МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Сумський державний університет
Центр, заочної, дистанційної та вечірньої форм навчання
Кафедра інформаційних технологій

«До захисту допущено»
В.о. завідувача кафедри

_____ Світлана ВАЩЕНКО

____ _____ 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня магістр

зі спеціальності 122 «Комп'ютерні науки»

освітньо-наукової програми «Інформаційні технології проектування»

на тему: Інформаційна технологія підтримки взаємодії користувачів у веб-соціальній мережі

Здобувача (ки) групи ІТ.мз-21с. Сулейманова Фаріда Мехмана Огли
(шифр групи) (прізвище, ім'я, по батькові)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

_____ Сулейманов Фарід Мехман Огли
(підпис) (Ім'я та ПРІЗВИЩЕ здобувача)

Керівник доцент кафедри інформаційних технологій,
доцент, к.т.н. Анна НЕНЯ _____
(посада, науковий ступінь, вчене звання, Ім'я та ПРІЗВИЩЕ) (підпис)

Суми – 2023

Сумський державний університет

Факультет електроніки та інформаційних технологій
Кафедра інформаційних технологій
Спеціальність 122 «Комп'ютерні науки»
Освітньо-професійна програма «Інформаційні технології проектування»

ЗАТВЕРДЖУЮ
В.о. зав. кафедри ІТ

_____ С. М. Ващенко
«_____» __ 2023 р.

ЗАВДАННЯ

на кваліфікаційну роботу магістра студентові

Сулейманов Фарід Мехман Огли

(прізвище, ім'я, по батькові)

1 Тема проекту Інформаційна технологія підтримки взаємодії користувачів у веб-соціальній мережі
затверджена наказом по університету від «05» вересня 2023 р. № 0465-VI

2 Термін здачі студентом закінченого проекту «7» грудня 2023 р.

3 Вхідні дані до проекту результати обговорення теми з керівником

4 Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) аналіз предметної області, постановка задачі, методи дослідження, проектування інформаційної технології підтримки користувачів в соціальній мережі, проектування та розробка соцільної мережі

5 Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) актуальність проблеми, мета проетку, постановка задачі, функціональні вимоги до проекту, контекстна діаграма IDEF0, декомпозиція IDEF0, схема інформаційної технології, діаграма варіантів використання, фізична

модель даних, засоби реалізації, практична реалізація, висновки

6. Консультанти випускної роботи із зазначенням розділів, що їх стосуються:

Розділ	**Консультант**	**Підпис, дата**	
		Завдання видав	**Завдання прийняв**

Дата видачі завдання _____.

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів випускної проекту	Термін виконання етапів проекту	Примітка
1	Ініціювання	до 23.09.2023	
2	Планування	до 04.10.2023	
3	Реалізація	до 25.11.2023	
4	Завершення	до 01.12.2023	
5	Оформлення документації	до 07.12.2023	

Магістрант _____ Сулейманов Ф.М.О.

Керівник роботи _____ к.т.н., доц. Неня А.В.

РЕФЕРАТ

Тема кваліфікаційної роботи магістра «Інформаційна технологія підтримки взаємодії користувачів у веб-соціальній мережі».

Пояснювальна записка складається зі вступу, 4 розділів, висновків, списку використаних джерел та літератури, з 22 найменувань, та 4 додатків. Загальний обсяг роботи − 85 сторінок, у тому числі 58 сторінки основного тексту, 3 сторінки списку використаних джерел, 25 сторінок додатків.

Кваліфікаційну роботу магістра присвячено створенню інформаційної технології підтримки взаємодії користувачів у веб-соціальній мережі.

В першому розділі роботи було проведено огляд досліджень і публікацій з представленої теми, розглянуто методи реалізації технології підтримки користувачів та захисту інформації, визначено переваги та недоліки даних методів.

Другий розділ присвячено формулюванню мети та задачі створюваного проекту, в рамках другого розділу також було проведено аналіз методів які використовуються компаніями для підтримки взаємодії користувачів в веб-соціальній мережі , в процесі якого були визначені їх переваги та недоліки.

Третій розділ присвячено проектуванню, а саме виконанню структурно-функціонального моделювання, моделюванню варіантів використання, проектуванню моделі бази даних, та проектуванню інформаційної технології.

У четвертому розділи роботи було описано процес створення технології, демонструються варіанти його використання.

Результатом виконання роботи є створена інформаційна технологія підтримки взаємодії користувачів у веб-соціальній мережі.

Ключові слова: інформаційна технологія, захист даних, MERN, MySQL SocketIO, React.js, Social media.

ЗМІСТ

ВСТУП

Актуальність. В сучасному світі, коли web 2.0 та пов'язані з нею соціальні мережі принесли величезний вплив на повсякденне життя, нові способи навчання почали привертати увагу через постійні дебати щодо включення технологій web 2.0 в освітній процес. Підтримка постійного зв’язку з колегами по роботі, друзями та рідними людьми стала зручнішою, адже тепер просто необхідно мати при собі девайс або персональний комп’ютер (далі ПК). Вплив блогів, Wiki, X, Meta, Instagram RSS-каналів та інших пов'язаних з ними соціальних мережевих інструментів у процесі взаємодії дуже помітний. Обмін інформацією що супроводжується відкритістю і великою мірою добровільної співпраці - ось що спонукає до досліджень у цій галузі інформатики.

Мета. Розробка інформаційної технологія підтримки взаємодії користувачів у веб-соціальній мережі. Основною метою є створення технології яка надаватиме доступ до соціальної мережі та чату, а також буде виконувати процеси з обробкою даних.

Мета декомпозується на наступні задачі:

- виконання аналізу проблемної області та ознайомлення з актуальними публікаціями;
- аналіз критеріїв та показників соціальної мережі, що будуть використовуватись при моделюванні соціальної мережі;
- розробка методики та алгоритму взаємодії користувачів;
- проведення аналізу продуктів аналогів;
- проведення проектування системи;
- розробка програмного додатку, що використовує стек технологій та базу даних;
- проведення тестування програмного продукту.

Об'єкт дослідження. Процес взаємодії користувачів у веб-соціальній мережі.

Предмет дослідження. Інформаційна технологія підтримки взаємодії користувачів у веб-соціальній мережі.

Практична новизна. Розробка інформаційної технології для підтримки взаємодії користувачів в веб-соціальній мережі має практичні переваги з використанням технології React.js. Дана технологія дозволяє ефективно вирішити питання швидкості обробки даних завдяки віртуальному DOM та алгоритму конкістенції. Завдяки React.js є можливість реалізовувати складні компоненти за готовими алгоритмами. Використання технології Context API дозволяє керувати станом додатку, що надасть можливості будувати різні сценарії поведінки додатку. Таким чином, впровадження React.js з технологіями дозволить забезпечити високу ефективність, швидкість та легкість управління взаємодією користувачів.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд останніх досліджень і публікацій

Нове покоління смартфонів та ПК стало головним чинником значного буму на ринку соціальних додатків. Це зростання відбувається з початку двадцять першого століття, і сьогодні ми переживаємо епоху великих технологічних інновацій та впровадженням багатьох атрибутів, таких як використання камери для верифікації користувача, вбудованих у мобільні телефони та ПК. Ці тенденції стали фундаментальними для зростання популярності соціальних мереж, що призвело до формування різноманітних характеристик використання на сучасному ринку [1].

Пристрої портативних технологій стають все більш присутніми в реальності людей, які стикаються з ринком технологій, і займають дуже важливе місце в повсякденному житті цих споживачів технологій[2]. Особливо це стосується молоді, хоча люди старшого віку також користуються послугами, пропонованими мобільними та десктопними технологіями.

Розвиток і широке розповсюдження таких пристроїв суттєво вплинули на збільшення онлайн-активності, переважно через віртуальні соціальні мережі. Це сприяло створенню багатофункціонального середовища, що, в свою чергу, збільшило гіперзв'язність споживачів [3]. Можливість доступу до різних функцій через один і той самий продукт чи послугу підвищує рівень використання споживачами і, таким чином, тримає їх на зв'язку в реальному часі.

Таке розвинене технологічне середовище дозволило віртуальним соціальним мережам стати феноменом інтеграції людей і технологій. Популярність додатків, які надають доступ до мобільних соціальних мереж, є прикладом такої інтеграції. Популярність пропорційна прогресу технологічної конвергенції з багатофункціональністю, що приносить диверсифікацію послуг в

одному пристрої [4]. Разом з цим, збільшення частоти доступу до соціальних мереж стимулювало більш широке використання мобільних комунікаційних додатків, таких як Facebook та Instagram, якими користуються багато споживачів технологій [5].

Переваги та недоліки мобільних та десктопних додатків описано в таблиці нижче.

Таблиця 1.1 – Плюси та мінуси мобільних та десктопних додатків

Критерії	Мобільні додатки	Десктопні додатки
Переваги		
1. Портативність	Зручність в користуванні при подорожах та руху	Звичайно працюють на комп’ютерах та ноутбуках
2. Поглиблені можливості	Мають доступ до функцій смартфону (геолокація, камера)	Мають більше ресурсів для обробки та візуалізації даних.
3. Легка інсталяція	Завантаження та оновлення через магазин додатків.	Потребують менше часу для встановлення.
4. Цільова аудиторія	Ширший охоплення аудиторії (більше смартфонів в порівнянні з комп'ютерами).	Призначені для більш конкретної аудиторії.
Недоліки		
1. Обмежені можливості	Менші можливості для складних обчислень та великого обсягу даних.	Мають більше ресурсів для обробки великих

		наборів даних та складних операцій.

Продовження таблиці 1.1 – Плюси та мінуси мобільних та десктопних додатків

Недоліки		
2. Розмір дисплея	Екрани мобільних пристроїв менші, що може бути обмеженням для великих навчальних чи робочих проектів.	Більші екрани дозволяють комфортно працювати з більшим обсягом інформації.
3. Операційна система	Вимагають розробки для різних ОС (iOS, Android) та версій.	Для десктопних ОС (Windows, macOS, Linux) додатки можуть бути написані для конкретної платформи.
4. Витрати на розробку	Вимагає розробки окремих версій для різних платформ.	Можливість створити універсальний додаток для різних ОС.
5. Витрати на тестування	Вимагає великих зусиль на тестування на різних пристроях та ОС.	Тестування може бути менш складним через стандартизованість обладнання.

Джерело: побудовано автором

Враховуючи всі переваги та недоліки мобільних та десктопних додатків було вирішено надати перевагу саме десктопній версії додатку з можливостями

захисту особистої інформації, та покращеній технології обміну даними між клієнтом та сервером.

При розробці веб-соціальної мережі інновації в сфері оптимізації та обміну даних між користувачами мають велике значення, адже це основа будь якого веб-додатку. При детальному моделюванню та тестуванні технології взаємодії є вірогідність що створений додаток матиме велику популярність серед користувачів.

2 ПОСТАНОВКА ЗАДАЧІ

2.1 Мета та задачі дослідження

Мета: розробити інформаційну технологію підтримки взаємодії користувачів у веб-соціальній мережі.

Функціональні вимоги:

1. Аналіз даних користувачів: система повинна здійснювати аналіз даних при взаємодії користувачів у веб-соціальних мережах.
2. Можливість зіставлення різних аспектів технологій, таких як інтерфейс, функціональність, безпека та продуктивність.
3. Здатність визначати основні фактори, що впливають на популярність, такі як кількість користувачів, активність, та оцінки користувачів.
4. Визначення слабких сторін та модернізація технологій: Система повинна вирішувати слабкі сторони існуючих технологій та пропонувати можливі шляхи їх модернізації.
5. Інтеграція розробленої технології в web-додаток для демонстрації технології
6. Забезпечення можливості тестування та демонстрації розробленого додатку.

Нефункціональні вимоги:

1. Система повинна забезпечити швидкий та ефективний аналіз та обробку інформації для користувачів.
2. Забезпечення конфіденційності та захисту даних під час аналізу та роботи з інформацією про існуючі технології.
3. Система повинна бути готовою до масштабування для обробки великої кількості даних та інформації.

4. Забезпечення стабільної та надійної роботи системи під час виконання аналізу та розробки веб-додатку.
5. Можливість інтеграції створеної системи взаємодії користувачів з існуючими веб-орієнтованими системами.

Задачі досягнення мети:

- проаналізувати існуючі технології взаємодії користувачів;
- проаналізувати існуючі показники популярності існуючих веб-додатків;
- визначити слабкі сторони популярних технологій та визначити способи їх модернізації;
- розробити веб-додаток для демонстрації запропонованої технології.

Гіпотеза: інтеграція створеної системи взаємодії користувачів в веб-орієнтованій системі надасть можливість зберегти ресурси системи та запобігти перенавантаженню інших компонентів під час обробки даних.

2.2 Методи дослідження

Огляд публікацій на тематику популярності соціальних мереж показав, що багато відомих компаній використовували чи використовують дві основні технології: WebSocket та Socket.IO [6]. Такі компанії як Meta, X, Google все більше почали використовувати технології та бібліотеки для оптимізації власних додатків. Найбільш популярні соціальні мережі мають великий попит цільової аудиторії протягом багатьох років. Тому інтеграція даних технологій допомагає розробникам скоротити час на тестування працездатності додатку, та зосередитися на основній частині розробки [7].

WebSocket API − це передова технологія, яка дозволяє відкривати двосторонній інтерактивний сеанс зв'язку між браузером користувача і сервером. За допомогою цього API ви можете надсилати повідомлення на сервер і

отримувати відповіді на основі подій без необхідності опитування сервера для отримання відповіді.

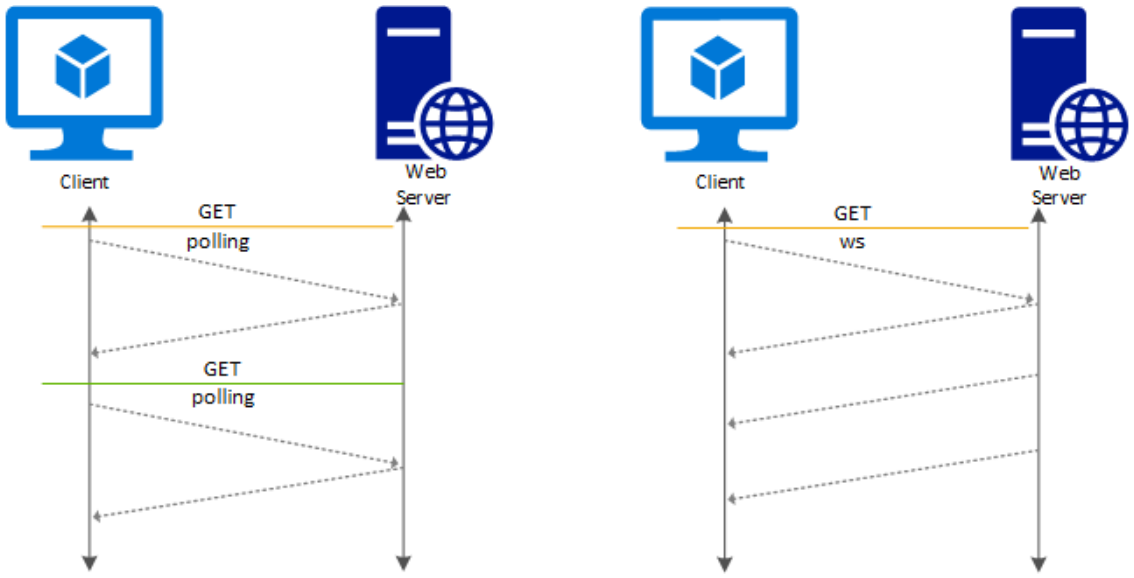


Рисунок 1.1 – Приклад роботи WebSocket.js

Джерело: [21]

Головні переваги WebSocket розкриваються в поєднанні з Node.js, який просто створений для обслуговування серверних додатків в режимі реального часу. Крім того, він заснований на JavaScript, так само, як і той, що використовується у веб-браузері, тому можна використовувати одну і ту ж мову як на фронтенді, так і на бекенді [8].

WebSocket - це протокол інтернет-зв'язку з цікавою особливістю: він забезпечує повнодуплексний канал через одне TCP-з'єднання.

За допомогою WebSockets клієнт і сервер можуть спілкуватися один з одним в режимі реального часу, як під час телефонної розмови: після підключення клієнт може отримувати дані від сервера без необхідності постійно оновлювати веб-сторінку. З іншого боку, сервер також зможе отримувати дані в реальному часі від клієнта в рамках того ж з'єднання.

Крім того, WebSockets керується подіями: як сервер, так і клієнт можуть реагувати на події та повідомлення. Даний протокол надає можливість прослуховувати подію з'єднання, запускати функцію, коли новий користувач підключається до сервера, надсилати повідомлення (по суті, подію) через сокет і багато іншого [9].

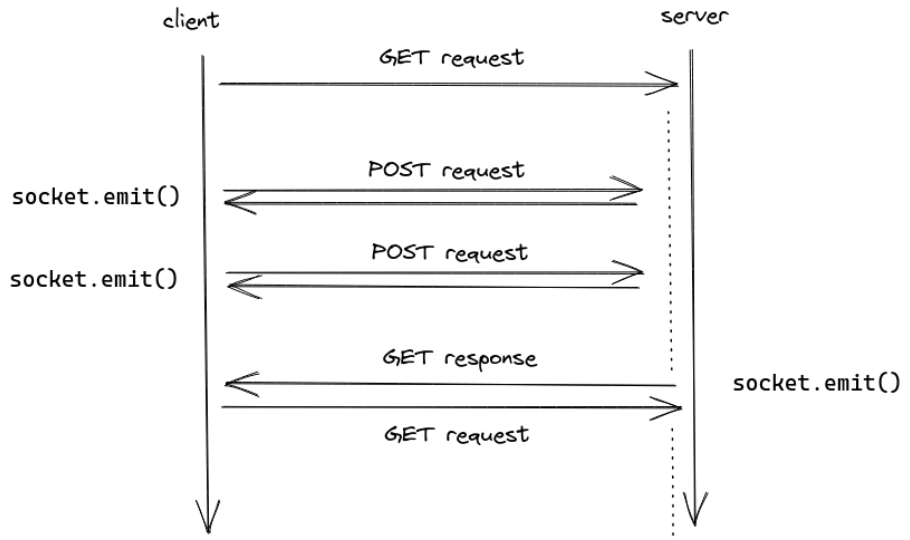


Рисунок 1.2 – Приклад виконання операції передачі даних з протоколом WebSocket.IO

Джерело: побудовано автором

Для більш детального аналізу технологій було створено порівняльну таблицю 1.2.

