Однак, оскільки Twitter призначений для мікроблогів з новинами в режимі реального часу, він дозволяє користувачам мобільних телефонів оновлювати нові повідомлення на веб-сайті Twitter не лише через мобільну програму Twitter, а й за допомогою служби коротких повідомлень (SMS).

Крім того, багато систем мікроблогів, які надають API, також сумісні з Twitter. Це означає, що якщо програмісти розробили нову мобільну програму в іншій системі мікроблогів, як-от StatusNet, вони також можуть легко й ефективно перенести програму в Twitter.

Більшість популярних платформ для соціальних мереж, в тому числі розглянутих вище, є комерційними. Вони пропонуються певними постачальниками послуг через Інтернет. Через мобільність і динамічність розподілених користувачів СМ у деяких ситуаціях ці платформи не в змозі задовольнити різноманітні вимоги.

Такі ситуації включають віддалені райони з поганим бездротовим покриттям і, отже, ненадійним доступом до Інтернету, автомобільні мережі, у яких мережеві ситуації дуже динамічні, а передача інформації чутлива до затримки часу, або сценарії катастрофи, коли комунікаційна інфраструктура

буде пошкоджена. Крім того, комерційні платформи часто мають деякі обмеження, які роблять їх не дуже гнучкими чи зручними.

Таблиця 1.1 – Деякі платформи для побудови різнотипних мобільних соціальних мереж

Проєкт	Опис	Категорія	Підтримка ad-hoc	Операційні системи	Можливість масштабу-вання
MobiSN	Семантична основа для мобільної спеціальної соціальної мережі, яка реалізована в Java 2 Micro Edition (J2ME)	Фреймворк	Так	Мобільні операційні системи, які підтримують J2ME	Немає
Ambient Talk	Керована подіями мова для розробки додатків у мобільній тимчасовій мережі	Мова програму-вання	Так	Мобільні операційні системи, які підтримують J2ME	–
MobiSoC	забезпечує загальну платформу, яка базується на сервіс-орієнтованій архітектурі для швидкої розробки та розгортання мобільних соціальних комп'ютерних програм	Проміжне програмне забезпе-чення	Немає	Windows Mobile	Так
Haggle	Мережева структура, орієнтована на дані, забезпечує безперебійне підключення до мережі та функціональність	Фреймворк	Так	Android, iOS	Так
Spiderweb	Мобільна програма, яка дозволяє створювати спонтанні соціальні мережі для співпраці через Bluetooth	Застосунок	Немає	Мобільні операційні системи, які підтримують J2ME	Немає
S-AFrame	Агентна багаторівнева структура з контекстно-залежним семантичним сервісом, підтримує ефективну розробку контекстно-залежних додатків	Фреймворк	Так	Мобільні операційні системи, які підтримують J2ME	Так

Тому багато дослідників у всьому світі розробляють нові рішення для СМ з відкритим вихідним кодом, використовуючи такі підходи, як нові мови, нові структури/проміжне програмне забезпечення тощо. Більшість цих рішень орієнтовано на мобільні розподілені мережі, які не тільки використовують переваги комерційних СМ, але й доповнюють їх. Доступні рішення, що підтримують мобільні соціальні мережі, які є результатом різних проектів, підсумовано в Таблиці 1.1 [19].

1.4 Висновки до розділу 1

Таким чином, підсумовуючи проведений аналіз предметної області розробки соціальних мереж можемо підсумувати наступне.

1. Незалежно від типу та призначення соціальної мережі, всі вони задовольняють вимогам обміну інформацією між користувачами цієї мережі для певних їх груп.

2. Сучасні соціальні мережі вимагають від програмного забезпечення, котре надає такі послуги, підтримувати мобільність користувачів і, відповідно, працювати як на стаціонарних комп'ютерах через браузери, так і на різноманітних мобільних пристроях з використанням спеціального клієнтського програмного забезпечення.

3. Сучасні найпопулярніші соціальні мережі відповідають вказаним вимогам і обираються користувачами у відповідності до цілі, з якою користувач хоче поширювати інформацію між користувачами цієї ж мережі.

2 ОСОБЛИВОСТІ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ СОЦІАЛЬНИХ МЕРЕЖ

2.1 Принципи проєктування та розробки соціальних мереж

Ніша соціальних медіа, як згадувалось вище, є дуже перспективною для бізнесу, хоча її займають великі гравці, такі як Facebook, Instagram, Flickr тощо. Ця ніша процвітає, але насправді є простір для нових ідей, і зацікавлення в розробці нової соціальної мережі завжди є.

Щоб зрозуміти, як побудувати соціальну мережу, спочатку визначимо, що таке соціальна мережа і які функції вона має. Це питання частково розглянуто в попередніх розділах роботи. Але тепер поглянемо на розробку СМ з точки зору тих, хто буде її створювати і потім розповсюджувати.

Отже, кожна соціальна мережа – це платформа (веб-сайт або мобільний додаток), де люди зустрічаються в Інтернеті, щоб спілкуватися чи ділитися інформацією, публікуючи дописи, надсилаючи повідомлення або залишаючи коментарі. Користувачі можуть заповнити свої особисті дані на сторінках облікового запису, щоб розповісти світу про себе. Часто представляється, як повний (в ідеалі) граф (Рисунок 2.1).

Класифікація соціальних мереж допоможе розібратися в гравцях ринку і може дати нові ідеї. Розглянемо основні типи СМ.

Класифікація за платформою.

– Доступ до соціальних мереж на основі веб можливий лише через браузери. Більшість популярних мереж, таких як Facebook і Twitter, починалися як веб-сайти. Це може бути у випадку, якщо соціальна мережа є частиною іншого проєкту. Наприклад, команда створила платформу електронного навчання з функцією соціальної мережі, де студенти можуть спілкуватися, співпрацювати один з одним і налагоджувати зв’язки.

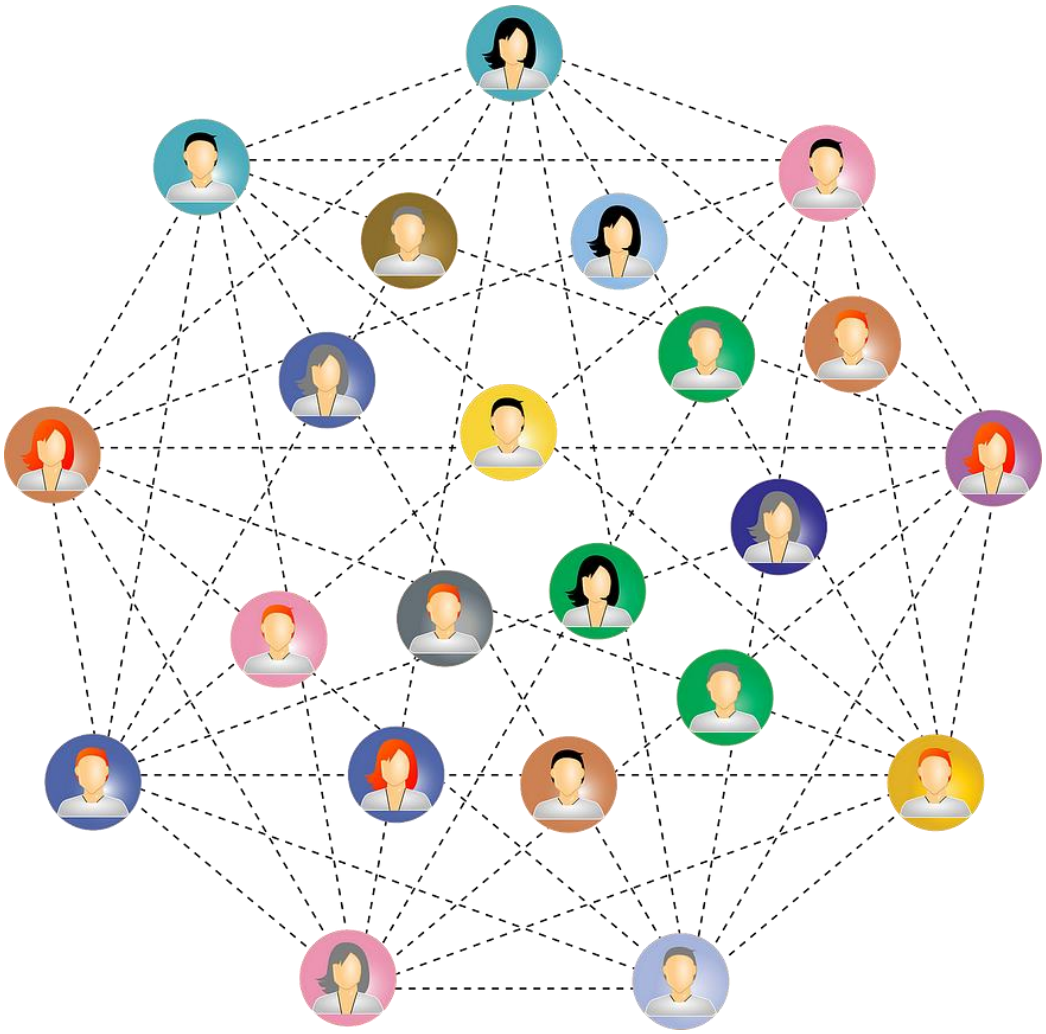

Рисунок 2.1 – Умовне зображення повного графу соціальної мережі

– Гібридні мережі мають як веб-версію, так і версію програми. Це найцінніший спосіб, оскільки не кожен користувач може завантажити додаткову програму на свій телефон і хоче отримати доступ до неї через браузер.

– Мобільні соціальні мережі розроблені для роботи лише на мобільних пристроях і планшетах. Ми вважаємо, що більшість соціальних мереж повинні мати мобільний додаток, оскільки користувачі щодня користуються соціальними мережами і хочуть мати додаток на головному екрані.

Класифікація за аудиторією.

– Існують загальні соціальні мережі, такі як Facebook, які об’єднують усіх людей. Це місце також зберігають Twitter, Instagram і Pinterest.

– Нішеві мережі мають на меті об’єднати людей з однаковими інтересами. Наприклад, Soundcloud був розроблений для музикантів, щоб ділитися своєю музикою, а Goodreads для книголюбів.

Класифікація за призначенням.

– Соціальні мережі знайомств допомагають людям знайти своє майбутнє кохання. Такі програми, як Badoo і Tinder, надають цю послугу.

– Інформаційні соціальні мережі створені для того, щоб надавати користувачам інформацію, яку вони шукають. Це включає веб-сайти з оглядами, такі як Yelp, Q&A, Quora.

– Освітні соціальні мережі дозволяють студентам спілкуватися, співпрацювати та ділитися знаннями один з одним.

– Комерційні мережі надають користувачам можливість робити покупки онлайн, а також спілкуватися всередині платформи. Наприклад, на CreativeMarket незалежні митці можуть продавати свої цифрові продукти, ставити лайки та стежити за іншими митцями та ділитися новинами.

– Мережі обміну мультимедіа дозволяють користувачам ділитися своїм вмістом і спілкуватися з іншими творцями. Візьмемо Flickr як приклад, де художники діляться своїми фотографіями та знаходять натхнення.

– Платформи B2C дозволяють людям знаходити інформацію про компанії, послуги чи продукти та робити покупки.

– Мережі соціальних зв’язків просто об’єднують людей, як це робить Facebook.

Тип проектованої соціальної мережі буде прямо залежати від мети та ідеї. Кроки створення соціальної мережі викладемо нижче.

Крок 1. Слід визначити свою цільову аудиторію.

Для кого створюється соціальна мережа? Хто і навіщо нею буде користуватися? Яку проблему можна вирішити з допомогою СМ? Дуже важливо відповісти на ці запитання, оскільки від цього залежатимуть стратегія, дизайн і маркетинг.

Визначення цільової аудиторії здійснюється шляхом аналізу наступних факторів.

Демографічні показники.

Демографічні показники допоможуть почати звузити аудиторію, вони включають:

– Вік.
– Стать.
– Професія/область зайнятості.
– Дохід.
– Сімейний стан.
– Рівень освіти.

Розташування.

Можливо, проектована соціальна мережа призначена для місцевої спільноти або працює лише в певній країні. Краще визначити це зараз, оскільки цей фактор буде використовуватися пізніше в маркетингу.

Психографіка включає поведінку аудиторії, хобі, інтереси та діяльність. Наприклад:

– Садівники.
– Любителі футболу.
– Домогосподарки.
– Збирачі фарб.
– Люди, які шукають найнижчу ціну.
– Люди, які часто роблять покупки в Інтернеті.

Існують рекомендації [20] поставитися до цього серйозно і витратити чимало часу, оскільки це є фундаментальним для всього вашого проекту.

Крок 2. Розробка стратегії.

Варто почати з малого і спочатку створити MVP (мінімально цінний продукт). Такий підхід допоможе побачити зворотній зв’язок і підтримку від аудиторії, перш ніж створити все (Рисунок 2.2). Гнучка методологія SCRUM, яку застосовують для організації роботи над проєктом більшість компаній,

допомагає економити час і гроші, поступово додаючи функції та вивчаючи відгуки користувачів.

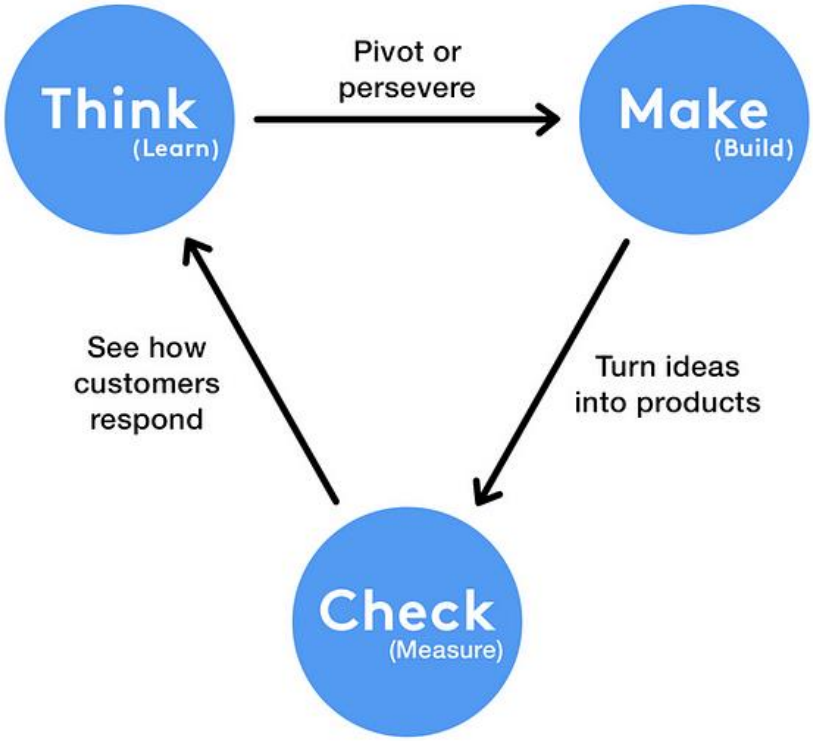

Рисунок 2.2 – Процес створення мінімально цінного продукту (MVP)

Отже, перше питання, на яке потрібно відповісти, це те, що таке є MVP в даному контексті. За якою головною особливістю приходять користувачі? Після цього подумайте про стратегію зростання та маркетинг, ви повинні бути готові до розширення. Складіть принаймні плани на перший і п'ять років .

Крок 3. Попередня розробка (прототипування).

Соціальна мережа – це велике та складне програмне забезпечення, для створення якого потрібна команда розробників. Коли вже є визначена цільова аудиторія, чітка ідея, розроблена стратегія розвитку та ідеї дизайну, необхідно розглянути деякі речі, перш ніж звертатися до постачальника послуг з розробки програмного забезпечення:

– Гроші. Розробка соціальної мережі займе від 200 до 1000 годин роботи в залежності від набору функцій. Тому на цьому етапі слід мати чітке

уявлення про вартість проєкту та вирішити, де шукати команду розробників: у своїй країні чи звернутись до аутсорсингових компаній закордоном.

– Платформа. Треба створити лише веб-сайт, лише програму чи обидва засоби? Це слід вирішити, перш ніж звертатися до команди розробників, щоб вони могли дати точну оцінку.

– Оцінка трафіку. Обсяг трафіку, який утримуватиме розроблювана мережа, визначатиме технологію, яку використовуватиме команда розробників.

Крок 4. Дизайн.

Сам дизайн складається з 4 етапів:

– Скетчінг. Дуже простий і грубий образ програми, щоб мати бачення кількості екранів і логіки між ними.

– Каркас. Це фундаментальний крок, який допомагає команді розробників побачити скелет соціальної мережі, розташування елементів і те, як користувач буде з ними взаємодіяти.

– Прототипування. Прототип – це робоча модель проекту, яка дає клієнту та команді розробників зрозуміти, як насправді працюватиме додаток. На цьому етапі збираються ідеї та вносяться зміни до процесу розробки, що економить час і гроші.

– UI/UX дизайн. Тут дизайнери перетворюють прототип на точне зображення майбутньої соціальної мережі, включаючи анімацію.

Крок 5. Розробка та забезпечення якості.

Це момент, коли ідея перетворюється на реальність. Цей процес включає бек-енд роботу, як-от налаштування серверів, баз даних, API тощо, і фронт-енд: розробка візуального вигляду та взаємодії веб-сайту або мобільного додатка.

Під час і після розробки проводяться перевірки якості, щоб переконатися, що все працює нормально та код не містить помилок, а також ми перевіряємо взаємодію з користувачем.

Крок 6. Видавництво та маркетинг.

Коли соціальна мережа розроблена, настав час показати її світові, розгорнувши веб-сайт на серверах або опублікувавши програму в Google Play

Market і AppStore. Ми рекомендуємо подумати про маркетинг до процесу розробки, а також почати його до завершення етапу розробки, щоб зібрати базу перших користувачів.

Крок 7. Підтримка.

Після випуску проекту необхідно забезпечити підтримку та обслуговування, особливо якщо це версія MVP. Це допомагає негайно вирішувати проблеми та надсилати оновлення на основі відгуків користувачів. Перший місяць життя соціальної мережі визначить її репутацію.

Метрики для проекту соціальної мережі дозволять відстежувати бізнес-аналітику вашої соціальної мережі, яка допоможе вам зрозуміти, як розвивається ваш проект, і вчасно прийняти правильні рішення. Основні метрики перераховані нижче за текстом.

Вартість залучення клієнта (Customer acquisition cost, CAC) – це загальна сума всіх грошей, витрачених на рекламу для залучення клієнта. Цей показник визначить масштабованість і прибутковість бізнес-моделі.

Рівень утримання користувачів – це пропорція кількості користувачів до дати їх першої реєстрації. Слід розуміти, чи залишаються користувачі у мережі.

Коефіцієнт відтоку вказує на кількість користувачів, які перестали користуватися веб-сайтом або програмою. Чим нижчий цей показник, тим краще.

