МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА СОЦІАЛЬНОЇ МЕРЕЖІ ДЛЯ ПРИЙНЯТТЯ РІШЕНЬ З ВИКОРИСТАННЯМ REACT JS, NODE JS ТА MONGO DB»**

Виконав: студент 4 курсу, групи 6.1210-2пі
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми програмна інженерія
(назва освітньої програми)
А.А. Горевий
(ініціали та прізвище)
Керівник доцент кафедри програмної інженерії, PhD Чопорова О.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент доцент кафедри фундаментальної та прикладної математики, доцент, к.ф.-м.н. Панасенко Є.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти бакалавр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма програмна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2023 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Горевому Артуру Андрійовичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка соціальної мережі для прийняття рішень з використанням React JS, Node JS та Mongo DB

керівник роботи Чопорова Оксана Володимирівна, PhD
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 21 » грудня 2023 року № 2180-с

2. Строк подання студентом роботи 03.06.2024 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Розробка соціальної мережі для прийняття рішень з використанням React JS, Node JS та Mongo DB.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація за темою докладу

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 25.12.2023 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	12.01.2024	
2.	Збір вихідних даних.	09.02.2024	
3.	Обробка методичних та теоретичних джерел.	08.03.2024	
4.	Розробка першого та другого розділу.	12.04.2024	
5.	Розробка третього розділу.	10.05.2024	
6.	Оформлення та нормоконтроль	27.05.2024	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	18.06.2024	

Студент _____ А.А. Горевий
(підпис) (ініціали та прізвище)

Керівник роботи _____ О.В. Чопорова
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка соціальної мережі для прийняття рішень з використанням React JS, Node JS та Mongo DB»: 65 с., 34 рис., 1 табл., 17 джерел, 1 додаток.

БАЗА ДАНИХ, ГОЛОСУВАННЯ, СОЦМЕРЕЖА, BACK-END, EXPRESS JS, FRONT-END, JAVA SCRIPT, MERN, MONGO DB, NODE JS, REACT JS.

Об’єкт дослідження: розробка соціальної мережі для прийняття рішень з використанням React JS, Node JS та Mongo DB.

Мета роботи: розробити соціальну мережу для прийняття рішень на основі голосування між двома зображеннями з використанням React JS, Node JS та Mongo DB.

Методи дослідження: моделювання, проєктування, програмний, аналітичний.

Результати та особливості додатку: було розроблено соцмережу за допомогою стеку технологій «MERN».

Ключовою особливістю цієї соцмережі є те що пост можна робити лише на основі двох фото і віповідно лайкнути можна тільки одну з них, таким чином відбувається голосування між цими фото.

Переваги розробленої соцмережі:

– простота використання: голосування за одну з двох фотографій є інтуїтивно зрозумілим і простим процесом, який не потребує додаткових пояснень;
– підвищена залученість: голосування за фото стимулює активну взаємодію з контентом, роблячи його більш цікавим та привабливим для користувачів;
– ефективний інструмент: соцмережа може допомогти авторам краще зрозуміти свою цільову аудиторію та збирати цінні відгуки.

SUMMARY

Bachelor’s qualifying paper «Development of a Social Network for Decision-making Using React JS, Node JS and Mongo DB»: 65 pages, 34 figures, 1 table, 17 references, 1 supplement.

SOCIAL NETWORK, BACK-END, DATABASE, EXPRESS JS, FRONT-END, JAVA SCRIPT, MERN, MONGO DB, NODE JS, REACT JS, VOTING.

Object of research: development of a social network for decision making using React JS, Node JS, and Mongo DB.

Purpose: to develop a social network for decision making based on voting between two images using React JS, Node JS and Mongo DB.

Research methods: modeling, projecting, programming, analytical.

Results and features of the application: a social network was developed using the MERN technology stack.

The key feature of this social network is that a post can be made only on the basis of two photos and only one of them can be liked, so there is way to vote between these photos.

The advantages of the developed social network:

– ease of use: voting for one of the two photos is an intuitive and simple process that does not require any additional explanation;
– increased engagement: voting for a photo stimulates active interaction with the content, making it more interesting and attractive to users;
– effective tool: the social network can help authors better understand their target audience and collect valuable feedback.

ЗМІСТ

ВСТУП

Соціальні мережі стали невід'ємною частиною сучасного суспільства. Вони активно впливають на різні аспекти нашого життя, починаючи від особистого спілкування і закінчуючи бізнесом та політикою. Актуальність соціальних мереж важко переоцінити, оскільки вони стали основним каналом комунікації, інформації та розваг для мільйонів людей по всьому світу.

Для багатьох компаній соціальні мережі стали основним інструментом маркетингу та просування бренду. Вони дозволяють взаємодіяти з клієнтами, аналізувати їх поведінку та вподобання, проводити рекламні кампанії та підвищувати впізнаваність бренду. Крім того, соцмережі сприяють розвитку малого бізнесу, надаючи платформу для продажу товарів і послуг.

Соціальні мережі відіграють значну роль у політичному житті. Вони використовуються для мобілізації громадськості, поширення політичних ідей та проведення виборчих кампаній. Однак, цей вплив також може мати негативні наслідки, включаючи втручання у вибори та поширення пропаганди.

Мета: спроєктувати та розробити соцмережу для прийняття рішень з використанням стеку MERN (Mongo DB, Express JS, React JS та Node JS).

Об'єктом дослідження є засоби стеку MERN (Mongo DB, Express JS, React JS та Node JS).

Предмет дослідження: створення соцмережі, для прийняття рішень.

Методи дослідження – моделювання, проєктування, програмний, аналітичний.

В першому розділі кваліфікаційної роботи показано опис проєкту, визначення до деяких термінів, а також опис інструментів розробки соцмережі.

В другому розділі показано етапи проєктування соцмережі, а також, приведено опис прецедентів системи, діаграму бази даних.

В третьому розділі показано етапи реалізації та тестування соцмережі.

1 ТЕОРЕТИЧНІ ВІДОМОСТІ

1.1 Опис проєкту

Voteween – це інноваційна соціальна мережа, створена за допомогою стеку MERN, яка надає користувачам можливість голосувати між двома варіантами за допомогою фотографій. Головна особливість Voteween полягає в тому, що кожен пост містить лише два фото, між якими користувачі можуть обрати, шляхом лайку одного з фото. Це створює динамічний та інтерактивний спосіб збору думок та вподобань.

У сучасному світі соціальні мережі відіграють ключову роль у комунікації, зборі інформації та формуванні громадської думки. Voteween підсилює цей аспект, пропонуючи простий і ефективний спосіб проведення опитувань. У великих корпораціях Voteween може бути інструментом для вибору продуктів або дизайнів на основі реальних вподобань користувачів. Користувачі можуть допомагати компаніям приймати важливі рішення, висловлюючи свої думки через голосування.

Політичні аспекти також знаходять своє місце у Voteween. За допомогою цієї платформи можна організовувати голосування між кандидатами або політичними програмами.

Перед участю у таких голосуваннях користувачі повинні пройти перевірку на актуальність їхніх державних документів, що забезпечує достовірність результатів та мінімізує можливість фальсифікацій.

Voteween створює простір для чесного та відкритого вираження думок, забезпечуючи при цьому захищеність та конфіденційність даних користувачів. Це робить платформу не лише корисною, але й актуальною у контексті сучасного суспільства, де швидкий обмін інформацією та прозорість мають вирішальне значення.

1.2 Визначення до термінів

Соціальна мережа – це мережа людей, які зустрічаються в Інтернеті для спілкування, розміщуючи інформацію та зображення, залишаючи коментарі чи надсилаючи повідомлення. Учасники можуть розширити свої особисті та ділові контакти, зв’язавшись з іншими на вебсайтах соціальних мереж та в додатках [1].

Користувач – людина, зареєстрована в системі з певними правами доступу до функціональності веб додатку.

Адміністратор – користувач системи, з можливістю редагування системи (Тобто створення (Create), перегляд (Read), зміну (Update) та видалення (Delete) (CRUD) користувачів, постів, коментарів і т.д.).

База даних (БД) – це організована структура, призначена для зберігання, зміни й обробки взаємопов’язаної інформації, переважно великих обсягів. Бази даних активно використовують для динамічних сайтів зі значними обсягами даних – часто це інтернет-магазини, соцмережі, корпоративні сайти [2].

1.3 Інструменти розробки серверної частини соцмережі

Node.js – це середовище виконання JavaScript, яке дає змогу запускати код JavaScript на стороні сервера. Воно використовує рушій V8 від Google для виконання JavaScript-коду, що дає змогу виконувати операції вводу/виводу асинхронно й ефективно, працювати з мережею та файловою системою. Написані на Node.js додатки призначені для використання як вебсервери, фреймворки, інструменти для резервного копіювання та багато інших проєктів. Node.js також дає змогу використовувати пакетний менеджер npm для встановлення та керування залежностями.

Node.js має низку особливостей і переваг. Наприклад, Node.js

заснований на асинхронній моделі програмування, що дає змогу ефективно обробляти велику кількість одночасних запитів без блокування процесу. Це особливо корисно для розробки мережевих серверів і застосунків, які працюють із великим обсягом даних.

Також важливо врахувати, що Node.js використовує JavaScript, який є широко поширеною мовою програмування. Це означає, що розробники можуть використовувати одну й ту саму мову на фронтенді та бекенді своїх додатків, що робить розробку та підтримку коду простішою та ефективнішою.

Важливо, що Node.js підтримує модульність, що дозволяє розробникам створювати та використовувати модулі для повторного використання коду. Велика вибірка модулів доступна через пакетний менеджер npm, що робить розробку додатків швидшою та зручнішою [3].

Express.js – це мінімалістичний та гнучкий фреймворк для вебзастосунків, побудованих на Node.js. Він надає широкий набір функціональності, що дозволяє розробникам створювати надійні API та вебдодатки легко і швидко.

Основні особливості Express:

– API: Express має безліч допоміжних HTTP-методів та проміжних обробників, що дозволяє легко створювати надійні API;
– продуктивність: цей фреймворк забезпечує тонкий прошарок базової функціональності для вебзастосунків, не спотворюючи звичну та зручну функціональність Node.js [4].

Express-validator – це middleware для фреймворка Express.js, яке дозволяє валідувати та перевіряти дані в маршрутах вашого Express додатка [5].

MongoDB – це база даних NoSQL, яка відрізняється від MySQL і PostgreSQL тим, що вона використовує нереляційну модель даних. Замість традиційних таблиць і стовпців, MongoDB використовує структуру, подібну до формату даних JSON, що дозволяє зберігати дані в більш гнучкому форматі.

В MongoDB дані зберігаються у форматі BSON (Binary JSON), який дозволяє робити запити до бази даних за допомогою JSON-подібного

синтаксису. Це може бути дуже зручно для розробників, які працюють з JavaScript, адже формат даних дуже подібний до того, з яким вони звикли працювати [6].

Mongoose – це бібліотека для моделювання об’єктів бази даних MongoDB, призначена для асинхронної роботи. Вона встановлює зв’язок між MongoDB та середовищем виконання JavaScript Node.js. Mongoose надає простий, заснований на схемі підхід до моделювання даних додатків [7].

JSON Web Token (JWT) – це стандарт токена доступу на основі JSON, який стандартизований в RFC 7519. Зазвичай використовується для передачі даних для аутентифікації в клієнт-серверних програмах.

Токени виглядають як рядки з «URL-безпечних» символів, що кодують інформацію. Вони складаються з трьох складових, розділених крапками (для читабельності розміщуємо на окремих рядках (див. рис. 1.1):

– заголовок (Header) – перша частина JWT це рядок, що закодовує звичайний JavaScript об’єкт, який описує токен, а також використаний алгоритм хешування;
– корисне навантаження (Payload) – друга частина JWT формує основу токена. Довжина корисного навантаження пропорційна кількості даних, збережених у JWT. Загальне правило: зберігати мінімум у JWT;
– підпис (Signature) – третя і кінцева частина JWT це підпис, згенерований на основі заголовка та корисного навантаження. Використовується для перевірки JWT [8].