Таблиця 1.2 – Порівняльна характеристика технологій WebSocket та Socket.IO

Характеристика	WebSocket	Socket.IO
Базовий протокол	WebSocket Protocol (ws:// та wss://)	WebSocket Protocol (ws:// та wss://)

Тип передачі даних	Двосторонній обмін даними між клієнтом та сервером.	Двосторонній обмін даними між клієнтом та сервером.

Продовження таблиці 1.2 – Порівняльна характеристика технологій WebSocket та Socket.IO

Характеристика	WebSocket	Socket.IO
Кроссплатформність	+	+
Автоматичне перез’єднання	-	-
Транспортний протокол	Тільки WebSocket	WebSocket, AJAX, JSONP
Рівень надійності	Вище, оскільки це базовий протокол	Вище завдяки додатковим можливостям (автоматичне перез'єднання)
Тип додатків	Зазвичай використовується для прямих, простих з'єднань.	Ідеально підходить для складних, розподілених додатків, що вимагають багато сполучень.
Масштабованість	Залежить від реалізації, але може бути складно масштабувати на великі навантаження.	Допомагає у керуванні більшим навантаженням завдяки додатковим можливостям.

Легкість використання	Висока. Низький рівень абстракції.	Висока. Вищий рівень абстракції дозволяє швидше розробляти.
Діагностика помилок	Потребує додаткових зусиль для обробки та відстеження помилок.	Надає додаткові інструменти для обробки та відстеження помилок.

Джерело: побудовано автором

Аналіз даних в порівняльній таблиці чітко визначає WebSocket як технологію, що забезпечує двосторонній зв'язок у реальному часі між клієнтом і сервером. На відміну від неї, Socket.IO − це бібліотека, яка забезпечує рівень абстракції над WebSockets, що полегшує створення додатків у реальному часі. Для розробки технології підтримки взаємодії користувачів було обрано бібліотеку Socket.IO через інтеграцію з React.js, а також меншу затребуваність ресурсів для інтеграції даної бібліотеки в технологію.

3 ПРОЕКТУВАННЯ

3.1 Структурно-функціональне моделювання

IDEF0 розшифровується як Integration Definition for Process Modelling - методологія з відкритим доступом, яка використовується для моделювання бізнесу та його процесів, щоб їх можна було зрозуміти та вдосконалити. Це тип блок-схеми [10].

Діаграми IDEF0 зазвичай включають наступні компоненти:

Контекстна діаграма - сама верхня діаграма в моделі IDEF0.

Діаграма "батько/нащадок" − ієрархія декомпозиції IDEF0, що використовує зв'язки "батько/нащадок".

Дерева вузлів − деревоподібні структури вузлів з корінням у вибраному вузлі, що використовуються для представлення повної декомпозиції IDEF0 на одній діаграмі. [10]. Контекстна діаграма інформаційної технології підтримки взаємодії користувачів у веб-соціальній мережі в нотації IDEF0 представлена на рисунку 3.1.

Компонентами синтаксису IDEF0 є блоки та дуги. Блоки представляють дії функції та описують те, що відбувається всередині неї. Дуги представляють дані та об'єкти, пов'язані з функцією [11].

Інформаційна технологія за допомогою стеку технології MERN надає користувачу швидку обробку даних а також зберігає особисті cookie файли, щоб користувач зміг переглянути історію взаємодії з web-додатком [12].

Методологія IDEF0 підтримує декомпозицію діаграми до рівня деталізації, необхідного для повного розуміння процесу.

Діаграма IDEF0 декомпозується, щоб забезпечити більш детальний обмін інформації між клієнтом та сервером. Декомпозиція інформаційної технології представлена на рисунку 3.2.

Рисунок 3.1 – Контекстна діаграма інформаційної технології в нотації IDEF0

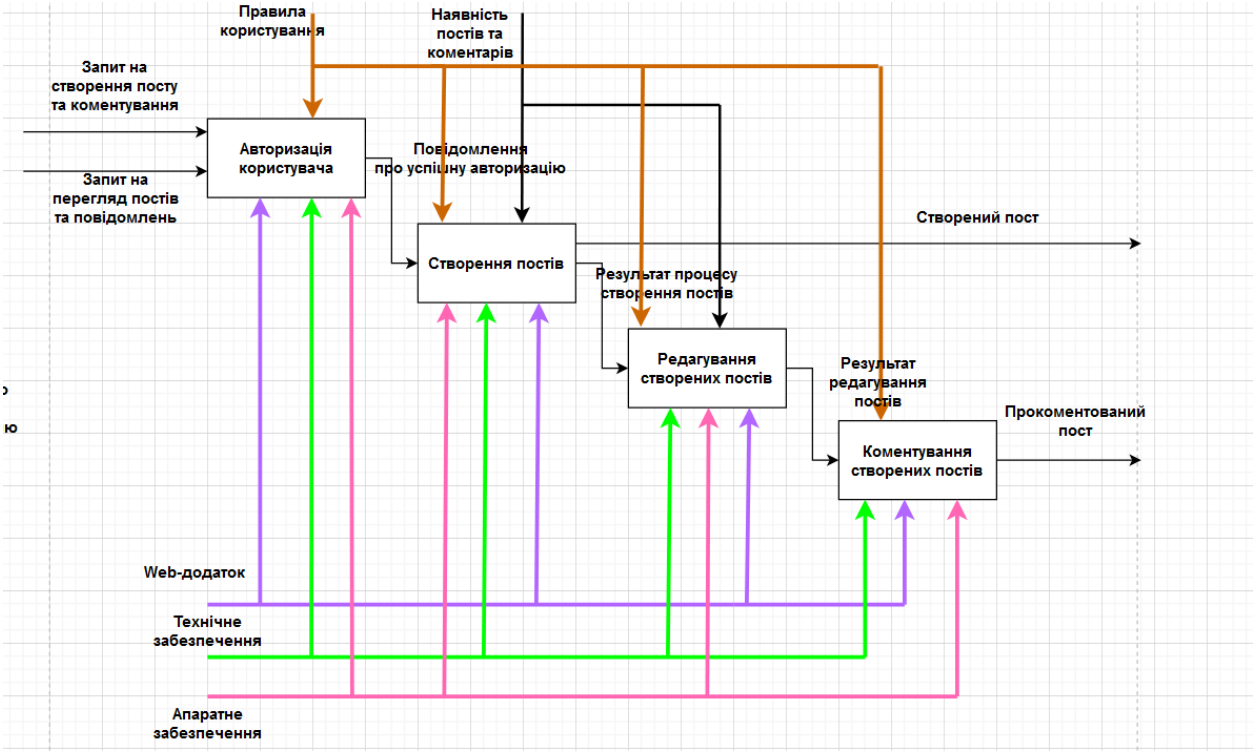


Рисунок 3.2 – Декомпозиція IDEF0-діаграми інформаційної технології

Діаграма IDEF0 інформаційної технології декомпонована на наступні блоки:

– авторизація користувача;

– створення постів;

– коментування створених постів

– листування з іншими користувачами

Для виконання функцій інформаційної технології необхідно створити веб-орієнтований додаток та за допомогою стеку технологій MERN створити клієнт-серверну архітектуру. Процес взаємодії в соціальній мережі полягає в створенні клієнт-серверної архітектури з використанням токену та хешуванням особистих даних для запобігання DDOS атак [13].

Результатом роботи всіх блоків є створена інформаційна технологія, яка надає користувачу можливість створювати власні пости, коментувати пости інших користувачів, змінювати особисту інформацію, а також вести листування з іншими користувачами в режимі реального часу.

3.2 Моделювання варіантів використання

Діаграма варіантів використання - це діаграма поведінки, яка візуалізує видимі взаємодії між акторами та системою, що розробляється [14]. Діаграма складається з системи, пов'язаних з нею варіантів використання та акторів і пов'язує їх один з одним:

Система: Що описується?

Актор: Хто використовує систему?

Варіант використання: Що роблять актори?

Актор - це спосіб представлення користувачів системи. Хоча вони представлені як люди, актори не обов'язково представляють людей - вони також можуть представляти нелюдські елементи, наприклад, систему електронної

пошти. Актор може бути як активним користувачем системи і запускати таким чином варіанти використання, так і пасивно використовуватися системою для реалізації варіантів використання. Коли актор визначений, він завжди повинен бути пов'язаний принаймні з одним варіантом використання. Актор представляє роль і тому може бути зайнятий кількома людьми або системами. Якщо у варіанті використання беруть участь кілька акторів, то це виражається через множинність. За замовчуванням це буде один. Щоб описати акторів якомога конкретніше, рекомендується також пов'язати їх з персонами.

Варіанти використання зазвичай представлені у вигляді овалів. Варіант використання представляє функціональність системи з точки зору користувача і описує цілі їх використання. Тут можна і потрібно ввести порядок дій - наприклад, заповнити форму, Ввести код авторизації і т.д. Якщо варіант використання деталізується лише через інший варіант використання, то він називається абстрактним і позначається таким чином у варіанті використання. Для варіанту використання також може бути введена кратність. Вона описує, як часто варіант використання може використовуватися [15].

Діаграма варіантів використання інформаційної технології підтримки взаємодії користувачів в представлена рисунку 3.3.

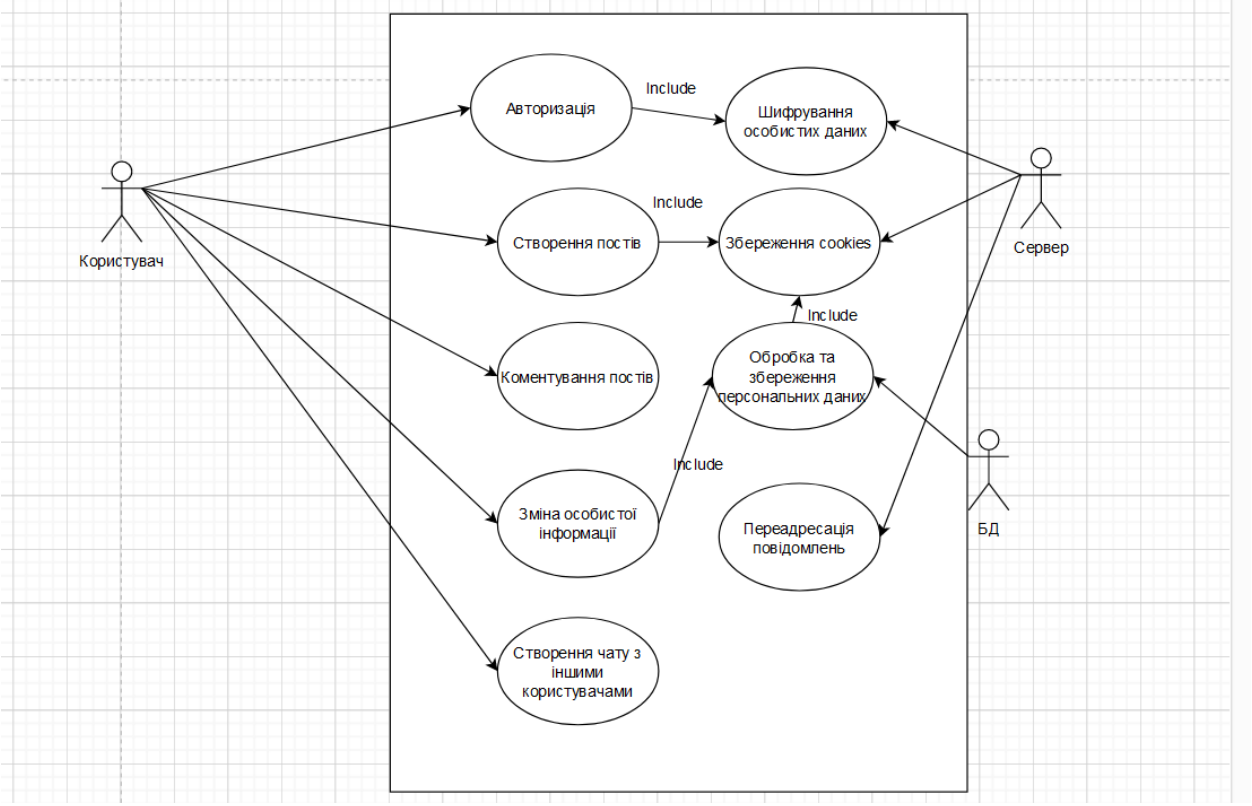


Рисунок 3.3 – Діаграми варіантів використання інформаційної технології підтримки взаємодії користувачів

Акторами в даний моделі виступають користувач, сервер та сховище даних

Діаграма з боку користувача містить наступні варіанти використання: авторизація, створення постів, коментування постів, зміна особистої інформації, створення чату з іншими користувачами.

З боку сервера діаграма містить варіанти використання «Шифрування особистих даних», «Збереження cookies» та «Переадресацію повідомлень». Дані варіанти використання пов’язані з актором «Користувач», адже при використанні технології клієнт напряму пов’язаний з сервером.

Діаграма з боку БД містить варіанти використання «Обробка та збереження особистих даних». Всі три актори взаємопов’язані між собою, адже клієнт опирається на дані, які були збережені на БД та оброблені за допомогою серверу.

3.3 Проектування моделі бази даних

Моделювання бази даних – це процес створення впорядкованого представлення компонентів даних, які він включає, за допомогою тексту та символів для зображення інформації та її потоків.

Це техніка для окреслення та візуалізації всіх різних місць, де програмне забезпечення або додаток зберігає дані, а також того, як ці різні джерела інформації будуть інтегруватися і перетікати один в одного.

Моделювання бази даних є важливим аспектом управління даними. Воно допомагає у визначенні інформаційних потреб для робочих процесів, надаючи візуальне представлення точок даних та їхньої поведінки [16].

Це дозволяє архітекторам системи визначити і зрозуміти, як дані будуть управлятися, модифікуватися, переглядатися і розподілятися по всій організації. Однією з видів моделювання даних є процес фізичного моделювання бази даних, який описує як система з технологією буде реалізована за допомогою конкретної системи СУБД. Мета – фактичне впровадження бази даних. [17]

На рисунку 3.4 представлена фізична моделб бази даних, які реалізована в веб-орієнтованій технології для підтримки взаємодії користувачів.

В процесі фізичного моделювання бази даних соціальної мережі було виділено створено такі сутності:

– Користувачі (user);

– Підписки (follow);

– Вподобайки (posts_liked_users);

– Пост (post).

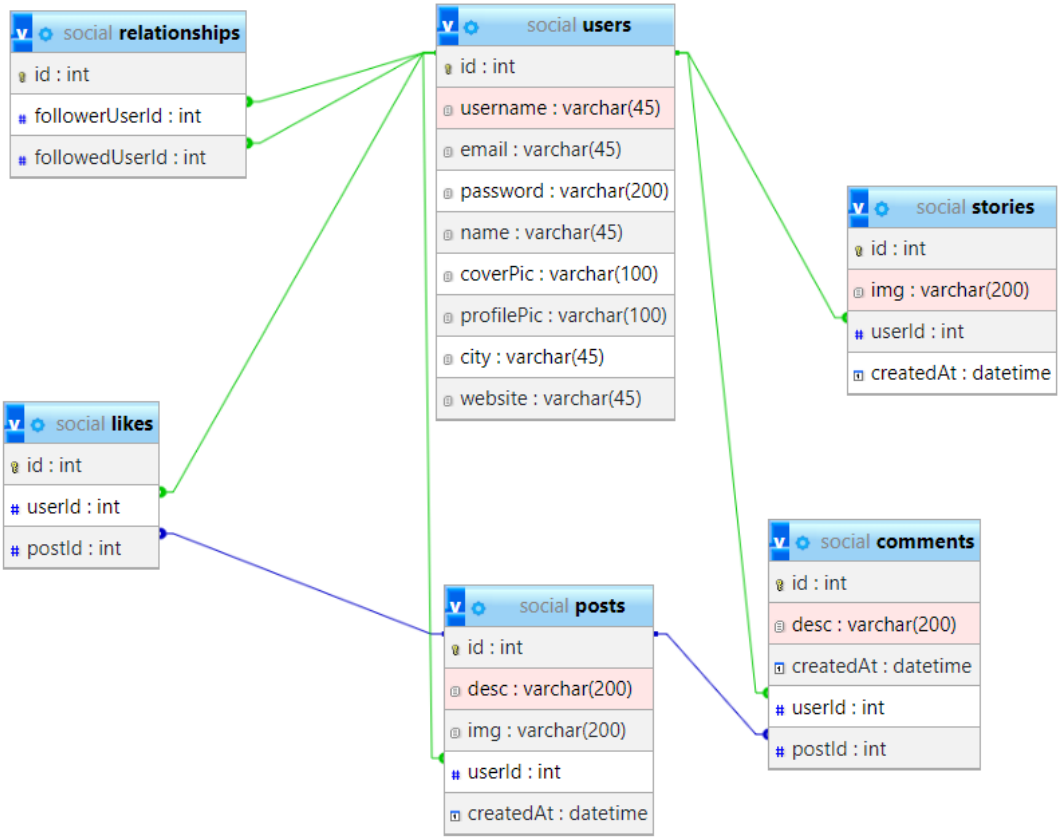


Рисунок 3.4 – Фізична модель даних бази даних соціальної мережі

База даних соціальної мережі зберігає особисті дані користувачів та їх матеріали, які були використані для постів. Для чату було використано сховище даних MongoDB, яка замість загальноприйнятих таблиць та полів використовує сутності. Головна перевага MongoDB – зручність в зберіганні слабо структурованої інформації. Також дана БД може працювати з JavaScript та повертати користувацькі функції в відповідь на запит. Серед головних переваг також можна відзначити повну сумісність з React.js.

На рисунках 3.5 – 3.7 представлено документи для зберігання даних про створені сесії, користувачів та повідомлень.

Рисунок 3.5 – Документ для зберігання даних про створені сесії між користувачами

Поле _id додається MongoDB для унікальної ідентифікації документа в колекції.

Рисунок 3.6 – Документ для зберігання даних про надіслані повідомлення

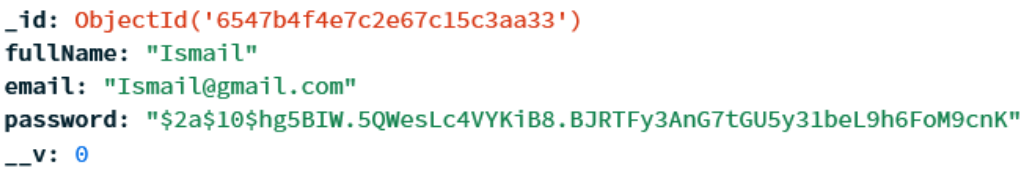


Рисунок 3.7 – Документ для зберігання даних про авторизованих користувачів

4 РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ

4.1 Архітектура інформаційної технології

Архітектура інформаційної технології складається з наступного стеку:

- MongoDB – нереляційна база даних, яка зберігає дані в документах JSON.