Залучення користувачів показує, як часто користувачі виконують дії у соціальній мережі, як довго вони залишаються всередині та як часто повертаються (кожні 2 дні або кожні 2 години до прикладу).

Зростання показує, як швидко база користувачів зростає з місяця в місяць.

Сума грошей, необхідна щомісяця для підтримки проекту. Він включає маркетинг, витрати на сервер, персонал тощо. Цей показник буде необхідним на етапах масштабування та збору коштів.

Процес створення соціальної мережі може бути складним, особливо якщо ви розробляєте щось нове та революційне. Далі перераховано декілька основних проблем, з якими розробник зіткнеться під час створення соціальної мережі.

Продуктивність. Зараз кожен проект не виживе на ринку, якщо його продуктивність низька. Користувач стає все більш затребуваним. Веб-сайт має завантажуватися менше ніж за 3 секунди, інакше 53% користувачів покинуть сайт, згідно з дослідженнями Google [21]. Мобільний додаток також має завантажуватися та працювати швидко, без уповільнень. Цього можна досягти, використовуючи потужні технології та правильну архітектуру.

Безпека. Користувачі очікують, що платформа СМ буде безпечна, і вони можуть контролювати свою конфіденційність. Люди повинні мати можливість приховувати свої особисті дані, видимість публікацій тощо. Крім того, обмін повідомленнями між людьми має бути приватним і зашифрованим.

Дизайн і досвід користувача (UI/UX).

Коли користувач потрапляє у веб- або мобільний додаток, він повинен миттєво зрозуміти, як ним користуватися, і отримати те, що він шукав. Це називається інтуїтивно зрозумілим дизайном і є необхідним у сучасному світі, тому що, як згадувалося раніше, користувач не хоче чекати або думати про те, як використовувати програму. Якщо створено абсолютно унікальний проект із великою кількістю функцій, надайте користувачеві приємну та швидку покрокову документацію.

Крім того, не забуваємо про красивий інтерфейс користувача та зручну взаємодію з користувачем за допомогою анімації та добре продуманої навігації, щоб зробити роботу користувача максимально приємною.

Маркетинг. Створити чудовий продукт – це половина шляху. Однією з найбільших проблем є правильний маркетинг і залучення достатньої кількості користувачів, щоб вижити на першому етапі. Якщо ви правильно визначили свою цільову аудиторію та створили продукт, який задовольняє її потреби, все має пройти гладко.

Скільки коштує створити соціальну мережу. Вартість кожного проекту оцінюється в залежності від особливостей і технологій. Але для загального бачення приблизна тривалість розробки соціальної мережі з логіном, особистим кабінетом, обміном повідомленнями та стрічкою новин становить близько 300

годин. Погодинна ставка розробника в США становить 50 – 200 доларів, в Україні вона становить 30 – 120 доларів [20].

2.2 Програмні архітектури соціальних мереж

Наведемо короткий опис деяких програмних архітектур, які використовуються у соціальних мережах:

1. Клієнт-серверна архітектура в Facebook.

Клієнт-серверна архітектура використовується в Facebook для забезпечення зв'язку між клієнтськими пристроями (веб-браузери, мобільні додатки) і серверами компанії [8]. Сервери Facebook забезпечують обробку запитів від користувачів, зберігання даних та надання вмісту (Рисунок 2.3).

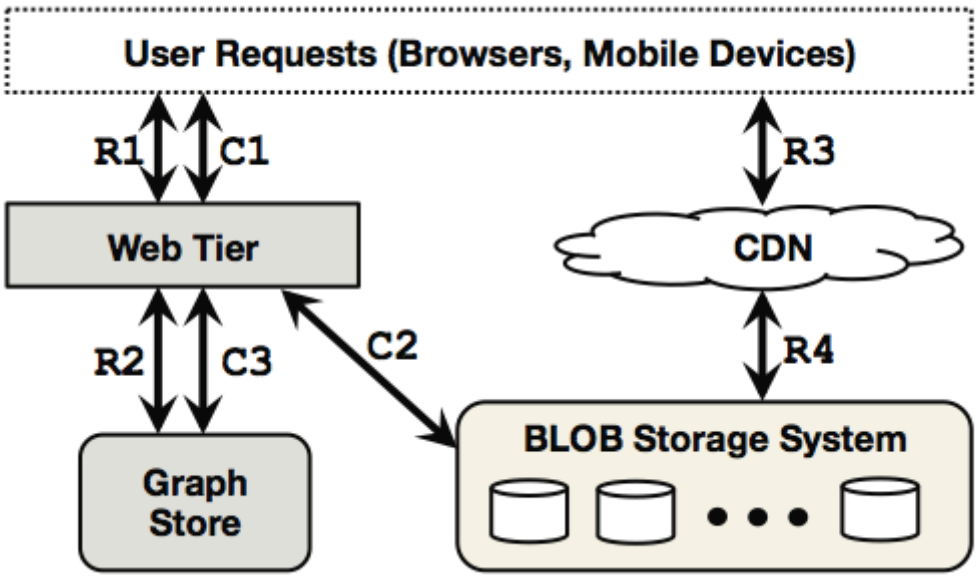

Рисунок 2.3 – Базова архітектура СМ Facebook

На рисунку 2.3 Використано наступні позначення:

1. User Requests – запити користувача від браузера чи з мобільного застосунку.

2. Web Tier – проміжне програмне забезпечення, котра працює разом з Web-сервером і забезпечує обмін даними між сервером СМ та клієнтською частиною згідно протоколу HTTPS.

3. CND – Content Delivery Network. Мережа і розповсюдження контенту. Представляє собою розподілену систему серверів, що забезпечу доступність медіаконтенту.

4. BLOB Storage System – Система зберігання бінарних даних. Фактично система зберігання медіаконтенту. Представляє собою розподілену систему зберігання файлів.

5. Graph Store – система зберігання інформації про взаємозв'язки учасників соцмережі. На логічному рівні представляє собою орієнтований граф.

6. Rn (Reading) – операції читання даних (від сервера до клієнта).

7. Cn (Create) – операції запису даних (від клієнта до сервера).

Це дозволяє користувачам отримувати доступ до своїх профілів, спілкуватися, ділитися вмістом та використовувати функціональні можливості платформи.

2. Мікросервісна архітектура в Twitter.

Twitter використовує мікросервісну архітектуру, де функціональність платформи розбита на окремі сервіси [9]. Кожен сервіс відповідає за конкретну функцію, таку як таймлайн, автентифікація, повідомлення тощо (Рисунок 2.4).

Основними елементами на рисунку 2.4 є власне мікросервіси, позначені голубими блоками, серед яких до прикладу Twitt Service, Graph Service та ін.

Також варто звернути увагу на те, що дана СМ побудована на програмних продуктах компанії Apache, а саме використовує стек Apache Kafka / Apache Spark / Hadoop для забезпечення потокового опрацювання великих даних в реальному часі.

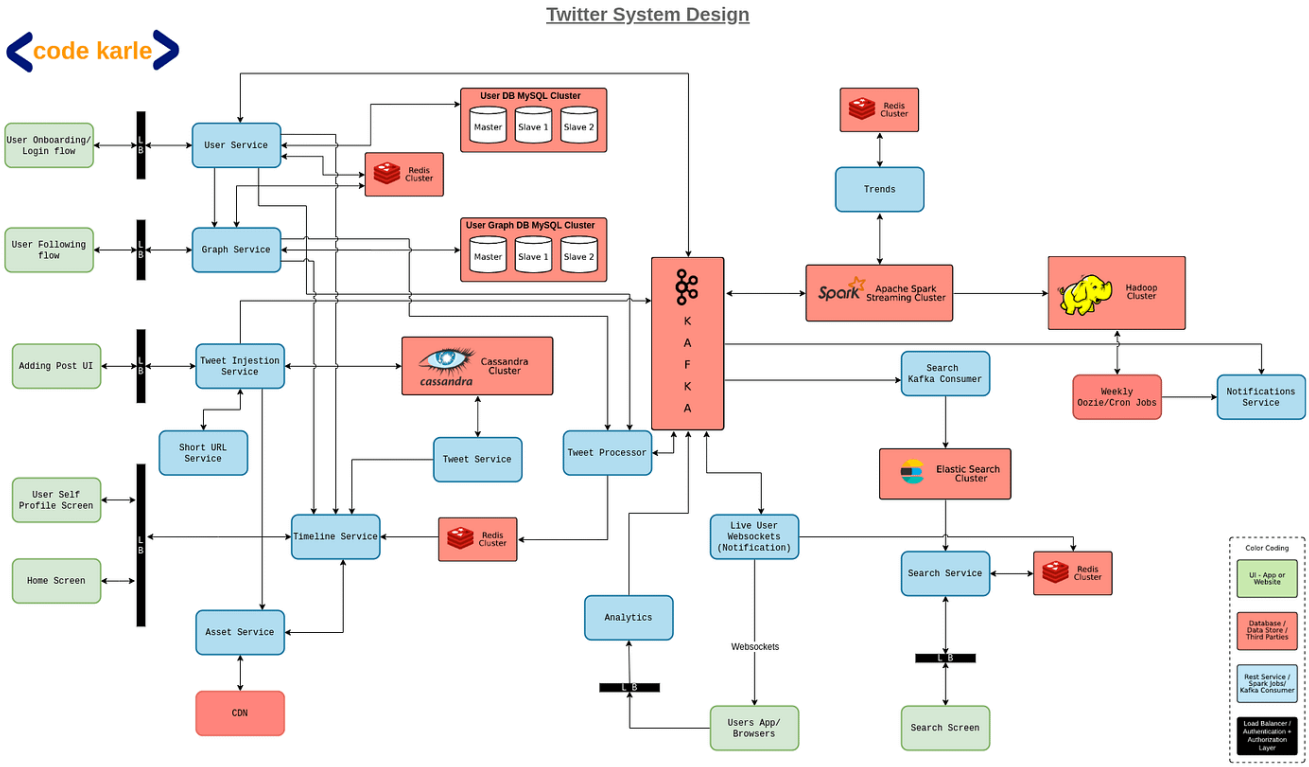

Рисунок 2.4 – Архітектура СМ Twitter

Ця архітектура дозволяє розробникам працювати над окремими частинами системи, спрощує масштабування і підтримку коду, а також полегшує впровадження нових функцій.

3. Архітектура з орієнтацією на послуги (сервіс-орієнтована архітектура – SOA) в LinkedIn.

LinkedIn використовує архітектуру, орієнтовану на послуги (Рисунок 2.5). Це означає, що функціональність платформи поділена на набір незалежних сервісів, кожен з яких відповідає за конкретну операцію або функцію (наприклад, пошук, сторінки профілів, повідомлення) [10].

Варто звернути увагу на ZooKeeper. Це програмний продукт компанії Apache. Що забезпечує координацію роботи складових частин розподіленого програмного продукту і базується на принципі прийому/передачі повідомлень на базі шаблону проєктування Publish–Subscribe (публікація-підписка). Це сприяє швидкості розробки, покращенню масштабованості і забезпеченню стабільності системи.

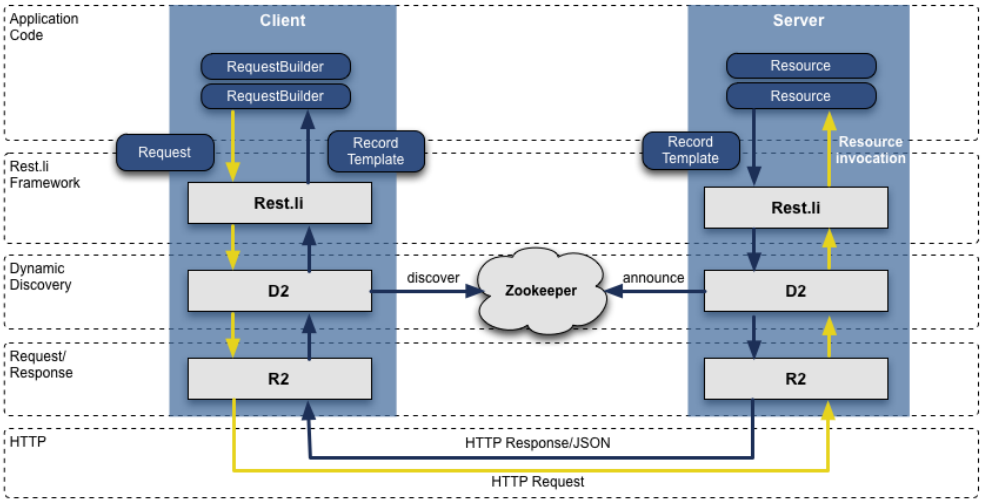

Рисунок 2.5 – Програмна архітектура CM LinkedIn

4. Архітектура з орієнтацією на події в Snapchat.

Snapchat використовує архітектуру, зорієнтовану на події, де основною одиницею обробки є події, які створюються користувачами (наприклад, відправка фото або відео) (Рисунок 2.6). Дана архітектура побудована на елементах власної розробки.

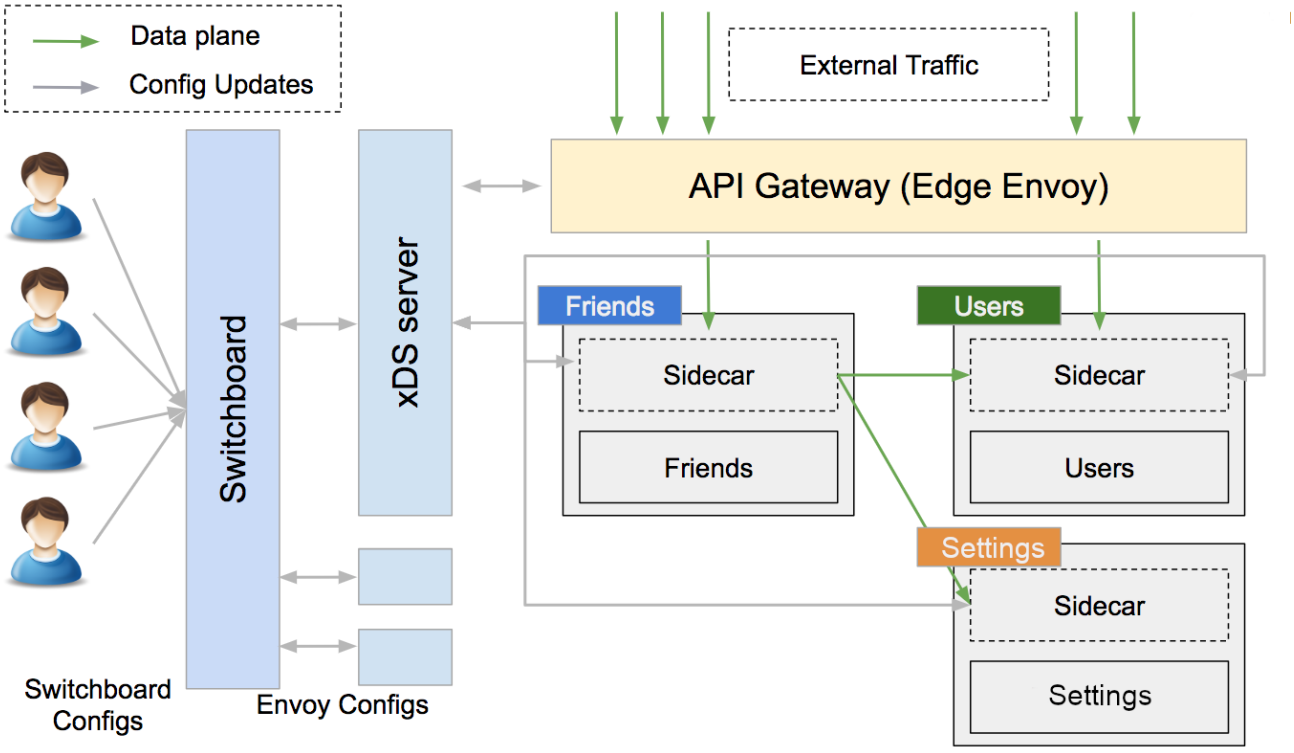

Рисунок 2.6 – Програмна архітектура CM Snapchat

Ці приклади, а також інші джерела, наприклад [12 – 15], вказують на різноманітність та унікальність програмних архітектур, які використовуються у великих соціальних мережах для забезпечення функціональності, продуктивності та ефективності платформи.

2.3 Висновки до розділу 2

1. Розробка програмного забезпечення для соціальної мережі вимагає від розробника реалізації всіх етапів життєвого циклу програмного продукту.
2. Крім власне розробки, важливими для успішного проєкту є етапи допроєктного оцінювання завдання на розробку мережі, аналіз цільової аудиторії користувачів, забезпечення процесів підтримки та супроводу працюючого програмного продукту.
3. Для реалізації сервісу соціальної мережі програмне забезпечення як правило розробляється командою, що працює за одним із гнучких принципів організації роботи команди, наприклад SCRUM.
4. Відповідно до гнучких методологій розробки важливим є створення MVP – Minimum viable product (мінімально життєздатний продукт), котрий реалізовуватиме основний функціонал. На подальших ітераціях розробки він доповнюватиметься новими елементами, функціями, властивостями. Таким чином повний набір вимог не обов'язково має бути сформульований до початку процесу розробки, що є властивим для традиційних водоспадних моделей життєвого циклу і спричиняє невдале завершення проєктів при недосконалому плануванні.
5. Для нашої розробки після аналізу програмних архітектур найпопулярніших соціальних мереж відповідно до завдання на роботу логічним рішенням буде застосувати архітектуру типу "клієнт-сервер", оскільки стек MERN передбачає використання відповідних технологій розробки, що реалізують обрану архітектуру.

6. В даній роботі зосередимося на створенні програмного забезпечення для соціально мережі, котре реалізовувати серверну частину та клієнта тільки для виконання в середовищі браузера.

3 РОЗРОБКА ПРОГРАМНОЇ ПЛАТФОРМИ ДЛЯ СОЦІАЛЬНОЇ МЕРЕЖІ

3.1 Основні технології front-end розробки

Для реалізації програмної платформи побудови соціальних мереж клієнт-серверної архітектури слід проаналізувати основні технології, що застосовуються для побудови клієнтської частини та сервера. Отже, для розробки веб-сторінок існує чіткий розподіл між інтерфейсом користувача і сервером. Бекенд-технології пов’язані з серверною частиною програм. У свою чергу, інтерфейс пов’язаний з браузером та інтерфейсами/взаємодіями користувача. Хоча обидві сторони однаково сприяють успіху проєкту, цей розділ зосередиться на найкращих варіантах інтерфейсної частини проєкту у вигляді фронт-енд скриптів для виконання в середовищі браузера.

Немає простої відповіді, коли справа доходить до вибору релевантної фронт-енд технології. Однак є деякі речі, які можна врахувати, перш ніж прийняти остаточне рішення. У наведеному нижче списку показані деякі фактори, які варто розглянути, щоб прояснити будь-які сумніви:

– Розмір проєкту.

