МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Сумський державний університет

Факультет електроніки та інформаційних технологій

Кафедра комп'ютерних наук

«До захисту допущено»

В.о. завідувача кафедри

Ігор ШЕЛЕХОВ

(підпис)

01 червня 2024р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня бакалавра

зі спеціальності 122 – Комп'ютерні науки,

освітньо-професійної програми «Інформатика»

на тему: «Інформаційна система управління контентом веб-форуму»

здобувача групи ІН – 01 Гриценка Олександра Ігоревича

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

Олександр ГРИЦЕНКО

(підпис)

Керівник,

асистент кафедри комп'ютерних наук,

кандидат фізико-математичних наук Ольга ШУТИЛЄВА

(підпис)

Суми – 2024

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«Затверджую»
В.о. завідувача кафедри
_____ Ігор ШЕЛЕХОВ
(підпис)

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавра

зі спеціальності 122 – Комп'ютерні науки, освітньо-професійної програми «Інформатика»
здобувача групи ІН- 01 Гриценка Олександра Ігоревича

1. Тема роботи: «Інформаційна система управління контентом веб-форуму» затверджена наказом по СумДУ від *«25» травня 2024 р. №0570-VI*
2. Термін здачі здобувачем кваліфікаційної роботи *до 01 червня 2024_ року*
3. Вхідні дані до кваліфікаційної роботи _____
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) *1) Аналіз проблеми предметної області, постановка й формування завдань дослідження. 2) Огляд технологій, що використовуються для розробки. 3) Розробка інформаційної системи* управління контентом веб-форуму. *4)Аналіз результатів.*
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)_____
6. Консультанти до проєкту (роботи), із значенням розділів проєкту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «08» квітня 2024 р.

Завдання прийняв до виконання _____ Керівник _____
(підпис) (підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	*Аналіз проблеми предметної області, постановка й формування завдань дослідження*	06.05.24-...	
2	*Огляд технологій, що використовуються для створення інформаційної системи управління контентом веб-форуму*		
3	*Створення інформаційної системи управління контентом веб-форуму*		
4	*Аналіз отриманих результатів*		
5	*Оформлення пояснювальної записки до кваліфікаційної роботи*	…- 31.05.24	

Здобувач вищої освіти _____ Керівник _____
(підпис) (підпис)

АНОТАЦІЯ

Записка: 72 стор., 35 рис., 1 додаток, 33 використаних джерел.

Обґрунтування актуальності теми роботи – Тема кваліфікаційної роботи є актуальною, оскільки присвячена розробці інформаційної системи управління контентом веб-форуму, спеціалізованого на обговоренні тату. Веб-форуми залишаються важливими платформами для обміну досвідом та знаннями. Створення ефективних методів і технологій для управління контентом сприяє покращенню якості обслуговування, підвищенню безпеки та зручності використання, що є актуальним у сучасних умовах розвитку веб-технологій.

Об’єкт дослідження — процес адміністрування форуму.

Мета роботи — розробка інформаційної система управління контентом веб-форуму з використанням з адмініструванням.

Методи дослідження — система керування контентом та користувачами на веб-форумах, інструменти розробки веб-додатків, а також методи прототипування та тестування для забезпечення ефективного функціонування інформаційної системи управління контентом.

Результати — розроблено інформаційну систему управління контентом веб-форуму, спеціалізованого на обговоренні тату. Система забезпечує реєстрацію та авторизацію користувачів, створення та управління постами, додавання зображень, лайки, відповіді на пости, а також отримання стрічки нових постів від користувачів. Проведено тестування розробки для забезпечення її стабільної та безпечної роботи.

ІНФОРМАЦІЙНА СИСТЕМА, УПРАВЛІННЯ КОНТЕНТОМ, ВЕБ-ФОРУМ, MERN, REACT, NODE.JS, MONGODB.

ЗМІСТ

ВСТУП

Актуальність. У сучасному світі, де інформаційні технології відіграють ключову роль у формуванні соціальних, економічних та культурних процесів, важливість надійних та ефективних систем управління контентом не може бути переоцінена. Особливе місце серед таких систем займають інформаційні системи для управління веб-форумами, які стають майданчиком для обговорень, обміну думками та знаннями між користувачами з усього світу. Веб-форуми не лише слугують місцем для спілкування, але й є значним ресурсом для збору та аналізу даних про публічні інтереси та потреби, що робить їх незамінним інструментом у багатьох сферах діяльності. Розвиток технологій відкрив нові можливості для удосконалення інформаційних систем управління веб-форумами, зокрема через використання сучасних вебтехнологій і методологій програмування. Системи, які здатні ефективно керувати великими обсягами даних, забезпечувати високий рівень безпеки та пропонувати гнучкі налаштування під конкретні завдання, стають вирішальними для успіху веб-платформ [1].

Об’єкт дослідження – процес розробки та управління інформаційними системами для веб-форумів.

Предмет дослідження – інформаційна система управління контентом для веб-форуму на базі MERN стеку (MongoDB, Express.js, React, Node.js).

Мета роботи – створити інформаційну систему управління контентом для веб-форуму, яка відповідатиме сучасним вимогам до функціональності, зручності інтерфейсу, безпеки, масштабованості та легкості в обслуговуванні.

Новизна результатів дослідження полягає у розробці системи управління контентом для веб-форуму, що використовує сучасний MERN стек (MongoDB, Express.js, React, Node.js). Це дозволить інтегрувати передові технології, забезпечити високу продуктивність, адаптивність та гнучкість у розширенні функціональності вебсайту. Крім того, система буде створена з урахуванням найкращих практик та потреб, які не були задоволені існуючими рішеннями, такими як phpBB, vBulletin, XenForo та Discourse.

Гіпотеза цього дослідження полягає в тому, що розробка сучасної інформаційної системи управління контентом для веб-форуму з використанням MERN стеку забезпечить покращений користувацький досвід, високу продуктивність, безпеку та масштабованість, сприяючи активному обміну інформацією та залученню користувачів.

Структура роботи складається зі вступу, огляду сучасних технологій управління проєктами та задачами, аналізу програмних продуктів-аналогів, постановки задачі, вибору інструментів для розробки, практичної реалізації та огляду використання програмного додатку, висновків, списку використаних джерел та додатків.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Вивчення існуючих систем

Інформаційна система управління контентом веб-форуму – це складний комплекс програмного забезпечення, призначений для створення, управління та оптимізації контенту в рамках онлайн-форумів. Ці системи надають інструменти для модерації дискусій, управління користувацькими акаунтами, а також для організації і структурування інформації. Їх основна мета – забезпечити зручне спілкування між користувачами та ефективне розподіл контенту. Основними компонентами інформаційної системи управління контентом веб-форуму є модулі а саме: управління контентом, управління користувачами, безпека, інтерфейс користувача, та пошук і фільтрації.

Для початку розглянемо один з найважливіших модулів інформаційної системи управління контентом веб-форуму – модуль управління контентом. Він об'єднує інструменти для створення, редагування, видалення та організації публікацій і тем на форумі, дозволяє категоризувати теми, впроваджувати теги для легшого пошуку та структурувати дискусії. Важливим є також модуль управління користувачами, який займається реєстрацією нових користувачів, управлінням їхніми профілями, а також наданням та обмеженням прав доступу, включаючи налаштування рівнів доступу для модераторів, адміністраторів та звичайних користувачів. Паралельно, модуль безпеки гарантує захист від несанкціонованого доступу та зловмисного втручання, включаючи аутентифікацію користувачів, шифрування даних і інструменти для виявлення і блокування спаму та вірусів. Додатково, модуль пошуку та фільтрації полегшує користувачам навігацію по контенту форуму завдяки розширеним функціям пошуку за ключовими словами, авторами постів, датами публікацій чи індексованими тегами. Завершує структуру системи інтерфейс користувача, який є ключовим для зручності користування та загального враження від системи; він повинен бути інтуїтивно зрозумілим, привабливим і адаптованим

під різні пристрої та розміри екранів.

Переваги використання інформаційних систем для управління контентом веб-форумів:

- Ефективне управління спільнотою. Централізоване управління дозволяє адміністраторам та модераторам легко контролювати активність на форумі, оперативно реагувати на порушення правил;
- Підвищення зацікавленості та утримання користувачів. Системи надають засоби для гейміфікації, винагород за активність, стимулюючи користувачів до більш активної участі у житті форуму;
- Адаптація під потреби бізнесу. Гнучкість у налаштуванні функцій та інтерфейсу під конкретні потреби та цілі спільноти.

Використання сучасних технологій та підходів в розробці інформаційних систем дозволяє створювати високоефективні та зручні інструменти для управління контентом, які відповідають сучасним вимогам до інтерактивності та безпеки.

1.2 Аналіз програмних продуктів-аналогів

Для аналізу існуючих рішень у сфері управління контентом веб-форумів можна розглянути декілька популярних платформ, кожна з яких має свої унікальні особливості та інструменти. Ось детальний огляд декількох відомих систем

phpBB — це одна з найстаріших платформ для створення веб-форумів, що базується на PHP [2]. Це відкрите програмне забезпечення, яке з'явилося у 2000 році і з того часу отримало широке поширення завдяки своїй гнучкості та активній спільноті користувачів. Основною перевагою phpBB є його модульність, що дозволяє користувачам легко додавати нові функції та адаптувати форум до своїх потреб. Платформа надає велику кількість шаблонів та доповнень, які можуть бути інтегровані для розширення функціоналу або зміни зовнішнього вигляду форуму. Підтримка багатомовності дозволяє

використовувати phpBB в міжнародних проектах.

Рисунок 1.1 – Візуальний вигляд **phpBB**

Особливості:

- Розширені права доступу та приватні повідомлення: Можливість детально налаштовувати права користувачів та груп;
- Підтримка різноманітних додатків: Бібліотека доповнень для налаштування форуму;
- Інструменти модерації: Комплексні засоби для управління контентом та користувачами.

vBulletin – це комерційне рішення для створення веб-форумів, яке вирізняється багатофункціональністю і масштабованістю [3]. Заснований у 2000 році, vBulletin швидко став популярним серед великих спільнот завдяки стабільності та різноманітності вбудованих інструментів. Платформа пропонує глибокі можливості для налаштування, включаючи власну систему шаблонів, розширений контент-менеджмент та інтеграцію з соціальними медіа. Хоча

vBulletin є платною платформою, його професійний підхід та високий рівень підтримки роблять його вибором для серйозних проєктів.

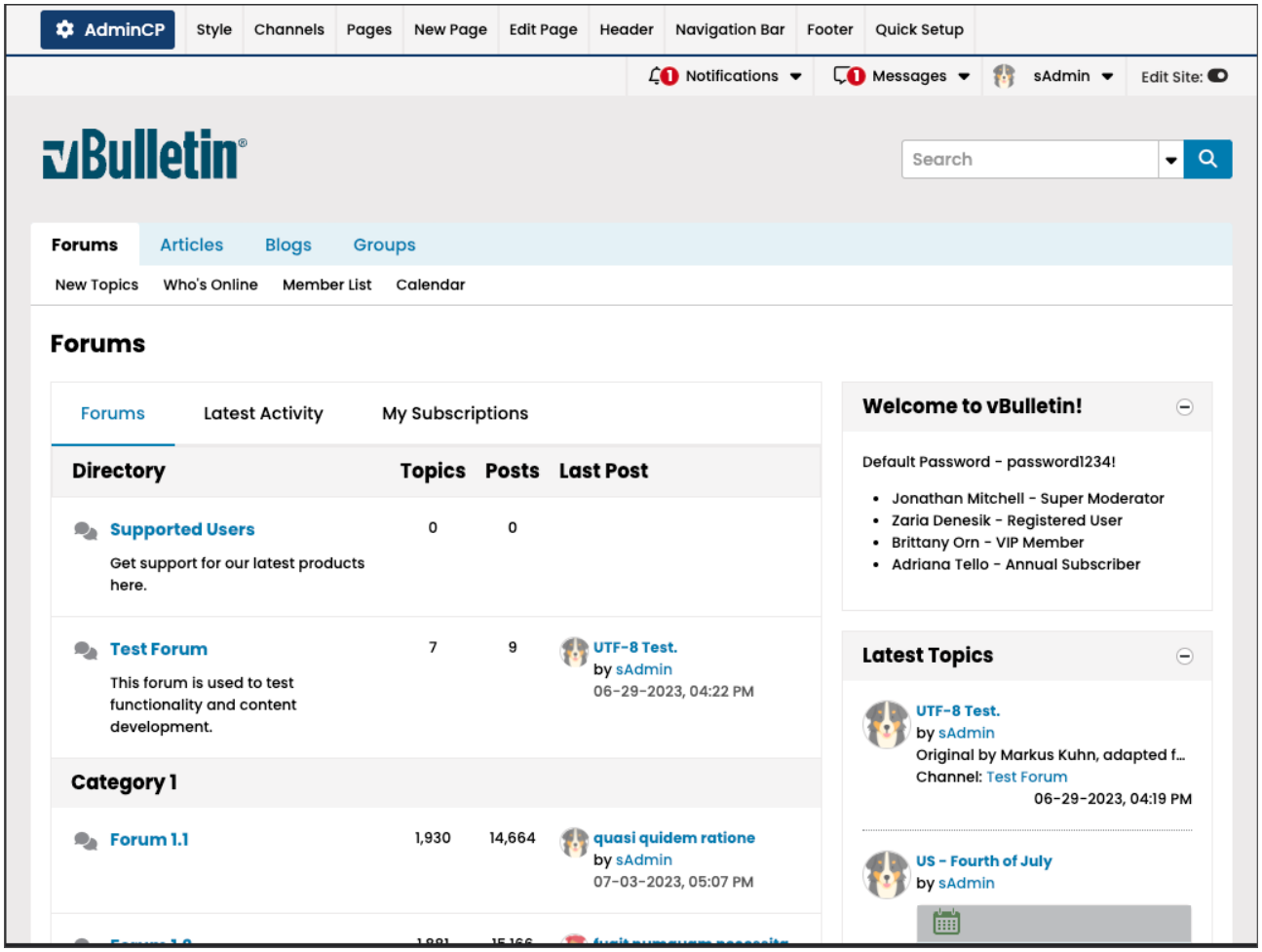

Рисунок 1.2 – Візуальний вигляд **vBulletin**

Особливості:

● Інтегрований контент-менеджер: Легке створення та управління статтями.

● Система лайків та рейтингів: Взаємодія користувачів через лайки та оцінки.

● Мобільні додатки: Підтримка мобільних версій для доступу на ходу.

XenForo Запущений у 2010 році, XenForo відразу здобув репутацію завдяки своєму сучасному інтерфейсу та широкому набору функцій. Розробники XenForo зосередилися на забезпеченні кращого досвіду користувача,

інтегрувавши сучасні технології, такі як ажурні обговорення та високу кастомізацію. Платформа також акцентує увагу на соціальних інтеракціях між користувачами, надаючи інструменти для залучення та втримання аудиторії [4].

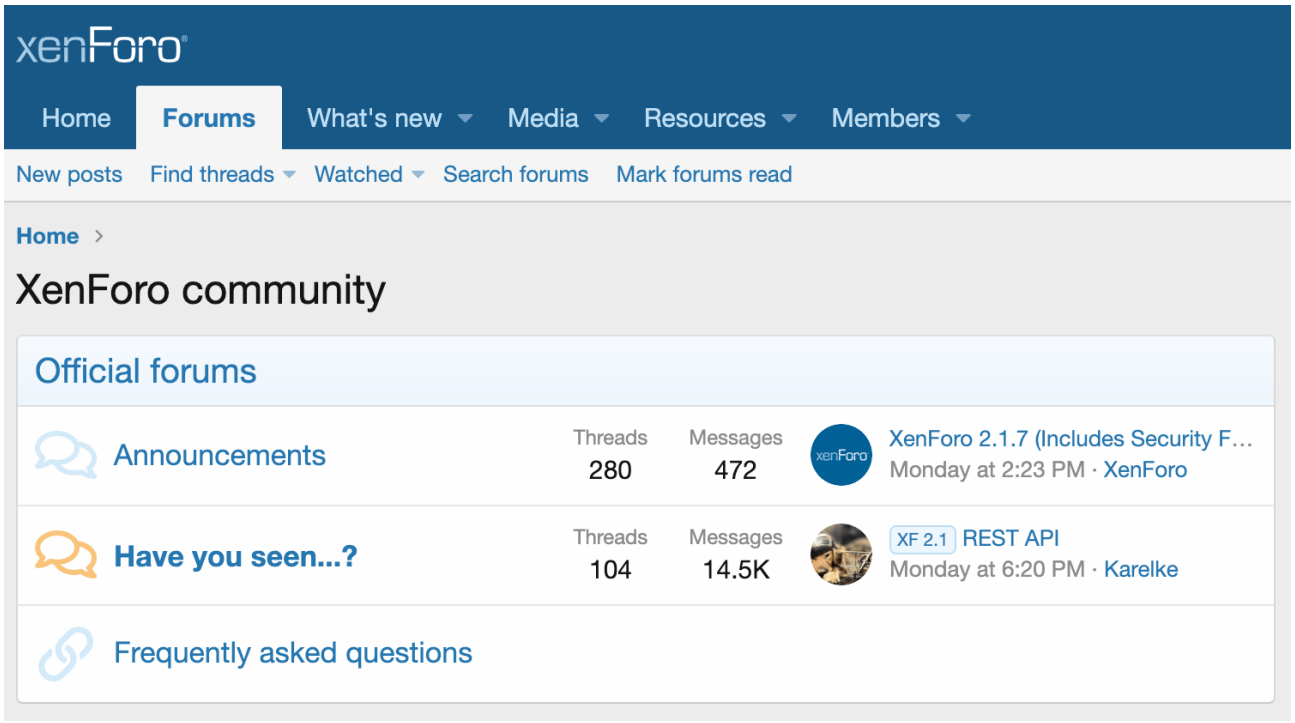

Рисунок 1.3 – Візуальний вигляд **XenForo**

Особливості:

- Соціальні мережеві функції: "Лайки", "репутація", обговорення;
- Оповіщення в реальному часі: Швидке інформування користувачів про оновлення;
- Підтримка сторонніх додатків: Можливість доповнювати форум новими функціями.

Discourse — це новітня платформа для веб-форумів, що відрізняється від своїх конкурентів зосередженням на забезпеченні найкращих практик спілкування та модерації. Заснована у 2013 році, Discourse розроблена з акцентом на простоту інтеграції, відкритість коду і використання новітніх веб-технологій [5]. Її можна використовувати як на власних серверах, так і в хмарі, що робить її доступною для широкого кола проектів.

