МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ОДЕСЬКА ЮРИДИЧНА АКАДЕМІЯ»
ФАКУЛЬТЕТ КІБЕРБЕЗПЕКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра інформаційних технологій

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«БАГАТОКОРИСТУВАЛЬНИЦЬКА СИСТЕМА БЛОГІНГУ»

Рівень «бакалавр»

Галузь знань 12 «Інформаційні технології», спеціальність 121 «Інженерія програмного забезпечення»

Виконав здобувач 4 курсу

Тітієвський Віталій Олегович

Керівник: к.пед.н., доцент

Лобода Юлія Геннадіївна

Рецензент: к.т.н., доцент

Манаков Сергій Юрійович

Одеса – 2024

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ОДЕСЬКА ЮРИДИЧНА АКАДЕМІЯ»
Факультет кібербезпеки та інформаційних технологій
Кафедра інформаційних технологій
Галузь знань: 12 Інформаційні технології
Спеціальність: 121 Інженерія програмного забезпечення
Назва освітньої програми: Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри інформаційних технологій
доцент Н.І. Логінова
_____ «____ »_____ 20__ року

З А В Д А Н Н Я
на кваліфікаційну (бакалаврську) роботу
здобувачу Тітієвському Віталію Олеговичу

1. Тема роботи: «Багатокористувальницька система блогінгу»
   Керівник роботи: к.пед.н, доцент Юлія Геннадіївна Лобода
   затверджені наказом НУ «ОЮА» № 809-06 від «02» квітня 2024 року
2. Дата видачі завдання «15» вересня 2023 року.
3. Строк подання здобувачем роботи «20» травня 2024 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської роботи	Строк виконання етапів роботи	Примітка
1	Аналіз предметної області	02.10.2023	
2	Проєктування веб-застосунку	04.12.2023	
3	Вибір програмних засобів для реалізації поставлених задач	28.01.2024	
4	Розробка клієнтської частини	14.02.2024	
5	Розробка серверної частини	15.03.2024	
6	Тестування функціональності сайту	26.04.2024	
7	Оформлення звітної частини	19.05.2024	

Здобувач _____
(підпис) (прізвище та ініціали)

Керівник роботи _____
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Тітієвський В.О. Багатокористувальницька система блогінгу. – Кваліфікаційна робота бакалавра за спеціальністю 121 «Інженерія програмного забезпечення».

Кваліфікаційна робота викладена на 53 сторінках основного тексту і 138 сторінках загального тексту, містить 4 розділи, 22 рисунки, 20 таблиць, 2 додатки, 20 використаних джерела.

Метою роботи є дослідження та реалізація багатокористувальницької системи блогінгу з використанням сучасних технологій та підходів до розробки програмного забезпечення.

Об'єктом дослідження є багатокористувальницька система блогінгу.

Предмет дослідження – процес розробки багатокористувальницької системи блогів

У кваліфікаційній роботі розроблено систему блогінгу з можливістю реєстрації, авторизації, оцінення та написання коментарів до постів. Досліджено специфіку розробки систем та організацію функціоналу й інтерфейсу за допомогою стеку технологій MERN. Систему можна використовувати як для особистих цілей, так і для комерційних проєктів, яким потрібна платформа для того, щоб спілкуватися з аудиторією та обмінюватися контентом.

Ключові слова: система, блогінг, JavaScript, Node.js, Express.js, React.js, MongoDB, MERN.

ABSTRACT

Titievskyi V.O. Multi-User Blogging System. – Bachelor’s qualifying work on specialty 121 “Software Engineering”.

The qualification work is laid out on 57 pages, contains 4 sections, 22 illustrations, 20 tables, 2 supplements, and 20 used sources.

The purpose of the work is the research and implementation of a multi-user blogging system using modern technologies and approaches to software development.

The object of research is a multi-user blogging system.

The subject of research is the process of developing a multi-user blogging system.

In the qualifying work, a blogging system was developed with the capability of registration, authorization, evaluation, and writing comments on posts. The specifics of system development and the organization of functionality and interface using the MERN technology stack were researched. The system can be used both for personal purposes and for commercial projects that need a platform to communicate with the audience and share content.

Keywords: system, blogging, JavaScript, Node.js, Express.js, React.js, MongoDB, MERN.

ЗМІСТ

СПИСОК УМОВНИХ ПОЗНАЧЕНЬ

— JWT – JSON Web Token

— MERN – MongoDB, Express, React, Node.js

— JS – JavaScript

— VS Code – Visual Studio Code

— HTTP – HyperText Transfer Protocol

— SPA – Single-page application

— SEO – Search Engine Optimisation

ВСТУП

За останнє десятиліття відбулися неймовірні зміни в засобах передачі інформації, здебільшого завдяки інтернет-технологіям. Соціальні мережі, блоги, месенджери тепер є не просто засобом спілкування, а інструментом для вираження думок, досвіду, ідей. Блог є важливим засобом для створення контенту, який впливає на кожного з нас, на нашу культуру.

У цьому контексті особливого значення набуває питання розробки та впровадження багатокористувальницької системи блогінгу, яка дозволяє користувачам зручно отримувати інформацію та надає змогу поширювати та реагувати на контент.

Об'єктом дослідження є багатокористувальницька система блогінгу.

Предмет дослідження – процес розробки багатокористувальницької системи блогів

Метою цієї роботи є дослідження та реалізація багатокористувальницької системи блогінгу з використанням сучасних технологій та підходів до розробки програмного забезпечення.

Для досягнення поставленої мети потрібно виконати наступні завдання:

— провести огляд існуючих рішень на ринку;

— визначити основні функціональні можливості, переваги та недоліки існуючих систем;

— обрати засоби розробки для реалізації;

— розробити загальну архітектуру системи;

— розробити систему;

— провести тестування функціональних можливостей системи.

Систему можна використовувати як для особистих цілей, так і для комерційних проєктів, яким потрібна платформа для того, щоб спілкуватися з аудиторією та обмінюватися контентом. Саме тому, робота над багатокористувальницькою системою блогінгу відкриває нові горизонти, щодо

спілкувань та обміну інформацією у всесвітній павутині, відповідаючи сучасним потребам користувачів в онлайн-середовищі.

Для реалізації кваліфікаційної роботи був обраний стек технологій MERN. MERN – це один із найпопулярніших стеків технологій для створення односторінкових застосунків. Цей стек технологій складається з: MongoDB, Express, ReactJS, NodeJS.

Результати кваліфікаційного дослідження апробовані в таких наукових публікаціях:

1) Тітієвський В.О. Специфіка розробки бекенду засобами Express.JS. *Кібербезпека в сучасному світі:актуальні виклики*: матер. III Всеукраїнської наук.-практ. конф. (м. Одеса, 25 листопада 2022 р.) Одеса: Видавничий дім «Гельветика», 2022. С. 112-115.

2) Тітієвський В.О. Техніки тест-дизайну. *Українське суспільство та наука в умовах воєнного стану й євроінтеграційних перетворень: виклики, напрями відновлення і розвитку*: у 2 т. : матер. Всеукраїнської наук.-практ. конф. здобувачів вищої освіти (м. Одеса, 20 травня 2023 р). Одеса: Видавництво «Юридика», 2023. Т. 2. С. 473-475.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ВИБІР ЗАСОБІВ РОЗРОБКИ

1.1 Специфіка розробки програмних систем

Розробка програмних систем – це складний та різноманітний процес, який вимагає ретельного аналізу потреб користувачів та вибору відповідних засобів розробки. Специфіка розробки програмних систем включає такі аспекти:

— першим етапом розробки програмної системи є збір та аналіз вимог до системи. Це включає визначення функціональних та нефункціональних вимог, а також взаємодію з користувачами для з'ясування їх потреб;

— розробка архітектури системи полягає у визначенні структури та взаємозв'язків між її компонентами. Це важливий крок, який визначає загальну організацію програми та її майбутню розширюваність;

— вибір мов програмування, фреймворків, баз даних та інших технологій залежить від потреб системи, вимог до продуктивності, масштабованості та інших факторів;

— етап розробки та тестування включає написання коду програми, його тестування на наявність помилок та відповідність вимогам, а також внесення необхідних змін;

— після завершення розробки програмна система впроваджується в експлуатацію, після чого може здійснюватися її супровід, включаючи підтримку, оновлення та виправлення помилок.

Існує багато різних архітектурних шаблонів, мов програмування, розробки та тестування для розробки програмного забезпечення.

Залежно від типу вибраного програмного забезпечення будуть обрані відповідні засоби розробки.

Це може бути звичайний десктопний або мобільний додаток, вебсайт, сервіс тощо.

Серед програмного забезпечення все більшої популярності набувають веб та мобільні додатки. Рейтинг мов програмування [3]:

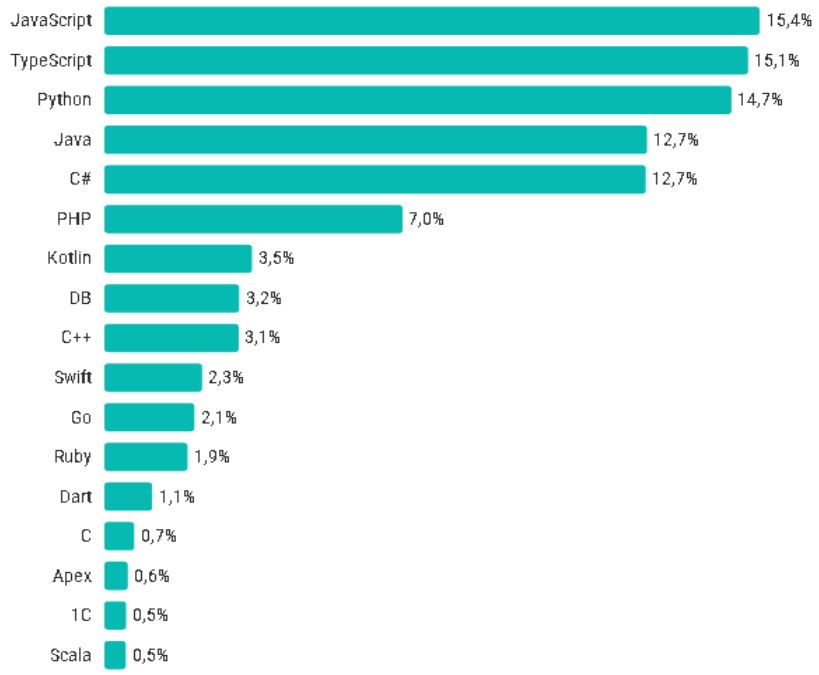

Рисунок 1.1 – Вигляд рейтингу мов програмування

На рис. 1.1 наглядно продемонстровано рейтинг мов програмування, де видно, що найбільш популярною мовою програмування прямо зараз є JavaScript. Це підтверджує твердження про популярність розробки саме веб та мобільних додатків.

Рейтинг платформ [3]:

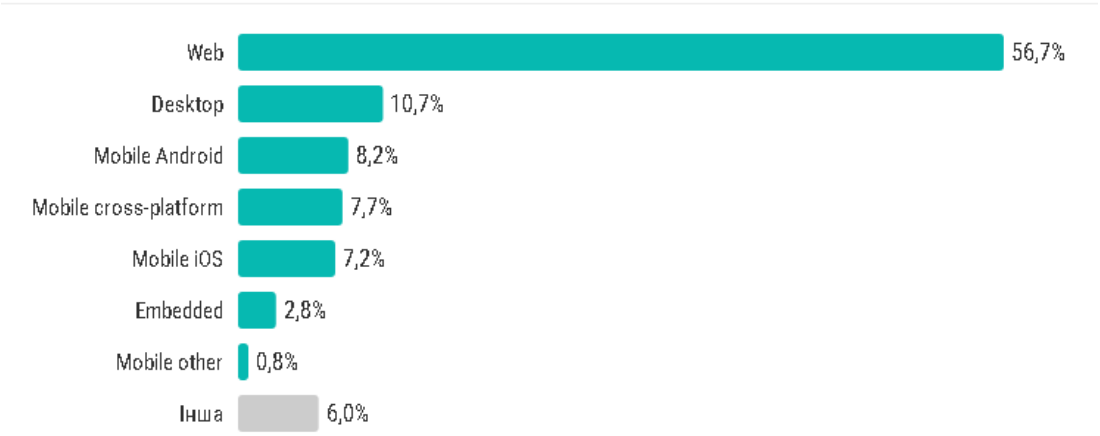

Рисунок 1.2 – Вигляд рейтингу платформ

Це підтверджує твердження про популярність розробки саме вебзастосунків.

1.2 Аналіз наявних аналогів

Аналіз наявних аналогів є важливим етапом у розробці будь-якого продукту або послуги. Цей процес дозволяє отримати глибше розуміння ринку, а також знайти можливості для подальшого вдосконалення продукту.

Аналіз аналогів допомогає визначити які продукту чи послуги вже існують, які їхні характеристики та як вони працюють. Це дозволяє визначити конкурентоспроможність власного продукту. Також шляхом аналізу аналогів можна виявити сильні та слабкі сторони конкурентів. Аналіз аналогів допомогає зменшити ризик невдачі, оскільки розробники можуть дослідити недоліки та помилки конкурентних продуктів та уникнути їх у власному проєкті.

У світі існує велика кількість систем блогінгу, можливості та переваги найпопулярніших з них ведені у табл. 1.1:

Таблиця 1.1 – Можливості аналогів та порівняння з конкурентами

Можливості	**Medium**	**Substack**	**MyBlog**
Легкість використання	+	+	+
Розширюваність функціоналу	-	-	+
Адаптивність	+	+	+
Можливість створення дискретної реклами	-	-	+

За допомогою аналізу наявних конкурентів продемонстровано переваги та недоліки схожих програмних продуктів.

SWOT-аналіз (сильні сторони, слабкі сторони, можливості та загрози) — це система, яка використовується для оцінки конкурентної позиції компанії та розробки стратегічного планування. SWOT-аналіз оцінює внутрішні та зовнішні фактори, а також поточний і майбутній потенціал [4].

SWOT-аналіз це важливий інструмент стратегічного управління, що допомогає оцінити внутрішні та зовнішні фактори, що впливають на продукт.

SWOT-аналіз багатокористувальницької системи блогінгу продемонстравно у табл 1.2:

Таблиця 1.2 – SWOT аналіз

Сильні сторони:	**Слабкі сторони:**
1. Ефективна монетизація через рекламу та інші канали	1. Потенційна проблема безпеки даних клієнтів 2. Конкуренція з іншими платформами для ведення блогів
Можливості:	**Загрози:**
1. Розширення функціоналу платформи, включаючи більш привабливі типи контенту (відео, аудіо) 2. Збільшення охоплення, співпрацюючи з відомими брендами та впливовими особами	1. Нові тенденції у всесвітній павутині змінюють популярність блогів 2. Конкуренція з боку нових учасників ринку, які можуть запропонувати інноваційні функції

Отже, SWOT-аналіз є важливим інструментом для розробки стратегії, що допомагає проєктам досягати успіху.

1.3 Вибір засобів розробки

Для вибору засобів розробки важливо враховувати специфіку системи, вимоги до продукту, компетентність команди розробників та інші фактори. Популярні засоби розробки програмних продуктів включають в себе мови програмування, такі як Java, Python, C#, фреймворки, такі як React, Angular, Django, та інші інструменти розробки програмного забезпечення.

При розробці системи було вирішено обрати мову програмування JavaScript з таких причин:

— JavaScript є основною мовою програмування для веброзробки, всі сучасні браузери підтримують її;

— JavaScript підтримує асинхронне програмування, що дозволяє ефективно обробляти велику кількість запитів до сервера;

— JavaScript може бути використаний як на клієнтський стороні, так і на серверній, що спрощує розробку.

При розробці вебзастосунку було вирішено обрати стек технологій MERN, це пов’язано з такими причинами:

— Зручність використання JavaScript. MERN стек базується на JavaScript, що робить його дуже зручним, оскільки можна використовувати одну мову програмування для розробки як клієнтської, так і серверної частини системи;

— гнучкість та масштабованість. Кожна компонента MERN стеку є досить гнучкою, що дозволяє легко масштабувати додаток в майбутньому, додавати нові функції та адаптувати його під змінні вимоги;

— спрощення розробки SPA. Використання React у складі MERN стеку дозволяє розробникам легко створювати односторінкові додатки (SPA), які забезпечують більш зручний та динамічний досвід для користувачів;

— MongoDB досить просто під’єднати і використовувати для зберігання даних користувачів.

1.3.1 Засоби розробки бази даних

MongoDB — це документоорієнтована база даних із необхідними масштабованістю та гнучкістю з потрібними запитами та індексуванням [5].

MongoDB має купу переваг, що можуть полегшити використання баз даних як з боку розробки, так і з боку використання, а саме:

— простота використання: MongoDB використовує JSON-подібні документи для зберігання даних, що дозволяє легко під`єднати її з багатьма сучасними програмними рішеннями;

— швидкодія: MongoDB добре скалюється горизонтально, що дозволяє розподіляти дані на кілька серверів для підвищення продуктивності та

масштабованості. Крім того, вона використовує внутрішню пам'ять, що може покращити швидкість доступу до даних;

— масштабованість: MongoDB добре підходить для проєктів з великим обсягом даних або високою потужністю завантаження, оскільки вона може легко масштабуватися як вертикально (додавання ресурсів до існуючих серверів) так і горизонтально (додавання нових серверів).

А найголовніше те, що MongoDB гарно поєднується з мовою JavaScript. MongoDB використовує JSON-подібні документи для зберігання даних, що чудово відповідає формату об`єктів JS. Також, комбінація MongoDB та Node.js дозволяє створити програмний продукт , використовуючи одну мову програмування як на клієнтській, так і на серверній стороні, що спрощує розробку коду. Загалом, комбінація MongoDB дозволяє створювати динамічні та ефективні програмні продукти набагато швидше та з меншою кількістю зусиль.

Також при розробці системи було використано таку платформу, як Firebase.

Firebase - це інтегрована платформа від Google, яка сприяє розробці мобільних і веб-додатків, допомагаючи підприємствам створювати, вдосконалювати та масштабувати свої продукти. Завдяки широкому набору інструментів, Firebase значно спрощує процес розробки продукту, зменшуючи час і зусилля, необхідні для створення MVP [6].

Її було обрано по таким причинам:

— простота використання: Firebase надає готове та легке у використанні хмарне зберігання, яке не потребує розгортання та налаштування окремого сервера бази даних, як MongoDB. Ви можете почати використовувати Firebase всього лише за декілька хвилин;

— швидкодія: Firebase пропонує швидке та ефективне хмарне зберігання, що може забезпечити швидкий доступ до фотографій для вашого додатку;

— інтегрованість з іншими сервісами Firebase: Firebase надає інші інструменти, такі як аутентифікація, аналітика, реальний час та інші, які можуть легко інтегруватися з хмарним зберіганням. Це робить Firebase привабливим вибором для повноцінного розробки додатків;

— масштабованість: Firebase дозволяє масштабувати ваше хмарне зберігання відповідно до потреб вашого додатку, навіть при збільшенні обсягу даних;

— безпека: Firebase надає можливості для налаштування прав доступу до файлів, що дозволяє контролювати, які користувачі мають доступ до фотографій.

1.3.2 Засоби розробки frontend частини

Під час розробки клієнтської частини багатокористувальницької системи блогінгу було вирішено використати бібліотека React.

React JS — це відкритий JavaScript-фреймворк, а точніше, бібліотекою JavaScript, яка використовується для розробки інтерфейсів користувача. Він був створений компанією Facebook і швидко набув популярності серед розробників з усього світу. Реакт дозволяє ефективно створювати застосунки з високою продуктивністю і масштабованістю. Одним з ключових концепцій у React JS є компоненти. Вони представляють собою незалежні блоки коду, які відповідають за рендеринг певної частини користувацького інтерфейсу [7].

React чудово підходить для розробки вебзастосунків, в тому числі і для багатокористувальницької системи блогів. Ось кілька причин, чому було використано саме React:

— React базується на компонентах, що дозволяє поділити веб-сторінку на маленькі, повторно використовувані частини. Це особливо корисно для блогінгу, де ми маємо елементи, такі як заголовки, пости, коментарі, форми, тощо;

— react використовує віртуальний DOM для оптимізації оновлення веб-сторінки. Це означає, що React визначає мінімальні зміни, необхідні для оновлення сторінки, і автоматично оновлює лише необхідні елементи. Це допомагає підвищити продуктивність нашого вебзастосунку;

— react дозволяє побудувати односторінковий додаток (SPA), де всі сторінки завантажуються разом, а потім динамічно оновлюються без перезавантаження сторінки. Це може полегшити навігацію для користувачів блогінгу і підвищити швидкість відгуку;

— react дозволяє легко розширювати функціональність блогінгу шляхом додавання нових компонентів або бібліотек. Це дозволяє легко впроваджувати нові функції, такі як коментарі, пошук, аналітика і т. д;

— react має велику активну спільноту розробників і багато корисних бібліотек, які допомагають в розробці.

Отже, використання React може зробити процес розробки вебзастосунків, зокрема блогів, більш ефективним і комфортним завдяки своїм перевагам і зручностям.

У вебзастосунку блогів неможливо обійтись без керування станами, оскільки це дозволяє зберігати та оновлювати дані на сторінці, відповідно до дій користувача.

Для цього було обрано Redux.

Redux – це бібліотека JS для передбачуваного та підтримуваного глобального управління станами. [8].

Ось декілька причин, чому було вирішено обрати саме Redux [8]:

— Redux допомагає писати програми, які ведуть себе узгоджено, запускаються в різних середовищах (клієнт, сервер і рідне) і їх легко тестувати;

— централізація стану та логіки вашої програми надає такі потужні можливості, як скасування/повторення, збереження стану та багато іншого;

— redux DevTools дозволяє легко відстежувати, коли, де, чому та як змінився стан вашої програми. Архітектура Redux дозволяє реєструвати зміни, використовувати «налагодження подорожей у часі» і навіть надсилати повні звіти про помилки на сервер;

— redux працює з будь-яким рівнем інтерфейсу користувача та має велику екосистему додатків, які відповідають вашим потребам.

Також важливою причиною є сумісність з React. React Redux добре поєднується з React і робить взаємодію між React компонентами та Redux станами легшою і прозорішою.

Під час розробки клієнтської частини багатокористувальницької системи блогінгу був використаний CSS-фреймоворк Tailwind.

Tailwind CSS — це фреймворк CSS, перш за все утиліти, який оптимізує веб-розробку , надаючи набір попередньо розроблених класів утиліт. Ці класи дозволяють швидко створювати стилі без написання власного CSS, сприяючи узгодженості та масштабованості. Підхід Tailwind зміщує фокус із традиційних компонентів CSS на функціональні класи, надаючи можливість розробникам ефективно створювати адаптивні та візуально привабливі інтерфейси з мінімальними зусиллями [9].

Tailwind було доцільно використати тому, що:

— Зростає швидкість розробки. Завдяки використанню класів, можна швидко і просто задавати стилі для елементів, без необхідності написання власних CSS правил;

— спрощується кастомізація. Tailwind надає велику кількість налаштувань, що дозволяють легко кастомізувати розміри, шрифти, кольори та навіть тему вебзастосунку;

— поліпшує SEO. Завдяки чітко визначеним класам та логіці розмітки, вебзастосунок стає більш індексованим пошуковими системами.

1.3.3 Засоби розробки backend частини

Під час розробки серверної частини багатокористувальницької системи блогінгу був використаний Node.js.

Зручність використання Node.js, як платформи мови програмування JavaScript, полягає у можливості використовувати одну мову і для бекенду, і для фронтенду, тобто надає повний стек розробки. Використання однієї мови

програмування означає гарну сумісність, високу продуктивність і можливість обміну та повторного використання компонентами, що покращує і прискорює розробку. Node.js надає можливість виконувати скрипти на сервері, що пришвидшує завантаження сторінки та знижує навантаження на процесор користувача. Крім того, Node.js простий для вивчення і має гарні можливості масштабованості [1].

Також було використано фреймворк Express.js

Express — це мінімальний і гнучкий фреймворк вебзастосунків Node.js, який забезпечує надійний набір функцій для веб- і мобільних додатків [10].

Express.js має сильні сторони, що будуть корисні при розробці багатокористувальницької системи блогінгу:

— Express.js має мінімалістичну структуру, що дозволяє швидко створювати вебзастосунки. Він надає лише основні функціональні можливості, але дозволяє розширювати функціональність за допомогою додаткових пакетів;

— Express.js надає розробникам велику свободу в області структури та організації веб-додатків. Він не нав'язує жорстких правил або конвенцій, що дозволяє реалізувати свої ідеї без зайвих обмежень.

— Express.js має велику та активну спільноту розробників, яка підтримує його розвиток та надає безліч корисних плагінів, модулів та інструментів для спрощення розробки.

— Express.js надає потужний механізм маршрутизації, який дозволяє ефективно керувати запитами та відповідями. Є можливість легко налаштовувати маршрути для різних URL-адрес та HTTP-методів.

Отже, Express.js – це потужний та гнучкий інструмент для розробки веб-додатків на Node.js, який допомагає розробникам швидко та ефективно створювати якісні програмні рішення.

Для безпечної авторизації користувача до вебсайту було використано JWT.

Веб-токен JSON (JWT) — це відкритий стандарт (RFC 7519), який визначає компактний і самодостатній спосіб безпечної передачі інформації між сторонами як об’єкт JSON. Цю інформацію можна перевірити та довіряти їй, оскільки вона

має цифровий підпис. JWT можна підписати за допомогою секрету (з алгоритмом HMAC) або пари відкритих/приватних ключів за допомогою RSA або ECDSA.[11].

Отже, JWT дозволяє вам авторизуватися на сайті, надаючи спеціальний токен, який підтверджує вашу ідентичність та дозволяє вам здійснювати дії, які вам доступні на сторінці.

1.4 Висновки до розділу

Отже, аналіз предметної області та вибір засобів розробки є важливим етапом створення системи. Було проведено ретельний аналіз, в якому було розглянуто різні аспекти розробки багатокористувальницької системи блогінгу.

На основі виконаних робіт можна зробити такі висновки:

— специфіка розробки програмних систем вимагає глубокого розуміння предметної області для визначення основних вимог до функціональності;

— аналіз наявних аналогів дозволяє виявити переваги та недоліки існуючих рішень, що допомагає уникнути повторення помилок і впровадити найкращі практики в системі;

— вибір засобів розробки бази даних, таких як MongoDB або Firebase, забезпечує надійне зберігання даних та їхню швидку обробку;

— використання сучасних методів розробки клієнтської частини, таких як React та Redux, дозволяє створювати інтуїтивно зрозумілий та надійний функціонал для користувачів;

— використання сучасних методів розробки серверної частини, таких як Express.js та Node.js, забезпечує стабільну роботу серверної частини та ефективну розробку запитів користувачів.