– Досвід команди в конкретних технологіях.

– Популярність інструменту/фреймворку (з великою популярністю зазвичай приходить велика спільнота, що може бути корисним).

– Чи складається дизайн з декількох сторінок зі схожими компонентами? Можливо, варто використати систему дизайну?

– Чи багато елементів з динамічними значеннями, як у якійсь інформаційній панелі? React (або інший JS фреймворк) може бути корисним.

– Масштабованість.

– Чи потрібен проєкту SSG або SSR? Next.js підтримує обидва.

– Скільки команд працює/буде працювати над проєктом? Мікрофронтенди можуть вирішити деякі проблеми у великих командах.

– Чи забезпечує конкретна технологія те, що потрібно вашому проєкту?

Хоча використання новітніх, найбільш трендових технологій означає, що ви отримуєте доступ до більш ефективних рішень для вирішення проблем, це не є стійким рішенням у довгостроковій перспективі. Частіше за все, перевірений послужний список працюючих, успішних проєктів є набагато кращим показником майбутнього успіху впровадження, ніж швидкоплинна популярність.

Спостерігайте за ринком, експериментуйте, пробуйте нове, але пам'ятайте, що ви завжди повинні приймати рішення, виходячи з потреб і проблем, які ви повинні вирішити у фронт-енд проєкті. Тому варто спочатку проаналізувати вимоги, а потім оцінити наявні можливості.

Проаналізуємо основні технології розробки.

Найновішим інструментам може бракувати належної підтримки, інтеграції чи документації, тому важливо враховувати популярність і успіх технології, як це видно на прикладі CSS і HTML, які, незважаючи на те, що вони не є «новими» чи «новими», залишаються основне для розробки інтерфейсу.

1. HTML (мова розмітки гіпертексту).

Почнемо з основ: HTML є основою розробки інтерфейсу, і її важливість навряд чи скоро зменшиться. Ця технологія дозволяє створювати структуру або «розмітку» веб-сайту, хоча вона може не пропонувати найдинамічніший інтерфейс користувача, елементи HTML надають основні інструменти для додавання кнопок, форм, контейнерів та інших функцій на веб-сторінку.

2. CSS (каскадні таблиці стилів).

Ще одна основна технологія зовнішнього інтерфейсу, CSS, дозволяє додавати додаткові правила стилю до створеної структури HTML, гарантуючи, що веб-програма буде адаптивною та візуально привабливою завдяки анімації, яка робить її інтерактивною та бездоганно працює на будь-якому розмірі екрана чи пристрої.

3. JavaScript.

JavaScript (JS) – це мова програмування, яка завжди була і залишатиметься однією з найважливіших інтерфейсних технологій для веб- і мобільних додатків, дозволяючи розробникам динамічно змінювати вміст як мобільних, так і настільних додатків і впроваджувати різні функції, такі як корзини покупок в онлайн-магазинах, складні анімації, калькулятори податків, ігри для веб-браузера тощо.

4. React.

React – це бібліотека JavaScript, яка пропонує новий синтаксис, але все ще працює на JavaScript, що робить створення веб-додатків швидшим і простішим завдяки її декларативному та компонентному характеру.

Початково React був розроблений компанією Facebook для підвищення продуктивності та обслуговування для внутрішнього використання. Тепер це проєкт із відкритим вихідним кодом із великою спільнотою розробників інтерфейсу та є одним із найпопулярніших інструментів JS.

5. Angular.

Angular – це повноцінний інтерфейсний фреймворк на основі JavaScript, який підтримується Google, а модульність і компонентний підхід, який він забезпечує, зібрали сильну спільноту серед розробників інтерфейсу по всьому світу.

Його простота робить цей фреймворк ідеальним для односторінкових (SPA – Single Page Application) додатків, і хоча деякі розробники кажуть, що вивчити Angular трохи важче, ніж React, це безумовно того варте, оскільки на ринку існує високий попит на експертів у цій технології, а попит на Angular розробників демонструє прогнозоване зростання приблизно на 31% між 2016 і 2026 роками [22].

6. Vue.

Vue вважається одним із найкращих варіантів інтерфейсу для односторінкових (SPA) веб-додатків і є фреймворком JavaScript, призначеним для створення інтерфейсів користувача, який забезпечує декларативний і

компонентний принцип програмування, подібний до React і Angular, з мільйонами щотижневих завантажень на npm , демонструючи свою популярність на ринку.

7. Vite.js.

Vite.js – це швидкий і легкий інструмент, призначений для сучасної веб-розробки. Він спеціально оптимізований для створення веб-додатків, які використовують сучасні фреймворки JavaScript, такі як React, Vue.js і Svelte. Vite.js досягає своєї швидкості завдяки використанню інноваційного сервера розробки, який використовує власні модулі ES у браузері, забезпечуючи швидку гарячу заміну модулів і швидкий початковий час завантаження.

Крім того, Vite.js підтримує такі функції, як попередня обробка CSS, підтримка TypeScript і систему плагінів для розширення його функціональності.

8. Svelte & SvelteKit.

Згідно зі звітом про стан JS на кінець 2022 року [23], Svelte йде відразу за трійкою кращих (React, Angular і Vue) з точки зору утримання, інтересу та обізнаності. Svelte – це популярна мова для створення веб-додатків, яка дозволяє розробникам писати реактивні компоненти, які ефективно оновлюють DOM. Він використовує компілятор для генерації оптимізованого коду JavaScript, зменшуючи кількість коду, який потрібно завантажити та виконати в браузері.

Для полегшення доступу SvelteKit побудований на основі Svelte, який забезпечує рендеринг на стороні сервера, маршрутизацію та інші функції, щоб полегшити створення складних веб-програм. SvelteKit використовує переваги моделі реактивного програмування Svelte і компілює код програми в набір серверних і клієнтських файлів JavaScript, оптимізованих для продуктивності.

9. Next.js.

Next.js – це інструмент, який допомагає створювати надшвидкі веб-сайти за допомогою React. Незалежно від того, чи це генерація статичного сайту, чи візуалізація на стороні сервера, ви можете вибрати будь-який підхід залежно від потреб проєкту. Продуктивність веб-сторінок на основі фреймворку Next.js є надзвичайною завдяки численним методам оптимізації, які він використовує, як-

от попереднє відтворення. Крім того, фреймворк також оптимізований для SEO, що є великою перевагою для онлайн-бізнесу та маркетингових проєктів.

10. TypeScript.

Усі, хто має певний досвід фронт-енд розробки, знають, що JavaScript код може бути складним, а іноді навіть небезпечним, коли мова заходить про типи. З TypeScript це вже не так.

Дана мова дозволяє писати звичайний JS-код з деяким додатковим синтаксисом для визначень типів. Ця технологія стає стандартом у сучасній розробці інтерфейсу, оскільки дозволяє розробникам TypeScript економити час в ході відладки дефектів, пов’язаних із типами даних. Крім того, це робить кінцеве програмне забезпечення більш стабільним.

TypeScript видає повідомлення про помилку кожного разу, коли ми робимо помилку, пов’язану з типами змінних, тому ми можемо внести належні коригування, перш ніж отримати звіти про помилки від клієнтів.

11. Gatsby.

Gatsby – це фреймворк, який поєднує найкращі частини React, GraphQL і webpack. Завдяки Gatsby ви можете створювати швидкі та реактивні користувацькі інтерфейси для споживачів, зберігаючи роботу розробника відносно приємною. Крім того, фреймворк надає деякі готові функції, як-от оптимізацію зображення та поділ коду, що дозволяє завантажувати функції на вимогу та покращує продуктивність проєкту. Рівень даних Gatsby базується на GraphQL.

12. React Native.

React Native – це фреймворк на основі React, який дозволяє створювати кросплатформні мобільні програми для платформ iOS і Android. Завдяки React Native розробка міжплатформного коду стала простішою та доступнішою. Розробники можуть створювати мобільні додатки, які виглядають і відчуваються рідними для багатьох платформ.

13. Flutter.

Flutter – це фреймворк із відкритим кодом, створений Google за допомогою мови програмування Dart. Створювати інтерактивні елементи інтерфейсу набагато простіше за допомогою віджетів Flutter, які дозволяють створювати динамічні та інтерактивні програми на основі попередньо створених шаблонів.

За допомогою Flutter веб-розробники можуть створювати крос-платформні програми для мобільних пристроїв з єдиною кодовою базою та однією мовою програмування.

14. Astro.

Astro – це новий фреймворк MPA (Modular Page Applications), який спрямований на спрощення процесу створення та розгортання веб-програм, дозволяючи розробникам використовувати для їх розробки будь-які популярні фреймворки.

Сильні сторони Astro включають швидкий час створення та здатність легко включати рендеринг на стороні сервера, гідратацію на стороні клієнта та динамічний імпорт. Він використовує концепцію Astro Islands, модель веб-архітектури, вперше розроблену Astro, для розробки інтерактивних компонентів інтерфейсу користувача на статичній сторінці HTML.

15. Monorepo.

Monorepo – це архітектурна концепція, на основі якої ви створюєте єдиний репозиторій, у якому зберігається код для кількох проєктів. Це створює єдине джерело коду та полегшує обмін компонентами та активами між проєктами.

16. Microfrontends.

Microfrontends – це архітектурна концепція, яка реалізує ідею мікросервісів у front-end розробці. Завдяки такому підходу можна розділити програму на основі її функцій, і кожною з них може керувати зовсім інша команда в іншому сховищі коду.

Якщо ви працюєте в основному з невеликими командами і хочете збільшити автономність і масштабованість своїх проєктів, Microfrontends може стати хорошим вибором.

17. Three.js.

Three.js – це бібліотека JavaScript, яка використовується для створення тривимірної веб-графіки та анімації. Величезною перевагою цього фреймворку є те, що з кожним роком він стає все популярнішим. На даний момент Three.js зібрав понад 94 000 зірок на GitHub у 2023 році [24]. З більшою підтримкою спільноти з часом можна очікувати покращень у налагодженні та документації.

18. Tailwind.

Tailwind – це структура CSS, яка дозволяє створювати стилі безпосередньо у розмітці HTML, додаючи відповідні класи до окремих елементів. Завдяки такому підходу створювати стилі та змінювати їх за потреби надзвичайно легко та швидко.

19. Remix.

Remix – це фреймворк React, який дозволяє використовувати рендеринг на стороні сервера. Він отримує дані на сервері та надає HTML безпосередньо користувачеві. Крім того, він поставляється з деякими додатковими вбудованими функціями, такими як вкладені сторінки, обмеження помилок і обробники стану завантаження.

20. Headless CMS.

CMS означає систему управління контентом. Це сховища вмісту, до яких можна отримати доступ через RESTful API або запити GraphQL, що особливо корисно для статичних сайтів і блогів. Headless CMS розділяє інтерфейс і сервер програми, полегшуючи роботу маркетологів і розробників, не заважаючи один одному. На ринку існує багато рішень Headless CMS, включаючи GraphCMS, Contentful, Contentstack і Prepr.

21. GraphQL.

GraphQL – це мова запитів для API. Вона дає саме те, що потрібно від сервера, без зайвих даних. Завдяки своїй продуктивності та дивовижним інструментам розробника, які постачаються з нею, GraphQL стала трендовою технологією в сучасному світі веб-розробки.

22. Web3 Apps.

Зростаюче впровадження технологій Web3 значно впливає на ландшафт розробки інтерфейсу. Оскільки децентралізовані програми продовжують набирати популярність, розробники все частіше звертаються до фреймворків та інструментів Web3 для створення інтерфейсів користувача, які взаємодіють із цими децентралізованими мережами. Щоб задовольнити вимоги цих додатків, розробники використовують різноманітні інтерфейсні фреймворки та бібліотеки Web3, такі як Web3.js, ethers.js, wagmi та Truffle.

23. PWA.

PWA означає прогресивний веб-додаток (Progressive Web App). Це програма, заснована на таких веб-технологіях, як HTML, CSS і JavaScript, із зовнішнім виглядом і функціями, схожими на власні мобільні програми. Коли відкрити сайт PWA на телефоні, смартфон запропонує вам встановити. PWA з кожним роком стають все більш популярними.

3.2 Аналіз стеку технологій MERN для побудови платформи для соціальних мереж

MERN означає MongoDB, Express, React, Node, тобто це перші літери назв чотирьох ключових технологій, які складають стек.

1. MongoDB – база даних документів.
2. Express(.js) – веб-фреймворк Node.js.
3. React(.js) – фреймворк JavaScript на стороні клієнта.
4. Node(.js) – головний веб-сервер JavaScript.

Express і Node складають середній (програмний) рівень. Express.js – це веб-фреймворк на стороні сервера, а Node.js – популярна та потужна серверна платформа JavaScript. Незалежно від того, який варіант ви виберете, ME(RVA)N є ідеальним підходом до роботи з JavaScript і JSON протягом всього часу роботи над проєктом.

Архітектура MERN дозволяє легко побудувати трирівневу архітектуру (фронт-енд, бек-енд, база даних), повністю використовуючи JavaScript і JSON (Рисунок 1.3).

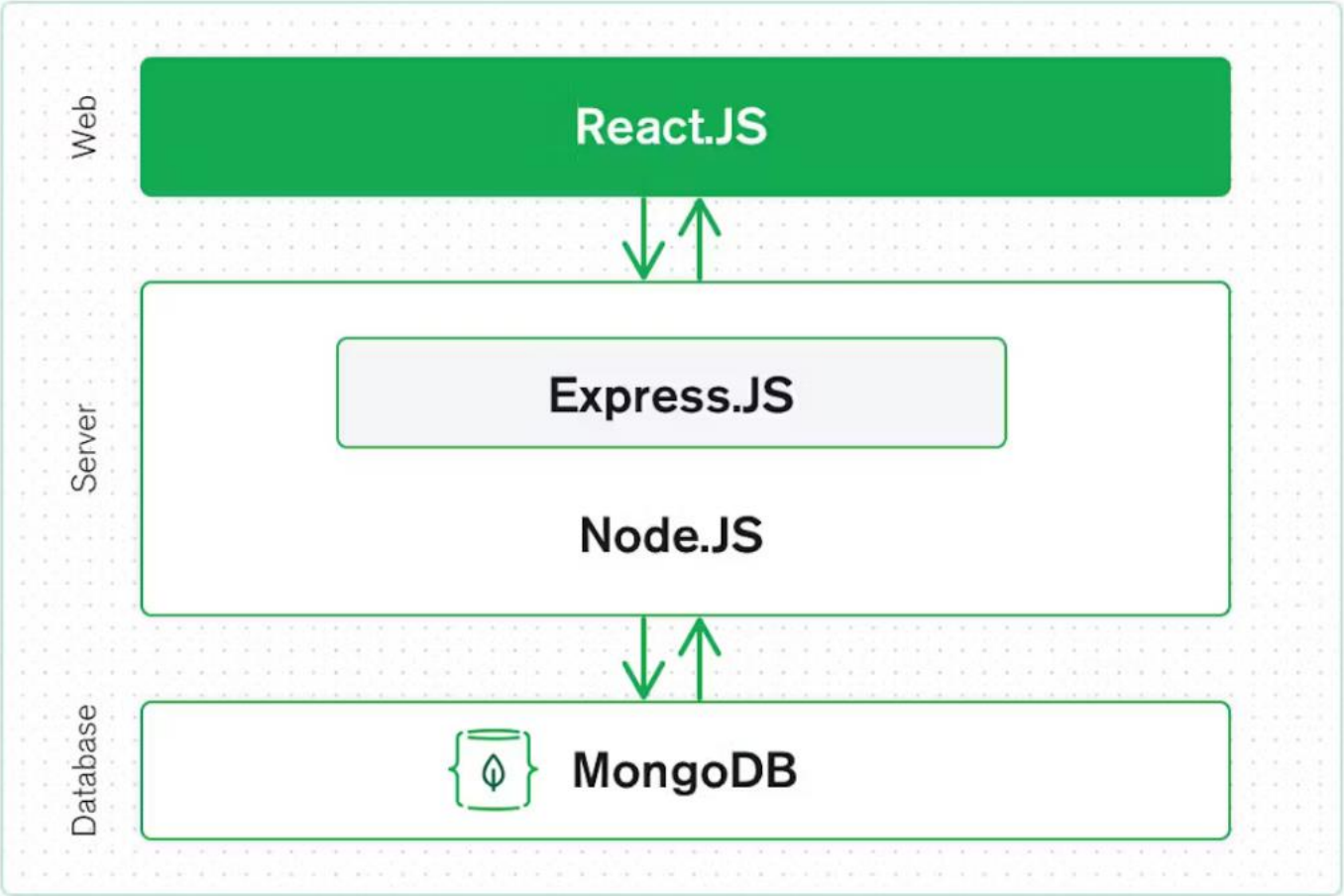

Рисунок 3.1 – Архітектура MERN

Опишемо складові цього стеку технологій.

3.2.2 Фронт-енд частина на React.js

Найвищим рівнем стеку MERN є React.js, декларативна структура JavaScript для створення динамічних клієнтських програм у HTML. React дозволяє створювати складні інтерфейси за допомогою простих компонентів, підключати їх до даних на сервері та відтворювати їх як HTML.

Сильна сторона React – це обробка інтерфейсів із збереженням стану, керованих даними, з мінімальною кількістю коду та мінімальними труднощами,

а також має всі переваги, які ви очікуєте від сучасної веб-платформи: чудова підтримка форм, обробки помилок, подій, списків тощо.

3.2.3 Сервер на фреймворках Express.js та Node.js

На цьому рівні реалізується бізнес-логіка роботи програми.

Наступним рівнем є серверна структура Express.js, яка працює на сервері Node.js. Express.js називає себе «швидким, непереборним, мінімалістичним веб-фреймворком для Node.js», і це справді саме те, чим він є. Express.js має потужні моделі для маршрутизації URL-адрес (зіставлення вхідної URL-адреси з функцією сервера) і обробки запитів і відповідей HTTP.

Роблячи XML HTTP-запити (XHR) або GET або POST з інтерфейсу React.js, ви можете підключитися до функцій Express.js, які забезпечують роботу вашої програми. Ці функції, у свою чергу, використовують драйвери MongoDB Node.js за допомогою зворотних викликів для доступу та оновлення даних у вашій базі даних MongoDB.

3.2.4 Рівень бази даних MongoDB

Якщо програма зберігає будь-які дані (профілі користувачів, вміст, коментарі, завантаження, події тощо), тоді, природно, потрібна база даних, з якою так само легко працювати, як з React, Express і Node.

Тут на допомогу приходить MongoDB: документи JSON, створені у фронт-енд частині React.js, можна надіслати на сервер Express.js, де їх можна обробити та (якщо вони валідні) зберегти безпосередньо в MongoDB для подальшого отримання.

Чи є MERN комплексним рішенням? Так, MERN це повний стек, який дотримується традиційної трирівневої архітектурної моделі, включаючи рівень зовнішнього відображення (React.js), рівень додатків (Express.js і Node.js) і рівень бази даних (MongoDB).