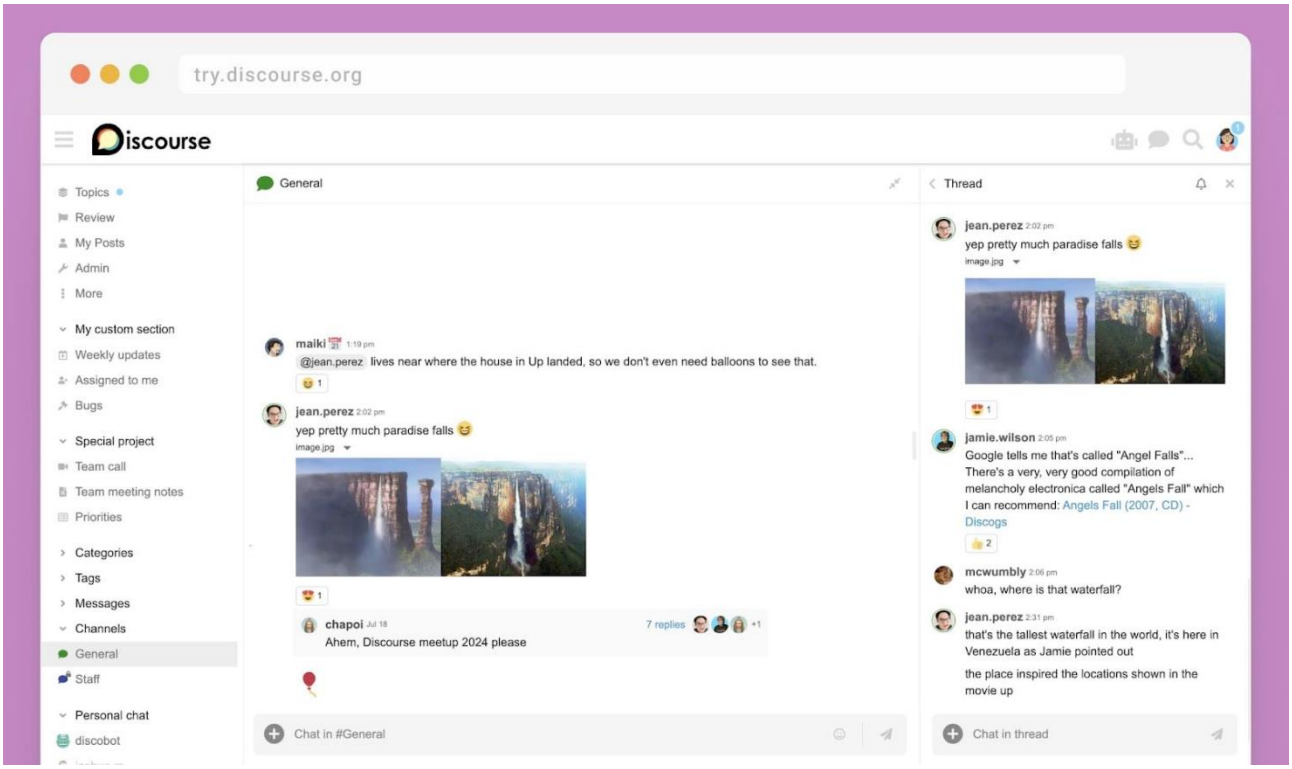

Рисунок 1.4 – Візуальний вигляд **Discourse**

Особливості:

- Мобільно-орієнтований дизайн: Оптимізація під смартфони та планшети;
- Інтеграція з сервісами: Підключення до популярних веб-сервісів;
- Розширені інструменти аналітики: Вбудовані засоби для аналізу активності користувачів.

Таблиця 1.1 – Порівняльна таблиця характеристик аналогів

Плат.	**Плюси**	**Мінуси**	**Технічна підтримка і спільнота**
phpBB	● Відкрите програмне забезпечення ● Гнучкі налаштування та багато доповнень ● Багатомовна підтримка ● Сильна спільнота розробників	● Застарілий інтерфейс без налаштувань ● Вищі вимоги до технічних знань для налаштування	● Активний форум користувачів ● Велика кількість документації та допомоги онлайн
vBulletin	● Інтегрований контент-менеджер для статей та блогів	● Висока вартість ліцензії	● Професійна підтримка від розробників

	● Підтримка мобільних додатків ● Розширені SEO налаштування	● Може вимагати значних ресурсів сервера	● Оплачувані послуги технічної підтримки
XenForo	● Сучасний інтерфейс і висока кастомізація ● Сильні соціальні інструменти (лайки, репутація) ● Швидкі оповіщення в реальному часі	● Вартість ліцензії ● Потребує регулярних оновлень для забезпечення безпеки	● Активна спільнота розробників та користувачів ● Форуми підтримки та розширення функціоналу
Discourse	● Відкритий код і сучасні веб-технології ● Чудова підтримка мобільних пристроїв ● Великі можливості інтеграції з іншими сервісами	● Вищі вимоги до хостингу (Ресурси сервера) ● Може бути складніше в налаштуванні для новачків	● Велика і дуже активна спільнота розробників ● Багато ресурсів для самостійної допомоги та налаштування

1.3 Головні принципи роботи веб-форуму

Основні принципи роботи веб-форуму є критичними для забезпечення його ефективного функціонування та задоволення потреб користувачів. Розуміння цих принципів є ключем до створення успішної платформи для обговорення та обміну інформацією.

Насамперед, веб-форум повинен забезпечувати надійне та безпечне середовище для користувачів. Це означає впровадження сучасних методів аутентифікації та авторизації, які гарантують, що тільки зареєстровані користувачі можуть створювати нові теми, коментувати та взаємодіяти з контентом. Крім того, необхідно забезпечити захист від спаму та ботів за допомогою капчі та інших методів верифікації, що знижують ризик автоматизованих атак. Важливим аспектом є також зберігання даних користувачів у захищеному вигляді, використовуючи шифрування та інші методи захисту даних.

Важливою складовою успішного веб-форуму є зручність використання та інтуїтивно зрозумілий інтерфейс. Веб-форум має бути простим у навігації, щоб

користувачі могли легко знаходити потрібну інформацію та брати участь у дискусіях. Для цього необхідно забезпечити логічну структуру категорій та підфорумів, що дозволяє швидко орієнтуватися у великому обсязі контенту. Крім того, інтерфейс повинен бути адаптивним, тобто коректно відображатися на різних пристроях – від десктопів до мобільних телефонів. Використання сучасних технологій, таких як реактивні бібліотеки і фреймворки, допомагає створити швидкий та зручний інтерфейс.

Якісний та структурований контент є ще одним важливим аспектом. Важливість його модерації та контролю контенту для підтримання високого рівня дискусій та запобігання поширенню небажаного або шкідливого контенту є дуже важливою. Модератори відіграють ключову роль у цьому процесі, контролюючи дотримання правил форуму та підтримуючи позитивну атмосферу для обговорень. Важливо впровадити механізми звітування про порушення, що дозволяє користувачам повідомляти модераторам про невідповідний контент.

Інтерактивність та взаємодія користувачів є невід'ємною частиною успішного веб-форуму. Платформа повинна надавати різноманітні інструменти для взаємодії, такі як системи лайків, рейтинги, можливість цитування повідомлень та інші соціальні функції. Це сприяє залученню користувачів та підтримці активності на форумі. Крім того, важливо забезпечити можливість приватного спілкування між користувачами за допомогою особистих повідомлень, що дозволяє вирішувати питання конфіденційного характеру або продовжувати обговорення поза загальним доступом.

Одним з ключових факторів успіху веб-форуму є його продуктивність та швидкість роботи. Користувачі очікують миттєвого відгуку на свої дії, тому важливо оптимізувати код та налаштування сервера для швидкої обробки запитів. Використання кешування, оптимізація запитів до бази даних та зменшення обсягу передаваних даних є критичними для забезпечення високої продуктивності. Також важливо проводити регулярний моніторинг системи для виявлення та усунення потенційних проблем до того, як вони вплинуть на користувачів.

Масштабованість та гнучкість системи є важливими аспектами, оскільки зростання кількості користувачів та контенту може значно збільшити навантаження на систему. Необхідно забезпечити можливість горизонтального масштабування, додавання нових серверів та баз даних для обробки зростаючого обсягу даних. Використання контейнеризації та оркестрації дозволяє легко керувати ресурсами та масштабувати систему за потреби. Крім того, важливо передбачити можливість інтеграції з іншими сервісами та додатками для розширення функціональності форуму.

Веб-форум повинен бути доступним для користувачів з різними потребами, включаючи людей з обмеженими можливостями. Це означає дотримання стандартів доступності, таких як WCAG (Web Content Accessibility Guidelines), та забезпечення можливості використання форуму за допомогою різних засобів доступу, включаючи клавіатуру, екранні читачі та інші допоміжні технології. Крім того, важливо забезпечити підтримку кількох мов для користувачів з різних країн та культурних середовищ.

Регулярне вдосконалення та оновлення системи є необхідністю у швидкозмінному світі технологій та потреб користувачів. Веб-форум повинен регулярно оновлювати програмне забезпечення, впроваджувати нові функції та вдосконалювати існуючі. Це допоможе підтримувати конкурентоспроможність форуму та задовольняти потреби користувачів. Регулярне тестування, збір зворотного зв'язку від користувачів та аналіз даних дозволяють вчасно виявляти та усувати недоліки, забезпечуючи постійний розвиток системи.

Нарешті, забезпечення високого рівня обслуговування та підтримки користувачів є важливим аспектом роботи веб-форуму. Користувачі повинні мати можливість отримувати оперативну допомогу з технічних питань, проблем з реєстрацією, доступом до контенту та інших питань. Для цього необхідно організувати службу підтримки, яка буде відповідати на запити користувачів через різні канали, включаючи електронну пошту, чат або форум допомоги. Важливо забезпечити швидку та ефективну підтримку, щоб користувачі відчували себе впевнено та задоволені використанням платформи.

Таким чином, основні принципи роботи веб-форуму мають бути орієнтовані на забезпечення зручності, безпеки, продуктивності та задоволення потреб користувачів. Ретельне планування, використання сучасних технологій та постійне вдосконалення системи допоможуть створити ефективну та популярну платформу для обміну інформацією та спілкування. Веб-форум, що дотримується цих принципів, зможе залучити широку аудиторію, підтримувати активність користувачів та забезпечувати високу якість обслуговування.

2 ОСНОВНІ ТЕОРЕТИЧНІ ВІДОМОСТІ

2.1 Аналіз процесів розробки та створення веб-сайтів

Процес розробки та створення веб-сайтів є складним і багатогранним, що включає в себе декілька ключових етапів. Кожен з цих етапів має свої особливості, методології та інструменти, що використовуються для досягнення кінцевої мети - створення функціонального, зручного та естетично привабливого веб-ресурсу. В даному розділі ми розглянемо основні етапи процесу розробки веб-сайтів, а також проаналізуємо методи і підходи, що використовуються на кожному з цих етапів.

Першим етапом у розробці веб-сайту є планування та збір вимог. Цей етап є критичним для успіху проекту, оскільки саме на цьому етапі визначаються цілі, завдання та очікування від веб-сайту. Основними завданнями цього етапу є:

- Визначення мети веб-сайту: Для чого створюється сайт? Які задачі він повинен виконувати? Яка його основна аудиторія;
- Збір вимог: Інформація, яку повинен містити сайт, функціональні вимоги, такі як інтерактивність, форми зворотного зв'язку, можливості пошуку тощо;
- Аналіз конкурентів: Вивчення веб-сайтів конкурентів, їхніх переваг та недоліків, для того щоб створити більш конкурентоспроможний продукт;
- Планування ресурсу: Визначення бюджету, строків реалізації проекту, необхідних ресурсів та технологій, які будуть використовуватися.

На цьому етапі активно залучаються всі зацікавлені сторони проекту, включаючи замовника, розробників, дизайнерів та інших учасників команди. Результатом цього етапу є технічне завдання, яке стане основою для подальших робіт.

Наступним етапом є дизайн та прототипування. На цьому етапі розробляється візуальна концепція майбутнього сайту, створюються його макети та прототипи. Основні задачі цього етапу включають:

- Створення каркасів (wireframes): Візуальні схеми, що показують розташування основних елементів на сторінках сайту [6];
- Дизайн інтерфейсу користувача (UI): Розробка зовнішнього вигляду сайту, включаючи кольорову палітру, типографіку, іконографіку та інші візуальні елементи [7];
- Прототипування: Створення інтерактивних прототипів, що дозволяють оцінити зручність використання (UX) і функціональність сайту до його реалізації [7].

Дизайн та прототипування є важливими етапами, оскільки вони дозволяють виявити та виправити потенційні проблеми ще до початку розробки. Вони також забезпечують замовнику можливість бачити і взаємодіяти з попередньою версією сайту, що дозволяє внести корективи на ранніх стадіях проекту.

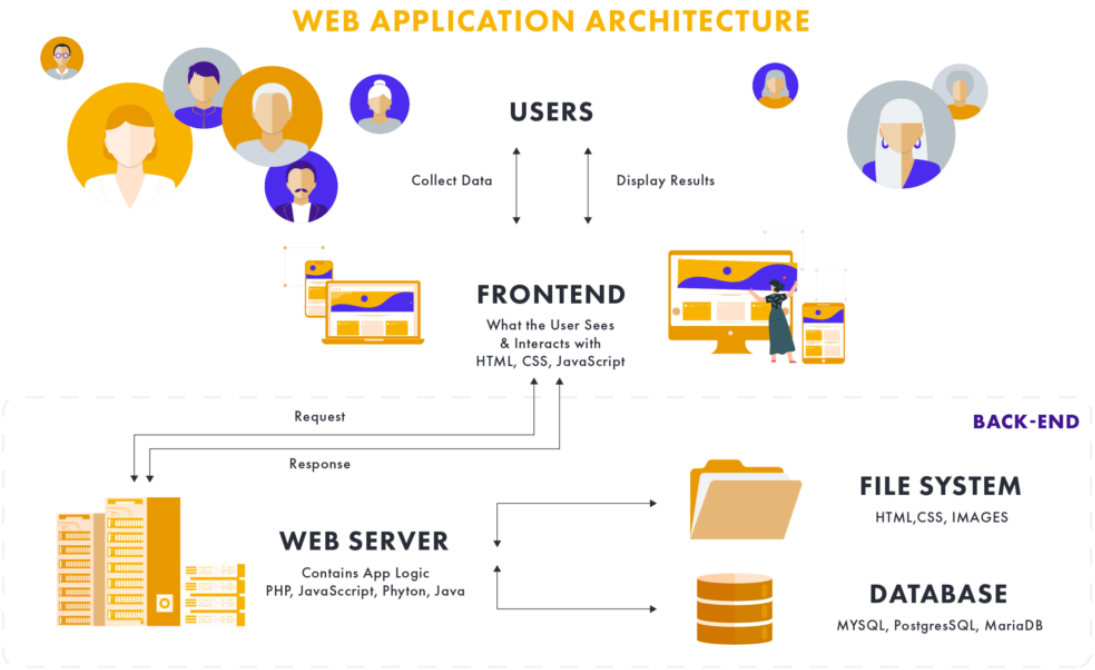

Рисунок 2.1 – Архітектура вебдодатків

Етап розробки включає в себе безпосереднє створення веб-сайту на основі затверджених прототипів та технічного завдання. Розробка ділиться на два основних аспекти: фронтенд та бекенд розробка.

- **Фронтенд розробка**: Включає створення користувацького інтерфейсу за допомогою HTML, CSS та JavaScript [8-9]. Фронтенд розробники відповідають за реалізацію дизайну, забезпечення інтерактивності та адаптивності сайту для різних пристроїв [10];
- **Бекенд розробка**: Включає розробку серверної частини сайту, де обробляються дані, взаємодія з базами даних, а також забезпечення безпеки та масштабованості [11]. Для бекенд розробки використовуються такі технології як Node.js, Python, Ruby on Rails та інші.

У процесі розробки важливу роль відіграє вибір технологій та інструментів, які будуть використовуватися. Наприклад, для фронтенд розробки можуть використовуватися фреймворки та бібліотеки, такі як React, Angular чи Vue.js, що значно спрощують процес розробки та підтримки сайту.

Після завершення етапу розробки сайт переходить до етапу тестування. Тестування є необхідним для виявлення та виправлення помилок, забезпечення коректної роботи всіх функцій та відповідності сайту вимогам [12]. Тестування може включати:

- Функціональне тестування: Перевірка роботи всіх функціональних елементів сайту, таких як форми, кнопки, посилання тощо;
- Тестування сумісності: Перевірка коректності відображення та роботи сайту на різних браузерах та пристроях;
- Тестування продуктивності: Оцінка швидкості завантаження сайту та його продуктивності при високому навантаженні;
- Безпекове тестування: Виявлення вразливостей та забезпечення захисту даних користувачів.

Тестування дозволяє забезпечити високий рівень якості продукту та мінімізувати ризики виникнення проблем після його запуску.Після успішного тестування сайт готовий до розгортання на продуктивному сервері. Розгортання включає:

- Налаштування серверного середовища: Встановлення та налаштування веб-сервера, баз даних, середовища виконання та інших необхідних компонентів;

- Міграція даних: Перенесення даних з тестового середовища на продуктивний сервер;
- Налаштування доменного імені та сертифікатів SSL: Забезпечення доступу до сайту через веб-адресу та забезпечення захищеного з'єднання.

Після розгортання сайт переходить в етап підтримки, який включає моніторинг його роботи, виправлення виникаючих помилок, оновлення контенту та впровадження нових функцій за потреби. Підтримка є важливим аспектом для забезпечення стабільної та безперебійної роботи веб-сайту.

Процес розробки та створення веб-сайтів є багатостадійним і включає планування, дизайн, розробку, тестування та розгортання. Кожен з цих етапів має свої особливості та вимагає використання відповідних методологій та інструментів. Важливою складовою успіху є чітке розуміння вимог та цілей проекту, активна комунікація між усіма учасниками команди, а також постійний контроль якості на всіх етапах. Завдяки правильному підходу та злагодженій роботі команди можна створити веб-сайт, що відповідає всім вимогам та очікуванням замовника.

2.2 Інструменти та методи прототипування сайтів

Прототипування сайтів є важливим етапом у процесі розробки веб-проектів. Воно дозволяє створити попередні версії веб-сторінок, які демонструють структуру, функціональність та дизайн майбутнього сайту. Використання прототипів допомагає розробникам, дизайнерам і замовникам узгодити бачення кінцевого продукту ще до початку його реалізації. У цьому розділі ми розглянемо основні методи та інструменти, що використовуються для прототипування веб-сайтів.

Прототипи можуть бути різних видів, залежно від рівня деталізації та мети їх створення. Одним з таких видів є каркаси (Wireframes), які представляють собою прості схематичні зображення, що показують розташування основних елементів на сторінці без деталізації дизайну. Вони використовуються для

визначення структури та функціональності сайту. Інший вид - макети (Mockups), які є більш деталізованими зображеннями, що включають елементи дизайну, такі як кольори, шрифти та графіку. Макети використовуються для візуалізації зовнішнього вигляду сайту. Також існують інтерактивні прототипи, що дозволяють користувачам взаємодіяти з елементами інтерфейсу, переходити між сторінками та виконувати основні дії. Вони використовуються для тестування користувацького досвіду (UX) і перевірки функціональності сайту.

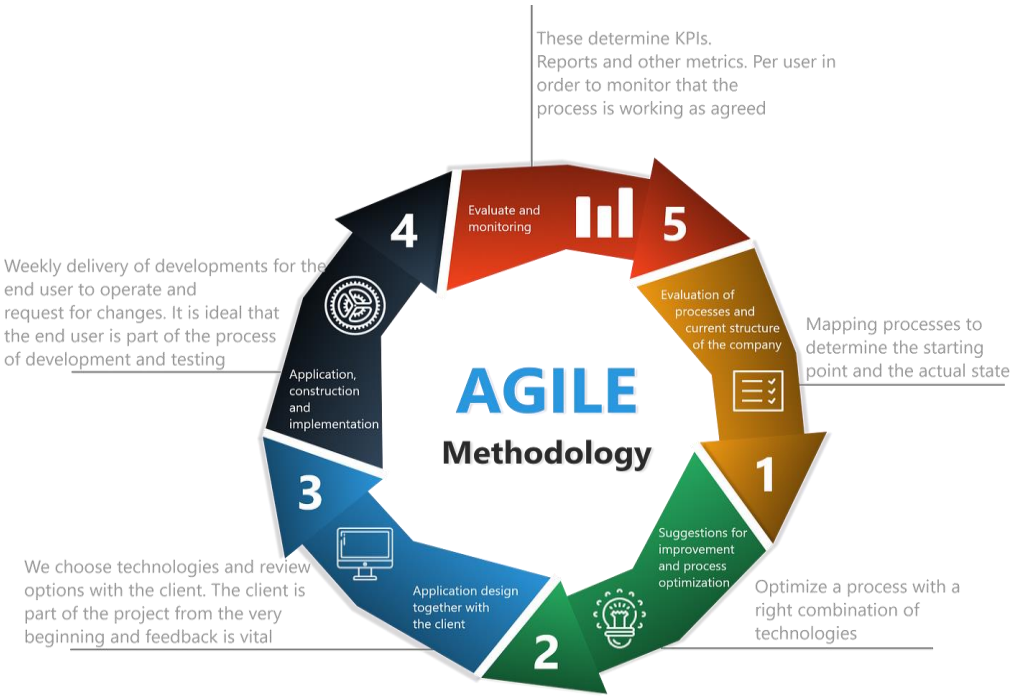

Рисунок 2.2 – Методологія Agile

Існує кілька методологій, які використовуються для прототипування сайтів. Найбільш поширені з них включають Agile, яка передбачає ітеративний підхід до розробки, де прототипи створюються, тестуються та вдосконалюються в рамках коротких ітерацій (спринтів). Це дозволяє швидко адаптуватися до змін вимог і отримувати зворотний зв'язок від користувачів на ранніх етапах проекту. Інша методологія, Design Thinking, орієнтована на користувача і включає п'ять основних етапів: емпатія, визначення проблеми, генерація ідей, створення прототипу та тестування. Прототипування є центральним етапом, де ідеї втілюються в реальні рішення для перевірки їх життєздатності. Lean UX

акцентує увагу на мінімізації відходів і максимізації цінності для користувача. Прототипи створюються швидко і з мінімальними витратами, а потім тестуються з реальними користувачами для отримання зворотного зв'язку та подальшого вдосконалення.