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9 // header
.eyJrZXkiOiJ2YWwiLCJpYXQiOjE0MjI2MDU0NDV9  // payload
.eUiabuiKv-8PYk2AkGY4Fb5KMZeorYBLw261JPQD5lM   // signature
```

Рисунок 1.1 – Структура соцмережі «Voteween»

Bcrypt – це функція хешування паролів, розроблена Нілсом Профосом та Девідом Маз’єресом на основі шифру Blowfish.

Сіль (salt): Bcrypt включає сіль для захисту від атак за допомогою райдужних таблиць. Сіль – це випадкове значення, яке додається до пароля перед хешуванням. Вона ускладнює роботу з райдужними таблицями та робить атаки перебором менш ефективними.

Адаптивність: Bcrypt є адаптивною функцією. Це означає, що з часом кількість ітерацій може збільшуватися, роблячи обчислення більш повільним. Навіть при зростаючій обчислювальній потужності вона залишається стійкою до атак перебором.

Хеш-функція: вхідні дані для Bcrypt – це рядок пароля (до 72 байт), числовий параметр вартості (cost) та 16-байтове (128-бітове) значення солі. Сіль зазвичай є випадковим значенням. Bcrypt використовує ці дані для обчислення 24-байтового (192-бітового) хешу.

Формат виводу: вихід Bcrypt має наступний формат: $2<a/b/x/y>$[cost]$[22 символи солі][31 символ хешу]. Наприклад, для вхідного пароля abc123xyz, вартості 12 та випадкової солі, вихід Bcrypt виглядає так: $2a$12$randomsalt$hashedvalue [9].

Multer – це middleware для Node.js, яке використовується для обробки multipart/form-data, зокрема для завантаження файлів. Воно побудоване на основі бібліотеки busboy для максимальної ефективності [10].

1.4 Інструменти розробки клієнтської частини соцмережі

React js – це інтерфейсна бібліотека, яка стала популярним середовищем для сучасної веброзробки у спільноті програмування JavaScript. Для чого потрібен react? У сучасному світі наші з вами дні важко уявити без мобільних і вебдодатків.

Діджиталізація розвивається з кожним днем і сьогодні вже майже не залишилося нічого, що може бути не оцифроване: від замовлення таксі та доставки їжі до проведення банківських транзакцій. І все це відбувається

завдяки ефективним платформам та інструментам, що забезпечують зручність роботи з користувачем. Однією з таких надійних інтерфейсних бібліотек є React.

Фреймворк React.js – це фреймворк і бібліотека JavaScript з відкритим вихідним кодом, розроблені та підтримувані Facebook і Instagram. Він використовується для швидкого та ефективного створення інтерактивних користувацьких інтерфейсів і вебдодатків із застосуванням значно меншої кількості коду, ніж під час використання звичайного JavaScript.

У React ми можемо розробити свої додатки та проєкти, створюючи компоненти, що повторно використовуються, які можна розглядати як незалежні блоки Lego. Ці компоненти являють собою окремі частини остаточного інтерфейсу, які в зібраному вигляді утворюють увесь призначений для користувача інтерфейс програми.

Основна роль React у застосунку – керування відображенням цього застосунку так само як буква V у шаблоні «модель-представлення-контролер» (MVC), забезпечуючи оптимальне та найефективніше виконання рендерингу. Замість того щоб розглядати весь призначений для користувача інтерфейс як єдине ціле, React.js пропонує розробникам розділити ці складні призначені для користувача інтерфейси на окремі компоненти, які можна використовувати повторно і які утворюють будівельні блоки всього призначеного для користувача інтерфейсу. При цьому платформа React JS поєднує в собі швидкість і ефективність JavaScript з більш ефективним методом маніпулювання DOM для швидкого рендерингу вебсторінок і створення високодинамічних вебдодатків [11].

Redux – це JavaScript-бібліотека, покликана спростити управління станом вашого вебдодатку. Її основне призначення полягає в тому, щоб зробити управління даними більш організованим і передбачуваним.

У центрі концепції Redux знаходиться сховище стану (Store). Це своєрідне сховище, де зібрані всі дані вашого застосунку. Однак важлива відмінність від традиційного підходу полягає в тому, що Redux пропонує

єдине централізоване сховище, доступне для всіх компонентів вашого застосунку. Це наче спільний банк даних, до якого можна звернутися з будь-якої точки застосунку.

Отже, цей інструмент дає нам змогу ефективно управляти даними, роблячи процес розроблення більш організованим і передбачуваним.

Принцип того, як працює redux, ґрунтується на таких концепціях:

– Application state: це об’єкт, який містить усі дані, необхідні для роботи програми;
– Actions: це об’єкти, які представляють зміни стану додатка;
– Reducers: це функції, які визначають, як стан програми змінюється у відповідь на дії [12].

React Router надає рішення для переключення та маршрутизації сторінок у React. Ви можете використовувати його для створення закладних URL-адрес для вебдодатків або для композиційного способу навігації в React Native [13].

1.5 Загальні вимоги

Загальні вимоги для розробки соціальної мережі “Voteween” на основі стеку MERN (MongoDB, Express.js, React.js, Node.js) можна розділити на кілька категорій: функціональні вимоги, нефункціональні вимоги, архітектурні вимоги, вимоги до безпеки та вимоги до продуктивності.

Функціональні вимоги.

Реєстрація та авторизація користувачів:

– можливість реєстрації нових користувачів;
– авторизація користувачів з використанням пошти та пароля;
– можливість зміни пароля.

Дані користувача:

– користувачі повинні мати можливість редагувати свої дані;
– можливість перегляду профілів інших користувачів.

Створення постів:

– можливість створення посту з двома фото;
– можливість додавання текстового опису до посту.

Голосування:

– користувачі можуть лайкнути лише одне з двох фото в пості;
– підрахунок голосів для кожного фото в реальному часі.

Коментарі:

– можливість коментування постів;
– можливість вподобання коментарів.

Нефункціональні вимоги:

– масштабованість: система має підтримувати велику кількість користувачів та постів;
– продуктивність: швидкий час завантаження сторінок та відповідей на запити;
– зручність користування: інтуїтивно зрозумілий інтерфейс користувача.

Архітектурні вимоги.

Стек технологій:

– Frontend: React.js;
– Backend: Node.js з використанням Express.js;
– база даних: MongoDB.

Вимоги до безпеки:

– аутентифікація та авторизація: використання JWT (JSON Web Token) для аутентифікації користувачів;
– захист даних: шифрування конфіденційної інформації (наприклад, паролів).

2 ПРОЄКТУВАННЯ СОЦМЕРЕЖІ

Проєктування застосунку перед його розробкою має критичне значення для успішної реалізації проєкту. Воно дозволяє створити чітке бачення кінцевого продукту, визначити ключові функціональні та нефункціональні вимоги, а також виявити можливі ризики та проблеми на ранніх етапах.

Перш за все, проєктування допомагає уникнути непорозумінь між членами команди та зацікавленими сторонами, забезпечуючи всім чітке розуміння цілей та обсягу проєкту. Це сприяє ефективній комунікації та зменшує ймовірність внесення змін на пізніх стадіях розробки, що може бути значно дорожчим та трудомістким.

Проєктування також дозволяє розробити структуру системи, визначити необхідні технології та інструменти, а також створити детальні технічні специфікації. Це забезпечує структурований підхід до розробки та полегшує управління проєктом, дозволяючи розробникам зосередитися на виконанні конкретних завдань та дотриманні термінів.

2.1 Проєктування структури соцмережі

Структура сайту – схема розташування його сторінок, категорій, підкатегорій та товарів. Це план, у якому побудовано логічний зв’язок між сторінками. Структуру сайту можна поділити на дві підкатегорії.

Зовнішня структура вебсайту – це макет сторінки із зазначенням розташування на ній блоків.

Внутрішня схема структури сайту – це відображення категорій, приналежність до них певних сторінок та матеріалів.

Типи та приклади структур сайтів.

Деревоподібна структура – нагадує кореневу систему дерева. Якщо

потрібно простіше порівняння, то генеалогічне дерево. Є головна сторінка, з неї виходять основні категорії, а з них картки товарів і т.д. Саме така логічна структура сайту є найпопулярнішою. Причина проста: вона найбільш зрозуміла та логічна.

Тегована структура – складніша, але функціональна структура. Її принцип полягає в створенні певних сторінок тегів за різними параметрами та характеристиками. Наприклад, ви перейшли до каталогу смартфонів в інтернет-магазині. У розділі «Смартфони» можуть бути додаткові теги: «Смартфони з великим екраном», «Смартфони на 2 SIM-карти» і так далі. Перевага такої схеми полягає у можливості залучення більшого обсягу трафіку, тому що розширюється список семантики. Тут можна використати низькочастотні запити.

Алфавітна структура – передбачає розміщення інформації по абетці. Використовується порівняно рідко через свою обмеженість. Основна сфера застосування алфавітної структури це структурування різних словників, енциклопедій тощо.

Хронологічна організація структури – це теж не найпоширеніший спосіб, але в деяких випадках він застосовується. Наприклад, для блогів і порталів новин, де користувачі можуть знаходити інформацію за датами.

Географічна організація структури – ця схема структури сайту використовується, якщо для доступу до інформації необхідне сортування за географічним розташуванням. Звичайно, подібна організація застосовна далеко не скрізь. Але для деяких проєктів вона є головною. Наприклад, для сайтів Booking.com, Doroga.ua та подібних.

Тематична організація структури – досить універсальна логічна структура сайту, яка має на увазі побудова вебресурсу за тематиками. Зручно і навігації, і сприйняття користувачами.

Гібридна організація структури – ця схема структури сайту передбачає використання відразу кількох методів організації інформації.

Як правило, гібридна модель застосовується на великих ресурсах:

інтернет-магазини, великі портали новин і так далі [14].

При розробці структури соціальної мережі «Voteween» було важливо спроєктувати просту та зручну структуру для користувачів, яка використовує елементи гібридного підходу (див. рис. 2.1).

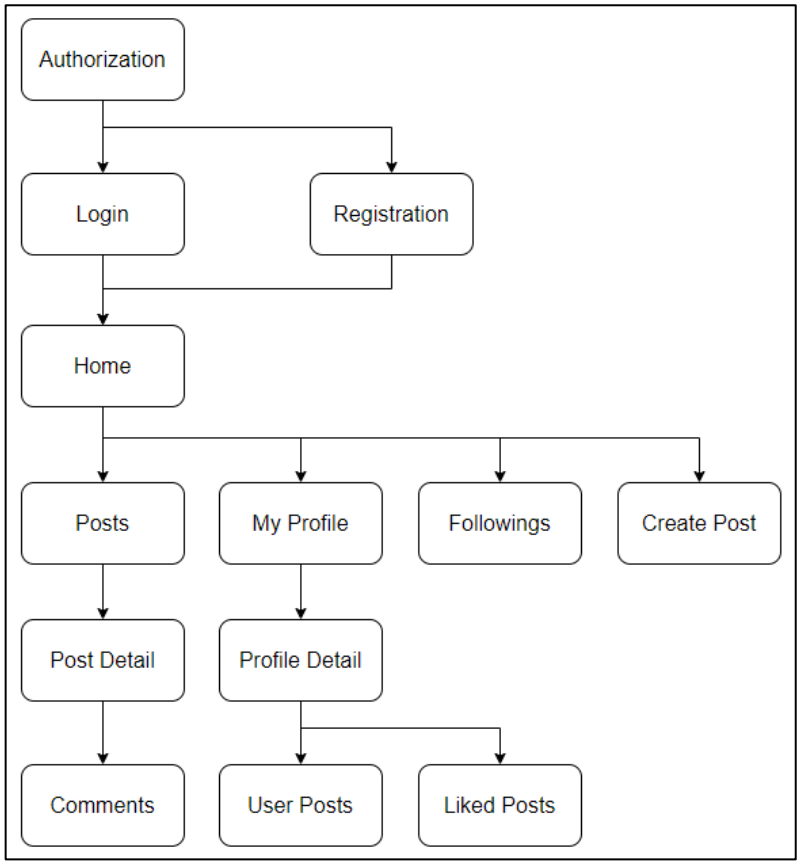

Рисунок 2.1 – Структура соцмережі «Voteween»

Спочатку користувачу необхідно авторизуватися в соцмережу. Для цього буде виділено окремо сторінку входу в соцмережу для зареєстрованих користувачів, та окремо сторінку реєстрації в соцмережі для нових користувачів.

Аналізуючи розроблену структуру можна зрозуміти що основним блоком є розділ «Home» після реєстрації чи авторизації користувача в соцмережу – має відкриватись головна сторінка соцмережі яка складається з навігаційної панелі, списку постів користувача та користувачів на яких він підписаний, список інших користувачів на яких не підписаний користувач соцмережі, список користувачів на яких підписаний даний користувач.

Важливим елементом структури є навігаційна панель. Навігаційна панель буде відображатись на будь-якому розділі соцмережі та надавати користувачеві можливість перейти на головну сторінку, можливість пошуку користувачів за їх іменем в соцмережі, можливість опублікувати пост, можливість змінити дані свого профілю, а також вихід зі свого акаунту.

При відображенні постів ми можемо натиснути на його вміст та перейти на деталі цього поста. Сам пост буде мати посилання на профіль користувача який зробив цей пост, також блок коментарів до цього посту буде мати аналогічне посилання відповідно до користувача.

Перейшовши на деталі поста буде можливість побачити коментарі які залишили користувачі під цим постом.

Якщо перейти на профіль користувача то має бути можливість підписатись або відписатись від цього користувача, переглянути список постів користувача та список постів які він вподобав.

2.2 Принципи роботи соцмережі

Unified Modeling Language (UML) – уніфікована мова моделювання. Розшифруємо: modeling передбачає створення моделі, що описує об’єкт. Unified (універсальний, єдиний) – підходить для широкого класу проєктованих програмних систем, різних областей додатків, типів організацій, рівнів компетентності, розмірів проєктів. UML описує об’єкт в єдиному заданому синтаксисі, тому де б ви не намалювали діаграму, її правила будуть зрозумілими для всіх, хто знайомий з цією графічною мовою – навіть в іншій країні.

Як будь-яка інша мова, UML має власні правила оформлення моделей і синтаксис. За допомогою графічної нотації UML можна візуалізувати систему, об’єднати всі компоненти в єдину структуру, уточнювати і покращувати модель у процесі роботи.

В мові UML є 12 типів діаграм, деякі з видів діаграм специфічні для певної системи і додатку. Найбільш доступними з них є:

– діаграма прецедентів (Use-case diagram);
– діаграма класів (Class diagram);
– діаграма активностей (Activity diagram);
– діаграма послідовності (Sequence diagram);
– діаграма розгортання (Deployment diagram);
– діаграма співробітництва (Collaboration diagram);
– діаграма об’єктів (Object diagram);
– діаграма станів (Statechart diagram).

В нашому випадку ми будемо використовувати діаграму прецедентів (Use-case diagram), діаграма класів (Class diagram), діаграма послідовності (Sequence diagram).

Діаграма прецедентів (Use-case diagram) – Діаграма прецедентів використовує 2 основних елементи:

– Actor (учасник) – множина логічно пов’язаних ролей, виконуваних при взаємодії з прецедентами або сутностями (система, підсистема або клас): учасником може бути людина, роль людини в системі чи інша система, підсистема або клас, які представляють щось поза сутністю;
– Use case (прецедент) – опис окремого аспекту поведінки системи з точки зору користувача: прецедент не показує, «як» досягається певний результат, а тільки «що» саме виконується.

Діаграма класів (Class diagram) – Класс (class) це категорія речей, що мають загальні атрибути та операції. Сама діаграма класів являє собою набір статичних, декларативних елементів моделі. Вона дає нам найбільш повне і розгорнуте уявлення про зв’язки в програмному коді, функціональність та інформацію про окремі класи. Часто додатки розробляються саме за допомогою діаграми класів.

Діаграма послідовності (Sequence Diagram) – використовується для

уточнення діаграм прецедентів – описує поведінкові аспекти системи. Діаграма послідовності відображає взаємодію об’єктів в динаміці, в часі. При цьому інформація набуває вигляду повідомлень, а взаємодія об’єктів передбачає обмін цими повідомленнями в рамках сценарію [15].

Спочатку ми створимо діаграму класів. На рисунку 2.2 показано діаграму класів соціальної мережі «Voteween». Діаграма містить такі класи: користувач, публікація, коментар.

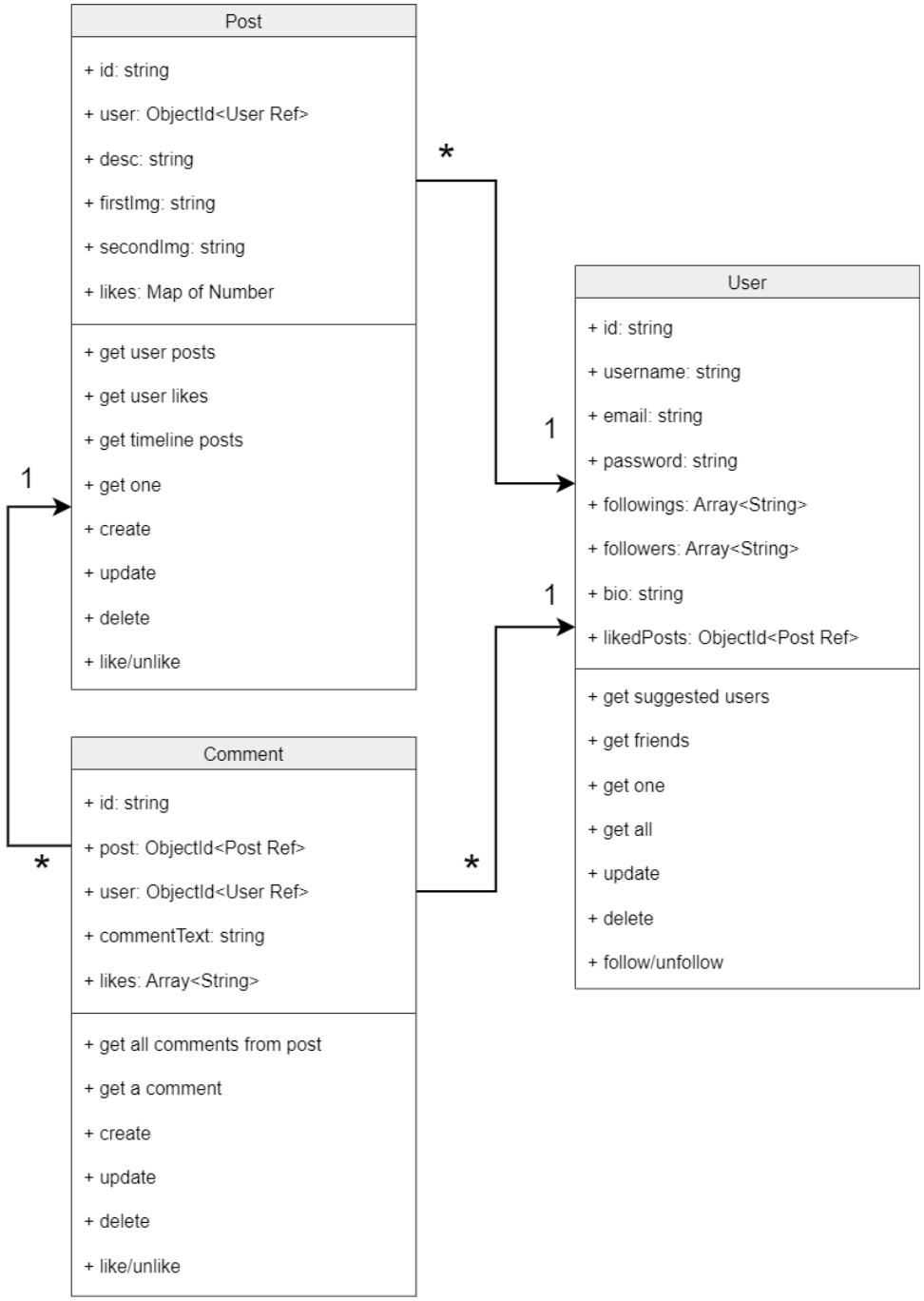

Рисунок 2.2 – Діаграма класів соціальної мережі «Voteween»

Для діаграми класів соціальної мережі «Voteween» створимо детальний опис в таблиці 2.1 з класами, атрибутами та методами, створеними для розробки соцмережі.

Таблиця 2.1 – Опис класів соціальної мережі «Voteween»

Параметр	**Значення**
User	
Атрибути	+ id – ідентифікатор користувача + username – ім’я користувача + email – поштова адреса для входу + password – пароль для входу + followings – список підписок користувача + followers – список підписників користувача + bio – біографія користувача + likedPosts – список лайкнутих постів
Методи	+ get suggested users – список користувачів на яких не підписаний користувач сесії + get friends – список друзів користувача сесії + get one – пошук користувача по id + get all – список всіх користувачів + update – оновити данні користувача + delete – видалити користувача + follow/unfollow – підписатись або відписатись на користувача
Post	
Атрибути	+ id – ідентифікатор поста + user – ім’я користувача поста + desc – опис поста + firstImg – перше фото поста + secondImg – друге фото поста + likes – список лайків

Продовження таблиці 2.1

Параметр	**Значення**
Post	
Методи	+ get user posts – список постів користувача + get user likes – список постів користувача які він вподобав + get timeline posts – список постів від друзів користувача + get one – пошук поста по id + create – створити пост + update – оновити пост + delete – видалити пост + like/unlike – лайк або дизлайк на одне з фото поста
Comment	
Атрибути	+ id – ідентифікатор коментаря + post – ідентифікатор поста + user – ідентифікатор користувача + commentText – текст коментаря + likes – лічильник лайків
Методи	+ get all comments from post – список коментарів до поста + get a comment – пошук коментаря + create – створити коментар + update – оновити коментар + delete – видалити коментар + like/unlike – лайк або дизлайк на коментар

Для зображення взаємодії об’єктів системи та їх поведінки у мові UML використовується діаграма послідовності. Основна її перевага цієї діаграми полягає у тому, що вона ілюструє зв’язки між об’єктами у послідовному порядку.

На рисунку 2.3 зображено діаграму послідовності процесу «Створення поста». На цій діаграмі розміщені об’єкти: користувач, клієнтська частина, серверна частина, база даних.

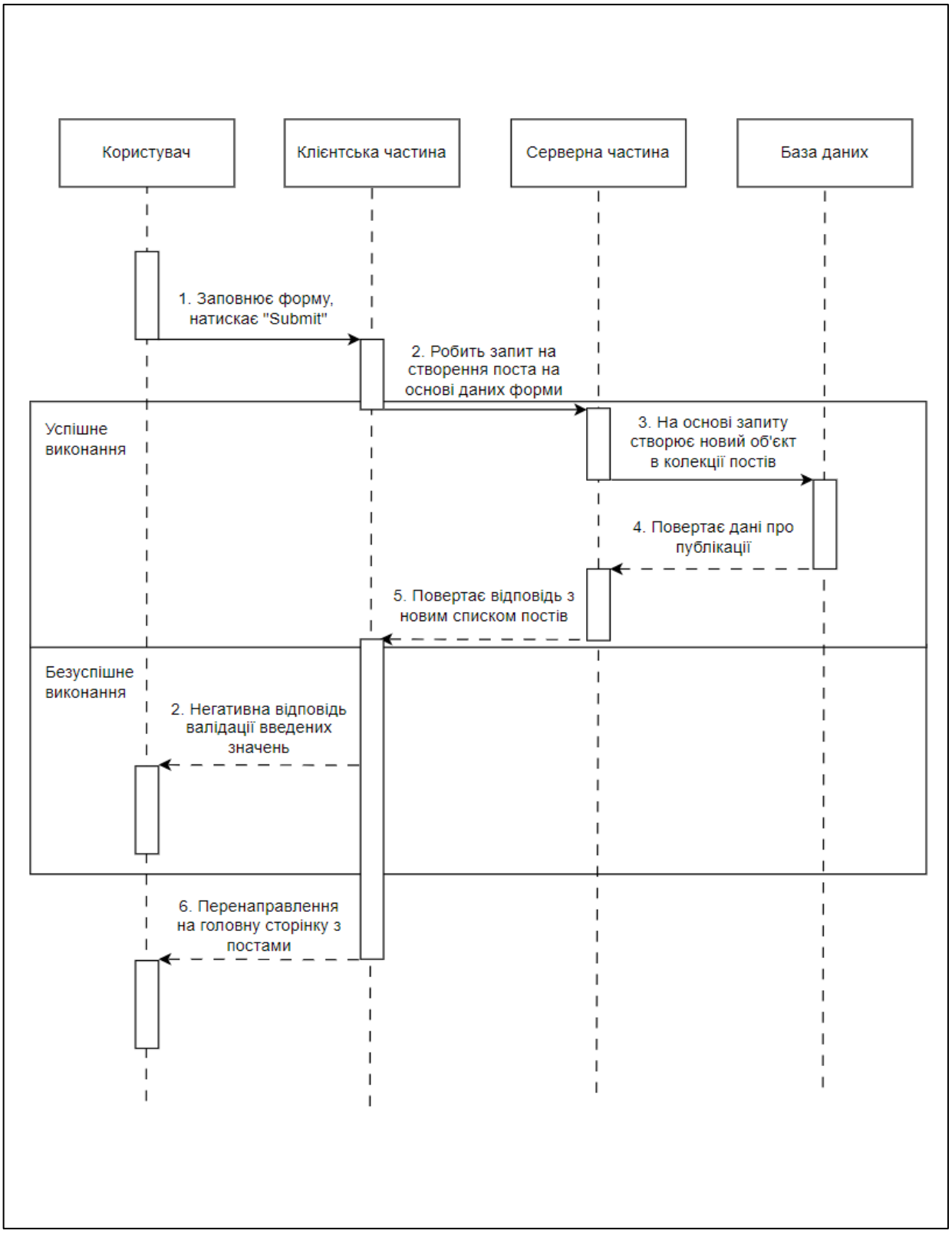

Рисунок 2.3 – Діаграма послідовності для процесу «Створення поста»

В побудованій діаграмі послідовності для процесу «Створення поста» на рисунку 2.3 відбуваються наступні дії.

Користувач натискає на кнопку «Upload post», відкривається вікно створення посту.

Після цього користувач заповнює всі необхідні поля та натискає кнопку «Submit».

Після цього наш запит йде на сервер, а вже сервер надсилає запит до бази даних з де на основі цього запиту в базі даних створюється новий об’єкт в колекції постів.

Успішним виконанням буде вважатись перенаправлення на сторінку з постами де буде відображатись наш новий пост.

Безуспішним виконанням буде вважатись негативна відповідь валідації для введених користувачем даних.

Діаграма прецедентів, або (Use-case diagram), забезпечує візуалізацію ролей у системі та їх взаємодії. Вона акцентує увагу на типах ролей і їхніх зв’язках із системою, не демонструючи послідовний порядок дій.

Ця діаграма відображає функціональні вимоги, показуючи, які дії система може виконувати з точки зору користувача. Інформацію можна передати як у текстовому вигляді, так і за допомогою графічного зображення.

На рисунку 2.4 представлена діаграма варіантів використання соціальної мережі «Voteween», що включає двох акторів: авторизованого і неавторизованого користувача.

На кожного актора розповсюджується різний рівень доступу до функціоналу. Найбільш обмежений функціонал у неавторизованого користувача, тому що він має доступ тільки до реєстрації або авторизаціїї.

Щодо авторизованого користувача, то він має додаткові функції, такі як: створення посту, перегляд постів користувачів, перегляд постів користувачів на яких він підписаний, видалити свій пост, вподобати пост, підписатись на користувача, відписатись від користувача, перегляд профілю користувача, оновити дані свого профілю, прокоментувати пост, вподобати коментар, вийти з соцмережі, реєстрація для авторизованого користувача стає недоступною.

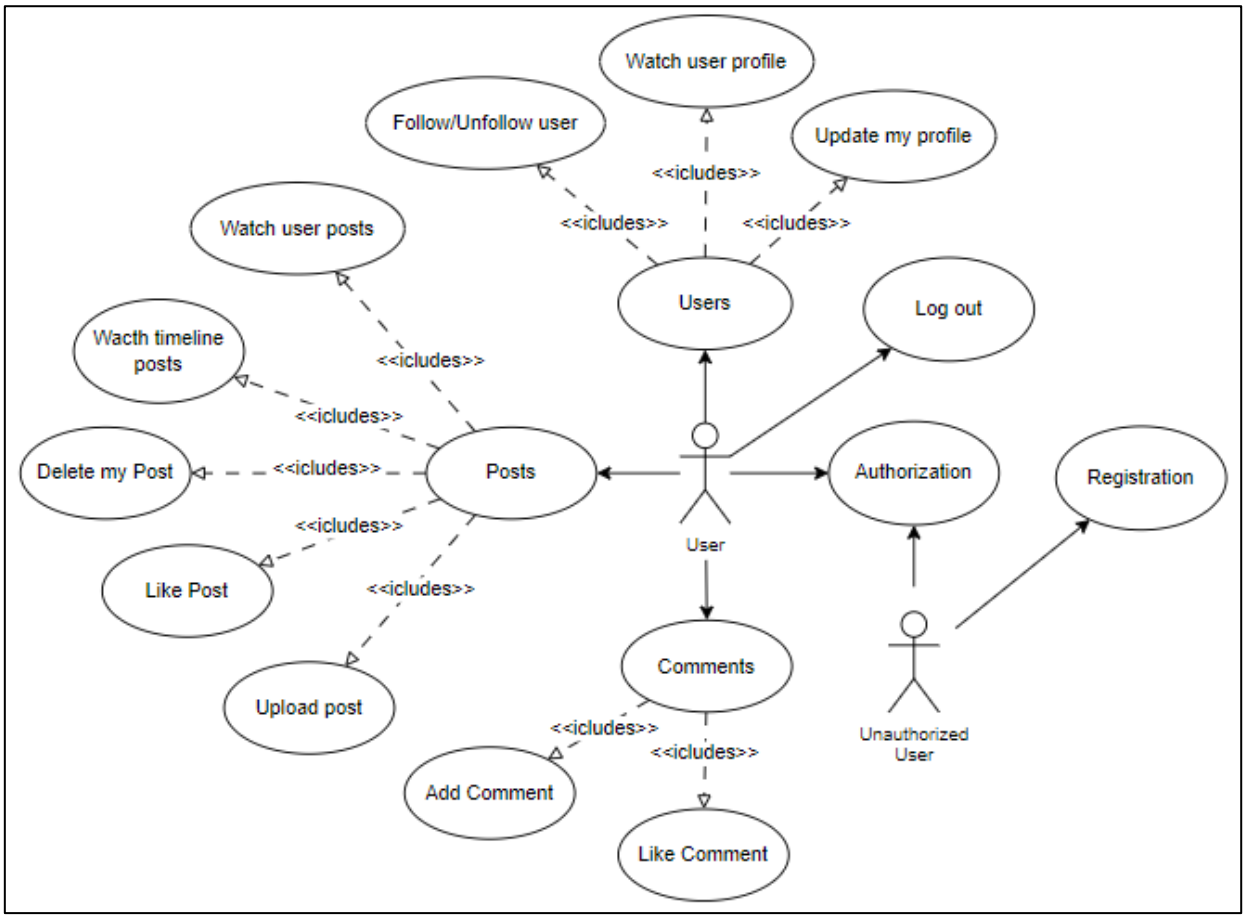

Рисунок 2.4 – Діаграма прецедентів соціальної мережі «Voteween»

2.3 Проєктування структури бази даних соцмережі

ER Diagram розшифровується як Entity Relationship Diagram (діаграма зв’язків сутностей), також відома як ERD – це діаграма, яка відображає взаємозв’язок наборів сутностей, що зберігаються в базі даних. Іншими словами, ER-діаграми допомагають пояснити логічну структуру баз даних.

Діаграми ER створюються на основі трьох основних понять: сутностей (Entities), атрибутів(Attributes) і відношеннь (Relationships).

Сутність – це реальна річ, жива чи нежива, яку легко впізнати і не впізнати. Це все, що є на підприємстві, яке має бути представлено в нашій базі даних. Це може бути фізична річ або просто факт про підприємство або подія, яка відбувається в реальному світі.

Сутністю може бути місце, особа, об’єкт, подія або концепція, яка

зберігає дані в базі даних. Характеристики сутностей повинні мати атрибут і унікальний ключ. Кожна сутність складається з деяких «атрибутів», які представляють цю сутність.

Атрибути – це однозначна властивість типу сутності або типу зв'язку. Наприклад, лекція може мати атрибути: час, дата, тривалість, місце тощо.

Відносини – це не що інше, як зв'язок між двома або більше сутностями. Наприклад, Том працює на кафедрі хімії.

Сутності беруть участь у відносинах. Ми часто можемо ідентифікувати зв'язки за допомогою дієслів або дієслівних словосполучень [16].

В діаграмах відношень сутностей (ER) існує кілька типів відношень, які допомагають відобразити зв'язки між сутностями. Декілька основних типів відношень наведемо нижче.

Один до одного (One-to-One): у цьому відношенні одна сутність пов'язана з однією і тією ж самою сутністю. Наприклад, "Людина" може мати один "Паспорт", а "Паспорт" також належить лише одній "Людині".

Один до багатьох (One-to-Many): у цьому відношенні одна сутність пов'язана з декількома сутностями іншого типу. Наприклад, "Категорія товарів" може мати багато "Товарів", але кожен "Товар" належить лише одній "Категорії товарів".

Багато до одного (Many-to-One): це протилежність відношення "Один до багатьох". Багато сутностей одного типу пов'язані з однією сутністю іншого типу. Наприклад, багато "Замовлень" можуть належати одному "Клієнту".

Багато до багатьох (Many-to-Many): у цьому відношенні багато сутностей одного типу пов'язані з багатьма сутностями іншого типу. Це відношення вимагає введення додаткової таблиці-посередника. Наприклад, "Студенти" можуть бути пов'язані з "Курсами", і кожен "Студент" може відвідувати багато "Курсів", а "Курс" може мати багато "Студентів".

ER-діаграма досить зручно ілюструє структуру бази даних. Тому було побудовано діаграму «сутність-зв'язок» для соцмережі «Voteween».

На рисунку 2.5 зображено ER-діаграму соціальної мережі, яка містить 3 сутності.

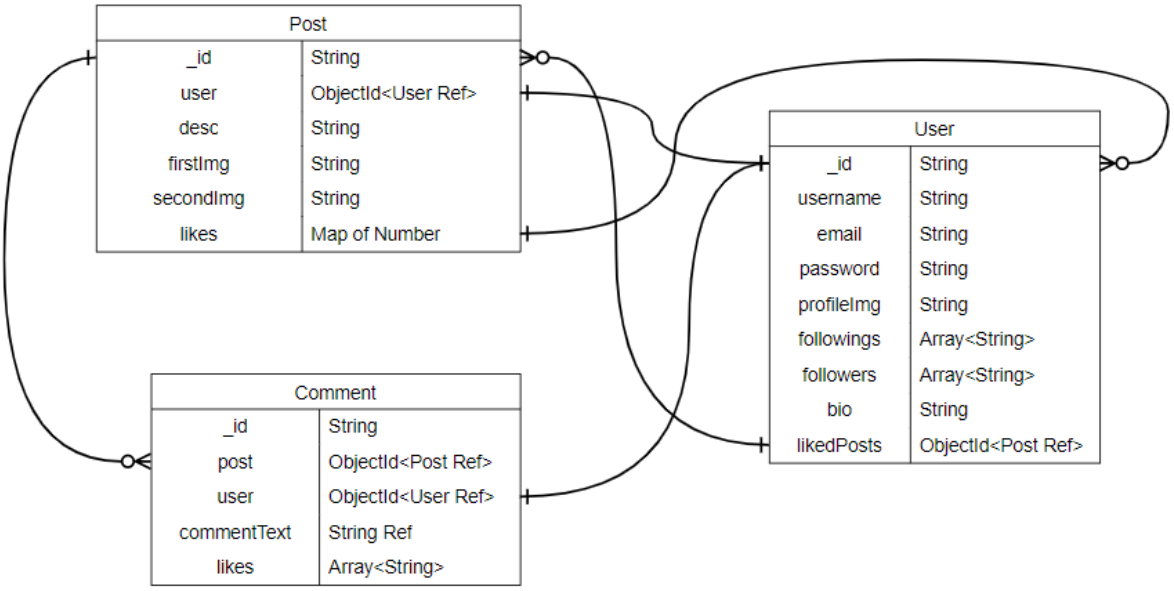

Рисунок 2.5 – ER-діаграма соціальної мережі «Voteween»

У даній ER-моделі зображено такі відношення та сутності:

– «Post» та «User» мають відношення «один-до-одного», тому що пост має тільки одного користувача як автора поста;
– «Post» та «Comment» мають відношення «один-до-багатьох», тому що пост може мати багато коментарів;
– «Post» та «User» також має відношення «один-до-багатьох», тому що один пост може мати багато користувачів в лічильнику вподобань;
– «User» та «Post» має відношення «один-до-багатьох» тому що один користувач може мати багато постів які він вподобав;
– «Comment» та «User» має відношення «один-до- одного» тому що один коментар може мати тільки одного користувача як автора коментаря.

3 РЕАЛІЗАЦІЯ СОЦМЕРЕЖІ

3.1 Реалізація серверної частини соцмережі

Для початку потрібно створити середовище розробки серверної частини. Для цього створюємо в папці проєкту нову папку «server».

Після цього з використанням програми «Visual Studio Code» та попередньо встановленим Node.js на робочій машині ми ініціалізуємо проєкт за допомогою команди «npm -init -y».

Команда npm init -y використовується для створення нового файлу package.json в поточному каталозі. package.json – це важливий файл для будь-якого проєкту на Node.js, оскільки він містить різну важливу інформацію про проєкт. Опція -y використовується для автоматичного заповнення значень за замовчуванням. Без цієї опції npm init запитає вас ввести ці значення вручну.

Тепер нам потрібно зробити інсталяцію всіх необхідних бібліотек та інструментів для розробки серверної чатини за допомогою одної команди: «npm i bcrypt cors dotenv express jsonwebtoken mongoose multer nodemon».

Дана команда встановлює наступні пакети в наш проєкт Node.js:

- bcrypt: це бібліотека, яка допомагає нам хешувати паролі;
- cors: це middleware, який дозволяє або забороняє cross-origin запити до нашого сервера;
- dotenv: цей модуль завантажує змінні середовища з файлу;
- express: це фреймворк для створення вебдодатків на Node.js;
- jsonwebtoken: це бібліотека для роботи з JSON Web Tokens;
- mongoose: це бібліотека, яка допомагає нам працювати з MongoDB;
- multer: це middleware для обробки multipart/form-data, який в основному використовується для завантаження файлів;
- nodemon: це інструмент, який допомагає автоматично перезапускати наш Node.js додаток, коли відбуваються зміни в коді.

Файл index.js є основним файлом серверного додатку. Він використовується для налаштування та запуску сервера. Саме через нього ми підключаємось до бази даних, підключаємо контролери, налаштовуємо middleware та маршрути.

На рисунку 3.1 зображено вміст нашого файлу index.js.

index.js M

server > index.js > ...