3.2.5 Переваги використання стеку MERN

Почнемо з MongoDB, бази даних документів у корені стеку MERN. MongoDB було розроблено для зберігання даних JSON нативно (технічно вона використовує двійкову версію JSON під назвою BSON), і все, від інтерфейсу командного рядка до мови запитів (MQL або MongoDB Query Language) побудовано на JSON і JavaScript.

MongoDB надзвичайно добре працює з Node.js і неймовірно спрощує зберігання, обробку та представлення даних JSON на кожному рівні програми. Для хмарних додатків MongoDB Atlas робить це ще простіше, надаючи автоматичне масштабування кластера MongoDB у хмарному провайдері за вибором замовника, так само просто, як кілька натискань кнопки.

Express.js (працює на Node.js) і React.js роблять програму JavaScript/JSON MERN повним стеком. Express.js – це серверний фреймворк додатків, який обгортає HTTP-запити та відповіді та дозволяє легко зіставляти URL-адреси з функціями на стороні сервера. React.js – це зовнішній фреймворк JavaScript для побудови інтерактивних інтерфейсів користувача в HTML і зв’язку з віддаленим сервером.

Ця комбінація означає, що дані JSON природним чином переміщаються в обидва боки, що робить їх швидким для створення та досить простим для відладки. Крім того, потрібно знати лише одну мову програмування та структуру документа JSON, щоб зрозуміти всю систему. Таким чином, MERN – це вибір для сучасних веб-розробників, які прагнуть швидко рухатися, особливо для тих, хто має досвід React.js.

3.2.6 Варіанти використання MERN

Як і в будь-якому веб-стеку, у MERN Є можливість створювати все, що завгодно, хоча він ідеально підходить для випадків, які пов’язані з JSON, є рідними для хмари та мають динамічні веб-інтерфейси.

Приклади включають керування робочим процесом, агрегацію новин, додатки та календарі справ, інтерактивні форуми/соціальні продукти.

Як і в інших популярних веб-стеках, у MERN можна розробляти все, що завгодно. Тим не менш, він ідеально підходить для хмарних проектів, де вам потрібні інтенсивні JSON і динамічні веб-інтерфейси. Трішки докладніше про задачі, для яких використовується MERN.

Календарі та програми для справ.

Календар або додаток для справ – це елементарний проект, який може розповісти багато про механізми стеку MERN. За допомогою ReactJS можна створити інтерфейс календаря або інтерфейс програми зі списком TODO. Зберігання даних, отримання доступу до них, редагування, відображення в додатку TODO стало можливим за допомогою MongoDB.

Інтерактивні форуми.

Іншим хорошим варіантом використання MERN є інтерактивний форум, який може бути платформою соціальних мереж або веб-сайтом, який дозволяє користувачам ділитися повідомленнями та спілкуватися. Тема інтерактивного форуму може бути або не бути визначеною заздалегідь.

Продукти соціальних мереж.

Інтерактивний форум – це лише один із варіантів використання стеку MERN для соціальних мереж. До них належать оголошення, дописи, міні-веб-додаток, вбудований у сторінку в соціальних мережах тощо.

3.3 Установка компонентів стеку MERN

3.3.1 Установка MongoDB

Для операційної системи Windows спочатку потрібно завантажити найновіший реліз даної документо-орієнтованої СКБД за адресою https://www.mongodb.com/try/download/community. Ту вибрати реліз, версію операційної системи. Після скачування установочного файлу запустити його на

виконання та пройти традиційні кроки встановлення, притаманні цьому процесові для більшості програм для ОС Windows.

MongoDB вимагає окремої папки для зберігання файлів баз даних. За замовчуванням цією папкою є `C:\data\db`. Таким чином, потрібно створити цю директорію, або іншу. В такому разі в налаштуваннях сервера MongoDB потрібно вказати, де саме будуть зберігатись файли баз даних. Для цього потрібно вказати відповідне значення змінної оточення dbpath шляхом виконання команди (шлях до БД вказаний умовно):

```
C:\>mongod.exe --dbpath "C:\data"
```

Далі сервер буде готовий до роботи.

Для встановлення MongoDB для ОС Linux, наприклад Ubuntu, слід виконати наступну команду, щоби імпортувати відкритий ключ GPG MongoDB:

```
sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 \
--recv 7F0CEB10
```

Далі створити файл /etc/apt/sources.list.d/mongodb.list. Потім оновити репозиторій за допомогою команди:

```
sudo apt-get update
```

Останній крок – установка власне MongoDB:

```
apt-get install mongodb-10gen = 4.2
```

Для запуску сервісу MongoDB виконати команду:

```
sudo service mongodb start
```

Для зупинки сервісу MongoDB виконати команду:

```
sudo service mongodb stop
```

Для перезапуску MongoDB виконати команду:

```
sudo service mongodb restart
```

Для роботи з MongoDB виконуємо команду:

```
mongo
```

3.3.2 Установка фреймворку Node.js

Налаштування локального середовища для локального комп'ютера передбачає установку наступних двох програмних компонентів, доступних на вашому комп’ютері: (a) текстовий редактор і (b) двійкові інстальовані файли Node.js.

Текстовий редактор можна встановити відповідно до преференцій користувача. І використовується для написання програмного коду. Назва та версія текстового редактора можуть відрізнятися в різних операційних системах. Файли, які ви створюєте за допомогою редактора, називаються вихідними файлами та містять вихідний код програми. Вихідні файли для програм Node.js зазвичай мають розширення ".js".

Вихідний код, записаний у вихідному файлі, є просто javascript. Тому потрібне ще середовище виконання Node.js. Інтерпретатор Node.js використовуватиметься для інтерпретації та виконання вашого коду JavaScript.

Дистрибутив Node.js поставляється, як двійковий файл, який можна встановити для операційних систем SunOS, Linux, Mac OS X і Windows із 32-розрядною (386) і 64-розрядною (amd64) архітектурами процесорів x86.

У даному розділі описано, як інсталювати бінарний дистрибутив Node.js у різних ОС. Спочатку залежно від ОС слід завантажити відповідний архів Node.js

за адресою https://nodejs.org/en/download/. На момент підготовки кваліфікаційної роботи найсвіжішими версіями були наступні (таблиця 3.1)

Таблиця 3.1 – Установочні пакети Node.js для найрозповсюдженіших операційних систем

OS	**Назва архіву**
Windows	node-v21.4.0-x64.msi
Linux	node-v21.4.0-linux-x64.tar.xz
Mac	node-v21.4.0-darwin-x64.tar.gz

Після завантаження виконуємо власне установку для UNIX/Linux/Mac OS X. Відповідно до своєї ОС слід розпакувати архів node-v21.4.0-**osname**.tar.gz у тимчасову директорію /tmp чи іншу і після цього розпакувати видобуті з першого архіву файли у /usr/local/nodejs. Наприклад:

```
$ cd /tmp
$ wget http://nodejs.org/dist/v6.3.1/node-v6.3.1-linux-
x64.tar.gz
$ tar xvfz node-v6.3.1-linux-x64.tar.gz
$ mkdir -p /usr/local/nodejs
$ mv node-v6.3.1-linux-x64/* /usr/local/nodejs
```

Далі добавляємо змінну оточення PATH до налаштувань операційної системи, щоби інтерпретатор коду був доступний з будь-якої локації у файловій системі відповідно до своєї ОС (таблиці 3.2):

Таблиця 3.2 – Редагування змінно оточення PATH

OS	**Output**
Linux	export PATH=$PATH:/usr/local/nodejs/bin
Mac	export PATH=$PATH:/usr/local/nodejs/bin
FreeBSD	export PATH=$PATH:/usr/local/nodejs/bin

Для ОС Windows слід скористатись файлом MSI та дотримуватись підказок, щоб установити Node.js. За замовчуванням програма встановлення використовує дистрибутив Node.js у C:\Program Files\nodejs. Інсталятор має

встановити каталог C:\Program Files\nodejs\bin у змінній середовища PATH Windows.

Можна перевірити коректність встановлення фреймворку шляхом створення простого js-файлу та його виконання. Вмістиме файлу з іменем **main.js** може містити вивід стрічки:

```
/* Hello, World! program in node.js */
console.log("Hello, World!")
```

Далі виконуємо файл:

```
$ node main.js
```

Коли все встановлено правильно, побачимо повідомлення: " Hello, World!"

3.3.3 Установка та налаштування Express.js

У цьому розділі опишемо, як розпочати розробку та використання Express Framework. Для початку у повинні бути встановлені Node та npm (менеджер пакетів вузлів). Для Windows використовується менеджер пакетів nvm. Ї установку описувати не будемо. Щоби переконатися, що node і npm встановлено, можна виконати наступні команди у терміналі:

```
node --version
npm -- version
```

Коли маємо результат, подібний до наступного, то все готово для подальших кроків:

```
v5.0.0
3.5.2
```

Реєстр npm – це загальнодоступна колекція пакетів коду з відкритим вихідним кодом для Node.js, інтерфейсних веб-програм, мобільних програм, роботів, маршрутизаторів та незліченних інших потреб спільноти JavaScript. npm дозволяє отримати доступ до всіх цих пакетів і встановити їх локально. Ви можете переглянути список пакетів, доступних на npm, на npmJS.

Є два способи встановити пакет за допомогою npm: глобально та локально.

Глобальний метод зазвичай використовується для встановлення інструментів розробки та пакетів на основі CLI. Щоб установити пакет глобально, використовуйте наступний код:

```
npm install -g <назва-пакета>
```

Локальний метод зазвичай використовується для встановлення фреймворків і бібліотек. Локально встановлений пакет можна використовувати лише в каталозі, де він встановлений. Щоб установити пакет локально, скористайтеся тією ж командою, що й вище, без прапорця -g.

```
npm install <назва-пакета>
```

Щоразу, коли ми створюємо проект за допомогою npm, нам потрібно надати файл package.json, який містить усі подробиці про наш проект. npm полегшує нам налаштування цього файлу. Приклад налаштувань найпростішого проєкту.

Крок 1. У вікні терміналу створюємо новий каталог та переходимо в нього (Рисунок 3.2).

Рисунок 3.2 – Підготовка проєкту Node.js

Крок 2. Далі створюємо файл package.json за допомогою команди:

```
npm init
```

Коли залишити всі установки за замовчуванням, т отримаємо повідомлення, як на рисунку 3.3.

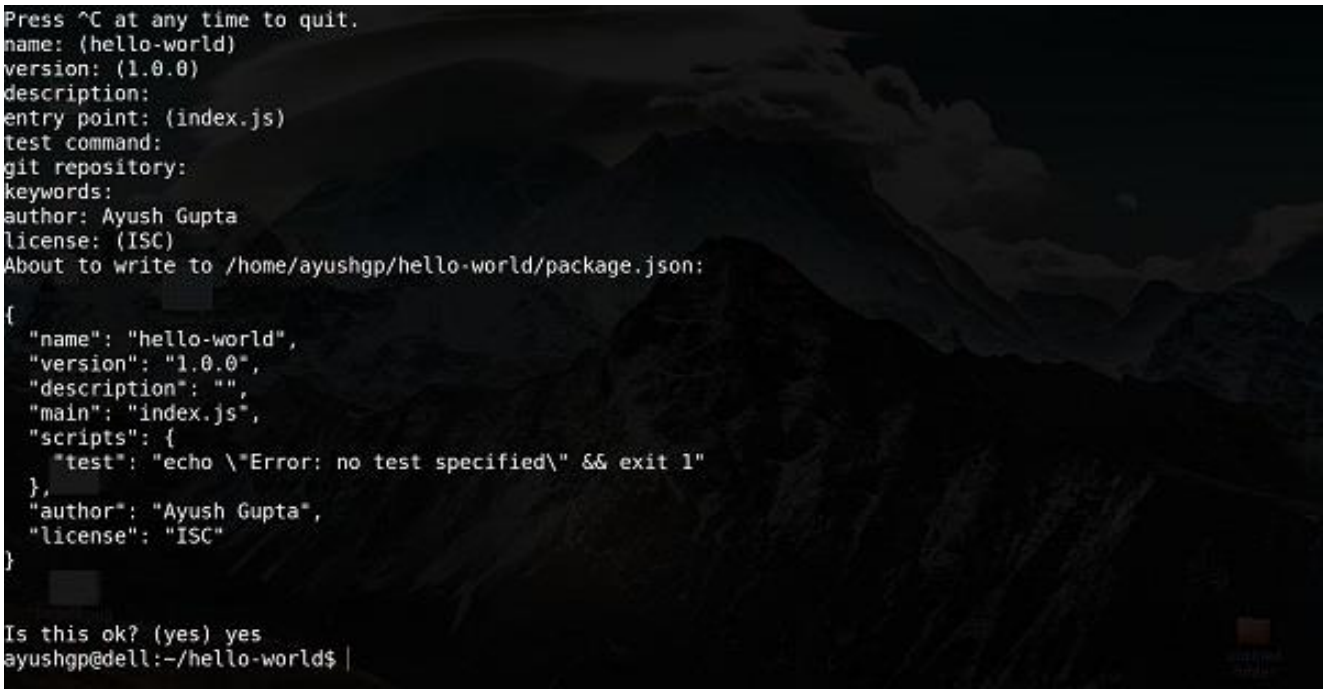

Рисунок 3.3 – Результат створення файлу package.json

Крок 3. Після установки package.json можна встановити сам фреймворк Express.js. Для цього та для добавлення його до пакету package.json слід виконати команду:

```
npm install --save express
```

Для підтвердження того, що Express.js встановлений, можна спробувати виконати наступний код:

```
ls node_modules
```

Для Windows:

```
dir node_modules
```

На разі це все, що потрібно, аби розпочати розробку за допомогою фреймворку Express. Щоб значно полегшити процес розробки, можна встановити інструмент від npm, nodemon. Цей інструмент перезапускає сервер, щойно вносимо зміни в будь-який із файлів, інакше нам потрібно буде

перезавантажувати сервер вручну після кожної зміни файлу. Щоб встановити nodemon, скористайтеся такою командою:

```
npm install -g nodemon
```

Тепер фреймворк Express готовий до роботи.

3.3.4 Встановлення та налаштування фреймворку ReactJS

У цьому розділі пояснюється встановлення бібліотеки React та пов’язаних із нею інструментів на локальному комп’ютері. Перш, ніж перейти до інсталяції, давайте спочатку перевіримо передумови.

React надає розробникам інструменти CLI для швидкого створення, розробки та розгортання веб-додатку на основі React. Інструменти React CLI залежать від Node.js і повинні бути вже встановлені у системі.

Для розробки легких функцій, таких як перевірка форми, діалогове вікно тощо, бібліотеку React можна безпосередньо включити у веб-додаток через CDN. Це схоже на використання бібліотеки jQuery у веб-додатку. Для помірних і великих програм рекомендується написати програму у вигляді декількох файлів, а потім використовувати збірник, такий як webpack, parcel, rollup тощо, щоб скомпілювати та згрупувати програму перед розгортанням коду.

Інструменти React (React toolchain) допомагають створювати, будувати, запускати та розгортати додаток React. React toolchain в основному надає стартовий шаблон проекту з усім необхідним кодом для завантаження програми.

Деякі з популярних інструментів для розробки програм React:

– Create React App − SPA-орієнтований інструментарій.

– Next.js − інструментальний ланцюжок, орієнтований на рендеринг на стороні сервера.

– Gatsby − ланцюжок інструментів, орієнтований на статичний вміст.

Інструменти, необхідні для розробки програми React:

– **The serve**, статичний сервер для обслуговування нашої програми під час розробки.

– Компілятор **Babel**.

– **Create React App CLI**.

У цьому розділі ми дізнаємось про основи вищезазначених інструментів і про те, як їх встановити.

Статичний сервер обслуговування.

The serve – це легкий веб-сервер. Він обслуговує статичні сайти та односторінкові програми. Завантажується швидко і споживає мінімум пам'яті. Його можна використовувати для обслуговування програми React. Його можна встановити за допомогою менеджера пакетів npm.

Babel – це компілятор JavaScript, який компілює багато варіантів (es2015, es6 тощо) JavaScript у стандартний код JavaScript, який підтримується всіма браузерами. React використовує JSX, розширення JavaScript для розробки коду інтерфейсу користувача. Babel використовується для компіляції коду JSX у код JavaScript.

Щоб установити Babel і його компаньйон React, треба виконати наведену нижче команду:

```
npm install babel-cli@6 babel-preset-react-app@3 -g
```

Babel допомагає писати програму в новому поколінні розширеного синтаксису JavaScript.

Create React App – це сучасний інструмент CLI для створення односторінкової програми React. Це стандартний інструмент, який підтримується спільнотою React. Він також працює з компілятором babel. Встановити Create React App у локальній системі можна, виконавши команду:

```
> npm install -g create-react-app
```

Набір інструментів React Create App використовує пакет react-scripts для створення та запуску програми. Як тільки ми почали працювати над додатком, ми можемо будь-коли оновити скрипт react до останньої версії за допомогою менеджера пакетів npm:

```
npm install react-scripts@latest
```

3.4 Розробка програмного забезпечення

В останньому підрозділі цієї роботи опишемо структуру програмного коду та продемонструємо деякі елементи інтерфейсу користувача для побудови програмної платформи для соціальних мереж.

Основна ідея цього продукту полягає в тому, що цей програмний каркас можна наповнити потрібним контентом і дизайном, створивши таким чином власний неповторний інтерфейс. Для спрощення окремий дизайн розробляти не будемо, а скористаємось оформленням та таблицями стилів мережі Facebook для демонстрації роботи платформи.

Загальна архітектура типу "клієнт-сервер", котрі ми згадували перших розділах цієї роботи, для реалізації засобами стеку MERN буде деталізована і приведена до загального архітектурного шаблону MVC (Mode – View – Controller). Таким чином, на стороні сервера буде реалізовано відображення бази даних (Модель) з доступом до БД MongoDB, а також бізнес-логіка (Контролер), що міститиме реалізацію бізнес-процесів в СМ (розміщення допису, пошук друзів тощо).

Клієнтська частина (View – подання) в архітектурі MVC містить презентаційну бізнес-логіку і призначена для реалізації інтерфейсу користувача. В ній реалізовуватиметься динамічне подання типових для соціальних мереж елементів користувача та діалогових вікон.

3.4.1 Створення серверної частини платформи для соціальних мереж

Почнемо опис серверної частини традиційно із відображення бази даних. Для цього в БД створимо шість колекцій для зберігання інформації про повідомлення в чаті, коментарів, запитів на дружбі, надсилання сповіщень, дописів, користувачів.

Схема каталогів для програмного коду моделей показана на рисунку 3.4.