Врешті решт не існує ідеальних засобів розробки бекенду, кожен з них має свою специфіку, свої сильні та слабкі сторони, які варто враховувати при виборі стеку програми.

Завдяки етапу аналізу вдалося визначити оптимальні засоби розробки для створення багатокористувальницької системи блогінгу, що дозволяє забезпечити високу якість та надійність кінцевого продукту.

2 ПРОЄКТУВАННЯ БАГАТОКОРИСТУВАЛЬНИЦЬКОЇ СИСТЕМИ БЛОГІНГУ

2.1 Архітектура системи

Для реалізації багатокористувальницької системи блогінгу було вирішено використати архітектуру SPA.

SPA (односторінкова програма) — це реалізація веб-програми, яка завантажує лише один веб-документ, а потім оновлює основний вміст цього єдиного документа за допомогою API JavaScript, наприклад Fetch , коли має бути показано інший вміст. Таким чином, це дозволяє користувачам використовувати веб-сайти без завантаження цілих нових сторінок із сервера, що може призвести до збільшення продуктивності та більш динамічного досвіду [12].

Незважаючи на переваги, такий підхід також має свої недоліки:

— Повільне завантаження при першому відвідуванні. При першому завантаженні потрібно завантажити всю програму, що може займати багато часу, особливо якщо програма велика;

— проблеми з безпекою. SPA може бути вразливими до деяких типів атак, таких як атаки на перехоплення і вставку коду (XSS), оскільки вони часто використовуються для відображення динамічного вмісту, який може містити потенційно шкідливий JavaScript;

— питання про витрати ресурсів. SPA може використовувати більше пам'яті та ресурсів браузера, оскільки весь вміст завантажується в один раз і зберігається в пам'яті протягом тривалого часу. Це може призвести до погіршення продуктивності, особливо на старих або обмежених пристроях.

Але, на мою думку, переваги значно перевищують недоліки, тому буде використана саме архітектура SPA.

Архітектура багатокористувальницької системи блогінгу складається з клієнтьскої частини, серверної частини та зв`язком між клієнтом та сервером. Зв`язком між клієнтом та сервером є HTTP запити.

При виконанні запитів можуть виникати різноманітні проблеми як на боці користувача, так і на боці сервера. Якщо користувач не зрозуміє в чому проблема, він просто вийде із вебзастосунку. Для розв’язання цієї проблеми існує функції в Expres.js та коди стану HTTP.

2.2 Інформаційна модель системи

У ході проєктування багатогористувальницької системи блогінгу були виділені моделі, що наведені в табл. 2.1-2.3:

Таблиця 2.1 – Модель «user» та її властивості

Властивість	Тип	Опис
_id	ObjectId	Унікальний ідентифікатор користувача
username	String	Ім’я користувача (унікальне)
email	String	Електронна пошта користувача (унікальна)
password	String	Пароль користувача
profilePicture	String	Посилання на профільне зображення користувача (за замовчуванням – посилання на стандартне зображення профілю)
isAdmin	Boolean	Прапорець, що вказує, чи є користувач адміністратором (за замовчуванням – false)
createdAt	Date	Дата та час реєстрації користувача
updatedAt	Date	Дата та час останнього оновлення даних користувача

Таблиця 2.2 – Модель «post» та її властивості

Властивість	Тип	Опис
_id	ObjectId	Унікальний ідентифікатор поста
userId	String	Ідентифікатор користувача, який створив пост
content	String	Зміст поста
title	String	Заголовок поста

Закінчення табл. 2.2

Властивість	Тип	Опис
image	String	Посилання на зображення поста (за замовчуванням – посилання на стандарте зображення поста)
category	String	Категорія поста (за замовчуванням – «uncategorized»
slug	String	Унікальний ідентифікатор для URL-адреси поста
createdAt	Date	Дата та час створення поста
updatedAt	Date	Дата та час останнього оновлення поста

Таблиця 2.3 – Модель «comment» та її властивості

Властивість	Тип	Опис
_id	ObjectId	Унікальний ідентифікатор коментаря
content	String	Зміст коментаря
postId	String	Ідентифікатор повідомлення, до якого належить коментар
userId	String	Ідентифікатор користувача, який щалишив коментар
likes	Array	Масив ідентифікаторів користувачів, які вподобали коментар
numberOfLikes	Number	Кількість лайків, які має коментар
createdAt	Date	Дата та час створення коментаря
updatedAt	Date	Дата та час останнього оновлення коментаря

На основі цих моделей в базі даних MongoDB буде створено 3 колекції: comments, posts, users.

2.3 Моделювання системи

Моделювання програмної системи є важливою складовою розробки програмного забезпечення. Нижче наведені діаграми для найбільш складних за розумінням компонентів у системі.

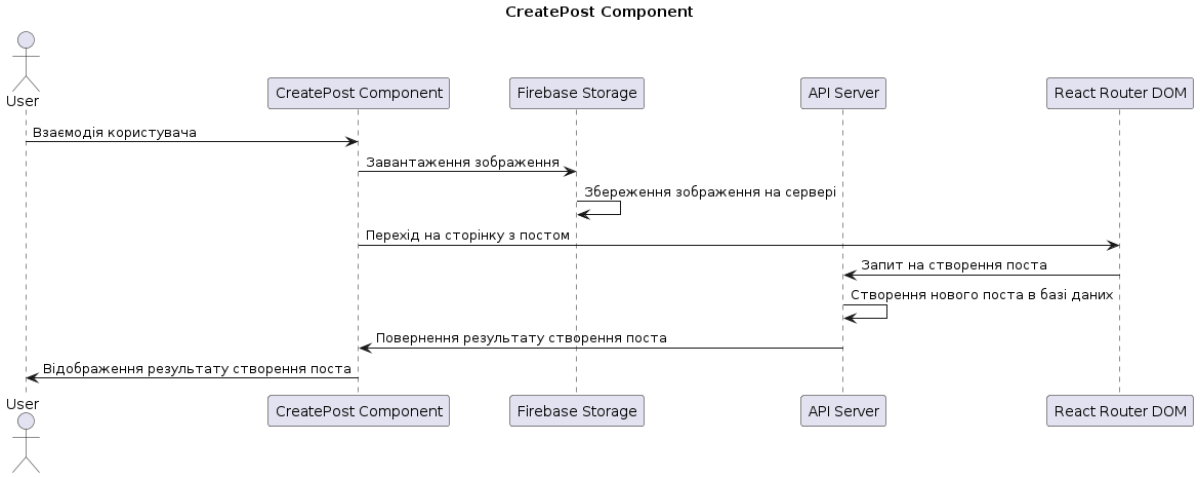

Рисунок 2.1 – Алгоритм компоненту створення поста

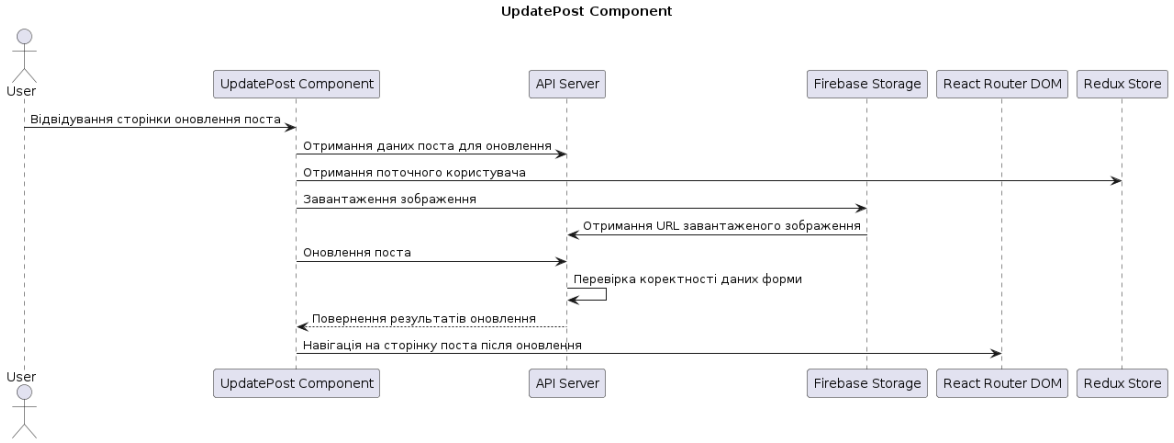

Рисунок 2.2 – Алгоритм компоненту оновлення поста

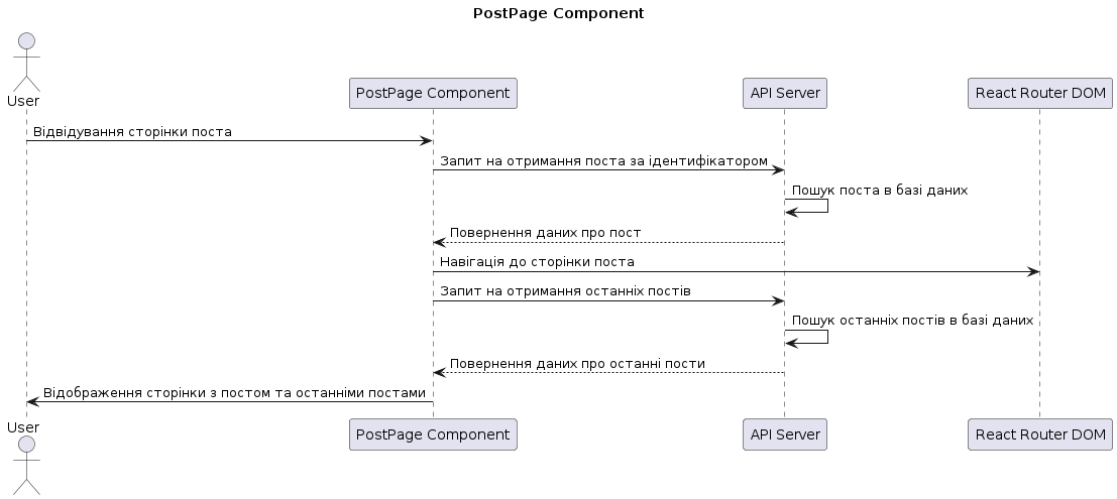

Рисунок 2.3 – Алгоритм компоненту поста

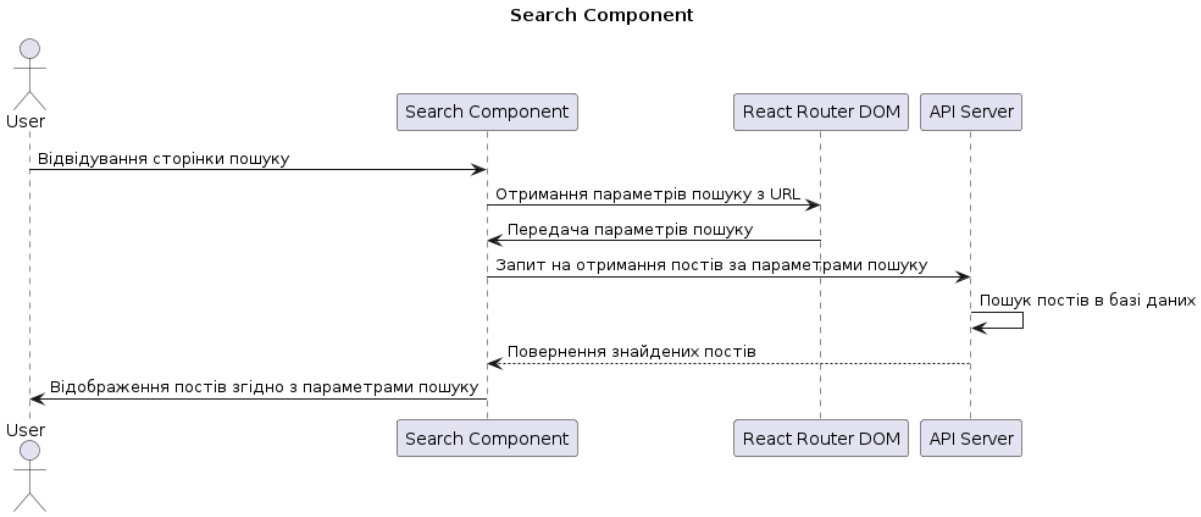

Рисунок 2.4 – Алгоритм компоненту пошуку постів

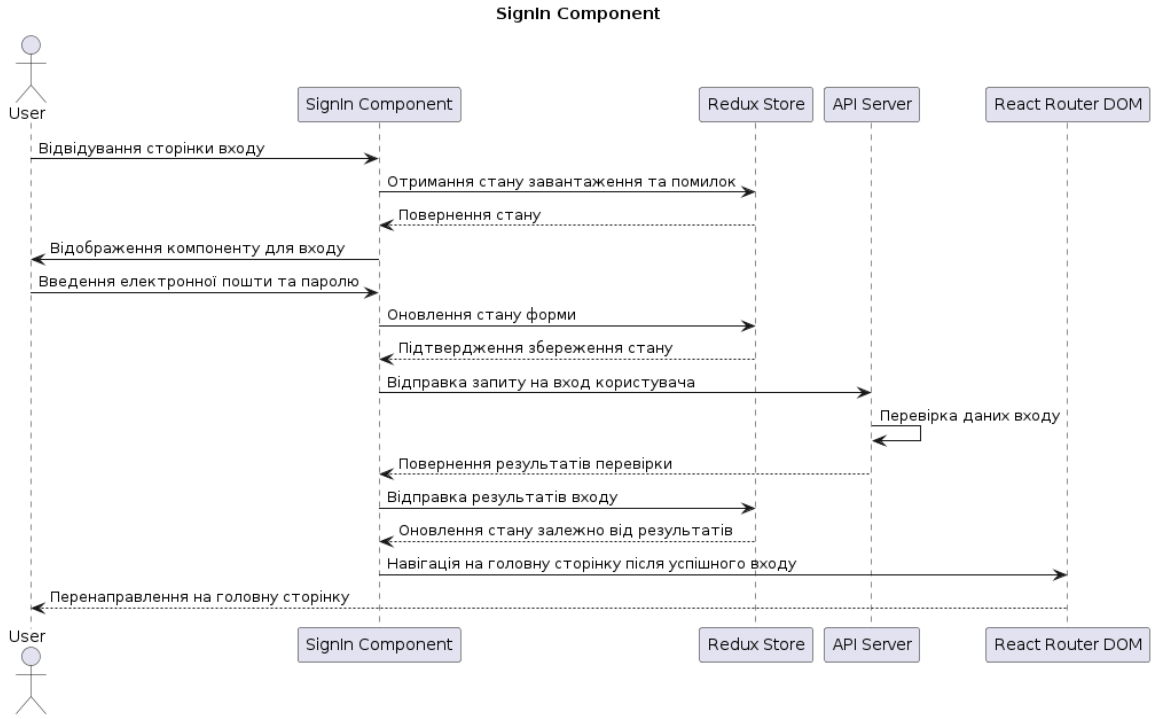

Рисунок 2.5 – Алгоритм компоненту авторизації

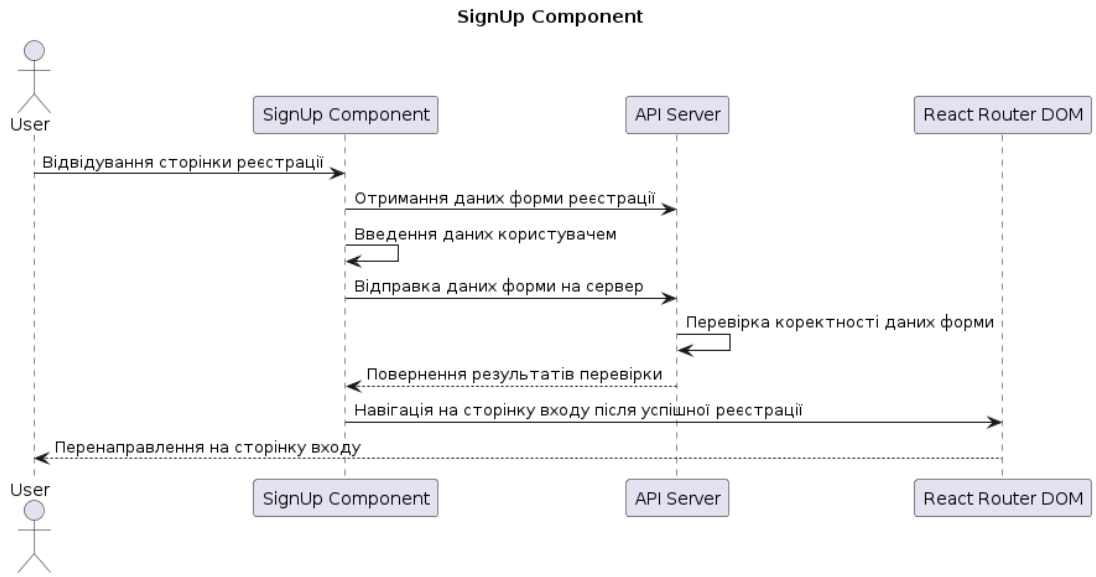

Рисунок 2.6 – Алгоритм компоненту реєстрації

2.4 Висновки до розділу 2

Проєктування багатокористувальницької системи блогінгу є важливим етапом створення системи. Було проведено детальний аналіз, в якому було розглянуто ключові аспекти архітектури, інформаційної моделі та моделювання системи.

На основі виконаних робіт можн зробити наступні висновки:

— Розробка архітектури системи дозволяє чітко визначити структуру і взаємодію між різними компонентами, що забезпечує гнучкість та масштабованість системи;

— розробка інформаційної моделі сприяє ефективному управлінню даними, дозволяючи зберігати, організовувати та обробляти інформацію;

— моделювання системи забезпечує спрощення написання коду, за рахунок створених моделей алгоритмів.

Завдяки етапу проєктування вдалося створити фундамент для багатокористувальницької системи блогінгу, що відповідає сучасним вимогам щодо продуктивності та надійності використання.

3 РОЗРОБКА БАГАТОКОРИСТУВАЛЬНИЦЬКОЇ СИСТЕМИ БЛОГІНГУ

3.1 Розробка баз даних

Використання баз даних є важливим елементом створення багатокористувальницької системи блогінгу з кількох причин:

— бази даних дозволяють зберігати структуровані дані, які потрібні для нашої системи;

— бази даних можуть масштабуватися для обробки великих обсягів даних та великої кількості одночасних запитів веб-застосунків;

— бази даних забезпечують механізми для збереження цілісності даних, це дозволяє уникнути втрати даних або некоректного збереження даних;

— використання баз даних дозволяє ефективно отримувати доступ до необхідних даних у вашому веб-застосунку, що робить роботу з даними більш ефективною та швидкою.

При розробці системи було використано MongoDB та Firebase.

3.1.1 MongoDB

Для того, щоб створити базу даних MongoDB, нам потрібно зайти на офіційний сайт MongoDB, зайти в свій акаунт, натиснути на кнопку «New project»:

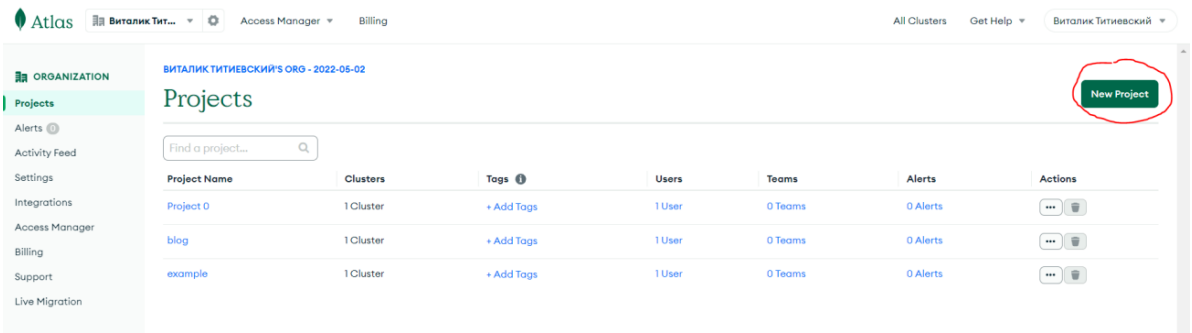

Рисунок 3.1 – Вигляд вкладки зі створенням проєкту MongoDB

Далі потрібно по інструкції виконати всі умови та підключити базу даних до коду:

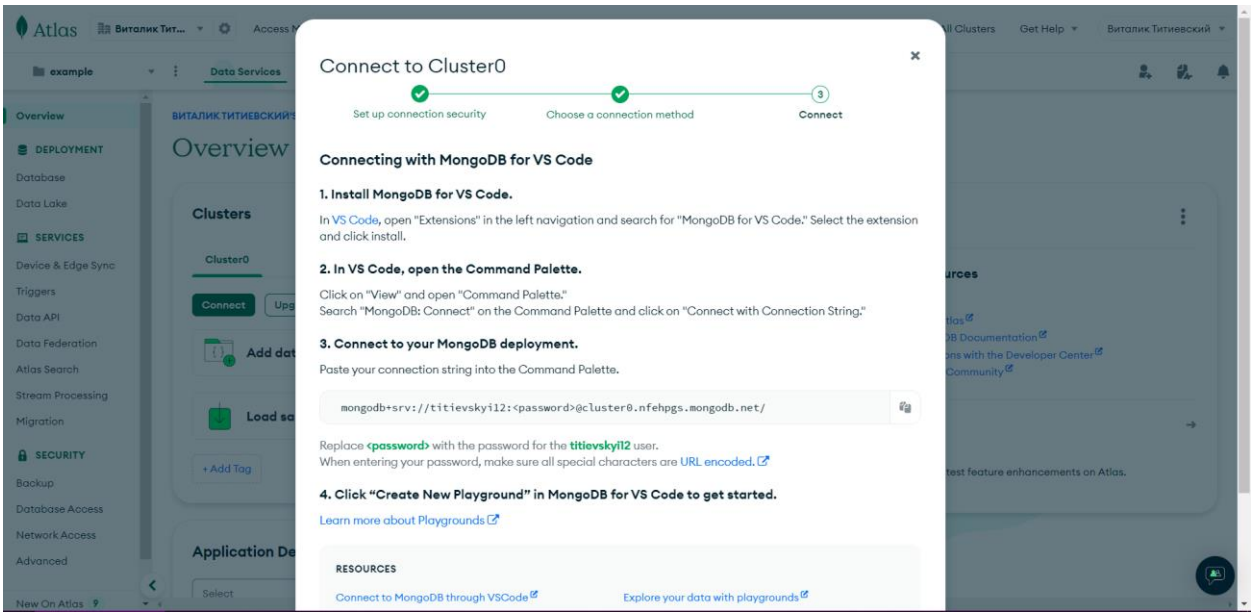

Рисунок 3.2 – Вигляд вкладки з під'єднанням до VS Code MongoDB

Також, у самому середовищі розробки потрібно інсталювати пакет «mongoose», для роботи з MongoDB.

Ось такий вигляд має база даних після створення колекцій на програмному рівні:

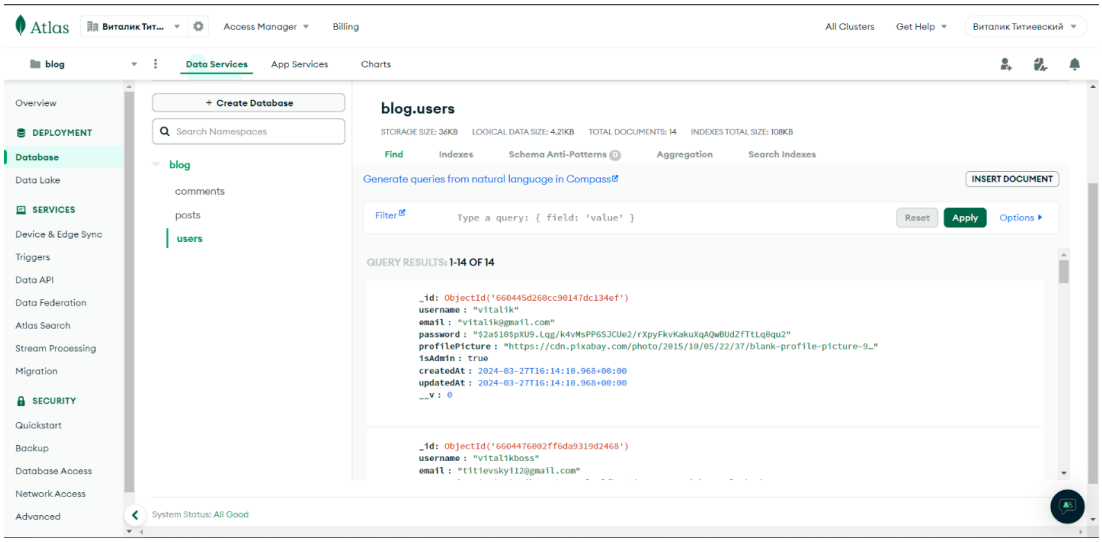

Рисунок 3.3 – Вигляд бази даних MongoDB

3.1.2 Firebase

Для того, щоб створити базу даних Firebase, нам потрібно зайти на офіційний сайт Firebase, зайти в свій акаунт, натиснути на кнопку «Add project»:

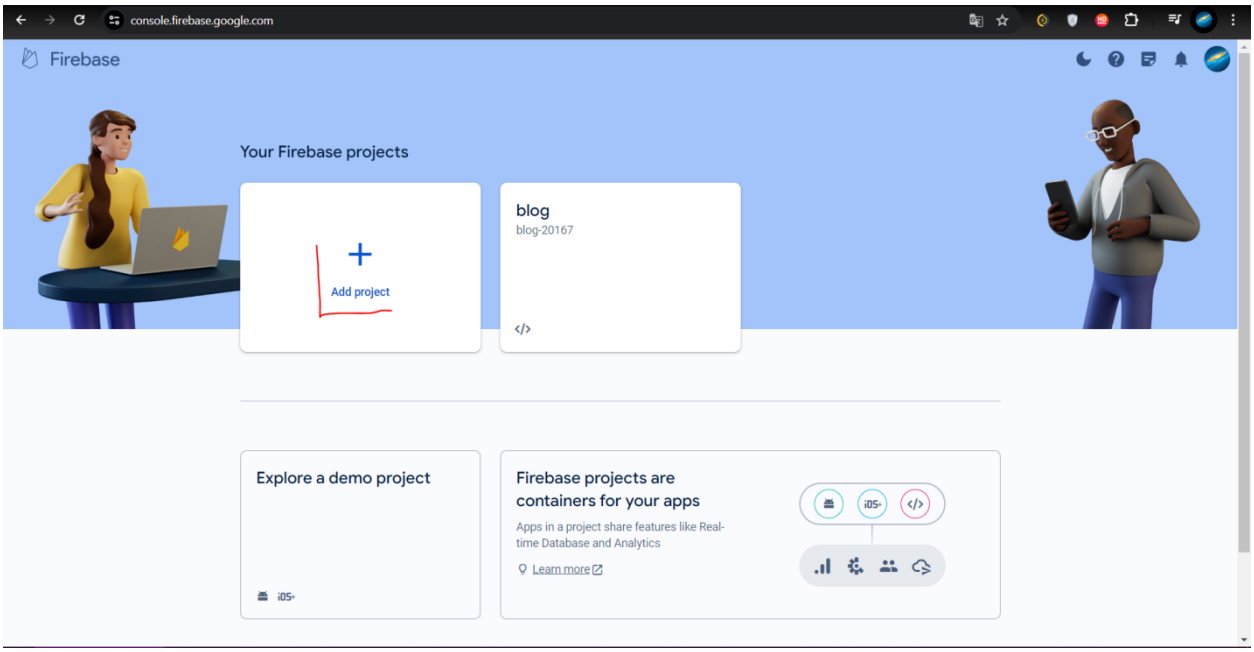

Рисунок 3.4 – Вигляд вкладки зі створенням проєкту Firebase

Після цього виконуємо невелику інструкції від платформи

Також, у самому середовищі розробки потрібно інсталювати пакет «firebase», для роботи з Firebase.