```
1   const commentController = require('./controllers/commentController')
2   const uploadController = require('./controllers/uploadController')
3   const authController = require('./controllers/authController')
4   const userController = require('./controllers/userController')
5   const postController = require('./controllers/postController')
6   const mongoose = require('mongoose')
7   const express = require('express')
8   const cors = require('cors')
9   const app = express()
10
11  // connect DB
12  mongoose.set('strictQuery', false);
13  mongoose
14      .connect(process.env.MONGO_URL)
15      .then(() => console.log('DB is connected'))
16      .catch((err) => console.log('DB error', err))
17
18  app.use('/images', express.static('public/images'))
19  app.use(cors())
20  app.use(express.json())
21  app.use(express.urlencoded({extended: true}))
22  app.use('/auth', authController)
23  app.use('/user', userController)
24  app.use('/post', postController)
25  app.use('/comment', commentController)
26  app.use('/upload', uploadController)
27
28  // connect app
29  app.listen(process.env.PORT, () => console.log('Server is connected'))
```

Рисунок 3.1 – Вміст файлу index.js серверної частини

Спочатку ми імпортуємо необхідні нам інструменти.

Далі ми підключаємось до бази даних MongoDB за допомогою бібліотеки mongoose. Детальний опис підключення до бази даних та налаштування серверу є в наступних розділах.

3.1.1 Налаштування бази даних соцмережі

Попередньо нам необхідно створити та налаштувати нашу бази даних в MongoDB, для цього необхідно виконали наступні кроки:

– авторизувались на офіційному сайті MongoDB;
– натиснули кнопку «Create Database»;
– обрати безкоштовний тариф «Shared Cluster» (В нашому випадку);
– в розділі баз даних обрали наш кластер та натиснули «Connect»;
– обираємо метод «Connect to your application»;
– копіюємо посилання через яке ми будемо підключатись до нашої бази даних (рис. 3.2);
– вставляємо це посилання для подальшого використання в наш файл «.env» (рис. 3.3).

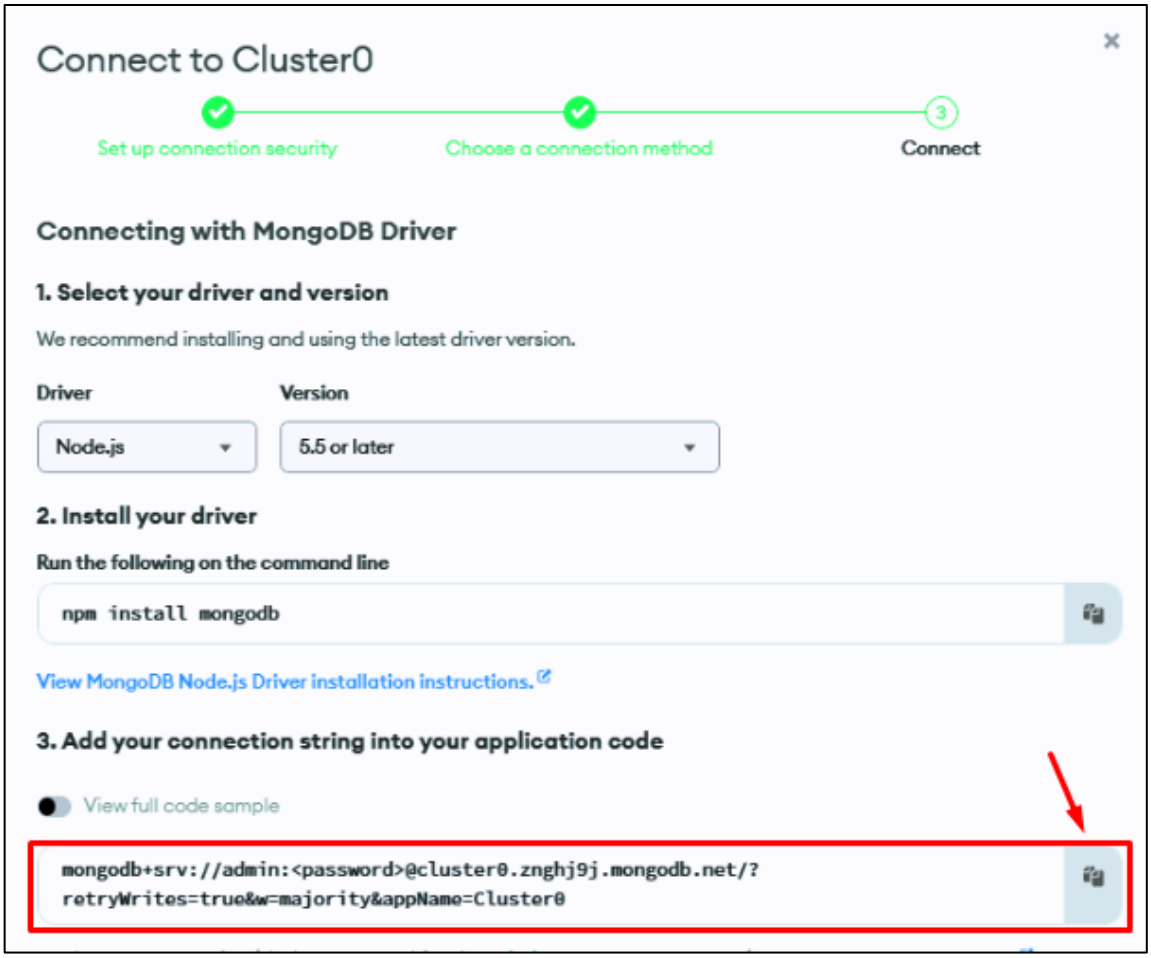

Рисунок 3.2 – Посилання для підключення до бази даних

Кроки використання бібліотеки mongoose:

– «mongoose.set('strictQuery', false)» – цей рядок встановлює параметр «strictQuery» в значення «false» (це означає, що Mongoose не буде викидати помилку, якщо в запиті до бази даних використовуються поля, які не визначені в схемі);
– «mongoose.connect(process.env.MONGO_URL)» – цей рядок використовує метод connect в mongoose для підключення до MongoDB, URL бази даних береться з змінних файлу «.env» (рис. 3.3);
– .then(() => console.log('DB is connected')) – цей рядок використовує проміси для обробки успішного підключення (якщо підключення до бази даних успішне, в консоль виводиться повідомлення “DB is connected”);
– .catch((err) => console.log('DB error', err)) – цей рядок використовує проміси для обробки помилок підключення (якщо виникає помилка підключення, вона виводиться в консоль з повідомленням “DB error”).

Файл .env (рис. 3.3) використовується для зберігання змінних середовища, які використовуються в нашому додатку. Це можуть бути конфіденційні дані, такі як ключі API, паролі, секрети JWT, URL-адреси баз даних тощо. В нашому випадку ми використовуємо такі змінні:

– MONGO_URL – це URL-адреса для підключення до нашої бази даних MongoDB;
– PORT – це порт, на якому прослуховується наш сервер Express;
– JWT_SECRET – це секретний ключ, який використовується для підпису та верифікації JWT (JSON Web Tokens) в нашому додатку.

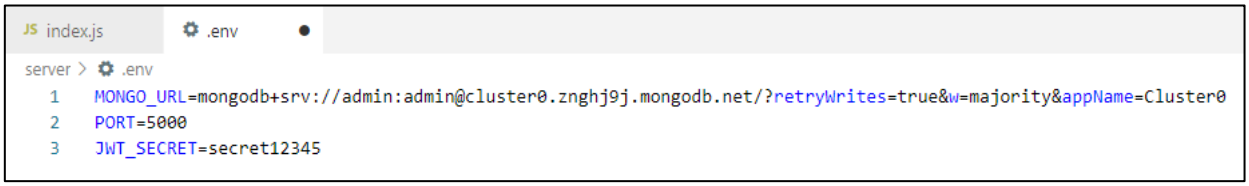

Рисунок 3.3 – Вміст файлу .env

3.1.2 Налаштування серверу соцмережі

Далі в файлі index.js відбувається налаштування вебсервера за допомогою Express.

Створення сервера: const app = express() створює новий екземпляр сервера Express.

Статичні файли: app.use('/images', express.static('public/images')) вказує на те, що файли в директорії public/images (рис. 3.4) будуть доступні для використання як статичні файли.

Налаштування middleware: app.use() використовується для налаштування middleware для сервера. cors() дозволяє cross-origin запити, express.json() та express.urlencoded({extended: true}) дозволяють серверу обробляти JSON та URL-encoded запити відповідно.

Настроювання маршрутів: app.use() також використовується для налаштування маршрутів для сервера. Наприклад, app.use('/auth', authController) означає, що всі запити, які починаються з /auth, будуть оброблятися authController.

Запуск сервера: app.listen(process.env.PORT, () => console.log('Server is connected')) використовується для запуску сервера на порту, який вказаний в файлі «.env» (рис. 3.3).

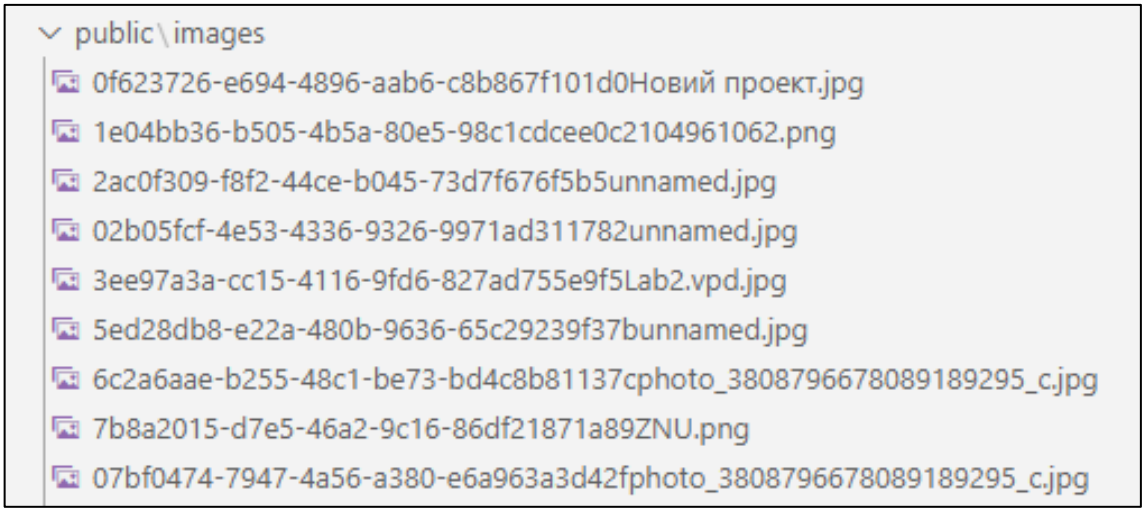

Рисунок 3.4 – Локальне сховище для фото «public\images»

3.1.3 Розробка схем для наших моделей

Схеми в MongoDB використовуються для визначення структури документів у колекції. Вони допомагають забезпечити узгодженість даних, встановлюючи визначені поля та типи даних для документів у колекції.

Наприклад в нашому випадку PostSchema визначає структуру документів у колекції “Post” (рис. 3.5). Вона вказує, що кожен пост повинен мати поля user, desc, firstImg, secondImg, likes, а також автоматично генеровані поля createdAt та updatedAt.

Post.js

server > models > Post.js > <unknown>