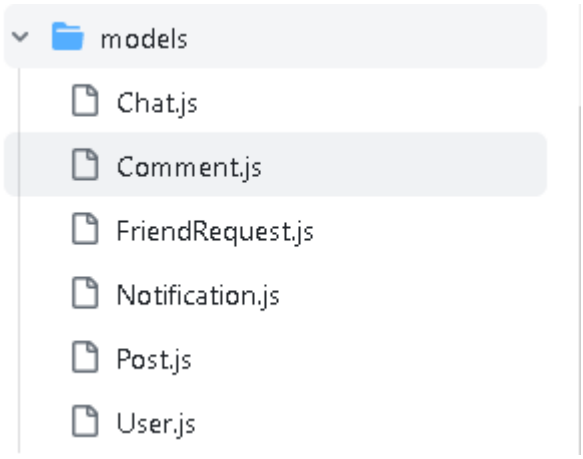

Рисунок 3.4 – Каталог та файли програмного коду реалізації моделей для доступу до бази даних

Програмний код цих файлів наведено у Додатку А.

Серверна бізнес-логіка була реалізована у вигляді контролерів. Кожен з них реалізовувати певний бізнес-процес. Наприклад, вхід користувача у систему, зміна пароля, вихід, створення дописів, створення повідомлення тощо.

Схема каталогів та файли контролерів показані на рисунку 3.5.

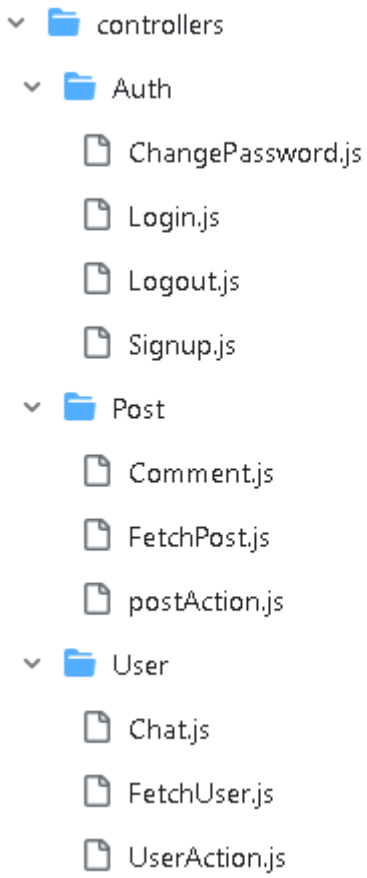

Рисунок 3.5 – Каталоги та файли контролерів серверної бізнес-логіки

Програмний код контролерів роботи з користувачами наведено у Додатку Б. В Додатку В наведено програмний код, що реалізує алгоритми створення контенту. Додаток Г містить програмний код контролерів, що реалізують дії користувача (запит дружби, написання повідомлення тощо).

3.4.2 Реалізація клієнтської (браузерної) частини платформи

В загальному клієнтську частину розроблюваного програмного продукту розроблено з елементами дизайну популярної СМ Facebook, але вони можуть бути легко замінені шляхом використання власних елементів дизайну: зображень, таблиць стилів.

"Скелет" інтерфейсу користувача реалізовано в каталозі Public, що містить html-розмітку початкової сторінки та логотип СМ (Рисунок 3.6).

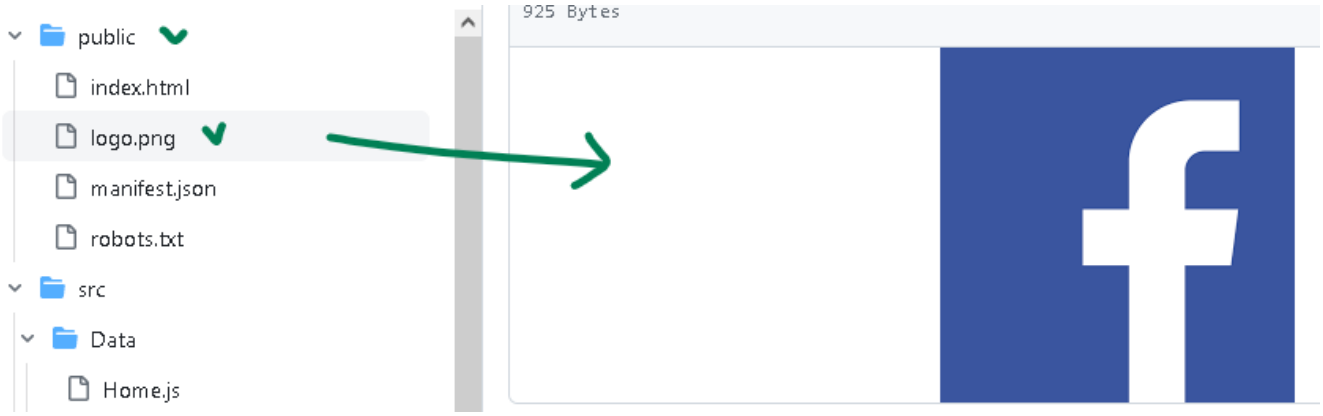

Рисунок 3.6 – Структура каталогу початкової сторінки та логотип СМ

Деякі інші елементи інтерфейсу розміщені у каталозі assets, який містить зображення для друзів, груп, логотип месенджера тощо (Рисунок 3.7).

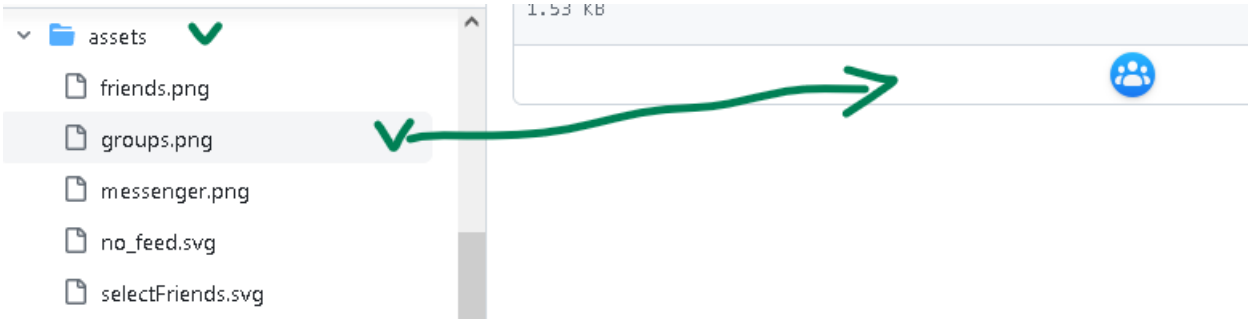

Рисунок 3.7 – Структура та вміст каталогу assets

Далі опишемо коротко деякі елементи фронт-енд частини проєкту. Реалізація цих елементів розділена по окремих каталогах, кожен з яких реалізує відповідний компонент інтерфейсу користувача. Список каталогів для кожного з цих компонентів показано на рисунку 3.8 і всі вони є підкаталогами каталогу components.

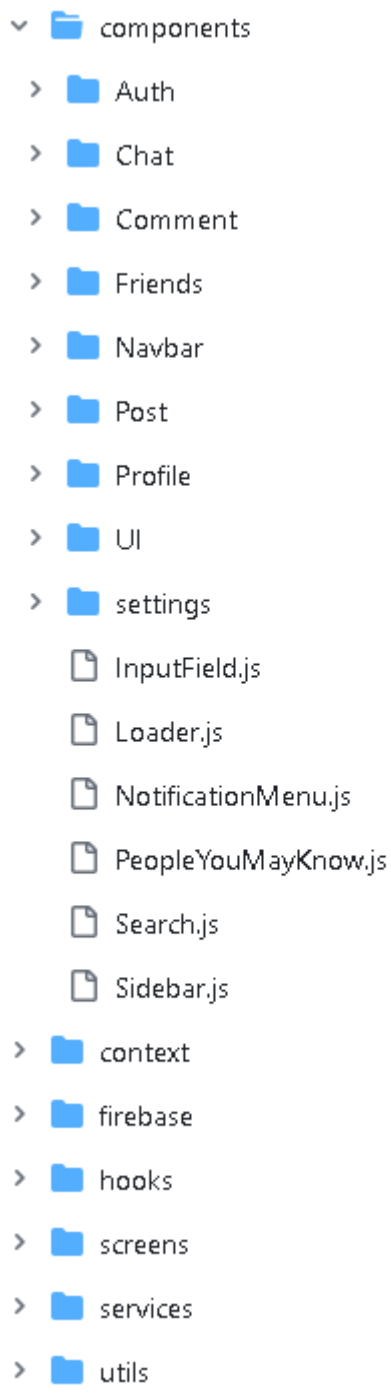

Рисунок 3.8 – Каталоги з реалізаціями елементів інтерфейсу користувача

Назви каталогів досить чітко дозволяють зрозуміти, який саме елемент вони реалізують. Наприклад, в каталозі Comments розміщені файли програмного коду для реалізації бізнес-логіки роботи з повідомленнями на стороні клієнта. В каталозі Friends містяться файли, що реалізують клієнтську бізнес-логіку отримання списку друзів, пошуку друзів тощо. Вміст каталогів Comments та Friends показано на рисунку 3.9.

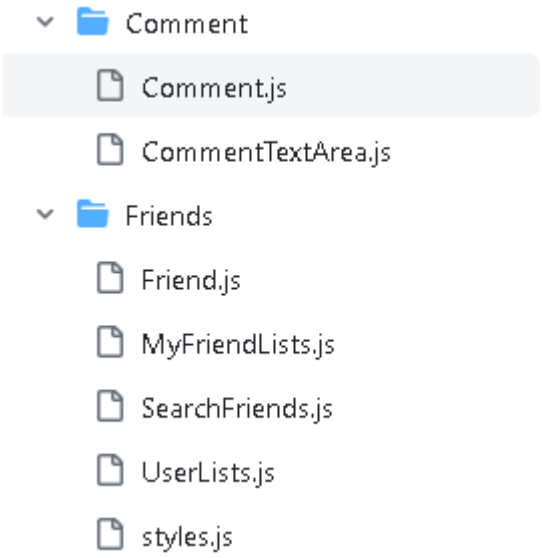

Рисунок 3.9 – Вміст каталогів Comments та Friends

В каталозі screens розміщені файли з програмник кодом, що відповідає за відображення сторінок авторизації, домашньої, вікна месенджера, створення допису, вікна налаштувань тощо. Вміст каталогу screens показано на рисунку 3.10.

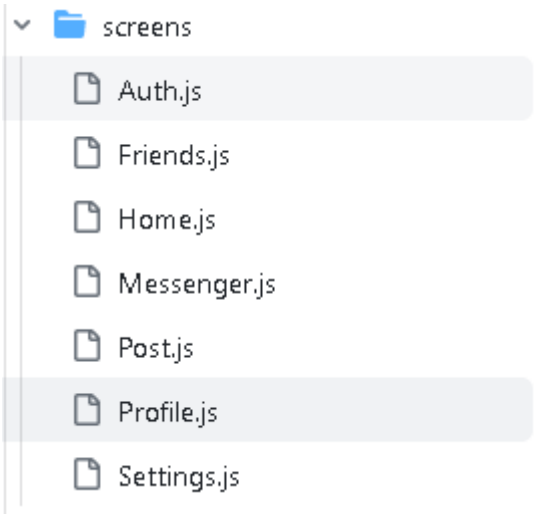

Рисунок 3.10 – Вміст каталогу screens

Програмний код деяких файлів з цих каталогів представлений у додатку Д. Відображення в додатках всіх файлів програмного коду будемо вважати недоцільним.

Для установки цього сервісу на хостингу має бути підтримка Node.js разом з іншими елементами стеку MERN.

4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1 Охорона праці та її актуальність в ІТ-сфері

Для підвищення ефективності системи управління охорони праці (СУОП) дуже важлива роль належить формуванню і розвитку інформаційної культури фахівців ІТ-технологій, яка впливає на удосконалення інформаційного контуру сучасних підприємств, дозволяє створювати надійні прогнози щодо стану умов праці, показників здоров'я та працездатності, виробничого травматизму і професійної захворюваності, визначати політику розвитку підприємств, установ та організацій на основі різноманітних стратегій охорони праці (інноваційні, маркетингові, інвестиційні, фінансові, технологічні, диверсифікаційні). Поряд з інформаційною культурою важливо використовувати в рамках СУОП «трикутник» її складових: правову, організаційну, управлінську.

В управлінні охороною праці потрібно реалізувати основні положення, окремі теоретико-методологічні підходи інформаційного менеджменту. Головну роль та відповідальність за стан СУОП мають нести фахівці служби охорони праці сучасного підприємства.

Сучасне суспільство називають постіндустріальним, постекономічним, інформаційним, оскільки йдеться про багатосторонні і кардинальні зміни у розвитку цивілізації.

На постіндустріальному етапі розвитку суспільства вирішальним фактором стає інформація. Її домінування ініціювала науково-технічна революція, яку ще іменують інформаційною, оскільки нею охоплена будь-яка інтелектуальна діяльність, починаючи з інформаційних образів штучного інтелекту у нових технологіях, економіки, і продовжуючи інформатизацією суспільства в умова світової глобалізації науки й освіти тощо.

Інформаційні технології розглядаються як потужний важіль економічного зростання України. Для цього необхідні значні стратегічні інвестиції у

комп’ютерну та комунікаційну інфраструктуру, програми досліджень і розробок, освітню галузь [26].

Під інформаційною культурою розуміють сукупність, складову НІТ (новітні інформаційні технології), технологічну, правову, психологічну, соціологічну та ергономічну підсистеми, що сприяють спрямованому впливу на протікання соціальних процесів у суспільстві, колективі і вихованню свідомого відношення людини до праці, виконання прав та обов’язків [27].

Поняття інформаційної культури виникло в процесі активізації дослідницької уваги до механізмів інформаційного обміну у зв’язку зі значним підвищення ролі інформації в соціокультурних процесах суспільства, яке розглядають як інформаційне суспільство знань, де в центрі знаходяться інформаційні технології.

Робота з інформацією та інформаційна культура в цілому є одним з найважливіших компонентів спроб компанії управляти змінами. Є три принципові причини, в силу яких сьогодні необхідно дбати про інформаційну культуру компанії.

По-перше, вона все більше і більше стає найважливішою частиною загальної організаційної (корпоративної) культури компанії. Все більше компаній розуміють необхідність перетворень, орієнтованих на задоволення очікувань споживача. Щоб сьогодні впливати на майбутнє, потрібно уявляти собі на що вона буде схожа. А для цього потрібно працювати з різноманітною діловою, професійною, технологічною, соціальною, ринковою та політичною інформацією.

По-друге, інформаційні технології роблять можливим створення в компаніях комп'ютерних мереж, за допомогою яких йде спілкування між менеджерами, але важливо знати, як люди використовують цю інформацію. Саме по собі створення такої мережі з усіма її робочими станціями і мультимедійними можливостями не гарантує того, що інформація буде використовуватися більш розумно і більш ефективно.

По-третє, для різних функціональних служб, підрозділів та робочих груп сучасних підприємств в сфері охорони праці інформаційна культура різна, а це означає відмінність методологічних підходів до процесів усвідомлення, збору, організації, обробки, поширення і використання інформації. Тому багато менеджерів погодяться з тим, що корпоративна інформаційна культура важлива для вироблення різних стратегій охорони праці та запровадження відповідних заходів з її вдосконалення.

Для деяких галузей, таких як розробка програмного забезпечення, інформаційна культура є необхідною умовою виживання, тому що зміна технологій в розробці програмного забезпечення відбувається кожні 6-8 місяців, а інвестиції на підготовку персоналу і освоєння нової технології величезні і у великих компаніях варіюються від 1,5 до 2 млрд. доларів на рік [27].

Аналіз свідчить, що інформатизація та інтеграція комунікаційного простору України сприяє різкому підвищенню інформаційної та професійної компетентності, ділової активності, стимулюванню конкуренції, створенню інноваційних підприємств та організацій, нових робочих місць, зниженню витрат на утримання управлінського апарату [25].

Поряд із задачами і здобутками окреслилися негативи використання інформаційних технологій:

1) надмірне інформаційне навантаження, суть якого полягає у тому, що кількість корисної інформації, яка надходить до мережі, перевищує психофізіологічні можливості її сприйняття людиною;

2) велика кількість інформації, яка сприймається, але не є корисною для фахівців в даний момент;

3) інформаційний голод, причиною якого є саме надлишок інформації, викликаний інформаційним перенавантаженням;

4) «інформоманія» як хвороба людини, яка робить останню знеособленою, залежною від перебування в інформаційному просторі і роботи з комп’ютером і чому вона віддає перевагу, уникаючи «живого» спілкування з людьми;

5) поява «кіберспільнот», що за своїми соціокультурними характеристиками набагато ближчі до представників інших культур у глобальному інформаційному просторі, ніж до своєї етнонаціональної спільноти чи решти населення, не охопленого Інтернетом;

6) індивідуалізм і дегуманізація способу життя «мешканців» Інтернету – відсутність готовності ділитися своїми знаннями.

Слід розуміти, що комп'ютерні технології, а особливо їх мережі істотно впливають на життєдіяльність людини, припускаючи глобалізацію і технократизацію суспільства. Але в ще більшій мірі цей вплив поширюється безпосередньо на центральну нервову систему, яка звикає працювати в дуже інтенсивному режимі багатозадачності, де вже переважають не тривалі логічні роздуми, а інтуїтивно-реактивні ланцюжки розумових формулювань у зв'язку з величезним обсягом оброблюваної щодня інформації, кількість якої зростає за експоненціальною швидкістю. Виникає припущення, що саме збільшення обсягу інформації та прискорення її обробки людиною може згубно вплинути на розвиток розумових здібностей людини.

Аналіз продуктивності розумової праці в найбільших за чисельністю фахівців IT-фірм показав, що велике значення з точки зору впливу на її результати має організаційна (корпоративна) культура. В цьому напрямі влаштовуються різні тімбілдинги, заходи, тренінги для розвитку персоналу. Також кожен керівник повинен добре розуміти свого співробітника, що саме для нього важливо, що його мотивує. Важливо відвести потрібну роль відповідному співробітнику, щоб він виконував ті завдання, які йому цікаві.

На подібних тренінгах в тому числі повинна розглядатися інформаційна культура працівника, в освоєнні, володінні, мотивуванні, застосуванні, перетворенні інформації із застосуванням сучасних інформаційних технологій і використанням цих умінь в навчанні з охорони праці і в подальшій професійній діяльності. Особливо вони будуть корисні, як доповнення до існуючих інструктажів з охорони праці на підприємстві, або як контроль психологічного стану та взаємовідносин у колективі.

Інформаційна культура як інтегративне утворення абсолютно не зводиться до розрізнених знань, вмінь та навичок роботи за комп'ютером. Вона передбачає інформативну спрямованість цілісної особистості, яка володіє мотивацією до застосування і засвоєння нових даних. Інформаційну культуру можна розглядати, як одну з граней особистісного розвитку промислових робітників. Це шлях універсалізації якостей людини.