MongoDB використовує концепцію ODM (зіставлення документів об’єктів). На основі створеної моделі Mongo генерує схему, а потім обмінюється даними з БД. Для перетворення моделі в схему використовується бібліотека mongoose.

- Express.js – Відповідає за серверну частину інформаційної технології та описує сценарії взаємодії клієнту з сервером.

Якщо користувачу необхідно отримати дані, які зберігаються в БД, то необхідно за допомогою параметрів відправити запит на сервер, які буде перевіряти Express.js.

Для обробки запитів створюються маршрутизатори та контролери. Маршрутизатор приймає запит від React.js і надає інформацію про контролер, який прийматиме запит. Контролер в свою чергу оцінює та виконує запит, який був представлений маршрутизатором. По суті контролер це функція зворотнього виклику.

- React.js – відповідає за користувацький інтерфейс.

В інформаційній технології через router відбувається перехід по сторінках. Під час маршрутизації будуть генеруватися компоненти. Також для організації HTTP запитів використовується бібліотека Axios.

- Node.js – середовище виконання процесів на основі DOM.

Для інтеграції модулів необхідно використовувати менеджер пакетів npm який надає можливість завантаження пакетів внутрішнього та зовнішнього вузла.

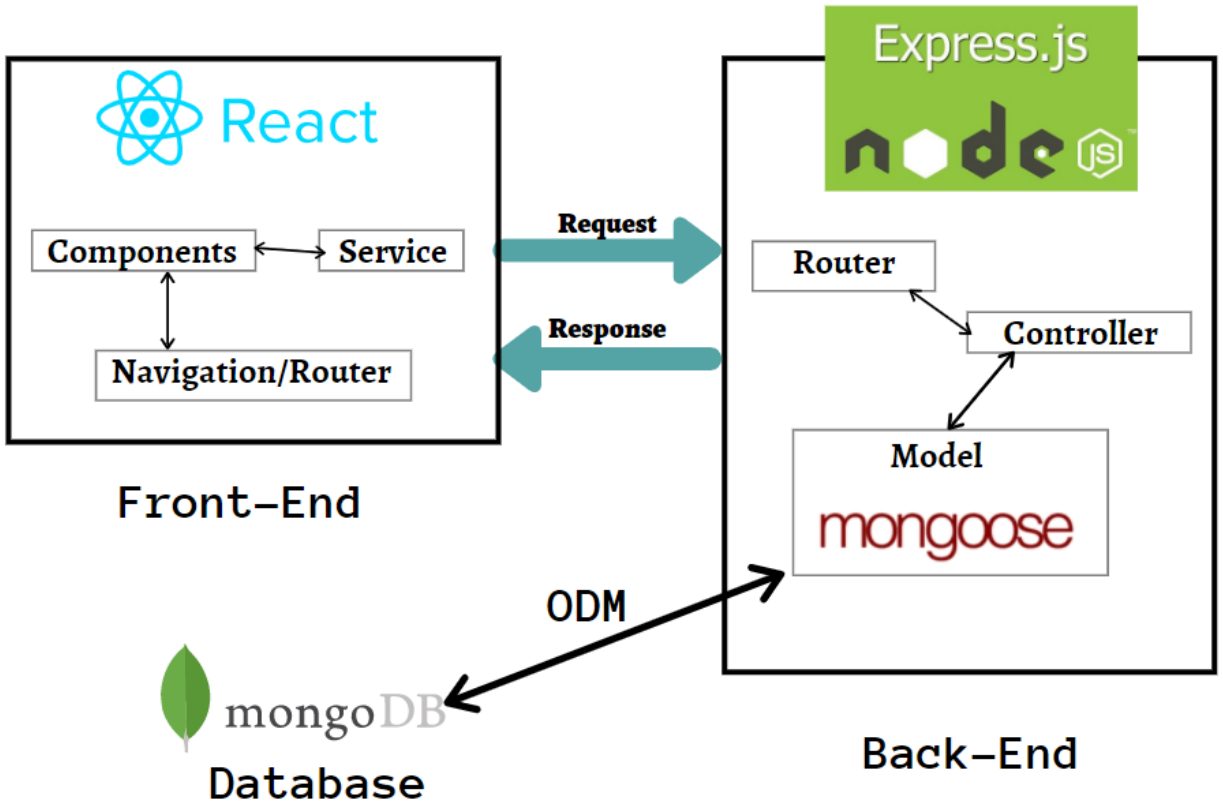


Рисунок 4.1 – Схема архітектури інформаційної технології

Джерело: [22]

4.2 Інтеграція БД в інформаційну технологію

Для початку необхідно створити базу даних для чату в хмарному середовищі MongoDB. Для цього необхідно перейти на сайт https://cloud.mongodb.com. Після авторизації в системі необхідно перейти в вкладку «Project» та натиснути на кнопку «New Project» (рис 4.2).

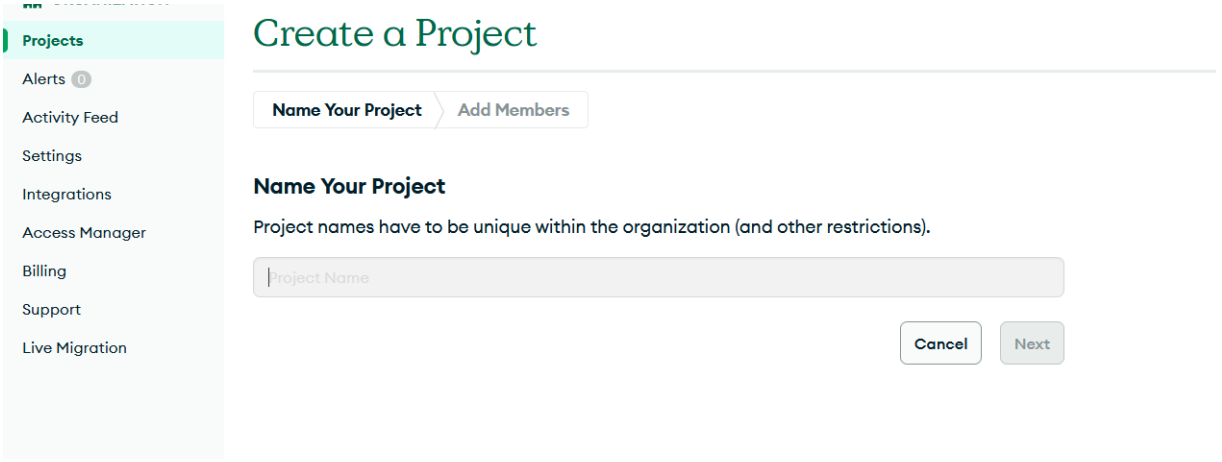


Рисунок 4.2 – Сторінка створення назви БД

Після того як було надано назву проекту, відбувається переадресація на сторінку щойно створеної БД. Наступним пунктом буде створення розгортання з вибором регіону та налаштуванням серверу. На рисунку 4.3 представлено результат налаштування серверу. Було обрано безкоштовний тип надання послуг. В якості регіону обрано Стокгольм, а в якості провайдеру було обрано послуги компанії AWS.

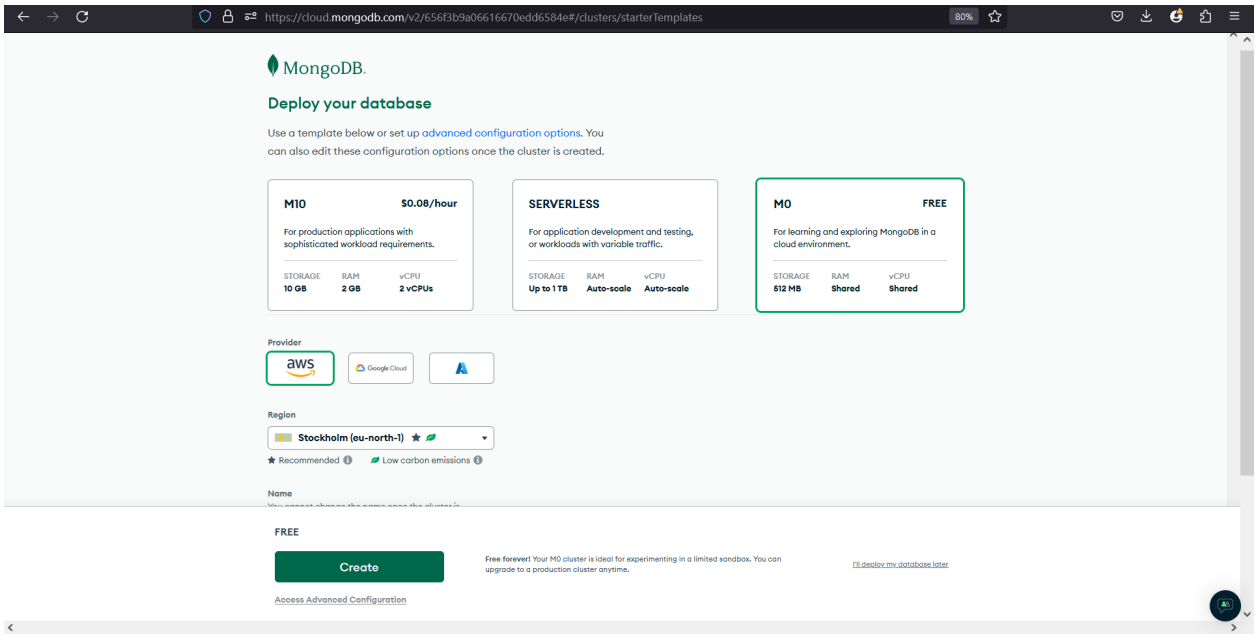


Рисунок 4.3 – Результат налаштування серверу

Тепер необхідно створити користувача (адміністратора БД). Для цього необхідно ввести логін та пароль, який буде надавати права доступу на коритсування БД. При потребі можна додати інших користувачів які матимуть аналогічний спект доступу до MongoDB. На рисунку 4.4 представлено процес створення користувача.

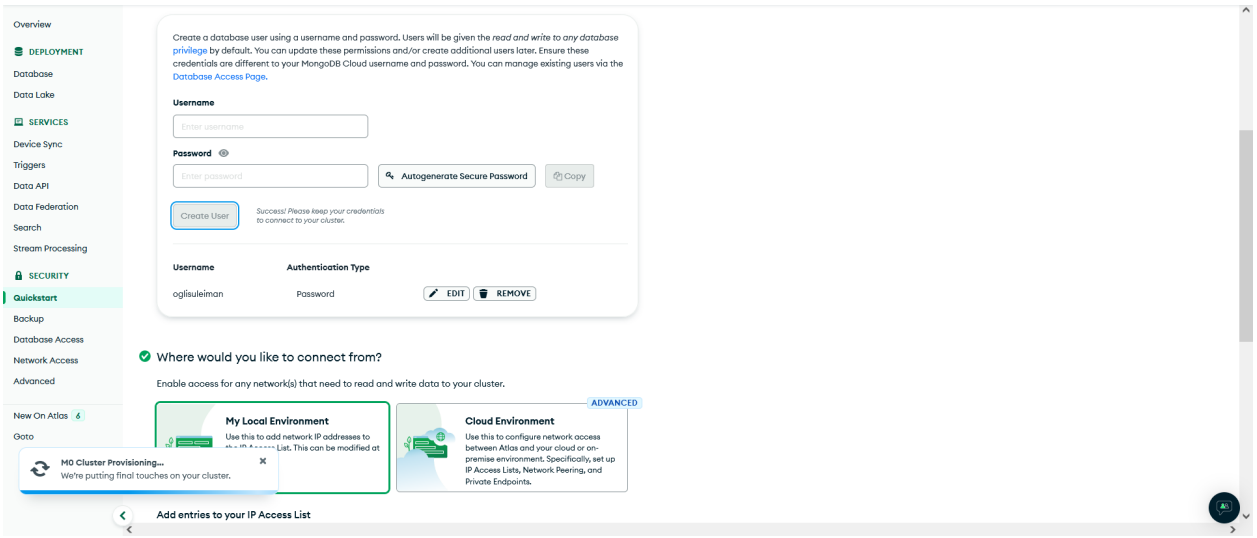


Рисунок 4.4 – Процес створення користувача

Після виконання всіх операцій з налаштуванням БД, користувач отримає унікальне посилання, яке необхідно інтегрувати в інформаційну систему. На рисунку 4.5 представлено сторінку з інформацією підключення до кластеру.

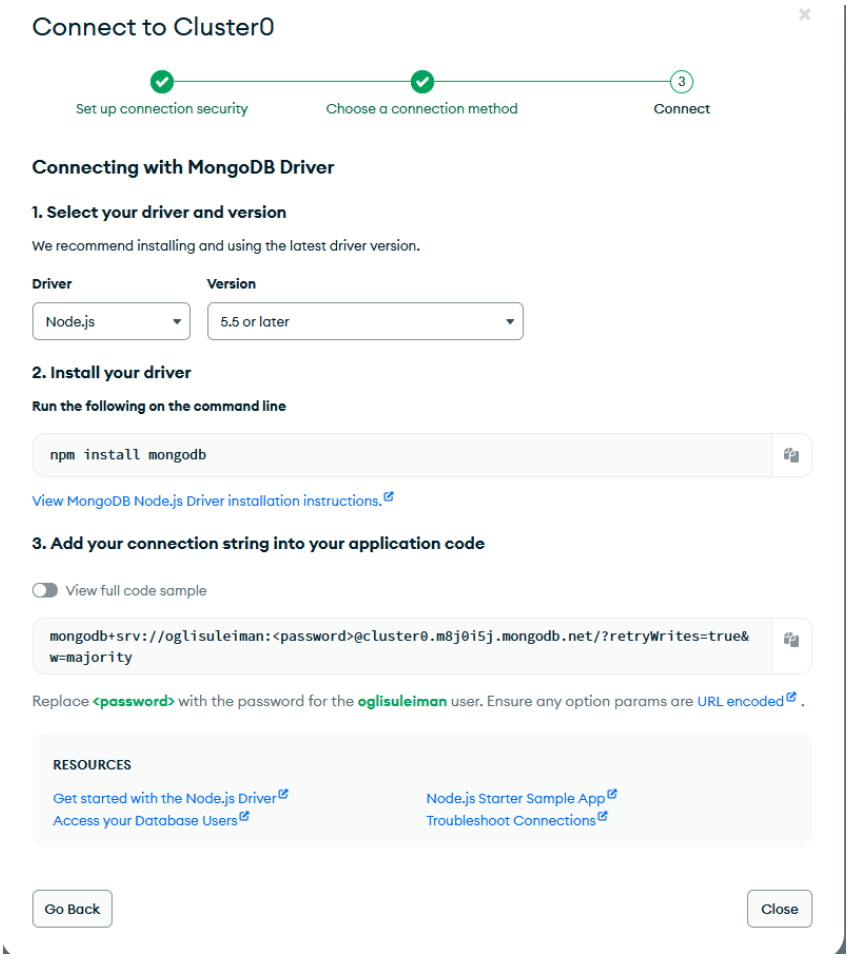
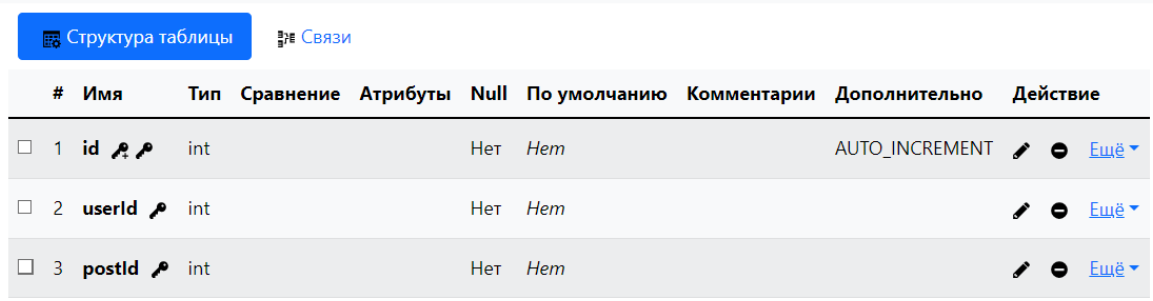


Рисунок 4.5 – Сторінка з інформацією підключення до кластеру

Для створення бази даних соціальної мережі використовується локальний web-сервер OpenServer з інструментом phpMyAdmin. За допомогою MySQL було створено таблиці та атрибути які пов’язні між собою за допомогою унікальних ключів. На рисунках 4.6 – 4.11 представлено створені таблиці з атрибутами бази даних “social_db”.

Рисунок 4.6 – Структура таблиці «comments»


Структура таблицы Связи

#	Имя	Тип	Сравнение	Атрибуты	Null	По умолчанию	Комментарии	Дополнительно	Действие
1	id	int			Нет	Нем		AUTO_INCREMENT	Ещё
2	userId	int			Нет	Нем			Ещё
3	postId	int			Нет	Нем			Ещё

Рисунок 4.7 – Структура таблиці «likes»


Структура таблицы Связи

#	Имя	Тип	Сравнение	Атрибуты	Null	По умолчанию	Комментарии	Дополнительно	Действие
1	id	int			Нет	Нем		AUTO_INCREMENT	Ещё
2	desc	varchar(200)	utf8mb4_0900_ai_ci		Да	NULL			Ещё
3	img	varchar(200)	utf8mb4_0900_ai_ci		Да	NULL			Ещё
4	userId	int			Нет	Нем			Ещё
5	createdAt	datetime			Да	NULL			Ещё

Рисунок 4.8 – Структура таблиці «posts»


Структура таблицы Связи

#	Имя	Тип	Сравнение	Атрибуты	Null	По умолчанию	Комментарии	Дополнительно	Действие
1	id	int			Нет	Нем		AUTO_INCREMENT	Ещё
2	followerUserId	int			Нет	Нем			Ещё
3	followedUserId	int			Нет	Нем			Ещё

Рисунок 4.9 – Структура таблиці «relationship»


Структура таблицы Связи

#	Имя	Тип	Сравнение	Атрибуты	Null	По умолчанию	Комментарии	Дополнительно	Действие
1	id	int			Нет	Нем		AUTO_INCREMENT	Ещё
2	img	varchar(200)	utf8mb4_0900_ai_ci		Нет	Нем			Ещё
3	userId	int			Нет	Нем			Ещё
4	createdAt	datetime			Да	NULL			Ещё

Рисунок 4.10 – Структура таблиці «stories»

Структура таблицы | Связи

#	Имя	Тип	Сравнение	Атрибуты	Null	По умолчанию	Комментарии	Дополнительно	Действие
1	**id**	int			Нет	*Нет*		AUTO_INCREMENT	Ещё
2	**username**	varchar(45)	*utf8mb4_0900_ai_ci*		Нет	*Нет*			Ещё
3	**email**	varchar(45)	*utf8mb4_0900_ai_ci*		Нет	*Нет*			Ещё
4	**password**	varchar(200)	*utf8mb4_0900_ai_ci*		Нет	*Нет*			Ещё
5	**name**	varchar(45)	*utf8mb4_0900_ai_ci*		Нет	*Нет*			Ещё
6	**coverPic**	varchar(100)	*utf8mb4_0900_ai_ci*		Да	*NULL*			Ещё
7	**profilePic**	varchar(100)	*utf8mb4_0900_ai_ci*		Да	*NULL*			Ещё
8	**city**	varchar(45)	*utf8mb4_0900_ai_ci*		Да	*NULL*			Ещё
9	**website**	varchar(45)	*utf8mb4_0900_ai_ci*		Да	*NULL*			Ещё

Рисунок 4.11 – Структура таблиці «users»

Після того як бази даних створені, необхідно виконати операцію підключення файлів БД до інформаційної технології та чату. Для підключення MongoDB необхідно створити connection.js файл в якому буде посилання на проект, який був створений в хмарному середовищі CloudMongo. На рисунках 4.12 – 4.13 представлено результат створення connection.js файлу та результат успішного підключення до БД.