```
1   const mongoose = require('mongoose')
2
3   const PostSchema = new mongoose.Schema({
4       user: {
5           type: mongoose.Schema.Types.ObjectId,
6           ref: "User",
7           required: true
8       },
9       desc: {
10          type: String,
11          required: true,
12          min: 8
13      },
14      firstImg: {
15          type: String,
16          required: true
17      },
18      secondImg: {
19          type: String,
20          required: true
21      },
22      likes: {
23          type: Map,
24          of: Number,
25          default: {}
26      }
27  }, {timestamps: true})
28
29  module.exports = mongoose.model("Post", PostSchema)
```

Рисунок 3.5 – Вигляд схеми поста в файлі «Post.js»

Файл «Post.js» (рис. 3.5) визначає схему поста (PostSchema) за допомогою бібліотеки Mongoose для MongoDB. Схема визначає структуру документів у колекції MongoDB (рис. 3.6). Опис частин коду в файлі «Post.js».

Const mongoose = require('mongoose') – імпортує бібліотеку Mongoose, яка використовується для роботи з MongoDB.

Const PostSchema = new mongoose.Schema({ ... }, {timestamps: true}) – створює нову схему Mongoose з визначеними полями. Параметр timestamps: true автоматично додає поля createdAt та updatedAt до схеми.

User, desc, firstImg, secondImg, likes – це поля, які визначені в схемі. Для кожного поля вказані типи даних, а також додаткові параметри, такі як required, ref, min, default.

User – обов'язкове поле, яке використовує ObjectId для посилання на модель "User".

Desc – обов'язкове поле типу String з мінімум 8 символів.

FirstImg і secondImg – обов'язкові поля типу String.

Likes – поле, яке використовує тип Map для зберігання пар ключ-значення, де ключ – це ID користувача, а значення це кількість індекс фото яку лайкнув користувач. Значення за замовчуванням – порожній об'єкт.

Module.exports = mongoose.model("Post", PostSchema) – експортує модель Post, яка використовує схему PostSchema.

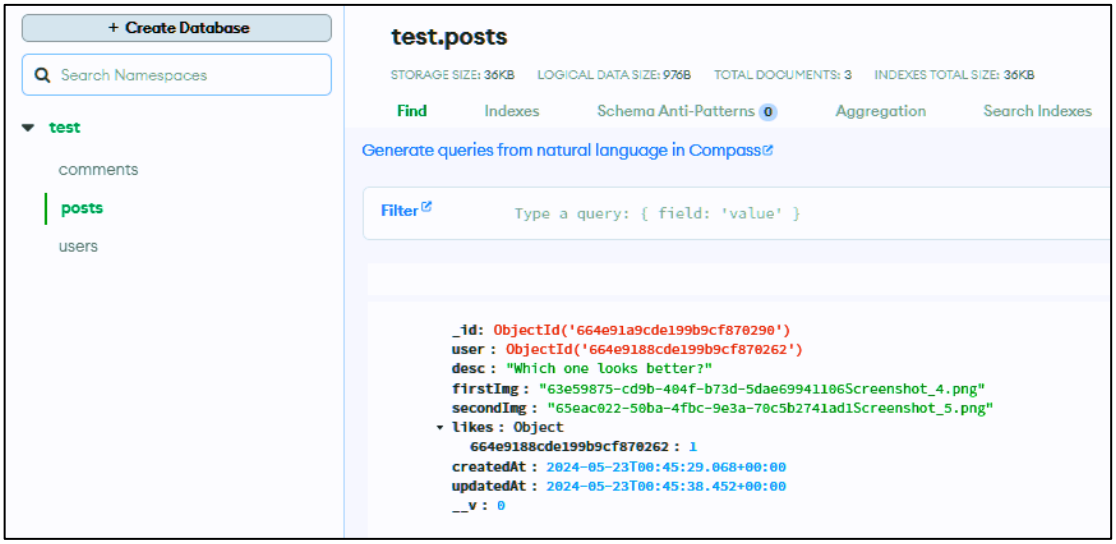

Рисунок 3.6 – Вигляд поста в колекції постів бази даних

3.1.4 Розробка контролерів для обробки запитів

Контролери в серверній частині додатку використовуються для обробки вхідних HTTP-запитів, виконання бізнес-логіки та надавання відповідей клієнту. Вони є частиною архітектурного шаблону MVC (Model-View-Controller).

Візьмемо для прикладу контролер користувача «userController.js» та розглянемо деякі його окремі методи (рис. 3.7).

Рисунок 3.7 – Бібліотеки та перший метод в контролері «userController.js»

UserController використовує модель User для взаємодії з базою даних MongoDB, а також middleware verifyToken для перевірки автентифікації користувача.

Спочатку ми робимо необхідні імпорти:

- const User = require('../models/User') – імпортує модель User, яка використовується для взаємодії з колекцією користувачів в базі даних MongoDB;
- const bcrypt = require('bcrypt') – імпортує модуль bcrypt, який використовується для хешування паролів;
- const userController = require("express").Router() – створює новий маршрутизатор Express, який буде використовуватися для обробки запитів, пов’язаних з користувачами;
- const Post = require('../models/Post') – імпортує модель Post, яка використовується для взаємодії з колекцією постів в базі даних MongoDB;
- const verifyToken = require('../middlewares/verifyToken') – імпортує middleware verifyToken, який робить перевірку токенів авторизації.

Опис обробки GET-запиту на прикладі методу get suggested users.

Метод get suggested users (рис. 3.7) визначає обробку маршруту /find/suggestedUsers. Цей контролер використовує middleware verifyToken для перевірки токену авторизації. Після цього він знаходить поточного користувача в базі даних за ID, отриманим з запиту.

Потім він знаходить всіх користувачів в базі даних, але не включає їхні паролі в результат. Це робиться для забезпечення безпеки, оскільки паролі не повинні передаватися або відображатися.

Потім він фільтрує список користувачів, щоб знайти тих, яких поточний користувач ще не слідує, і які не є поточним користувачем. Це робиться для того, щоб запропонувати користувачам нових людей для слідкування.

Якщо кількість рекомендованих користувачів більше 5, то список обмежується до перших 5 користувачів. Це робиться для того, щоб не перевантажувати користувача великою кількістю рекомендацій.

Нарешті, він відправляє список рекомендованих користувачів як відповідь на запит.

Якщо сталася помилка, відправляється повідомлення про помилку. Опис обробки PUT-запиту на прикладі методу update user (рис 3.8).

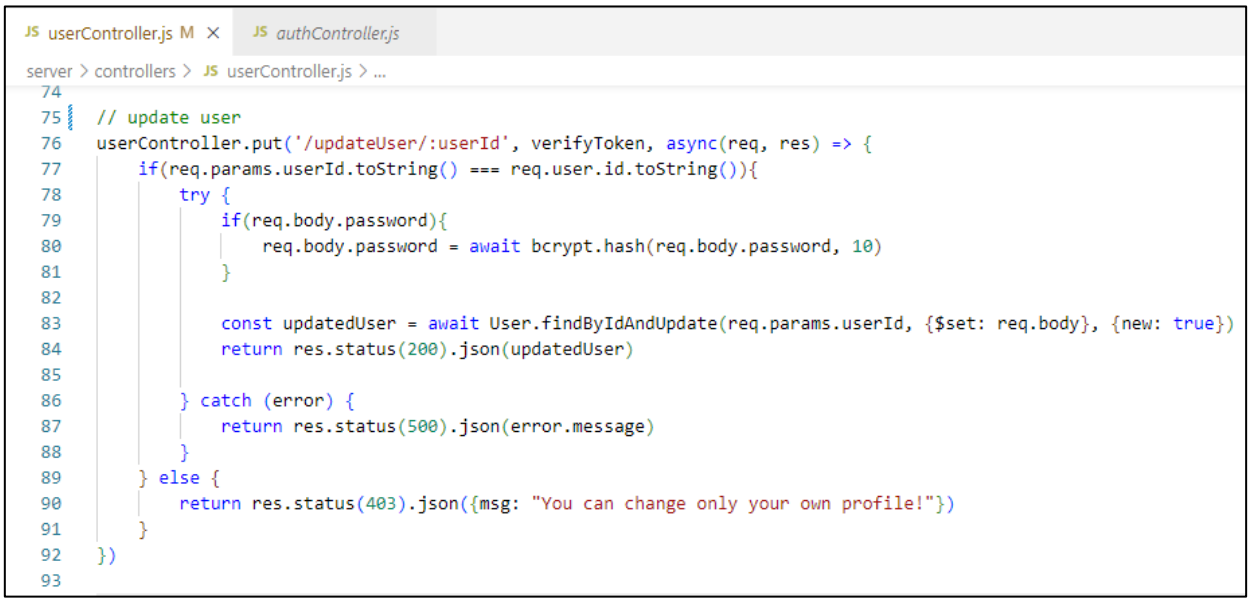

Рисунок 3.8 – Метод Update User в контролері «userController.js»

Метод update user визначає обробку для PUT-запитів на маршрут /updateUser/:userId. Цей обробник використовує middleware verifyToken для перевірки токену авторизації.

Він перевіряє, чи userId користувача в параметрах запиту співпадає з ID користувача, отриманим з токену авторизації. Якщо це не так, він відправляє повідомлення про помилку з кодом статусу 403, що означає “Forbidden” (заборонено).

Якщо ID співпадають, він перевіряє, чи було надано новий пароль в тілі запиту. Якщо так, він хешує новий пароль за допомогою bcrypt з сіллю 10.

Потім він знаходить користувача в базі даних за ID, отриманим з параметра запиту, і оновлює поля користувача даними з тіла запиту. Він використовує опцію {new: true}, щоб повернути оновленого користувача з бази даних.

Якщо оновлення пройшло успішно, він відправляє оновленого користувача як відповідь на запит з кодом статусу 200, що означає “OK”.

Якщо сталася помилка, відправляється повідомлення про помилку з кодом статусу 500, що означає “Internal Server Error”.

Опис обробки DELETE-запиту на прикладі методу delete user (рис 3.9).