Оволодіння інформаційною культурою сприяє реальному розумінню особистістю свого місця, себе і своєї ролі у виробничому колективі. Вона має сприяти формуванню нового покоління фахівців інформаційного суспільства, який повинен володіти наступними навичками: виділення релевантної, значущої інформації, диференціації вихідних даних, розробки інформативних критеріїв її оцінки інформації, вміло використовувати її в рамках СУОП.

Сьогодні продовжує діяти стратегічне правило «Можливості комп’ютерної техніки обмежені тільки нашими уявленнями» [25].

4.2 Шкідлива дія шуту та вібрації і захист від неї

Для запобігання шкідливої дії шуму і вібрації на організм працюючих проводяться технічні, організаційні і медикопрофілактичні заходи.

Одним з основних технічних заходів є зменшення при експлуатації та на стадії проектування, конструювання обладнання причин шуму і вібрації в самому джерелі утворення. Досягають цього завдяки використанню раціональної конструкції обладнання, заміни ударної дії деталей і машин коливальною, з’єднання елементів гнучкими зв’язками, врівноважування обертових частин механізмів, заміни металевих деталей пластмасовими, забезпечення різних власних частот коливань механізму з частотою збуджуючої сили. Аеродинамічний шум може бути зменшений застосуванням глушників та повітропроводів зі змінним перерізом. Шум трансформаторів (електромагнітний шум) знижується, якщо застосувати листи заліза як складових осердя трансформатора з малою магнітострикцією, серцевини.

Якщо неможливо ізолювати чи знизити шум і вібрацію самого джерела, потрібно:

– ізолювати джерело шуму або вібрації від навколишнього середовища засобами вібро- та звукоізоляції;

– раціонально планувати виробничі приміщення, що мають інтенсивні джерела шуму;

– збільшувати звукопоглинання внутрішніх поверхонь приміщення шляхом звукопоглинальних покриттів.

Принцип роботи звукоізоляційних екранів оснований на відбиванні звукової хвилі від різних екранів, стін, кожухів обладнання. Шумливі агрегати слід закривати звукоізоляційними кожухами з виводом назовні органів керування та контрольних приладів. Звукоізоляційні екрани виготовляють з металу, деревини, пластмаси та інших щільних матеріалів. Екрани зсередини покривають звукопоглинаючими матеріалами (скловатою пінополіуретаном), а по периметру кожуха – віброізоляційними підкладками (гума).

Вихідними даними для розрахунку параметрів необхідного екрану є спектр шуму, який необхідно ослабити, кількість екранів, через які проходить шум, їх площа, акустичні характеристики приміщення. За розрахованими значеннями необхідної звукової ізоляційної здатності екрану підбирається матеріал конструкції й екрану.

Принцип звукопоглинання оснований на явищі трансформації коливальної енергії звуку в теплову через втрати при терті. Найбільші втрати при терті мають пористі, волокнисті і перфоровані матеріали: поролон, пемзолітові і деревоволокнисті плити тощо. Енергія звукової хвилі переходить у теплову енергію, причому, ефект звукоізоляції збільшується з ростом частоти звукової хвилі. Звукопоглинаючими матеріалами оббивають стелі, стіни. Щоб одержати ефективну звукоізоляцію, найбільш доцільно застосовувати багатошарові огородження з м’якими прошарками (мінеральна вата).

Важливим технічним рішенням у забезпеченні виробничих умов є вдосконалення ручних віброінструментів. Для цього використовують

віброгасіння, змінюють ударний вузол, проводять балансування частин, що обертаються.

Послаблення локальної вібрації і передачі вібрації на підлогу і сидіння досягається засобами віброізоляції і вібропоглинання, застосуванням пружинних і гумових амортизаторів, прокладок тощо. Для обмеження поширення вібрацій через ґрунт, між фундаментом і ґрунтом залишають повітряні проміжки, які називаються акустичними розривами.

В останні роки знаходять застосування динамічні віброгасники, в яких створюються вібрації, що співпадають по частоті і протилежні по фазі вібрації машини, коливання якої необхідно зменшити.

До організаційних заходів по боротьбі з шумом та вібрацією на виробництві відносяться: впровадження раціонального режиму праці і відпочинку, обмеження часу роботи при використанні ручного інструменту, який створює вібрацію.

Глушники звуку застосовуються для зменшення шуму аеродинамічних установок (вентиляторів, пневмоінструментів, газотурбінних, дизельних, компресорних установок). Вони поділяються на активні, які поглинають звукову енергію, що на них поступила, і реактивні, які відбивають цю енергію. Потужні джерела шуму як правило розміщують в окремих приміщеннях, які віддалені від постійних робочих місць. Ізоляційні кабіни або екрани застосовують як екрани робочих місць для зменшення зовнішніх шумів.

Якщо не вдається зменшити рівень шуму і вібрації на робочому місці до нормативних значень та необхідно використовувати засоби індивідуального захисту: рукавиці, взуття, навушники, м’які шоломи, які зменшують рівень звукового тиску на 40-50 дБ.

У процесі виробництв, експлуатації і зберігання радіоелектронних засобів можуть виникати механічні і динамічні дії, що характеризуються широким діапазоном частот коливань, а також амплітудою, прискоренням і часом дії. Рівень механічних дій визначається умовами транспортування й експлуатації.

Необхідно розрізняти два види механічних дій: удари і вібрації. Удар виникає, коли апаратура отримує швидку зміну прискорення (піддаються удару входи кабелів, джгути, резистори, конденсатори, напівпровідникові діоди і тріоди, силові трансформатори, дроселі тощо). Вібрації – довготривалі знакозмінні процеси, які впливають на роботу апаратури при безпосередньому контакті з джерелом коливань або через повітряне середовище.

У результаті дії вібрацій і удару можуть бути наступні ушкодження апаратури: порушення герметичності через псування паяльних, зварних і клеєних швів і появи тріщин у метало-скляних спаях; повне руйнування корпусів або окремих їх частин через механічний резонанс або циклічну втому; обривання монтажних зв’язків, відшарування багатошарових друкованих плат, руйнування підставок; вихід з ладу електричних контактів; модуляція розмірів хвилеводних трактів; коаксіальних кабелів, конденсаторів змінної ємності, коливальних контурів, електровакуумних приладів, зміщення положення органів настроювання і управління.

Під впливом вібрацій може статись зміна параметрів напівпровідникових приладів, вольт амперних характеристик діодів, транзисторів. Все це призводить до руйнування конструкцій за рахунок явищ втоми. Радіоелектронна апаратура (РЕА) повинна мати віброміцність, вібростійкість, ударостійкість.

Захист РЕА здійснюється наступними групами методів:

– зменшується інтенсивність джерел вібрації шляхом балансування, зменшення зазорів, віброізоляції джерела вібрацій;

– зменшується величина дій, що передається апаратом шляхом віброізоляції, демпфірування, виключення резонансів, активного віброзахисту за допомогою ексцентриків, маятників, гіроскопів;

– використання найбільш добротні і жорсткі компоненти і вузли;

– застосовуються амортизатори.

Захист часом, захист віддалю, усунення джерела тепловиділення, теплоізоляція, охолодження гарячої поверхні, забезпечення тепловіддачі тіла людини та індивідуальні засоби захисту. Захист часом передбачає обмеження

часу перебування робітника в зоні дії інфрачервоного випромінювання. Потужність випромінювання можна знизити за рахунок конструкторських і технологічних рішень (зміною нагрівання виробів у нагрівальних пічках індукційним нагріванням та ін.) і за рахунок покриття поверхні, яка нагрівається, тепло ізолювальним матеріалом.

Якщо теплоізоляція неможлива, тоді захист від прямої дії інфрачервоного випромінювання здійснюється екрануванням. Екрани можуть бути прозорими, напівпрозорими і непрозорими. У свою чергу вони поділяються на тепловідбивальні, тепловідвідні та теплопоглинальні; стаціонарні і нестаціонарні.

Застосовують також прозору водяну завісу у вигляді суцільної тонкої водяної плівки. Вода є активним поглиначем інфрачервоного випромінювання.

Перегрівання людини попереджують раціональним режимом пиття, режимом праці та гідро процедурами. Спецодяг виготовляється з незаймистого, стійкого до інфрачервоного випромінювання, м’якого і повітронепроникного матеріалу (тканина з металевим покриттям відбиває 90% інфрачервоного випромінювання). Для захисту очей застосовують світлофільтри зі спеціального жовто-зеленого або синього скла.

Першочергові заходи – це конструкторські і технологічні рішення, які виключають генерацію або понижують інтенсивність випромінювання. Спеціальні засоби захисту (екранування джерел випромінювання, фарбування стін у світлі кольори) попереджують розповсюдження і знижують інтенсивність цих випромінювань у виробничих приміщеннях. Очі захищають окулярами або щитками зі склом – світлофільтром. Для захисту шкіри використовують мазі з речовинами – світлофільтрами для цих променів (салол, саліцилово-метиловий ефір та ін.), а також спецодяг з бавовняних тканин і грубововняного сукна. Руки захищають рукавицями.

ВИСНОВКИ

Таким чином, під час виконання кваліфікаційної роботи бакалавра було побудовано прототип програмної платформи, котрий дасть можливість побудувати користувачеві власну соціальну мережу. Кастомізація платформи може бути виконана шляхом використання власних зображень, логотипів, таблиць стилів. Тобто вимагає втручання в програмний код та структуру каталогів проєкту.

Добавлення функції налаштування цієї мережі через інтерфейс користувача виходить за межі даної кваліфікаційної роботи, оскільки вимагатиме створення системи керування контентом – CMS (Content Management System).

В ході виконання кваліфікаційної роботи бакалавра було виконано усі завдання, а саме:

1. Здійснено огляд предметної області створення соціальних мереж.
2. Здійснено аналіз основних наявних платформ для створення соціальних мереж.
3. Виконано огляд програмних архітектур, які використовуються в сучасних соціальних мережах.
4. Розроблено прототип програмної платформи для створення соціальних мереж.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Al-Deen, H. S. N., & Hendricks, J. A. (2011). Social media: usage and impact. Lexington books.

2. Bell, G. (2009). Building social Web applications: Establishing community at the heart of your site. " O'Reilly Media, Inc.".

3. Dahlberg, L. (2001). Computer-mediated communication and the public sphere: A critical analysis. Journal of Computer-mediated communication, 7(1), JCMC714.

4. Hasell, A., & Chinn, S. (2023). The Political Influence of Lifestyle Influencers? Examining the Relationship Between Aspirational Social Media Use and Anti-Expert Attitudes and Beliefs. Social Media+ Society, 9(4), 20563051231211945.

5. Facebook Engineering. [Електронний ресурс]. – Режим доступу: URL: https://engineering.fb.com/, вільний. – Загл. з екрану.

6. Twitter Engineering. [Електронний ресурс]. – Режим доступу: URL: https://blog.twitter.com/engineering/en_us.html, вільний. – Загл. з екрану.

7. Kleppmann, M. Designing Data-Intensive Applications. O`Reilly. 2014. 144 p. ISBN: 978-1-4493-7332-0 1-4493-7332-1.

8. Facebook's software architecture. [Електронний ресурс]. – Режим доступу: URL: http://muratbuffalo.blogspot.com/2014/10/facebooks-software-architecture.html, вільний. – Загл. з екрану.

9. Twitter System Architecture. [Електронний ресурс]. – Режим доступу: URL: https://interviewnoodle.com/twitter-system-architecture-8dafce16aec4. вільний. – Загл. з екрану.

10. A Brief History of Scaling LinkedIn. [Електронний ресурс]. – Режим доступу: URL: https://engineering.linkedin.com/architecture/brief-history-scaling-linkedin. вільний. – Загл. з екрану.

11. From Monolith to Multicloud Micro-Services: Inside Snap’s Service Mesh. – Режим доступу: URL: https://eng.snap.com/monolith-to-multicloud-microservices-snap-service-mesh. вільний. – Загл. з екрану.

12. Thelwall, M. (2009). Social network sites: Users and uses. In: M. Zelkowitz (Ed.), Advances in Computers 76. Amsterdam: Elsevier (pp. 19-73). – Режим доступу: URL: http://www.scit.wlv.ac.uk/~cm1993/papers/SocialNetworkSitesUsersUses.pdf. вільний.

13. Chandan, A.L. (2015). Technical Survey on Social Networking Sites-Analytical Methods. Journal of emerging technologies and innovative research. Journal of emerging technologies and innovative research.

14. A survey on social network sites' functional features. [Електронний ресурс]. – Режим доступу: URL: https://www.researchgate.net/publication/230859339_A_survey_on_social_network_sites%27_functional_features. вільний. – Загл. з екрану.

15. Thomas Paul, Antonino Famulari, Thorsten Strufe, A survey on decentralized Online Social Networks, Computer Networks, Volume 75, Part A, 2014, Pages 437-452, ISSN 1389-1286.

16. Social Media and News Fact Sheet. November 15, 2023. [Електронний ресурс]. – Режим доступу: URL: https://www.pewresearch.org/journalism/fact-sheet/social-media-and-news-fact-sheet/ вільний. – Загл. з екрану.

17. Claudia Diaz, Seda Gürses, Understanding the landscape of privacy technologies, Extended abstract of invited talk at Information Security Summit, 2012.

18. C. Lampe, N. Ellison, and C. Steinfield, “A Face(book) in the crowd: Social searching vs. social browsing,” in Proc. ACM CSCW, 2006, pp. 167–170.

19. X. Hu, T. H. S. Chu, V. C. M. Leung, E. C. . -H. Ngai, P. Kruchten and H. C. B. Chan, "A Survey on Mobile Social Networks: Applications, Platforms, System Architectures, and Future Research Directions," in IEEE Communications Surveys & Tutorials, vol. 17, no. 3, pp. 1557-1581, 2015.

20. Social Network Development: Types, Challenges, Technologies, Costs. [Електронний ресурс]. – Режим доступу: URL: https://medium.com/hackernoon/social-network-development-types-challenges-technologies-costs-1e185da3b2a9. вільний. – Загл. з екрану.

21. Learn more, do more. Marketing and Analytics Resources – Google Marketing Platform. [Електронний ресурс]. – Режим доступу: URL: https://marketingplatform.google.com/about/resources/ вільний. – Загл. з екрану.

22. The Demand For AngularJS Developers. [Електронний ресурс]. – Режим доступу: URL: https://icanbecreative.com/article/the-demand-for-angularjs-developers/ вільний. – Загл. з екрану.

23. Rendering Frameworks. [Електронний ресурс]. – Режим доступу: URL: https://2022.stateofjs.com/en-US/libraries/rendering-frameworks/ вільний. – Загл. з екрану.

24. GitHub - mrdoob/three.js: JavaScript 3D Library. [Електронний ресурс]. – Режим доступу: URL: https://github.com/mrdoob/three.js/ вільний. – Загл. з екрану.

25. Стручок, В. С., Стручок, О. С., & Мудра, Д. В. (2017). Навчальний посібник до написання розділу дипломного проекту та дипломної роботи ″Безпека в надзвичайних ситуаціях ″для студентів всіх спец. денної, заочної (дистанційної) та екстернатної форм навчання.

26. Стручок, В. С. (2022). Техноекологія та цивільна безпека. Частина «Цивільна безпека». Навчальний посібник.

27. Шконда В.В., Кальянов А.В., Давыдов П.Г. Феномен синергетики: наука – общество – образование: Монография / Ред. Шконда В.В. – Донецк: Норд-Пресс, 2009. – 156 с.

28. Жидецький, В. Ц., Джигирей, В. С., & Мельников, О. В. (2000). Основи охорони праці. Львів: Афіша, 350, 132-136.

29. Навакатіян О.О., Кальниш В.В., Стрюков С.М. Охорона праці користувачів комп’ютерних відеодисплейних терміналів. - К.:1997. - 400с.

ДОДАТКИ

Додаток А

Програмний код моделей даних

А.1 Файл Chat.js (колекція для повідомлень)

```
const { Schema, model } = require('mongoose')

const chatSchema = new Schema(
  {
    sender: {
      type: Schema.Types.ObjectId,
      ref: 'User',
    },
    receiver: {
      type: Schema.Types.ObjectId,
      ref: 'User',
    },
    body: {
      type: Object,
      required: true,
    },
  },
  { timestamps: true },
)

module.exports = model('Chat', chatSchema)
```

А.2 Файл Comments.js (колекція для коментарів)

```
const { Schema, model } = require('mongoose')

const commentSchema = new Schema({
  post: {
    type: Schema.Types.ObjectId,
    ref: 'Post',
  },

  user: {
    type: Schema.Types.ObjectId,
    ref: 'User',
  },
  body: {
    image: String,
    text: {
      type: String,
      trim: true,
    },
```

```
  },

  likes: [{ type: Schema.Types.ObjectId, ref: 'User' }],
})

module.exports = model('Comment', commentSchema)
```

А.3 Файл FriendRequest.js (колекція запитів на дружбу)

```
const mongoose = require('mongoose')

const Schema = mongoose.Schema

const FriendRequestSchema = new Schema(
  {
    sender: {
      type: Schema.Types.ObjectId,
      ref: 'User',
    },
    receiver: {
      type: Schema.Types.ObjectId,
      ref: 'User',
    },
    isAccepted: {
      type: Boolean,
      default: false,
    },
  },
  { timestamps: true },
)

module.exports = mongoose.model('FriendRequest', FriendRequestSchema)
```

А.4 Файл Notification.js (колекція для сповіщень)

```
const { Schema, model } = require("mongoose")


const notificationSchema = new Schema({
    body: {
        type: String,
        required: true
    },

    user: {
        type: Schema.Types.ObjectId,
        ref: "User"
    }
```

```
}, { timestamps: true })


module.exports = model("Notification",notificationSchema)
```

A.5 Файл Post.js (колекція для дописів)

```
const { Schema, model } = require('mongoose')

const postSchema = new Schema(
  {
    user: {
      type: Schema.Types.ObjectId,
      ref: 'User',
    },
    content: {
      type: String,
      trim: true,
    },
    body: {
      feelings: {
        type: String,
        trim: true,
      },
      with: [
        {
          type: Schema.Types.ObjectId,
          ref: 'User',
        },
      ],
      at: {
        type: String,
        trim: true,
      },
      date: String,
    },
    image: String,
    isProfilePost: {
      type: Boolean,
      default: false,
    },

    profilePostData: {
      coverImage: String,
      profileImage: String,
    },

    privacy: {
```

```
      type: String,
      enum: ['Only me', 'Public'],
      default: 'Public',
    },

    likes: [
      {
        type: Schema.Types.ObjectId,
        ref: 'User',
      },
    ],

    hearts: [
      {
        type: Schema.Types.ObjectId,
        ref: 'User',
      },
    ],
  },
  { timestamps: true },
)

module.exports = model('Post', postSchema)
```