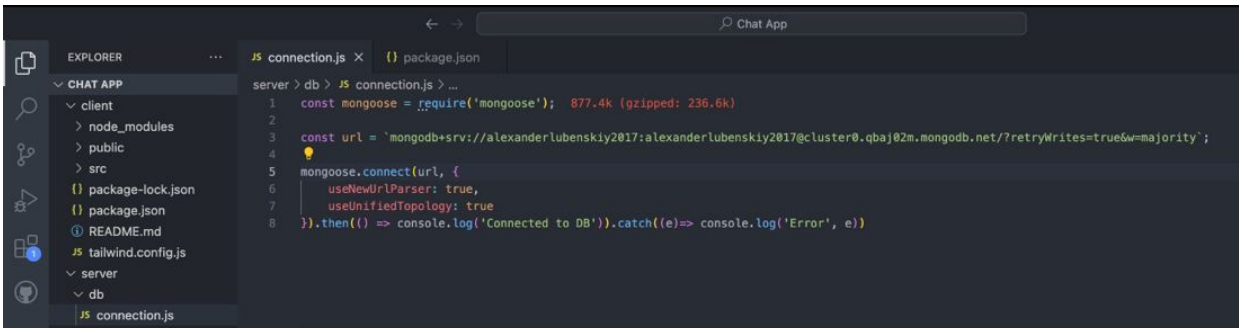


Рисунок 4.12 - Результат створення connection.js файлу

```
[nodemon] 3.0.1
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node app.js`
listening on port 8000
Connected to DB
```

Рисунок 4.13 – Результат успішного підключення до БД

Для підключення БД MySQL до інформаційної технології необхідно використати бібліотеку npm – mysql. На рисунку 4.14 представлено результат створення файлу connect.js.

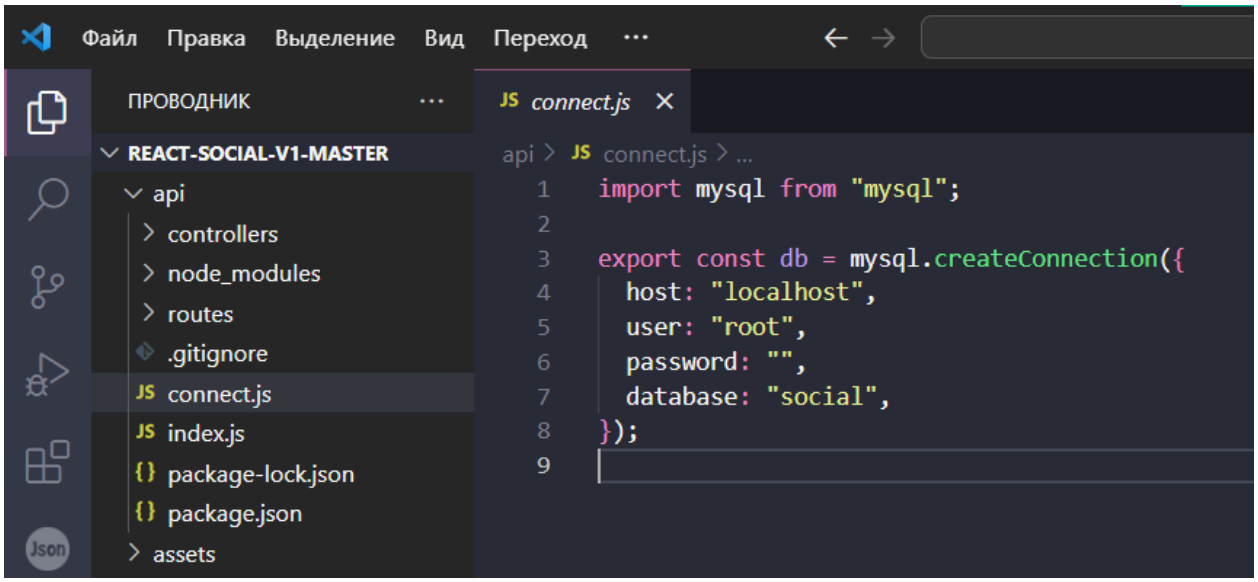


Рисунок 4.14 – Результат створення файлу connect.js

4.3 Створення технології хешування даних

Для розробки технології хешування даних необхідно встановити всі необхідні бібліотеки, які будуть використовуватися в процесі розробки технології. Для цього необхідно виконати команду npm install та встановити всі залежності

для серверної частини. На рисунку 4.15 представлено бібліотеки для створення технології хешування даних.

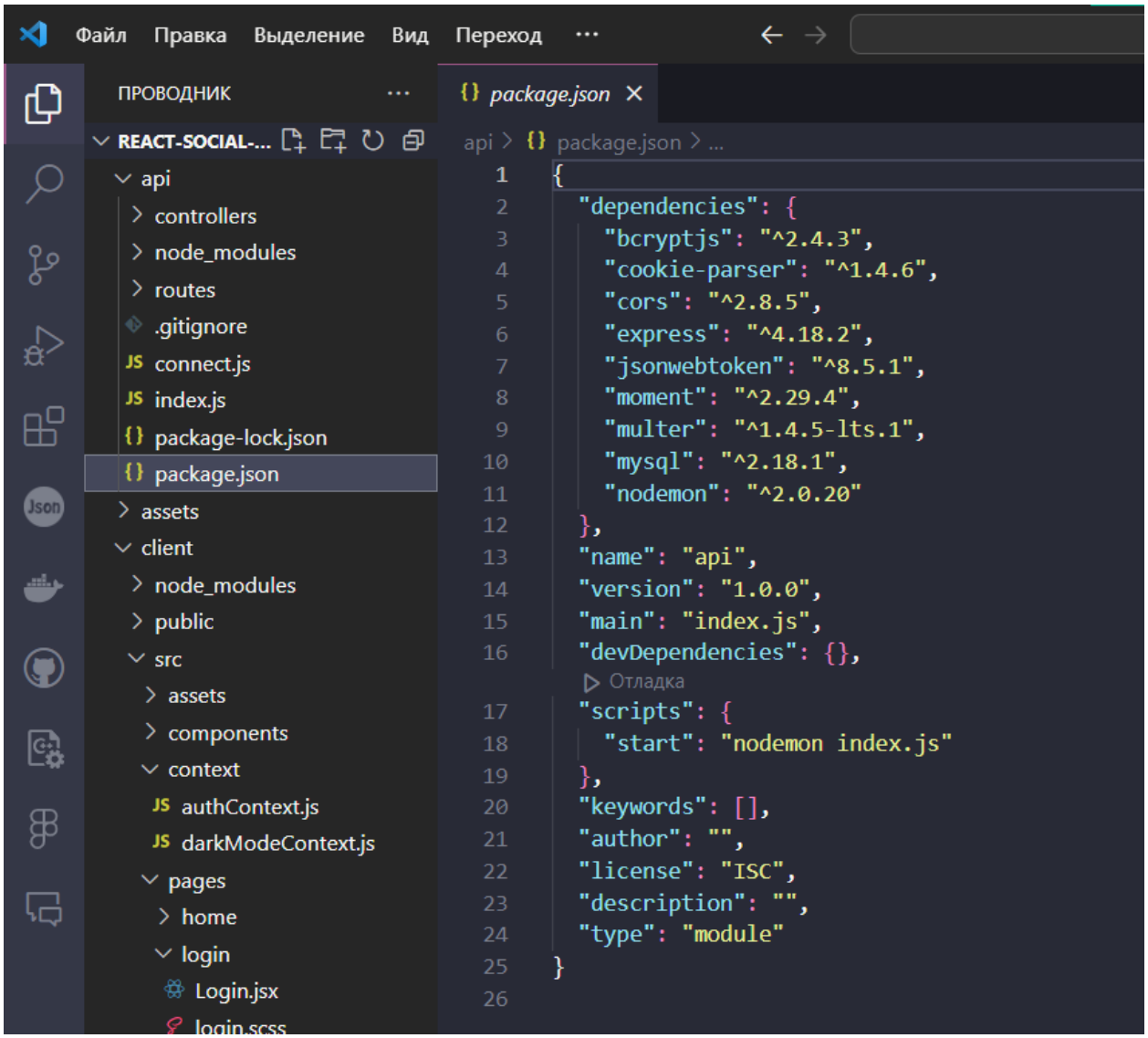


Рисунок 4.15 – Результат встановлення залежностей для серверної частини

Для початку необхідно розробити систему хешування даних при авторизації користувача в соціальній мережі. Для цього було використано бібліотеки bccrypt.js та jsonwebtoken. За допомогою адаптивної хеш-функції формування ключа відбуватиметься збереження паролів та даних користувача в базі даних. Це надасть можливість захистити дані користувачів в разі проникнення в інформаційну технологію вірусу або проведення ДДОС атаки. Для створення та

підтвердження токенів, які необхідні під час аутентифікації або авторизації користувачів було використано бібліотеку jsonwebtoken.

На рисунку 4.16 представлено результат створення технології хешування та підтвердження особистих даних під час авторизації. Повний лістинг коду наведено в додатку Б.

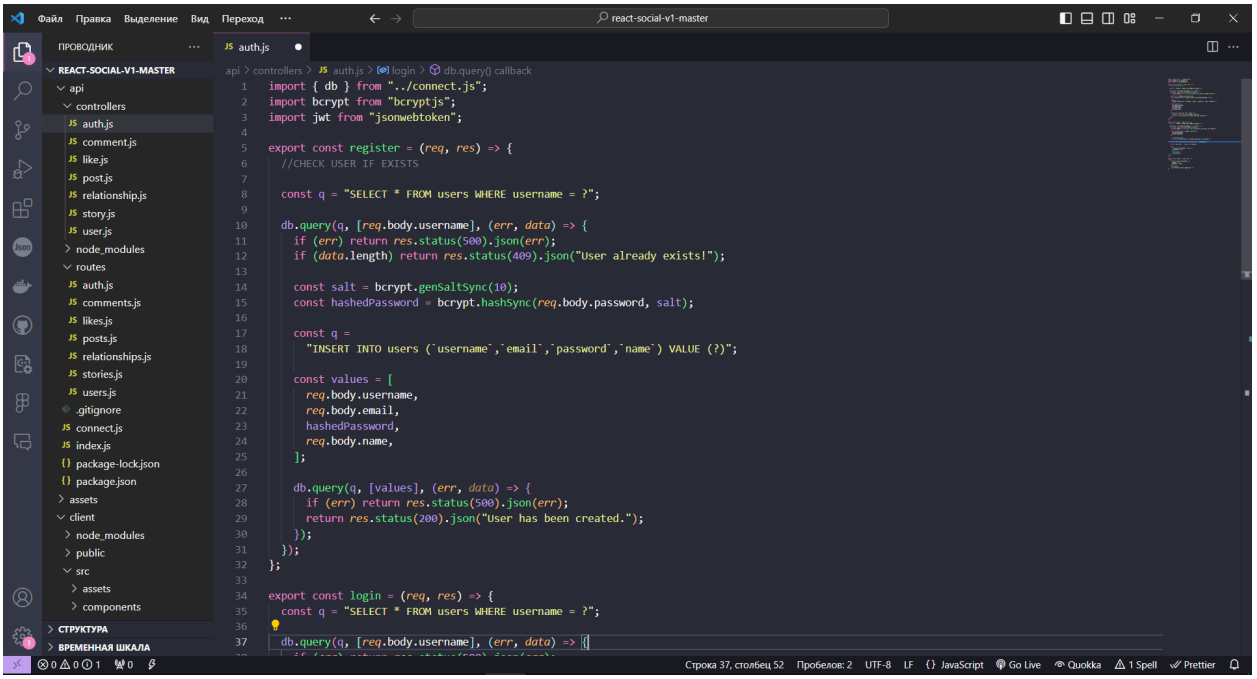


Рисунок 4.16 - Результат створення технології хешування та підтвердження особистих даних під час авторизації

Аналогічно дану технологію було інтегровано в контролери створення посту та оновлення особистих даних на сторінці користувача. На рисунках 4.17 – 4.18 представлено результат інтеграції технології в контролер user.js та user.js.

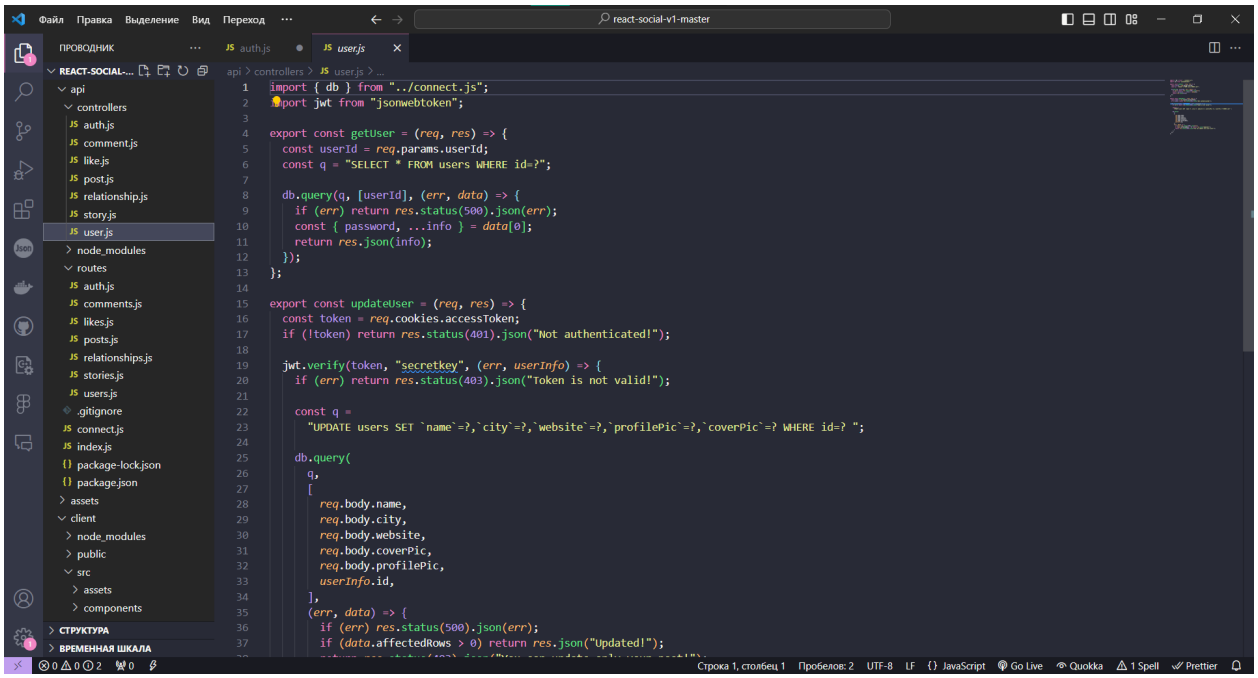


Рисунок 4.17 – Результат інтеграції технології в контролер user.js

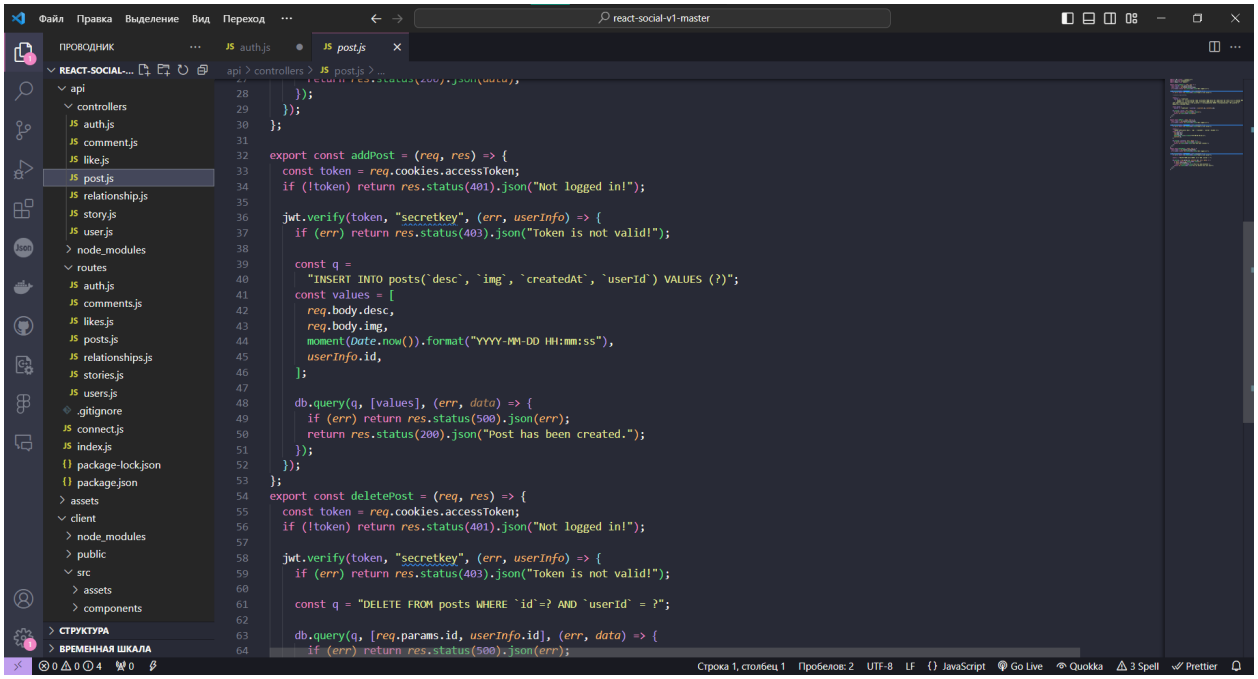


Рисунок 4.18 – Результат інтеграції технології в контролер post.js

4.4 Реалізація клієнтської частини інформаційної технології

Аналогічно з серверною частиною, необхідно перш за все встановити залежності в клієнтській частині. Для цього потрібно використати команду npm install. На рисунку 4.19 представлено створені залежності в файлі package.json.