На ринку існує безліч інструментів для прототипування, кожен з яких має свої переваги та особливості. Figma є хмарним інструментом для дизайну та прототипування, що дозволяє командам співпрацювати в режимі реального часу. Вона підтримує створення каркасів, макетів та інтерактивних прототипів. Її основні переваги включають можливість спільної роботи, потужні інструменти для дизайну та інтеграція з іншими сервісами. Sketch є популярним інструментом для дизайну та прототипування, що використовується переважно на macOS. Sketch має зручний інтерфейс та багатий набір функцій для створення макетів та інтерактивних прототипів. Інструмент також підтримує численні плагіни, що розширюють його можливості. Adobe XD, інструмент від Adobe, забезпечує повний цикл прототипування від каркасів до інтерактивних прототипів. Він інтегрується з іншими продуктами Adobe, такими як Photoshop та Illustrator, що робить його зручним для дизайнерів, які вже використовують екосистему Adobe. InVision є платформою для створення інтерактивних прототипів та спільної роботи над дизайном. Вона дозволяє легко створювати інтерактивні прототипи, коментувати їх і отримувати зворотний зв'язок від команди та клієнтів. Платформа також підтримує інтеграцію з іншими інструментами для дизайну, такими як Sketch та Photoshop. Axure RP є потужним інструментом для створення детальних інтерактивних прототипів з високим рівнем функціональності. Axure RP дозволяє створювати складні інтерфейси з динамічними елементами та логікою, що робить його ідеальним для прототипування складних веб-додатків.

Процес прототипування складається з кількох ключових етапів, які дозволяють створити якісний та функціональний прототип веб-сайту. Спочатку визначаються основні вимоги до сайту, цільова аудиторія, функціональні можливості та бажаний користувацький досвід. На цьому етапі важливо залучити всіх зацікавлених сторін для узгодження бачення проекту. На основі

зібраних вимог створюються прості схематичні зображення сторінок сайту, що показують розташування основних елементів. Каркаси допомагають визначити структуру сайту та його навігацію. Після затвердження каркасів створюються більш деталізовані макети, що включають кольорову схему, шрифти, іконки та інші візуальні елементи. Макети дозволяють оцінити зовнішній вигляд сайту. На основі макетів створюються інтерактивні прототипи, що дозволяють користувачам взаємодіяти з елементами інтерфейсу. Інтерактивні прототипи допомагають перевірити функціональність сайту та виявити можливі проблеми з користувацьким досвідом. Інтерактивні прототипи тестуються з реальними користувачами або командою розробників для отримання зворотного зв'язку. Це дозволяє виявити проблеми та внести необхідні корективи. На основі отриманого зворотного зв'язку прототип удосконалюється, виправляються виявлені проблеми та додаються нові функції. Цей етап може повторюватися кілька разів, поки прототип не досягне бажаного рівня якості.

Використання прототипів у процесі розробки веб-сайтів має кілька ключових переваг. Прототипи допомагають узгодити бачення проекту між розробниками, дизайнерами та замовниками, що зменшує ризик непорозумінь. Вони дозволяють виявити та виправити проблеми на ранніх стадіях проекту, що знижує витрати на їх усунення на пізніших етапах. Інтерактивні прототипи дозволяють тестувати користувацький досвід і отримувати зворотний зв'язок від реальних користувачів, що допомагає створити зручний та інтуїтивно зрозумілий інтерфейс. Прототипи дозволяють швидко вносити зміни та тестувати нові ідеї, що зменшує витрати на розробку та підвищує ефективність роботи команди.

Прототипування сайтів є важливим етапом у процесі їх розробки, що дозволяє створити якісний та функціональний продукт. Використання різних методів і інструментів для прототипування допомагає узгодити бачення проекту, виявити та виправити проблеми на ранніх стадіях, покращити користувацький досвід і зекономити час та ресурси. Завдяки правильному підходу до прототипування можна забезпечити успіх веб-проекту та задовольнити вимоги та очікування замовника.

2.3 Технологічні стеки для веб-розробки

У сучасному світі веброзробка є складним і багатогранним процесом, який потребує використання різних технологій та інструментів. Технологічний стек (stack) – це сукупність технологій, фреймворків та бібліотек, які використовуються для розробки веб-додатків. Правильний вибір технологічного стеку є ключовим для успішної реалізації проекту, оскільки він визначає продуктивність, масштабованість, безпеку та зручність обслуговування веб-додатку. У цьому розділі ми розглянемо основні технологічні стеки для веб-розробки та їхні особливості.

LAMP стек є одним з найстаріших та найпоширеніших технологічних стеків для веб-розробки [13]. Він включає чотири основні компоненти: Linux, Apache, MySQL та PHP.

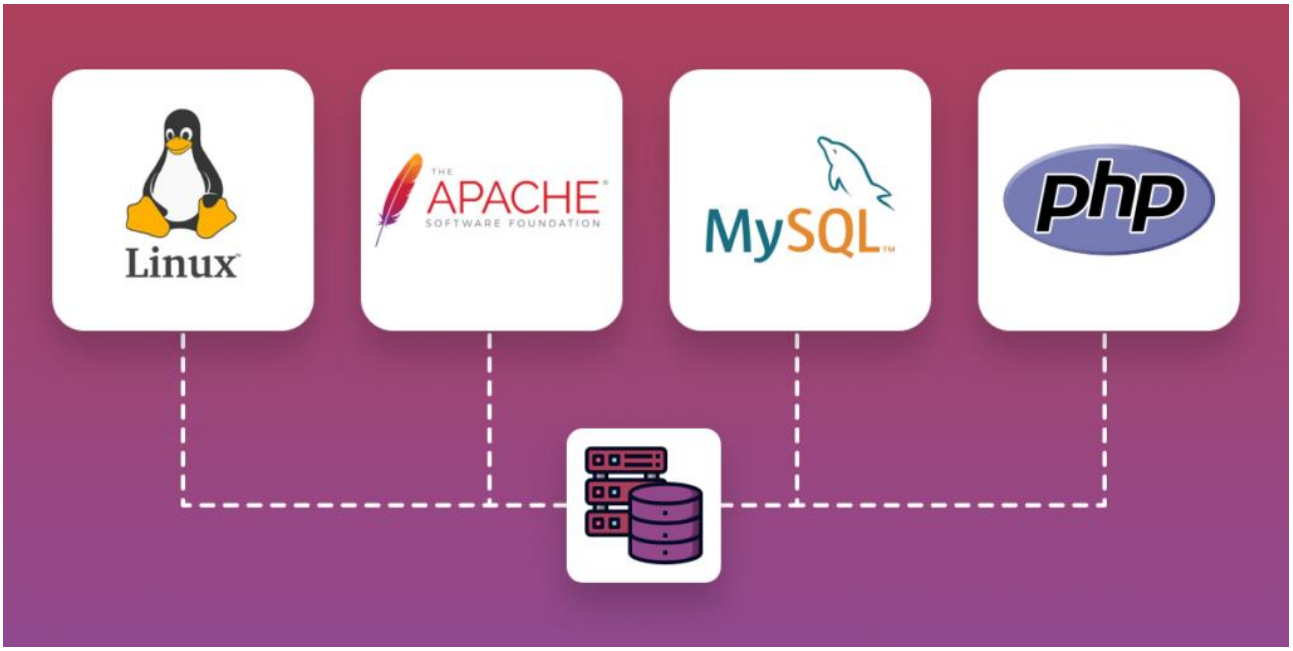

Рисунок 2.3 – Технологічних стек LAMP

Linux забезпечує стабільне та безпечне середовище для роботи веб-серверів. Apache – це веб-сервер, який обробляє запити користувачів і доставляє їм веб-сторінки. MySQL є системою керування базами даних, яка використовується для зберігання та управління даними веб-додатків. PHP – це

мова програмування, яка використовується для створення динамічних веб-сторінок та серверної логіки. LAMP стек є популярним завдяки своїй відкритості, надійності та широкій підтримці спільноти. Він підходить для розробки різноманітних веб-додатків, від простих блогів до складних корпоративних систем [13].

MEAN стек є сучасним підходом до веб-розробки, який включає MongoDB, Express.js, Angular та Node.js. MongoDB – це NoSQL база даних, яка використовує документи у форматі JSON для зберігання даних. Вона забезпечує високу продуктивність та масштабованість. Express.js є легким та гнучким веб-фреймворком для Node.js, який спрощує створення серверних додатків та API.

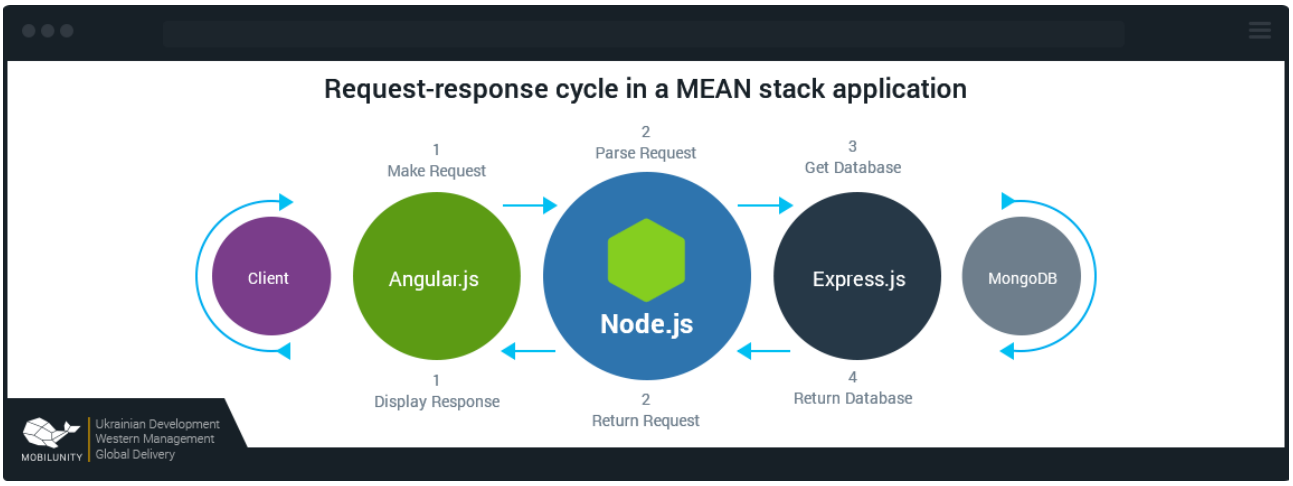

Рисунок 2.4 – Технологічних стек MEAN

Angular – це фронтенд фреймворк, розроблений Google, який використовується для створення динамічних та інтерактивних веб-додатків. Node.js є середовищем виконання для JavaScript, яке дозволяє запускати серверний код на стороні сервера, забезпечуючи високу продуктивність та ефективність обробки запитів. MEAN стек є популярним серед розробників завдяки своїй єдиній мові програмування (JavaScript) для всієї архітектури додатку, що спрощує процес розробки та обслуговування [1a].

MERN стек є варіацією MEAN стеку, де замість Angular використовується React. MongoDB використовується для зберігання даних, Express.js для

створення серверних додатків та API, React для створення користувацьких інтерфейсів, а Node.js – для виконання серверного коду [1].

React дозволяє створювати компонентно-орієнтовані інтерфейси з високою продуктивністю. MERN стек забезпечує високу продуктивність та гнучкість, що робить його популярним вибором для розробки сучасних веб-додатків.

JAMstack є підходом до веб-розробки, який акцентує увагу на використанні статичних сайтів та динамічного контенту через API. Основні компоненти JAMstack включають JavaScript для створення інтерактивних функцій на стороні клієнта, API для виконання серверних функцій та статичні HTML-файли, що генеруються на основі шаблонів або статичних сайтогенераторів (наприклад, Gatsby або Hugo). JAMstack забезпечує високу продуктивність, безпеку та простоту розгортання. Він є ідеальним вибором для розробки блогів, маркетингових сайтів та інших додатків з великим обсягом статичного контенту [15].

Ruby on Rails (RoR) є популярним фреймворком для розробки веб-додатків, який базується на мові програмування Ruby. Ruby відома своєю простотою та елегантністю. Rails спрощує розробку веб-додатків за допомогою набору конвенцій та інструментів. RoR дозволяє швидко створювати веб-додатки завдяки великій кількості вбудованих функцій та бібліотек. Він є популярним вибором для стартапів та невеликих проектів, де важлива швидкість розробки [16].

Django є фреймворком для веб-розробки на мові програмування Python. Python відомий своєю читабельністю та простотою. Django забезпечує повний набір інструментів для створення веб-додатків, включаючи адміністрування, автентифікацію, роботу з базами даних та багато іншого. Django є популярним вибором для розробки масштабованих та безпечних веб-додатків завдяки своїй потужності та розширюваності [17].

Вибір технологічного стеку залежить від кількох факторів, включаючи вимоги до проекту, наявні ресурси, досвід команди та майбутню підтримку

додатку. Основні критерії вибору включають продуктивність, масштабованість, безпеку, вартість розробки та підтримки, а також наявність активної спільноти. Важливо оцінити, як обраний стек буде впливати на швидкість та продуктивність додатку, чи підтримує він масштабування, щоб додаток міг рости разом з бізнесом, а також забезпечити захист даних від атак. Також слід враховувати вартість розробки, обслуговування та ліцензій на використання технологій. Обраний стек повинен мати активну спільноту та достатню підтримку для вирішення проблем та оновлення технологій.

Технологічний стек є важливим аспектом у розробці веб-додатків, оскільки він визначає продуктивність, масштабованість, безпеку та зручність обслуговування. Існує багато різних стеків, кожен з яких має свої переваги та особливості. Вибір відповідного стеку залежить від конкретних вимог проекту, ресурсів та досвіду команди. Важливо ретельно оцінити всі фактори перед прийняттям рішення, щоб забезпечити успішну реалізацію проекту та задовольнити вимоги користувачів.

2.4 Стек MERN

MERN стек технологій є одним із найбільш популярних і ефективних рішень для розробки сучасних веб-додатків [18]. Цей стек включає чотири основні компоненти: MongoDB, Express.js, React та Node.js. Кожен з цих компонентів має свої особливості та переваги, що робить їх спільне використання потужним інструментом для створення інтерактивних та масштабованих веб-додатків. У цьому розділі ми детально розглянемо кожен з компонентів MERN стеку та їх взаємодію в контексті веб-розробки [19].

MongoDB є NoSQL базою даних, яка зберігає дані у форматі JSON-подібних документів. Це відмінний вибір для веб-додатків завдяки своїй гнучкості, масштабованості та продуктивності [20]. Основні особливості MongoDB включають:

- Гнучкість схеми: MongoDB не потребує попереднього визначення схеми

бази даних, що дозволяє зберігати документи з різними структурами в одній колекції. Це особливо корисно для додатків, що часто змінюються або мають різнорідні дані;

- Масштабованість: MongoDB підтримує горизонтальне масштабування шляхом шардінгу, що дозволяє розподіляти дані по кількох серверах та забезпечувати високу продуктивність при великих обсягах даних;
- Продуктивність: Завдяки індексації, агрегаційним фреймворкам та можливостям для кешування, MongoDB забезпечує високу швидкість виконання запитів та обробки даних;
- Інтеграція з JavaScript: Оскільки документи MongoDB мають формат JSON, їх легко використовувати з JavaScript, що робить MongoDB ідеальним для MERN стеку.

Express.js є легким та гнучким веб-фреймворком для Node.js, який спрощує створення серверних додатків та API. Він забезпечує набір потужних функцій для розробки веб-додатків та API [21]. Основні особливості Express.js включають:

- Простота використання: Express.js має мінімалістичний інтерфейс, що робить його легким у вивченні та використанні. Це дозволяє розробникам швидко створювати серверні додатки.
- Модульність: Завдяки підтримці мідлварів (middleware), Express.js дозволяє легко розширювати функціональність додатків за допомогою підключення зовнішніх бібліотек та модулів.
- Гнучкість: Express.js не нав'язує жорстких структур чи шаблонів, що дає розробникам свободу у виборі архітектури та підходів до розробки.
- Підтримка RESTful API: Express.js є ідеальним для створення RESTful API, що дозволяє легко інтегрувати серверну логіку з фронтенд додатками на React.

React є бібліотекою для створення користувацьких інтерфейсів, розробленою Facebook. Вона дозволяє створювати компонентно-орієнтовані інтерфейси з високою продуктивністю [22]. Основні особливості React

включають:

- Компонентний підхід: React дозволяє розробникам створювати інтерфейси з окремих, багаторазових компонентів. Це сприяє модульності та полегшує підтримку коду.
- Віртуальний DOM: React використовує віртуальний DOM для оптимізації оновлень інтерфейсу, що забезпечує високу продуктивність додатків навіть при великих обсягах даних та складних інтерфейсах.
- Одностороннє зв'язування даних: React реалізує одностороннє зв'язування даних, що спрощує розуміння потоку даних в додатку та зменшує ймовірність помилок.
- Екосистема: React має велику та активну спільноту, що забезпечує доступ до безлічі бібліотек, інструментів та ресурсів для розробки.

Node.js є середовищем виконання для JavaScript на стороні сервера, що базується на рушії V8 від Google. Воно дозволяє запускати серверний код на стороні сервера, забезпечуючи високу продуктивність та ефективність обробки запитів [23]. Основні особливості Node.js включають:

- Подієво-орієнтована архітектура: Node.js використовує неблокуючу модель введення-виведення, що дозволяє ефективно обробляти великий обсяг одночасних запитів.
- Єдина мова програмування: Використання JavaScript як на стороні сервера, так і на стороні клієнта, спрощує розробку та обслуговування додатків.
- Широкий набір модулів: Node.js має великий репозиторій модулів NPM (Node Package Manager), що дозволяє легко підключати та використовувати сторонні бібліотеки та інструменти.
- Масштабованість: Node.js підтримує горизонтальне масштабування через кластеризацію, що дозволяє ефективно розподіляти навантаження на кілька процесів.

Взаємодія компонентів MERN стеку дозволяє створювати потужні та масштабовані веб-додатки. Кожен з компонентів виконує свою роль, забезпечуючи ефективність та гнучкість розробки.