```
94   // delete
95   userController.delete('/deleteUser/:userId', verifyToken, async(req, res) => {
96       if(req.params.userId === req.user.id){
97           try {
98               await User.findByIdAndDelete(req.user.id)
99               return res.status(200).json({msg: "Successfully deleted user"})
100          } catch (error) {
101              return res.status(500).json(error.message)
102          }
103      } else {
104          return res.status(403).json({msg: "You can delete only your own profile!"})
105      }
106  })
```

Рисунок 3.9 – Метод Delete User в контролері «userController.js»

Метод delete user визначає обробку для DELETE-запитів на маршрут /deleteUser/:userId. Цей контролер використовує middleware verifyToken для перевірки токену авторизації.

Він перевіряє, чи ID користувача в параметрах запиту співпадає з ID користувача, отриманим з токену авторизації. Якщо це не так, він відправляє повідомлення про помилку з кодом статусу 403, що означає “Forbidden” (заборонено).

Якщо ID співпадають, він намагається видалити користувача з бази даних за ID, отриманим з параметра запиту. Якщо видалення пройшло успішно, він відправляє повідомлення про успішне видалення з кодом статусу 200, що означає “OK”.

Якщо сталася помилка, відправляється повідомлення про помилку з кодом статусу 500, що означає “Internal Server Error”.

В файлі «userController» немає обробки POST-запиту для створення нового користувача, натомість за створення нових користувачів відповідає контролер «authController». Детальніше про обробку POST-запиту, авторизацію та реєстрацію описано в наступному розділі.

3.1.5 Розробка авторизації з використанням JWT та bcrypt

Авторизація потрібна для того, щоб визначити, які ресурси або дії доступні конкретному користувачу або програмі. Це важливий аспект безпеки, який допомагає запобігти несанкціонованому доступу до конфіденційної інформації або критичних функцій системи.

У контексті вебдодатків, авторизація зазвичай включає перевірку того, чи має користувач право на виконання певних дій, таких як перегляд, створення, редагування або видалення даних. Наприклад, користувач може мати право переглядати інформацію про свій обліковий запис, але не мати права переглядати інформацію про інші облікові записи.

В деяких випадках необхідно бути авторизованим щоб мати доступ до деяких даних або функцій, наприклад щоб оновити пост (рис. 3.8) потрібно пройти перевірку на токен авторизації, це відбувається за допомогою функції «verifyToken».

Розглянемо детальніше функцію перевірки токену «verifyToken.js» (рис. 3.10).

verifyToken.js

server > middlewares > verifyToken.js > verifyToken