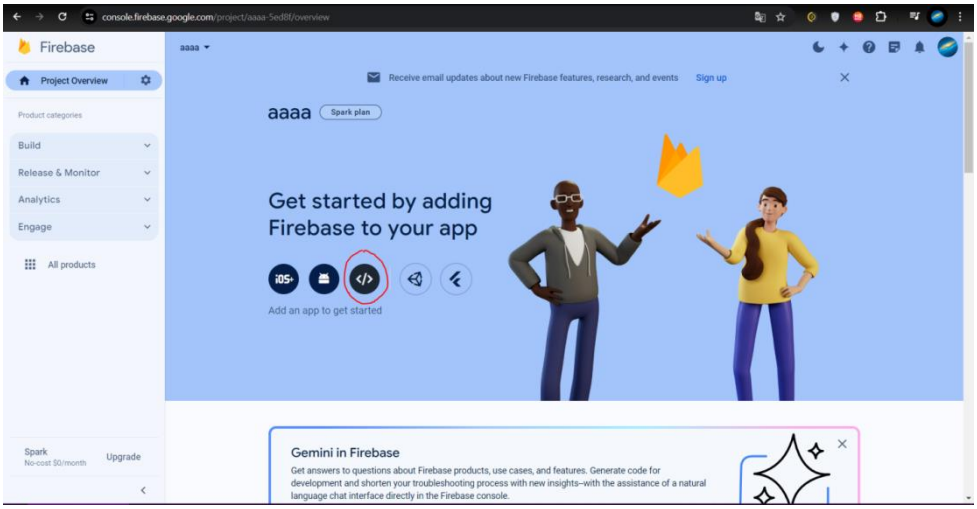

Рисунок 3.5 – Вигляд вкладки зі створення вебзастосунку

Інсталюємо пакет firebase та виконуємо інструкцію:

```
// Import the functions you need from the SDKs you need
import { initializeApp } from "firebase/app";
// TODO: Add SDKs for Firebase products that you want to use
// https://firebase.google.com/docs/web/setup#available-libraries

// Your web app's Firebase configuration
// For Firebase JS SDK v7.20.0 and later, measurementId is optional
const firebaseConfig = {
  apiKey: [redacted],
  authDomain: "blog-20167.firebaseapp.com",
  projectId: "blog-20167",
  storageBucket: "blog-20167.appspot.com",
  messagingSenderId: "834642137705",
  appId: "1:834642137705:web:a825fcb123f9d9a76e88f8",
  measurementId: "G-EBLQ41V83S"
};

// Initialize Firebase
export const app = initializeApp(firebaseConfig);
```

Рисунок 3.6 – Вигляд файлу firebase.js

В цій базі даних будуть зберігатися фото постів:

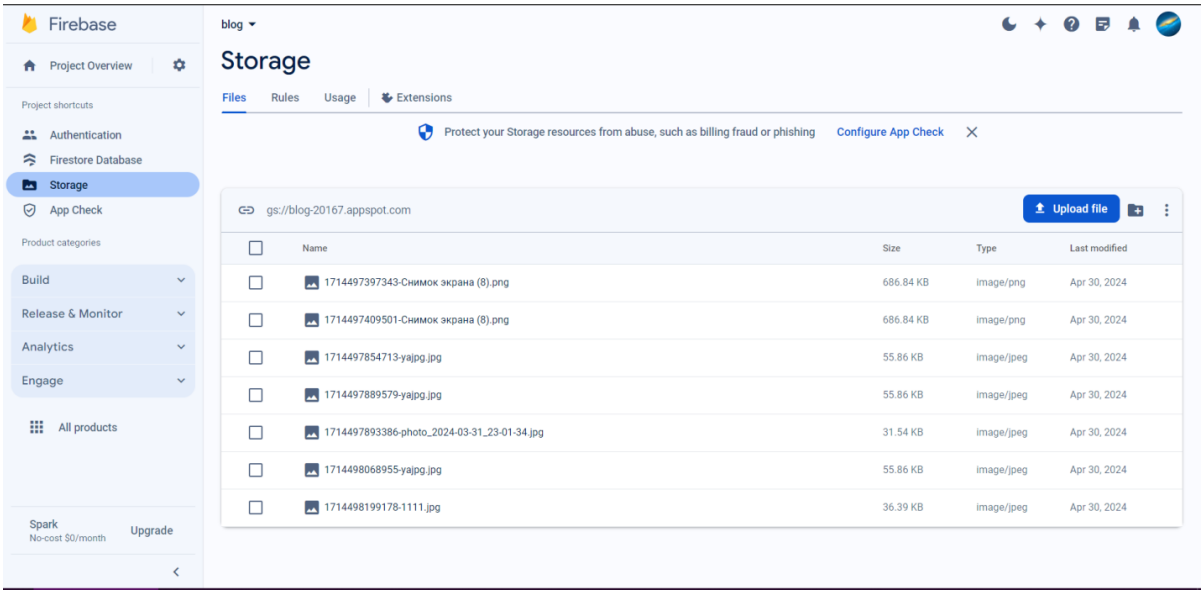

Рисунок 3.7 – Вигляд сховища Firebase

3.2 Розробка клієнтської частини засобами React

Клієнтська частина багатокористувальницької системи блогінгу складається з компонентів, сторінок, редаксу, файлу налаштування стилів, файлу з шляхами та ключового файлу.

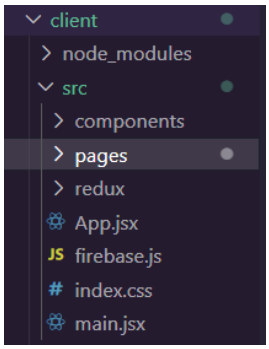

Рисунок 3.8 – Структура папок клієнтської частини

3.2.1 Сторінки

Сторінки визначають окремі сторінки або маршрути в системі. Вони відповідають за відображення конкретного вмісту на певному URL-адресі.

На рис 3.9 продемонстровано список сторінок у багатокористувальницькій системі блогінгу:

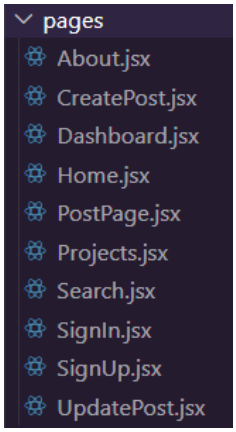

Рисунок 3.9 – Список сторінок

Опис усіх сторінок системи наведений у табл. 3.1:

Таблиця 3.1 – Опис сторінок

Сторінка	Опис
CreatePost	Сторінка зі створенням постів для блогінгу
DashBoard	Сторінка зі збором статистики блогінгу
Home	Основна сторінка зі всіма постами
PostPage	Сторінка поста
Search	Сторінка пошуку постів
SignIn	Сторінка авторизації користувачів
SignUp	Сторінка реєстрації користувачів
UpdatePost	Сторінка оновлення постів
About	Сторінка про мене
Projects	Сторінка проєктів

Отже, було описано 10 сторінок, що будуть допомагати організовувати, навігувати відображати вміст багатокористувальницької системи блогінгу, забезпечуючи кращий досвід користувача.

3.2.2 Компоненти

Компоненти – це основна будівельна одиниця в React. Вони дозволяють розділити інтерфейс користувача на невеликі, повторно використовувані частини, які легко керувати та підтримувати.

Опис усіх компонентів системи наведений у табл. 3.2:

Таблиця 3.2 – Опис компонентів

Компонент	Опис
Header	Компонент відповідає за відображення верхньої панелі навігації. Містить логотип, поле пошуку, кнопку для зміни теми, аватар користувача, кнопку входу/виходу та посилання на різні сторінки системи
Footer	Компонент відображає нижню частину системи. Містить інформацію про блог, посилання на різні сторінки системи, а також іконки для соціальних мереж

Закінчення табл. 3.2

Компонент	Опис
DashboardComp	Компонент відображає статистику на панелі керування для адміністраторів.
DashComments	Компонент призначений для для відображення списку коментарів з можливістю видалення кожного коментаря на панелі керування для адміністраторів
DashPosts	Компонент призначений для для відображення списку постів з можливістю видалення кожного поста на панелі керування для адміністраторів
DashProfile	Компонент призначений для управління профілем користувача. Є можливість завантажити та змінити зображення профілю, оновити інформацію користувача, видалити та вийти з облікового запису
DashSidebar	Компонент відображає бокову панель для керування профілем користувача, постами, користувачами та коментарями
DashUsers	Компонент призначений для для відображення списку користувачів з можливістю видалення кожного користувача на панелі керування для адміністраторів
Comment	Компонент відображає коментарі на сторінці, де кожен коментар має можливість вподобати, редагувати та видаляти його
CommentSection	Компонент відображає секцію коментарів для певного посту
OAuth	Компонент реалізує автентифікацію користувачів за допомогою сервісу Google
OnlyAdminePrivate	Компонент є приватним маршрутом, призначений лише для адміністраторів. Забезпечує захищений доступ до певних частин системи лише для адмністраторів
PostCard	Призначений для відображення окремого поста
PrivateRoute	Компонент призначений для забезпечення доступу до приватних сторінок тільки авторизованим користувачам
ScrollToTop	Компонент призначений для автоматичної прокрутки сторінки вгору, коли користувач перемикається між сторінками
ThemeProvider	Компонент призначений для відображення та зміни стилів сторінки

Отже, було описано 16 компонентів, що допоможуть розширити код системи, забезпечуючи його ефективну розробку.

3.2.3 Редакс

Redux допомогає покращити організацію та керування станом системи, забезпечує простоту взаємодії з даними.

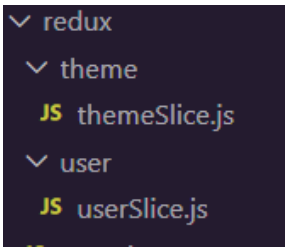

Рисунок 3.10 – Структура папок редаксу

Файл *theme.js* потрібен для управління темою додатку

Файл *userSlice.js* потрібен для створення способу для управління станом користувача в Redux-сторі та обробки різних дій, пов'язаних з користувачем, таких як автентифікація, оновлення даних та видалення користувача.

3.3 Розробка серверної частини з використанням Express.js та Node.js

Серверна частина багатокористувальницької системи блогінгу складається з контролерів, моделей, роутів, утиліт та ключового файлу.

На рис 3.10 продемонстровано список сторінок у багатокористувальницькій системі блогінгу:

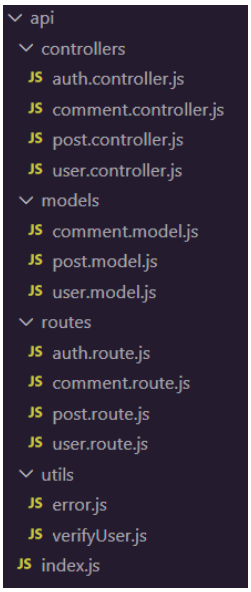

Рисунок 3.10 – Структура папок серверної частини

3.3.1 Контролери

Контролери обробляють запити, які надходять від користувачів, і визначають, які дії потрібно виконати відповідно до цих запитів. Також вони взаємодіють з моделями для отримання або збереження даних та з рендерінгом відповідей для користувачів.

Опис усіх компонентів системи наведений у табл. 3.3:

Таблиця 3.3 – Опис контролерів

Контролер	Опис
auth.controller	Контролер, що відповідає за автентифікаю користувачів
comment.controller	Контролер, що відповідає за взаємодію з коментарями
post.controller	Контролер, що відповідає за взаємодію з постами
user.controller	Контролер, що відповідає за взаємодію з користувачами

Отже, було описано 4 контролери, що допоможуть взаємодіяти з користувачем.

3.3.2 Моделі

Моделі використовуються в контексті баз даних, вони описують структуру даних та визначають спосіб їх зберігання, доступу тощо.

Опис усіх моделей системи наведений у табл. 3.4:

Таблиця 3.4 – Опис моделей

Модель	Опис
comment.model	Модель, що містить поля необхідні для колекції коментарів
post.model	Модель, що містить поля необхідні для колекції постів
user.model	Модель, що містить поля необхідні для колекції користувачів

Отже, були визначені моделі коментарів, постів, користувачів для MongoDB.

3.3.3 Роути

Роути визначають шляхи URL, які користувачі можуть використовувати для взаємодії з системою.

Опис усіх роутів системи наведений у табл. 3.5:

Таблиця 3.5 – Опис роутів

Роут	Опис
auth.route	Визначає маршрути обробки HTTP-запитів для реєстрації, входу та автентифікації користувачів
comment.route	Визначає маршрути та обробники для роботи з коментарями в системі
post.route	Визначає маршрути та обробники для роботи з постами в системі
user.route	Визначає маршрути та обробники для роботи з користувачами в системі

Отже, було описано 4 роути, що знадобляться у подальшій роботі системи.

3.3.4 Утиліти

Утиліти містять допоміжні функції, які зменшують дублювання коду та спрощують керування функціональністю в різних частинах додатку.

Опис усіх утиліт системи наведений у табл. 3.6:

Таблиця 3.6 – Опис утиліт

Утиліта	Опис
error	Утиліта повертає новий об’єкт типу «Error» з властивостями «statusCode» та «message»
verifyUser	Утиліта використовується для перевірки токену доступу в запиті

За допомогою цих утиліт не потрібно постійно писати один і той самий код.

3.4 Висновки до розділу 3

Отже, розробка багатокористувальницької системи блогінгу є найважливішим етапом створення програмного продукту. Було проведено практичну частину, в якій продемонстровано процей розробки системи.

На основі виконаних робіт на даному етапі можна зробити наступні висновки:

— розробка баз даних з використанням MongoDB та Firebase забезпечило гнучкість і масштабованість системи, дозволяючи ефективно зберігати та обробляти великі обсяги даних;

— розробка клієнтської частини з використанням React та Redux сприяло створенню інтерактивних функцій, зручних для користувачів;

— розробка серверної частини з використанням Express.js та Node.js дозволило створити надійні та високопродуктивні серверні рішення, які ефективно обробляють запити користувачів.

Завдяки даному етапу вдалося розробити багатокористувальницьку систему блогінгу, яка відповідає сучасним умовам щодо продуктивності, масштабованості та зручності використання.

4 Тестування та оцінка якості вебзастосунку

4.1 Тестування

Тестування програмного забезпечення – це процес оцінки та перевірки того, що програмний продукт або програма виконує те, що він повинен робити. Переваги якісного тестування включають запобігання помилкам і покращення продуктивності[13].

Одним із основних методів тестування багатокористувальницької системи блогінгу було обране ручне тестування.

Ручне тестування — це процес тестування програмного забезпечення, у якому тестові випадки виконуються вручну без використання будь-яких автоматизованих інструментів. Усі тестові випадки виконуються тестером вручну відповідно до точки зору кінцевого користувача. Це гарантує, чи працює програма, як зазначено в документі вимоги, чи ні. Тестові випадки плануються та впроваджуються для завершення майже 100 відсотків програми. Звіти про тестові випадки також створюються вручну [14].

Також при тестуванні багатокористувальницької системи блогінгу було використано спеціальний інструмент тестування «Selenium IDE» для проведення автотестів.

4.2 Техніки тест-дизайну

Техніки тест-дизайну – це стратегії, які допомагають писати кращі тестові випадки. Переваги використання технік тест-дизайну полягають у можливості створювати менше тестів, забезпечуючи широке покриття вимог [2].

Наприклад, для того, щоб перевірити правильність поведінки роботи системи при введенні різної кількості символів у паролі, доцільно використовувати метод аналізу граничних значень. Якщо у функціональних

вимогах до довжини паролів для автентифікації зазначено межі від 6-ти до 20-ти символів, то варто при тестуванні граничних значень взяти такі значення [2]:

– кількість символів менше граничного значення – 5;

– кількість символів для нижньої межі граничного значення – 6;

– кількість символів для верхньої межі граничного значення – 20;

– кількість символів, що більше граничного значення, – 21.

У табл. 4.1 подано тест-кейси, створені на основі аналізу граничних значень.

Для створення повної й ефективної стратегії тестування варто комбінувати техніки тест-дизайну. Кожна техніка має свої переваги та недоліки, комбінування декількох технік дозволяє отримувати повний опис можливих дефектів [2].

Отже, під час тестування доцільно використовувати техніки тест-дизайну, що дозволяють зменшити процент пропуску дефектів у продакшн.

4.3 Розробка тест-кейсів

Тестовий кейс - це послідовність дій, які виконуються над системою з метою перевірки відповідності вимогам програмного забезпечення та коректності її роботи.Мета цього тестового прикладу полягає у перевірці правильності роботи різних функцій у системі та у підтвердженні відповідності системи всім вимогам, стандартам та інструкціям, встановленим замовником. Використання тестових випадків дозволяє виявляти помилки, що можуть виникнути під час розробки, або дефекти, які можуть бути пропущені під час спеціальних тестів.

Далі наведені тест-кейси для багатокористувальницької системи блогінгу:

Таблиця 4.1 – Тест-кейс перевірки реєстрації користувача

Назва	**Передумови**	**Кроки**	**Очікуваний результат**
Реєстрація користувача з невалідними даними	Не мати існуючого користувача.	1. Зайти на сайт 2. Натиснути на кнопку «Sign in» у правому верхньому куті екрану. 3. Натиснути на кнопку Sign Up під формою авторизації 4. У полі "Your username" ввести «destroyer228». 5. У полі «Your email» ввести "Іванов Іван Іванович". 6. У полі «Your password» ввести "1234". 7. Натиснути на кнопку "Sign up".	Система вкаже на помилку, оскільки кількість символів менше граничного значення (4 символів)
Реєстрація користувача з валідними даними	Не мати існуючого користувача.	1. Зайти на сайт 2. Натиснути на кнопку «Sign in» у правому верхньому куті екрану. 3. Натиснути на кнопку Sign Up під формою авторизації 4. У полі "Your username" ввести «destroyer228». 5. У полі «Your email» ввести "Іванов Іван Іванович". 6. У полі «Your password» ввести "12345". 7. Натиснути на кнопку "Sign up".	Система дозволить продовжити реєстрацію, оскільки кількість символів еквівалентне нижній межі граничного значення (5 символів)
Реєстрація користувача з валідними даними	Не мати існуючого користувача.	1. Зайти на сайт 2. Натиснути на кнопку «Sign in» у правому верхньому куті екрану. 3. Натиснути на кнопку Sign Up під формою авторизації 4. У полі "Your username" ввести «destroyer228». 5. У полі «Your email» ввести "Іванов Іван Іванович". 6. У полі «Your password» ввести "123456789012345678 90". 7. Натиснути на кнопку "Sign up".	Система дозволить продовжити реєстрацію, оскільки кількість символів еквівалентне верхній межі граничного значення (20 символів)
Реєстрація користувача з невалідними даними	Не мати існуючого користувача.	1. Зайти на сайт 2. Натиснути на кнопку «Sign in» у правому верхньому куті екрану. 3. Натиснути на кнопку Sign Up під формою авторизації 4. У полі "Your username" ввести «destroyer228». 5. У полі «Your email» ввести "Іванов Іван Іванович". 6. У полі «Your password» ввести "123456789012345678 901". 7.Натиснути на кнопку "Sign up".	Система вкаже на помилку, оскільки кількість символів більше граничного значення (21 символ)

Таблиця 4.3 – Тест-кейс перевірки авторизації користувача

Назва	**Передумова**	**Кроки**	**Очікуваний результат**
Авторизація користувача	Мати існуючого користувача	1. Зайти на сайт 2. Натиснути на кнопку "Sign In" у правому верхньому куті екрану 3. В поле "Your email" ввести "test@gmail.com" 4. В поле "Your password" ввести "test123" 5. Натиснути на кнопку "Sign In"	Реєстрація успішна. Сторінка реєстрації переадресовується на сторінку авторизації

Таблиця 4.4 – Тест-кейс перевірки зміни паролю

Назва	**Передумова**	**Кроки**	**Очікуваний результат**
Зміна паролю	Мати вже раніше зареєстрованого з такими Тестовими даними. Логін — “test1@gmail.com”, пароль — “test1”	1. Натиснути на іконку "чоловічка" у правому верхньому куті екрану 2. Натиснути на пункт «Profile» 3. У полі “password” ввести “test12” 4. Натиснути на кнопку «Update»	Система змінить пароль і повідомить «User's profile updated successfully»

Таблиця 4.5 – Тест-кейси перевірки пошуку постів

Назва	**Передумова**	**Кроки**	**Очікуваний результат**
Пошук поста зі словом «birthday» у пості	Мати вже раніше зареєстрованого з такими Тестовими даними. Логін — “test1@gmail.com”, пароль — “test1”	1. Зайти на сайт 2. Натиснути на поле для вводу «Search» 3. Вписати слово «birthday» 4. Натиснути Enter	Відкриється сторінка з постами, де є слово «birthday»
Пошук поста зі словом «weekend» у пості	Мати вже раніше зареєстрованого з такими Тестовими даними. Логін — “test1@gmail.com”, пароль — “test1”	1. Зайти на сайт 2. Натиснути на поле для вводу «Search» 3. Вписати слово «birthday» 4. Натиснути Enter	Відкриється сторінка з постами, де є слово «weekend»
Пошук поста зі словом «family» у пості	Мати вже раніше зареєстрованого з такими Тестовими даними. Логін — “test1@gmail.com”, пароль — “test1”	1. Зайти на сайт 2. Натиснути на поле для вводу «Search» 3. Вписати слово «family» 4. Натиснути Enter	Відкриється сторінка з постами, де є слово «family»

Таблиця 4.6 – Тест-кейс перевірки додавання коментаря

Назва	**Передумова**	**Кроки**	**Очікуваний результат**
Додавання коментаря під постом	Мати вже раніше зареєстрованого з такими Тестовими даними. Логін — “test@gmail.com”, пароль — “test123”	1. Зайти на сайт 2. Натиснути на перший пост 3. Перейти до коментарів 4. Під постом натиснути на поле «Add a comment…» 5. Написати «Such a wonderful post!» 6. Натиснути на кнопку "Submit"	Коментар успішно доданий.

Таблиця 4.7 – Тест-кейс перевірки додавання лайку до коментаря

Назва	**Передумова**	**Кроки**	**Очікуваний результат**
Додавання лайку до коментаря	Мати вже раніше зареєстрованого з такими Тестовими даними. Логін — “test@gmail.com”, пароль — “test123”	1. Зайти на сайт 2. Натиснути на перший пост 3. Перейти до коментарів 4. Натиснути на кнопку лайку під першим коментарем	Лайк до коментарю успішно доданий

Таблиця 4.8 – Тест-кейс перевірки редагування коментаря

Назва	**Передумова**	**Кроки**	**Очікуваний результат**
Додавання лайку до коментаря	Мати вже раніше зареєстрованого з такими Тестовими даними. Логін — “test@gmail.com”, пароль — “test123”	1. Зайти на сайт 2. Натиснути на перший пост 3. Перейти до коментарів 4. Натиснути на кнопку «Edit» під першим коментарем 5. Написати «What a great post!» 6. Натиснути на кнопку "Save"	Коментар успішно відредагований

Тестовий-кейс це важливий компонент документації тестування програмного забезпечення. Цей документ містить детальний опис кроків, конкретних умов і параметрів, необхідних для відповідної перевірки реалізації функції, що тестується, або її частин [15].

4.4 Автоматизоване тестування

Автоматизоване тестування — це впровадження інструменту автоматизації для виконання тестів. При автоматизованому тестуванні тестувальник створює

тестові сценарії, які можуть включати в себе введення даних, натискання кнопок, перевірку результатів тощо. Після цього тестові сценарії запускаються автоматично за допомогою спеціальних інструментів тестування [16].

Selenium є одним з найпопулярніших інструментів для автоматизованого тестування вебсайтів. Це відкрите програмне забезпечення, що дозволяє взаємодіяти з вебсайтами, що використовують різні браузери, імітуючи поведінку користувача [17].

Для багатокористувальницької системи блогінгу за допомогою автоматизованих тестів було виконано тестування роботи додавання коментаря за таким протоколом:

— відкрити сайт;
— натиснути на перший пост;
— перейти до коментарів;
— вписати коментар «Such a wonderful post!»;
— натиснути на кнопку «Submit».