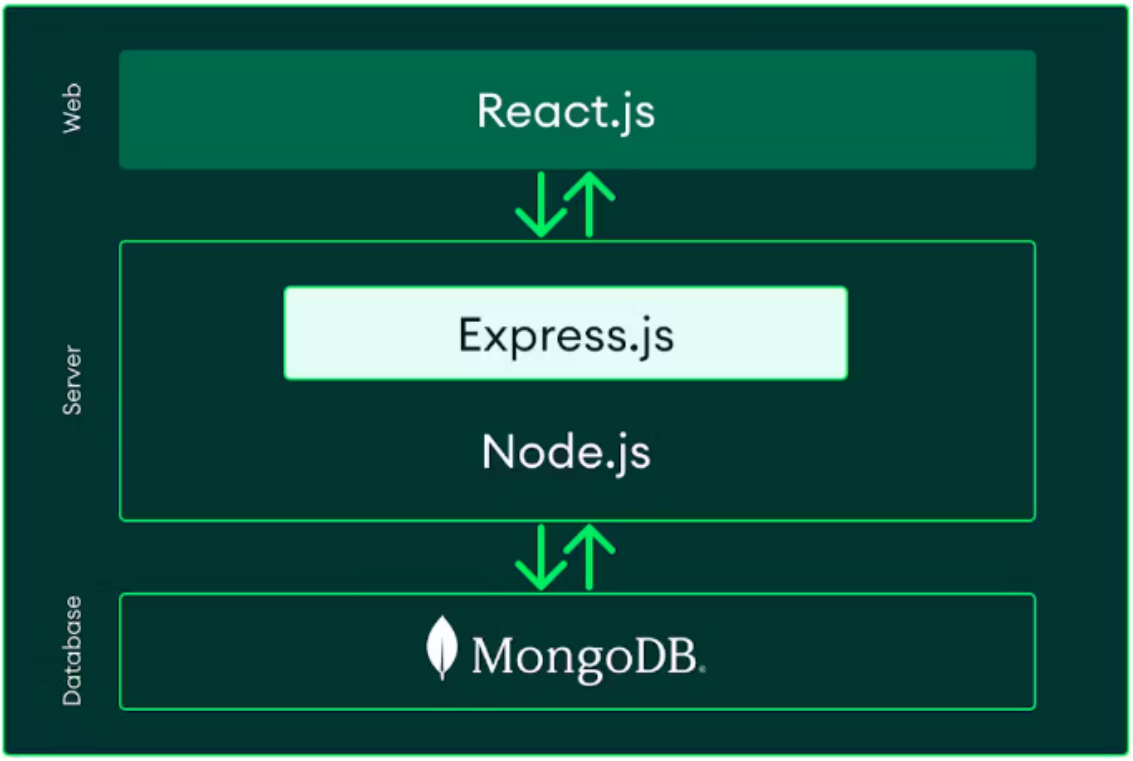

Рисунок 2.5 – Клієнт серверна архітектура за допомогою MEAN

MongoDB використовується для зберігання даних у вигляді документів. Ці дані можуть бути легко отримані та оброблені на сервері за допомогою Express.js. Express.js надає API для обробки запитів від клієнтів та взаємодії з базою даних. Node.js забезпечує виконання серверного коду та обробку запитів завдяки своїй подієво-орієнтованій архітектурі. На стороні клієнта React використовується для створення інтерфейсу користувача. Компоненти React взаємодіють з API, створеними на Express.js, для отримання та відображення даних з MongoDB.

Використання MERN стеку має кілька ключових переваг. По-перше, єдина мова програмування (JavaScript) використовується для всіх компонентів, що спрощує процес розробки та обслуговування додатків. По-друге, кожен з компонентів стеку є потужним інструментом сам по собі, а їх поєднання забезпечує високу продуктивність, гнучкість та масштабованість. По-третє, велика та активна спільнота розробників MERN забезпечує доступ до численних ресурсів, бібліотек та інструментів, що полегшує розробку та підтримку додатків.

MERN стек використовується для розробки різноманітних веб-додатків, включаючи соціальні мережі, платформи електронної комерції, блоги, системи управління контентом та багато інших. Завдяки своїй гнучкості та потужності, MERN стек дозволяє створювати як прості односторінкові додатки, так і складні багатофункціональні системи.

MERN стек технологій є потужним та ефективним інструментом для розробки сучасних веб-додатків. Використання MongoDB, Express.js, React та Node.js забезпечує високу продуктивність, гнучкість та масштабованість додатків. Єдина мова програмування (JavaScript) спрощує процес розробки та обслуговування, а велика та активна спільнота розробників забезпечує доступ до численних ресурсів та інструментів. Завдяки правильному використанню MERN стеку можна створювати потужні, ефективні та масштабовані веб-додатки, які задовольняють вимоги сучасних користувачів.

3. ПРАКТИЧНА РЕАЛІЗАЦІЯ

3.1 Процес розробки веб-форуму

Процес розробки веб-форуму є багатоступеневим і включає різні етапи, кожен з яких має свої особливості та вимоги [24-25]. У цьому розділі опишемо кожен з етапів, демонструючи коди з поясненнями для кожного кроку.

Перший етап розробки включає підключення необхідних імпортів для роботи з Node.js та Express. Розпочинаємо з налаштування основних залежностей проекту. На цьому етапі важливо забезпечити підключення основних бібліотек, які використовуються для створення сервера та обробки HTTP-запитів . Express забезпечує основну функціональність сервера, mongoose підключається до бази даних MongoDB, bodyParser обробляє дані запитів, а cors дозволяє обробляти запити з інших доменів.

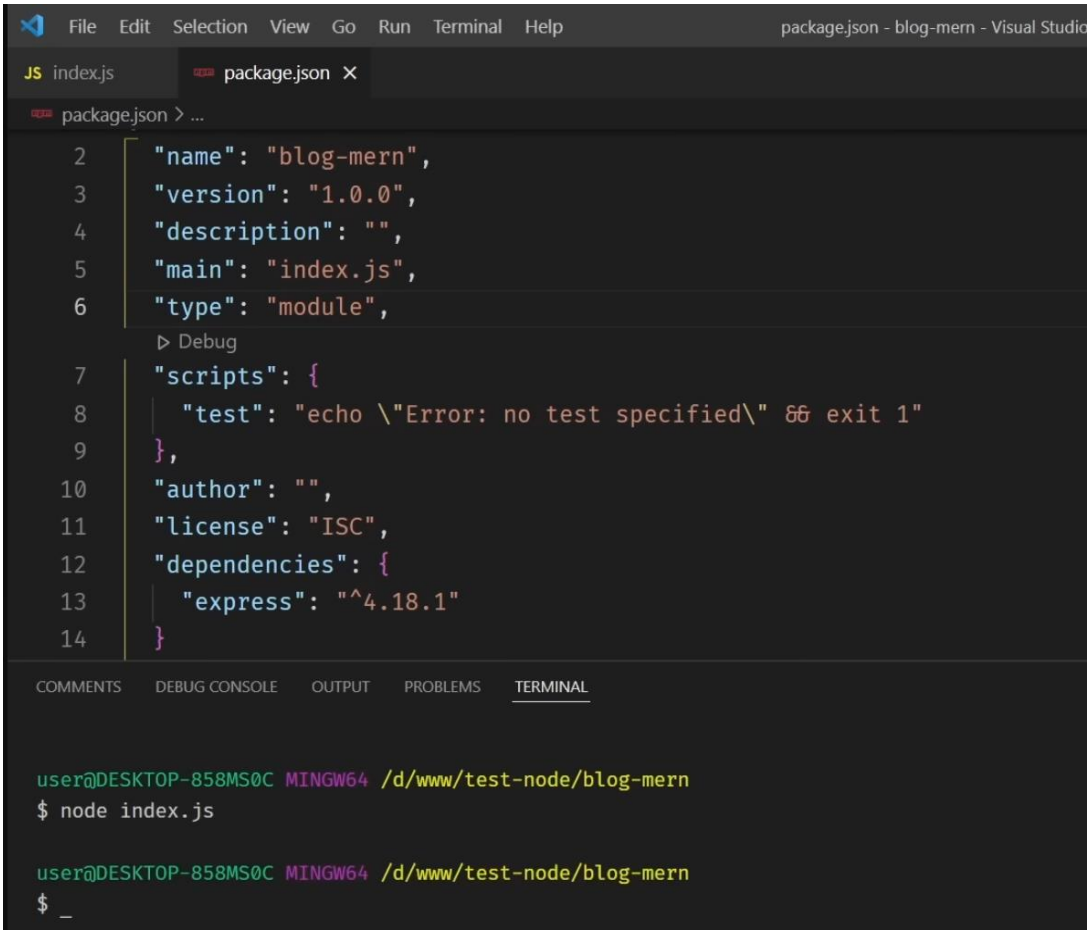

Рисунок 3.1 – Підключення імпортів

Node.js є середовищем виконання, що дозволяє запускати JavaScript-код на сервері. Express – це фреймворк для Node.js, який спрощує створення веб-додатків. MongoDB є нереляційною базою даних, яка використовує документи у форматі JSON. Mongoose – це бібліотека для Node.js, яка дозволяє працювати з MongoDB у стилі об'єктно-документного моделювання. BodyParser використовується для обробки JSON-запитів, а cors забезпечує крос-доменну взаємодію.

Наступний крок – це створення початкової програми на Express. Це дозволяє запустити сервер і встановити основні маршрути. Цей крок включає налаштування сервера для обробки запитів у форматі JSON та дозволяє крос-доменні запити. Крім того, створюється маршрут, який відповідає на запити до кореневого маршруту з простим повідомленням "Hello, World!".

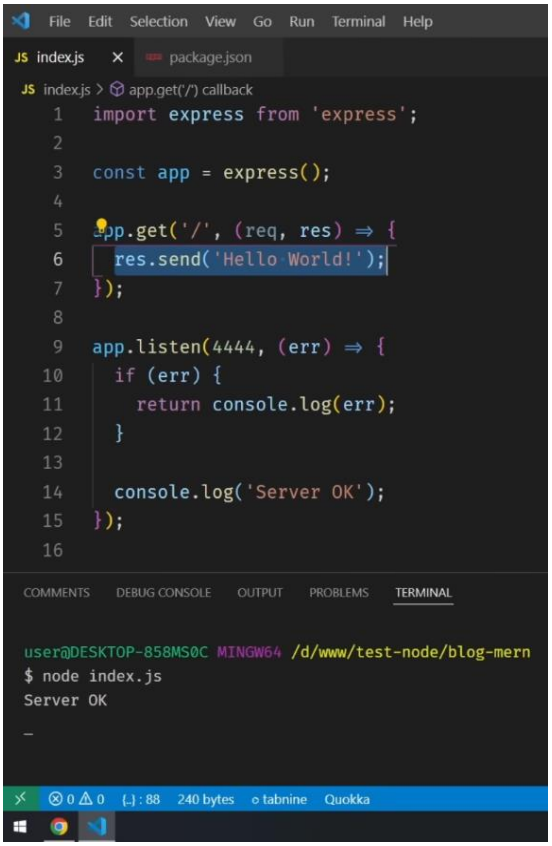

Рисунок 3.2 – Створення початкової програми на Express

Цей етап включає створення основного додатка на Express, який відповідає на HTTP-запити. Використовуючи метод app.get, визначаємо маршрут, який

відповідає на запити з кореневої URL. Запуск сервера здійснюється методом app.listen, що дозволяє серверу прослуховувати запити на вказаному порту.

Для забезпечення безпеки встановлюється пакет JSON Web Token (JWT). Це дозволяє створювати токени для автентифікації користувачів. JWT дозволяє створювати підписані токени, які можуть бути використані для підтвердження особи користувача під час запитів до сервера.

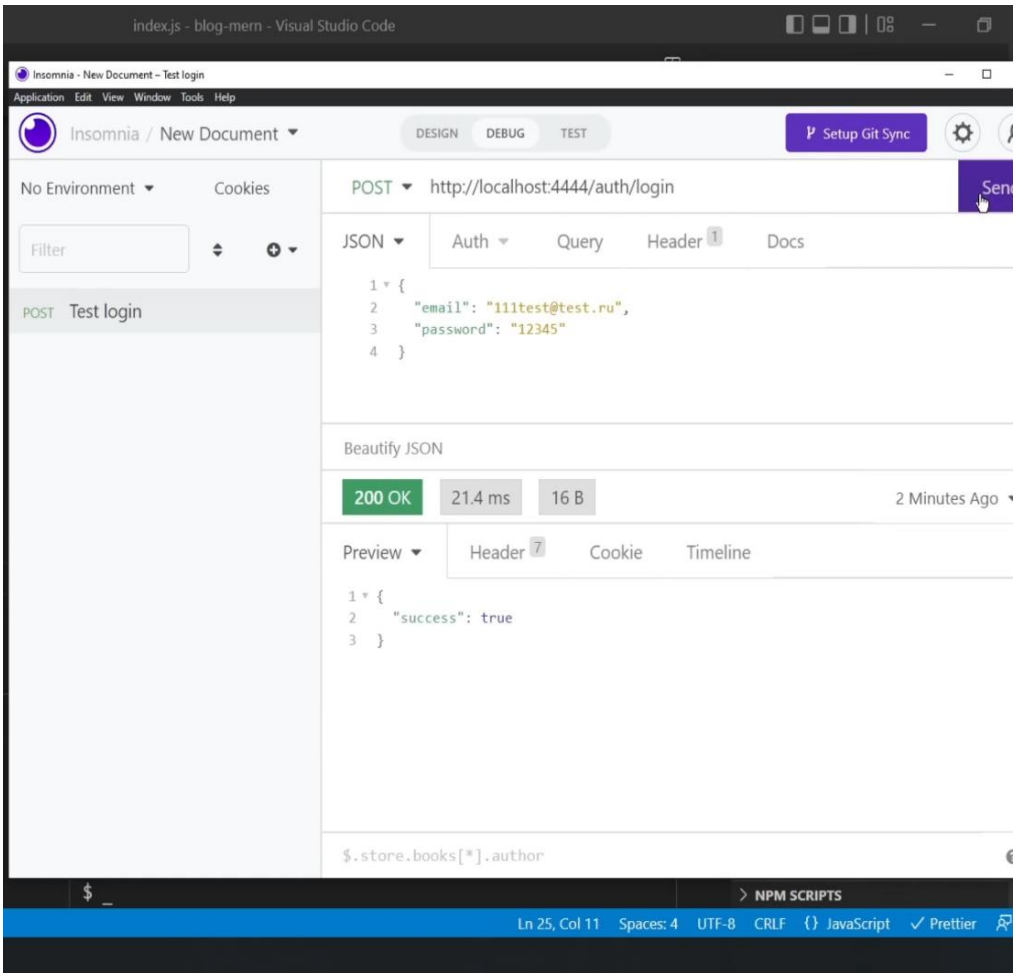

Рисунок 3.3 – Робота JWT

JSON Web Token (JWT) – це стандарт, який визначає компактний і самодостатній спосіб передачі інформації між сторонами як JSON-об'єкт. Ця інформація може бути перевірена та довірена завдяки цифровому підпису. JWT часто використовується для автентифікації та авторизації в веб-додатках [26].

Наступний етап – підключення до бази даних MongoDB. Для цього

використовується mongoose. Цей крок підключає проект до локальної бази даних MongoDB, використовуючи mongoose. Використання mongoose дозволяє працювати з базою даних за допомогою об'єктно-документної моделі, що значно спрощує взаємодію з даними.

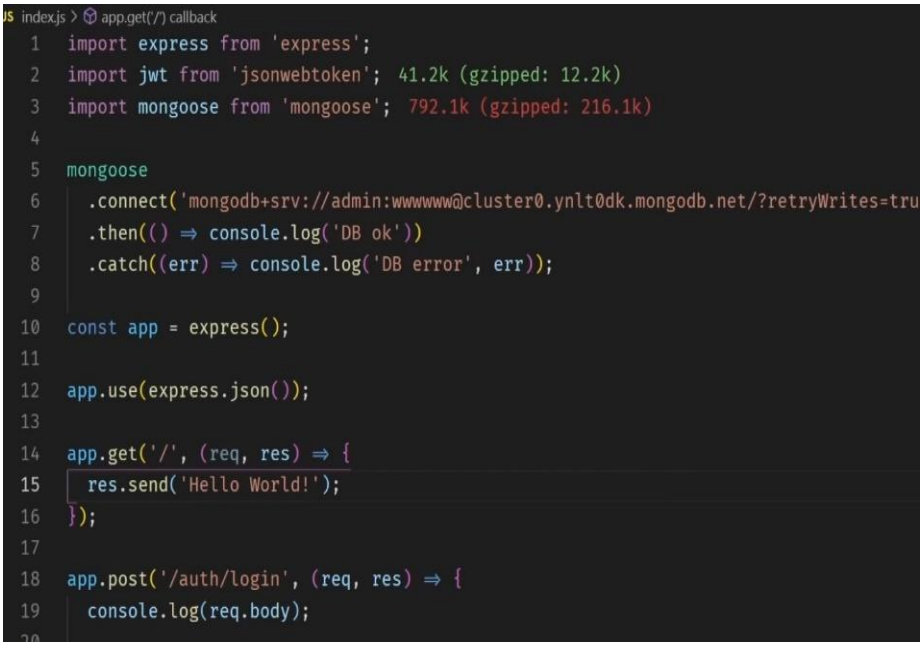

Рисунок 3.4 – Підключення до бази даних

Mongoose є бібліотекою для MongoDB, яка дозволяє легко взаємодіяти з базою даних, створювати схеми для документів, перевіряти дані перед збереженням і здійснювати складні запити до бази даних. Це спрощує процес роботи з MongoDB і дозволяє розробникам зосередитися на бізнес-логіці додатка [27].

Для реєстрації користувачів створена модель користувача. Це включає визначення схеми користувача та створення відповідної моделі. Ця схема визначає структуру документів у колекції користувачів, включаючи обов'язкові поля username, email та password. Використання схеми дозволяє забезпечити цілісність даних та уникнути помилок.

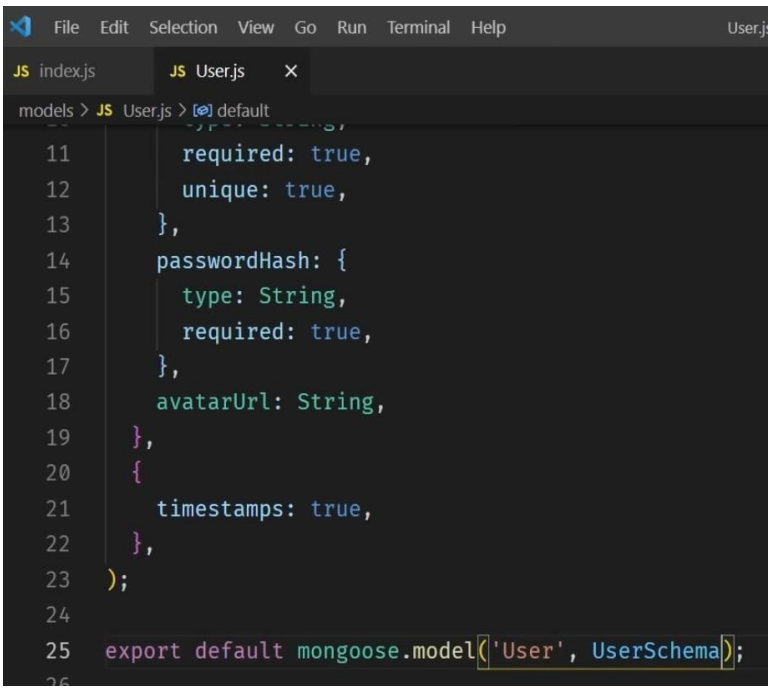

Рисунок 3.5 – Створення моделі користувача

Модель користувача визначає, як дані користувачів зберігатимуться у базі даних. Використання схем у mongoose дозволяє чітко визначити структуру даних та встановити правила валідації для кожного поля. Це допомагає запобігти помилкам та забезпечити консистентність даних.

Для початкової валідації даних використовується бібліотеку Joi. Це дозволяє перевіряти дані перед збереженням у базу даних. Цей крок створює схему валідації для реєстрації користувачів, перевіряючи, що username та password мають мінімальну довжину 6 символів, а email є валідною електронною адресою.