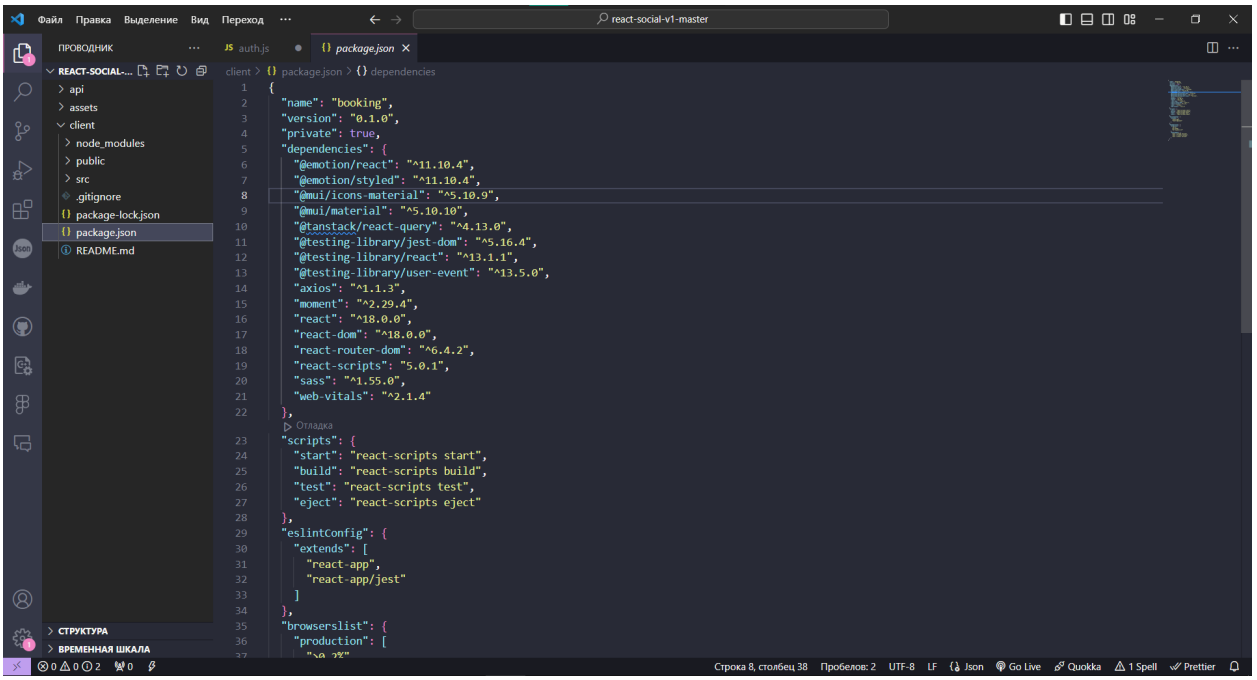


Рисунок 4.19 – Результат створення залежностей в файлі package.json.

Після цього необхідно створити компонент роутеру, який надаватиме кожному компоненту власну адресу переадресації. Для цього необхідно імпортувати всі необхідні компоненти до сторінки App.js. На рисунку 4.20 представлено результат створення сторінки App.js. Повний лістинг коду представлено в додатку В.

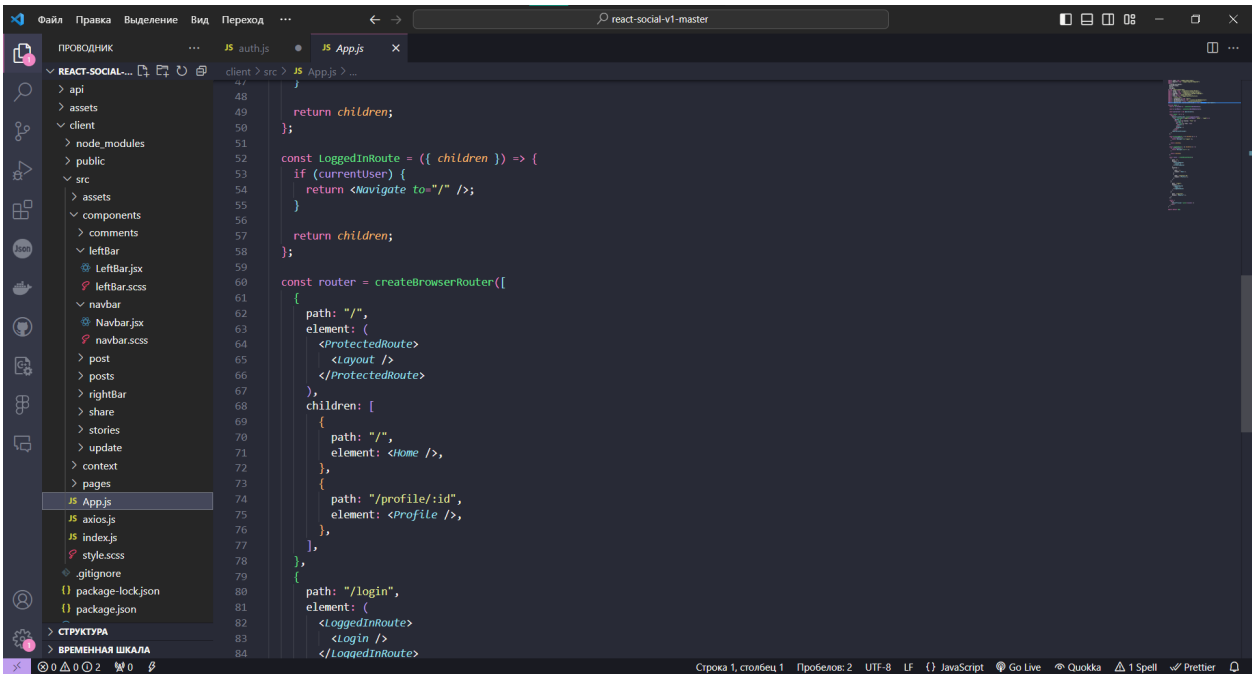


Рисунок 4.20 – Результат створення сторінки навігації App.js

Для інтуїтивно зрозумілого інтерфейсу було створено компоненти, які представлені в таблиці 4.1. Для кожного компоненту було створено власний роутер та за допомогою CSS візуалізовано відповідно до дизайну. Лістинг коду style.scss представлено в додатку Г.

Таблиця 4.1 – Створені компоненти та їх задачі

Номер компоненту	Назва компоненту	Задача
1	Comments.jsx	Відображення створених коментарів користувача до постів
2	LeftBar.jsx	Відображення статичного навігаційного меню

3	Navbar.jsx	Відображення профілю користувача та можливості зміни кольору інтерфейсу

Продовження таблиці 4.1 – Створені компоненти та їх задачі

4	Post.jsx	Відображення створених постів користувача
5	RightBar.jsx	Статичний блок відображення інформації про дії інших користувачів
6	Share.jsx	Блок заповнення інформації яка буде відображена в пості
7	Stories.jsx	Блок з історіями інших користувачів
8	Update.jsx	Блок зміни особистої інформації користувача

Джерело: побудовано автором

При створенні чату використовувались також клієнтська та серверна частина. В серверну частину було інтегровано технологію хешування даних. На рисунку 4.21 представлено результат хешування повідомлень в документі особистих повідомлень.

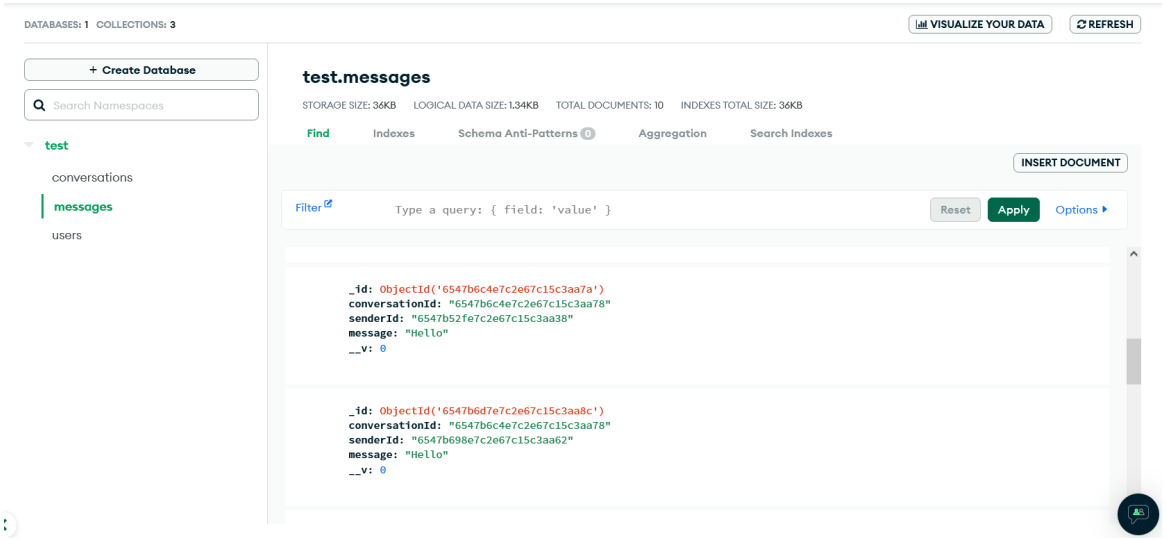


Рисунок 4.21 – Результат інтеграції технології хешування в серверну частину чату

Також для чату за допомогою технології Socket.IO було реалізовано технологію обміну повідомленями. Перевага використання даної технології заключається в використанні проксі та балансу навантаження. Тобто якщо буде велика кількість користувачів то за допомогою даної технології можна буде оптимізувати навантаження на систему та сервер що збереже ресурси ПК. Також перевагою використання даної технології є підтримка автоматичного перепідключення при втраті зв’язку.

На рисунку 4.22 представлено результат інтеграції технології Socket.IO в компоненти чату. Повний лістинг коду app.js предстaлено в додатку Д.

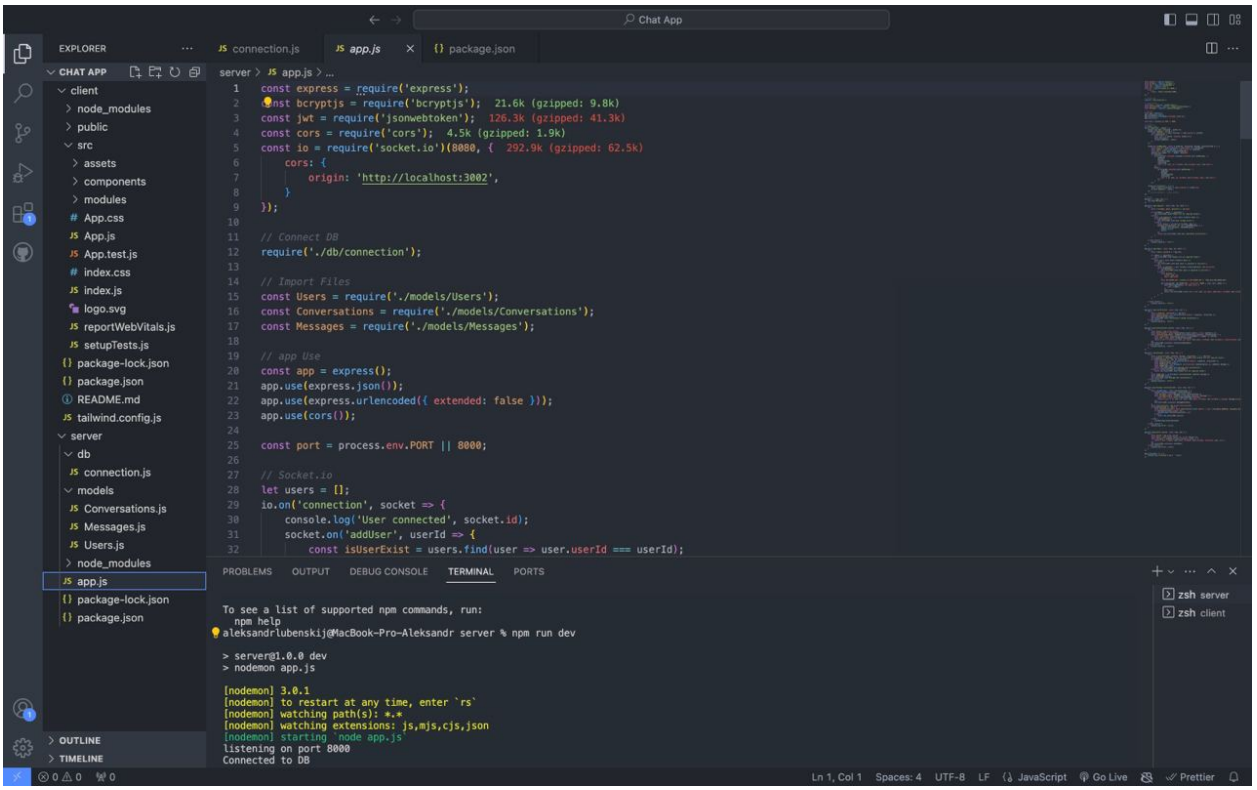


Рисунок 4.22 – Результат інтеграції технології Socket.IO в компоненти чату

4.5 Настанови з використання

На початку використання соціальної мережі користувача зустрічає форма авторизації. На рисунку 4.23 представлено сторінка авторизації. Якщо користувач не мав до цього сторінки в соціальній мережі то від має змогу реєстрації (рис. 4.24).

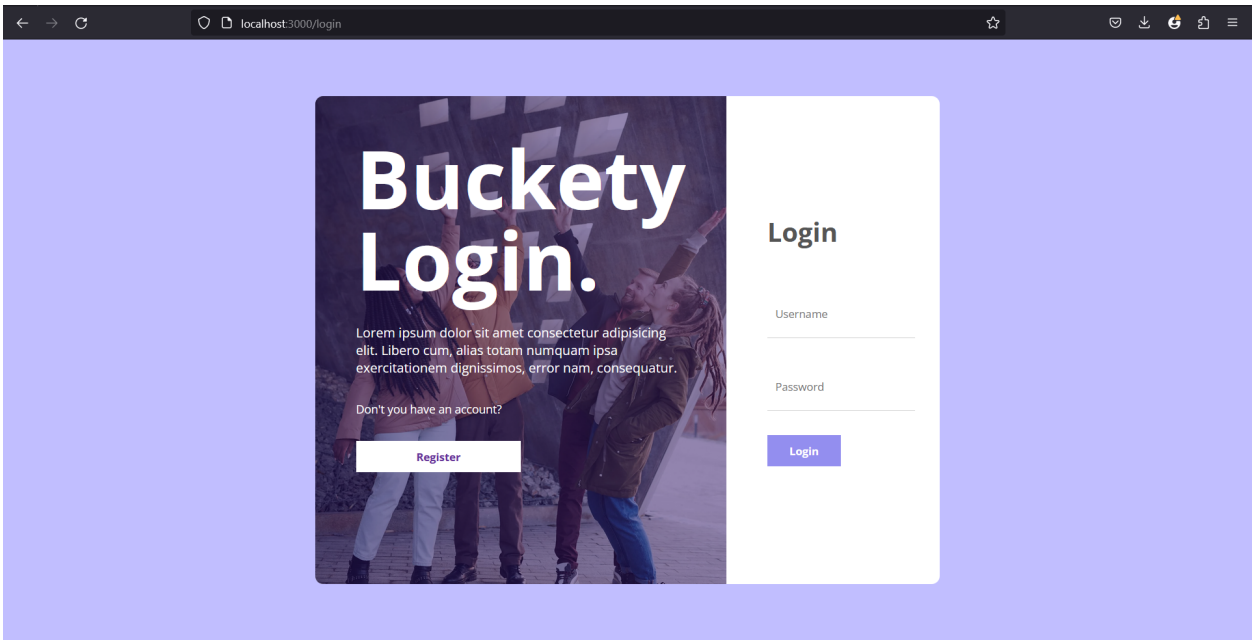


Рисунок 4.23 – Сторінка авторизації

Рисунок 4.24 – Сторінка реєстрації нового користувача

Після успішної авторизації/реєстрації користувач потрапляє на сторінку з постами та інформацією інших користувачів. Головна сторінка соціальної мережі представлена на рисунку 4.25.

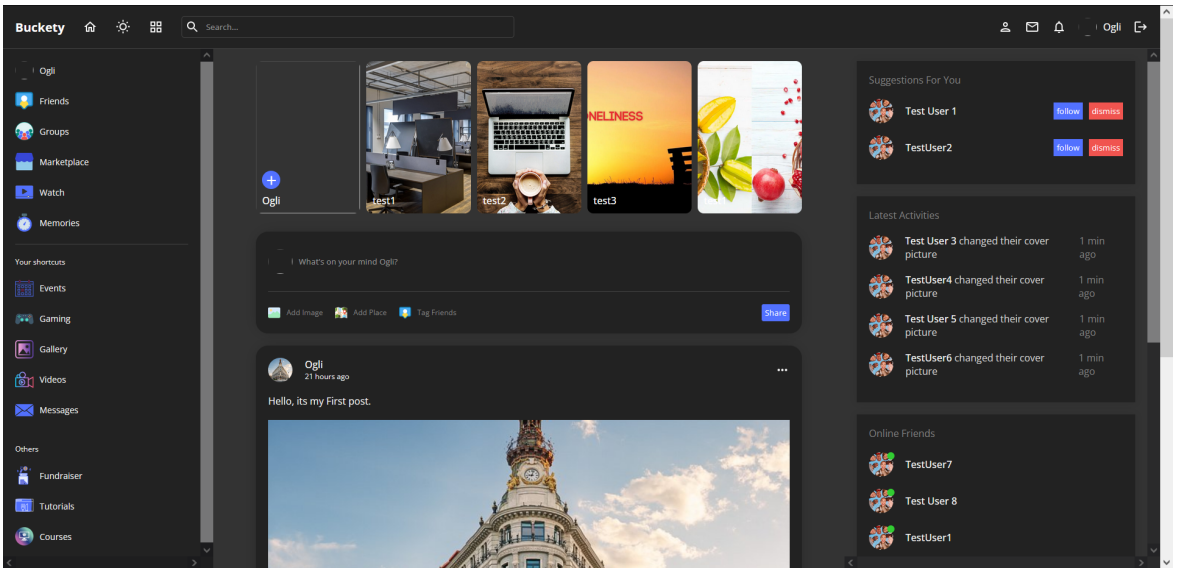


Рисунок 4.25 – Вигляд головної сторінки соціальної мережі

Якщо користувачу необхідно додати більше інформації про себе то він може перейти на особисту сторінку на натиснути на кнопку «Update». Після цього користувач за допомогою форми оновлення особистих даних зможе додати інформацію про себе. На рисунку 4.26 представлено вигляд форми оновлення особистих даних.