На рис. 4.1 продемонстровано результати автотесту:

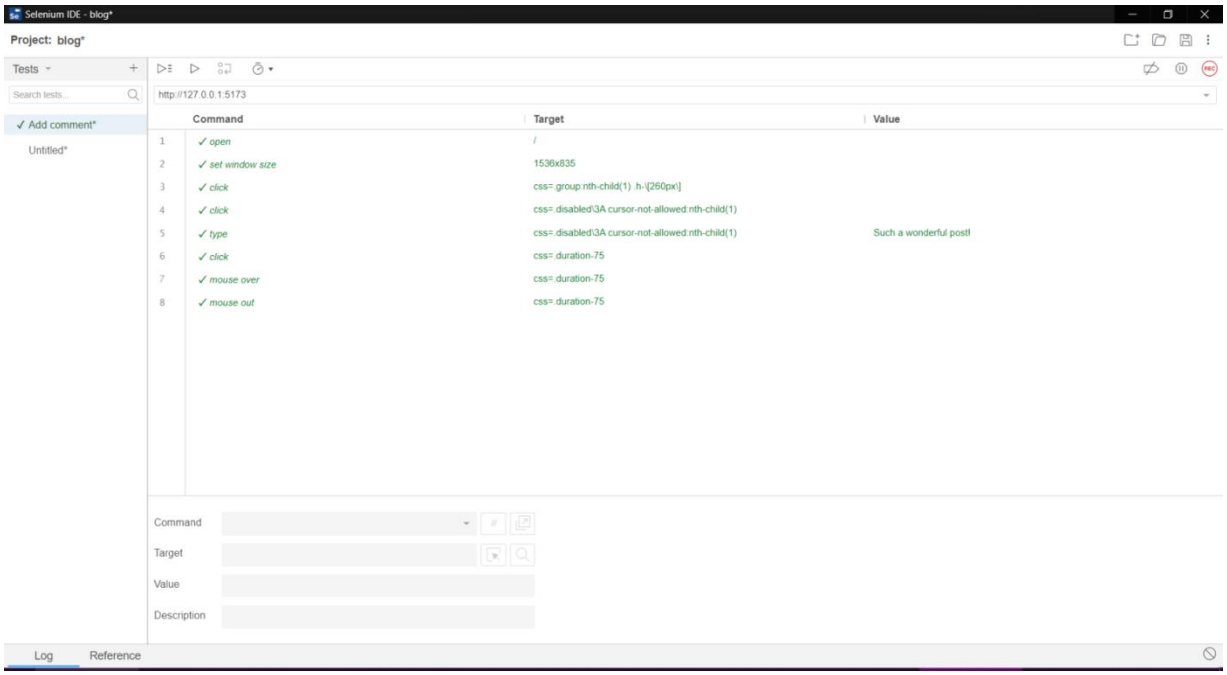

Рисунок 4.1 – Проходження тесту додавання коментаря

Також за допомогою Silenium IDE було проведено тестування редагування коментарів за таким протоколом:

— відкрити сайт;

— натиснути на перший пост;

— перейти до коментарів;

— натиснути на кнопку «Edit».

— вписати коментар «What a great post!»;

— натиснути на кнопку «Save».

На рис. 4.2 продемонстровано результати автотесту:

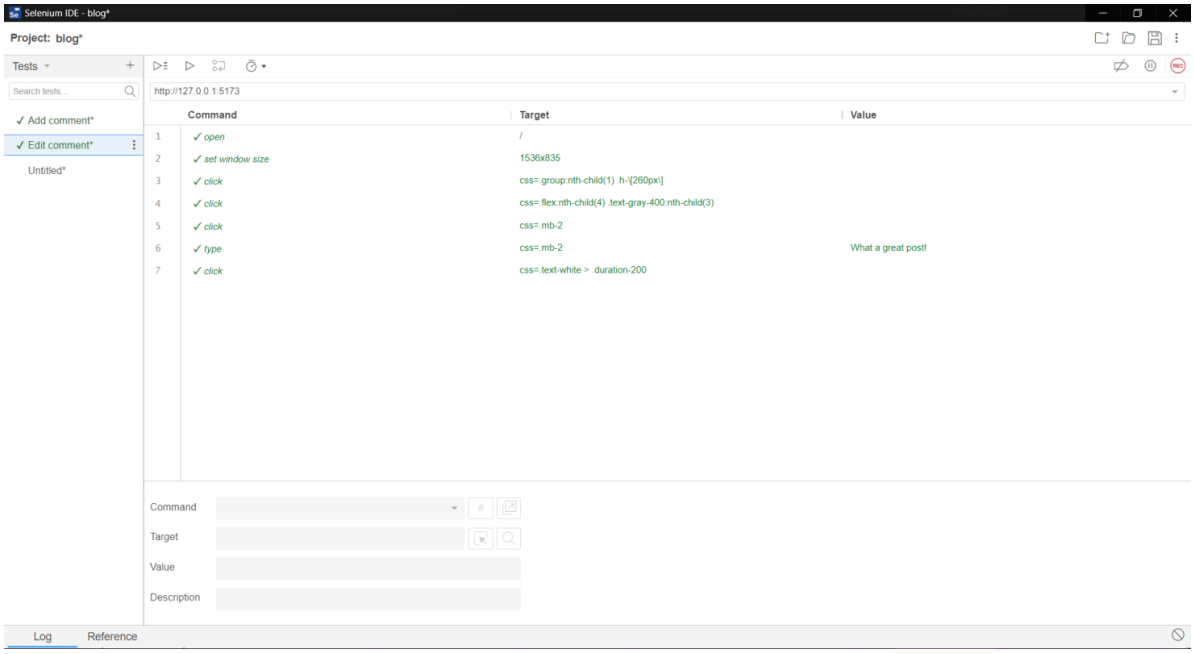

Рисунок 4.2 – Проходження тесту редагування коментаря

Тестування за допомогою «Selenium IDE» значно спрощує перевірку функціональності багатокористувальницької системи блогінгу.

4.5 Тестування адаптивності

Користувачі використовують різні типи пристроїв, такі як: смартфони, планшети, десктопні комп’ютери, ноутбуки тощо. На різних пристроях вебзастосунок демонструється по-різному, тому важливо робити адаптивний дизайн, щоб на різних пристроях були доступні всі функції для користувача. Якщо користувач не зможе використати якусь функцію – він перейде до іншого вебзастосунку, що сприяє зниженню попиту на продукт.

Тому було вирішено провести тестування адаптивності.

Тестування адаптивності було вирішено провести за допомогою Chrome DevTools.

Chrome DevTools – це набір інструментів веб-розробника, вбудованих безпосередньо у браузер Google Chrome. DevTools дозволяє оперативно редагувати сторінки та швидко діагностувати проблеми, що допомагає швидше створювати якісніші веб-сайти [18].

Chrome DevTools – це корисний спосіб перевірити як система адаптується до різних розмірів екранів і пристроїв.

На рис. 4.3 продемонстровано вигляд системи блогінгу в екрані телефона «Iphone XR»:

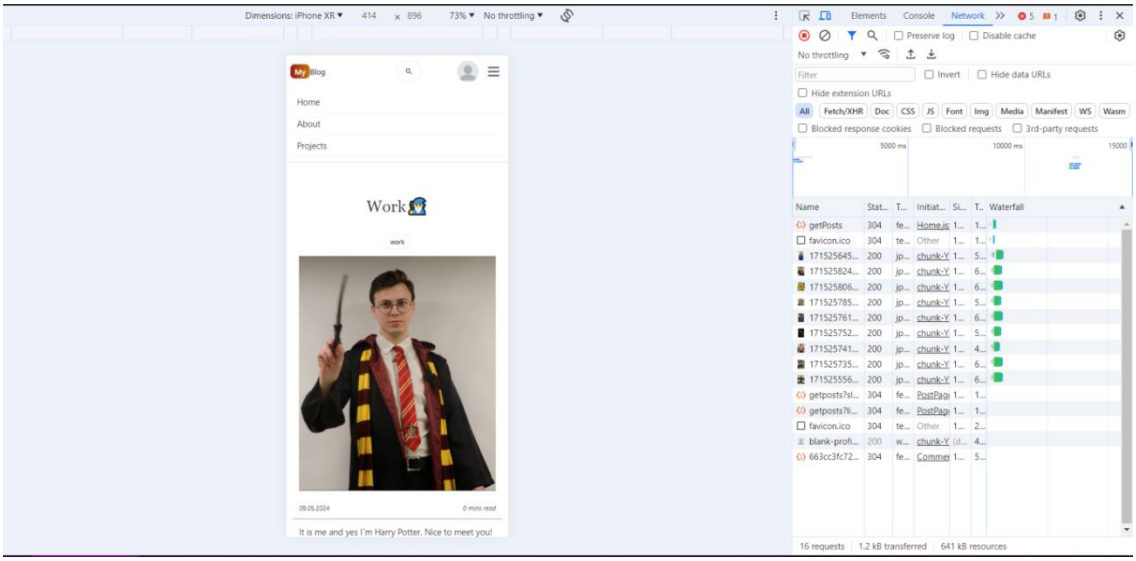

Рисунок 4.3 – Вигляд вебзастосунку

На рис. 4.4 продемонстровано вигляд системи блогінгу в екрані планшета «Ipad Air»:

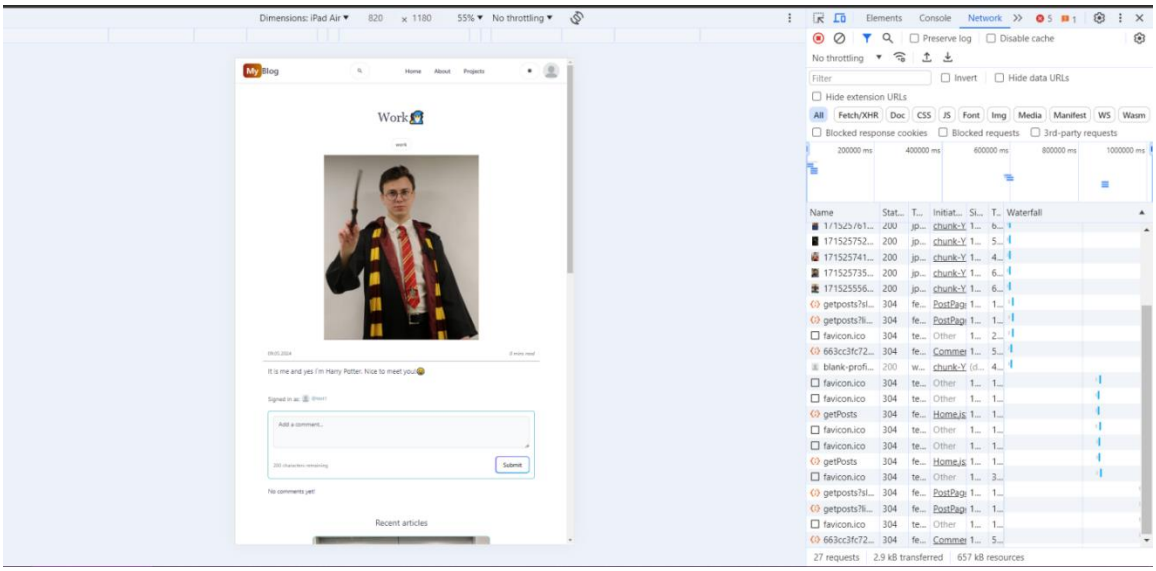

Рисунок 4.4 – Вигляд вебзастосунку

Таким чином, за допомогою DevTools було проведено тестування вебзастосунку на адаптивність до різних розмірів екранів.

4.6 Розробка чек-ліста

Чек-ліст – це список завдань або елементів, які має виконати тестувальник, щоб переконатися, що програмне забезпечення перевірено ретельно[19].

При проходженні чекліста тестувальник зазначає статус навпроти кожного пункту, який протестовано. Можливі такі варіанти статусів[20]:

— «Passed» – перевірка пройдена успішно, багів не знайдено;

— «Failed» – знайдений один або більше багів;

— «Blocked» – неможливо перевірити, тому що один з багів блокує поточну перевірку

— «In Progress» – поточний пункт, над яким працює тестувальник;

— «Not run» – ще не перевірено;

— «Skipped» – пункт перевірятися не буде з певної причини. Наприклад, поточний функціонал ще не реалізований.

Таблиця 4.9 – Вигляд чекліста

№	Назва	Статус	Коментар
1	Реєстрація нового користувача з валідними даними	PASS	
2	Реєстрація нового користувача з невалідними даними	PASS	Система не дала змогу зареєструвати користувача з невалідними даними
3	Авторизація користувача	PASS	
4	Зміна паролю	PASS	
5	Пошук поста зі словом «birthday» у пості	PASS	
6	Пошук поста зі словом «family» у пості	PASS	
7	Додавання коментаря до поста	PASS	
8	Додавання лайку до коментаря під постом	PASS	

4.7 Висновки до розділу 4

Отже, тестування багатокористувальницької системи блогінгу є важливим етапом розробки програмного продукту. Була проведена практична частина, в якій був продемонстрований процес тестування багатокористувальницької системи блогінгу.

На основі проведеного тестування можна зробити наступні висновки:

— тестування адаптивності дозволяє забезпечити якісне відображення інформації на різних пристроях;

— автоматизовані тести дозволяє ефективно проводити тестування при швидкому внесенні змін в продукт;

— використання тест-кейсів та чеклістів допомагає охопити всі можливі сценарії роботи вебсайту та виявити проблеми з відображенням, функціоналом та безпекою.

За допомогою тестування вдалося забезпечити якість та надійність багатокористувальницької системи блогінгу.

ВИСНОВКИ

Метою цієї роботи було дослідження та реалізація багатокористувальницької системи блогінгу з використанням сучасних технологій і підходів до розробки програмного забезпечення. Для досягнення цієї мети було обрано стек технологій MERN (MongoDB, Express.js, React, Node.js).

На етапі аналізу предметної області та вибору засобів розробки було проведено аналіз специфіки розробки програмних систем і наявних аналогів, що дозволило виявити переваги та недоліки існуючих рішень. Було обрано оптимальні засоби для розробки бази даних (MongoDB та Firebase), клієнтської частини (React та Redux) та серверної частини (Express.js і Node.js). Це дало змогу забезпечити надійне зберігання даних, створити динамічний інтерфейс користувача та реалізувати ефективну обробку серверних запитів.

На етапі проєктування системи було розроблено архітектуру, інформаційну модель та здійснено моделювання системи. Це дозволило чітко визначити структуру та взаємодію компонентів, що забезпечує гнучкість, масштабованість і підтримуваність системи. Інформаційна модель забезпечила ефективне управління даними, а моделювання дозволило попередньо розробити алгоритми для подальшої роботи.

На етапі розробки системи було реалізовано базу даних, клієнтську та серверну частини. Створено структуру баз даних для надійного зберігання та обробки інформації. Використано React для розробки інтерактивного та динамічного інтерфейсу користувача, що сприяє зручності та задоволенню користувацьких потреб. Серверна частина, розроблена за допомогою Express.js і Node.js, забезпечує стабільну роботу та ефективну обробку запитів.

Таким чином, поставлені завдання були успішно виконані. Створена багатокористувальницька система блогінгу відповідає сучасним вимогам щодо продуктивності, масштабованості та зручності використання. Вона є актуальною для особистих та комерційних проєктів, забезпечуючи платформу для спілкування з аудиторією та обміну контентом. Система відкриває нові можливості для

взаємодії та обміну інформацією у всесвітній павутині, відповідаючи сучасним потребам користувачів в онлайн-середовищі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Тітієвський В.О. Специфіка розробки бекенду засобами Express.JS. *Кібербезпека в сучасному світі:актуальні виклики*: матер. III Всеукраїнської наук.-практ. конф. (м. Одеса, 25 листопада 2022 р.) Одеса: Видавничий дім «Гельветика», 2022. С. 112-115.
2. Тітієвський В.О. Техніки тест-дизайну. *Українське суспільство та наука в умовах воєнного стану й євроінтеграційних перетворень: виклики, напрями відновлення і розвитку*: у 2 т. : матер. Всеукраїнської наук.-практ. конф. здобувачів вищої освіти (м. Одеса, 20 травня 2023 р). Одеса: Видавництво «Юридика», 2023. Т. 2. С. 473-475.
3. Рейтинг мов програмування 2024. TypeScript в трійці лідерів, Python з`являється у всіх нішах, а Rust – улюблена мова. URL: https://dou.ua/lenta/articles/language-rating-2024/
4. SWOT Analysis: How to with table and example. URL: https://www.investopedia.com/terms/s/swot.asp
5. What is MongoDB? URL: https://www.mongodb.com/company/what-is-mongodb
6. Плюси і мінуси Firebase. URL: https://uk.goodteam.dev/blog/plyusy-i-minusy-firebase
7. Що таке React JS? Як почати вивчати Реакт? Навички для react developer. URL: https://cases.media/en/article/sho-take-react-js-yak-pochati-vivchati-reakt-navichki-dlya-react-developer
8. Redux. URL: https://redux.js.org/
9. Introduction to Tailwind CSS. URL: https://www.geeksforgeeks.org/introduction-to-tailwind-css/
10. Express. URL: https://expressjs.com/
11. What is JSON Web Token? URL: https://jwt.io/introduction
12. SPA (Single-page application). URL: https://developer.mozilla.org/en-US/docs/Glossary/SPA

13. What is software testing? URL: https://www.ibm.com/topics/software-testing
14. Manual testing. URL: https://www.javatpoint.com/manual-testing#
15. How to write test cases for software testing: a complete guide. URL: https://luxequality.com/blog/how-to-write-test-cases-for-software-testing/
16. Automated software testing guide. URL: https://smartbear.com/learn/automated-testing/what-is-automated-testing/
17. What is Selenium IDE? URL: https://www.browserstack.com/guide/what-is-selenium-ide#
18. Інструменти розробника. URL: https://developer.chrome.com/docs/devtools
19. Checklist VS Test Case set. URL: https://medium.com/@case_lab/checklist-vs-test-case-set-3b304aed7a2d
20. Що таке чеклісти та як з ними працювати. URL: https://training.qatestlab.com/blog/technical-articles/work-with-checklist/

ДОДАТКИ

Додаток А. ПРОГРАМНИЙ КОД

Серверна частина

Файл auth.controller.js

```
import User from '../models/user.model.js';
import bcryptjs from 'bcryptjs';
import { errorHandler } from '../utils/error.js';
import jwt from 'jsonwebtoken';

export const signup = async (req, res, next) => {
  const { username, email, password } = req.body;

  if (!username || !email || !password || username === '' || email === '' || password === '' || password.length < 5 || password.length > 20) {
    return next(errorHandler(400, 'All fields are required and password should be between 5 and 20 characters long'));
  }

  const hashedPassword = bcryptjs.hashSync(password, 10);

  const newUser = new User({
    username,
    email,
    password: hashedPassword,
  });

  try {
    await newUser.save();
    res.json('Signup successful');
  } catch (error) {
    next(error);
  }
};

export const signin = async (req, res, next) => {
  const { email, password } = req.body;

  if (!email || !password || email === '' || password === '') {
    return next(errorHandler(400, 'All fields are required'));
  }

  try {
    const validUser = await User.findOne({ email });
    if (!validUser) {
```

```
          return next(errorHandler(404, 'User not found'));
        }
        const    validPassword    =    bcryptjs.compareSync(password,
validUser.password);
        if (!validPassword) {
          return next(errorHandler(400, 'Invalid password'));
        }
        const token = jwt.sign(
          { id: validUser._id, isAdmin: validUser.isAdmin },
          process.env.JWT_SECRET
        );

        const { password: pass, ...rest } = validUser._doc;

        res
          .status(200)
          .cookie('access_token', token, {
            httpOnly: true,
          })
          .json(rest);
      } catch (error) {
        next(error);
      }
    };

    export const google = async (req, res, next) => {
      const { email, name, googlePhotoUrl } = req.body;
      try {
        const user = await User.findOne({ email });
        if (user) {
          const token = jwt.sign(
            { id: user._id, isAdmin: user.isAdmin },
            process.env.JWT_SECRET
          );
          const { password, ...rest } = user._doc;
          res
            .status(200)
            .cookie('access_token', token, {
              httpOnly: true,
            })
            .json(rest);
        } else {
          const generatedPassword =
            Math.random().toString(36).slice(-8) +
            Math.random().toString(36).slice(-8);
          const                    hashedPassword                    =
bcryptjs.hashSync(generatedPassword, 10);
          const newUser = new User({
            username:
              name.toLowerCase().split(' ').join('') +
              Math.random().toString(9).slice(-4),
            email,
            password: hashedPassword,
```

```
            profilePicture: googlePhotoUrl,
          });
          await newUser.save();
          const token = jwt.sign(
            { id: newUser._id, isAdmin: newUser.isAdmin },
            process.env.JWT_SECRET
          );
          const { password, ...rest } = newUser._doc;
          res
            .status(200)
            .cookie('access_token', token, {
              httpOnly: true,
            })
            .json(rest);
        }
      } catch (error) {
        next(error);
      }
    };
```

Файл comment.controller.js

```
import Comment from '../models/comment.model.js';

export const createComment = async (req, res, next) => {
  try {
    const { content, postId, userId } = req.body;

    if (userId !== req.user.id) {
      return next(
        errorHandler(403, 'You are not allowed to create this
comment')
      );
    }

    const newComment = new Comment({
      content,
      postId,
      userId,
    });
    await newComment.save();

    res.status(200).json(newComment);
  } catch (error) {
    next(error);
  }
};

export const getPostCom = async (req, res, next) => {
  try {
    const comments = await Comment.find({ postId: req.params.postId
}).sort({
```

```
      createdAt: -1,
    });
    res.status(200).json(comments);
  } catch (error) {
    next(error);
  }
};

export const likeComment = async (req, res, next) => {
  try {
    const comment = await Comment.findById(req.params.commentId);
    if (!comment) {
      return next(errorHandler(404, 'Comment not found'));
    }
    const userIndex = comment.likes.indexOf(req.user.id);
    if (userIndex === -1) {
      comment.numberOfLikes += 1;
      comment.likes.push(req.user.id);
    } else {
      comment.numberOfLikes -= 1;
      comment.likes.splice(userIndex, 1);
    }
    await comment.save();
    res.status(200).json(comment);
  } catch (error) {
    next(error);
  }
};

export const editComment = async (req, res, next) => {
  try {
    const comment = await Comment.findById(req.params.commentId);
    if (!comment) {
      return next(errorHandler(404, 'Comment not found'));
    }
    if (comment.userId !== req.user.id && !req.user.isAdmin) {
      return next(
        errorHandler(403, 'You are not allowed to edit this
comment')
      );
    }

    const editedComment = await Comment.findByIdAndUpdate(
      req.params.commentId,
      {
        content: req.body.content,
      },
      { new: true }
    );
    res.status(200).json(editedComment);
  } catch (error) {
    next(error);
  }
```

```
};

export const deleteComment = async (req, res, next) => {
  try {
    const comment = await Comment.findById(req.params.commentId);
    if (!comment) {
      return next(errorHandler(404, 'Comment not found'));
    }
    if (comment.userId !== req.user.id && !req.user.isAdmin) {
      return next(
        errorHandler(403, 'You are not allowed to delete this
comment')
      );
    }
    await Comment.findByIdAndDelete(req.params.commentId);
    res.status(200).json('Comment has been deleted');
  } catch (error) {
    next(error);
  }
};

export const getcomments = async (req, res, next) => {
  if (!req.user.isAdmin)
    return next(errorHandler(403, 'You are not allowed to get all
comments'));
  try {
    const startIndex = parseInt(req.query.startIndex) || 0;
    const limit = parseInt(req.query.limit) || 9;
    const sortDirection = req.query.sort === 'desc' ? -1 : 1;
    const comments = await Comment.find()
      .sort({ createdAt: sortDirection })
      .skip(startIndex)
      .limit(limit);
    const totalComments = await Comment.countDocuments();
    const now = new Date();
    const oneMonthAgo = new Date(
      now.getFullYear(),
      now.getMonth() - 1,
      now.getDate()
    );
    const lastMonthComments = await Comment.countDocuments({
      createdAt: { $gte: oneMonthAgo },
    });
    res.status(200).json({ comments, totalComments,
lastMonthComments });
  } catch (error) {
    next(error);
  }
};
```

Файл post.controller.js

```
import Post from '../models/post.model.js';
```

```
import { errorHandler } from '../utils/error.js';

export const create = async (req, res, next) => {
  if (!req.user.isAdmin) {
    return next(errorHandler(403, 'You are not allowed to create a
post'));
  }
  if (!req.body.title || !req.body.content) {
    return next(errorHandler(400, 'Please provide all required
fields'));
  }
  const slug = req.body.title
    .split(' ')
    .join('-')
    .toLowerCase()
    .replace(/[^a-zA-Z0-9-]/g, '');
  const newPost = new Post({
    ...req.body,
    slug,
    userId: req.user.id,
  });
  try {
    const savedPost = await newPost.save();
    res.status(201).json(savedPost);
  } catch (error) {
    next(error);
  }
};

export const getposts = async (req, res, next) => {
  try {
    const startIndex = parseInt(req.query.startIndex) || 0;
    const limit = parseInt(req.query.limit) || 9;
    const sortDirection = req.query.order === 'asc' ? 1 : -1;
    const queryFilters = {
      ...(req.query.userId && { userId: req.query.userId }),
      ...(req.query.category && { category: req.query.category }),
      ...(req.query.slug && { slug: req.query.slug }),
      ...(req.query.postId && { _id: req.query.postId }),
      ...(req.query.searchTerm && {
        $or: [
          { title: { $regex: req.query.searchTerm, $options: 'i' }
},
          { content: { $regex: req.query.searchTerm, $options: 'i' }
},
        ],
      }),
    };
    const posts = await Post.find(queryFilters)
      .sort({ updatedAt: sortDirection })
      .skip(startIndex)
      .limit(limit);
```

```
    const totalPosts = await Post.countDocuments(queryFilters);

    const now = new Date();

    const oneMonthAgo = new Date(
      now.getFullYear(),
      now.getMonth() - 1,
      now.getDate()
    );

    const lastMonthPosts = await Post.countDocuments({
      createdAt: { $gte: oneMonthAgo },
      ...queryFilters,
    });

    res.status(200).json({
      posts,
      totalPosts,
      lastMonthPosts,
    });
  } catch (error) {
    next(error);
  }
};

export const deletepost = async (req, res, next) => {
  if (!req.user.isAdmin || req.user.id !== req.params.userId) {
    return next(errorHandler(403, 'You are not allowed to delete
this post'));
  }
  try {
    await Post.findByIdAndDelete(req.params.postId);
    res.status(200).json('The post has been deleted');
  } catch (error) {
    next(error);
  }
};

export const updatepost = async (req, res, next) => {
  if (!req.user.isAdmin || req.user.id !== req.params.userId) {
    return next(errorHandler(403, 'You are not allowed to update
this post'));
  }
  try {
    const updatedPost = await Post.findByIdAndUpdate(
      req.params.postId,
      {
        $set: {
          title: req.body.title,
          content: req.body.content,
          category: req.body.category,
          image: req.body.image,
        },
```

```
      },
      { new: true }
    );
    res.status(200).json(updatedPost);
  } catch (error) {
    next(error);
  }
};
```