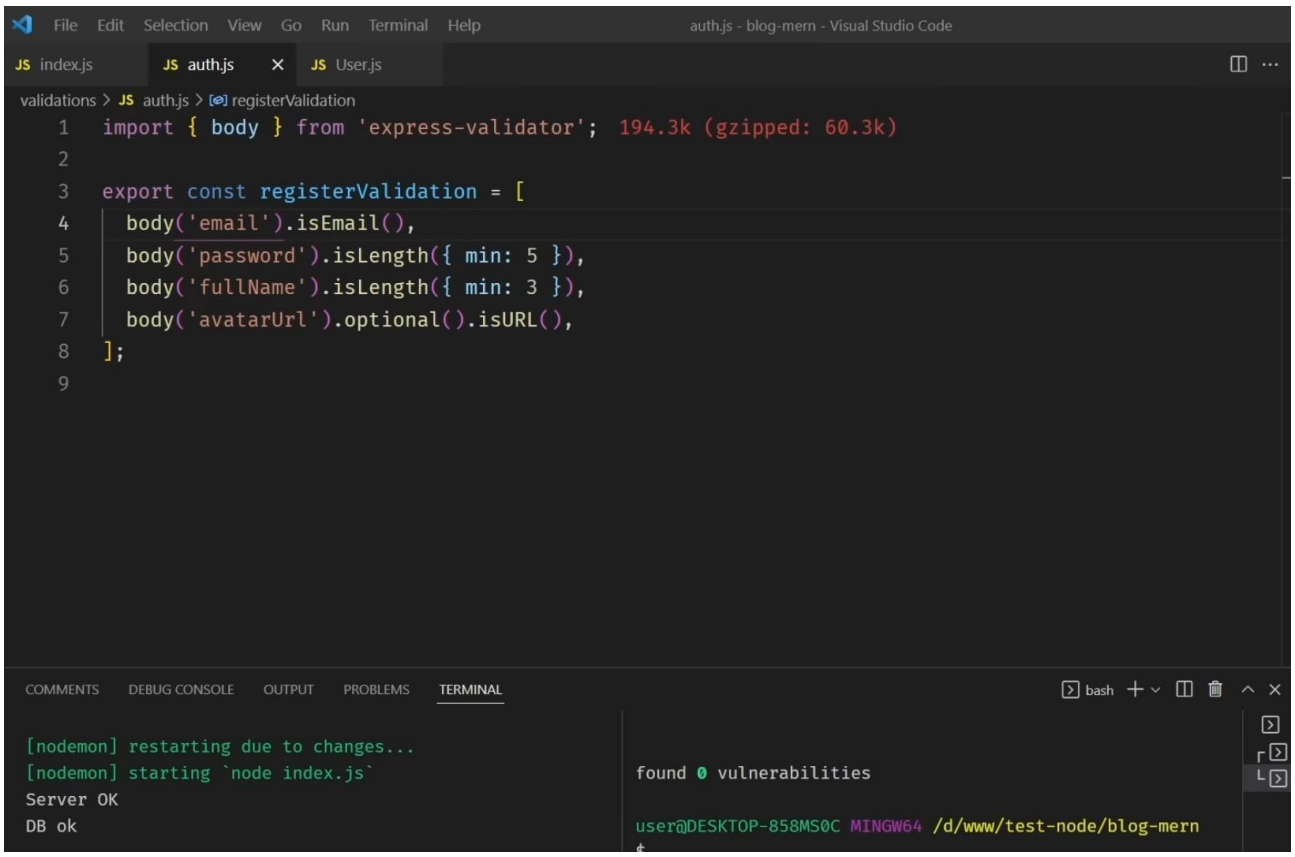

Рисунок 3.6 – Початкова валідація даних

Joi – це бібліотека для валідації даних JavaScript. Вона дозволяє створювати схеми валідації для об'єктів, масивів та інших типів даних, перевіряти їх на відповідність цим схемам і повертати детальні повідомлення про помилки [28]. Це допомагає забезпечити цілісність даних перед їх збереженням у базу даних.

Для шифрування паролів використовуємо Bcrypt. Це забезпечує безпеку збереження паролів у базі даних. Bcrypt використовує алгоритм хешування, що робить паролі практично неможливими для зламу. Використання Bcrypt додає додатковий рівень безпеки, ускладнюючи спроби зловмисників підібрати паролі.

Bcrypt – це бібліотека для хешування паролів, яка використовує алгоритм Blowfish для створення хешів. Вона автоматично додає хеш до пароля перед хешуванням, що робить хеші унікальними навіть для однакових паролів. Це забезпечує додатковий рівень безпеки при зберіганні паролів [29].

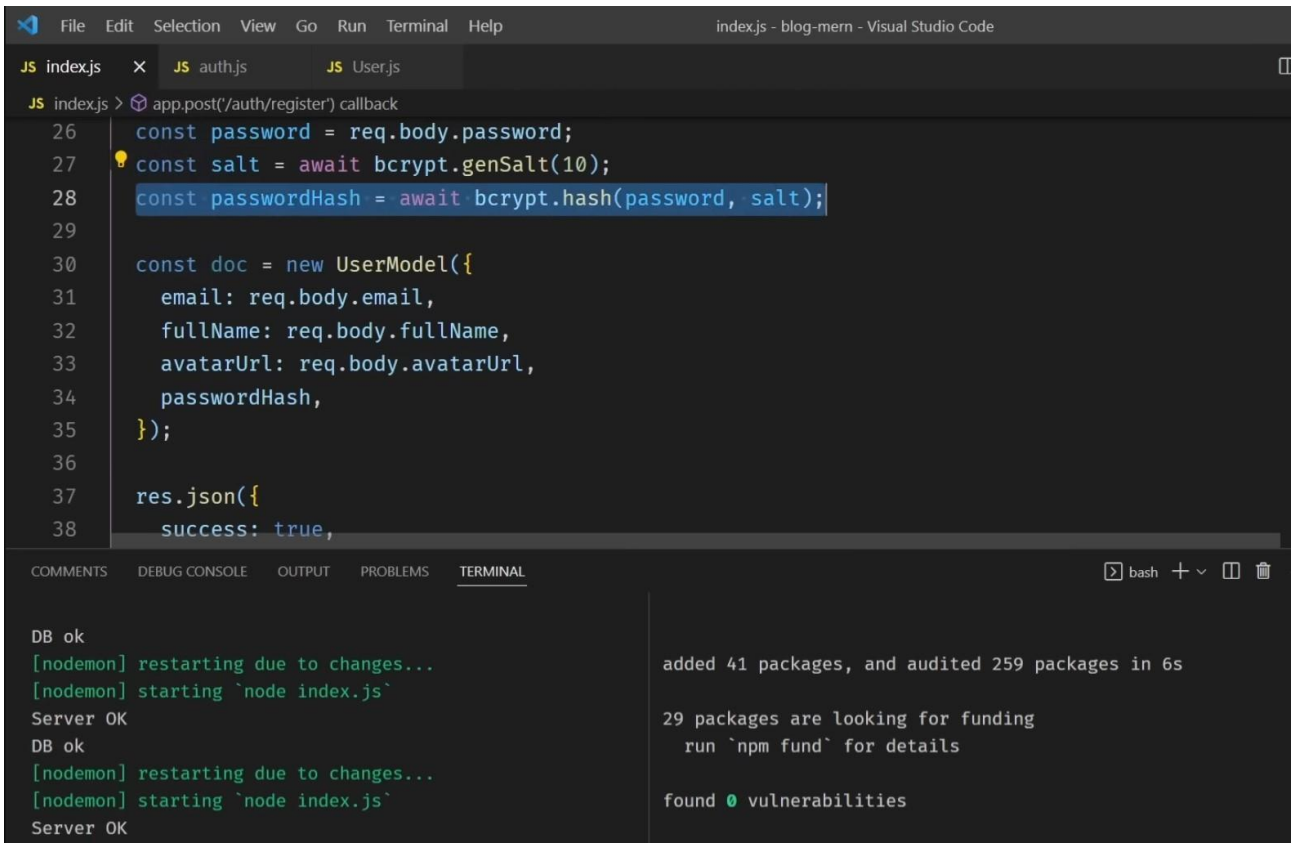

Рисунок 3.7 – Використання Bcrypt для шифрування паролів

Далі реалізовуємо маршрути для реєстрації користувачів, включаючи обробку помилок. Цей крок обробляє POST-запити для реєстрації нових користувачів. Він включає валідацію даних, перевірку наявності користувача з таким же email, хешування паролів та збереження нового користувача у базі даних.

Процес реєстрації користувача включає кілька етапів: спочатку дані користувача перевіряються на відповідність схемі валідації, потім перевіряється, чи існує вже користувач з таким email, якщо ні, то пароль хешується за допомогою Bcrypt і новий користувач зберігається у базі даних. Якщо виникає помилка на будь-якому етапі, сервер повертає відповідне повідомлення про помилку.

Після цього додаємо функціонал для створення нових постів на форумі. Цей крок створює схему посту та відповідний маршрут для створення нових постів. Кожен пост містить заголовок, вміст, автора та дату створення.

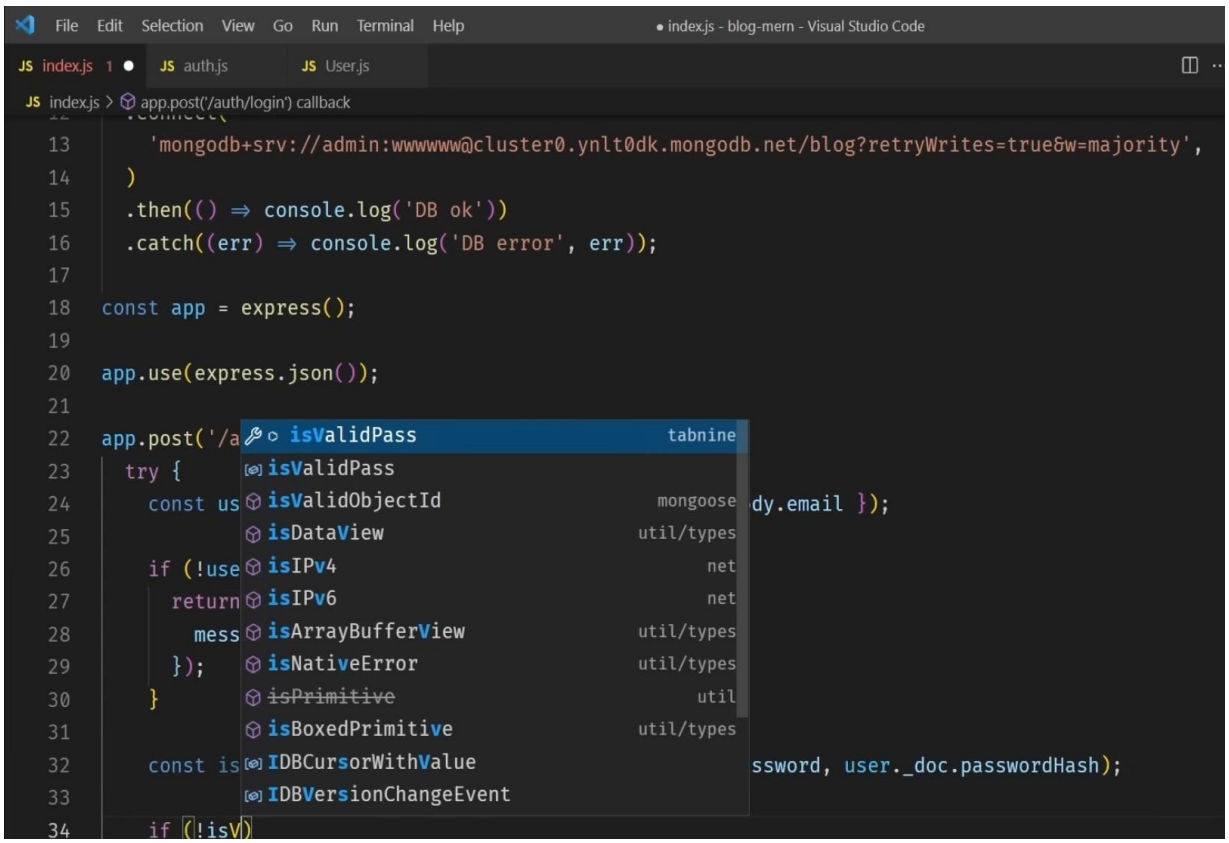

Рисунок 3.8 – Реєстрація користувача з обробкою помилок

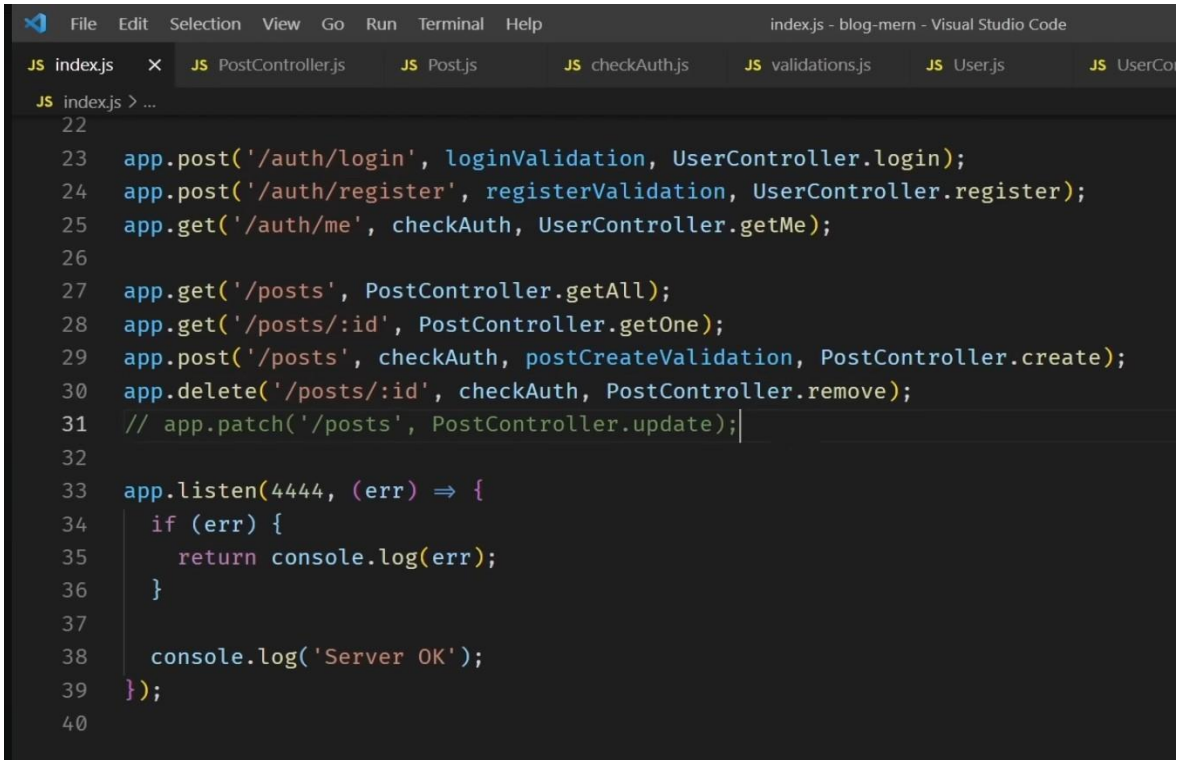

Рисунок 3.9 – Функціонал для додавання постів

Пости на форумі є основним видом контенту, який створюють користувачі. Схема посту визначає структуру даних для кожного посту, включаючи заголовок, вміст, автора та дату створення. Маршрут для створення постів обробляє запити від користувачів і зберігає нові пости у базі даних.

Для додавання зображень до постів використовуємо Multer. Цей крок налаштовує Multer для зберігання завантажених зображень у директорії "uploads" з унікальними іменами файлів [30]. Маршрут обробляє POST-запити для завантаження файлів.

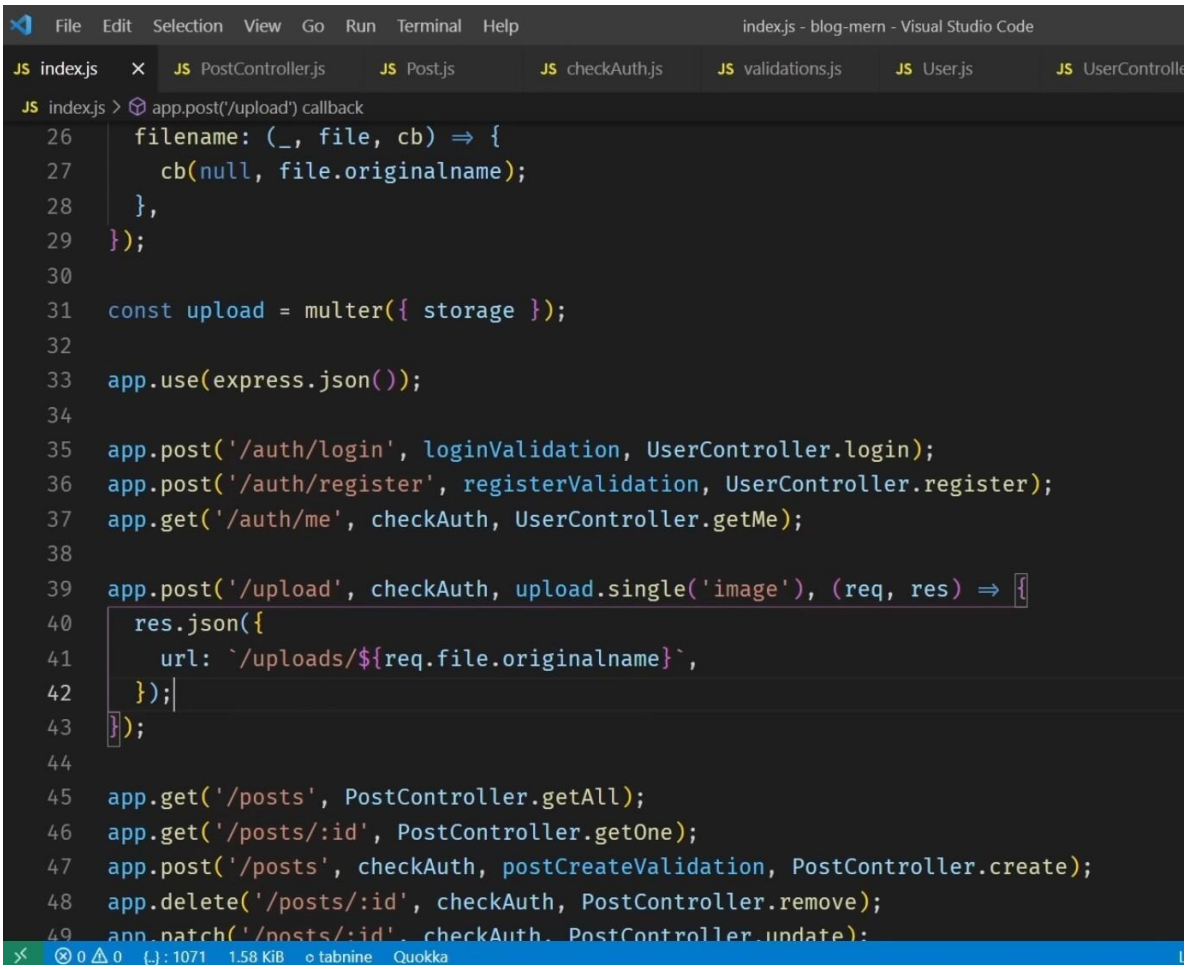

Рисунок 3.10 – Додавання функціоналу для завантаження зображень

Multer – це проміжне програмне забезпечення для Node.js, яке дозволяє обробляти файли, завантажені через форми. Воно дозволяє налаштувати місце зберігання файлів, імена файлів та інші параметри, що забезпечує гнучкість у

обробці завантажень. Додавання зображень до постів дозволяє користувачам ділитися візуальним контентом, що значно підвищує інтерактивність та привабливість форуму.

На фронтенді використовується React Router для управління маршрутизацією. Цей крок налаштовує маршрутизацію на фронтенді, дозволяючи перемикатися між різними сторінками додатку без перезавантаження сторінки.