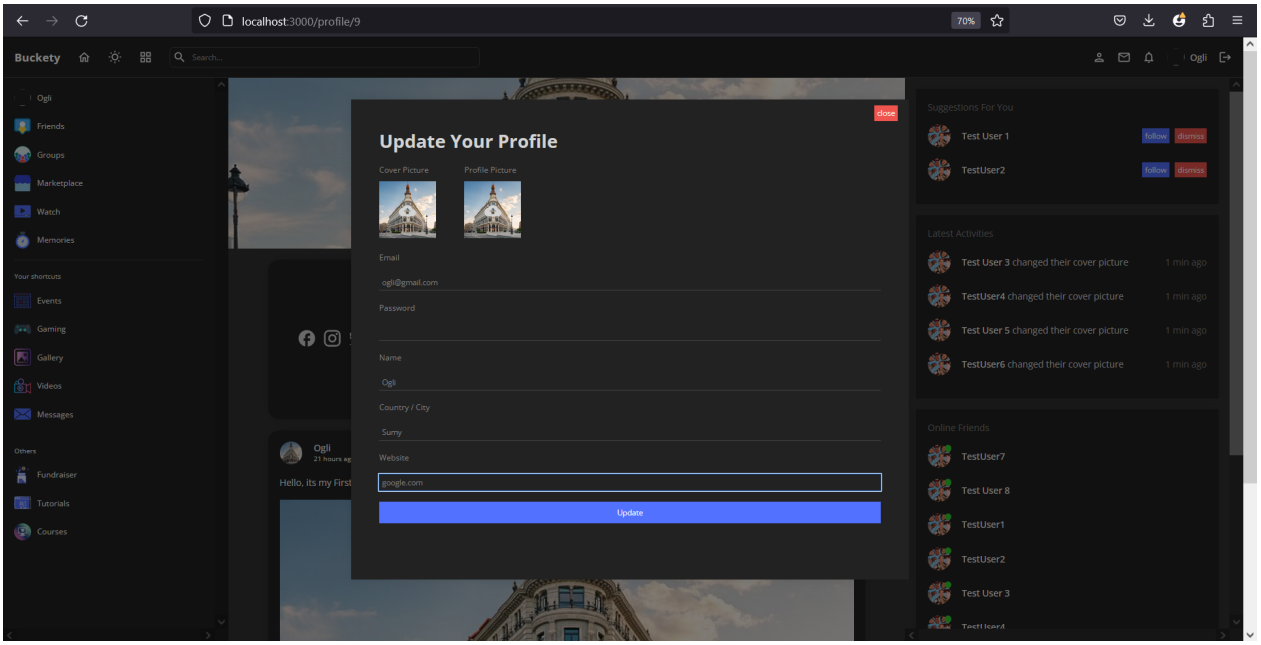


Рисунок 4.26 – Форма оновлення особистих даних

Дана соціальна мережа надає користувачу можливість створювати власні пости та завантажувати результат на головну сторінку. На рисунку 4.27 представлено процес створення власного посту.

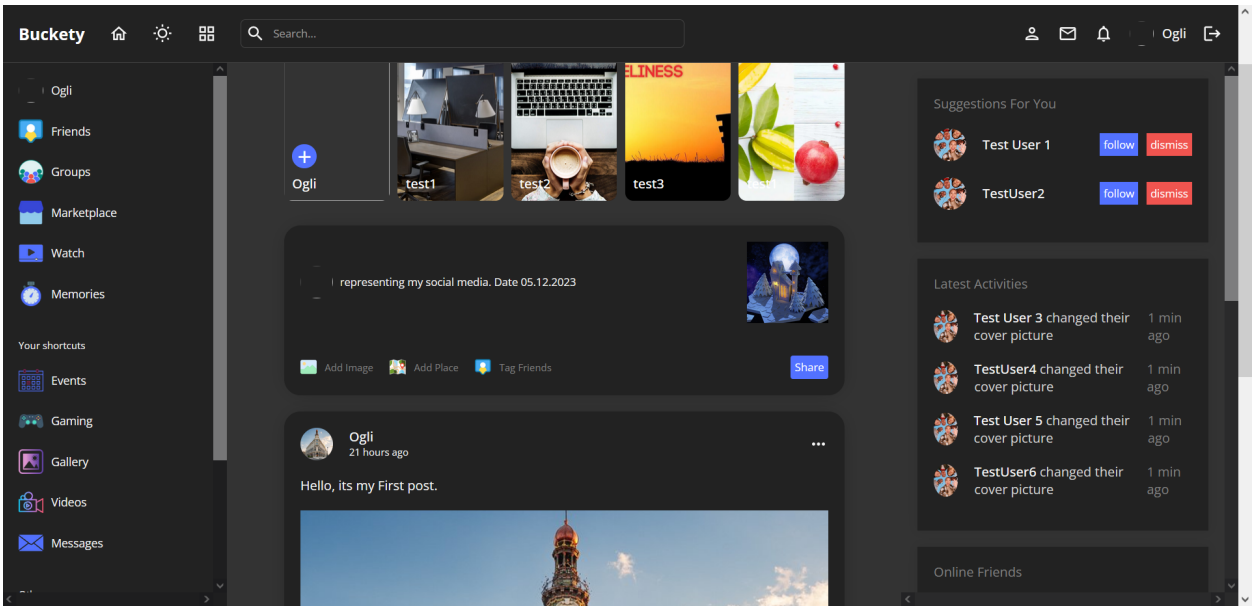


Рисунок 4.27 – Процес створення власного посту

Після натиснення кнопки «Share» користувач поширює створений пост на головну сторінку соціальної мережі. На рисунку 7.28 представлено результат створення посту.

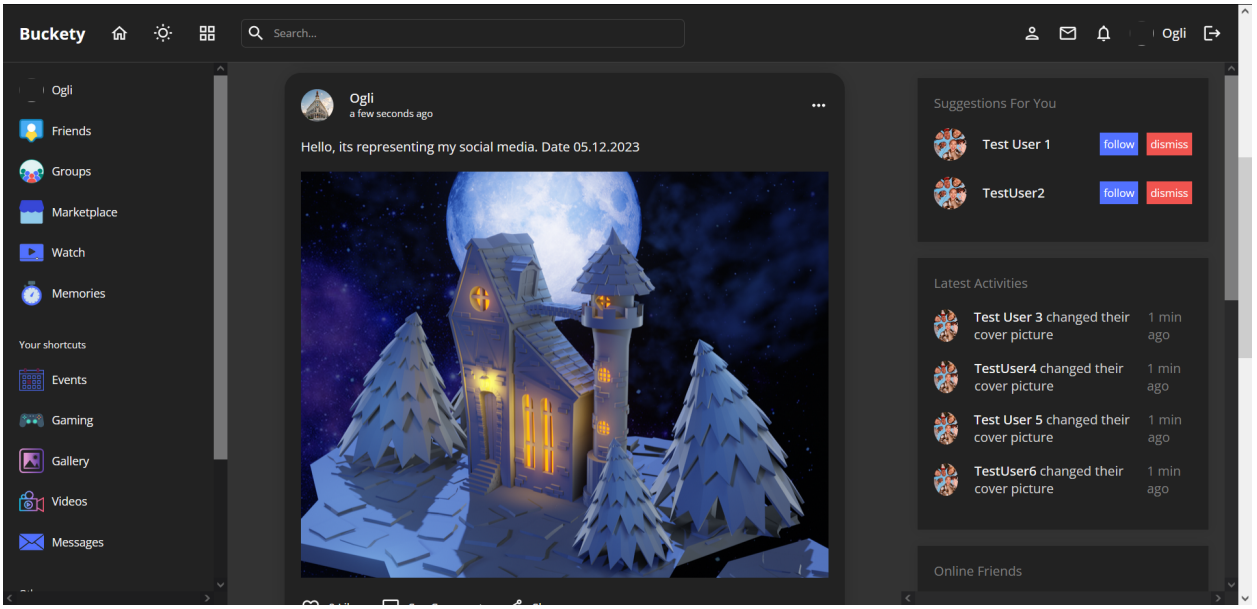


Рисунок 4.28 – Результат створення власного посту

Після того як пост створено, користувач має змогу додати коментар та поставити вподобайку (рис. 4.29).

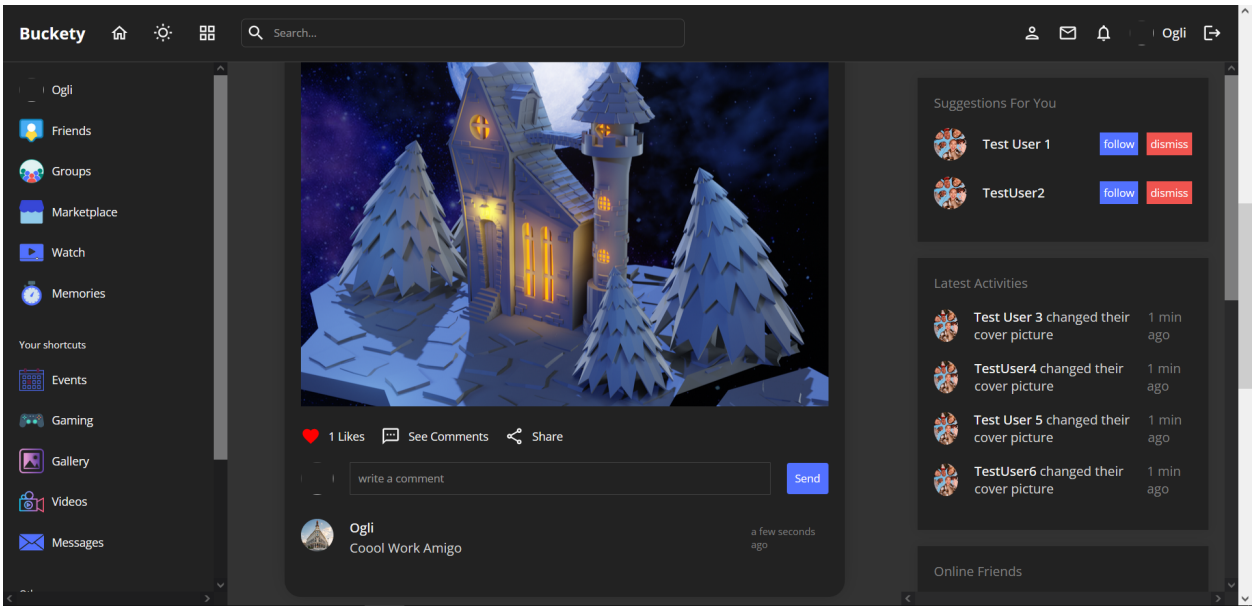


Рисунок 2.29 – Результат додавання вподобайки та коментарів

Також якщо користувачу не подобається колір інтерфейсу то він може змінити на білий (рис. 4.30).

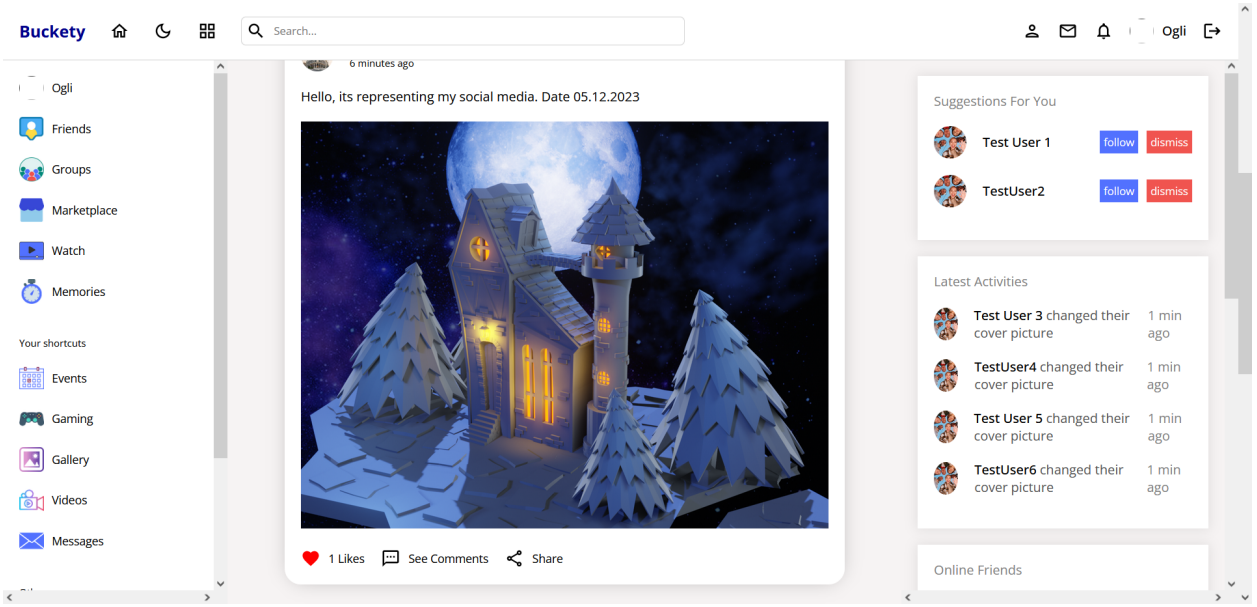


Рисунок 4.30 – Результат зміни кольору інтерфейсу

Якщо користувач хоче відредагувати власні пости то система надає можливість видалення створених постів. На рисунку 4.31 представлено результат видалення створеного посту.

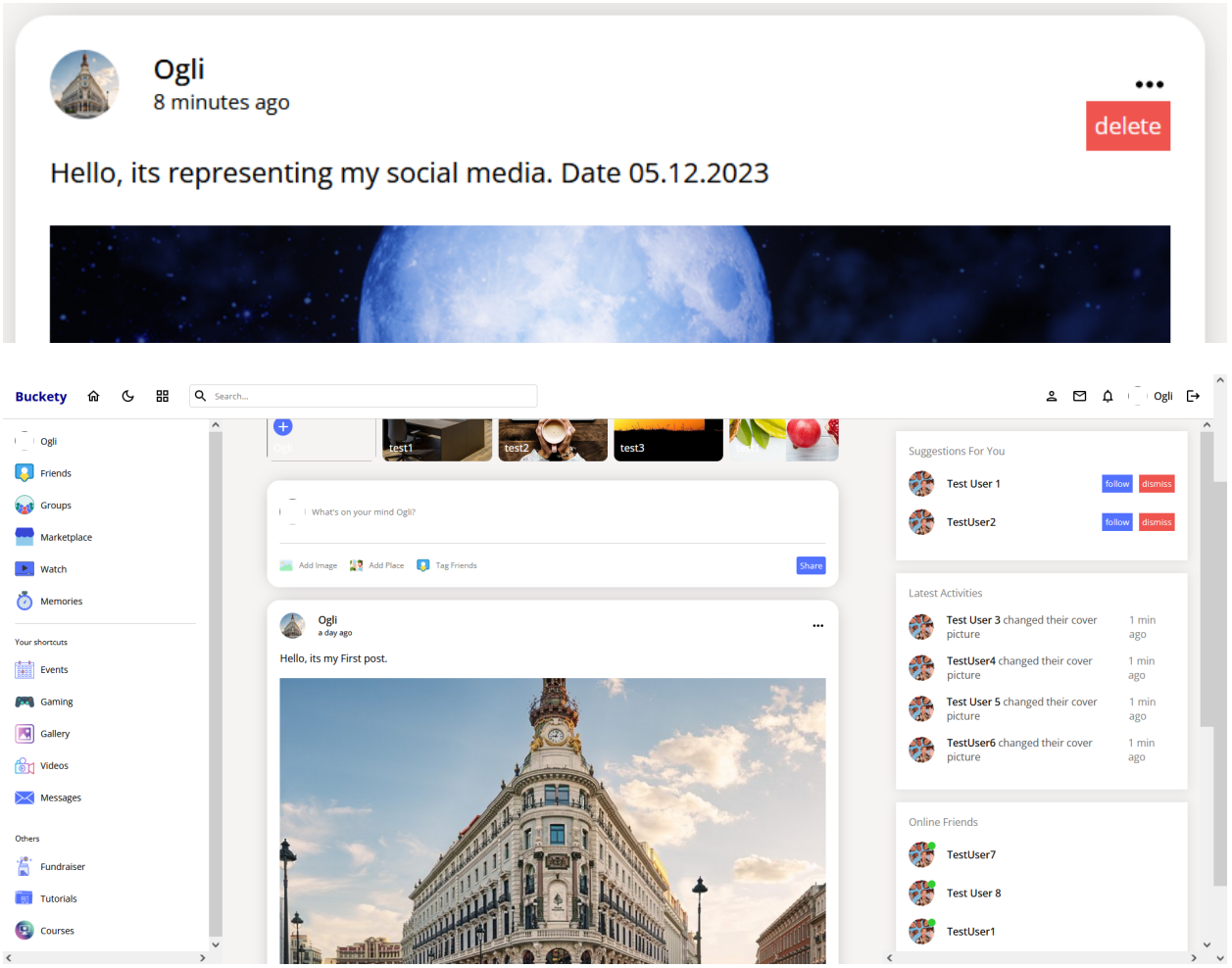


Рисунок 4.31 – Результат видалення посту

При використанні чату в реальному часі користувач може обрати будь якого користувача, який також зареєстрований в системі та надіслати йому повідомлення. На рисунку 4.32 представлено результат відправки повідомлень з різних облікових записів користувачів.