Файл user.controller.js

```
import bcryptjs from 'bcryptjs';
import { errorHandler } from '../utils/error.js';
import User from '../models/user.model.js';

export const test = (req, res) => {
  res.json({ message: 'It`s working!' });
};

export const updateUser = async (req, res, next) => {
  if (req.user.id !== req.params.userId) {
    return next(errorHandler(403, 'You are not allowed to update
this user'));
  }
  if (req.body.password) {
    if (req.body.password.length < 6) {
      return next(errorHandler(400, 'Password must be at least 6
characters'));
    }
    req.body.password = bcryptjs.hashSync(req.body.password, 10);
  }
  if (req.body.username) {
    if (req.body.username.length < 7 || req.body.username.length >
20) {
      return next(
        errorHandler(400, 'Username must be between 7 and 20
characters')
      );
    }
    if (req.body.username.includes(' ')) {
      return next(errorHandler(400, 'Username cannot contain
spaces'));
    }
    if (req.body.username !== req.body.username.toLowerCase()) {
      return next(errorHandler(400, 'Username must be lowercase'));
    }
    if (!req.body.username.match(/^[a-zA-Z0-9]+$/)) {
      return next(
        errorHandler(400, 'Username can only contain letters and
numbers')
      );
    }
  }
```

```
  try {
    const updatedUser = await User.findByIdAndUpdate(
      req.params.userId,
      {
        $set: {
          username: req.body.username,
          email: req.body.email,
          profilePicture: req.body.profilePicture,
          password: req.body.password,
        },
      },
      { new: true }
    );
    const { password, ...rest } = updatedUser._doc;
    res.status(200).json(rest);
  } catch (error) {
    next(error);
  }
};

export const deleteUser = async (req, res, next) => {
  if (!req.user.isAdmin && req.user.id !== req.params.userId) {
    return next(errorHandler(403, 'You are not allowed to delete
this user'));
  }
  try {
    await User.findByIdAndDelete(req.params.userId);
    res.status(200).json('User has been deleted');
  } catch (error) {
    next(error);
  }
};

export const signout = (req, res, next) => {
  try {
    res
      .clearCookie('access_token')
      .status(200)
      .json('User has been signed out');
  } catch (error) {
    next(error);
  }
};

export const getClients = async (req, res, next) => {
  if (!req.user.isAdmin) {
    return next(errorHandler(403, 'You are not allowed to see all
users'));
  }
  try {
    const startIndex = parseInt(req.query.startIndex) || 0;
    const limit = parseInt(req.query.limit) || 9;
    const sortDirection = req.query.sort === 'asc' ? 1 : -1;
```

```
    const users = await User.find()
      .sort({ createdAt: sortDirection })
      .skip(startIndex)
      .limit(limit);

    const usersWithoutPassword = users.map(({ _doc }) => {
      const { password, ...rest } = _doc;
      return rest;
    });

    const totalUsers = await User.countDocuments();

    const now = new Date();

    const oneMonthAgo = new Date(
      now.getFullYear(),
      now.getMonth() - 1,
      now.getDate()
    );
    const lastMonthUsers = await User.countDocuments({
      createdAt: { $gte: oneMonthAgo },
    });

    res.status(200).json({
      users: usersWithoutPassword,
      totalUsers,
      lastMonthUsers,
    });
  } catch (error) {
    next(error);
  }
};

export const getClient = async (req, res, next) => {
  try {
    const user = await User.findById(req.params.userId);
    if (!user) {
      return next(errorHandler(404, 'User not found'));
    }
    const { password, ...rest } = user._doc;
    res.status(200).json(rest);
  } catch (error) {
    next(error);
  }
                                    };
```

Файл comment.model.js

```
import mongoose from 'mongoose';

const commentSchema = new mongoose.Schema(
```

```
  {
    content: {
      type: String,
      required: true,
    },
    postId: {
      type: String,
      required: true,
    },
    userId: {
      type: String,
      required: true,
    },
    likes: {
      type: Array,
      default: [],
    },
    numberOfLikes: {
      type: Number,
      default: 0,
    },
  },
  { timestamps: true }
);

const Comment = mongoose.model('Comment', commentSchema);

export default Comment;
```

Файл post.model.js

```
import mongoose from 'mongoose';

const postSchema = new mongoose.Schema(
  {
    userId: {
      type: String,
      required: true,
    },
    content: {
      type: String,
      required: true,
    },
    title: {
      type: String,
      required: true,
      unique: true,
    },
    image: {
      type: String,
```

```
      default:
        'https://www.hostinger.com/tutorials/wp-
content/uploads/sites/2/2021/09/how-to-write-a-blog-post.png',
    },
    category: {
      type: String,
      default: 'uncategorized',
    },
    slug: {
      type: String,
      required: true,
      unique: true,
    },
  },
  { timestamps: true }
);

const Post = mongoose.model('Post', postSchema);

export default Post;
```

Файл user.model.js

```
import mongoose from 'mongoose';

const userSchema = new mongoose.Schema(
  {
    username: {
      type: String,
      required: true,
      unique: true,
    },
    email: {
      type: String,
      required: true,
      unique: true,
    },
    password: {
      type: String,
      required: true,
    },
    profilePicture: {
      type: String,
      default:
        'https://cdn.pixabay.com/photo/2015/10/05/22/37/blank-
profile-picture-973460_960_720.png',
    },
    isAdmin: {
      type: Boolean,
      default: false,
```

```
    },
  },
  { timestamps: true }
);

const User = mongoose.model('User', userSchema);

export default User;
```

Файл auth.route.js

```
import express from 'express';
import { google, signin, signup } from
'../controllers/auth.controller.js';

const router = express.Router();

router.post('/signup', signup);
router.post('/signin', signin);
router.post('/google', google)

export default router;
```

Файл comment.route.js

```
import express from 'express';
import { verifyToken } from '../utils/verifyUser.js';
import {
  createComment,
  deleteComment,
  editComment,
  getPostCom,
  getcomments,
  likeComment,
} from '../controllers/comment.controller.js';

const router = express.Router();

router.post('/create', verifyToken, createComment);
router.get('/getPostCom/:postId', getPostCom);
router.put('/likeComment/:commentId', verifyToken, likeComment);
router.put('/editComment/:commentId', verifyToken, editComment);
router.delete('/deleteComment/:commentId', verifyToken,
deleteComment);
router.get('/getcomments', verifyToken, getcomments);

export default router;
```

Файл post.route.js

```
import express from 'express';
import { verifyToken } from '../utils/verifyUser.js';
import { create, deletepost, getposts, updatepost } from
'../controllers/post.controller.js';

const router = express.Router();

router.post('/create', verifyToken, create)
router.get('/getposts', getposts)
router.delete('/deletepost/:postId/:userId', verifyToken,
deletepost)
router.put('/updatepost/:postId/:userId', verifyToken, updatepost)

export default router;
```

Файл user.route.js

```
import express from 'express';
import {
  deleteUser,
  getClient,
  getClients,
  signout,
  test,
  updateUser,
} from '../controllers/user.controller.js';
import { verifyToken } from '../utils/verifyUser.js';

const router = express.Router();

router.get('/test', test);
router.put('/update/:userId', verifyToken, updateUser);
router.delete('/delete/:userId', verifyToken, deleteUser);
router.post('/signout', signout);
router.get('/getClients', verifyToken, getClients);
router.get('/:userId', getClient);

export default router;
```

Файл error.js

```
export const errorHandler = (statusCode, message) => {
  const error = new Error();
  error.statusCode = statusCode;
  error.message = message;
  return error;
};
```

Файл verifyUser.js

```
import jwt from 'jsonwebtoken';
import { errorHandler } from './error.js';
export const verifyToken = (req, res, next) => {
  const token = req.cookies.access_token;
  if (!token) {
    return next(errorHandler(401, 'Unauthorized'));
  }
  jwt.verify(token, process.env.JWT_SECRET, (err, user) => {
    if (err) {
      return next(errorHandler(401, 'Unauthorized'));
    }
    req.user = user;
    next();
  });
};
```

Файл index.js

```
import express from 'express';
import mongoose from 'mongoose';
import dotenv from 'dotenv';
import userRoutes from './routes/user.route.js';
import authRoutes from './routes/auth.route.js';
import postRoutes from './routes/post.route.js';
import commentRoutes from './routes/comment.route.js';
import cookieParser from 'cookie-parser';
import path from 'path';

dotenv.config();

const connectDB = async () => {
  try {
    await mongoose.connect(process.env.MONGO);
    console.log('MongoDb is connected');
  } catch (err) {
    console.log(err);
  }
};

connectDB();

const __dirname = path.resolve();
const app = express();

app.use(express.json());
app.use(cookieParser());
app.use('/api/user', userRoutes);
app.use('/api/auth', authRoutes);
```

```
app.use('/api/post', postRoutes);
app.use('/api/comment', commentRoutes);
app.use(express.static(path.join(__dirname, '/client/dist')));
app.get('*', (req, res) => {
  res.sendFile(path.join(__dirname,      'client',      'dist',
'index.html'));
});

app.use((err, req, res, next) => {
  const statusCode = err.statusCode || 500;
  const message = err.message || 'Internal Server Error';
  res.status(statusCode).json({
    success: false,
    statusCode,
    message,
  });
});

const PORT = process.env.PORT || 3000;
app.listen(PORT, () => {
  console.log(`Server is running on port ${PORT}!`);
});
```

Клієнтська частина

Файл Comment.jsx

```
import moment from 'moment';
import { useEffect, useState } from 'react';
import { FaThumbsUp } from 'react-icons/fa';
import { useSelector } from 'react-redux';
import { Button, Textarea } from 'flowbite-react';

export default function Comment({ comment, onLike, onEdit, onDelete
}) {
  const [user, setUser] = useState({});
  const [isEditing, setIsEditing] = useState(false);
  const [editedContent, setEditedContent] =
useState(comment.content);
  const { currentUser } = useSelector((state) => state.user);

  useEffect(() => {
    const getClient = async () => {
      try {
        const res = await fetch(`/api/user/${comment.userId}`);
        const data = await res.json();
        if (res.ok) {
          setUser(data);
        }
      } catch (error) {
        console.log(error.message);
      }
    };
    getClient();
  }, [comment]);

  const handleEdit = () => {
    setIsEditing(true);
    setEditedContent(comment.content);
  };

  const handleSave = async () => {
    try {
      const res = await
fetch(`/api/comment/editComment/${comment._id}`, {
        method: 'PUT',
        headers: {
          'Content-Type': 'application/json',
        },
        body: JSON.stringify({
          content: editedContent,
        }),
      });
      if (res.ok) {
        setIsEditing(false);
```

```
      onEdit(comment, editedContent);
    }
  } catch (error) {
    console.log(error.message);
  }
};

return (
  <div className='flex p-4 border-b dark:border-gray-600 text-sm'>
    <div className='flex-shrink-0 mr-3'>
      <img
        className='w-10 h-10 rounded-full bg-gray-200'
        src={user.profilePicture}
        alt={user.username}
      />
    </div>
    <div className='flex-1'>
      <div className='flex items-center mb-1'>
        <span className='font-bold mr-1 text-xs truncate'>
          {user ? `@${user.username}` : 'anonymous user'}
        </span>
        <span className='text-gray-500 text-xs'>
          {moment(comment.createdAt).fromNow()}
        </span>
      </div>
      {isEditing ? (
        <>
          <Textarea
            className='mb-2'
            value={editedContent}
            onChange={(e) => setEditedContent(e.target.value)}
          />
          <div className='flex justify-end gap-2 text-xs'>
            <Button
              type='button'
              size='sm'
              gradientDuoTone='redToYellow'
              onClick={handleSave}
            >
              Save
            </Button>
            <Button
              type='button'
              size='sm'
              gradientDuoTone='redToYellow'
              outline
              onClick={() => setIsEditing(false)}
            >
              Cancel
            </Button>
          </div>
        </>
      ) : (
```

```
          <>
            <p className='text-gray-500 pb-2'>{comment.content}</p>
            <div className='flex items-center pt-2 text-xs border-t
dark:border-gray-700 max-w-fit gap-2'>
              <button
                type='button'
                onClick={() => onLike(comment._id)}
                className={`text-gray-400 hover:text-blue-500 ${
                  currentUser &&
                  comment.likes.includes(currentUser._id) &&
                  '!text-blue-500'
                }`}
              >
                <FaThumbsUp className='text-sm' />
              </button>
              <p className='text-gray-400'>
                {comment.numberOfLikes > 0 &&
                  comment.numberOfLikes +
                    ' ' +
                    (comment.numberOfLikes === 1 ? 'like' :
'likes')}
              </p>
              {currentUser &&
                (currentUser._id === comment.userId ||
currentUser.isAdmin) && (
                  <>
                    <button
                      type='button'
                      onClick={handleEdit}
                      className='text-gray-400 hover:text-blue-500'
                    >
                      Edit
                    </button>
                    <button
                      type='button'
                      onClick={() => onDelete(comment._id)}
                      className='text-gray-400 hover:text-red-500'
                    >
                      Delete
                    </button>
                  </>
                )}
            </div>
          </>
        )}
      </div>
    </div>
  );
}
```

Файл CommentSection.jsx

```
import { Alert, Button, Modal, TextInput, Textarea } from 'flowbite-
react';
import { useEffect, useState } from 'react';
import { useSelector } from 'react-redux';
import { Link, useNavigate } from 'react-router-dom';
import Comment from './Comment';
import { HiOutlineExclamationCircle } from 'react-icons/hi';

export default function CommentSection({ postId }) {
  const { currentUser } = useSelector((state) => state.user);
  const [comment, setComment] = useState('');
  const [commentError, setCommentError] = useState(null);
  const [comments, setComments] = useState([]);
  const [showModal, setShowModal] = useState(false);
  const [commentToDelete, setCommentToDelete] = useState(null);
  const navigate = useNavigate();

  const handleSubmit = async (e) => {
    e.preventDefault();
    if (comment.length > 200) {
      return;
    }
    try {
      const res = await fetch('/api/comment/create', {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json',
        },
        body: JSON.stringify({
          content: comment,
          postId,
          userId: currentUser?.userId,
        }),
      });
      const data = await res.json();
      if (res.ok) {
        setComment('');
        setCommentError(null);
        setComments([data, ...comments]);
      }
    } catch (error) {
      setCommentError(error.message);
    }
  };

  useEffect(() => {
    const getComments = async () => {
      try {
        const res = await
fetch(`/api/comment/getPostCom/${postId}`);
        if (res.ok) {
          const data = await res.json();
          setComments(data);
```

```
        }
      } catch (error) {
        console.log(error.message);
      }
    };
    getComments();
  }, [postId]);

  const handleLike = async (commentId) => {
    try {
      if (!currentUser) {
        navigate('/sign-in');
        return;
      }
      const res = await
fetch(`/api/comment/likeComment/${commentId}`, {
        method: 'PUT',
      });
      if (res.ok) {
        const data = await res.json();
        setComments(
          comments.map((comment) =>
            comment._id === commentId
              ? {
                  ...comment,
                  likes: data.likes,
                  numberOfLikes: data.likes.length,
                }
              : comment
          )
        );
      }
    } catch (error) {
      console.log(error.message);
    }
  };

  const handleEdit = async (comment, editedContent) => {
    setComments(
      comments.map((c) =>
        c._id === comment._id ? { ...c, content: editedContent } : c
      )
    );
  };

  const handleDelete = async (commentId) => {
    setShowModal(false);
    try {
      if (!currentUser) {
        navigate('/sign-in');
        return;
      }
```

```
      const res = await
fetch(`/api/comment/deleteComment/${commentId}`, {
        method: 'DELETE',
      });
      if (res.ok) {
        const data = await res.json();
        setComments(comments.filter((comment) => comment._id !==
commentId));
      }
    } catch (error) {
      console.log(error.message);
    }
  };

  return (
    <div className='max-w-2xl mx-auto w-full p-3'>
      {currentUser ? (
        <div className='flex items-center gap-1 my-5 text-gray-500
text-sm'>
          <p>Signed in as:</p>
          <img
            className='h-5 w-5 object-cover rounded-full'
            src={currentUser.profilePicture}
            alt=''
          />
          <Link
            to={'/dashboard?tab=profile'}
            className='text-xs text-cyan-600 hover:underline'
          >
            @{currentUser.username}
          </Link>
        </div>
      ) : (
        <div className='text-sm text-teal-500 my-5 flex gap-1'>
          You must be signed in to comment.
          <Link className='text-blue-500 hover:underline'
to={'/sign-in'}>
            Sign In
          </Link>
        </div>
      )}
      {currentUser && (
        <form
          onSubmit={handleSubmit}
          className='border border-teal-500 rounded-md p-3'
        >
          <Textarea
            placeholder='Add a comment...'
            rows='3'
            maxLength='200'
            onChange={(e) => setComment(e.target.value)}
            value={comment}
          />
```

```
          <div className='flex justify-between items-center mt-5'>
            <p className='text-gray-500 text-xs'>
              {200 - comment.length} characters remaining
            </p>
            <Button outline gradientDuoTone='redToYellow'
type='submit'>
              Submit
            </Button>
          </div>
          {commentError && (
            <Alert color='failure' className='mt-5'>
              {commentError}
            </Alert>
          )}
        </form>
      )}
      {comments.length === 0 ? (
        <p className='text-sm my-5'>No comments yet!</p>
      ) : (
        <>
          <div className='text-sm my-5 flex items-center gap-1'>
            <p>Comments</p>
            <div className='border border-gray-400 py-1 px-2
rounded-sm'>
              <p>{comments.length}</p>
            </div>
          </div>
          {comments.map((comment) => (
            <Comment
              key={comment._id}
              comment={comment}
              onLike={handleLike}
              onEdit={handleEdit}
              onDelete={(commentId) => {
                setShowModal(true);
                setCommentToDelete(commentId);
              }}
            />
          ))}
        </>
      )}
      <Modal
        show={showModal}
        onClose={() => setShowModal(false)}
        popup
        size='md'
      >
        <Modal.Header />
        <Modal.Body>
          <div className='text-center'>
            <HiOutlineExclamationCircle className='h-14 w-14 text-
gray-400 dark:text-gray-200 mb-4 mx-auto' />
```

```
            <h3 className='mb-5 text-lg text-gray-500 dark:text-
gray-400'>
              Are you sure you want to delete this comment?
            </h3>
            <div className='flex justify-center gap-4'>
              <Button
                color='failure'
                onClick={() => handleDelete(commentToDelete)}
              >
                Yes, I'm sure
              </Button>
              <Button color='gray' onClick={() =>
setShowModal(false)}>
                No, cancel
              </Button>
            </div>
          </div>
        </Modal.Body>
      </Modal>
    </div>
  );
}
```

Файл DashboardComp.jsx

```
import { useEffect, useState } from 'react';
import { useSelector } from 'react-redux';
import {
  HiAnnotation,
  HiArrowNarrowUp,
  HiDocumentText,
  HiOutlineUserGroup,
} from 'react-icons/hi';
import { Button, Table } from 'flowbite-react';
import { Link } from 'react-router-dom';

export default function DashboardComp() {
  const [users, setUsers] = useState([]);
  const [comments, setComments] = useState([]);
  const [posts, setPosts] = useState([]);
  const [totalUsers, setTotalUsers] = useState(0);
  const [totalPosts, setTotalPosts] = useState(0);
  const [totalComments, setTotalComments] = useState(0);
  const [lastMonthUsers, setLastMonthUsers] = useState(0);
  const [lastMonthPosts, setLastMonthPosts] = useState(0);
  const [lastMonthComments, setLastMonthComments] = useState(0);
  const { currentUser } = useSelector((state) => state.user);

  useEffect(() => {
    const fetchUsers = async () => {
      try {
        const res = await fetch('/api/user/getClients?limit=5');
```

```
        const data = await res.json();
        if (res.ok) {
          setUsers(data.users);
          setTotalUsers(data.totalUsers);
          setLastMonthUsers(data.lastMonthUsers);
        }
      } catch (error) {
        console.log(error.message);
      }
    };

    const fetchPosts = async () => {
      try {
        const res = await fetch('/api/post/getposts?limit=5');
        const data = await res.json();
        if (res.ok) {
          setPosts(data.posts);
          setTotalPosts(data.totalPosts);
          setLastMonthPosts(data.lastMonthPosts);
        }
      } catch (error) {
        console.log(error.message);
      }
    };

    const fetchComments = async () => {
      try {
        const res = await fetch('/api/comment/getcomments?limit=5');
        const data = await res.json();
        if (res.ok) {
          setComments(data.comments);
          setTotalComments(data.totalComments);
          setLastMonthComments(data.lastMonthComments);
        }
      } catch (error) {
        console.log(error.message);
      }
    };

    if (currentUser?.isAdmin) {
      fetchUsers();
      fetchPosts();
      fetchComments();
    }
  }, [currentUser]);

  return (
    <div className='p-3 md:mx-auto'>
      <div className='flex-wrap flex gap-4 justify-center'>
        <div className='flex flex-col p-3 dark:bg-slate-800 gap-4
md:w-72 w-full rounded-md shadow-md'>
          <div className='flex justify-between'>
            <div className=''>
```

```
              <h3 className='text-gray-500 text-md uppercase'>Total
Users</h3>
              <p className='text-2xl'>{totalUsers}</p>
            </div>
            <HiOutlineUserGroup className='bg-teal-600  text-white
rounded-full text-5xl p-3 shadow-lg' />
          </div>
          <div className='flex  gap-2 text-sm'>
            <span className='text-green-500 flex items-center'>
              <HiArrowNarrowUp />
              {lastMonthUsers}
            </span>
            <div className='text-gray-500'>Last month</div>
          </div>
        </div>
        <div className='flex flex-col p-3 dark:bg-slate-800 gap-4
md:w-72 w-full rounded-md shadow-md'>
          <div className='flex justify-between'>
            <div className=''>
              <h3 className='text-gray-500 uppercase'>
                Total Comments
              </h3>
              <p className='text-2xl'>{totalComments}</p>
            </div>
            <HiAnnotation className='bg-indigo-600  text-white
rounded-full text-5xl p-3 shadow-lg' />
          </div>
          <div className='flex  gap-2 text-sm'>
            <span className='text-green-500 flex items-center'>
              <HiArrowNarrowUp />
              {lastMonthComments}
            </span>
            <div className='text-gray-500'>Last month</div>
          </div>
        </div>
        <div className='flex flex-col p-3 dark:bg-slate-800 gap-4
md:w-72 w-full rounded-md shadow-md'>
          <div className='flex justify-between'>
            <div className=''>
              <h3 className='text-gray-500 text-md uppercase'>Total
Posts</h3>
              <p className='text-2xl'>{totalPosts}</p>
            </div>
            <HiDocumentText className='bg-lime-600  text-white
rounded-full text-5xl p-3 shadow-lg' />
          </div>
          <div className='flex  gap-2 text-sm'>
            <span className='text-green-500 flex items-center'>
              <HiArrowNarrowUp />
              {lastMonthPosts}
            </span>
            <div className='text-gray-500'>Last month</div>
          </div>
```

```
        </div>
      </div>
      <div className='flex flex-wrap gap-4 py-3 mx-auto justify-
center'>
        <div className='flex flex-col w-full md:w-auto shadow-md p-2
rounded-md dark:bg-gray-800'>
          <div className='flex justify-between  p-3 text-sm font-
semibold'>
            <h1 className='text-center p-2'>Recent users</h1>
            <Button outline gradientDuoTone='redToYellow'>
              <Link to={'/dashboard?tab=users'}>See all</Link>
            </Button>
          </div>
          <Table hoverable>
            <Table.Head>
              <Table.HeadCell>User image</Table.HeadCell>
              <Table.HeadCell>Username</Table.HeadCell>
            </Table.Head>
            {users &&
              users.map((user) => (
                <Table.Body key={user._id} className='divide-y'>
                  <Table.Row className='bg-white dark:border-gray-
700 dark:bg-gray-800'>
                    <Table.Cell>
                      <img
                        src={user.profilePicture}
                        alt='user'
                        className='w-10 h-10 rounded-full bg-gray-
500'
                      />
                    </Table.Cell>
                    <Table.Cell>{user.username}</Table.Cell>
                  </Table.Row>
                </Table.Body>
              ))}
          </Table>
        </div>
        <div className='flex flex-col w-full md:w-auto shadow-md p-2
rounded-md dark:bg-gray-800'>
          <div className='flex justify-between  p-3 text-sm font-
semibold'>
            <h1 className='text-center p-2'>Recent comments</h1>
            <Button outline gradientDuoTone='redToYellow'>
              <Link to={'/dashboard?tab=comments'}>See all</Link>
            </Button>
          </div>
          <Table hoverable>
            <Table.Head>
              <Table.HeadCell>Comment content</Table.HeadCell>
              <Table.HeadCell>Likes</Table.HeadCell>
            </Table.Head>
            {comments &&
              comments.map((comment) => (
```

```
                <Table.Body key={comment._id} className='divide-y'>
                  <Table.Row className='bg-white dark:border-gray-
700 dark:bg-gray-800'>
                    <Table.Cell className='w-96'>
                        <p className='line-clamp-
2'>{comment.content}</p>
                    </Table.Cell>
                    <Table.Cell>{comment.numberOfLikes}</Table.Cell>
                  </Table.Row>
                </Table.Body>
              ))}
          </Table>
        </div>
        <div className='flex flex-col w-full md:w-auto shadow-md p-2
rounded-md dark:bg-gray-800'>
          <div className='flex justify-between  p-3 text-sm font-
semibold'>
            <h1 className='text-center p-2'>Recent posts</h1>
            <Button outline gradientDuoTone='redToYellow'>
              <Link to={'/dashboard?tab=posts'}>See all</Link>
            </Button>
          </div>
          <Table hoverable>
            <Table.Head>
              <Table.HeadCell>Post image</Table.HeadCell>
              <Table.HeadCell>Post Title</Table.HeadCell>
              <Table.HeadCell>Category</Table.HeadCell>
            </Table.Head>
            {posts &&
              posts.map((post) => (
                <Table.Body key={post._id} className='divide-y'>
                  <Table.Row className='bg-white dark:border-gray-
700 dark:bg-gray-800'>
                    <Table.Cell>
                      <img
                        src={post.image}
                        alt='user'
                        className='w-14 h-10 rounded-md bg-gray-500'
                      />
                    </Table.Cell>
                    <Table.Cell className='w-
96'>{post.title}</Table.Cell>
                    <Table.Cell className='w-
5'>{post.category}</Table.Cell>
                  </Table.Row>
                </Table.Body>
              ))}
          </Table>
        </div>
      </div>
    </div>
  );
}
```