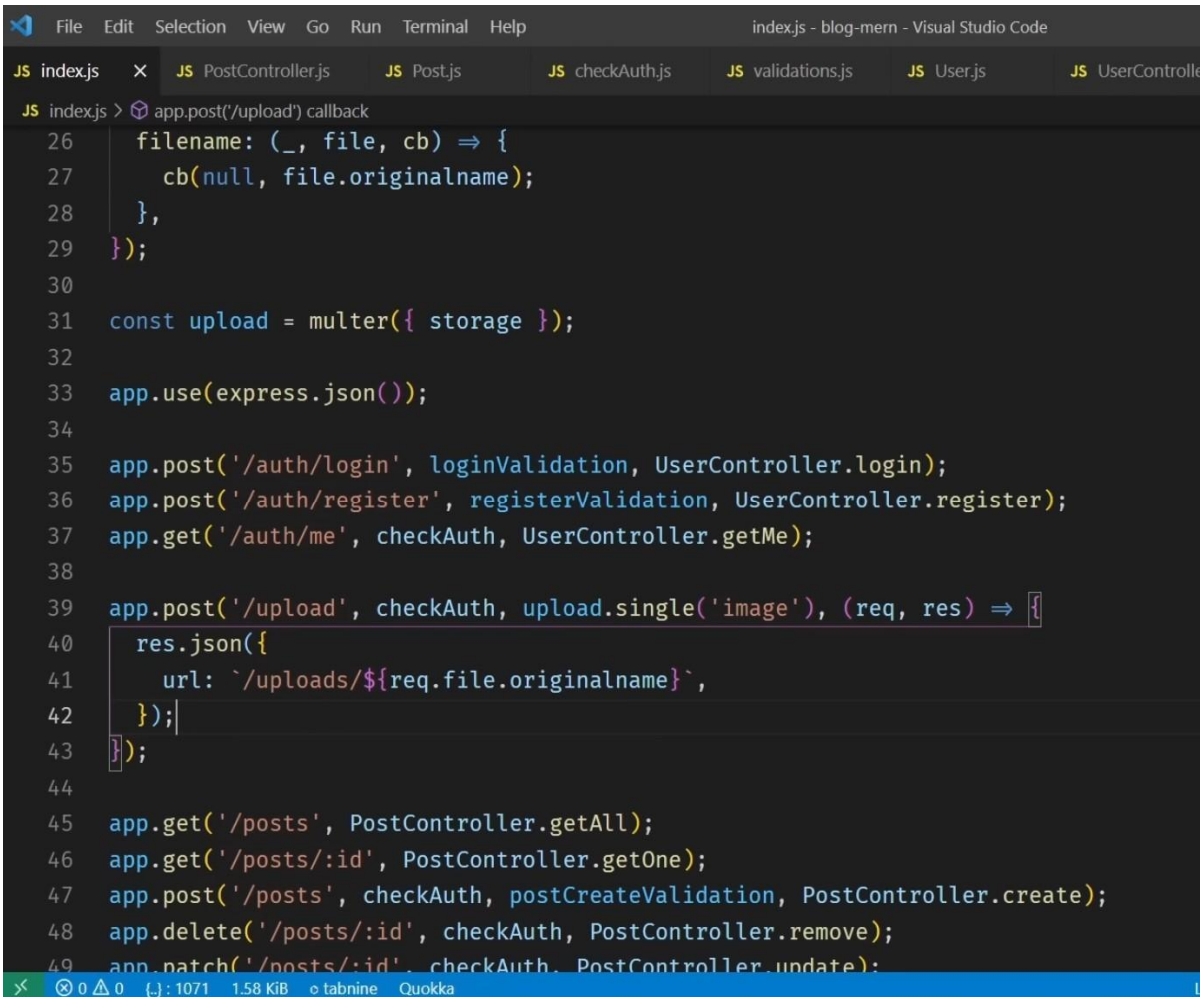

Рисунок 3.11 – Підключення React Router

React Router – це бібліотека для маршрутизації у додатках React. Вона дозволяє визначати різні маршрути та компоненти, які повинні відображатися для кожного маршруту [31]. Використання React Router забезпечує плавну навігацію між сторінками додатку без перезавантаження браузера, що значно

покращує користувацький досвід.

Для управління станом використовується Redux Toolkit. Цей крок налаштовує Redux Toolkit для управління станом користувача у додатку. Використання Redux дозволяє централізовано керувати станом та полегшує обробку складних взаємодій між компонентами.

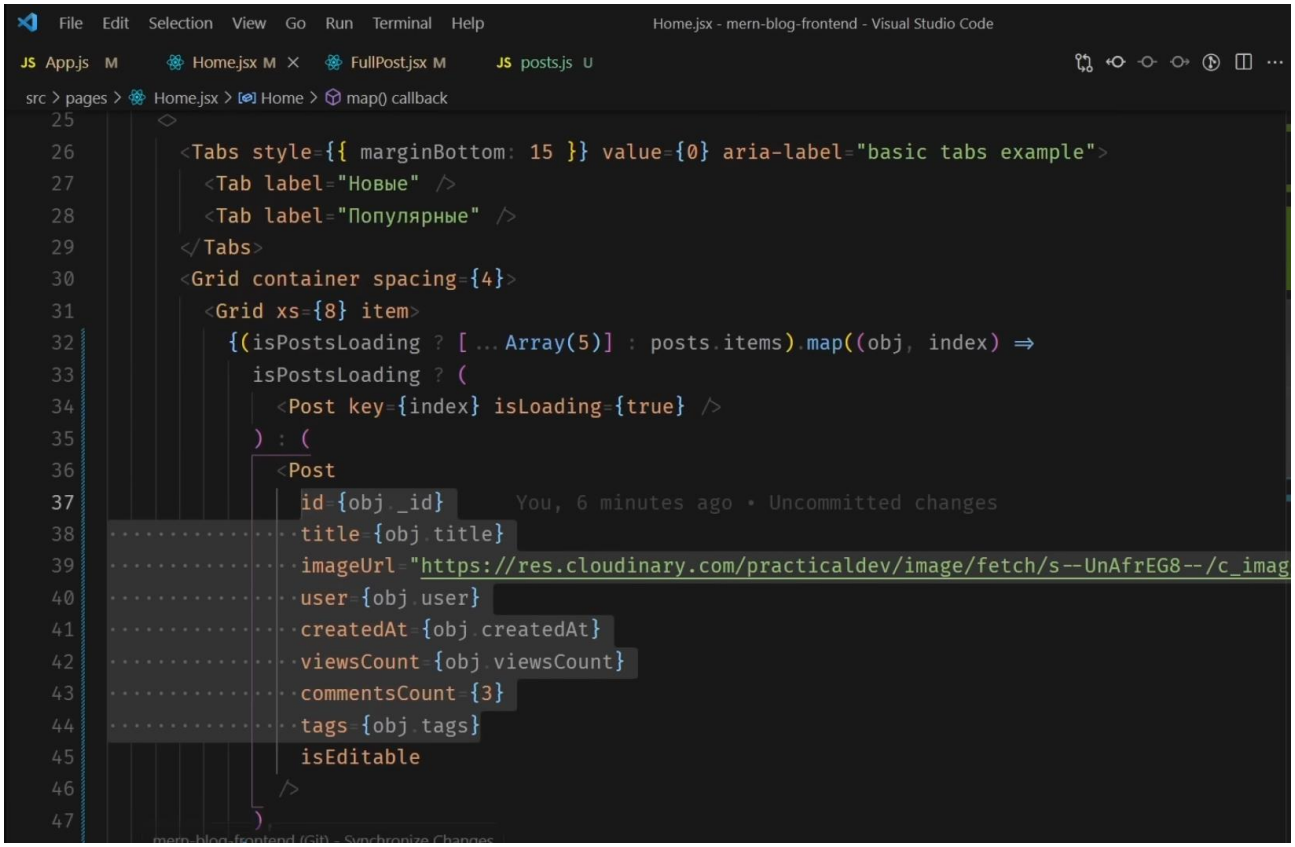

Рисунок 3.12 – Підключення Redux Toolkit

Redux Toolkit – це набір інструментів для ефективного управління станом у додатках React. Вона включає зручні утиліти для створення редюсерів, дій та сховищ, що спрощує налаштування та використання Redux [32]. Централізоване управління станом дозволяє легко відстежувати та контролювати зміни стану у додатку, що полегшує розробку та підтримку складних додатків.

Останній крок – додавання візуалізації постів на фронтенді. Цей крок отримує список постів з сервера та відображає їх на сторінці. Axios – це бібліотека для здійснення HTTP-запитів з браузера або Node.js. Вона дозволяє легко відправляти запити до серверу та обробляти відповіді. Візуалізація постів

включає отримання списку постів з серверу та відображення їх на сторінці у вигляді компонентів React. Це забезпечує динамічне відображення контенту та покращує взаємодію користувачів з форумом.

Таким чином, розробка веб-форуму включає в себе низку кроків, кожен з яких має важливе значення для забезпечення функціональності, безпеки та зручності використання платформи. Починаючи з підключення необхідних імпортів та налаштування базового сервера на Express, через створення моделей для користувачів та постів, до підключення фронтенд технологій, таких як React Router та Redux Toolkit, кожен етап сприяє створенню повноцінного та ефективного веб-додатку [33].

3.2 Візуалізація сайту

Перший елемент візуалізації – це екран авторизації, де користувачі можуть увійти в систему за допомогою своїх облікових даних. Інтерфейс включає поля для введення електронної пошти та паролю, а також кнопку для підтвердження входу. В разі помилки або неправильних даних, система виводить відповідне повідомлення.

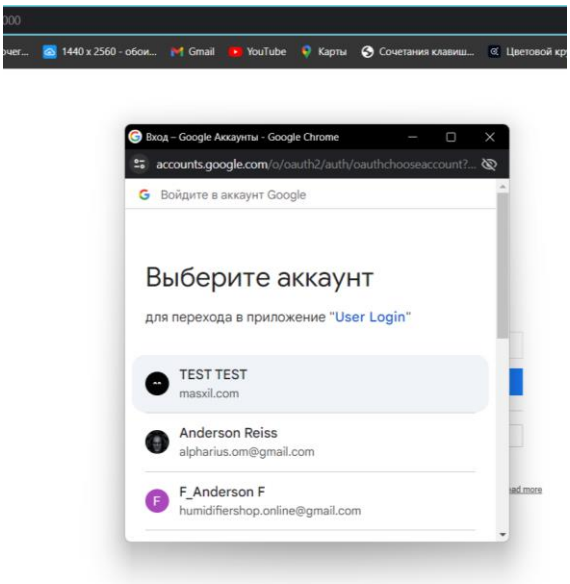

Рисунок 3.13 – Авторизація

Після успішної авторизації користувачі, які мають роль адміністратора, можуть побачити список активних користувачів. Це дозволяє адміністраторам моніторити активність користувачів на форумі. Інтерфейс відображає імена користувачів, їх статус та можливі дії, які можна виконати з їх обліковими записами.

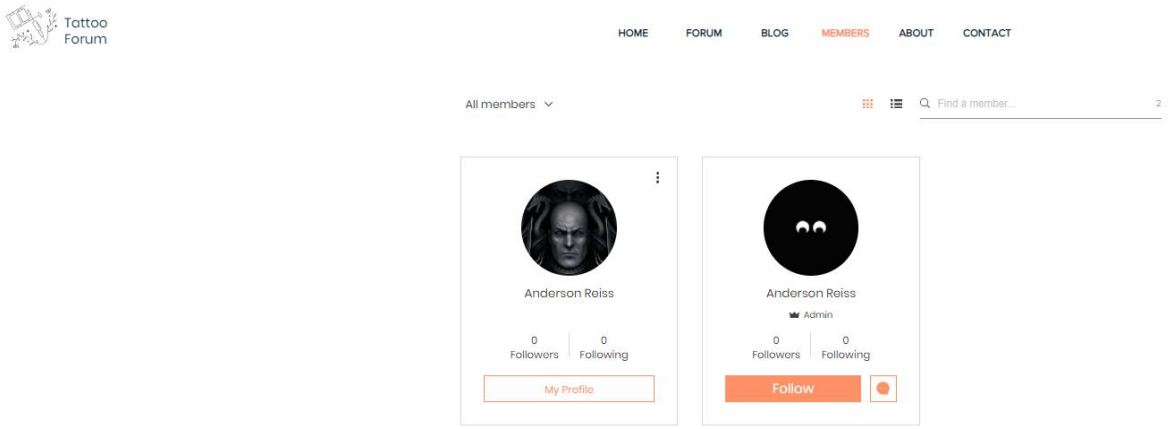

Рисунок 3.14 – Активні користувачі

Головна сторінка є центральною частиною веб-форуму, де користувачі можуть бачити останні пости, популярні теми та навігаційне меню для переходу до інших розділів сайту. Вона включає в себе огляд основних категорій та швидкий доступ до основних функцій.

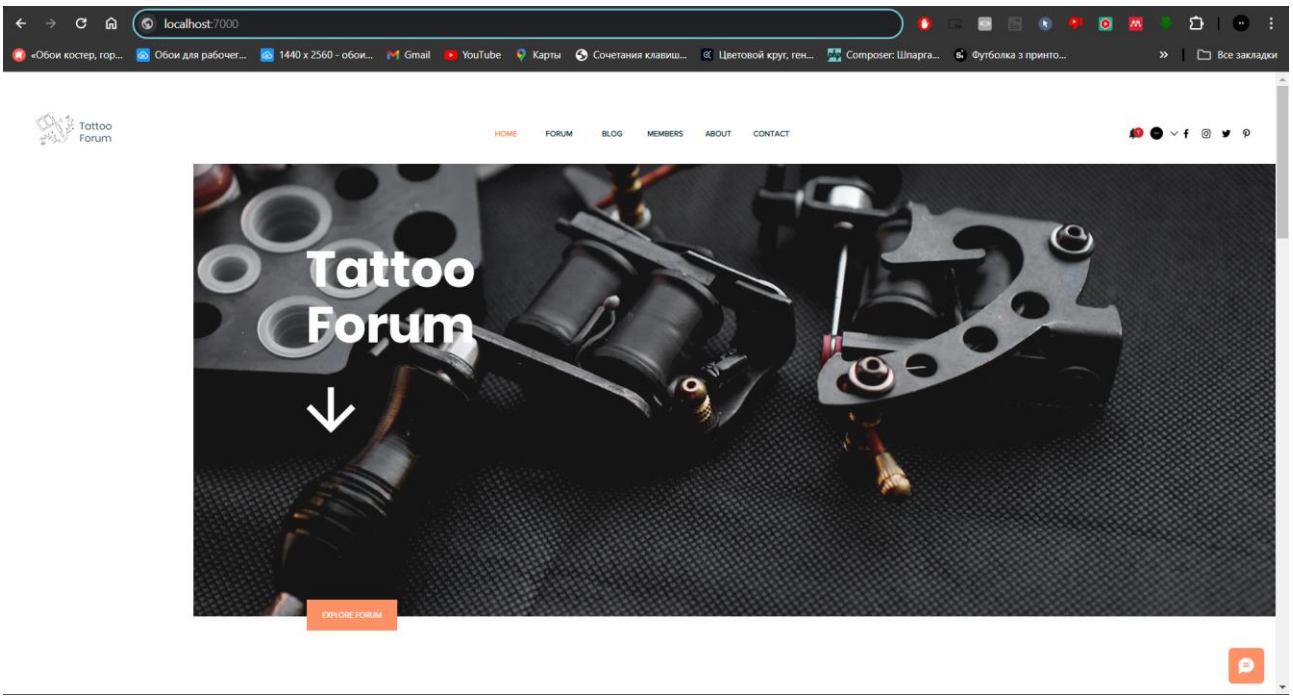

Рисунок 3.15 – Головна сторінка

Ця частина головної сторінки містить додаткові елементи, такі як віджети, посилання на нові дискусії та розділи, що представляють інтерес для користувачів. Тут також можуть бути розміщені рекламні банери або оголошення від адміністрації.

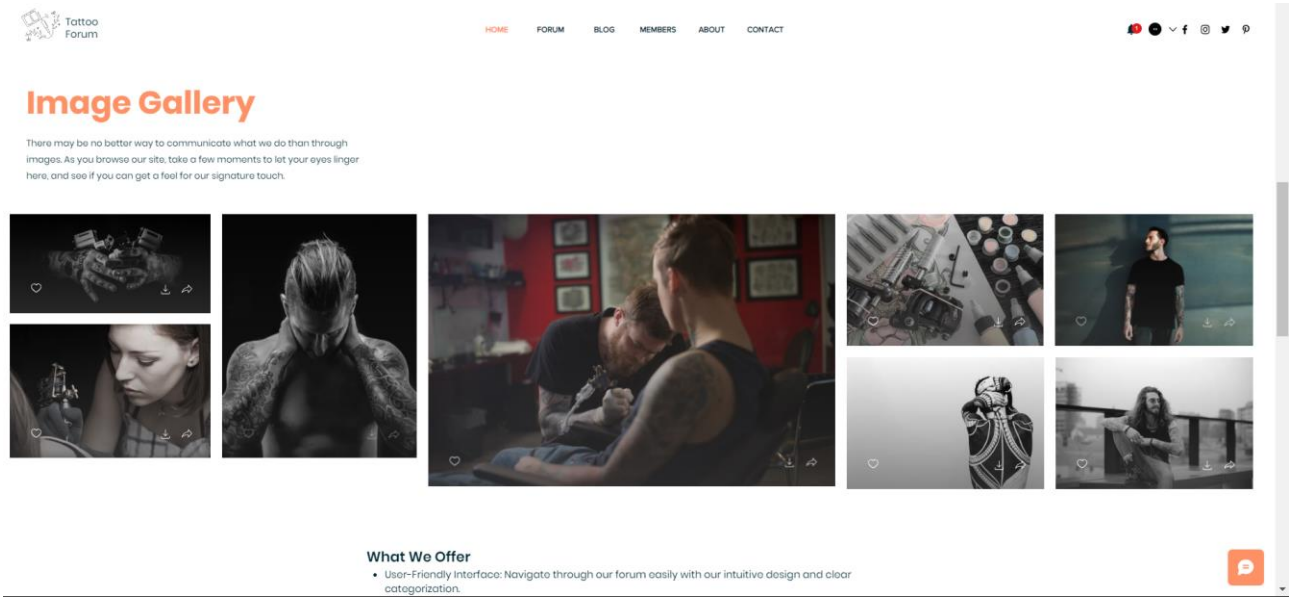

Рисунок 3.16 – Продовження головної сторінки

Footer – це нижня частина головної сторінки, яка містить контактну інформацію, посилання на політики конфіденційності, умови використання та

інші важливі відомості. Він також містить посилання на соціальні мережі та додаткові навігаційні елементи

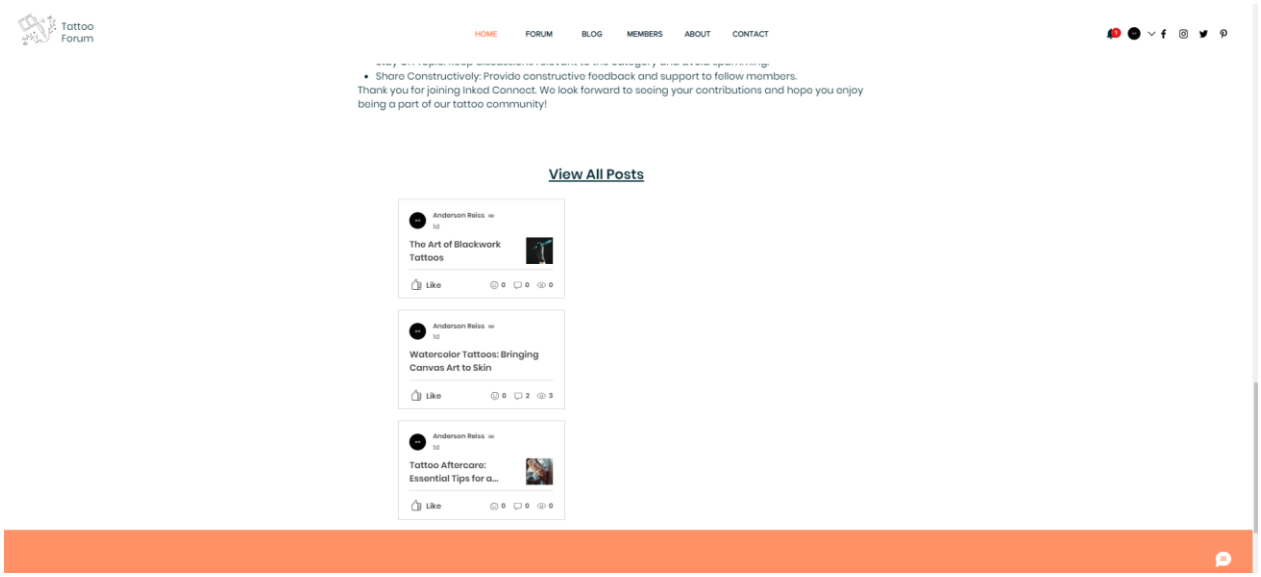

Рисунок 3.17 – Footer головної сторінки

Цей елемент візуалізації дозволяє користувачам створювати нові пости або блоги. Інтерфейс включає форми для введення заголовку, тексту посту та завантаження зображень. Користувачі можуть додавати теги для кращої категоризації та пошуку постів.