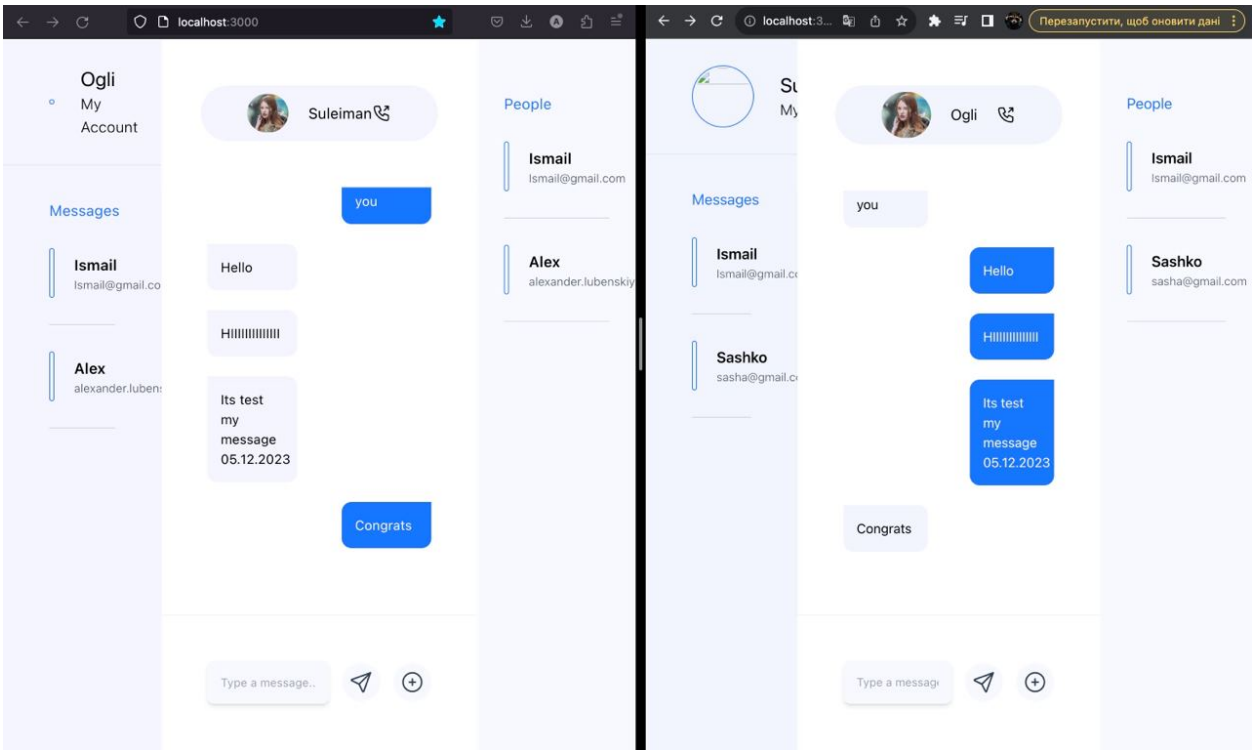


Рисунок 4.32 – Процес листування двох користувачів з різних облікових записів

ВИСНОВКИ

В рамках виконання кваліфікаційної роботи бакалавра було розроблено інформаційної технології підтримки взаємодії користувачів у веб-соціальній мережі. Досліджувана технологія базується на використанні передових технологій для надання безперервного зв’язку між клієнтом та сервером, а саме: MongoDB, Express.js, React.js, Node.js.

Впровадження досліджуваної технології дозволить в реальному часі оброблювати та передавати користувачу інформацію. Крім того, система використовує методи шифрування даних для більш безпечного захисту даних користувачів.

Проведений аналіз існуючих інструментів для підтримки взаємодії користувачів у веб-соціальних мережах вказує на значну потребу в розробці інформаційної технології, яка здатна адаптуватися до мінливих трендів та вимог спільнот.

За допомогою структурно функціонального моделювання було визначено основні процеси які надає інформаційна технологія.

Отже, запропонована інформаційна технологія підтримки взаємодії користувачів у веб-соціальній мережі є актуальною та володіє великим потенціалом для оптимізації взаємодії, захисту особистих даних та покращення залучення аудиторії до платформи. Її використання сприятиме створенню більш динамічних та відкритих веб-спільнот.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Russo, M., Bergami, M., &Morgandin G. (2018). Surviving a day without smartphones. MIT Sloan Management Review, 59(2), 7-9;

2. Suki N., & Suki N. M. (2019). Acquiring travel-related information from mobile social networking services, 32(4), 5-8;

3. Hollebeek L. D. & Macky K. (2019). Digital content marketing`s role in fostering consumer engagement, trust, and value: Framework, fundamental propositions and implications, 45, 27-41;

4. Stockdale L.A. & Coyne S.M. (2020). Bored and online: Reasons for using social media, problematic social networking site use, and behavionar outcomes across the transition from adolescence to emerging adulthood, 79 (1), 173-183;

5. Yogesh K. Dwivedi, D.Laurie Hughes, J. Carlson (2021). Setting the future of digital and social media marketing research: Perspectives and reseach propositions [Електронний ресурс] – режим доступу до ресурсу: https://www.sciencedirect.com/science/article/pii/S0268401220308082#bib0610 ;

6. Socket.IO vs WebSocket: Keys Differences [Електронний ресурс] – режим доступу до ресурсу: https://apidog.com/articles/socket-io-vs-websocket/

7. Using Social Media Marketing in the Digital Era: A Necessity or a Choice, M.T. Khanom 12(3):88-98 [Електронний ресурс] – режим доступу до ресурсу:https://www.researchgate.net/publication/370578753_Using_Social_Media_Marketing_in_the_Digital_Era_A_Necessity_or_a_Choice ;

8. Офіційна документація WebSocket API (WebSockets) [Електронний ресурс] – режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API ;

9. Офіційна документація Socket.IO [Електронний ресурс] – режим доступу до ресурсу: https://socket.io/docs/v4/ ;

10. The Use of IDEF0 for the Design and Specification of Methodologies, by Adrien Presley Retrieved October, 1998, from https://www.researchgate.net/publication/2447898_The_Use_of_IDEF0_for_the_Design_and_Specification_of_Methodologies ;

11. Gary R. Waissi, M. Demir, Jane E. Humble, B.Lev , G. (2015). Automation of strategy using IDEF0 — A proof of concept from https://www.sciencedirect.com/science/article/pii/S2214716015000111 ;

12. How to Use MERN Stack: A Complete Guide — Mongo DB tutorial. (n.d.). Retrieved November 08, 2023, from https://www.mongodb.com/languages/mern-stack-tutorial ;

13. Vasav (2023). Client Server Architecture: Types, Examples, & Benefits from https://www.redswitches.com/blog/client-server-architecture/ ;

14. UML Use Case Diagram Tutorial IO [Електронний ресурс] – режим доступу до ресурсу: https://www.lucidchart.com/pages/uml-use-case-diagram ;

15. Allison Lynch (2023). UML Use Case Diagram Symbols [Електронний ресурс] – режим доступу до ресурсу: https://www.edrawsoft.com/uml-use-case-symbols.html ;

16. Richard Makara (2022). Data Modeling for Business Intelligence: Optimizing Decision Making [Електронний ресурс] – режим доступу до ресурсу: https://reconfigured.io/blog/data-modeling-for-business-intelligence-optimizing-decision-making ;

17. David Taylor (October 27, 2023). What is Data Modelling? Types (Conceptual, Logical, Physical) [Електронний ресурс] – режим доступу до ресурсу: https://www.guru99.com/data-modelling-conceptual-logical.html ;

18. Paolo Kukhnavets (January 12, 2023). Work Breakdown Structure Examples (WBS) that You Can Use as References in 2023 [Електронний ресурс] – режим доступу до ресурсу: https://blog.ganttpro.com/en/work-breakdown-structure-example-wbs/ ;

19. Diana Ramos (December 8, 2021). Convert Work Breakdown Structures to Gantt Charts for Project Success [Електронний ресурс] – режим доступу до ресурсу: https://www.smartsheet.com/content/wbs-gantt ;

20. Diana Ramos (May 3, 2021). The Advantages and Limitations of Gantt Charts in Project Management [Електронний ресурс] – режим доступу до ресурсу: https://www.smartsheet.com/content/gantt-chart-pros-cons .

21. Real-time communication with WebSocket and Node.js Електронний ресурс] – режим доступу до ресурсу: https://www.merixstudio.com/blog/real-time-communication-websockets-nodejs/

22. Illustration about MERN stack Електронний ресурс] – режим доступу до ресурсу: https://medium.com/techiepedia/what-exactly-a-mern-stack-is-60c304bffbe4

ДОДАТОК А

ПЛАНУВАННЯ РОБІТ

Деталізація мети проекту методом SMART. Продуктом дипломного проекту є інформаційна технологія підтримки взаємодії користувачів у веб-соціальній мережі. Результати деталізації методом SMART розміщені у табл. А.1.

Таблиця А.1 − Деталізація мети методом SMART

Specific (конкретна)	Створити інформаційну технологію підтримки взаємодії користувачів у веб-соціальній мережі розробити веб-додаток для демонстрації роботи представленої технології.
Measurable (вимірювана)	Результатом роботи є розроблена інформаційна технологія підтримки взаємодії користувачів в веб-соціальній мережі, що надає можливість користувачу взаємодіяти з особистою сторінкою та спілкуватися з іншими користувачами
Achievable (досяжна)	Для виконання проекту наявні необхідні знання стеку технологій React.js, Express.js, Node.js, мову програмування JavaScript, баз даних MySQL та Firebase (Google) та навичок написання документації. Мета є досяжною, оскільки вона враховує можливості технічних ресурсів та експертів, необхідних для створення та впровадження інформаційної технології та веб-додатку.
Relevant (реалістична)	Розроблена технологія дозволить покращити процес взаємодії користувачів в системі шляхом оптимізації запитів до БД, хешуванням даних, та обробки даних з

	використанням технології WebSocket.IO. Результат роботи сприятиме поліпшенню взаємодії користувачів.

Продовження таблиці А.1 − Деталізація мети методом SMART

Time-framed (обмежена у часі)	У проекті передбачений конкретний термін для створення та впровадження технології та веб-додатку. Це дозволяє визначити та контролювати графік робіт і забезпечити своєчасне завершення проекту.

Джерело: побудовано автором

Планування змісту структури робіт IT-проекту (WBS).

Work Breakdown Structure також відома як WBS надає можливість створення структури шляхом розбиття робіт − полегшити управління великими, багатоетапними проектами. Це досягається шляхом розбиття проектів на менші завдання та процеси. Ці окремі завдання можуть виконуватися і завершуватися одночасно різними командами або членами команди, що сприяє швидшому та ефективнішому виконанню проекту.

Структура розбиття робіт має вигляд детальної діаграми, яка визначає і розбиває результати і пов'язані з ними заходи, необхідні для завершення. Основний, кінцевий результат знаходиться у верхній частині діаграми, а під ним розташовані підзадачі. WBS часто використовується в поєднанні з діаграмою Ганта та програмним забезпеченням для управління IT-проектами, щоб покращити планування та виконання складних проектів [18].

Структурна діаграма дерева розбиття робіт, також звана WBS-діаграмою, відображає кожен рівень WBS через пов'язану мережеву діаграму. Сам проект знаходиться у верхній частині діаграми. Кожен підрівень під ним відображає ієрархію WBS. Як в Primavera P6, так і в Microsoft Project, деревоподібна діаграма

не відображається автоматично під час створення і перегляду WBS. Однак обидві програми дозволяють перемикати відображення WBS при необхідності.

WBS діаграма проекту зображена на рисунку А.1.

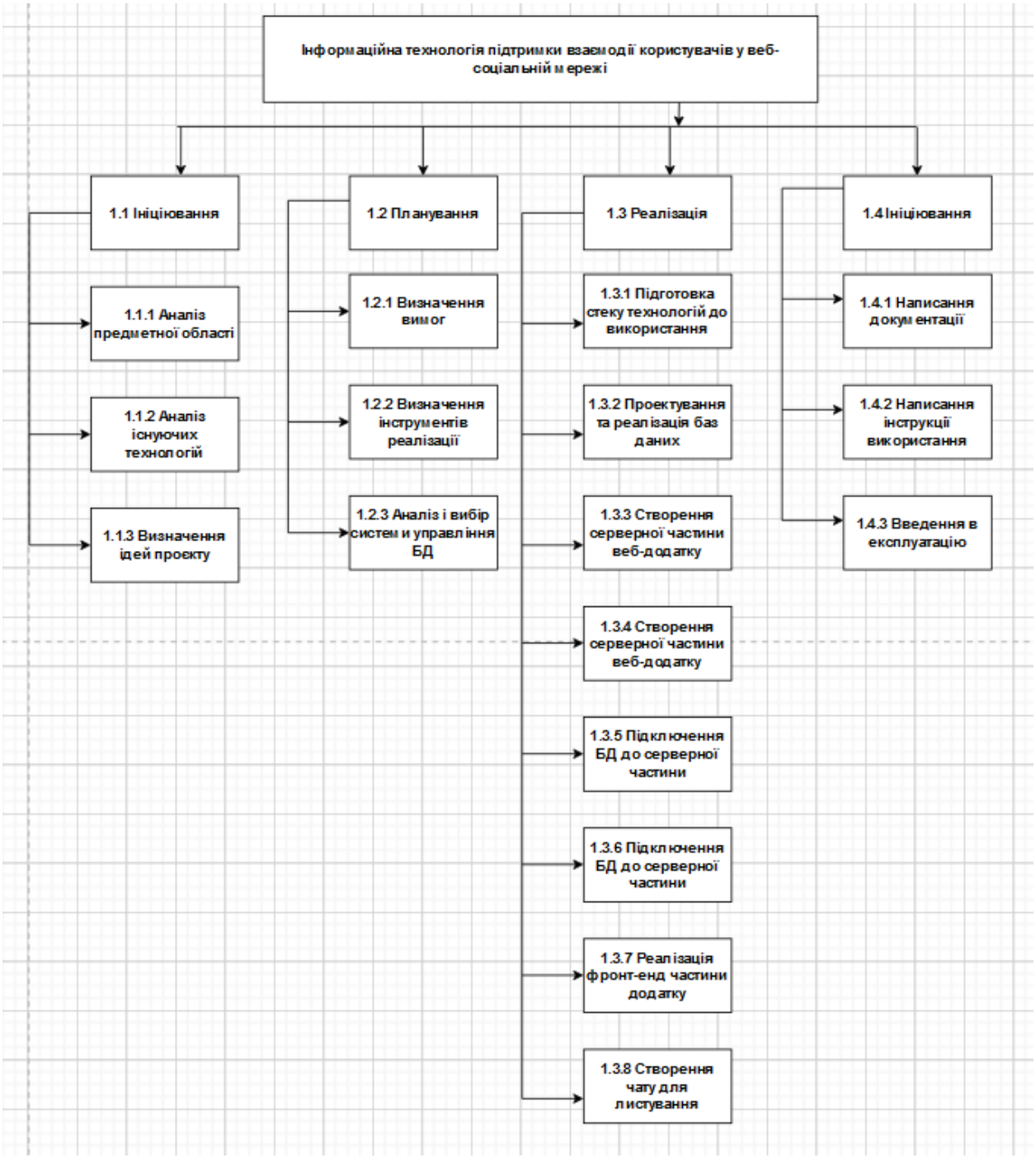


Рисунок А.1 − WBS структура проекту

Організаційна структура проекту (OBS)

Організаційна структура проекту (Organization Breakdown Structure, OBS) – це ієрархічна модель, що описує встановлену організаційну структуру для планування проекту, управління ресурсами, відстеження часу і витрат, розподілу витрат, звітності про доходи/прибуток та управління роботами.

Структура розбиття робіт (WBS) фіксує всі елементи проектів в організованому вигляді. Розбиття великих, складних проектів на менші частини забезпечує кращу основу для організації та управління поточними і майбутніми проектами. WBS полегшує розподіл ресурсів, призначення завдань, вимірювання і контроль вартості проекту та виставлення рахунків. WBS використовується на початку проекту для визначення обсягу робіт, визначення центрів витрат і є відправною точкою для розробки проектних планів/діаграм Ганта [19].

Організаційна структура проекту групує подібні проектні роботи або "робочі пакети" і пов'язує їх зі структурою організації. OBS (також відома як організаційна структура) використовується для визначення відповідальності за управління проектами, звітність про витрати, виставлення рахунків, бюджетування та контроль за проектами. OBS надає організаційну, а не засновану на завданнях перспективу проекту. Ієрархічна структура OBS дозволяє агрегувати (згортати) інформацію про проект на вищі рівні. Коли визначені проектні обов'язки і призначені роботи, OBS і WBS з'єднуються, забезпечуючи можливість потужної аналітики для вимірювання ефективності проекту і роботи персоналу на дуже високому рівні (наприклад, продуктивність бізнес-підрозділу) або до деталей (наприклад, робота користувача над завданням). OBS структура проекту наведена на рисунку А.2.

Рисунок А.2. – OBS структура проекту

Побудова календарного графіка виконання IT-проекту

Діаграма Ганта використовує часові шкали для відображення завершення кожного завдання в проекті. Ці часові шкали показують, як завдання пов'язані між собою. Діаграми Ганта корисні тим, що вони дають загальне уявлення про діяльність і прогрес проєкту.

Діаграми Ганта показують, як робота над завданням може початися тільки після завершення пов'язаного з ним завдання. Така інформація допомагає менеджерам зрозуміти перешкоди в проєкті та приймати важливі рішення щодо найкращого способу просування проєкту вперед. [20].

За допомогою програми Microsoft Project Professional 2021 побудовано діаграму Ганта (рис. А.3 − А.4).