Файл DashComments.jsx

```
import { Modal, Table, Button } from 'flowbite-react';
import { useEffect, useState } from 'react';
import { useSelector } from 'react-redux';
import { HiOutlineExclamationCircle } from 'react-icons/hi';

export default function DashComments() {
  const { currentUser } = useSelector((state) => state.user);
  const [comments, setComments] = useState([]);
  const [showMore, setShowMore] = useState(true);
  const [showModal, setShowModal] = useState(false);
  const [commentIdToDelete, setCommentIdToDelete] = useState('');

  useEffect(() => {
    const fetchComments = async () => {
      try {
        const res = await fetch(`/api/comment/getcomments`);
        const data = await res.json();
        if (res.ok) {
          setComments(data.comments);
          setShowMore(data.comments.length >= 9);
        }
      } catch (error) {
        console.log(error.message);
      }
    };
    if (currentUser.isAdmin) {
      fetchComments();
    }
  }, [currentUser._id]);

  const handleShowMore = async () => {
    const startIndex = comments.length;
    try {
      const res = await fetch(
        `/api/comment/getcomments?startIndex=${startIndex}`
      );
      const data = await res.json();
      if (res.ok) {
        setComments((prev) => [...prev, ...data.comments]);
        setShowMore(data.comments.length >= 9);
      }
    } catch (error) {
      console.log(error.message);
    }
  };

  const handleDeleteComment = async () => {
    setShowModal(false);
    try {
```

```
      const res = await fetch(
        `/api/comment/deleteComment/${commentIdToDelete}`,
        {
          method: 'DELETE',
        }
      );
      const data = await res.json();
      if (res.ok) {
        setComments((prev) =>
          prev.filter((comment) => comment._id !==
commentIdToDelete)
        );
        setShowModal(false);
      } else {
        console.log(data.message);
      }
    } catch (error) {
      console.log(error.message);
    }
  };

  return (
    <div className='table-auto overflow-x-scroll md:mx-auto p-3
scrollbar scrollbar-track-slate-100 scrollbar-thumb-slate-300
dark:scrollbar-track-slate-700 dark:scrollbar-thumb-slate-500'>
      {currentUser.isAdmin && comments.length > 0 ? (
        <>
          <Table hoverable className='shadow-md'>
            <Table.Head>
              <Table.HeadCell>Date updated</Table.HeadCell>
              <Table.HeadCell>Comment content</Table.HeadCell>
              <Table.HeadCell>Number of likes</Table.HeadCell>
              <Table.HeadCell>PostId</Table.HeadCell>
              <Table.HeadCell>UserId</Table.HeadCell>
              <Table.HeadCell>Delete</Table.HeadCell>
            </Table.Head>
            {comments.map((comment) => (
              <Table.Body key={comment._id} className='divide-y'>
                <Table.Row className='bg-white dark:border-gray-700
dark:bg-gray-800'>
                  <Table.Cell>
                    {new
Date(comment.updatedAt).toLocaleDateString()}
                  </Table.Cell>
                  <Table.Cell>{comment.content}</Table.Cell>
                  <Table.Cell>{comment.numberOfLikes}</Table.Cell>
                  <Table.Cell>{comment.postId}</Table.Cell>
                  <Table.Cell>{comment.userId}</Table.Cell>
                  <Table.Cell>
                    <span
                      onClick={() => {
                        setShowModal(true);
                        setCommentIdToDelete(comment._id);
```

```
                    }}
                    className='font-medium text-red-500
hover:underline cursor-pointer'
                  >
                    Delete
                  </span>
                </Table.Cell>
              </Table.Row>
            </Table.Body>
          ))}
        </Table>
        {showMore && (
          <button
            onClick={handleShowMore}
            className='w-full text-teal-500 self-center text-sm
py-7'
          >
            Show more
          </button>
        )}
      </>
    ) : (
      <p>You have no comments yet!</p>
    )}
    <Modal
      show={showModal}
      onClose={() => setShowModal(false)}
      popup
      size='md'
    >
      <Modal.Header />
      <Modal.Body>
        <div className='text-center'>
          <HiOutlineExclamationCircle className='h-14 w-14 text-
gray-400 dark:text-gray-200 mb-4 mx-auto' />
          <h3 className='mb-5 text-lg text-gray-500 dark:text-
gray-400'>
            Are you sure you want to delete this comment?
          </h3>
          <div className='flex justify-center gap-4'>
            <Button color='failure' onClick={handleDeleteComment}>
              Yes, I'm sure
            </Button>
            <Button color='gray' onClick={() =>
setShowModal(false)}>
              No, cancel
            </Button>
          </div>
        </div>
      </Modal.Body>
    </Modal>
  </div>
);
```

```
}
```

Файл DashPosts.jsx

```
import { Modal, Table, Button } from 'flowbite-react';
import { useEffect, useState } from 'react';
import { useSelector } from 'react-redux';
import { Link } from 'react-router-dom';
import { HiOutlineExclamationCircle } from 'react-icons/hi';

export default function DashPosts() {
  const { currentUser } = useSelector((state) => state.user);
  const [userPosts, setUserPosts] = useState([]);
  const [showMore, setShowMore] = useState(true);
  const [showModal, setShowModal] = useState(false);
  const [postIdToDelete, setPostIdToDelete] = useState('');

  useEffect(() => {
    const fetchPosts = async () => {
      try {
        const res = await
fetch(`/api/post/getposts?userId=${currentUser._id}`);
        const data = await res.json();
        if (res.ok) {
          setUserPosts(data.posts);
          setShowMore(data.posts.length >= 9);
        }
      } catch (error) {
        console.log(error.message);
      }
    };
    if (currentUser.isAdmin) {
      fetchPosts();
    }
  }, [currentUser._id]);

  const handleShowMore = async () => {
    const startIndex = userPosts.length;
    try {
      const res = await fetch(

`/api/post/getposts?userId=${currentUser._id}&startIndex=${startInde
x}`
      );
      const data = await res.json();
      if (res.ok) {
        setUserPosts((prev) => [...prev, ...data.posts]);
        setShowMore(data.posts.length >= 9);
      }
    } catch (error) {
      console.log(error.message);
    }
```

```
  };

  const handleDeletePost = async () => {
    setShowModal(false);
    try {
      const res = await fetch(
        `/api/post/deletepost/${postIdToDelete}/${currentUser._id}`,
        {
          method: 'DELETE',
        }
      );
      const data = await res.json();
      if (!res.ok) {
        console.log(data.message);
      } else {
        setUserPosts((prev) =>
          prev.filter((post) => post._id !== postIdToDelete)
        );
      }
    } catch (error) {
      console.log(error.message);
    }
  };

  return (
    <div classNam='table-auto overflow-x-scroll md:mx-auto p-3
scrollbar scrollbar-track-slate-100 scrollbar-thumb-slate-300
dark:scrollbar-track-slate-700 dark:scrollbar-thumb-slate-500'>
      {currentUser.isAdmin && userPosts.length > 0 ? (
        <>
          <Table hoverable className='shadow-md'>
            <Table.Head>
              <Table.HeadCell>Date updated</Table.HeadCell>
              <Table.HeadCell>Post image</Table.HeadCell>
              <Table.HeadCell>Post title</Table.HeadCell>
              <Table.HeadCell>Category</Table.HeadCell>
              <Table.HeadCell>Delete</Table.HeadCell>
              <Table.HeadCell>
                <span>Edit</span>
              </Table.HeadCell>
            </Table.Head>
            {userPosts.map((post) => (
              <Table.Body key={post._id} className='divide-y'>
                <Table.Row className='bg-white dark:border-gray-700
dark:bg-gray-800'>
                  <Table.Cell>
                    {new Date(post.updatedAt).toLocaleDateString()}
                  </Table.Cell>
                  <Table.Cell>
                    <Link to={`/post/${post.slug}`}>
                      <img
                        src={post.image}
                        alt={post.title}
```

```
                        className='w-20 h-10 object-cover bg-gray-
500'
                      />
                    </Link>
                  </Table.Cell>
                  <Table.Cell>
                    <Link
                      className='font-medium text-gray-900
dark:text-white'
                      to={`/post/${post.slug}`}
                    >
                      {post.title}
                    </Link>
                  </Table.Cell>
                  <Table.Cell>{post.category}</Table.Cell>
                  <Table.Cell>
                    <span
                      onClick={() => {
                        setShowModal(true);
                        setPostIdToDelete(post._id);
                      }}
                      className='font-medium text-red-500
hover:underline cursor-pointer'
                    >
                      Delete
                    </span>
                  </Table.Cell>
                  <Table.Cell>
                    <Link
                      className='text-teal-500 hover:underline'
                      to={`/update-post/${post._id}`}
                    >
                      <span>Edit</span>
                    </Link>
                  </Table.Cell>
                </Table.Row>
              </Table.Body>
            ))}
          </Table>
          {showMore && (
            <button
              onClick={handleShowMore}
              className='w-full text-teal-500 self-center text-sm
py-7'
            >
              Show more
            </button>
          )}
        </>
      ) : (
        <p>You have no posts yet!</p>
      )}
      <Modal
```

```
        show={showModal}
        onClose={() => setShowModal(false)}
        popup
        size='md'
      >
        <Modal.Header />
        <Modal.Body>
          <div className='text-center'>
            <HiOutlineExclamationCircle className='h-14 w-14 text-
gray-400 dark:text-gray-200 mb-4 mx-auto' />
            <h3 className='mb-5 text-lg text-gray-500 dark:text-
gray-400'>
              Are you sure you want to delete this post?
            </h3>
            <div className='flex justify-center gap-4'>
              <Button color='failure' onClick={handleDeletePost}>
                Yes, I'm sure
              </Button>
              <Button color='gray' onClick={() =>
setShowModal(false)}>
                No, cancel
              </Button>
            </div>
          </div>
        </Modal.Body>
      </Modal>
    </div>
  );
}
```

Файл DashProfile.jsx

```
import { Alert, Button, Modal, TextInput } from 'flowbite-react';
import { useEffect, useRef, useState } from 'react';
import { useSelector } from 'react-redux';
import {
  getDownloadURL,
  getStorage,
  ref,
  uploadBytesResumable,
} from 'firebase/storage';
import { app } from '../firebase';
import { CircularProgressbar } from 'react-circular-progressbar';
import 'react-circular-progressbar/dist/styles.css';
import {
  updateStart,
  updateSuccess,
  updateFailure,
  deleteUserStart,
  deleteUserSuccess,
  deleteUserFailure,
  signoutSuccess,
```

```
} from '../redux/user/userSlice';
import { useDispatch } from 'react-redux';
import { HiOutlineExclamationCircle } from 'react-icons/hi';
import { Link } from 'react-router-dom';

export default function DashProfile() {
  const { currentUser, error, loading } = useSelector((state) =>
state.user);
  const [imageFile, setImageFile] = useState(null);
  const [imageFileUrl, setImageFileUrl] = useState(null);
  const [imageFileUploadProgress, setImageFileUploadProgress] =
useState(null);
  const [imageFileUploadError, setImageFileUploadError] =
useState(null);
  const [imageFileUploading, setImageFileUploading] =
useState(false);
  const [updateUserSuccess, setUpdateUserSuccess] = useState(null);
  const [updateUserError, setUpdateUserError] = useState(null);
  const [showModal, setShowModal] = useState(false);
  const [formData, setFormData] = useState({});
  const filePickerRef = useRef();
  const dispatch = useDispatch();

  const handleImageChange = (e) => {
    const file = e.target.files[0];
    if (file) {
      setImageFile(file);
      setImageFileUrl(URL.createObjectURL(file));
    }
  };

  useEffect(() => {
    if (imageFile) {
      uploadImage();
    }
  }, [imageFile]);

  const uploadImage = async () => {
    setImageFileUploading(true);
    setImageFileUploadError(null);
    const storage = getStorage(app);
    const fileName = new Date().getTime() + imageFile.name;
    const storageRef = ref(storage, fileName);
    const uploadTask = uploadBytesResumable(storageRef, imageFile);
    uploadTask.on(
      'state_changed',
      (snapshot) => {
        const progress =
          (snapshot.bytesTransferred / snapshot.totalBytes) * 100;

        setImageFileUploadProgress(progress.toFixed(0));
      },
      (error) => {
```

```
        setImageFileUploadError(
          'Could not upload image (File must be less than 2MB)'
        );
        setImageFileUploadProgress(null);
        setImageFile(null);
        setImageFileUrl(null);
        setImageFileUploading(false);
      },
      () => {
        getDownloadURL(uploadTask.snapshot.ref).then((downloadURL)
=> {
          setImageFileUrl(downloadURL);
          setFormData({ ...formData, profilePicture: downloadURL });
          setImageFileUploading(false);
        });
      }
    );
  };

  const handleChange = (e) => {
    setFormData({ ...formData, [e.target.id]: e.target.value });
  };

  const handleSubmit = async (e) => {
    e.preventDefault();
    setUpdateUserError(null);
    setUpdateUserSuccess(null);
    if (Object.keys(formData).length === 0) {
      setUpdateUserError('No changes made');
      return;
    }
    if (imageFileUploading) {
      setUpdateUserError('Please wait for image to upload');
      return;
    }
    try {
      dispatch(updateStart());
      const res = await fetch(`/api/user/update/${currentUser._id}`,
{
        method: 'PUT',
        headers: {
          'Content-Type': 'application/json',
        },
        body: JSON.stringify(formData),
      });
      const data = await res.json();
      if (!res.ok) {
        dispatch(updateFailure(data.message));
        setUpdateUserError(data.message);
      } else {
        dispatch(updateSuccess(data));
        setUpdateUserSuccess("User's profile updated successfully");
      }
```

```
    } catch (error) {
      dispatch(updateFailure(error.message));
      setUpdateUserError(error.message);
    }
  };

  const handleDeleteUser = async () => {
    setShowModal(false);
    try {
      dispatch(deleteUserStart());
      const res = await fetch(`/api/user/delete/${currentUser._id}`,
{
        method: 'DELETE',
      });
      const data = await res.json();
      if (!res.ok) {
        dispatch(deleteUserFailure(data.message));
      } else {
        dispatch(deleteUserSuccess(data));
      }
    } catch (error) {
      dispatch(deleteUserFailure(error.message));
    }
  };

  const handleSignout = async () => {
    try {
      const res = await fetch('/api/user/signout', {
        method: 'POST',
      });
      const data = await res.json();
      if (!res.ok) {
        console.log(data.message);
      } else {
        dispatch(signoutSuccess());
      }
    } catch (error) {
      console.log(error.message);
    }
  };

  return (
    <div className='max-w-lg mx-auto p-3 w-full'>
      <h1 className='my-7 text-center font-semibold text-
3xl'>Profile</h1>
      <form onSubmit={handleSubmit} className='flex flex-col gap-4'>
        <input
          type='file'
          accept='image/*'
          onChange={handleImageChange}
          ref={filePickerRef}
          hidden        />
        <div
```

```
          className='relative w-32 h-32 self-center cursor-pointer
shadow-md overflow-hidden rounded-full'
          onClick={() => filePickerRef.current.click()}
        >
          {imageFileUploadProgress && (
            <CircularProgressbar
              value={imageFileUploadProgress || 0}
              text={`${imageFileUploadProgress}%`}
              strokeWidth={5}
              styles={{
                root: {
                  width: '100%',
                  height: '100%',
                  position: 'absolute',
                  top: 0,
                  left: 0,
                },
                path: {
                  stroke: `rgba(62, 152, 199, ${
                    imageFileUploadProgress / 100
                  })`,
                },
              }}
            />
          )}
          <img
            src={imageFileUrl || currentUser.profilePicture}
            alt='user'
            className={`rounded-full w-full h-full object-cover
border-8 border-[lightgray] ${
              imageFileUploadProgress &&
              imageFileUploadProgress < 100 &&
              'opacity-60'
            }`}
          />
        </div>
        {imageFileUploadError && (
          <Alert color='failure'>{imageFileUploadError}</Alert>
        )}
        <TextInput
          type='text'
          id='username'
          placeholder='username'
          defaultValue={currentUser.username}
          onChange={handleChange}
        />
        <TextInput
          type='email'
          id='email'
          placeholder='email'
          defaultValue={currentUser.email}
          onChange={handleChange}
        />
```

```
          <TextInput
            type='password'
            id='password'
            placeholder='password'
            onChange={handleChange}
          />
          <Button
            type='submit'
            gradientDuoTone='redToYellow'
            outline
            disabled={loading || imageFileUploading}
          >
            {loading ? 'Loading...' : 'Update'}
          </Button>
          {currentUser.isAdmin && (
            <Link to={'/create-post'}>
              <Button
                type='button'
                gradientDuoTone='redToYellow'
                className='w-full'
              >
                Create a post
              </Button>
            </Link>
          )}
        </form>
        <div className='text-red-500 flex justify-between mt-5'>
          <span onClick={() => setShowModal(true)} className='cursor-
pointer'>
            Delete Account
          </span>
          <span onClick={handleSignout} className='cursor-pointer'>
            Sign Out
          </span>
        </div>
        {updateUserSuccess && (
          <Alert color='success' className='mt-5'>
            {updateUserSuccess}
          </Alert>
        )}
        {updateUserError && (
          <Alert color='failure' className='mt-5'>
            {updateUserError}
          </Alert>
        )}
        {error && (
          <Alert color='failure' className='mt-5'>
            {error}
          </Alert>
        )}
        <Modal
          show={showModal}
          onClose={() => setShowModal(false)}
```

```
        popup
        size='md'
      >
        <Modal.Header />
        <Modal.Body>
          <div className='text-center'>
            <HiOutlineExclamationCircle className='h-14 w-14 text-
gray-400 dark:text-gray-200 mb-4 mx-auto' />
            <h3 className='mb-5 text-lg text-gray-500 dark:text-
gray-400'>
              Are you sure you want to delete your account?
            </h3>
            <div className='flex justify-center gap-4'>
              <Button color='failure' onClick={handleDeleteUser}>
                Yes, I'm sure
              </Button>
              <Button color='gray' onClick={() =>
setShowModal(false)}>
                No, cancel
              </Button>
            </div>
          </div>
        </Modal.Body>
      </Modal>
    </div>
  );
}
```

Файл DashSidebar.jsx

```
import { Sidebar } from 'flowbite-react';
import {
  HiUser,
  HiArrowSmRight,
  HiDocumentText,
  HiOutlineUserGroup,
  HiAnnotation,
  HiChartPie,
} from 'react-icons/hi';
import { useEffect, useState } from 'react';
import { Link, useLocation } from 'react-router-dom';
import { signoutSuccess } from '../redux/user/userSlice';
import { useDispatch, useSelector } from 'react-redux';

export default function DashSidebar() {
  const location = useLocation();
  const dispatch = useDispatch();
  const { currentUser } = useSelector((state) => state.user);
  const [tab, setTab] = useState('');
  useEffect(() => {
    const urlParams = new URLSearchParams(location.search);
    const tabFromUrl = urlParams.get('tab');
```

```
  if (tabFromUrl) {
    setTab(tabFromUrl);
  }
}, [location.search]);
const handleSignout = async () => {
  try {
    const res = await fetch('/api/user/signout', {
      method: 'POST',
    });
    const data = await res.json();
    if (!res.ok) {
      console.error(data.message);
    } else {
      dispatch(signoutSuccess());
    }
  } catch (error) {
    console.error(error.message);
  }
};
return (
  <Sidebar className='w-full md:w-56'>
    <Sidebar.Items>
      <Sidebar.ItemGroup className='flex flex-col gap-1'>
        {currentUser && currentUser.isAdmin && (
          <Link to='/dashboard?tab=dash'>
            <Sidebar.Item
              active={tab === 'dash' || !tab}
              icon={HiChartPie}
              as='div'
            >
              Dashboard
            </Sidebar.Item>
          </Link>
        )}
        <Link to='/dashboard?tab=profile'>
          <Sidebar.Item
            active={tab === 'profile'}
            icon={HiUser}
            label={currentUser.isAdmin ? 'Admin' : 'User'}
            labelColor='dark'
            as='div'
          >
            Profile
          </Sidebar.Item>
        </Link>
        {currentUser.isAdmin && (
          <Link to='/dashboard?tab=posts'>
            <Sidebar.Item
              active={tab === 'posts'}
              icon={HiDocumentText}
              as='div'
            >
              Posts
```

```
              </Sidebar.Item>
            </Link>
          )}
          {currentUser.isAdmin && (
            <>
              <Link to='/dashboard?tab=users'>
                <Sidebar.Item
                  active={tab === 'users'}
                  icon={HiOutlineUserGroup}
                  as='div'
                >
                  Users
                </Sidebar.Item>
              </Link>
              <Link to='/dashboard?tab=comments'>
                <Sidebar.Item
                  active={tab === 'comments'}
                  icon={HiAnnotation}
                  as='div'
                >
                  Comments
                </Sidebar.Item>
              </Link>
            </>
          )}
          <Sidebar.Item
            icon={HiArrowSmRight}
            className='cursor-pointer'
            onClick={handleSignout}
          >
            Sign Out
          </Sidebar.Item>
        </Sidebar.ItemGroup>
      </Sidebar.Items>
    </Sidebar>
  );
}
```

Файл DashUsers.jsx

```
import { Modal, Table, Button } from 'flowbite-react';
import { useEffect, useState } from 'react';
import { useSelector } from 'react-redux';
import { HiOutlineExclamationCircle } from 'react-icons/hi';
import { FaCheck, FaTimes } from 'react-icons/fa';

export default function DashUsers() {
  const { currentUser } = useSelector((state) => state.user);
  const [users, setUsers] = useState([]);
  const [showMore, setShowMore] = useState(true);
  const [showModal, setShowModal] = useState(false);
  const [userIdToDelete, setUserIdToDelete] = useState('');
```

```
  useEffect(() => {
    const fetchUsers = async () => {
      try {
        const res = await fetch(`/api/user/getClients`);
        const data = await res.json();
        if (res.ok) {
          setUsers(data.users);
          setShowMore(data.users.length >= 9);
        }
      } catch (error) {
        console.error(error.message);
      }
    };

    if (currentUser.isAdmin) {
      fetchUsers();
    }
  }, [currentUser._id]);

  const handleShowMore = async () => {
    const startIndex = users.length;
    try {
      const res = await
fetch(`/api/user/getClients?startIndex=${startIndex}`);
      const data = await res.json();
      if (res.ok) {
        setUsers((prev) => [...prev, ...data.users]);
        setShowMore(data.users.length >= 9);
      }
    } catch (error) {
      console.error(error.message);
    }
  };

  const handleDeleteUser = async () => {
    try {
      const res = await fetch(`/api/user/delete/${userIdToDelete}`,
{
        method: 'DELETE',
      });
      const data = await res.json();
      if (res.ok) {
        setUsers((prev) => prev.filter((user) => user._id !==
userIdToDelete));
        setShowModal(false);
      } else {
        console.error(data.message);
      }
    } catch (error) {
      console.error(error.message);
    }
  };
```

```
  return (
    <div className='table-auto overflow-x-scroll md:mx-auto p-3
scrollbar scrollbar-track-slate-100 scrollbar-thumb-slate-300
dark:scrollbar-track-slate-700 dark:scrollbar-thumb-slate-500'>
      {currentUser.isAdmin && users.length > 0 ? (
        <>
          <Table hoverable className='shadow-md'>
            <Table.Head>
              <Table.HeadCell>Date created</Table.HeadCell>
              <Table.HeadCell>User image</Table.HeadCell>
              <Table.HeadCell>Username</Table.HeadCell>
              <Table.HeadCell>Email</Table.HeadCell>
              <Table.HeadCell>Admin</Table.HeadCell>
              <Table.HeadCell>Delete</Table.HeadCell>
            </Table.Head>
            {users.map((user) => (
              <Table.Body className='divide-y' key={user._id}>
                <Table.Row className='bg-white dark:border-gray-700
dark:bg-gray-800'>
                  <Table.Cell>
                    {new Date(user.createdAt).toLocaleDateString()}
                  </Table.Cell>
                  <Table.Cell>
                    <img
                      src={user.profilePicture}
                      alt={user.username}
                      className='w-10 h-10 object-cover bg-gray-500
rounded-full'
                    />
                  </Table.Cell>
                  <Table.Cell>{user.username}</Table.Cell>
                  <Table.Cell>{user.email}</Table.Cell>
                  <Table.Cell>
                    {user.isAdmin ? (
                      <FaCheck className='text-green-500' />
                    ) : (
                      <FaTimes className='text-red-500' />
                    )}
                  </Table.Cell>
                  <Table.Cell>
                    <span
                      onClick={() => {
                        setShowModal(true);
                        setUserIdToDelete(user._id);
                      }}
                      className='font-medium text-red-500
hover:underline cursor-pointer'
                    >
                      Delete
                    </span>
                  </Table.Cell>
                </Table.Row>
```

```
              </Table.Body>
            ))}
          </Table>
          {showMore && (
            <button
              onClick={handleShowMore}
              className='w-full text-teal-500 self-center text-sm
py-7'
            >
              Show more
            </button>
          )}
        </>
      ) : (
        <p>You have no users yet!</p>
      )}
      <Modal
        show={showModal}
        onClose={() => setShowModal(false)}
        popup
        size='md'
      >
        <Modal.Header />
        <Modal.Body>
          <div className='text-center'>
            <HiOutlineExclamationCircle className='h-14 w-14 text-
gray-400 dark:text-gray-200 mb-4 mx-auto' />
            <h3 className='mb-5 text-lg text-gray-500 dark:text-
gray-400'>
              Are you sure you want to delete this user?
            </h3>
            <div className='flex justify-center gap-4'>
              <Button color='failure' onClick={handleDeleteUser}>
                Yes, I'm sure
              </Button>
              <Button color='gray' onClick={() =>
setShowModal(false)}>
                No, cancel
              </Button>
            </div>
          </div>
        </Modal.Body>
      </Modal>
    </div>
  );
}
```