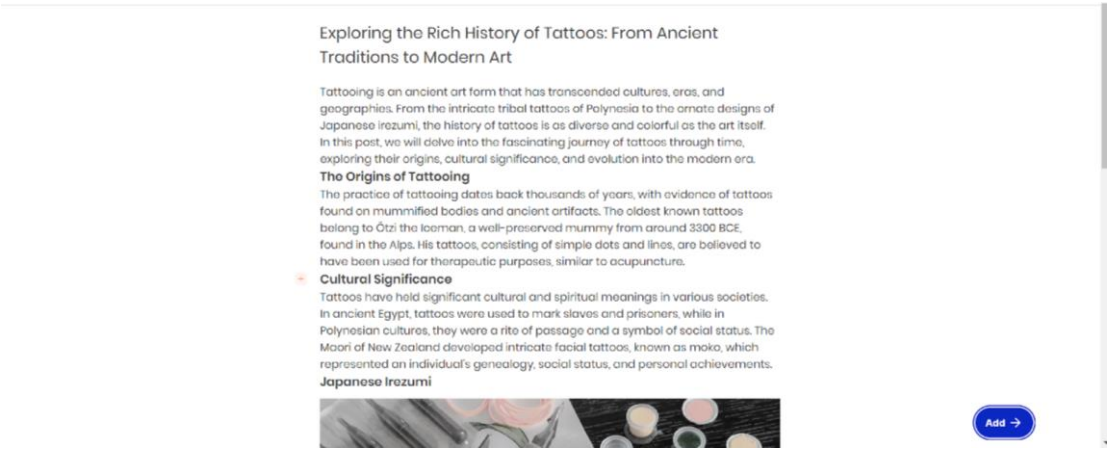

Рисунок 3.18 – Додавання контенту на сайт

Пости відображаються у вигляді карток, що містять заголовок, короткий текстовий опис, ім'я автора, дату публікації та прикріплені зображення. Це забезпечує зручний та візуально привабливий спосіб перегляду контенту.

Рисунок 3.19 – Візуальний вигляд посту

Цей інтерфейс дозволяє користувачам залишати коментарі до постів, взаємодіяти з іншими користувачами, ставити лайки та відповідати на коментарі. Форма для коментування включає поле для тексту та кнопку для відправлення коментаря.

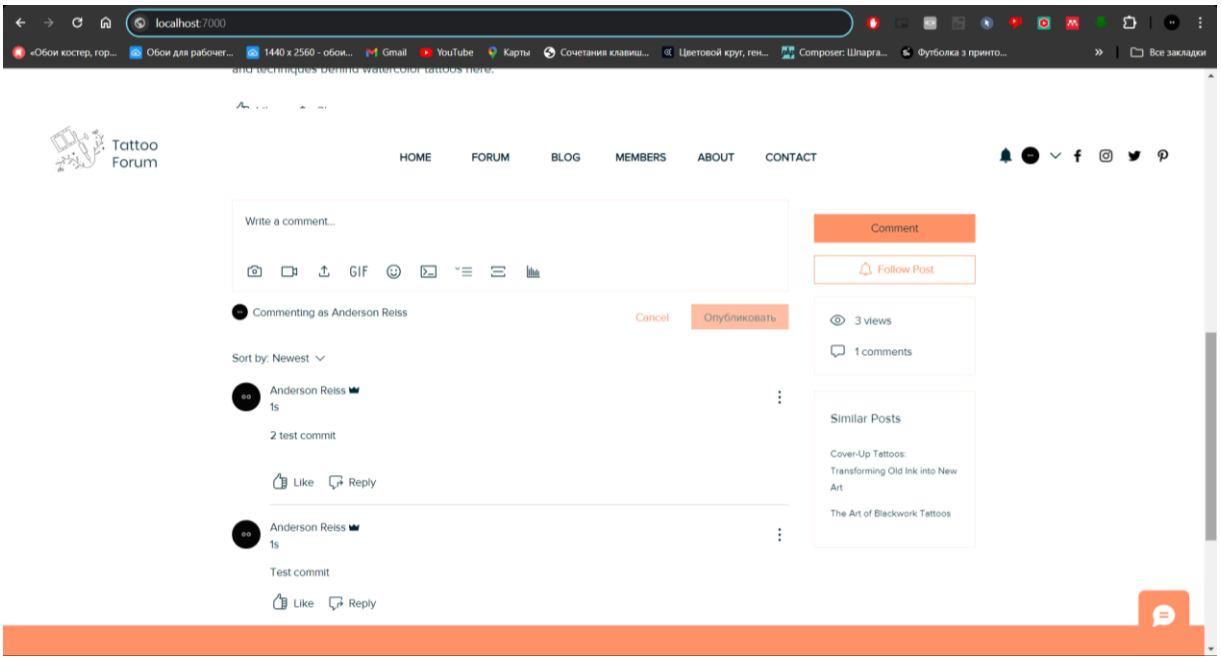

Рисунок 3.20 – Додавання коментарів на пости

Категорії блога допомагають структурувати контент на сайті. Кожна категорія має свою сторінку, де користувачі можуть переглядати пости, відфільтровані за цією категорією. Це дозволяє користувачам легко знаходити цікаві їм теми.

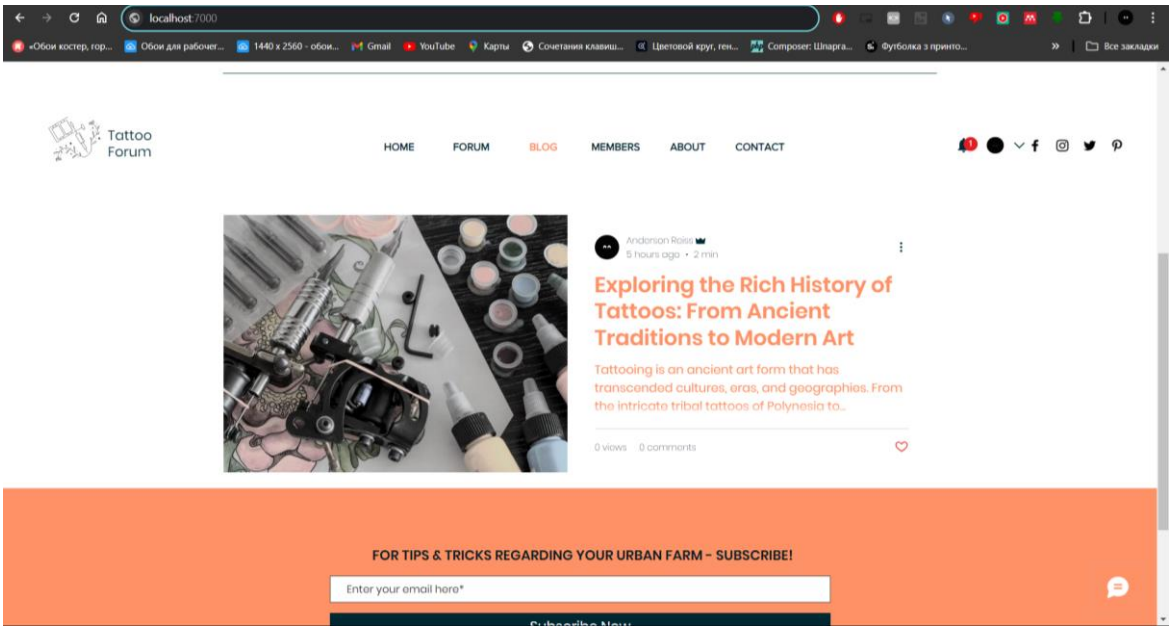

Рисунок 3.21 – Категорія блог

Сторінка профілю користувача містить інформацію про користувача, його активність на форумі, створені пости та коментарі. Тут користувач може редагувати свої дані, змінювати аватар та налаштування акаунту.

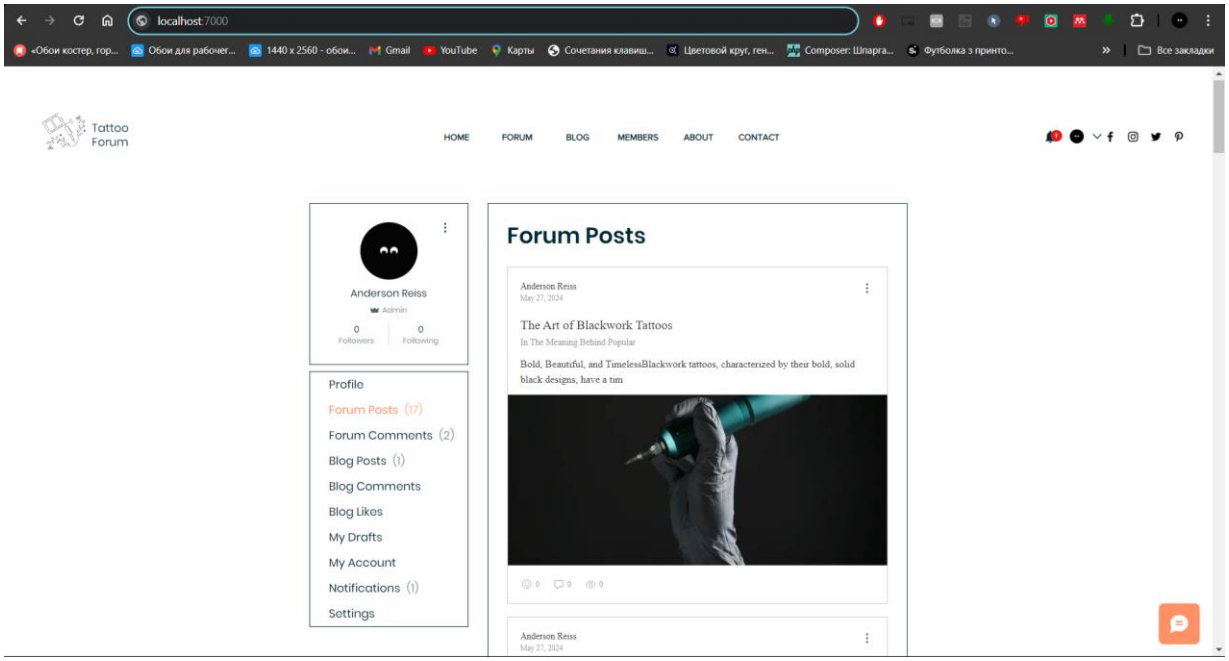

Рисунок 3.22 – Сторінка з профілем

Цей розділ відображає найбільш обговорювані та популярні теми на форумі. Користувачі можуть швидко приєднатися до дискусій, переглянути останні коментарі та поділитися своїми думками.

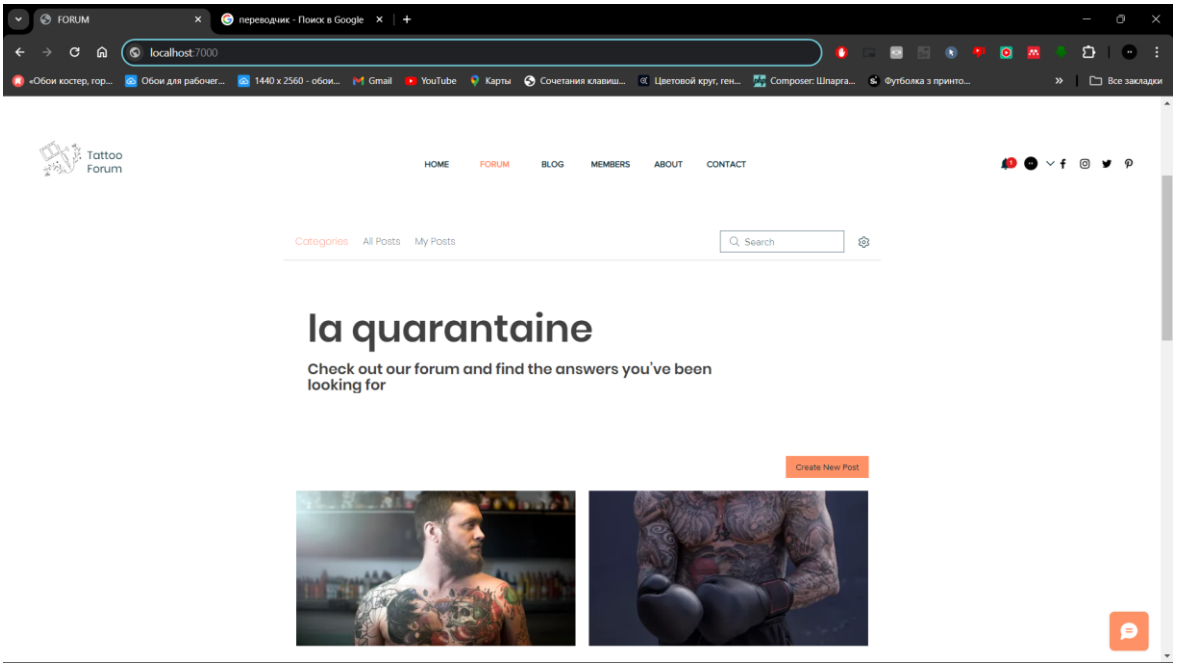

Рисунок 3.23 – Головні теми для обговорення

Візуалізація сайту є важливою складовою, що дозволяє забезпечити зручність та привабливість користувацького інтерфейсу. Кожен з представлених елементів допомагає користувачам легко взаємодіяти з платформою та насолоджуватися спілкуванням на форумі.

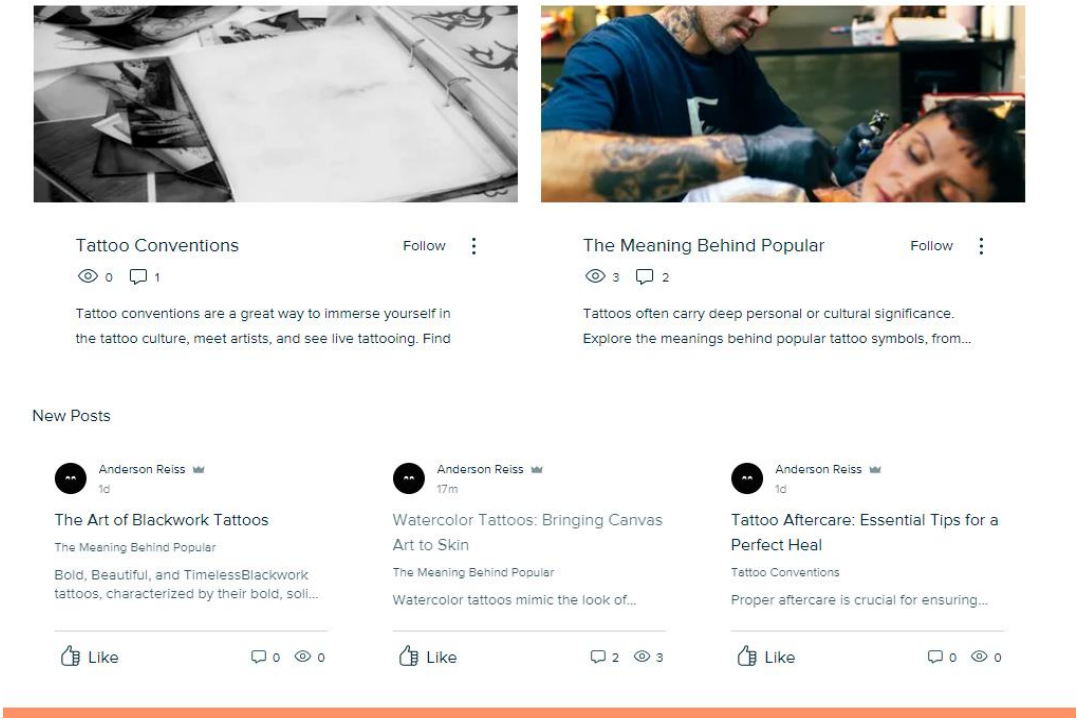

Рисунок 3.24 – Головні теми для обговорення на форумі

3.3 Тестування

Jest – це потужний та гнучкий фреймворк для тестування JavaScript, створений Facebook. Він забезпечує простий та ефективний спосіб написання та запуску тестів для коду. Jest підтримує як модульне тестування, так і інтеграційне тестування, а також забезпечує можливість мокінгу (імітації) модулів для більш ізольованого тестування.

Основні характеристики Jest включають автоматичне визначення тестів, можливість мокінгу модулів, знімки стану (snapshot testing), паралельне виконання тестів та покриття коду. Jest автоматично визначає файли тестів на основі конфігурації або стандартних найменувань файлів, дозволяє імітувати функції та модулі для створення ізольованих середовищ тестування, зберігає знімки рендерингу компонентів для порівняння з актуальним станом під час виконання тестів, виконує тести паралельно, що значно пришвидшує процес тестування, та може генерувати звіти про покриття коду, що допомагає зрозуміти, які частини коду були протестовані.

При запуску, Jest сканує проект на наявність тестових файлів. Це можуть бути файли з розширенням .test.js або .spec.js. Після цього Jest запускає тести, викликаючи кожну тестову функцію та перевіряючи результати. Якщо тест використовує імітацію модулів, Jest замінює оригінальні модулі моками для ізольованого тестування. Після виконання тестів, Jest генерує звіти з результатами, включаючи інформацію про покриття коду.

Для написання тестів для функцій постів використовується Jest та Supertest. Supertest дозволяє здійснювати HTTP-запити до API кінцевих точок та перевіряти їх відповіді. Для цього підключаємося до тестової бази даних, створюємо тестового користувача та отримуємо токен для авторизації. Під час виконання тестів перевіряємо створення постів, отримання постів за ID, видалення постів, лайк/анлайк постів, відповідь на пост, отримання постів у стрічці та отримання постів користувача за ім'ям. Після виконання тестів базу даних очищують і підключення закривають.

Наприклад, для створення посту перевіряється, що новий пост створюється успішно і що не можна створити пост без тексту. Для отримання посту перевіряється, що пост можна отримати за ID та що повертається відповідне повідомлення, якщо пост не знайдено. Для видалення посту перевіряється, що пост видаляється успішно за ID та що повертається відповідне повідомлення, якщо пост не знайдено. Лайк/анлайк посту перевіряється, що пост можна лайкнути та анлайкнути. Для відповіді на пост перевіряється, що відповідь на пост створюється успішно та що повертається відповідне повідомлення, якщо пост не знайдено.

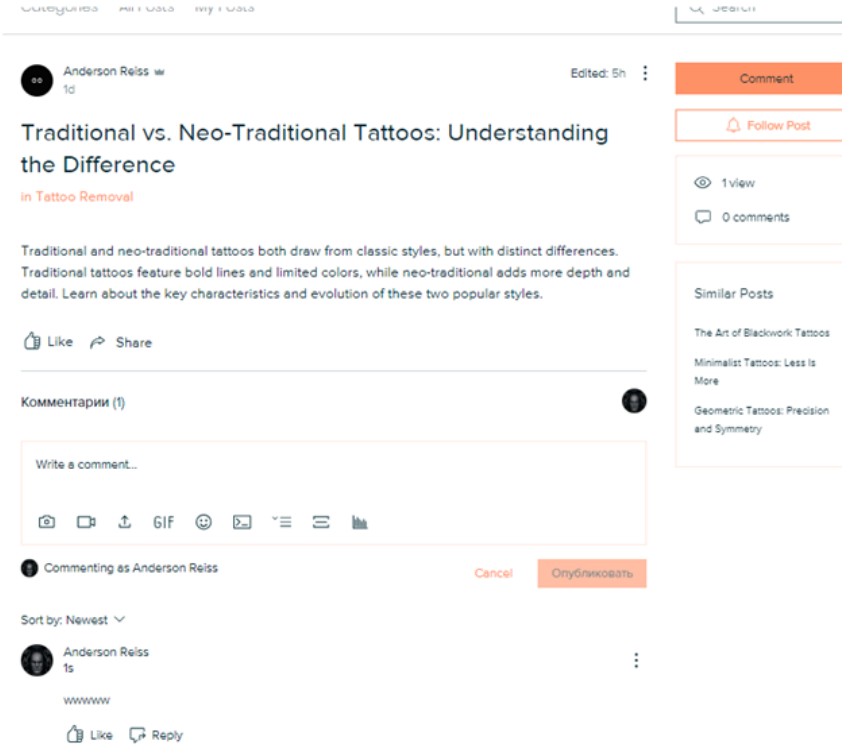

Рисунок 3.25 – Успішна робота теста

Для отримання постів у стрічці перевіряється, що користувач отримує пости у стрічці та що повертається відповідне повідомлення, якщо користувача не знайдено. Для отримання постів користувача перевіряється, що пости можна отримати за ім'ям користувача та що повертається відповідне повідомлення, якщо користувача не знайдено.

Для запуску тестів використовують команду npx jest. Після виконання тестів, отримуємо результати у вигляді звіту, що показує, які тести пройшли успішно, а які – ні. У нашому випадку, всі тести пройшли успішно, що підтверджує правильність роботи реалізованих функцій.

Як показано на Рисунку 3.24, всі тести пройшли успішно, що підтверджує правильність роботи реалізованих функцій.