Інформаційна технологія підтримки взаємодії користувачів у веб-соціальній мережі	58 дней	Вт 12.09.23	Пт 01.12.23			
Ініціалізація проекту	2 дней	Вт 12.09.23	Ср 13.09.23			
Опрашювання ТЗ	1 день	Чт 14.09.23	Чт 14.09.23		Сулейманов Ф. М.	
Аналіз сервісів	1 день	Пт 15.09.23	Пт 15.09.23	3	Сулейманов Ф. М.	$10000
Пошук контенту	1 день	Сб 16.09.23	Сб 16.09.23	4	Сулейманов Ф. М.	$1,116.00
Ідентифікація проекту	1 день	Вс 17.09.23	Вс 17.09.23	5	Марченко А.В.	$72.00
Планування проекту	**15 дней**	Пн 18.09.23	Пт 06.10.23	6		$72.00
Структура проекту	2 дней	Ср 18.10.23	Чт 19.10.23	7	Сулейманов Ф. М.	$72.00
Розробка дизайну та макету web-соціальної мережі	2 дней	Пт 20.10.23	Пн 23.10.23	8	Сулейманов Ф. М.	$900.00
Розробка календарного плану «інформаційна технологія підтримки взаємодії користувачів у веб-соціальній мережі»	5 дней	Пн 23.10.23	Пт 27.10.23	9	Сулейманов Ф. М.	$1,432.00
Управління ресурсами	1 день	Сб 28.10.23	Сб 28.10.23	10	Сулейманов Ф. М.	$72.00
Управління фінансами	1 день	Вс 29.10.23	Вс 29.10.23	11	Сулейманов Ф. М.	$72.00
Управління якістю	1 день	Пн 30.10.23	Пн 30.10.23	12	Сулейманов Ф. М.	$100.00
Оптимізація плану	1 день	Вт 31.10.23	Вт 31.10.23	13	Сулейманов Ф. М.	$72.00
Затвердження плану	1 день	Ср 01.11.23	Ср 01.11.23	14	Марченко А.В.	$72.00
Реалізація проекту	15 дней	Вс 01.10.23	Чт 19.10.23		Сулейманов Ф. М.	$7,020.00
Розробка інформаційної технології	5 дней	Ср 01.11.23	Вт 07.11.23		Сулейманов Ф. М.	$3,250.00
Реалізація дизайну та макету	4 дней	Вт 07.11.23	Пт 10.11.23		Сулейманов Ф. М.	$1,300.00
Розробка Front-end частини	2 дней	Пт 10.11.23	Пн 13.11.23	18	Сулейманов Ф. М.	$1,500.00

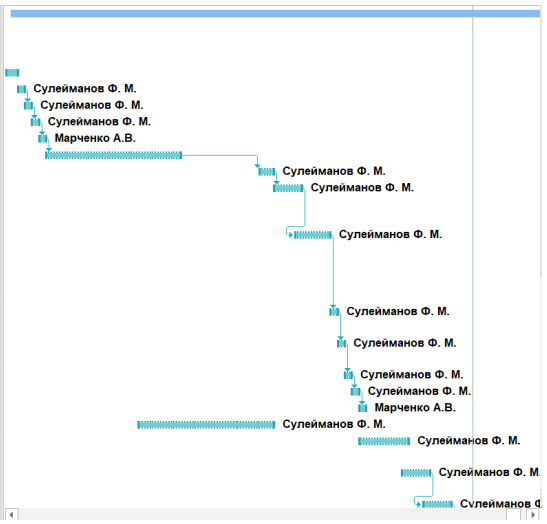


Рисунок А.3. – Діаграма Ганта, частина 1

Управління якістю	1 день	Пн 30.10.23	Пн 30.10.23	12	Сулейманов Ф. М.	$100.00
Оптимізація плану	1 день	Вт 31.10.23	Вт 31.10.23	13	Сулейманов Ф. М.	$72.00
Затвердження плану	1 день	Ср 01.11.23	Ср 01.11.23	14	Марченко А.В.	$72.00
Реалізація проекту	15 дней	Вс 01.10.23	Чт 19.10.23		Сулейманов Ф. М.	$7,020.00
Розробка інформаційної технології	5 дней	Ср 01.11.23	Вт 07.11.23		Сулейманов Ф. М.	$3,250.00
Реалізація дизайну та макету	4 дней	Вт 07.11.23	Пт 10.11.23		Сулейманов Ф. М.	$1,300.00
Розробка Front-end частини	2 дней	Пт 10.11.23	Пн 13.11.23	18	Сулейманов Ф. М.	$1,500.00
Розробка Back-end частини	5 дней	Пн 13.11.23	Пт 17.11.23	19	Сулейманов Ф. М.	$350.00
Проектування БД	3 дней	Пт 17.11.23	Вт 21.11.23	20	Сулейманов Ф. М.	$100.00
Створення компонентів	1 день	Вт 21.11.23	Вт 21.11.23	21	Сулейманов Ф. М.	$100.00
Створення зв'язку між користувачем та додатком	**5 дней**	Вт 21.11.23	Пн 27.11.23		Сулейманов Ф. М.	$3,270.00
Розробка модулю створення посту з завантаженням фото	1 день	Вт 21.11.23	Вт 21.11.23		Сулейманов Ф. М.	$270.00
Розробка модулів реєстрації та авторизації нового користувача	1 день	Ср 22.11.23	Ср 22.11.23	24	Сулейманов Ф. М.	$1,500.00
Розробка модулю створення чату	2 дней	Чт 23.11.23	Пт 24.11.23	25	Сулейманов Ф. М.	$1,500.00
Реалізація зв'язку БД з додатком	2 дней	Пт 24.11.23	Пн 27.11.23	26	Сулейманов Ф. М.	$500.00
Тестування	2 дней	Пн 27.11.23	Вт 28.11.23	27	Сулейманов Ф. М.	$500.00
Завершення проекту	6 дней	Вт 28.11.23	Вт 05.12.23	28	Сулейманов Ф. М.	$432.00
Створення документації	1 день	Вт 28.11.23	Вт 28.11.23	29	Сулейманов Ф. М.	$72.00
Створення інструкції користувача	2 дней	Ср 29.11.23	Чт 30.11.23	30	Сулейманов Ф. М.	$72.00
Запуск і підтримка	2 дней	Сб 30.12.23	Пн 01.01.24	31	Сулейманов Ф. М.	$72.00

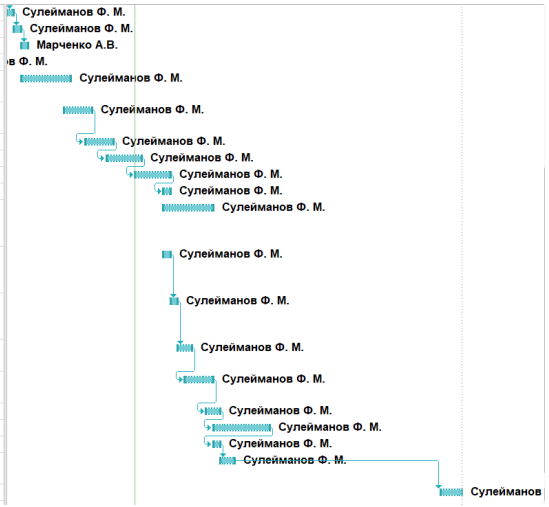


Рисунок А.4 – Діаграма Ганта, частина 2

Управління ризиками. Ризик визначається як можливість будь-якої негативної події, яка може статися через зовнішні або внутрішні фактори, і яку можна зменшити за допомогою превентивних дій. Всі проекти схильні до ризиків. Насправді існує нескінченна кількість речей, які можуть завадити вам досягти поставлених цілей під час роботи над проектом. Управління ризиками мінімізує ті загрози, які можуть призвести до провалу проекту, і дозволяє контролювати графік, бюджет і вимоги до якості проекту.

Таблиця А.2. Ймовірність виникнення і величина ризику

№	Ризики	Виникнення	Втрати
1	Проблеми з безпекою та конфіденційністю даних	2	4
2	Активність шкідливих користувачів і спам	3	3
3	Технічні неполадки і відмова серверу	3	5
4	Недостатня масштабованість	2	3

5	Питання з юридичною відповідальністю і правовими обмеженнями	4	5
6	Відсутність резервних копій даних	2	3

Таблиця А.3 – Матриця впливу

Вірогідність виникнення	Матриця впливу				
5			3	5	
4		4			
3		6	2		
2				1	
1					
Ступінь впливу	1	2	3	4	5

ДОДАТОК Б

Лістинг коду «auth.js»

```
import { db } from "../connect.js";
import bcrypt from "bcryptjs";
import jwt from "jsonwebtoken";

export const register = (req, res) => {


  const q = "SELECT * FROM users WHERE username = ?";

  db.query(q, [req.body.username], (err, data) => {
    if (err) return res.status(500).json(err);
    if (data.length) return res.status(409).json("User
already exists!");
    //CREATE A NEW USER
    //Hash the password
    const salt = bcrypt.genSaltSync(10);
    const hashedPassword =
bcrypt.hashSync(req.body.password, salt);

    const q =
```

```
      "INSERT INTO users
(`username`,`email`,`password`,`name`) VALUE (?)";

    const values = [
      req.body.username,
      req.body.email,
      hashedPassword,
      req.body.name,
    ];

    db.query(q, [values], (err, data) => {
      if (err) return res.status(500).json(err);
      return res.status(200).json("User has been
created.");
    });
  });
};

export const login = (req, res) => {
  const q = "SELECT * FROM users WHERE username = ?";

  db.query(q, [req.body.username], (err, data) => {
    if (err) return res.status(500).json(err);
```

```
    if (data.length === 0) return
res.status(404).json("User not found!");

    const checkPassword = bcrypt.compareSync(
      req.body.password,
      data[0].password
    );

    if (!checkPassword)
      return res.status(400).json("Wrong password or
username!");

    const token = jwt.sign({ id: data[0].id },
"secretkey");

    const { password, ...others } = data[0];

    res
      .cookie("accessToken", token, {
        httpOnly: true,
      })
      .status(200)
      .json(others);
```

```
  });
};

export const logout = (req, res) => {
  res
    .clearCookie("accessToken", {
      secure: true,
      sameSite: "none",
    })
    .status(200)
    .json("User has been logged out.");
};
```

ДОДАТОК В

Лістинг коду «App.js»

```
import Login from "./pages/login/Login";
import Register from "./pages/register/Register";
import {
  createBrowserRouter,
  RouterProvider,
  Outlet,
  Navigate,
} from "react-router-dom";
import Navbar from "./components/navbar/Navbar";
import LeftBar from "./components/leftBar/LeftBar";
import RightBar from "./components/rightBar/RightBar";
import Home from "./pages/home/Home";
import Profile from "./pages/profile/Profile";
import "./style.scss";
import { useContext } from "react";
import { DarkModeContext } from
"./context/darkModeContext";
import { AuthContext } from "./context/authContext";
import { QueryClient, QueryClientProvider } from
"@tanstack/react-query";
```

```
function App() {
  const { currentUser } = useContext(AuthContext);

  const { darkMode } = useContext(DarkModeContext);

  const queryClient = new QueryClient();

  const Layout = () => {
    return (
      <QueryClientProvider client={queryClient}>
        <div className={`theme-${darkMode ? "dark" :
"light"}`}>
          <Navbar />
          <div style={{ display: "flex" }}>
            <LeftBar />
            <div style={{ flex: 6 }}>
              <Outlet />
            </div>
            <RightBar />
          </div>
        </div>
      </QueryClientProvider>
```

```
  );
};

const ProtectedRoute = ({ children }) => {
  if (!currentUser) {
    return <Navigate to="/login" />;
  }

  return children;
};

const LoggedInRoute = ({ children }) => {
  if (currentUser) {
    return <Navigate to="/" />;
  }

  return children;
};

const router = createBrowserRouter([
  {
    path: "/",
```

```
    element: (
      <ProtectedRoute>
        <Layout />
      </ProtectedRoute>
    ),
    children: [
      {
        path: "/",
        element: <Home />,
      },
      {
        path: "/profile/:id",
        element: <Profile />,
      },
    ],
  },
  {
    path: "/login",
    element: (
      <LoggedInRoute>
        <Login />
      </LoggedInRoute>
```

```
      ),
    },
    {
      path: "/register",
      element: <Register />,
    },
  ]);

  return (
    <div>
      <RouterProvider router={router} />
    </div>
  );
}

export default App;
```

ДОДАТОК Г

Лістинг коду «style.scss»

```
$themes: (
  light: (
    textColor: #000,
    bg: white,
    logo: darkblue,
    bgSoft: #f6f3f3,
    textColorSoft: #555,
    border: lightgray,
  ),
  dark: (
    textColor: whitesmoke,
    bg: #222,
    logo: white,
    bgSoft: #333,
    textColorSoft: lightgray,
    border: #444,
  ),
);

@mixin themify($themes) {
```

```
  @each $theme, $map in $themes {
    .theme-#{$theme} & {
      $theme-map: () !global;
      @each $key, $submap in $map {
        $value: map-get(map-get($themes, $theme),
"#{$key}");
        $theme-map: map-merge(
          $theme-map,
          (
            $key: $value,
          )
        ) !global;
      }
      @content;
      $theme-map: null !global;
    }
  }
}

@function themed($key) {
  @return map-get($theme-map, $key);
}
```

```
@mixin mobile {
  @media (max-width: 480px) {
    @content;
  }
}
@mixin tablet {
  @media (max-width: 960px) {
    @content;
  }
}
```

ДОДАТОК Д

Лістинг коду «app.js»

```
const express = require('express');
const bcryptjs = require('bcryptjs');
const jwt = require('jsonwebtoken');
const cors = require('cors');
const io = require('socket.io')(8080, {
    cors: {
        origin: 'http://localhost:3002',
    }
});

// Connect DB
require('./db/connection');

// Import Files
const Users = require('./models/Users');
const Conversations = require('./models/Conversations');
const Messages = require('./models/Messages');

// app Use
const app = express();
```

```
app.use(express.json());
app.use(express.urlencoded({ extended: false }));
app.use(cors());

const port = process.env.PORT || 8000;

// Socket.io
let users = [];
io.on('connection', socket => {
    console.log('User connected', socket.id);
    socket.on('addUser', userId => {
        const isUserExist = users.find(user => user.userId
=== userId);
        if (!isUserExist) {
            const user = { userId, socketId: socket.id };
            users.push(user);
            io.emit('getUsers', users);
        }
    });

    socket.on('sendMessage', async ({ senderId,
receiverId, message, conversationId }) => {
```

```
        const receiver = users.find(user => user.userId
=== receiverId);
        const sender = users.find(user => user.userId ===
senderId);
        const user = await Users.findById(senderId);
        console.log('sender :>> ', sender, receiver);
        if (receiver) {

io.to(receiver.socketId).to(sender.socketId).emit('getMess
age', {
                senderId,
                message,
                conversationId,
                receiverId,
                user: { id: user._id, fullName:
user.fullName, email: user.email }
            });
            }else {
                io.to(sender.socketId).emit('getMessage',
{
                    senderId,
                    message,
                    conversationId,
                    receiverId,
```

```
                    user: { id: user._id, fullName:
user.fullName, email: user.email }
                });
            }
        });

    socket.on('disconnect', () => {
        users = users.filter(user => user.socketId !==
socket.id);
        io.emit('getUsers', users);
    });
    // io.emit('getUsers', socket.userId);
});

// Routes
app.get('/', (req, res) => {
    res.send('Welcome');
})

app.post('/api/register', async (req, res, next) => {
    try {
        const { fullName, email, password } = req.body;
```

```
        if (!fullName  !email  !password) {
            res.status(400).send('Please fill all required
fields');
        } else {
            const isAlreadyExist = await Users.findOne({
email });
            if (isAlreadyExist) {
                res.status(400).send('User already
exists');
            } else {
                const newUser = new Users({ fullName,
email });
                bcryptjs.hash(password, 10, (err,
hashedPassword) => {
                    newUser.set('password',
hashedPassword);
                    newUser.save();
                    next();
                })
                return res.status(200).send('User
registered successfully');
            }
        }
```

```
    } catch (error) {
        console.log(error, 'Error')
    }
})

app.post('/api/login', async (req, res, next) => {
    try {
        const { email, password } = req.body;

        if (!email || !password) {
            res.status(400).send('Please fill all required
fields');
        } else {
            const user = await Users.findOne({ email });
            if (!user) {
                res.status(400).send('User email or
password is incorrect');
            } else {
                const validateUser = await
bcryptjs.compare(password, user.password);
                if (!validateUser) {
                    res.status(400).send('User email or
password is incorrect');
```

```
                } else {
                    const payload = {
                        userId: user._id,
                        email: user.email
                    }
                    const JWT_SECRET_KEY =
process.env.JWT_SECRET_KEY || 'THIS_IS_A_JWT_SECRET_KEY';
jwt.sign(payload, JWT_SECRET_KEY, { expiresIn: 84600 },
async (err, token) => {
                        await Users.updateOne({ _id:
user._id }, {
                            $set: { token }
                        })
                        user.save();
                        return res.status(200).json({
user: { id: user._id, email: user.email, fullName:
user.fullName }, token: token })
                    })
                }
            }
        }

    } catch (error) {
```

```
        console.log(error, 'Error')
    }
})

app.post('/api/conversation', async (req, res) => {
    try {
        const { senderId, receiverId } = req.body;
        const newCoversation = new Conversations({
members: [senderId, receiverId] });
        await newCoversation.save();
        res.status(200).send('Conversation created
successfully');
    } catch (error) {
        console.log(error, 'Error')
    }
})

app.get('/api/conversations/:userId', async (req, res) =>
{
    try {
        const userId = req.params.userId;
        const conversations = await Conversations.find({
members: { $in: [userId] } });
```

```
        const conversationUserData =
Promise.all(conversations.map(async (conversation) => {
            const receiverId =
conversation.members.find((member) => member !== userId);
            const user = await Users.findById(receiverId);
            return { user: { receiverId: user._id, email:
user.email, fullName: user.fullName }, conversationId:
conversation._id }
        }))
        res.status(200).json(await conversationUserData);
    } catch (error) {
        console.log(error, 'Error')
    }
})

app.post('/api/message', async (req, res) => {
    try {
        const { conversationId, senderId, message,
receiverId = '' } = req.body;
        if (!senderId || !message) return
res.status(400).send('Please fill all required fields')
        if (conversationId === 'new' && receiverId) {
            const newCoversation = new Conversations({
members: [senderId, receiverId] });
```

```
            await newCoversation.save();
            const newMessage = new Messages({
conversationId: newCoversation._id, senderId, message });
            await newMessage.save();
            return res.status(200).send('Message sent
successfully');
        } else if (!conversationId && !receiverId) {
            return res.status(400).send('Please fill all
required fields')
        }
        const newMessage = new Messages({ conversationId,
senderId, message });
        await newMessage.save();
        res.status(200).send('Message sent successfully');
    } catch (error) {
        console.log(error, 'Error')
    }
})

app.get('/api/message/:conversationId', async (req, res)
=> {
    try {
        const checkMessages = async (conversationId) => {
```

```
            console.log(conversationId, 'conversationId')
            const messages = await Messages.find({
conversationId });
            const messageUserData =
Promise.all(messages.map(async (message) => {
                const user = await
Users.findById(message.senderId);
                return { user: { id: user._id, email:
user.email, fullName: user.fullName }, message:
message.message }
            }));
            res.status(200).json(await messageUserData);
        }
        const conversationId = req.params.conversationId;
        if (conversationId === 'new') {
            const checkConversation = await
Conversations.find({ members: { $all: [req.query.senderId,
req.query.receiverId] } });
            if (checkConversation.length > 0) {
                checkMessages(checkConversation[0]._id);
            } else {
                return res.status(200).json([])
            }
        } else {
```

```
            checkMessages(conversationId);
        }
    } catch (error) {
        console.log('Error', error)
    }
})
app.get('/api/users/:userId', async (req, res) => {
    try {
        const userId = req.params.userId;
        const users = await Users.find({ _id: { $ne:
userId } });
        const usersData = Promise.all(users.map(async
(user) => {
            return { user: { email: user.email, fullName:
user.fullName, receiverId: user._id } }
        }))
        res.status(200).json(await usersData);
    } catch (error) {
        console.log('Error', error)
    }
})

app.listen(port, () => {
```

```
    console.log('listening on port ' + port);
})
```