```
1  const jwt = require("jsonwebtoken")
2
3  const verifyToken = (req, res, next) => {
4      if(!req.headers.authorization) return res.status(403).json({msg: "Not authorized. Token not found"})
5
6      if(req.headers.authorization && req.headers.authorization.startsWith("Bearer ")){
7          const token = req.headers.authorization.split(" ")[1]
8          jwt.verify(token, process.env.JWT_SECRET, (err, data) => {
9              if(err) return res.status(403).json({msg: "Wrong or expired token"})
10             else {
11                 req.user = data
12                 next()
13             }
14         })
15     }
16 }
17
18 module.exports = verifyToken
```

Рисунок 3.10 – Функція перевірки токену «verifyToken.js»

Файл verifyToken.js має функцію що використовується для перевірки токенів JWT, що надходять з клієнтських запитів. Ця функція є проміжним

обробником (middleware) в Express.js, який використовується для перевірки автентифікації перед обробкою запиту.

Функція спочатку перевіряє, чи є в заголовку запиту поле authorization. Якщо його немає, вона відправляє відповідь зі статусом 403 та повідомленням “Not authorized. Token not found”.

Якщо поле authorization існує і починається з “Bearer”, функція витягує токен, розділивши рядок по пробілу і взявши другу частину.

Потім використовується jwt.verify для перевірки токену. Ця функція приймає токен, секретний ключ (змінна середовища JWT_SECRET) та функцію зворотного виклику.

У функції зворотного виклику перевіряється, чи є помилка. Якщо є, відправляється відповідь та статус 403 та повідомленням “Wrong token”.

Якщо помилки немає, дані, що декодуються з токену, додаються до об’єкта запиту як req.user, і викликається next(), щоб продовжити до наступного обробника запиту.

Тепер розглянемо наш контролер авторизації «authController.js» та його окремі методи (рис. 3.11).

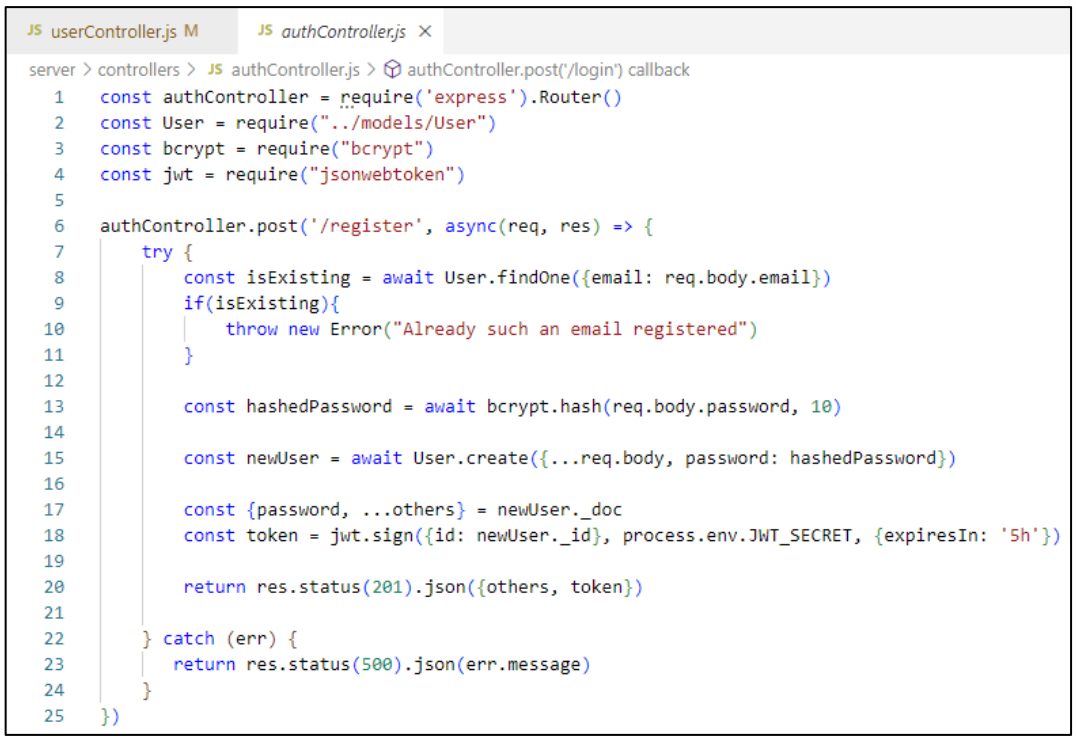

Рисунок 3.11 – Бібліотеки та метод реєстрації в контролері «authController.js»

Контролер авторизації визначає два маршрути для обробки запитів реєстрації та входу користувачів. Він використовує модуль User для взаємодії з базою даних користувачів, bcrypt для хешування паролів та jsonwebtoken для створення токенів JWT.

Маршрут /register обробляє POST-запити для реєстрації нових користувачів (рис. 3.11). Він перевіряє, чи вже існує користувач з такою ж електронною поштою. Якщо так, він відправляє повідомлення про помилку. Якщо ні, він хешує пароль за допомогою bcrypt, створює нового користувача з даними запиту та хешованим паролем, генерує токен JWT та відправляє відповідь з новим користувачем та токеном.

Маршрут /login обробляє POST-запити для входу користувачів (рис. 3.12). Він знаходить користувача за електронною поштою, перевіряє, чи відповідає наданий пароль збереженому хешу пароля. Якщо користувача не знайдено або паролі не співпадають, він відправляє повідомлення про помилку. Якщо все вірно, він генерує токен JWT та відправляє відповідь з даними користувача та токеном.

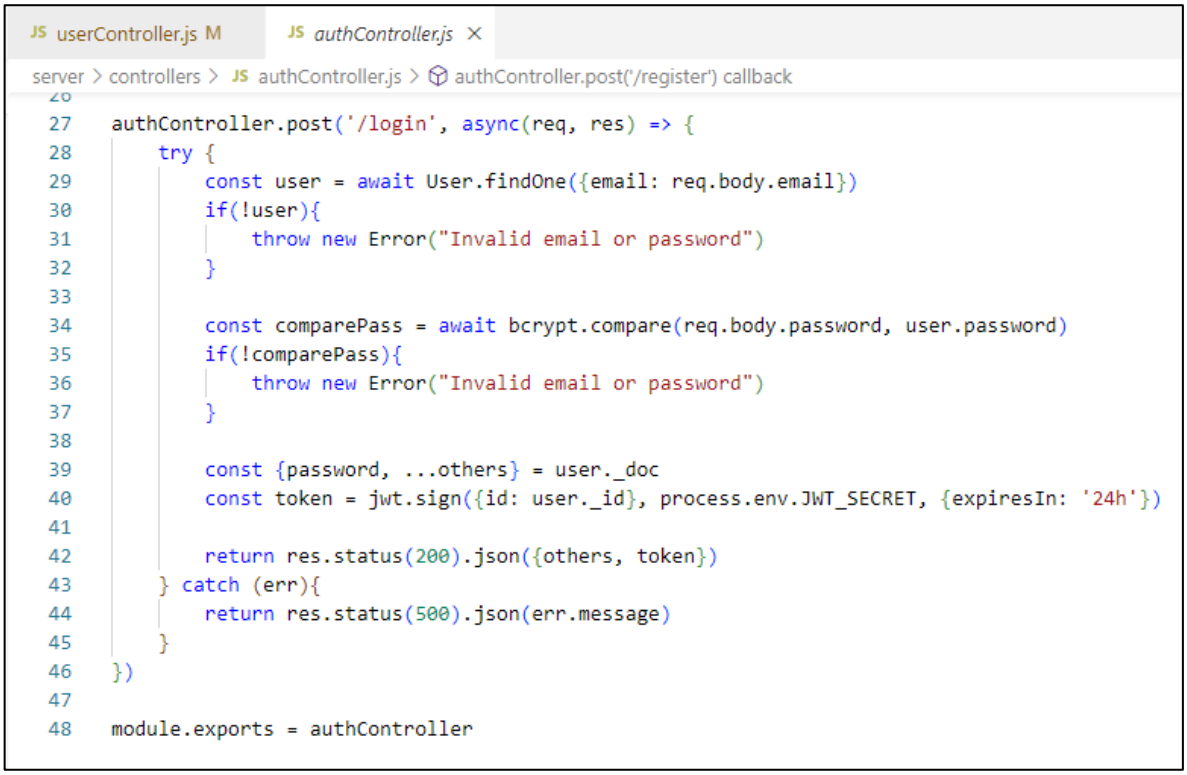

Рисунок 3.12 – Метод авторизації в контролері «authController.js»

Цей процес авторизації використовує токени JWT для автентифікації користувачів після реєстрації/входу. Коли користувач виконує запит до захищеного маршруту, він повинен надіслати цей токен в заголовку Authorization свого запиту. Сервер потім перевіряє цей токен, щоб переконатися, що користувач є дійсним.

3.2 Реалізація клієнтської частини соцмережі

Для початку потрібно створити середовище розробки клієнтської частини. Для цього створюємо в папці проєкту нову папку «client».

Після цього ми створюємо проєкт клієнтської частини за допомогою команди «npx create-react-app .».

Команда npx create-react-app . використовується для створення нового проєкту React в поточній директорії.

Create-react-app – це інструмент, що створює новий проєкт React з попередньо налаштованим середовищем розробки.

Npx – це інструмент, що входить до складу npm (Node Package Manager), який дозволяє виконувати пакети Node.js без їх попереднього встановлення.

Точка (.) в кінці команди вказує, що новий проєкт React має бути створений в поточній директорії. Якщо ви замініте точку іменем директорії, create-react-app створить нову директорію з цим іменем і встановить туди проєкт React.

Тепер нам потрібно зробити інсталяцію всіх необхідних бібліотек та інструментів для розробки клієнтської частини за допомогою одної команди: «npm i react-icons @redux/toolkit react-redux redux-persist timeago.js».

Дана команда встановлює п’ять пакетів npm в наш проєкт:

а) react-icons: цей пакет містить іконки, які можна використовувати в React-додатках – він включає іконки з різних бібліотек, таких як Font Awesome, Ionicons, Material Design та інші;

б) @redux/toolkit: це офіційний, оптимізований набір інструментів для Redux, який спрощує написання коду Redux – він включає утиліти для автоматичного створення дій та редукторів, роботи з асинхронним кодом та іншого;

в) react-redux: це офіційна бібліотека зв'язку для використання Redux разом з React – вона дозволяє вашим компонентам React взаємодіяти зі стором Redux;

г) redux-persist: цей пакет дозволяє зберігати стан вашого додатку Redux в локальному сховищі браузера або іншому місці зберігання, щоб він міг вижити при перезавантаженні сторінки;

д) timeago.js: ця бібліотека дозволяє легко форматувати часові мітки як відносний час (наприклад, «5 хвилин тому»).

Файл index.js є вхідною точкою для нашого React-додатку (рис. 3.13). Він використовує ReactDOM.createRoot для створення кореня React-додатку в елементі з ідентифікатором 'root'.

index.js

client > src > index.js > ...

```
1  import React from 'react';
2  import ReactDOM from 'react-dom/client';
3  import App from './App';
4  import { BrowserRouter } from 'react-router-dom';
5  import { Provider } from 'react-redux';
6  import { persistor, store } from './redux/store';
7  import { PersistGate } from 'redux-persist/integration/react';
8
9  const root = ReactDOM.createRoot(document.getElementById('root'));
10 root.render(
11   <Provider store={store}>
12     <PersistGate persistor={persistor}>
13       <BrowserRouter>
14         <App />
15       </BrowserRouter>
16     </PersistGate>
17   </Provider>
18 );
19
```

Рисунок 3.13 – Вміст файлу index.js клієнтської частини

Спочатку ми імпортуємо необхідні модулі та компоненти, включаючи React, ReactDOM, App, BrowserRouter, Provider, persistor, store та PersistGate.

ReactDOM.createRoot використовується для створення кореня React-додатку в елементі з ідентифікатором 'root'.

Root.render використовується для рендерингу кореневого компонента React.

Provider компонент з react-redux використовується для надання доступу до Redux store для всіх компонентів в додатку.

PersistGate компонент з redux-persist використовується для збереження та відновлення стану Redux при перезавантаженні сторінки.

BrowserRouter компонент з react-router-dom використовується для реалізації маршрутизації в додатку.

3.2.1 Розробка головного компонента App.js

Розглянемо детальніше головний компонент App.js, який містить всю його логіку та структуру (рис. 3.14).

App.js 1, M

client > src > App.js > App

```
import './App.css';
import Navbar from './components/navbar/Navbar';
import Footer from './components/footer/Footer';
import Home from './components/home/Home';
import {Routes, Route, Navigate} from 'react-router-dom';
import Login from 'components/login/Login';
import Signup from 'components/signup/Signup';
import Upload from 'components/upload/Upload';
import React from 'react';
import ProfileDetail from 'components/profileDetail/ProfileDetail';
import { useSelector } from 'react-redux';
import PostDetails from 'components/postDetail/PostDetail';

function App() {
  const {user} = useSelector((state) => state.auth)

  return (
    <div>
      <Navbar />
      <Routes>
        <Route path='/' element={user ? <Home /> : <Navigate to='/login' />} />
        <Route path='/login' element={!user ? <Login /> : <Navigate to='/' />} />
        <Route path='/signup' element={!user ? <Signup /> : <Navigate to='/' />} />
        <Route path='/upload' element={user ? <Upload /> : <Navigate to='/login' />} />
        <Route path='/profileDetail/:id' element={user ? <ProfileDetail /> : <Navigate to='/login' />} />
        <Route path='/postDetail/:id' element={user ? <PostDetails /> : <Navigate to='/login' />} />
      </Routes>
      <Footer />
    </div>
  );
}

export default App;
```

Рисунок 3.14 – Вміст файлу App.js

Файл App.js є головним компонентом. Він включає в себе маршрутизацію та основні компоненти додатку.

Опис того що відбувається в файлі (рис 3.14).

Імпортуються необхідні модулі та компоненти, включаючи Navbar, Footer, Home, Routes, Route, Navigate, Login, Signup, Upload, ProfileDetail, useSelector та PostDetails.

Визначається компонент App, який використовує useSelector для отримання поточного користувача з Redux store.

Всередині компонента App рендеряться Navbar, Routes та Footer. Routes включає в себе різні маршрути, які відповідають за рендеринг різних компонентів в залежності від поточного URL.

Кожен маршрут використовує тернарний оператор для перевірки, чи є користувач авторизованим. Якщо користувач авторизований, він переходить до відповідного компонента. Якщо користувач не авторизований, він перенаправляється на сторінку входу.

Наприкінці файлу компонент App експортується за замовчуванням, щоб його можна було використовувати в інших частинах додатку.

3.2.2 Розробка компонента реєстрації Signup.jsx

Файл Signup.jsx є компонентом реєстрації. Він включає в себе форму реєстрації та логіку для обробки подій реєстрації.

Опис того що відбувається в файлі (рис 3.15) та (рис 3.16).

Імпортуються необхідні модулі та компоненти, включаючи React, useState, classes з модуля стилів, зображення, Link, useNavigate з react-router-dom, useDispatch з react-redux та дію register з authSlice.

Визначається компонент Signup, який використовує useState для створення стану для username, email, password та error. Він також використовує useDispatch для створення функції dispatch, яка дозволяє відправляти дії до

Redux store, та useNavigate для створення функції navigate, яка дозволяє перенаправляти користувача до інших маршрутів.

Визначається функція handleSignup, яка викликається при поданні форми. Вона перевіряє, чи заповнені всі поля, а потім виконує POST-запит до сервера для реєстрації нового користувача. Якщо запит успішний, вона відправляє дію register з отриманими даними до Redux store та перенаправляє користувача на головну сторінку. Якщо виникає помилка, вона встановлює error в true на 3 секунди.

В рендерингу компонента Signup відображається форма реєстрації з полями для username, email та password, кнопкою для подання форми та посиланням для переходу до сторінки входу. Якщо error є true, відображається повідомлення про помилку.

Наприкінці файлу компонент Signup експортується за замовчуванням, щоб його можна було використовувати в інших частинах додатку.

Рисунок 3.15 – Вміст файлу Signup.jsx частина 1

Signup.jsx

client > src > components > signup > Signup.jsx > Signup > handleSignup

```
8   const Signup = () => {
42    return (
43      <div className={classes.signUpContainer}>
44        <div className={classes.signUpWrapper}>
45          <div className={classes.signUpLeftSide}>
46            <img src={img} className={classes.leftImg} />
47          </div>
48          <div className={classes.signUpRightSide}>
49            <h2 className={classes.title}>Sigh Up</h2>
50            <form onSubmit={handleSignup} className={classes.signUpForm}>
51              <input type="text" placeholder='Username' onChange={(e) => setUsername(e.target.value)}/>
52              <input type="email" placeholder='Email' onChange={(e) => setEmail(e.target.value)}/>
53              <input type="password" placeholder='Password' onChange={(e) => setPassword(e.target.value)}/>
54              <button type='submit' className={classes.submitBtn}>Sign Up</button>
55              <p>Already have an account? <Link to='/login'>Login</Link></p>
56            </form>
57            {
58              error && (
59                <div className={classes.errorMessage}>
60                  Wrong data
61                </div>
62              )
63            }
64          </div>
65        </div>
66      </div>
67    )
68  }
69
70  export default Signup
```

Рисунок 3.16 – Вміст файлу Signup.jsx частина 2

3.2.3 Розробка компонентів головної сторінки

В нашому випадку головна сторінка Home.jsx (рис. 3.17) використовує наступні компоненти:

– ProfileCard: цей компонент відображає деяку інформацію про профіль користувача;
– SuggestedUsers: цей компонент відображає список користувачів, на яких можна підписатися;
– Posts: цей компонент відображає список постів;
– Rightside: цей компонент відображає список користувачів, на яких ви підписалися.

В свою чергу компонент Posts.jsx (рис. 3.18) використовує інший компонент Post.jsx.

Home.jsx

client > src > components > home > Home.jsx > Home

```jsx
import React from 'react'
import classes from './home.module.css'
import ProfileCard from 'components/profileCard/ProfileCard'
import SuggestedUsers from 'components/suggestedUsers/SuggestedUsers'
import Posts from 'components/posts/Posts'
import Rightside from 'components/rightside/Rightside'

const Home = () => {
  return (
    <div className={classes.container}>
      <div className={classes.left}>
        <ProfileCard />
        <SuggestedUsers />
      </div>
      <Posts />
      <Rightside/>
    </div>
  )
}

export default Home
```

Рисунок 3.17 – Вміст файлу Home.jsx

Posts.jsx 1

client > src > components > posts > Posts.jsx > ...