Файл Footer.jsx

```
import { Footer } from 'flowbite-react';
import { Link } from 'react-router-dom';
```

```
import { BsFacebook, BsInstagram, BsTwitter, BsGithub, BsDribbble }
from 'react-icons/bs';

export default function FooterCom() {
  return (
    <Footer container className='border border-t-8 border-teal-500'>
      <div className='w-full max-w-7xl mx-auto'>
        <div className='grid w-full justify-between sm:flex md:grid-
cols-1'>
          <div className='mt-5'>
            <Link
              to='/'
              className='self-center whitespace-nowrap text-lg
sm:text-xl font-semibold dark:text-white'
            >
              <span className='px-2 py-1 bg-gradient-to-r from-red-
800 to-yellow-500 rounded-lg text-white'>
                My
              </span>
              Blog
            </Link>
          </div>
          <div className='grid grid-cols-2 gap-8 mt-4 sm:grid-cols-3
sm:gap-6'>
            <div>
              <Footer.Title title='About' />
              <Footer.LinkGroup col>
                <Footer.Link
                  href='/about'
                  target='_blank'
                  rel='noopener noreferrer'
                >
                  My Blog
                </Footer.Link>
              </Footer.LinkGroup>
            </div>
            <div>
              <Footer.Title title='Follow us' />
              <Footer.LinkGroup col>
                <Footer.Link href='#'>Discord</Footer.Link>
              </Footer.LinkGroup>
            </div>
            <div>
              <Footer.Title title='Legal' />
              <Footer.LinkGroup col>
                <Footer.Link href='#'>Privacy Policy</Footer.Link>
                <Footer.Link href='#'>Terms &amp;
Conditions</Footer.Link>
              </Footer.LinkGroup>
            </div>
          </div>
        </div>
        <Footer.Divider />
```

```
        <div className='w-full sm:flex sm:items-center sm:justify-
between'>
          <Footer.Copyright
            href='#'
            by="My blog"
            year={new Date().getFullYear()}
          />
          <div className="flex gap-6 sm:mt-0 mt-4 sm:justify-
center">
            <Footer.Icon href='#' icon={BsFacebook}
label="Facebook"/>
            <Footer.Icon href='#' icon={BsInstagram}
label="Instagram"/>
            <Footer.Icon href='#' icon={BsTwitter} label="Twitter"/>
            <Footer.Icon href='#' icon={BsGithub} label="Github"/>
            <Footer.Icon href='#' icon={BsDribbble}
label="Dribbble"/>
          </div>
        </div>
      </div>
    </Footer>
  );
}
```

Файл Header.jsx

```
import { Avatar, Button, Dropdown, Navbar, TextInput } from
'flowbite-react';
import { Link, useLocation, useNavigate } from 'react-router-dom';
import { AiOutlineSearch } from 'react-icons/ai';
import { FaMoon, FaSun } from 'react-icons/fa';
import { useSelector, useDispatch } from 'react-redux';
import { toggleTheme } from '../redux/theme/themeSlice';
import { signoutSuccess } from '../redux/user/userSlice';
import { useEffect, useState } from 'react';

export default function Header() {
  const path = useLocation().pathname;
  const location = useLocation();
  const navigate = useNavigate();
  const dispatch = useDispatch();
  const { currentUser } = useSelector((state) => state.user);
  const { theme } = useSelector((state) => state.theme);
  const [searchTerm, setSearchTerm] = useState('');

  useEffect(() => {
    const urlParams = new URLSearchParams(location.search);
    const searchTermFromUrl = urlParams.get('searchTerm');
    if (searchTermFromUrl) {
      setSearchTerm(searchTermFromUrl);
    }
```

```
  }, [location.search]);

  const handleSignout = async () => {
    try {
      const res = await fetch('/api/user/signout', {
        method: 'POST',
      });
      const data = await res.json();
      if (!res.ok) {
        console.log(data.message);
      } else {
        dispatch(signoutSuccess());
      }
    } catch (error) {
      console.log(error.message);
    }
  };

  const handleSubmit = (e) => {
    e.preventDefault();
    const urlParams = new URLSearchParams(location.search);
    urlParams.set('searchTerm', searchTerm);
    const searchQuery = urlParams.toString();
    navigate(`/search?${searchQuery}`);
  };

  return (
    <Navbar className='border-b-2'>
      <Link
        to='/'
        className='self-center whitespace-nowrap text-sm sm:text-xl
font-semibold dark:text-white'
      >
        <span className='px-2 py-1 bg-gradient-to-r from-red-800 to-
yellow-500 rounded-lg text-white'>
          My
        </span>
        Blog
      </Link>
      <form onSubmit={handleSubmit}>
        <TextInput
          type='text'
          placeholder='Search...'
          rightIcon={AiOutlineSearch}
          className='hidden lg:inline'
          value={searchTerm}
          onChange={(e) => setSearchTerm(e.target.value)}
        />
      </form>
      <Button className='w-12 h-10 lg:hidden' color='gray' pill>
        <AiOutlineSearch />
      </Button>
      <div className='flex gap-2 md:order-2'>
```

```
        <Button
          className='w-12 h-10 hidden sm:inline'
          color='gray'
          pill
          onClick={() => dispatch(toggleTheme())}
        >
          {theme === 'light' ? <FaSun /> : <FaMoon />}
        </Button>
        {currentUser ? (
          <Dropdown
            arrowIcon={false}
            inline
            label={
              <Avatar alt='user' img={currentUser.profilePicture}
rounded />
            }
          >
            <Dropdown.Header>
              <span className='block text-
sm'>@{currentUser.username}</span>
              <span className='block text-sm font-medium truncate'>
                {currentUser.email}
              </span>
            </Dropdown.Header>
            <Link to={'/dashboard?tab=profile'}>
              <Dropdown.Item>Profile</Dropdown.Item>
            </Link>
            <Dropdown.Divider />
            <Dropdown.Item onClick={handleSignout}>Sign
out</Dropdown.Item>
          </Dropdown>
        ) : (
          <Link to='/sign-in'>
            <Button gradientDuoTone='redToYellow' outline>
              Sign In
            </Button>
          </Link>
        )}
        <Navbar.Toggle />
      </div>
      <Navbar.Collapse>
        <Navbar.Link active={path === '/'} as={'div'}>
          <Link to='/'>Home</Link>
        </Navbar.Link>
        <Navbar.Link active={path === '/about'} as={'div'}>
          <Link to='/about'>About</Link>
        </Navbar.Link>
        <Navbar.Link active={path === '/projects'} as={'div'}>
          <Link to='/projects'>Projects</Link>
        </Navbar.Link>
      </Navbar.Collapse>
    </Navbar>
  );
```

```
}
```

Файл OAuth.jsx

```
import { Button } from 'flowbite-react';
import { AiFillGoogleCircle } from 'react-icons/ai';
import { GoogleAuthProvider, signInWithPopup, getAuth } from
'firebase/auth';
import { app } from '../firebase';
import { useDispatch } from 'react-redux';
import { signInSuccess } from '../redux/user/userSlice';
import { useNavigate } from 'react-router-dom';

export default function OAuth() {
    const auth = getAuth(app);
    const dispatch = useDispatch();
    const navigate = useNavigate();

    const handleGoogleClick = async () => {
        const provider = new GoogleAuthProvider();
        provider.setCustomParameters({ prompt: 'select_account' });

        try {
            const resultsFromGoogle = await signInWithPopup(auth,
provider);
            const res = await fetch('/api/auth/google', {
                method: 'POST',
                headers: { 'Content-Type': 'application/json' },
                body: JSON.stringify({
                    name: resultsFromGoogle.user.displayName,
                    email: resultsFromGoogle.user.email,
                    googlePhotoUrl: resultsFromGoogle.user.photoURL,
                }),
            });
            const data = await res.json();
            if (res.ok) {
                dispatch(signInSuccess(data));
                navigate('/');
            }
        } catch (error) {
            console.log(error);
        }
    };

    return (
        <Button type='button' gradientDuoTone='pinkToOrange' outline
onClick={handleGoogleClick}>
            <AiFillGoogleCircle className='w-6 h-6 mr-2'/>
            Continue with Google
        </Button>
    );
}
```

Файл OnlyAdminPrivateRoute.jsx

```
import { useSelector } from 'react-redux';
import { Outlet, Navigate } from 'react-router-dom';

export default function OnlyAdminPrivateRoute() {
  const { currentUser } = useSelector((state) => state.user);
  return currentUser && currentUser.isAdmin ? (
    <Outlet />
  ) : (
    <Navigate to='/sign-in' />
  );
}
```

Файл PostCard.jsx

```
import { Link } from 'react-router-dom';

export default function PostCard({ post }) {
  return (
    <div className='group relative w-full border border-teal-500
hover:border-2 h-[400px] overflow-hidden rounded-lg sm:w-[430px]
transition-all'>
      <Link to={`/post/${post.slug}`}>
        <img
          src={post.image}
          alt='post cover'
          className='h-[260px] w-full object-top   object-cover
group-hover:h-[200px] transition-all duration-300 z-20'
        />
      </Link>
      <div className='p-3 flex flex-col gap-2'>
        <p className='text-lg font-semibold line-clamp-
2'>{post.title}</p>
        <span className='italic text-sm'>{post.category}</span>
        <Link
          to={`/post/${post.slug}`}
          className='z-10 group-hover:bottom-0 absolute bottom-[-
200px] left-0 right-0 border border-teal-500 text-teal-500 hover:bg-
teal-500 hover:text-white transition-all duration-300 text-center
py-2 rounded-md !rounded-tl-none m-2'
        >
          Read article
        </Link>
      </div>
    </div>
  );
}
```

Файл PrivateRoute.jsx

```
import { useSelector } from 'react-redux';
import { Outlet, Navigate } from 'react-router-dom';

export default function PrivateRoute() {
  const { currentUser } = useSelector((state) => state.user);
  return currentUser ? <Outlet /> : <Navigate to='/sign-in' />;
}
```

Файл ScrollToTop.jsx

```
import { useEffect } from 'react';
import { useLocation } from 'react-router-dom';

const ScrollToTop = () => {
  const { pathname } = useLocation();
  useEffect(() => {
    window.scrollTo(0, 0);
  }, [pathname]);
  return null;
};

export default ScrollToTop;
```

Файл ThemeProvider.jsx

```
import { useSelector } from 'react-redux';

export default function ThemeProvider({ children }) {
  const { theme } = useSelector((state) => state.theme);
  return (
    <div className={theme}>
      <div className='bg-white text-gray-700 dark:text-gray-200
dark:bg-[rgb(16,23,42)] min-h-screen'>
        {children}
      </div>
    </div>
  );
}
```

Файл About.jsx

```
export default function About() {
  return (
    <div className='min-h-screen flex items-center justify-center'>
      <div className='max-w-2xl mx-auto p-3 text-center'>
        <div>
          <h1 className='text-3xl font font-semibold text-center my-
7'>
            About My Blog
```

```
          </h1>
          <div className='text-md text-gray-500 flex flex-col gap-
6'>
            <p>
              Welcome to My Blog!
            </p>

            <p>
              London is the capital of Great Britain.
            </p>

          </div>
        </div>
      </div>
    </div>
  );
}
```

Файл CreatePost.jsx

```
import { useState } from 'react';
import { useNavigate } from 'react-router-dom';
import { Alert, Button, FileInput, Select, TextInput } from
'flowbite-react';
import ReactQuill from 'react-quill';
import 'react-quill/dist/quill.snow.css';
import { getDownloadURL, getStorage, ref, uploadBytesResumable }
from 'firebase/storage';
import { app } from '../firebase';
import { CircularProgressbar } from 'react-circular-progressbar';
import 'react-circular-progressbar/dist/styles.css';

export default function CreatePost() {
    const [file, setFile] = useState(null);
    const [imageUploadProgress, setImageUploadProgress] =
useState(null);
    const [imageUploadError, setImageUploadError] = useState(null);
    const [formData, setFormData] = useState({});
    const [publishError, setPublishError] = useState(null);
    const navigate = useNavigate();

    const handleUploadImage = async () => {
        try {
            if (!file) {
                setImageUploadError('Please select an image');
                return;
            }
            setImageUploadError(null);
            const storage = getStorage(app);
            const fileName = new Date().getTime() + '-' + file.name;
            const storageRef = ref(storage, fileName);
```

```
            const uploadTask = uploadBytesResumable(storageRef,
file);
            uploadTask.on(
                'state_changed',
                (snapshot) => {
                    const progress = (snapshot.bytesTransferred /
snapshot.totalBytes) * 100;
                    setImageUploadProgress(progress.toFixed(0));
                },
                (error) => {
                    setImageUploadError('Image upload failed');
                    setImageUploadProgress(null);
                },
                () => {

getDownloadURL(uploadTask.snapshot.ref).then((downloadURL) => {
                        setImageUploadProgress(null);
                        setImageUploadError(null);
                        setFormData({ ...formData, image:
downloadURL });
                    });
                }
            );
        } catch (error) {
            setImageUploadError('Image upload failed');
            setImageUploadProgress(null);
            console.log(error);
        }
    };

    const handleSubmit = async (e) => {
        e.preventDefault();
        try {
            const res = await fetch('/api/post/create', {
                method: 'POST',
                headers: {
                    'Content-Type': 'application/json',
                },
                body: JSON.stringify(formData),
            });
            const data = await res.json();
            if (!res.ok) {
                setPublishError(data.message);
                return;
            }
            if (res.ok) {
                setPublishError(null);
                navigate(`/post/${data.slug}`);
            }
        } catch (error) {
            setPublishError('Something went wrong');
        }
    };
```

```
    return (
        <div className='p-3 max-w-3xl mx-auto min-h-screen'>
            <h1 className='text-center text-3xl my-7 font-
semibold'>Create a post</h1>
            <form className='flex flex-col gap-4'
onSubmit={handleSubmit}>
                <div className='flex flex-col gap-4 sm:flex-row
justify-between'>
                    <TextInput
                        type='text'
                        placeholder='Title'
                        required
                        id='title'
                        className='flex-1'
                        onChange={(e) =>
                            setFormData({ ...formData, title:
e.target.value })
                        }
                    />
                    <Select
                        onChange={(e) =>
                            setFormData({ ...formData, category:
e.target.value })
                        }
                    >
                        <option value='uncategorized'>Select a
category</option>
                        <option value='work'>Work</option>
                        <option value='myself'>Myself</option>
                        <option value='dayOff'>Day off</option>
                    </Select>
                </div>
                <div className='flex gap-4 items-center justify-
between border-4 border-teal-500 border-dotted p-3'>
                    <FileInput
                        type='file'
                        accept='image/*'
                        onChange={(e) => setFile(e.target.files[0])}
                    />
                    <Button
                        type='button'
                        gradientDuoTone='redToYellow'
                        size='sm'
                        outline
                        onClick={handleUploadImage}
                        disabled={imageUploadProgress}
                    >
                        {imageUploadProgress ? (
                            <div className='w-16 h-16'>
                                <CircularProgressbar
                                    value={imageUploadProgress}
```

```
                                        text={`${imageUploadProgress ||
0}%`}
                                    />
                                </div>
                            ) : (
                                'Upload Image'
                            )}
                        </Button>
                    </div>
                    {imageUploadError && <Alert
color='failure'>{imageUploadError}</Alert>}
                    {formData.image && (
                        <img
                            src={formData.image}
                            alt='upload'
                            className='w-full h-72 object-cover'
                        />
                    )}
                    <ReactQuill
                        theme='snow'
                        placeholder='Write something...'
                        className='h-72 mb-12'
                        required
                        onChange={(value) => {
                            setFormData({ ...formData, content: value
});
                        }}
                    />
                    <Button type='submit' gradientDuoTone='redToYellow'>
                        Publish
                    </Button>
                    {publishError && (
                        <Alert className='mt-5' color='failure'>
                            {publishError}
                        </Alert>
                    )}
                </form>
            </div>
        );
}
```

Файл Dashboard.jsx

```
import { useEffect, useState } from 'react';
import { useLocation } from 'react-router-dom';
import DashSidebar from '../components/DashSidebar';
import DashProfile from '../components/DashProfile';
import DashPosts from '../components/DashPosts';
import DashUsers from '../components/DashUsers';
import DashComments from '../components/DashComments';
import DashboardComp from '../components/DashboardComp';
```

```
export default function Dashboard() {
  const location = useLocation();
  const [tab, setTab] = useState('');
  useEffect(() => {
    const urlParams = new URLSearchParams(location.search);
    const tabFromUrl = urlParams.get('tab');
    if (tabFromUrl) {
      setTab(tabFromUrl);
    }
  }, [location.search]);
  return (
    <div className='min-h-screen flex flex-col md:flex-row'>
      <div className='md:w-56'>
        {/* Sidebar */}
        <DashSidebar />
      </div>
      {/* profile... */}
      {tab === 'profile' && <DashProfile />}
      {/* posts... */}
      {tab === 'posts' && <DashPosts />}
      {/* users */}
      {tab === 'users' && <DashUsers />}
      {/* comments  */}
      {tab === 'comments' && <DashComments />}
      {/* dashboard comp */}
      {tab === 'dash' && <DashboardComp />}
    </div>
  );
}
```

Файл Home.jsx

```
import { Link } from 'react-router-dom';
import { useEffect, useState } from 'react';
import PostCard from '../components/PostCard';

export default function Home() {
  const [posts, setPosts] = useState([]);

  useEffect(() => {
    const fetchPosts = async () => {
      const res = await fetch('/api/post/getPosts');
      const data = await res.json();
      setPosts(data.posts);
    };
    fetchPosts();
  }, []);
  return (
    <div>
      <div className='flex flex-col gap-6 p-28 px-3 max-w-6xl mx-
auto '>
```

```
        <h1 className='text-3xl font-bold lg:text-6xl'>Welcome to my
Blog</h1>
        <p className='text-gray-500 text-xs sm:text-sm'>
          Here you can see all my life
        </p>
        <Link
          to='/search'
          className='text-xs sm:text-sm text-teal-500 font-bold
hover:underline'
        >
          View all posts
        </Link>
      </div>
      <div className='max-w-6xl mx-auto p-3 flex flex-col gap-8 py-
7'>
        {posts && posts.length > 0 && (
          <div className='flex flex-col gap-6'>
            <h2 className='text-2xl font-semibold text-
center'>Recent Posts</h2>
            <div className='flex flex-wrap gap-4'>
              {posts.map((post) => (
                <PostCard key={post._id} post={post} />
              ))}
            </div>
            <Link
              to={'/search'}
              className='text-lg text-teal-500 hover:underline text-
center'
            >
              View all posts
            </Link>
          </div>
        )}
      </div>
    </div>
  );
}
```

Файл PostPage.jsx

```
import { Button, Spinner } from 'flowbite-react';
import { useEffect, useState } from 'react';
import { Link, useParams } from 'react-router-dom';
import CommentSection from '../components/CommentSection';
import PostCard from '../components/PostCard';

export default function PostPage() {
  const { postSlug } = useParams();
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(false);
  const [post, setPost] = useState(null);
```

```
  const [recentPosts, setRecentPosts] = useState(null);

  useEffect(() => {
    const fetchPost = async () => {
      try {
        setLoading(true);
        const res = await
fetch(`/api/post/getposts?slug=${postSlug}`);
        const data = await res.json();
        if (!res.ok) {
          setError(true);
          setLoading(false);
          return;
        }
        if (res.ok) {
          setPost(data.posts[0]);
          setLoading(false);
          setError(false);
        }
      } catch (error) {
        setError(true);
        setLoading(false);
      }
    };
    fetchPost();
  }, [postSlug]);

  useEffect(() => {
    const fetchRecentPosts = async () => {
      try {
        const res = await fetch(`/api/post/getposts?limit=3`);
        const data = await res.json();
        if (res.ok) {
          setRecentPosts(data.posts);
        }
      } catch (error) {
        console.log(error.message);
      }
    };
    fetchRecentPosts();
  }, []);

  if (loading)
    return (
      <div className='flex justify-center items-center min-h-
screen'>
        <Spinner size='xl' />
      </div>
    );

  return (
    <main className='p-3 flex flex-col max-w-6xl mx-auto min-h-
screen'>
```

```
      {post && (
        <>
          <h1 className='text-3xl mt-10 p-3 text-center font-serif
max-w-2xl mx-auto lg:text-4xl'>
            {post.title}
          </h1>
          <Link
            to={`/search?category=${post.category}`}
            className='self-center mt-5'
          >
            <Button color='gray' pill size='xs'>
              {post.category}
            </Button>
          </Link>
          <img
            src={post.image}
            alt={post.title}
            className='ml-auto mr-auto p-3 max-w-[400px] w-full
object-cover '
          />
          <div className='flex justify-between p-3 border-b border-
slate-500 mx-auto w-full max-w-2xl text-xs'>
            <span>{new
Date(post.createdAt).toLocaleDateString()}</span>
            <span className='italic'>
              {(post.content.length / 1000).toFixed(0)} mins read
            </span>
          </div>
          <div
            className='p-3 max-w-2xl mx-auto w-full post-content'
            dangerouslySetInnerHTML={{ __html: post.content }}
          ></div>
          <CommentSection postId={post._id} />
        </>
      )}

      {recentPosts?.length > 0 && (
        <div className='flex flex-col justify-center items-center
mb-5'>
          <h1 className='text-xl mt-5'>Recent articles</h1>
          <div className='flex flex-wrap gap-5 mt-5 justify-center'>
            {recentPosts.map((post) => (
              <PostCard key={post._id} post={post} />
            ))}
          </div>
        </div>
      )}
    </main>
  );
}
```

Файл Projects.jsx

```
export default function Projects() {
  return (
    <div className="min-h-screen max-w-2xl mx-auto flex justify-
center items-center flex-col gap-6 p-3">
      <h1 className="text-3xl font-semibold">Projects</h1>
      <p className="text-md text-gray-500">
        Currently, there are no new projects available.
      </p>
    </div>
  );
}
```

Файл Search.jsx

```
export default function Search() {
  const [sidebarData, setSidebarData] = useState({
    searchTerm: '',
    sort: 'desc',
    category: 'uncategorized',
  });
  const [posts, setPosts] = useState([]);
  const [loading, setLoading] = useState(false);
  const [showMore, setShowMore] = useState(false);
  const location = useLocation();
  const navigate = useNavigate();

  useEffect(() => {
    const urlParams = new URLSearchParams(location.search);
    const searchTermFromUrl = urlParams.get('searchTerm');
    const sortFromUrl = urlParams.get('sort');
    const categoryFromUrl = urlParams.get('category');
    if (searchTermFromUrl || sortFromUrl || categoryFromUrl) {
      setSidebarData({
        ...sidebarData,
        searchTerm: searchTermFromUrl,
        sort: sortFromUrl,
        category: categoryFromUrl,
      });
    }
    fetchPosts(urlParams);
  }, [location.search]);

  const fetchPosts = async (urlParams) => {
    try {
      setLoading(true);
      const res = await
fetch(`/api/post/getposts?${urlParams.toString()}`);
      if (!res.ok) throw new Error('Failed to fetch posts');
      const data = await res.json();
      setPosts(data.posts);
      setLoading(false);
      setShowMore(data.posts.length === 9);
```

```
    } catch (error) {
      console.error(error);
      setLoading(false);
    }
  };

  const handleChange = (e) => {
    const { id, value } = e.target;
    setSidebarData({ ...sidebarData, [id]: value });
  };

  const handleSubmit = (e) => {
    e.preventDefault();
    const urlParams = new URLSearchParams(location.search);
    for (const key in sidebarData) {
      urlParams.set(key, sidebarData[key]);
    }
    navigate(`/search?${urlParams.toString()}`);
  };

  const handleShowMore = async () => {
    try {
      const numberOfPosts = posts.length;
      const urlParams = new URLSearchParams(location.search);
      urlParams.set('startIndex', numberOfPosts);
      const res = await
fetch(`/api/post/getposts?${urlParams.toString()}`);
      if (!res.ok) throw new Error('Failed to fetch more posts');
      const data = await res.json();
      setPosts([...posts, ...data.posts]);
      setShowMore(data.posts.length === 9);
    } catch (error) {
      console.error(error);
    }
  };

  return (
    <div className='flex flex-col md:flex-row'>
      <div className='p-7 border-b md:border-r md:min-h-screen
border-gray-500'>
        <form className='flex flex-col gap-8'
onSubmit={handleSubmit}>
          {/* Form inputs */}
        </form>
      </div>
      <div className='w-full'>
        {/* Search results */}
      </div>
    </div>
  );
}
```

Файл SignIn.jsx

```
export default function SignIn() {
  const [formData, setFormData] = useState({});
  const { loading, error: errorMessage } = useSelector((state) =>
state.user);
  const dispatch = useDispatch();
  const navigate = useNavigate();

  const handleChange = (e) => {
    const { id, value } = e.target;
    setFormData(prevData => ({ ...prevData, [id]: value.trim() }));
  };

  const handleSubmit = async (e) => {
    e.preventDefault();
    const { email, password } = formData;
    if (!email || !password) {
      return dispatch(signInFailure('Please fill all the fields'));
    }

    dispatch(signInStart());

    try {
      const res = await fetch('/api/auth/signin', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify(formData),
      });
      const data = await res.json();

      if (!res.ok) {
        return dispatch(signInFailure(data.message));
      }

      dispatch(signInSuccess(data));
      navigate('/');
    } catch (error) {
      dispatch(signInFailure(error.message));
    }
  };

  return (
    <div className='min-h-screen mt-20'>
      <div className='flex p-3 max-w-3xl mx-auto flex-col md:flex-
row md:items-center gap-5'>
        {/* Left section */}
        {/* Right section */}
        <div className='flex-1'>
          <form className='flex flex-col gap-4'
onSubmit={handleSubmit}>
            <div>
```

```
              <Label value='Your email' />
              <TextInput
                type='email'
                placeholder='name@company.com'
                id='email'
                onChange={handleChange}
                required
              />
            </div>
            <div>
              <Label value='Your password' />
              <TextInput
                type='password'
                placeholder='**********'
                id='password'
                onChange={handleChange}
                required
              />
            </div>
            <Button gradientDuoTone='redToYellow' type='submit'
disabled={loading}>
              {loading ? <><Spinner size='sm' /><span className='pl-
3'>Loading...</span></> : 'Sign In'}
            </Button>
            <OAuth />
          </form>
          <div className='flex gap-2 text-sm mt-5'>
            <span>Don't have an account?</span>
            <Link to='/sign-up' className='text-blue-500'>Sign
Up</Link>
          </div>
          {errorMessage && (
            <Alert className='mt-5' color='failure'>
              {errorMessage}
            </Alert>
          )}
        </div>
      </div>
    </div>
  );
}
```