A.6 Файл User.js (колекція для користувачів)

```
const mongoose = require('mongoose')

const Schema = mongoose.Schema

const UserSchema = new Schema(
  {
    name: {
      type: String,
      trim: true,
      required: true,
    },
    email: {
      type: String,
      trim: true,
      unique: true,
      required: true,
    },
    password: {
      type: String,
      required: true,
    },

    profile_pic: {
```

```
      type: String,
      default: '',
    },

    cover_image: {
      type: String,
      default: '',
    },

    bio: {
      type: String,
      default: '',
      trim: true,
    },
    active: {
      type: Boolean,
      default: false,
    },
    socketId: {
      type: String,
      default: '',
    },
    jwtToken: [String],

    location: {
      type: Object,
    },
    education: {
      type: String,
      trim: true,
    },

    friends: [{ type: Schema.Types.ObjectId, ref: 'User' }],
  },

  { timestamps: true },
)

UserSchema.index({ name: 'text', email: 'text'})
module.exports = mongoose.model('User', UserSchema)
```

Додаток Б

Програмний код авторизації

Б.1 Файл Signup.js (створення облікового запису)

```
const User = require('../../models/User')
const jwt = require('jsonwebtoken')
const bcrypt = require('bcrypt')
const ValidateEmail = require('../../utils/ValidateEmail')
const {JWT_SECRET,JWT_EXP} = require("../../config")

module.exports = async (req, res) => {
  const { name, email, password } = req.body
  let error = {}
  if (!name || name.trim().length === 0) {
    error.name = 'name field must be required'
  }
  if (!ValidateEmail(email)) {
    error.email = 'email address should be valid '
  }
  if (!email || email.trim().length === 0) {
    error.email = 'email field must be required'
  }
  if (!password || password.trim().length === 0) {
    error.password = 'password must be required'
  }

  if (Object.keys(error).length) {
    return res.status(422).json({ error })
  }

  try {
    const user = await User.findOne({ email })
    if (user) res.status(400).json({ error: 'email already exists' })

    const hashPassword = await bcrypt.hash(password, 8)
    const registerUser = new User({
      name,
      email,
      password: hashPassword,
    })

    const saveUser = await registerUser.save()
    const token = jwt.sign({ userId: saveUser.id }, JWT_SECRET, {
      expiresIn: JWT_EXP,
    })

    saveUser.active = true
    await saveUser.save()
```

```
    res.status(201).json({
      message: `Account created for ${email}`,
      data: {
        token,
      },
    })
  } catch (err) {
    console.log(err)
    return res.status(500).json({error:"Something went wrong"})
  }
}
```

Б.2 Файл Login.js (вхід користувача)

```
const User = require('../../models/User')
const jwt = require('jsonwebtoken')
const bcrypt = require('bcrypt')
const ValidateEmail = require('../../utils/ValidateEmail')
const {JWT_SECRET,JWT_EXP} = require("../../config")

module.exports = async (req, res) => {
  const { email, password } = req.body
  let error = {}

  if (!ValidateEmail(email)) {
    error.email = 'email address should be valid '
  }
  if (!email || email.trim().length === 0) {
    error.email = 'email field must be required'
  }

  if (!password || password.trim().length === 0) {
    error.password = 'password must be required'
  }

  if (Object.keys(error).length) {
    return res.status(422).json({ error })
  }

  try {
    const user = await User.findOne({ email })
    if (!user) {
      return res.status(400).json({ error: 'email not found' })
    }

    const verifyPassword = await bcrypt.compare(password, user.password)
    if (!verifyPassword) {
      return res.status(400).json({ error: 'password is incorrect' })
    }
```

```
    const token = jwt.sign({ userId: user.id }, JWT_SECRET, {
      expiresIn: JWT_EXP,
    })

    user.active = true
    await user.save()

    return res.status(201).json({
      message: 'login successfully',
      data: {
        token,
      }
    })
  } catch (err) {
    console.log(err)
    return res.status(500).json({error:"Something went wrong"})
  }
}
```

Б.3 Файл Logout.js (вихід користувача)

```
const User = require('../../models/User')

module.exports = async (req, res) => {
  try {
    const user = await User.findById(req.userId)
    if (user) {
      user.active = false
      await user.save()

      const accountData = {
        id: user.id,
        name: user.name,
        email: user.email,
        profile_pic: user.profile_pic,
      }

      res
        .status(201)
        .json({ message: 'logout successfully', account: accountData })
    } else {
      res.status(404).json({ error: 'user not found' })
    }
  } catch (err) {
    console.log(err)
    return res.status(500).json({error:"Something went wrong"})
  }
}
```

Додаток В.

Програмний код створення контенту

В.1 Файл postAction.js (створення допису)

```
const User = require('../../models/User')
const Post = require('../../models/Post')
const FilterPostData = require('../../utils/FilterPostData')
const sendDataToFriends = require("../../utils/socket/SendDataToFriend")
exports.createPost = async (req, res) => {
  let { content, privacy, image, body } = req.body

  if (!content && content.trim().length === 0 && !image) {
    return res.status(422).json({
      error:
        'Post Image or Write Some Content  to Post. Can`t upload empty post',
    })
  }
  try {
    const createPost = new Post({
      image,
      privacy,
      content,
      user: req.userId,
      isProfilePost: false,
    })

    const savePost = await createPost.save()

    if (Object.keys(body).length) {
      savePost.body = {
        feelings: body.feelings,
        at: body.at,
        date: body.date,
        with: body.with.map((user) => user),
      }

      await savePost.save()
    }
    const post = await Post.findById(savePost.id)
      .populate('user')
      .populate({ path: 'body.with', select: '_id name' })
    const postData = FilterPostData(post)

    res
      .status(201)
      .json({ message: 'post created successfully', post: postData })
```

```
    let dataToSend = {
      req, key: "new-post", data: postData,
      notif_body:`${post.user.name} has created new post`
    }
    await sendDataToFriends(dataToSend)

  } catch (err) {
    console.log(err)
    return res.status(500).json({error:"Something went wrong"})
  }
}

exports.likeDislikePost = async (req, res) => {
  try {
    const post = await Post.findById(req.params.postId)
      .populate('user')
      .populate({ path: 'body.with', select: '_id name' })
    if (!post) {

      return res.status(404).json({ error: 'post not found' })
    }

    let postData

    const index = post.likes.indexOf(req.userId)
    if (index !== -1) {
      post.likes.splice(index, 1)
      await post.save()
      postData = FilterPostData(post)
      res.status(200).json({ message: 'removed likes', post: postData })
      await sendDataToFriends({ req, key: "post-like-change", data: postData })
      return;

    }

    post.likes.push(req.userId)
    await post.save()
    postData = FilterPostData(post)
    res.status(200).json({ message: 'add like', post: postData })
    await sendDataToFriends({ req, key: "post-like-change", data: postData })
  } catch (err) {
    console.log(err)
    return res.status(500).json({error:"Something went wrong"})
  }
}
```

В.2 Файл FetchPosts.js (отримання матеріалу)

```
const Post = require('../../models/Post')
const Comment = require('../../models/Comment')
const FilterPostData = require('../../utils/FilterPostData')

exports.fetchPostById = async (req, res) => {
  try {
    const post = await Post.findById(req.params.postId)
      .populate('user')
      .populate({ path: 'body.with'})

    let postData = FilterPostData(post)

    res.status(200).json({ post: postData })
  } catch (err) {
    console.log(err)
    return res.status(500).json({error:"Something went wrong"})
  }
}

exports.fetchAllPosts = async (req, res) => {
  let page = parseInt(req.query.page || 0)
  let limit = 3

  try {
    const posts = await Post.find()
      .sort({ createdAt: -1 })
      .limit(limit)
      .skip(page * limit)
      .populate('user')
      .populate({ path: 'body.with' })

    let postsData = posts.map((post) => FilterPostData(post))

    const totalCount = await Post.estimatedDocumentCount().exec()
    const paginationData = {
      currentPage:page,
      totalPage:Math.ceil(totalCount/limit),
      totalPost:totalCount
    }
    res.status(200).json({ posts: postsData,pagination:paginationData })
  } catch (err) {
    console.log(err)
    return res.status(500).json({error:"Something went wrong"})
  }
}
```

В.3 Файл Comment.js (створення коментаря)

```
const User = require('../../models/User')
const Post = require('../../models/Post')
const Comment = require('../../models/Comment')
const FilterCommentData = require('../../utils/FilterCommentData')
const sendDataToFriends = require('../../utils/socket/SendDataToFriend')

exports.createComment = async (req, res) => {
  const { text, image } = req.body
  if (!text || (text.trim().length === 0 && !image)) {
    return res.status(422).json({ error: 'enter something or comment image' })
  }
  try {
    const post = await Post.findById(req.params.postId)
    if (!post) {
      return res.status(404).json({ error: 'post not found' })
    }

    let body = {}
    if (image) {
      body.image = image
    }

    if (text) {
      body.text = text
    }

    const createComment = new Comment({
      user: req.userId,
      post: req.params.postId,
      body,
    })

    const saveComment = await createComment.save()
    const comment = await Comment.findById(saveComment.id).populate(
      'user',
      '_id name  profile_pic',
    )

    const filterComment = FilterCommentData(comment)
    res.status(201).json({
      message: 'commented on post successfully',
      comment: filterComment,
    })
    await sendDataToFriends({ req, key: 'post-comment', data: filterComment })
  } catch (err) {
    console.log(err)
    return res.status(500).json({error:"Something went wrong"})
```

```
  }
}

exports.fetchComments = async (req, res) => {
  let page = parseInt(req.query.page || 0)
  let limit = 3

  try {
    const comments = await Comment.find({ post: req.params.postId })
      .sort({ createdAt: -1 })
      .limit(limit)
      .skip(page * limit)
      .populate('user', '_id name profile_pic')
    const filterComments = comments.map((comment) => FilterCommentData(comment))
    const totalCount = await Comment.countDocuments({ post: req.params.postId })

    const paginationData = {
      currentPage: page,
      totalPage: Math.ceil(totalCount / limit),
      totalComments: totalCount,
    }
    res
      .status(200)
      .json({ comments: filterComments, pagination: paginationData })
  } catch (err) {
    console.log(err)
    return res.status(500).json({error:"Something went wrong"})
  }
}

exports.likeDislikeComment = async (req, res) => {
  try {
    const comment = await Comment.findById(req.params.commentId).populate(
      'user',
    )

    if (!comment) {
      return res.status(404).json({ error: 'comment not found' })
    }

    let commentData

    const index = comment.likes.indexOf(req.userId)
    if (index !== -1) {
      comment.likes.splice(index, 1)
      await comment.save()

      commentData = FilterCommentData(comment)
```

```
      res.status(200).json({ message: 'removed likes', comment: commentData })

      await sendDataToFriends({
        req,
        key: 'comment-like-change',
        data: commentData,
      })
      return
    }

    comment.likes.push(req.userId)
    await comment.save()
    commentData = FilterCommentData(comment)
    res.status(200).json({ message: 'add like', comment: commentData })
    await sendDataToFriends({
      req,
      key: 'comment-like-change',
      data: commentData,
    })
  } catch (err) {
    console.log(err)
    return res.status(500).json({error:"Something went wrong"})
  }
}
```

Додаток Г.

Програмний код реалізації дій користувача в профілі

Г.1 Файл UserAction.js (запити дружби, сповіщення)

```
const User = require('../../models/User')
const FriendRequest = require('../../models/FriendRequest')
const FilterUserData = require('../../utils/FilterUserData')
const Notification = require('../../models/Notification')
const CreateNotification = require('../../utils/CreateNotification')

exports.sendFriendRequest = async (req, res) => {
  try {
    const user = await User.findById(req.params.userId)
    if (!user) {
      return res.status(404).json({ error: 'User not found' })
    }

    if (req.userId == req.params.userId) {
      return res
        .status(400)
        .json({ error: 'You cannot send friend request to yourself' })
    }

    if (user.friends.includes(req.userId)) {
      return res.status(400).json({ error: 'Already Friends' })
    }

    const friendRequest = await FriendRequest.findOne({
      sender: req.userId,
      receiver: req.params.userId,
    })

    if (friendRequest) {
      return res.status(400).json({ error: 'Friend Request already send' })
    }

    const newFriendRequest = new FriendRequest({
      sender: req.userId,
      receiver: req.params.userId,
    })

    const save = await newFriendRequest.save()

    const friend = await FriendRequest.findById(save.id).populate('receiver')

    const chunkData = {
```

```
      id: friend.id,
      user: FilterUserData(friend.receiver),
    }

    res
      .status(200)
      .json({ message: 'Friend Request Sended', friend: chunkData })

    const sender = await FriendRequest.findById(save.id).populate('sender')
    let notification = await CreateNotification({
      user: req.params.userId,
      body: `${sender.sender.name} has send you friend request`,
    })
    const senderData = {
      id: sender.id,
      user: FilterUserData(sender.sender),
    }

    if (user.socketId) {
      req.io
        .to(user.socketId)
        .emit('friend-request-status', { sender: senderData })
      req.io.to(user.socketId).emit('Notification', { data: notification })
    }
  } catch (err) {
    console.log(err)
    return res.status(500).json({error:"Something went wrong"})
  }
}

exports.acceptFriendRequest = async (req, res) => {
  try {
    const friendsRequest = await FriendRequest.findById(req.params.requestId)
    if (!friendsRequest) {
      return res
        .status(404)
        .json({ error: 'Request already accepted or not sended yet' })
    }

    const sender = await User.findById(friendsRequest.sender)
    if (sender.friends.includes(friendsRequest.receiver)) {
      return res.status(400).json({ error: 'already in your friend lists' })
    }
    sender.friends.push(req.userId)
    await sender.save()

    const currentUser = await User.findById(req.userId)
    if (currentUser.friends.includes(friendsRequest.sender)) {
      return res.status(400).json({ error: 'already  friend ' })
```

```
    }
    currentUser.friends.push(friendsRequest.sender)
    await currentUser.save()

    const chunkData = FilterUserData(sender)

    await FriendRequest.deleteOne({ _id: req.params.requestId })
    res
      .status(200)
      .json({ message: 'Friend Request Accepted', user: chunkData })

    let notification = await CreateNotification({
      user: sender.id,
      body: `${currentUser.name} has accepted your friend request`,
    })
    if (sender.socketId) {
      let currentUserData = FilterUserData(currentUser)
      req.io.to(sender.socketId).emit('friend-request-accept-status', {
        user: currentUserData,
        request_id: req.params.requestId,
      })
      req.io.to(sender.socketId).emit('Notification', { data: notification })
    }
  } catch (err) {
    console.log(err)
    return res.status(500).json({error:"Something went wrong"})
  }
}

exports.cancelSendedFriendRequest = async (req, res) => {
  try {
    const friendsRequest = await FriendRequest.findById(
      req.params.requestId,
    ).populate('receiver')
    if (!friendsRequest) {
      return res
        .status(404)
        .json({ error: 'Request already cenceled or not sended yet' })
    }
    await FriendRequest.deleteOne({ _id: req.params.requestId })

    res.status(200).json({ message: 'Friend Request Canceled' })
    if (friendsRequest.receiver.socketId) {
      req.io
        .to(friendsRequest.receiver.socketId)
        .emit('sended-friend-request-cancel', {
          requestId: req.params.requestId,
        })
    }
```

```
  } catch (err) {
    console.log(err)
    return res.status(500).json({error:"Something went wrong"})
  }
}

exports.declineFriendRequest = async (req, res) => {
  try {
    const friendsRequest = await FriendRequest.findById(
      req.params.requestId,
    ).populate('sender')
    if (!friendsRequest) {
      return res
        .status(404)
        .json({ error: 'Request already declined or not sended yet' })
    }
    await FriendRequest.deleteOne({ _id: req.params.requestId })

    res.status(200).json({ message: 'Friend Request Declined' })
    if (friendsRequest.sender.socketId) {
      req.io
        .to(friendsRequest.sender.socketId)
        .emit('received-friend-request-decline', {
          requestId: req.params.requestId,
        })
    }
  } catch (err) {
    console.log(err)
    return res.status(500).json({error:"Something went wrong"})
  }
}

exports.updateProfilePic = async (req, res) => {
  const { profile_url } = req.body
  try {
    const user = await User.findById(req.userId)
    user.profile_pic = profile_url
    await user.save()

    const getUser = await User.findById(req.userId).populate('friends')
    const userData = FilterUserData(getUser)

    const friends = getUser.friends.map((friend) => {
      return {
        ...FilterUserData(friend),
      }
    })

    userData.friends = friends
```

```
    res.status(200).json({ message: 'profile image updated', user: userData })
  } catch (err) {
    console.log(err)
    return res.status(500).json({error:"Something went wrong"})
  }
}

exports.updateCoverPic = async (req, res) => {
  const { cover_url } = req.body
  try {
    const user = await User.findById(req.userId)
    user.cover_image = cover_url
    await user.save()

    const getUser = await User.findById(req.userId).populate('friends')
    const userData = FilterUserData(getUser)

    const friends = getUser.friends.map((friend) => {
      return {
        ...FilterUserData(friend),
      }
    })

    userData.friends = friends
    res.status(200).json({ message: 'profile image updated', user: userData })
  } catch (err) {
    console.log(err)
    return res.status(500).json({error:"Something went wrong"})
  }
}

exports.updateProfile = async (req, res) => {
  try {
    const user = await User.findById(req.userId)

    if (req.params.input === 'name') {
      user.name = req.body.name
    }
    if (req.params.input === 'email') {
      user.email = req.body.email
    }

    if (req.params.input === 'bio') {
      user.bio = req.body.bio
    }
    if (req.params.input === 'location') {
      user.location = req.body.location
    }
    if (req.params.input === 'education') {
```

```
      user.education = req.body.education
    }

    await user.save()
    res.status(200).json({ message: 'Updated Successfully' })
  } catch (err) {
    console.log(err)
    return res.status(500).json({error:"Something went wrong"})
  }
}

exports.clearNotification = async (req, res) => {
  try {
    await Notification.deleteMany({ user: req.userId })
    res.status(200).json({ message: 'Notification Cleared Successfully' })
  } catch (err) {
    console.log(err)
    return res.status(500).json({error:"Something went wrong"})
  }
}
```