```jsx
import React from 'react'
import { useEffect } from 'react'
import { useState } from 'react'
import {useSelector} from 'react-redux'
import Post from '../post/Post'
import classes from './posts.module.css'

const Posts = () => {
  const [posts, setPosts] = useState([])
  const {token} = useSelector((state) => state.auth)

  useEffect(() => {
    const fetchPosts = async() => {
      try {
        const res = await fetch(`http://localhost:5000/post/timeline/posts`, {
          headers: {
            'Authorization': `Bearer ${token}`
          }
        })
        const data = await res.json()
        setPosts(data)
      } catch (error) {
        console.error(error)
      }
    }
    fetchPosts()
  }, [])

  return (
    <div className={classes.container}>
      {posts?.map((post) => (
        <Post key={post._id} post={post} />
      ))}
    </div>
  )
}

export default Posts
```

Рисунок 3.18 – Вміст файлу Posts.jsx

3.2.4 Розробка компонента поста

Основною функцією соцмережі Voteween є можливість голосувати між двома фото в одному пості, тому важливо реалізувати зручність цього функціоналу на клієнтській частині.

Функціонал вподобань між двома фотографіями реалізований за допомогою стану React та HTTP-запитів до сервера.

Повний опис функціоналу компоненту Post.jsx (Див. Додаток А Файл Post.jsx).

Стан likedPhotoIndex відслідковує, яке фото користувач вже вподобав. Це може бути 0 (перше фото), 1 (друге фото) або null (жодне фото не вподобано).

Стани firstPhotoLikes та secondPhotoLikes відслідковують кількість вподобань для кожного фото.

При монтуванні компонента використовується useEffect для встановлення початкового значення likedPhotoIndex та кількості вподобань для кожного фото.

Функція handleLikePost викликається, коли користувач натискає на кнопку вподобання. Вона приймає індекс фото як аргумент.

Всередині handleLikePost, виконується HTTP-запит PUT до сервера для вподобання або відміни вподобання фото.

На основі відповіді сервера та поточного стану likedPhotoIndex, виконуються наступні дії:

- якщо користувач вже вподобав фото і натиснув на те ж саме фото, він «відміняє» вподобання;
- якщо користувач вже вподобав одне фото, але натиснув на інше, він «перемикає» своє вподобання;
- якщо користувач ще не вподобав жодного фото, то фото яке «вподобали» вважається тим на яке вперше натиснули.

Після кожної з цих дій, стан likedPhotoIndex, firstPhotoLikes та secondPhotoLikes оновлюється відповідно.

В рендері компонента, використовуються значення likedPhotoIndex, firstPhotoLikes та secondPhotoLikes для відображення відповідних іконок вподобань та кількості вподобань для кожного фото.

3.3 Тестування соцмережі

Тестування програмного забезпечення – це процес, під час якого проводяться експерименти для виявлення помилок і дефектів у програмі. Воно дає змогу переконатися, що ПЗ працює коректно, відповідає вимогам і очікуванням користувачів, а також працює надійно і безпечно.

Ми будемо використовувати ручне тестування – процес, у якому тестувальники виконують тестові сценарії та перевіряють функціональність програмного продукту вручну. Вони стежать за кожним кроком тестового процесу й активно взаємодіють із застосунком, перевіряючи його працездатність, користувацький інтерфейс і відповідність вимогам [17].

При відкритті соцмережі на неавторизованого користувача чекає сторінка реєстрації (рис. 3.19) де він має або ввести дані та стати новим користувачем соцмережі, або ввійти з даними існуючого облікового запису перейшовши за посиланням «Login» (рис. 3.20).

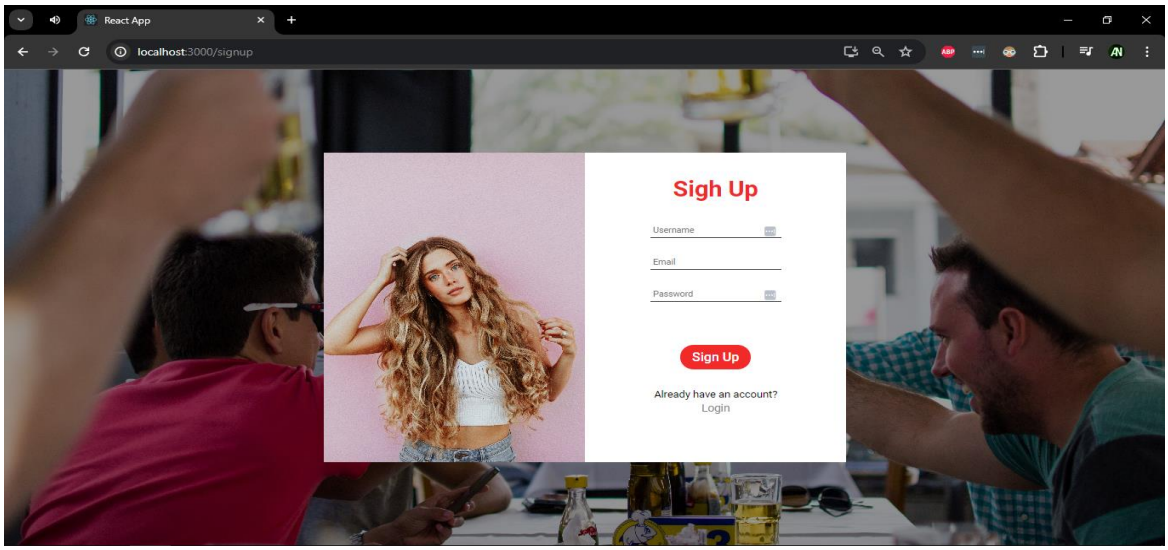

Рисунок 3.19 – Вигляд сторінки реєстрації в соцмережі

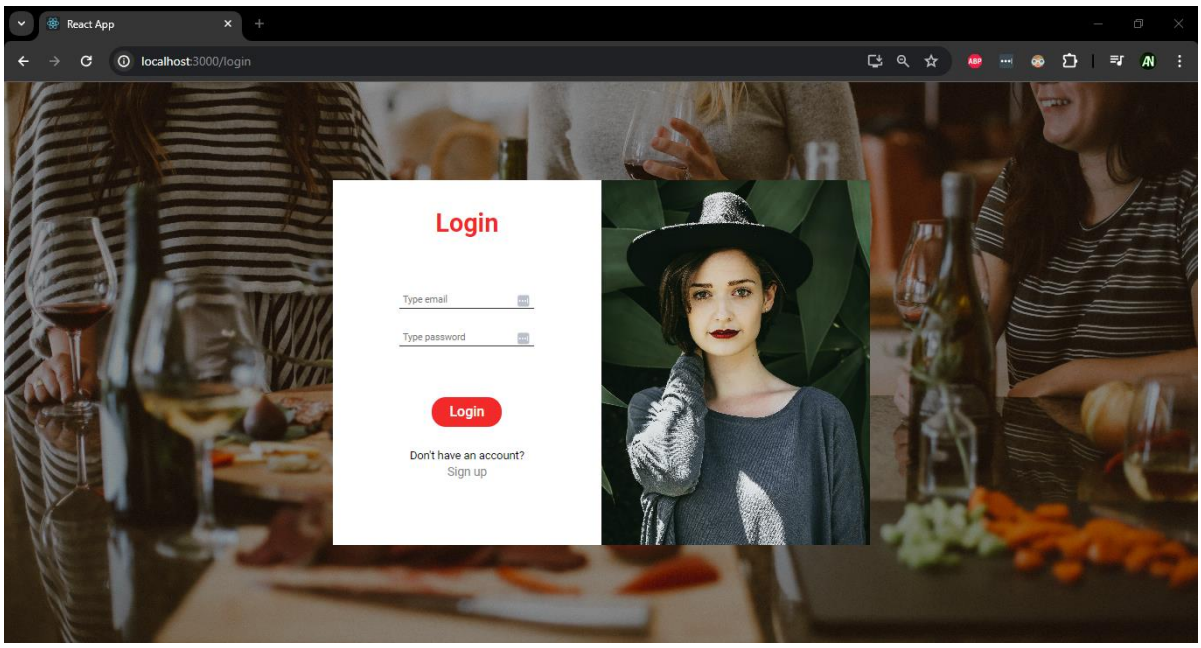

Рисунок 3.20 – Вигляд сторінки авторизації в соцмережі

Після авторизації в соцмережу нас зустрічає головна сторінка (рис. 3.21), де має відображатися ліва частина яка складається з наших даних як користувача та списку доступних користувачів на яких можна підписатись. Центральна частина має складатися з постів інших користувачів на яких ми підписані разом з нашими постами. Права частина має складатися зі списку користувачів на яких ми підписані.

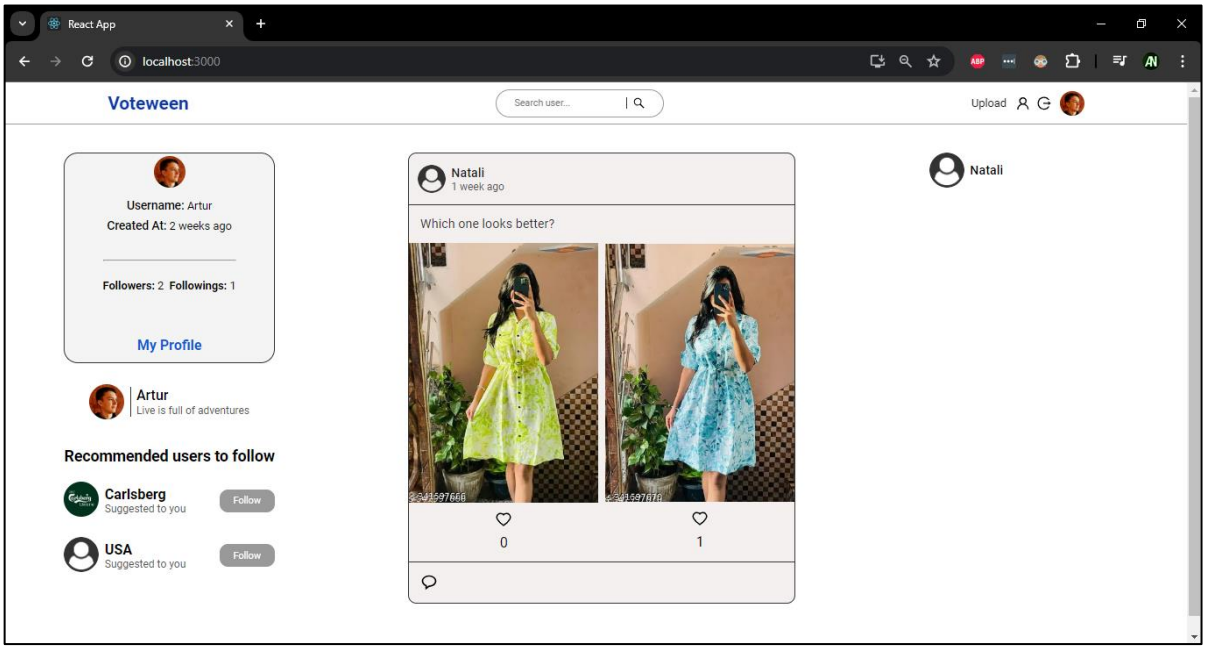

Рисунок 3.21 – Вигляд головної сторінки в соцмережі

Також всі інші сторінки після авторизації будуть мати навігаційну панель «Navbar» (рис 3.21). На навігаційній панелі мають бути функції які надають можливості пошуку користувача за іменем (рис. 3.22), опублікувати пост (рис. 3.23), перейти в особистий профіль (рис. 3.25), оновити дані свого профілю (рис. 3.24), вийти з соцмережі.

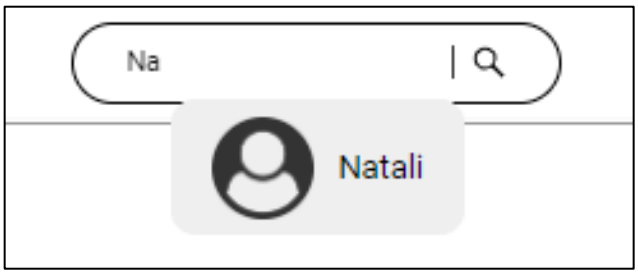

Рисунок 3.22 – Функція пошуку користувача

Натиснувши на посилання з назвою «Upload» нас перенаправляє на сторінку /upload де знаходиться форма публікації посту.

Рисунок 3.23 – Форма публікації поста

Важливо зазначити що публікації в нашій соцмережі можливо робити виключно з двох фото.

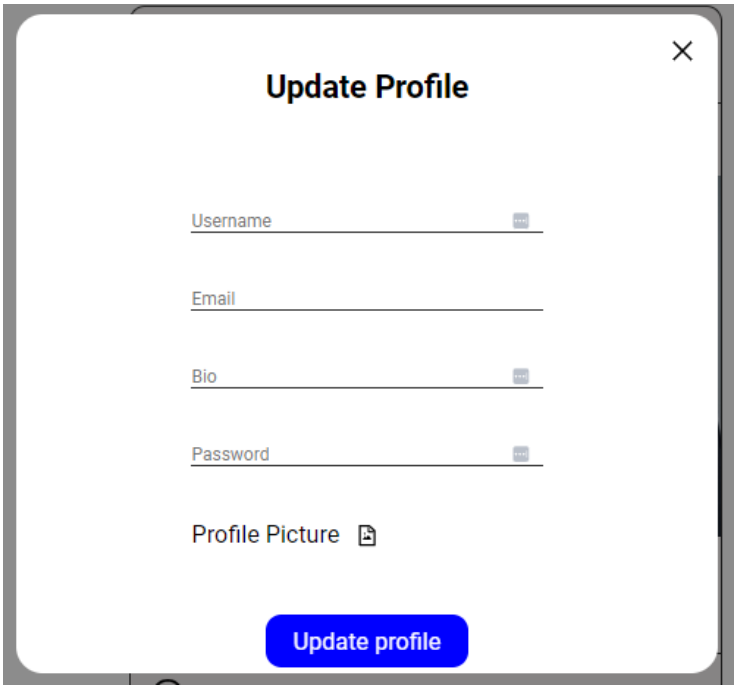

Рисунок 3.24 – Форма зміни даних особистого профілю

Дана форма з’являється в режимі Pop-up поверх контенту на сторінці.

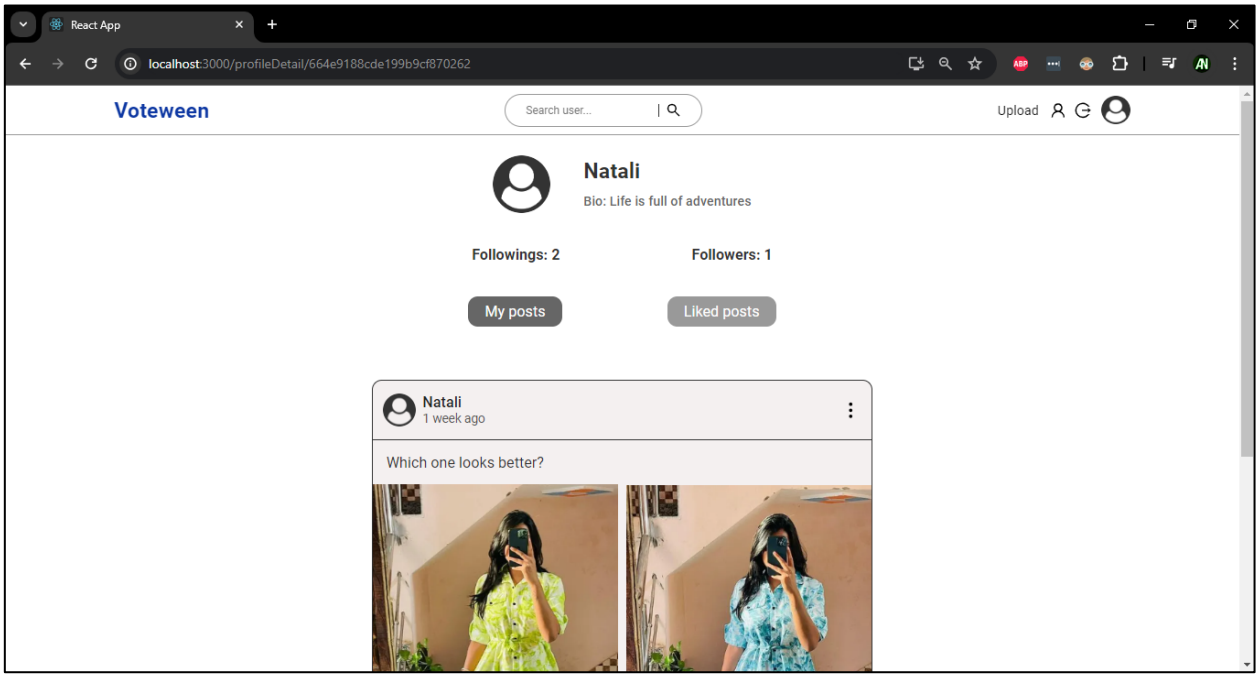

Рисунок 3.25 – Вигляд особистого профілю

В особистому профілі в нас є 2 розділи – Мої пости та Лайкнуті пости де з’являються всі пости що ми вподобали.

Види взаємодії з постом:

– видалити пост (якщо це наш пост);
– перейти в профіль користувача;
– лайкнути одне з фото поста (рис. 3.26);
– прокоментувати пост (рис. 3.27).

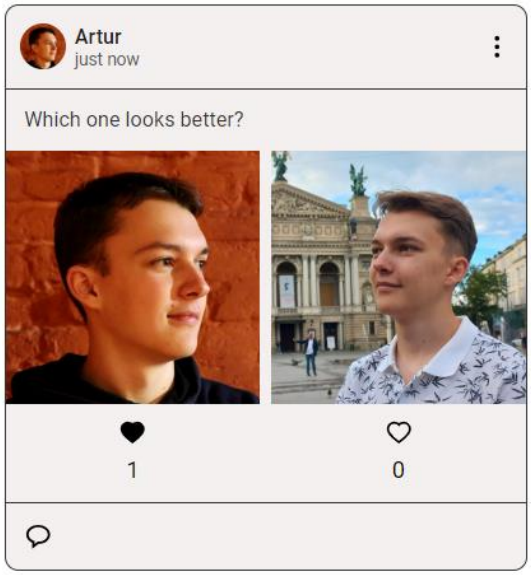

Рисунок 3.26 – Вигляд посту в якому лайкнули одне з фото

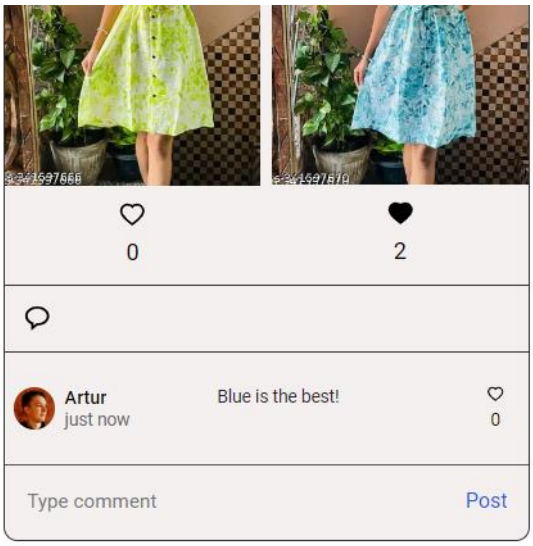

Рисунок 3.27 – Вигляд коментарів до посту

Перейшовши в профіль користувача ми можемо на нього підписатись та побачити всі його пости, але не можемо побачити пости які він вподобав (рис. 3.28).

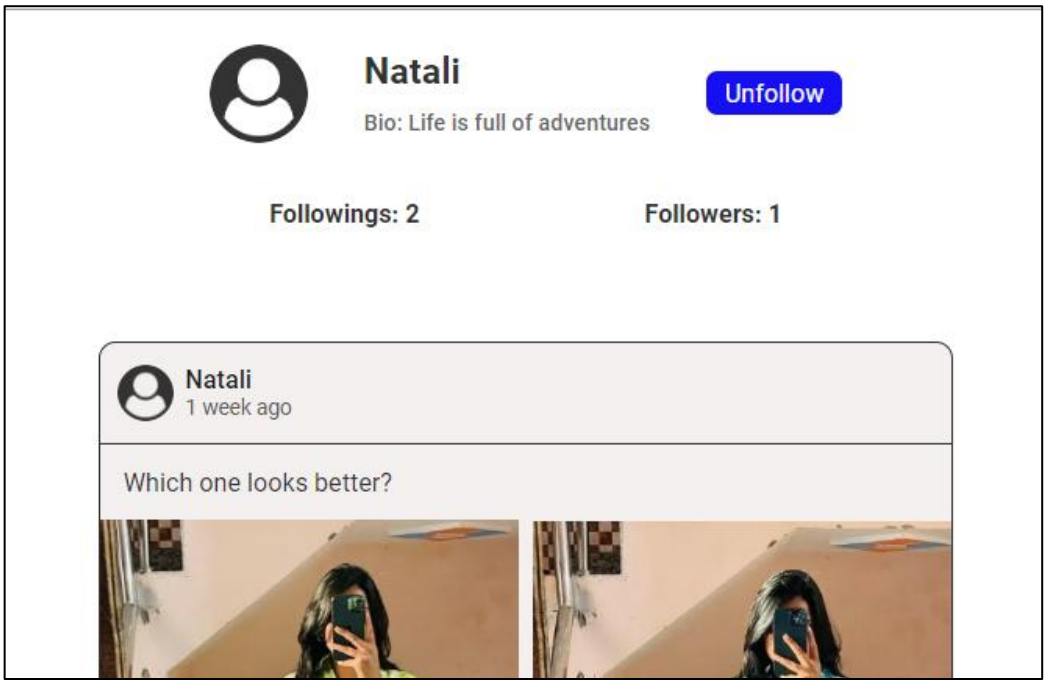

Рисунок 3.28 – Вигляд профілю іншого користувача

ВИСНОВКИ

В результаті кваліфікаційної роботи було успішно розроблено соціальну мережу для прийняття рішень «Voteween» за допомогою стеку MERN, який складається з таких основних інструментів:

- Mongo DB для зберігання даних застосунку, таких як користувацькі профілі, пости, коментарі і т.д.;
- Express.js для побудови серверної логіки застосунку, маршрутизації запитів, взаємодії з базою даних і обробки HTTP-запитів і відповідей;
- React.js для побудови клієнтської частини застосунку, тобто інтерфейсу користувача, React.js дозволяє створювати інтерактивні і динамічні вебсторінки, що реагують на дії користувача;
- Node.js для запуску серверної частини застосунку, надаючи середовище для виконання JavaScript-коду на сервері, а також для створення серверних API та логіки.

На етапі проєктування при розробці структури соціальної мережі «Voteween» було спроєктовано просту та зручну структуру для користувачів, яка використовує елементи гібридного підходу.

Також були спроєктовані такі діаграми як:

- діаграма класів яка описує основні класи та зв’язок між ними;
- діаграма послідовності для процесу «Створення поста» на якій візуально зображено як відбувається даний процес на кожному з об’єктів діаграми;
- діаграма прецедентів на якій зображено які частини доступу має той чи інший актор системи;
- діаграма ER на якій зображено відношення та сутності соцмережі.

ПЕРЕЛІК ПОСИЛАНЬ

1. Shama H. Full-Stack React Projects: Modern web development using React 16, Node, Express, and MongoDB. Packt Publishing, 2018. 46 p.
2. Kirupa C. Learning React: A Hands-On Guide to Building Web Applications Using React and Redux. Addison-Wesley Professional, 2018. 32 p.
3. Rud A. Що таке Node.js та для чого він потрібен? URL: https://hyperhost.ua/info/uk/shho-take-nodejs-ta-dlya-cogo-vin-potriben (дата звернення: 22.01.2024).
4. Express JS – Фреймворк для веб-застосунків, побудованих на Node.js. URL: https://expressjs.com/uk/ (дата звернення: 30.01.2024).
5. Express-validator introduction. URL: https://express-validator.github.io/docs (дата звернення: 08.02.2024).
6. Koziuliya A. Все, що потрібно знати про бази даних. URL: https://dan-it.com.ua/uk/blog/vse-shho-potribno-znati-pro-bazi-danih-dlja-pochatkivciv-mysql-postgresql-mongodb/ (дата звернення: 19.02.2024).
7. Hall J. Getting Started with MongoDB & Mongoose. URL: https://www.mongodb.com/developer/languages/javascript/getting-started-with-mongodb-and-mongoose/ (дата звернення: 26.02.2024).
8. Як використовувати JSON Web Tokens (JWT) для автентифікації URL: https://devzone.org.ua/post/iak-vykorystovuvaty-json-web-tokens-jwt-dlia-avtentyfikatsiyi (дата звернення: 04.03.2024).
9. Provos N., Mazier D. A Future-Adaptable Password Scheme. URL: https://www.openbsd.org/papers/bcrypt-paper.pdf (дата звернення: 17.03.2024).
10. Multer – middleware for handling multipart/form-data. URL: https://expressjs.com/en/resources/middleware/multer.html (дата звернення: 27.03.2024).
11. Що таке React JS і для чого він потрібен? URL: https://dan-

it.com.ua/uk/blog/chto-takoe-react-js-i-dlja-chego-on-nuzhen/ (дата звернення: 05.04.2024).

12. Для чого і коли використовується Redux. URL: https://foxminded.ua/shcho-take-redux/ (дата звернення: 14.04.2024).
13. React Router DOM: How to handle routing in web apps. URL: https://blog.logrocket.com/react-router-dom-tutorial-examples/ (дата звернення: 23.04.2024).
14. Іванина Р., Шепель Є. Як вивести правильну структуру сайту. URL: https://elit-web.ua/ua/blog/kak-vyvesti-tekuschuju-strukturu-sajta (дата звернення: 01.05.2024).
15. Марголін О. UML для бізнес-моделювання: для чого потрібні діаграми процесів. URL: https://evergreens.com.ua/ua/articles/uml-diagrams.html (дата звернення: 01.05.2024).
16. Peterson R. Entity Relationship (ER) Diagram Model with DBMS Example. URL: https://www.guru99.com/er-diagram-tutorial-dbms.html (дата звернення: 03.05.2024).
17. Тестування програмного забезпечення: типи, види та застосування. URL: https://foxminded.ua/testuvannia-prohramnoho-zabezpechennia/ (дата звернення: 03.05.2024).

ДОДАТОК А

Файл Post.jsx