Файл SignUp.jsx

```
export default function SignUp() {
  const [formData, setFormData] = useState({});
  const [errorMessage, setErrorMessage] = useState(null);
  const [loading, setLoading] = useState(false);
  const navigate = useNavigate();

  const handleChange = (e) => {
    const { id, value } = e.target;
    setFormData(prevData => ({ ...prevData, [id]: value.trim() }));
```

```
  };

  const handleSubmit = async (e) => {
    e.preventDefault();
    const { username, email, password } = formData;
    if (!username || !email || !password) {
      return setErrorMessage('Please fill out all fields.');
    }

    setLoading(true);
    setErrorMessage(null);

    try {
      const res = await fetch('/api/auth/signup', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify(formData),
      });
      const data = await res.json();

      if (!res.ok) {
        return setErrorMessage(data.message);
      }

      setLoading(false);
      navigate('/sign-in');
    } catch (error) {
      setErrorMessage(error.message);
      setLoading(false);
    }
  };

  return (
    <div className='min-h-screen mt-20'>
      <div className='flex p-3 max-w-3xl mx-auto flex-col md:flex-
row md:items-center gap-5'>
        {/* left */}
        <div className='flex-1'>
          <Link to='/' className='font-bold dark:text-white text-
4xl'>
            <span className='px-2 py-1 bg-gradient-to-r from-red-800
to-yellow-500 rounded-lg text-white'>
              My
            </span>
            Blog
          </Link>
          <p className='text-sm mt-5'>
            Welcome to my blog!
          </p>
        </div>
        {/* right */}
        <div className='flex-1'>
```

```
          <form classNameName='flex flex-col gap-4'
onSubmit={handleSubmit}>
            <div>
              <Label value='Your username' />
              <TextInput
                type='text'
                placeholder='Username'
                id='username'
                onChange={handleChange}
                required
              />
            </div>
            <div>
              <Label value='Your email' />
              <TextInput
                type='email'
                placeholder='name@company.com'
                id='email'
                onChange={handleChange}
                required
              />
            </div>
            <div>
              <Label value='Your password' />
              <TextInput
                type='password'
                placeholder='Password'
                id='password'
                onChange={handleChange}
                required
              />
            </div>
            <Button
              gradientDuoTone='redToYellow'
              type='submit'
              disabled={loading}
            >
              {loading ? (
                <>
                  <Spinner size='sm' />
                  <span classNameName='pl-3'>Loading...</span>
                </>
              ) : (
                'Sign Up'
              )}
            </Button>
            <OAuth />
          </form>
          <div classNameName='flex gap-2 text-sm mt-5'>
            <span>Have an account?</span>
            <Link to='/sign-in' classNameName='text-blue-500'>Sign
In</Link>
          </div>
```

```
          {errorMessage && (
            <Alert classNam='mt-5' color='failure'>
              {errorMessage}
            </Alert>
          )}
        </div>
      </div>
    </div>
  );
}
```

Файл UpdatePost.jsx

```
export default function UpdatePost() {
  const [file, setFile] = useState(null);
  const [imageUploadProgress, setImageUploadProgress] =
useState(null);
  const [imageUploadError, setImageUploadError] = useState(null);
  const [formData, setFormData] = useState({});
  const [publishError, setPublishError] = useState(null);
  const { postId } = useParams();
  const navigate = useNavigate();
  const { currentUser } = useSelector((state) => state.user);

  useEffect(() => {
    try {
      const fetchPost = async () => {
        const res = await
fetch(`/api/post/getposts?postId=${postId}`);
        const data = await res.json();
        if (!res.ok) {
          setPublishError(data.message);
          return;
        }
        setPublishError(null);
        setFormData(data.posts[0]);
      };
      fetchPost();
    } catch (error) {
      console.log(error.message);
    }
  }, [postId]);

  const handleChange = (e) => {
    const { id, value } = e.target;
    setFormData((prevData) => ({ ...prevData, [id]: value.trim()
}));
  };

  const handleUpdloadImage = async () => {
    try {
      if (!file) {
```

```
        setImageUploadError('Please select an image');
        return;
      }
      setImageUploadError(null);
      const storage = getStorage(app);
      const fileName = new Date().getTime() + '-' + file.name;
      const storageRef = ref(storage, fileName);
      const uploadTask = uploadBytesResumable(storageRef, file);
      uploadTask.on(
        'state_changed',
        (snapshot) => {
          const progress = (snapshot.bytesTransferred /
snapshot.totalBytes) * 100;
          setImageUploadProgress(progress.toFixed(0));
        },
        (error) => {
          setImageUploadError('Image upload failed');
          setImageUploadProgress(null);
        },
        () => {
          getDownloadURL(uploadTask.snapshot.ref).then((downloadURL)
=> {
            setImageUploadProgress(null);
            setImageUploadError(null);
            setFormData((prevData) => ({ ...prevData, image:
downloadURL }));
          });
        }
      );
    } catch (error) {
      setImageUploadError('Image upload failed');
      setImageUploadProgress(null);
      console.log(error);
    }
  };

  const handleSubmit = async (e) => {
    e.preventDefault();
    try {
      const res = await
fetch(`/api/post/updatepost/${formData._id}/${currentUser._id}`, {
        method: 'PUT',
        headers: {
          'Content-Type': 'application/json',
        },
        body: JSON.stringify(formData),
      });
      const data = await res.json();
      if (!res.ok) {
        setPublishError(data.message);
        return;
      }
      setPublishError(null);
```

```
      navigate(`/post/${data.slug}`);
    } catch (error) {
      setPublishError('Something went wrong');
    }
  };

  return (
    <div className='p-3 max-w-3xl mx-auto min-h-screen'>
      <h1 className='text-center text-3xl my-7 font-semibold'>Update
post</h1>
      <form className='flex flex-col gap-4' onSubmit={handleSubmit}>
        {/* Form fields */}
        <div className='flex flex-col gap-4 sm:flex-row justify-
between'>
          {/* Title input */}
          <TextInput
            type='text'
            placeholder='Title'
            required
            id='title'
            className='flex-1'
            onChange={handleChange}
            value={formData.title}
          />
          {/* Category select */}
          <Select
            onChange={handleChange}
            value={formData.category}
            id='category'
          >
            <option value='uncategorized'>Select a category</option>
            <option value='work'>Work</option>
            <option value='myself'>Myself</option>
            <option value='dayOff'>Day off</option>
          </Select>
        </div>
        {/* Image upload */}
        <div className='flex gap-4 items-center justify-between
border-4 border-teal-500 border-dotted p-3'>
          <FileInput
            type='file'
            accept='image/*'
            onChange={(e) => setFile(e.target.files[0])}
          />
          <Button
            type='button'
            gradientDuoTone='redToYellow'
            size='sm'            outline
            onClick={handleUpdloadImage}
            disabled={imageUploadProgress}
          >
            {imageUploadProgress ? (
              <div className='w-16 h-16'>
```

```
                <CircularProgressbar
                  value={imageUploadProgress}
                  text={`${imageUploadProgress || 0}%`}
                />
              </div>
            ) : (
              'Upload Image'
            )}
          </Button>
        </div>
        {/* Image upload error */}
        {imageUploadError && <Alert
color='failure'>{imageUploadError}</Alert>}
        {/* Display uploaded image */}
        {formData.image && (
          <img
            src={formData.image}
            alt='upload'
            className='w-full h-72 object-cover'
          />
        )}
        {/* Content editor */}
        <ReactQuill
          theme='snow'
          value={formData.content}
          placeholder='Write something...'
          className='h-72 mb-12'
          required
          onChange={(value) => {
            setFormData({ ...formData, content: value });
          }}
        />
        {/* Submit button */}
        <Button type='submit' gradientDuoTone='redToYellow'>
          Update post
        </Button>
        {/* Display publish error */}
        {publishError && (
          <Alert className='mt-5' color='failure'>
            {publishError}
          </Alert>
        )}
      </form>
    </div>
  );
}
```

Файл themeSlice.js

```
import {createSlice} from '@reduxjs/toolkit';

const initialState = {
```

```
    theme: 'light',
};

const themeSlice = createSlice({
    name: 'theme',
    initialState,
    reducers: {
        toggleTheme: (state) => {
            state.theme = state.theme === 'light' ? 'dark' :
'light';
        },
        }
});

export const {toggleTheme} = themeSlice.actions;

export default themeSlice.reducer;
```

Файл userSlice.js

```
import { createSlice } from '@reduxjs/toolkit';

const initialState = {
  currentUser: null,
  error: null,
  loading: false,
};

const userSlice = createSlice({
  name: 'user',
  initialState,
  reducers: {
    signInStart: (state) => {
      state.loading = true;
      state.error = null;
    },
    signInSuccess: (state, action) => {
      state.currentUser = action.payload;
      state.loading = false;
      state.error = null;
    },
    signInFailure: (state, action) => {
      state.loading = false;
      state.error = action.payload;
    },
    updateStart: (state) => {
      state.loading = true;
      state.error = null;
    },
    updateSuccess: (state, action) => {
      state.currentUser = action.payload;
      state.loading = false;
```

```
      state.error = null;
    },
    updateFailure: (state, action) => {
      state.loading = false;
      state.error = action.payload;
    },
    deleteUserStart: (state) => {
      state.loading = true;
      state.error = null;
    },
    deleteUserSuccess: (state) => {
      state.currentUser = null;
      state.loading = false;
      state.error = null;
    },
    deleteUserFailure: (state, action) => {
      state.loading = false;
      state.error = action.payload;
    },
    signoutSuccess: (state) => {
      state.currentUser = null;
      state.error = null;
      state.loading = false;
    },
  },
});

export const {
  signInStart,
  signInSuccess,
  signInFailure,
  updateStart,
  updateSuccess,
  updateFailure,
  deleteUserStart,
  deleteUserSuccess,
  deleteUserFailure,
  signoutSuccess,
} = userSlice.actions;

export default userSlice.reducer;
```

Файл store.js

```
import { configureStore, combineReducers } from '@reduxjs/toolkit';
import userReducer from './user/userSlice';
import themeReducer from './theme/themeSlice';
import { persistReducer, persistStore } from 'redux-persist';
import storage from 'redux-persist/lib/storage';

const rootReducer = combineReducers({
```

```
  user: userReducer,
  theme: themeReducer,
});

const persistConfig = {
  key: 'root',
  storage,
  version: 1,
};

const persistedReducer = persistReducer(persistConfig, rootReducer);

export const store = configureStore({
  reducer: persistedReducer,
  middleware: (getDefaultMiddleware) =>
    getDefaultMiddleware({ serializableCheck: false }),
});

export const persistor = persistStore(store);
```

Файл App.jsx

```
import { BrowserRouter, Routes, Route } from 'react-router-dom';
import Home from './pages/Home';
import About from './pages/About';
import SignIn from './pages/SignIn';
import Dashboard from './pages/Dashboard';
import Projects from './pages/Projects';
import SignUp from './pages/SignUp';
import Header from './components/Header';
import Footer from './components/Footer';
import PrivateRoute from './components/PrivateRoute';
import OnlyAdminPrivateRoute from
'./components/OnlyAdminPrivateRoute';
import CreatePost from './pages/CreatePost';
import UpdatePost from './pages/UpdatePost';
import PostPage from './pages/PostPage';
import ScrollToTop from './components/ScrollToTop';
import Search from './pages/Search';

export default function App() {
  return (
    <BrowserRouter>
      <ScrollToTop />
      <Header />
      <Routes>
        <Route path='/' element={<Home />} />
        <Route path='/about' element={<About />} />
        <Route path='/sign-in' element={<SignIn />} />
        <Route path='/sign-up' element={<SignUp />} />
        <Route path='/search' element={<Search />} />
```

```
        <Route element={<PrivateRoute />}>
          <Route path='/dashboard' element={<Dashboard />} />
        </Route>
        <Route element={<OnlyAdminPrivateRoute />}>
          <Route path='/create-post' element={<CreatePost />} />
          <Route path='/update-post/:postId' element={<UpdatePost
/>} />
        </Route>

        <Route path='/projects' element={<Projects />} />
        <Route path='/post/:postSlug' element={<PostPage />} />
      </Routes>
      <Footer />
    </BrowserRouter>
  );
}
```

Файл firebase.js

```
// Import the functions you need from the SDKs you need
import { initializeApp } from "firebase/app";
// TODO: Add SDKs for Firebase products that you want to use
// https://firebase.google.com/docs/web/setup#available-libraries

// Your web app's Firebase configuration
// For Firebase JS SDK v7.20.0 and later, measurementId is optional
const firebaseConfig = {
  apiKey: "XXX",
  authDomain: "blog-20167.firebaseapp.com",
  projectId: "blog-20167",
  storageBucket: "blog-20167.appspot.com",
  messagingSenderId: "834642137705",
  appId: "1:834642137705:web:a825fcb123f9d9a76e88f8",
  measurementId: "G-EBLQ41V83S"
};

// Initialize Firebase
export const app = initializeApp(firebaseConfig);
```

Файл index.css

```
@tailwind base;
@tailwind components;
@tailwind utilities;

body {
  height: 100vh;
}

.ql-editor {
  font-size: 1.05rem;
```

```
}

.post-content p {
  margin-bottom: 0.5rem;
}

.post-content h1 {
  font-size: 1.5rem;
  font-weight: 600;
  font-family: sans-serif;
  margin: 1.5rem 0;
}

.post-content h2 {
  font-size: 1.4rem;
  font-family: sans-serif;
  margin: 1.5rem 0;
}

.post-content a {
  color: rgb(73, 149, 199);
  text-decoration: none;
}

.post-content a:hover {
  text-decoration: underline;
}

.dark .post-content a {
  color: red;
}
```

Файл main.jsx

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import App from './App.jsx';
import './index.css';
import { store, persistor } from './redux/store.js';
import { Provider } from 'react-redux';
import { PersistGate } from 'redux-persist/integration/react';
import ThemeProvider from './components/ThemeProvider.jsx';

ReactDOM.createRoot(document.getElementById('root')).render(
  <PersistGate persistor={persistor}>
    <Provider store={store}>
      <ThemeProvider>
        <App />
      </ThemeProvider>
    </Provider>
  </PersistGate>
```

```
);
```

Додаток Б. Копії документів про апробацію результатів роботи

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний університет «Одеська юридична академія»

Наукове товариство молодих вчених, аспірантів та студентів

УКРАЇНСЬКЕ СУСПІЛЬСТВО ТА НАУКА В УМОВАХ ВОЄННОГО СТАНУ Й ЄВРОІНТЕГРАЦІЙНИХ ПЕРЕТВОРЕНЬ: ВИКЛИКИ, НАПРЯМИ ВІДНОВЛЕННЯ І РОЗВИТКУ

МАТЕРІАЛИ
Всеукраїнської науково-практичної конференції
здобувачів вищої освіти

Одеса
20 травня 2023 року

У двох томах

Том 2

Видавництво
«Юридика»
2023

ТІТІЄВСЬКИЙ ВІТАЛІЙ ОЛЕГОВИЧ

студент 3-го курсу факультету кібербезпеки
та інформаційних технологій
Національного університету «Одеська юридична академія»

ТЕХНІКИ ТЕСТ-ДИЗАЙНУ

З кожним роком кількість «мобільних» користувачів всесвітньої павутини зростає. Згідно з дослідженням [1], зараз у світі приблизно 6,567 млрд людей користується смартфонами. Не дивно, що спілкування, робота і навіть набридлива реклама перенеслися в телефон, а разом з цим у телефон перенеслись і комерційні проекти. Адже набагато легше купувати речі, не виходячи з власної домівки. Тому успішність бізнесу комерційних підприємців багато в чому залежить від якості, функціональності та зручності власного вебсайту та/або вебзастосунку.

При створенні програмного продукту важливо не лише його розробити, а й протестувати на предмет дотримання функціональних вимог, інтуїтивно зрозумілого і зручного інтерфейсу, стабільності, безпеки тощо.

Створення тест-кейсів є невід'ємною складовою роботи будь-якого тестувальника. Для підготовки ефективних тест-кейсів використовуються спеціальні техніки тестування.

Техніки тест-дизайну – це стратегії, які полягають у можливості підходити до тестування систематично та зосереджено, забезпечуючи широке покриття вимог [2]. Існують численні техніки тест-дизайну, найпопулярнішими з яких є [3]:

- еквівалентний поділ (Equivalence partitioning, EP) – тестування програмного забезпечення розділяє вхідну область програми на класи даних, з яких повинні бути розроблені тест-кейси. Так, для діапазону припустимих числових показників від 5 до 15 варто вибрати один коректний показник з діапазону, наприклад, 10, й один некоректний показник за межами діапазону, наприклад, 3;
- аналіз граничних значень (Boundary value analysis, BVA) – включає перевірку максимально допустимих, мінімально допустимих, внутрішніх та зовнішніх меж, стандартних значень та можливих значень похибок. Якщо використати приклад, згаданий вище, як величини для позитивного тестування, то мінімальною і максимальною межами будуть 5 і 15, а значеннями більшим і меншим меж – 4 і 16. Аналіз граничних значень може бути застосований до будь-яких обмежень, що маються в записах, полях, файлах або будь-якому типі сутностей;
- попарне тестування (Pairwise testing) – це методика, що дозволяє створювати набір тестових даних з повного набору вхідних даних у системі з мінімізацією кількості тест-кейсів, при цьому не втрачаючи якість тестування;
- тестування за допомогою таблиць прийняття рішень – це одна з технік тест-дизайну методом чорної скриньки, що стосується

динамічного аналізу. Вона також відома як таблиця причинно-наслідкових зв'язків, а схематична демонстрація логіки відома як причинно-наслідковий графік. Це візуальне подання використовується для отримання таблиці рішень.

Наприклад, для того, щоб перевірити правильність поведінки роботи системи при введенні різної кількості символів у паролі, доцільно використовувати метод аналізу граничних значень. Якщо у функціональних вимогах до довжини паролів для автентифікації зазначено межі від 6 до 20 символів, то варто при тестуванні граничних значень взяти такі значення:

- кількість символів менше граничного значення – 5;
- кількість символів для нижньої межі граничного значення – 6;
- кількість символів для верхньої межі граничного значення – 20;
- кількість символів, що більше граничного значення, – 21.

У табл. 1 подано тест-кейси, створені на основі аналізу граничних значень.

Для створення повної й ефективної стратегії тестування варто комбінувати техніки тест-дизайну. Кожна техніка має свої переваги та недоліки, комбінування декількох технік дозволяє отримувати повний опис можливих дефектів.

Таблиця 1

Вигляд тест-кейсів

Відкрити сайт https://www.staff-clothes.com/ua/	1. Зайти на сайт https://www.staff-clothes.com/ua/. 2. Натиснути на іконку «чоловічка» у правому верхньому куті екрану. 3. Відкрити форму реєстрації. 4. У полі «Номер телефону» ввести «+380964580901». 5. У полі «ПІБ» ввести «Іванов Іван Іванович». 6. У полі пароль ввести «12345». 7. Натиснути на кнопку «Отримати код».	Система вкаже на помилку, оскільки кількість символів менше граничного значення (5 символів)
Відкрити сайт https://www.staff-clothes.com/ua/	1. Зайти на сайт https://www.staff-clothes.com/ua/. 2. Натиснути Відкрити на іконку «чоловічка» у правому верхньому куті екрану. 3. Відкрити форму реєстрації. 4. У полі «Номер телефону» ввести «+380964580901». 5. У полі «ПІБ» ввести «Іванов Іван Іванович». 6. У полі пароль ввести «123456». 7. Натиснути на кнопку «Отримати код».	Система дозволить продовжити реєстрацію, оскільки кількість символів еквівалентне нижній межі граничного значення (6 символів)

Закінчення таблиці 1

Відкрити сайт https://www.staff-clothes.com/ua/	1. Зайти на сайт https://www.staff-clothes.com/ua/. 2. Натиснути на іконку «чоловічка» у правому верхньому куті екрану. 3. Відкрити форму реєстрації. 4. У полі «Номер телефону» ввести «+380964580901». 5. У полі «ПІБ» ввести «Іванов Іван Іванович». 6. В полі пароль ввести «12345678901234567890». Натиснути на кнопку «Отримати код».	Система дозволить продовжити реєстрацію, оскільки кількість символів еквівалентне верхній межі граничного значення (20 символів)
Відкрити сайт https://www.staff-clothes.com/ua/	1. Зайти на сайт https://www.staff-clothes.com/ua/. 2. Натиснути на іконку «чоловічка» у правому верхньому куті екрану. 3. Відкрити форму реєстрації. 4. У полі «Номер телефону» ввести «+380964580901». 5. У полі «ПІБ» ввести «Іванов Іван Іванович». 6. У полі пароль ввести «123456789012345678901». 7. Натиснути на кнопку «Отримати код».	Система вкаже на помилку, оскільки кількість символів більше граничного значення (21 символ)

Техніки тест-дизайну в руках кваліфікованого спеціаліста стають потужним інструментом виявлення дефектів. Отже, тестування є критично важливим методом перевірки, що займає велику частину ресурсів проєкту.

Список використаних джерел

1. How many people own smartphones. URL: https://explodingtopics.com/blog/smartphone-stats#smartphone-top-picks
2. Test design techniques QA Engineers should know. URL: http://www.qamadness.com/5-test-design-techniques-qa-engineers-should-know/
3. Software testing techniques with test case design examples. URL: https://www-guru99-com.translate.goog/software-testing-techniques.html?_x_tr_sl=en&_x_tr_tl=ru&_x_tr_hl=ru&_x_tr_pto=sc

Ключові слова: тест-дизайн, тестування, тестовий випадок.
Key words: test design, testing, test case.

Науковий керівник: к. т. н., доцент Трофименко О. Г.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Одеська юридична академія»
Факультет кібербезпеки та інформаційних технологій
Кафедра кібербезпеки

КІБЕРБЕЗПЕКА В СУЧАСНОМУ СВІТІ: АКТУАЛЬНІ ВИКЛИКИ

МАТЕРІАЛИ IV ВСЕУКРАЇНСЬКОЇ НАУКОВО-ПРАКТИЧНОЇ КОНФЕРЕНЦІЇ

25 листопада 2022 року, м. Одеса

Одеса, 2022

Список використаних джерел:

1. White Box криптография. URL: https://b-it.com.ua/ua/white-box-kriptografiya. (Дата звернення 18.10.2022).
2. White Box Cryptography - Everything You Need to Know. URL: https://www.appknox.com/blog/white-box-cryptography/. (Дата звернення 18.10.2022).
3. WBC: protecting cryptographic keys in software applications. URL: http://www.whiteboxcrypto.com/. (Дата звернення 18.10.2022).
4. White Box Cryptography. URL: https://cpl.thalesgroup.com/software-monetization/white-box-cryptography. (Дата звернення 18.10.2022).

Ключові слова: White Box, Black Box, Gray Box, спосіб ,хакер, захист, ключ.
Keywords: White Box, Black Box, Gray Box, method, hacker, protection, key

Науковий керівник: *к.т.н., доц. Задерейко О.*

Тітієвський Віталій

Національний університет «Одеська юридична академія»

студент 3 курсу факультету кібербезпеки та інформаційних технологій

СПЕЦИФІКА РОЗРОБКИ БЕКЕНДУ ЗАСОБАМИ EXPRESS.JS

Розробка бекенду (від англ. back-end) вебсайту або вебзастосунка відповідає за серверну сторону багатокомпонентної вебсистеми, на якій відбуваються основні процеси, приховані від користувача [1]. Головна мета бекенду полягає у тому, щоб швидко відібрати та подати користувачеві дані за його запитом у зручному для нього вигляді. Також бекенд відповідає за організацію і зберігання даних у відповідних базах даних, за швидкий доступ і релевантний відбір даних за запитами.

Існують різні серверні мови програмування та фреймворки для розробки бекенду. Нині поширено для цього використовують Node.js, React.js, PHP, Ruby, Python, C# тощо.

Зручність використання Node.js, як платформи мови програмування JavaScript, полягає у можливості використовувати одну мову і для бекенду, і для фронтенду, тобто надає повний стек розробки. Використання однієї мови програмування означає гарну сумісність, високу продуктивність і можливість обміну та повторного використання компонентами, що покращує і прискорює розробку. Node.js надає можливість виконувати скрипти на сервері, що пришвидшує завантаження сторінки та знижує навантаження на процесор користувача. Крім того, Node.js простий для вивчення і має гарні можливості масштабованості.

Для спрощення розробки можна скористатись численними фреймворками Node.js. Серед найбільш популярних: Express.js, Hapi.js, Koa.js, LoopBack, Sails.js, Kraken.js, Meteor та інші.

Засоби Express.js дозволяють створювати вебсайти різного розміру та складності і при цьому забезпечити високу продуктивність та критичну масштабованість. Швидкий і легкий Express.js має потужний функціонал і підтримує функції проміжного оброблення запитів (middleware) [2].

Загалом Express.js може використовувати різні типи проміжних обробників на рівні програми, на рівні маршрутизатора, сторонніх постачальників програмного забезпечення, а також для обробки помилок та вбудовані проміжні обробники. Задачами функцій проміжного оброблення запитів є почергового виконання коду функцій запитів і внесення змін в об'єкти запитів та відповідей. Ці функції виконуються по колу (циклічно, конвеєром) і кожна з них передає керування до наступної функції. Наприклад, функція з аналізу вхідних json-запитів, яка помістить дані в req.body, звідки їх буде легко брати:

```
app.use(express.json())
```

При виконанні запитів можуть виникати різноманітні проблеми як на боці користувача, так і на боці сервера. Якщо користувач не зрозуміє в чому проблема,

він просто вийде із вебзастосунку. Для розв'язання цієї проблеми існує функція res.status(xxx).json(), де xxx – це код стану HTTP:

```
return res.status(404).json({message: 'Користувач не знайдений'})
```

Крім написання обробників для запитів різними HTTP-методами в різних URL-адресах (маршрутах), Express.js має механізми налаштування загальних параметрів вебзастосунків (наприклад, порт для підключення, розташування шаблонів, які використовуються для відображення відповіді), інтеграції: з механізмами рендерингу «view», для генерації відповідей, вставляючи дані в шаблони. Загалом Express.js має сумісні пакети проміжного програмного забезпечення для вирішення практично будь-яких проблем з веброзробкою. Є бібліотеки для роботи з cookie-файлами, сеансами, входами користувачів, параметрами URL, POST, заголовками безпеки тощо.

При цьому Express.js є мінімалістичним фреймворком і дозволяє розробникам вибирати будь-які інструменти та технології. Він не має конкретного механізму шаблонізації, парсера тіла запиту, обробника помилок тощо. З одного боку, це привертає розробників-початківців, які можуть ставити і використовувати свої улюблені інструменти. Проте, з іншого, на проєктах із мікросервісною архітектурою це спричиняє серйозні проблеми неузгодженості та відсутність чіткої структури коду.

Врешті решт не існує ідеальних засобів розробки бекенду, кожен з них має свою специфіку, свої сильні та слабкі сторони, які варто враховувати при виборі стеку програми.

Список використаних джерел:

1. The top 10 backend web development frameworks for 2022. URL: https://www.westagilelabs.com/blog/top-backend-web-development-frameworks/

2. Comparing the most popular Node.js frameworks in 2021. URL: https://scoutapm.com/blog/nodejs-frameworks

Ключові слова: Express.js, Node.js, бекенд, вебсайт, вебзастосунок.