Г.2 Файл FetchUsers.js (отримання інформації про користувача)

```
const User = require('../../models/User')
const Notification = require('../../models/Notification')
const FriendRequest = require('../../models/FriendRequest')
const FilterUserData = require('../../utils/FilterUserData')

exports.fetchUserById = async (req, res) => {
  try {
    const user = await User.findById(req.params.user_id).populate('friends')
    const userData = FilterUserData(user)

    res.status(200).json({ user: userData })
  } catch (err) {
    console.log(err)
    return res.status(500).json({error:"Something went wrong"})
  }
}

exports.fetchRecommandedUsers = async (req, res) => {
  try {
    const users = await User.find()
      .where('_id')
      .ne(req.userId)
      .populate('friends')

    const usersData = users.map((user) => {
      return FilterUserData(user)
```

```
    })
    res.status(200).json({ users: usersData })
  } catch (err) {
    console.log(err)
    return res.status(500).json({error:"Something went wrong"})
  }
}
exports.me = async (req, res) => {
  try {
    const user = await User.findById(req.userId).populate('friends')
    if (!user) {
      return res.status(404).json({ error: 'user not found' })
    }

    const userData = FilterUserData(user)

    const friends = user.friends.map((friend) => {
      return {
        ...FilterUserData(friend),
      }
    })

    userData.friends = friends
    const notifications = await Notification.find({ user: req.userId }).sort({
      createdAt: -1,
    })
    let notifData = notifications.map((notif) => {
      return {
        id: notif.id,
        body: notif.body,
        createdAt: notif.createdAt,
      }
    })

    res.status(200).json({ user: userData, notifications: notifData })
  } catch (err) {
    console.log(err)
    return res.status(500).json({error:"Something went wrong"})
  }
}

exports.fetchIncommingFriendRequest = async (req, res) => {
  try {
    const friends = await FriendRequest.find({
      $and: [{ isAccepted: false }, { receiver: req.userId }],
    }).populate('sender', '_id name profile_pic active')

    const friendsData = friends.map((friend) => {
      return {
```

```
        id: friend.id,
        user: FilterUserData(friend.sender),
      }
    })

    res.status(200).json({ friends: friendsData })
  } catch (err) {
    console.log(err)
    return res.status(500).json({error:"Something went wrong"})
  }
}

exports.fetchSendedFriendRequest = async (req, res) => {
  try {
    const friends = await FriendRequest.find({
      $and: [{ isAccepted: false }, { sender: req.userId }],
    }).populate('receiver')
    const friendsData = friends.map((friend) => {
      return {
        id: friend.id,
        user: FilterUserData(friend.receiver),
      }
    })

    res.status(200).json({ friends: friendsData })
  } catch (err) {
    console.log(err)
    return res.status(500).json({error:"Something went wrong"})
  }
}

exports.searchUsers = async (req, res) => {
  try {
    const users = await User.find({
      name: { $regex: req.query.name, $options: 'i' },
    }).populate("friends")

    const usersData = users.map((user) => FilterUserData(user))

    res.status(200).json({ users: usersData })
  } catch (err) {
    console.log(err)
    return res.status(500).json({error:"Something went wrong"})
  }
}
```

Г.3 Файл Chat.js (обмін повідомленнями)

```
const Chat = require('../../models/Chat')
```

```
const User = require('../../models/User')
const FilterUserData = require('../../utils/FilterUserData')

exports.sendMessageToFriend = async (req, res) => {
  const { text, image } = req.body

  if (!text && !image) {
    return res
      .status(422)
      .json({ error: 'Don`t send empty message type something' })
  }
  try {
    const friend = await User.findById(req.params.friendId)
    if (!friend) {
      return res.status(404).json({ error: 'Friend Not Found' })
    }

    const chat = new Chat({
      sender: req.userId,
      receiver: req.params.friendId,
      body: {
        text: text || '',
        image: image || '',
      },
    })

    const saveChat = await chat.save()

    const getChat = await Chat.findById(saveChat.id)
      .populate('sender')
      .populate('receiver')
    const chatdata = {
      id: saveChat.id,
      sender: FilterUserData(getChat.sender),
      receiver: FilterUserData(getChat.receiver),
      body: getChat.body,
      createdAt: getChat.createdAt,
    }
    res.status(201).json({ data: chatdata })
    if (getChat.receiver.socketId) {
      req.io
        .to(getChat.receiver.socketId)
        .emit('new-message', { data: chatdata })
    }
  } catch (err) {
    console.log(err)
    return res.status(500).json({error:"Something went wrong"})
  }
}
```

```
exports.getFriendMessages = async (req, res) => {
  try {
    const messages = await Chat.find({
      $or: [
        { sender: req.userId, receiver: req.params.friendId },
        { receiver: req.userId, sender: req.params.friendId },
      ],
    })
      .populate('sender')
      .populate('receiver')

    const messagesData = messages.map((message) => {
      return {
        id: message.id,
        sender: FilterUserData(message.sender),
        receiver: FilterUserData(message.receiver),
        body:message.body,
        createdAt: message.createdAt,

      }
    })

    res.status(200).json({ data: messagesData })
  } catch (err) {
    console.log(err)
    return res.status(500).json({error:"Something went wrong"})
  }
}
```

Додаток Д

Програний код фронт-енд частини проєкту

Д.1 Файл Comments.js (створення коментаря)

```
import {
    Avatar,
    Button,
    Divider,
    Paper,
    Typography,
    List,
    ListItem,
    ListItemText,
    ListItemAvatar,
  } from '@material-ui/core'
  import React, { useContext } from 'react'
  import AvartarText from '../UI/AvartarText'
  import { FontAwesomeIcon } from '@fortawesome/react-fontawesome'
  import { faThumbsUp as filledLike } from '@fortawesome/free-solid-svg-icons'
  import { faThumbsUp } from '@fortawesome/free-regular-svg-icons'
  import { likeDislikeComment } from '../../services/PostServices'
  import { PostContext, UserContext, UIContext } from '../../App'
  function Comment({ comment }) {
    const { postDispatch } = useContext(PostContext)
    const { userState } = useContext(UserContext)
    const { uiDispatch, uiState } = useContext(UIContext)

    function handleLikeComment() {
      likeDislikeComment(comment.id).then((res) => {
        if (res.data) {
          postDispatch({ type: 'LIKE_UNLIKE_COMMENT', payload: res.data.comment })
          uiDispatch({
            type: 'SET_MESSAGE',
            payload: { color: 'success', text: res.data.message, display: true },
          })
        }
      })
    }

    function isLiked() {
      return comment.likes.includes(userState.currentUser.id)
    }

    const listItems = (
      <ListItem alignItems="flex-start">
        <ListItemAvatar>
```

```
      {comment.user.profile_pic ? (
        <Avatar>
          <img
            src={comment.user.profile_pic}
            style={{ width: '100%', height: '100%' }}
            alt={comment.user.name}
          />
        </Avatar>
      ) : (
        <AvartarText
          text={comment.user.name}
          bg={comment.user.active ? 'seagreen' : 'tomato'}
        />
      )}
    </ListItemAvatar>
    <ListItemText
      primary={
        <Typography style={{ color: uiState.darkMode && '#fff' }}>
          {comment.user.name}
        </Typography>
      }
      secondary={
        <>
          {comment.body.text && comment.body.text}

          {comment.body.image && (
            <Paper elevation={0} style={{ padding: '8px' }}>
              <Avatar
                variant="square"
                style={{ width: '40%', height: '100%' }}
              >
                <img
                  src={comment.body.image}
                  style={{ width: '100%', height: '100%' }}
                  alt=""
                />
              </Avatar>
            </Paper>
          )}
        </>
      }
    />
  </ListItem>
)
return (
  <div
    style={{ marginTop: '16px', marginBottom: !uiState.mdScreen && '50px' }}
  >
    <List>
```

```
          {listItems}

          <div
            style={{
              display: 'flex',
              alignItems: 'center',
              justifyContent: 'flex-end',
            }}
          >
            <Button
              onClick={handleLikeComment}
              size="small"
              color="primary"
              startIcon={
                isLiked() ? (
                  <FontAwesomeIcon icon={filledLike} size="sm" />
                ) : (
                  <FontAwesomeIcon icon={faThumbsUp} size="sm" />
                )
              }
            >
              ({comment.likes.length})
            </Button>
          </div>
          <Divider variant="inset" component="li" />
        </List>
      </div>
    )
  }

  export default Comment
```

Д.2 Файл Friend.js (встановлення відношення дружби)

```
import React, { Fragment, useContext } from 'react'
import {
  Avatar,
  Typography,
  makeStyles,
  Card,
  CardActionArea,
  CardContent,
} from '@material-ui/core'
import { UIContext, UserContext } from '../../App'
import AvartarText from '../UI/AvartarText'

const useStyles = makeStyles((theme) => ({
  container: {
    paddingLeft: '6px',
```

```
    paddingTop: '6px',
    paddingBottom: '8px',
    '&:hover': { background: 'whitesmoke', cursor: 'pointer' },
  },
}))
function Friend({ user, children }) {
  const { userDispatch } = useContext(UserContext)
  const { uiState, uiDispatch } = useContext(UIContext)
  const classes = useStyles()

  return (
    <Fragment>
      <Card
        elevation={0}
        style={{
          width: '100%',
          backgroundColor: uiState.darkMode ? 'rgb(36,37,38)' :null,
        }}
      >
        <CardActionArea
          onClick={() => {
            userDispatch({ type: 'ADD_SELECTED_USER_PROFILE', payload: user })
            uiDispatch({ type: 'SET_DRAWER', payload: false })
          }}
        >
          <CardContent
            style={{
              display: 'flex',
              alignItems: 'center',
              justifyContent: 'flex-start',
            }}
          >
            {user.profile_pic ? (
              <Avatar style={{ width: '60px', height: '60px' }}>
                <img
                  src={user.profile_pic}
                  style={{ width: '100%', height: '100%' }}
                />
              </Avatar>
            ) : (
              <AvartarText
                text={user.name}
                bg={user.active ? 'seagreen' : ' tomato'}
              />
            )}

            <div
              style={{
                display: 'flex',
```

```
                flexDirection: 'column',
                marginLeft: '8px',
              }}
            >
              <Typography
                variant="subtitle1"
                style={{ marginBottom: '4px', fontWeight: '600' }}
              >
                {user.name}
              </Typography>
              <Typography variant="body2">{7} mutual friends</Typography>
            </div>
          </CardContent>
        </CardActionArea>

        {children}
      </Card>
    </Fragment>
  )
}

export default Friend
```

Д.3 Файл SearchFriends.js (пошук друзів)

```
import React, { useState } from 'react'
import {
  Avatar,
  Button,
  Dialog,
  DialogActions,
  DialogContent,
  IconButton,
  List,
  ListItemIcon,
  ListItemText,
  TextField,
  Typography,
  ListItem,
  CircularProgress,
} from '@material-ui/core'
import { Link } from 'react-router-dom'
import useSearchFriends from '../../hooks/useSearchFriends'
import { Search } from '@material-ui/icons'
import AvartarText from '../UI/AvartarText'

function SearchFriends() {
  const [open, setOpen] = useState(null)
  const [name, setName] = useState('')
```

```
const { searchFriends, friends, loading } = useSearchFriends()

const handleSearch = () => {
  searchFriends(name)
}
const handleClose = () => {
  setOpen(false)
}

const handleOpen = () => {
  setOpen(true)
}

return (
  <div>
    <IconButton onClick={handleOpen}>
      <Search />
    </IconButton>
    <Dialog fullWidth maxWidth="sm" open={open} onClose={handleClose}>
      <DialogContent>
        <TextField
          style={{ width: '100%' }}
          value={name}
          onChange={(e) => setName(e.target.value)}
          variant="outlined"
          placeholder="Enter Friends Name"
        />
        <Button
          style={{ width: '100%', marginTop: '16px' }}
          variant="contained"
          color="primary"
          onClick={handleSearch}
        >
          Search
        </Button>
        {friends.length ? (
          <Typography variant="h4" style={{ marginTop: '20px' }}>
            Search Friends ({friends.length})
          </Typography>
        ) : null}
        {loading ? (
          <div
            style={{
              display: 'flex',
              alignItems: 'center',
              justifyContent: 'center',
              marginTop: '20px',
            }}
          >
```

```
      <CircularProgress />
    </div>
  ) : (
    <List>
      {friends &&
        friends.map((user) => (
          <ListItem
            button
            onClick={handleClose}
            component={Link}
            to={`/profile/${user.id}`}
            key={user.id}
          >
            <ListItemIcon>
              {user.profile_pic ? (
                <Avatar
                  style={{
                    width: '60px',
                    height: '60px',
                  }}
                >
                  <img
                    src={user.profile_pic}
                    width="100%"
                    height="100%"
                    alt={user.name}
                  />
                </Avatar>
              ) : (
                <AvartarText
                  text={user.name}
                  bg={user.active ? 'seagreen' : 'tomato'}
                />
              )}
            </ListItemIcon>
            <ListItemText style={{ marginLeft: '8px' }}>
              <Typography
                style={{ fontSize: '17px', fontWeight: '700' }}
              >
                {user.name}
              </Typography>
              <Typography>{user.email}</Typography>
            </ListItemText>
          </ListItem>
        ))}
    </List>
  )}
</DialogContent>
<DialogActions>
```

```
          <Button onClick={handleClose}>Cancle</Button>
        </DialogActions>
      </Dialog>
    </div>
  )
}

export default SearchFriends
```

Д.4 Файл Auth.js (екран авторизації користувача)

```
import React, { useContext, useState } from 'react'
import RecentAccounts from '../components/Auth/RecentAccount/RecentAccounts'
import LoginForm from '../components/Auth/LoginForm'
import SignupForm from '../components/Auth/SignupForm'
import {
  Button,
  Container,
  Divider,
  Grid,
  Paper,
  Typography,
} from '@material-ui/core'
import { UIContext } from '../App'

const Auth = () => {
  const [toggleLoginForm, setToggleLoginForm] = useState(true)
  const { uiState } = useContext(UIContext)
  return (
    <div style={{ paddingBottom: '100px', minHeight: '100vh' }}>
      <Container>
        <Grid
          container
          justify="center"
          alignItems="flex-start"
          direction="column"
          style={{ paddingTop: '30px' }}
        >
          <Typography
            variant="h4"
            style={{
              fontWeight: 800,
              color: uiState.darkMode ? 'white' : 'black',
            }}
          >
            Facebook
          </Typography>
          <Typography
            style={{
```

```
        fontWeight: 800,
        color: uiState.darkMode ? 'white' : 'black',
      }}
      variant="h6"
    >
      Recents Login
    </Typography>
    <Typography
      style={{
        marginTop: '16px',
        fontWeight: 800,
        color: uiState.darkMode ? 'white' : 'black',
      }}
      variant="body2"
    >
      click your picture or add an account
    </Typography>
  </Grid>

  <Grid container spacing={3} style={{ marginTop: '20px' }}>
    <Grid item xs={12} sm={6} md={8}>
      <RecentAccounts />
    </Grid>
    <Grid item xs={12} sm={6} md={4}>
      <Paper
        elevation={8}
        style={{
          padding: '16px',
          display: 'flex',
          flexDirection: 'column',
        }}
      >
        {toggleLoginForm ? <LoginForm /> : <SignupForm />}

        <Divider />
        <Button
          onClick={() => setToggleLoginForm(!toggleLoginForm)}
          style={{
            marginTop: '32px',
            background: 'rgb(74,183,43)',
            color: '#fff',
          }}
        >
          {toggleLoginForm
            ? 'Create New Account'
            : ' Already have an Account'}
        </Button>
      </Paper>
    </Grid>
```

```
        </Grid>
      </Container>
    </div>
  )
}

export default Auth
```

Д.5 Файл Messenger.js (інтерфейс месенджера)

```
import React, { useContext, useEffect } from 'react'
import {
  Avatar,
  Container,
  Grid,
  Paper,
  Typography,
} from '@material-ui/core'
import { ChatContext, UIContext } from '../App'
import Messages from '../components/Chat/Messages'
import DrawerBar from '../components/Navbar/DrawerBar'

import Friends from '../components/Chat/Friends'
import MessageTextArea from '../components/Chat/MessageTextArea'
import FriendNotSelected from '../components/Chat/FriendNotSelected'
import AvartarText from '../components/UI/AvartarText'

function Messenger() {
  const { uiDispatch, uiState } = useContext(UIContext)
  const { chatState, chatDispatch } = useContext(ChatContext)
  useEffect(() => {
    uiDispatch({ type: 'SET_NAV_MENU', payload: true })
    uiDispatch({ type: 'SET_DRAWER', payload: false })

    return () => {
      uiDispatch({ type: 'SET_NAV_MENU', payload: false })
      uiDispatch({ type: 'SET_DRAWER', payload: false })
      chatDispatch({ type: 'SET_SELECTED_FRIEND', payload: null })
    }
  }, [])

  return (
    <div
      style={{
        paddingTop: '100px',
        paddingBottom: '40px',
        minHeight: '100vh',
      }}
    >
```

```
{!uiState.mdScreen && (
  <DrawerBar>
    <Friends />
  </DrawerBar>
)}
<Container>
  <Paper style={{ backgroundColor: uiState.darkMode && 'rgb(36,37,38)' }}>
    <Grid
      container
      justify="center"
      alignItems="flex-start"
      spacing={2}
      style={{ padding: '16px' }}
    >
      {uiState.mdScreen && (
        <Grid
          item
          md={4}
          xs={12}
          sm={12}
          style={{
            height: '80vh',
            overflowY: 'scroll',
            overflowX: 'hidden',
            scrollbarColor: !uiState.darkMode
              ? '#fff rgb(240,242,245)'
              : ' rgb(36,37,38) rgb(24,25,26)',
          }}
        >
          <Paper elevation={0}>
            <Friends />
          </Paper>
        </Grid>
      )}
      {chatState.selectedFriend ? (
        <Grid
          item
          md={8}
          xs={12}
          sm={12}
          style={{
            height: '80vh',
            display: 'flex',
            flexDirection: 'column',
            justifyContent: 'space-between',
            alignItems: 'center',
            margin: 'auto',
          }}
        >
```

```
<Paper
  elevation={0}
  style={{
    display: 'flex',
    alignItems: 'center',
    padding: '16px',
    width: '100%',
    position: 'sticky',
    top: 0,
    backgroundColor: uiState.darkMode && 'rgb(36,37,38)',
  }}
>
  {chatState.selectedFriend.profile_pic ? (
    <Avatar>
      <img
        src={chatState.selectedFriend.profile_pic}
        style={{ width: '100%', height: '100%' }}
      />
    </Avatar>
  ) : (
    <AvartarText
      text={chatState.selectedFriend.name}
      bg={
        chatState.selectedFriend.active ? 'seagreen' : 'tomato'
      }
    />
  )}
  <Typography style={{ marginLeft: '16px' }}>
    {chatState.selectedFriend.name}
  </Typography>
</Paper>

<Paper
  elevation={0}
  style={{
    padding: '16px',
    width: '100%',
    height: '60vh',
    overflowY: 'scroll',
    overflowX: 'hidden',
    scrollbarColor: !uiState.darkMode
      ? '#fff #fff'
      : ' rgb(36,37,38) rgb(36,37,38)',

    backgroundColor: uiState.darkMode && 'rgb(36,37,38)',
  }}
>
  <Messages />
</Paper>
```

```
                <div
                  style={{
                    position: 'sticky',
                    bottom: 0,
                    left: 0,
                    width: '100%',
                  }}
                >
                  <MessageTextArea />
                </div>
              </Grid>
            ) : (
              <FriendNotSelected />
            )}
          </Grid>
        </Paper>
      </Container>
    </div>
  )
}

export default Messenger
```