```
import React from 'react'
import { useEffect } from 'react'
import { useState } from 'react'
import { useDispatch, useSelector } from 'react-redux'
import { format } from 'timeago.js'
import { Link } from 'react-router-dom'
import man from '../../assets/man.jpg'
import { capitalizeFirstLetter } from '../../util/capitalizeFirstLetter'
import { HiOutlineDotsVertical } from 'react-icons/hi'
import { AiFillHeart, AiOutlineHeart } from 'react-icons/ai'
import { BiMessageRounded } from 'react-icons/bi'
import classes from './post.module.css'
import Comment from '../comment/Comment'
import { likePost } from '../../redux/authSlice'

const Post = ({ post }) => {
  const { user, token } = useSelector((state) => state.auth)
  const [comments, setComments] = useState([])
  const [commentText, setCommentText] = useState('')
  const [isCommentEmpty, setIsCommentEmpty] = useState(false)
  const [showDeleteModal, setShowDeleteModal] = useState(false)
  const [showComment, setShowComment] = useState(false)
  const dispatch = useDispatch()

  const [likedPhotoIndex, setLikedPhotoIndex] = useState(null);

  const [firstPhotoLikes, setFirstPhotoLikes] = useState(0);
  const [secondPhotoLikes, setSecondPhotoLikes] = useState(0);

  useEffect(() => {
    const initialLikedPhotoIndex = post.likes[user._id];
    setLikedPhotoIndex(initialLikedPhotoIndex);
  }, [post, user._id]);

  useEffect(() => {
    const likesArray = Object.values(post.likes);
    const firstPhotoLikes = likesArray.filter(like => like === 0).length;
    const secondPhotoLikes = likesArray.filter(like => like === 1).length;
    setFirstPhotoLikes(firstPhotoLikes);
    setSecondPhotoLikes(secondPhotoLikes);
  }, [post]);

  useEffect(() => {
    const fetchComments = async () => {
```

```
      try {
        const res = await fetch(`http://localhost:5000/comment/${post._id}`, {
          headers: {
            "Authorization": `Bearer ${token}`
          }
        })

        const data = await res.json()
        setComments(data)
      } catch (error) {
        console.error(error)
      }
    }
    fetchComments()
  }, [post._id])

  const deletePost = async () => {
    try {
      await fetch(`http://localhost:5000/post/${post._id}`, {
        headers: {
          "Authorization": `Bearer ${token}`
        },
        method: 'DELETE'
      })
      window.location.reload()
    } catch (error) {
      console.error(error)
    }
  }

  const handleLikePost = async(photoIndex) => {
    try {
      const res = await
fetch(`http://localhost:5000/post/like/${post._id}/${photoIndex}`, {
        headers: {
          "Authorization": `Bearer ${token}`

        },
        method: "PUT"
      })
      const updatedPost = await res.json();

      if (likedPhotoIndex != null) {
        // Change like photo
        if (likedPhotoIndex != photoIndex) {
          setLikedPhotoIndex(photoIndex)
          if (photoIndex == 0) {
            setFirstPhotoLikes(firstPhotoLikes + 1)
            setSecondPhotoLikes(secondPhotoLikes - 1)
          } else {
            setFirstPhotoLikes(firstPhotoLikes - 1)
```

```
          setSecondPhotoLikes(secondPhotoLikes + 1)
        }
      }
      // Dislike
      else {
        setLikedPhotoIndex(null)
        if (photoIndex == 0) {
          setFirstPhotoLikes(firstPhotoLikes - 1);
        } else {
          setSecondPhotoLikes(secondPhotoLikes - 1);
        }
        dispatch(likePost(updatedPost))
      }
    }
    // Like
    else {
      setLikedPhotoIndex(photoIndex)
      if (photoIndex == 0) {
        setFirstPhotoLikes(firstPhotoLikes + 1);
      } else {
        setSecondPhotoLikes(secondPhotoLikes + 1);
      }
      dispatch(likePost(updatedPost))
    }
  } catch (error) {
    console.error(error)
  }
}

const handlePostComment = async () => {
  if (commentText === '') {
    setIsCommentEmpty(true)
    setTimeout(() => {
      setIsCommentEmpty(false)
    }, 2000)
    return
  }

  try {
    const res = await fetch(`http://localhost:5000/comment`, {
      headers: {
        "Content-Type": 'application/json',
        "Authorization": `Bearer ${token}`
      },
      method: 'POST',
      body: JSON.stringify({ commentText, post: post._id })
    })

    const data = await res.json()

    setComments(prev => [...prev, data])
```

```
      setCommentText('')
    } catch (error) {
      console.error(error)
    }
  }


  return (
    <div className={classes.container}>
      <div className={classes.wrapper}>
        <div className={classes.top}>
          <Link to={`/profileDetail/${post?.user?._id}`}
className={classes.topLeft}>
            <img src={post?.user?.profileImg ?
`http://localhost:5000/images/${post?.user?.profileImg}` : man}
className={classes.profileUserImg} />
            <div className={classes.profileMetadata}>
              <span>{capitalizeFirstLetter(post.user.username)}</span>
              <span>{format(post.createdAt)}</span>
            </div>
          </Link>
          {
            (user._id === post.user._id) &&
            <HiOutlineDotsVertical size={25} onClick={() =>
setShowDeleteModal(prev => !prev)} />
          }
          {
            showDeleteModal && (
              <div className={classes.deleteModal}>
                <h3>Delete Post</h3>
                <div className={classes.buttons}>
                  <button onClick={deletePost}>Yes</button>
                  <button onClick={() => setShowDeleteModal(prev =>
!prev)}>No</button>
                </div>
              </div>
            )
          }
        </div>
        <div className={classes.center}>
          <div className={classes.desc}>{post.desc}</div>
          <div className={classes.postImgs}>
            <div className={classes.postImg}>
                <Link to={`/postDetail/${post._id}`} className={classes.postImg}>
                  <img src={post?.firstImg ?
`http://localhost:5000/images/${post?.firstImg}` : ''} />
                </Link>
                <div className={classes.likeBtn}>
                {likedPhotoIndex === 0 ? <AiFillHeart onClick={() =>
handleLikePost(0)} /> : <AiOutlineHeart onClick={() => handleLikePost(0)} />}
                </div>
```

```
                <p>{firstPhotoLikes}</p>
            </div>
            <div className={classes.postImg}>
                <Link to={`/postDetail/${post._id}`} className={classes.postImg}>
                  <img src={post?.secondImg ?
`http://localhost:5000/images/${post?.secondImg}` : ''} />
                </Link>
                <div className={classes.likeBtn}>
                {likedPhotoIndex === 1 ? <AiFillHeart onClick={() =>
handleLikePost(1)} /> : <AiOutlineHeart onClick={() => handleLikePost(1)} />}
                </div>
                <p>{secondPhotoLikes}</p>
            </div>
          </div>
        </div>
        <div className={`${classes.controls} ${showComment &&
classes.showComment}`}>
          <div className={classes.controlsLeft}>
            <BiMessageRounded onClick={() => setShowComment(prev => !prev)} />
          </div>
        </div>
        {
          showComment &&
          <>
            <div className={classes.comments}>
              {
                comments?.length > 0 ? comments.map((comment) => (
                  <Comment c={comment} key={comment._id} />
                )) : <span style={{ marginLeft: '12px', fontSize: '20px' }}>No
comments</span>
              }
            </div>
            <div className={classes.postCommentSection}>
              <input
                value={commentText}
                onChange={(e) => setCommentText(e.target.value)}
                type="text"
                className={classes.inputSection}
                placeholder='Type comment'
              />
              <button onClick={handlePostComment}>Post</button>
            </div>
            {isCommentEmpty && <span className={classes.emptyCommentMsg}>You
can't post empty comment!</span>}
          </>
        }
      </div>
    </div>
  )
}
export default Post
```