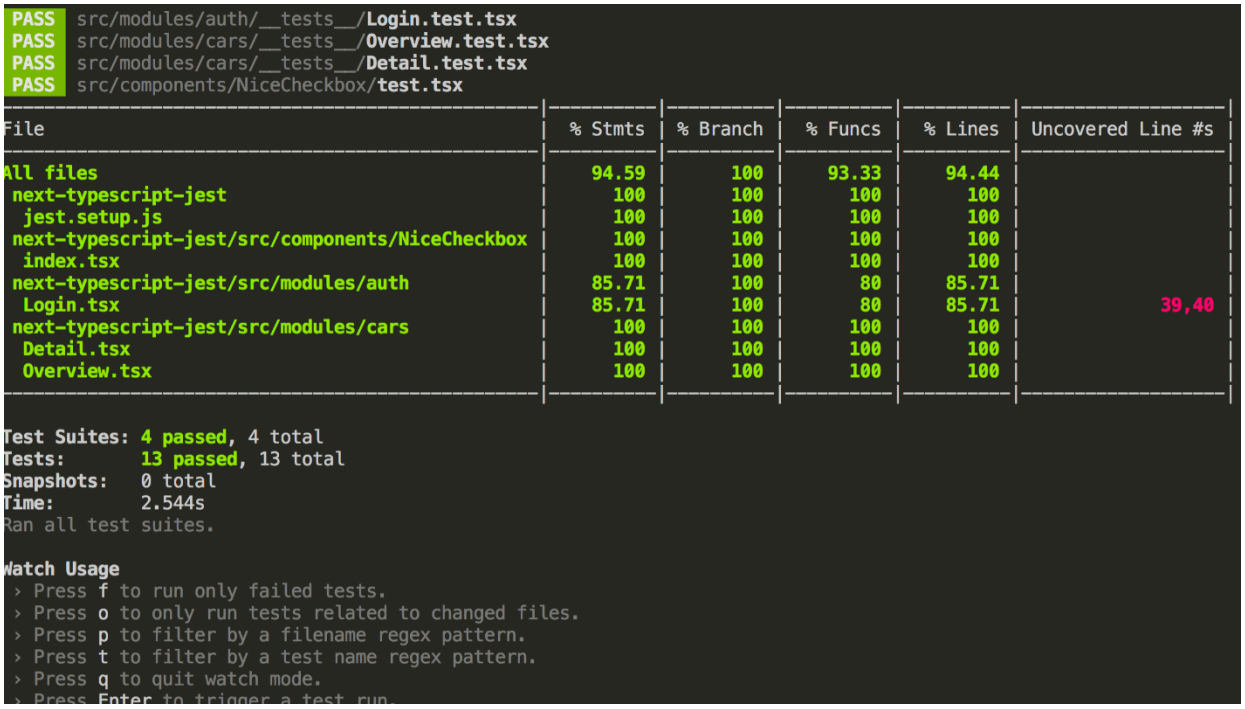

Рисунок 3.26 – Успішні результати тестів

На цьому процес розробки є закінченим, результати тестування свідчать про його успішність.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було проведено комплексне дослідження та розробку інформаційної системи управління контентом для веб-форуму, базованого на стеку MERN (MongoDB, Express, React, Node.js). Основна мета полягала у створенні ефективної, надійної та зручної у використанні системи, яка відповідала б сучасним вимогам до веб-додатків.

Перш за все, було виконано детальний аналіз процесів розробки та створення веб-сайтів, що дозволило визначити ключові етапи цього процесу та обрати найбільш підходящі інструменти та методи для реалізації проекту. Використання сучасних технологій, таких як MongoDB для зберігання даних, Express.js для створення серверної логіки, React для розробки інтерфейсу користувача та Node.js для забезпечення продуктивності та масштабованості, дозволило створити потужний та ефективний веб-додаток.

На етапі проектування та прототипування було розроблено каркаси та макети майбутньої системи, що дозволило узгодити бачення проекту між розробниками та замовником, а також виявити та виправити потенційні проблеми ще до початку розробки. Це сприяло значному зниженню ризиків та підвищенню якості кінцевого продукту.

Процес розробки включав створення як фронтенд, так і бекенд компонентів системи. Завдяки використанню React було створено інтуїтивно зрозумілий та привабливий інтерфейс, який забезпечує високу зручність використання. Express.js та Node.js дозволили реалізувати складну серверну логіку, забезпечуючи швидку та надійну обробку запитів користувачів. MongoDB забезпечила надійне та гнучке зберігання даних, що дозволяє легко масштабувати систему в майбутньому.

Особливу увагу було приділено питанням безпеки та продуктивності. Впровадження сучасних методів аутентифікації та авторизації, шифрування даних та використання інструментів для оптимізації продуктивності дозволило створити систему, яка відповідає високим стандартам безпеки та забезпечує

швидку та стабільну роботу навіть при високих навантаженнях.

У результаті виконання роботи було розроблено інформаційну систему управління контентом для веб-форуму, яка не тільки відповідає сучасним вимогам, але й перевершує їх завдяки використанню новітніх технологій та підходів до розробки. Ця система забезпечує високу продуктивність, гнучкість, масштабованість та зручність використання, що робить її ідеальним рішенням для управління контентом веб-форумів різного масштабу та спрямування.
Важливим аспектом роботи стало також вивчення та аналіз існуючих рішень на ринку, таких як phpBB, vBulletin, XenForo та Discourse. Це дозволило врахувати їхні переваги та недоліки при розробці власної системи та забезпечити її конкурентоспроможність.

Отже, розроблена система є результатом комплексного підходу до проектування та розробки, який поєднує сучасні технології, передові методи та практики, а також глибоке розуміння потреб користувачів. Вона має потенціал стати важливим інструментом для організації та управління веб-форумами, сприяючи розвитку онлайн-спільнот та забезпечуючи зручний та безпечний простір для обміну інформацією та знаннями.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Monika Mehra, Manish Kumar, Anjali Maurya, Charu Sharma, Shanu. “MERN Stack Web Development”, 2021.
2. Robert T. Douglass , Mike Little , Jared W. Smith "Building Online Communities with Drupal, phpBB, and WordPress", 2006.
3. S. S. Sami, T. Rahman, K. S. Hasan and J. Siddique, "An application program interface for vBulletin," 2008 11th International Conference on Computer and Information Technology, Khulna, Bangladesh, 2008
4. Xenforo. "Xenforo Official Community." [Електронний ресурс] – Режим доступу: https://xenforo.com/community/resources/
5. Discourse. "What is Discourse?" [Електронний ресурс] – Режим доступу: https://www.discourse.org/about
6. Wireframe. "What is wireframing? " [Електронний ресурс] – Режим доступу: https://www.figma.com/resource-library/what-is-wireframing/
7. UI/UX. "Що таке ux/ui-дизайн" [Електронний ресурс] – Режим доступу: https://redstone.media/shcho-take-ux-ui-dysayn/
8. W3Schools. "HTML and CSS Basics." [Електронний ресурс] – Режим доступу: https://www.w3schools.com/html/
9. Kane, Robert. "JavaScript Everywhere: Building Cross-Platform Applications with GraphQL, React, React Native, and Electron." O'Reilly Media, 2019.
10. Ktiriani, Q., Arief Kelik Nugroho and Eddy Maryanto. “Frontend development in the final study management system (sipeda) at the engineering faculty of jenderal soedirman university.” Jurnal Teknik Informatika (Jutif)”, (Apr. 2022), 321-329.
11. Filipova, O., Vilão, R. “Backend Development. In: Software Development From A to Z”, 2018.
12. Elliotte Rusty Harold. "Effective Node.js: Best Practices for the Javacript

Backend." Manning Publications, 2020.

13. LAMP. "What is a Lamp Stack?" [Електронний ресурс] – Режим доступу: https://aws.amazon.com/what-is/lamp-stack/

14. MEAN. "What Is the MEAN Stack?" [Електронний ресурс] – Режим доступу: https://www.mongodb.com/resources/languages/mean-stack

15. JAMstack. "What is Jamstack?" [Електронний ресурс] – Режим доступу: https://jamstack.org/

16. Ruby on Rails. "Ruby on Rails Guides" [Електронний ресурс] – Режим доступу: https://guides.rubyonrails.org/

17. Django. "Django documentation" [Електронний ресурс] – Режим доступу: https://docs.djangoproject.com/en/5.0/

18. Morgan, Tyler. "The Complete Guide to Building a Scalable Web Application Using the MERN Stack." Independently published, 2021.

19. Hunter, Adam. "Full-Stack React Projects: Modern web development using React, Node, Express, and MongoDB." Packt Publishing, 2018.

20. Banker, Kyle. "MongoDB in Action: Covers MongoDB version 4.0, Third Edition." Manning Publications, 2019.

21. Express documentation. "Express Official Documentation." [Електронний ресурс] – Режим доступу: https://expressjs.com/en/4x/api.html

22. React documentation. "React Official Documentation." [Електронний ресурс] – Режим доступу: https://reactjs.org/docs/getting-started.html

23. Node.js Foundation. "Node.js Official Documentation." [Електронний ресурс] – Режим доступу: https://nodejs.org/en/docs/

24. Medium article. "Building a MERN Stack Application." [Електронний ресурс] – Режим доступу: https://medium.com/swlh/building-a-mern-stack-application-37c3a8ecf02c

25. Morgan, Tyler. "The Complete Guide to Building a Scalable Web Application Using the MERN Stack." Independently published, 2021.

26. S. I. Adam, J. H. Moedjahedy and J. Maramis, "RESTful Web Service Implementation on Unklab Information System Using JSON Web Token (JWT),"

2020.

27. Mongoose. "Mongoose Official Documentation." [Електронний ресурс] – Режим доступу: https://mongoosejs.com/

28. Joi documentation. "Joi Official Documentation." [Електронний ресурс] – Режим доступу: https://joi.dev/

29. Bcrypt documentation. "Bcrypt Official Documentation." [Електронний ресурс] – Режим доступу: https://www.npmjs.com/package/bcrypt

30. Multer documentation. "Multer Official Documentation." [Електронний ресурс] – Режим доступу: https://www.npmjs.com/package/multer

31. React Router documentation. " React Router Official Documentation." [Електронний ресурс] – Режим доступу: https://reactrouter.com/en/main

32. Redux Toolkit documentation. "Redux Toolkit Official Documentation." [Електронний ресурс] – Режим доступу: https://redux-toolkit.js.org/introduction/getting-started

33. Williams, Ethan. "React and Redux: Creating Modern Web Applications." Packt Publishing, 2020.

ДОДАТОК

```
import Post from "../models/postModel.js";
import User from "../models/userModel.js";
import { v2 as cloudinary } from "cloudinary";

const createPost = async (req, res) => {
	try {
		const { postedBy, text } = req.body;
		let { img } = req.body;

		if (!postedBy || !text) {
			return res.status(400).json({ error: "Postedby and text fields are required" });
		}

		const user = await User.findById(postedBy);
		if (!user) {
			return res.status(404).json({ error: "User not found" });
		}

		if (user._id.toString() !== req.user._id.toString()) {
			return res.status(401).json({ error: "Unauthorized to create post" });
		}

		const maxLength = 500;
		if (text.length > maxLength) {
			return res.status(400).json({ error: `Text must be less than ${maxLength} characters` });
		}
```

```
            if (img) {
                const uploadedResponse = await cloudinary.uploader.upload(img);
                img = uploadedResponse.secure_url;
            }

            const newPost = new Post({ postedBy, text, img });
            await newPost.save();

            res.status(201).json(newPost);
        } catch (err) {
            res.status(500).json({ error: err.message });
            console.log(err);
        }
};

const getPost = async (req, res) => {
        try {
            const post = await Post.findById(req.params.id);

            if (!post) {
                return res.status(404).json({ error: "Post not found" });
            }

            res.status(200).json(post);
        } catch (err) {
            res.status(500).json({ error: err.message });
        }
};
```

```
const deletePost = async (req, res) => {
	try {
		const post = await Post.findById(req.params.id);
		if (!post) {
			return res.status(404).json({ error: "Post not found" });
		}

		if (post.postedBy.toString() !== req.user._id.toString()) {
			return res.status(401).json({ error: "Unauthorized to delete post" });
		}

		if (post.img) {
			const imgId = post.img.split("/").pop().split(".")[0];
			await cloudinary.uploader.destroy(imgId);
		}

		await Post.findByIdAndDelete(req.params.id);

		res.status(200).json({ message: "Post deleted successfully" });
	} catch (err) {
		res.status(500).json({ error: err.message });
	}
};

const likeUnlikePost = async (req, res) => {
	try {
		const { id: postId } = req.params;
		const userId = req.user._id;

		const post = await Post.findById(postId);
```

```
            if (!post) {
                return res.status(404).json({ error: "Post not found" });
            }

            const userLikedPost = post.likes.includes(userId);

            if (userLikedPost) {
                // Unlike post
                await Post.updateOne({ _id: postId }, { $pull: { likes: userId } });
                res.status(200).json({ message: "Post unliked successfully" });
            } else {
                // Like post
                post.likes.push(userId);
                await post.save();
                res.status(200).json({ message: "Post liked successfully" });
            }
        } catch (err) {
            res.status(500).json({ error: err.message });
        }
};

const replyToPost = async (req, res) => {
        try {
            const { text } = req.body;
            const postId = req.params.id;
            const userId = req.user._id;
            const userProfilePic = req.user.profilePic;
            const username = req.user.username;
```

```
            if (!text) {
                return res.status(400).json({ error: "Text field is required" });
            }

            const post = await Post.findById(postId);
            if (!post) {
                return res.status(404).json({ error: "Post not found" });
            }

            const reply = { userId, text, userProfilePic, username };

            post.replies.push(reply);
            await post.save();

            res.status(200).json(reply);
        } catch (err) {
            res.status(500).json({ error: err.message });
        }
};

const getFeedPosts = async (req, res) => {
    try {
        const userId = req.user._id;
        const user = await User.findById(userId);
        if (!user) {
            return res.status(404).json({ error: "User not found" });
        }

        const following = user.following;
```

```
		const feedPosts = await Post.find({ postedBy: { $in: following } }).sort({ createdAt: -1 });

		res.status(200).json(feedPosts);
	} catch (err) {
		res.status(500).json({ error: err.message });
	}
};

const getUserPosts = async (req, res) => {
	const { username } = req.params;
	try {
		const user = await User.findOne({ username });
		if (!user) {
			return res.status(404).json({ error: "User not found" });
		}

		const posts = await Post.find({ postedBy: user._id }).sort({ createdAt: -1 });

		res.status(200).json(posts);
	} catch (error) {
		res.status(500).json({ error: error.message });
	}
};

export { createPost, getPost, deletePost, likeUnlikePost, replyToPost, getFeedPosts, getUserPosts };
```

postController.Jest.js

```
import request from 'supertest';
import app from '../app'; // припустимо, що app - це ваш Express додаток
import mongoose from 'mongoose';
import Post from '../models/postModel';
import User from '../models/userModel';

describe('Post Controller', () => {
  let user;
  let token;

  beforeAll(async () => {
    // Підключення до тестової бази даних
    await mongoose.connect('mongodb://localhost:27017/test_db', {
      useNewUrlParser: true,
      useUnifiedTopology: true,
    });

    // Створення тестового користувача
    user = new User({ username: 'testuser', email: 'testuser@example.com', password: 'password' });
    await user.save();

    // Отримання токену
    token = 'Bearer somevalidjsonwebtoken'; // Замініть на реальний спосіб отримання токену
  });

  afterAll(async () => {
    // Видалення всіх записів з бази даних
    await User.deleteMany({});
```

```
  await Post.deleteMany({});
  await mongoose.connection.close();
});

describe('createPost', () => {
  it('should create a new post', async () => {
    const response = await request(app)
      .post('/api/posts')
      .set('Authorization', token)
      .send({
        postedBy: user._id,
        text: 'This is a test post',
      });

    expect(response.status).toBe(201);
    expect(response.body.text).toBe('This is a test post');
  });

  it('should not create a post without text', async () => {
    const response = await request(app)
      .post('/api/posts')
      .set('Authorization', token)
      .send({
        postedBy: user._id,
      });

    expect(response.status).toBe(400);
    expect(response.body.error).toBe('Postedby and text fields are required');
  });
});
```

```
  describe('getPost', () => {
    let post;

    beforeEach(async () => {
      post = new Post({ postedBy: user._id, text: 'This is a test post' });
      await post.save();
    });

    it('should get a post by id', async () => {
      const response = await request(app).get(`/api/posts/${post._id}`);

      expect(response.status).toBe(200);
      expect(response.body.text).toBe('This is a test post');
    });

    it('should return 404 if post not found', async () => {
      const response = await request(app).get('/api/posts/123456789012345678901234');

      expect(response.status).toBe(404);
      expect(response.body.error).toBe('Post not found');
    });
  });

  describe('deletePost', () => {
    let post;

    beforeEach(async () => {
      post = new Post({ postedBy: user._id, text: 'This is a test post' });
```

```
    await post.save();
  });

  it('should delete a post by id', async () => {
    const response = await request(app)
      .delete(`/api/posts/${post._id}`)
      .set('Authorization', token);

    expect(response.status).toBe(200);
    expect(response.body.message).toBe('Post deleted successfully');
  });

  it('should return 404 if post not found', async () => {
    const response = await request(app)
      .delete('/api/posts/123456789012345678901234')
      .set('Authorization', token);

    expect(response.status).toBe(404);
    expect(response.body.error).toBe('Post not found');
  });
});

describe('likeUnlikePost', () => {
  let post;

  beforeEach(async () => {
    post = new Post({ postedBy: user._id, text: 'This is a test post' });
    await post.save();
  });
```

```
  it('should like a post', async () => {
    const response = await request(app)
      .post(`/api/posts/${post._id}/like`)
      .set('Authorization', token);

    expect(response.status).toBe(200);
    expect(response.body.message).toBe('Post liked successfully');
  });

  it('should unlike a post', async () => {
    post.likes.push(user._id);
    await post.save();

    const response = await request(app)
      .post(`/api/posts/${post._id}/like`)
      .set('Authorization', token);

    expect(response.status).toBe(200);
    expect(response.body.message).toBe('Post unliked successfully');
  });
});

describe('replyToPost', () => {
  let post;

  beforeEach(async () => {
    post = new Post({ postedBy: user._id, text: 'This is a test post' });
    await post.save();
  });
```

```
  it('should reply to a post', async () => {
    const response = await request(app)
      .post(`/api/posts/${post._id}/reply`)
      .set('Authorization', token)
      .send({ text: 'This is a test reply' });

    expect(response.status).toBe(200);
    expect(response.body.text).toBe('This is a test reply');
  });

  it('should return 404 if post not found', async () => {
    const response = await request(app)
      .post('/api/posts/123456789012345678901234/reply')
      .set('Authorization', token)
      .send({ text: 'This is a test reply' });

    expect(response.status).toBe(404);
    expect(response.body.error).toBe('Post not found');
  });
});

describe('getFeedPosts', () => {
  beforeEach(async () => {
    user.following = [user._id];
    await user.save();

    const post = new Post({ postedBy: user._id, text: 'This is a test post' });
    await post.save();
  });
```

```
  it('should get feed posts for the user', async () => {
    const response = await request(app)
      .get('/api/posts/feed')
      .set('Authorization', token);

    expect(response.status).toBe(200);
    expect(response.body.length).toBeGreaterThan(0);
  });

  it('should return 404 if user not found', async () => {
    const invalidToken = 'Bearer invalidjsonwebtoken';
    const response = await request(app)
      .get('/api/posts/feed')
      .set('Authorization', invalidToken);

    expect(response.status).toBe(404);
    expect(response.body.error).toBe('User not found');
  });
});

describe('getUserPosts', () => {
  let post;

  beforeEach(async () => {
    post = new Post({ postedBy: user._id, text: 'This is a test post' });
    await post.save();
  });

  it('should get posts by username', async () => {
    const response = await request(app).get(`/api/posts/user/${user.username}`);
```

```
  expect(response.status).toBe(200);
  expect(response.body.length).toBeGreaterThan(0);
});

it('should return 404 if user not found', async () => {
  const response = await request(app).get('/api/posts/user/invalidusername');

  expect(response.status).toBe(404);
  expect(response.body.error).toBe('User not found');
});
```