

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)
на тему:

«Веб орієнтований інформаційний додаток »

Виконав студент групи ПЗ-19-2
ТРОЦЬКИЙ Владислав Валерійович

Керівник роботи:
ЛОБАНЧИКОВА Надія Миколаївна

Рецензент:
МОРОЗОВ Андрій Васильович

Житомир – 2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

В.о. зав. кафедри інженерії
програмного забезпечення
Андрій МОРОЗОВ
«14» лютого 2023 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **Троцький Владислав Валерійович.**

Керівник роботи: **Лобанчикова Надія Миколаївна.**

Тема роботи: **«Веб орієнтований інформаційний додаток»**,
затверджена Наказом закладу вищої освіти від **«14» лютого 2023 р., №70с.**

Вихідні дані для роботи: Об'єктом дослідження є процеси та інформаційні технології побудови веборієнтованих інформаційних додатків. Предметом дослідження є моделі, методи, алгоритми та засоби побудови веборієнтованого інформаційного додатку з фокусом на актуальний інформаційний контент і зручний інтерфейс користувача.

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Лобанчикова Н.М.	14.02.2023р.	14.02.2023р.
2	Лобанчикова Н.М.	17.04.2023р.	18.04.2023р.
3	Лобанчикова Н.М.	01.05.2023р.	01.05.2023р.

Календарний план

№ з/П	Назва етапів випускної кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Постановка задачі	3 березня 2023 – 15 березня 2023	Виконано
2	Пошук, огляд та аналіз аналогічних розробок	15 березня 2023 – 19 березня 2023	Виконано
3	Формулювання технічного завдання та опрацювання літературних джерел	19 березня 2023 – 29 березня 2023	Виконано
4	Проектування структури системи	29 березня 2023 – 6 квітня 2023	Виконано
5	Написання програмного коду	6 квітня 2023 – 25 травня 2023	Виконано
6	Тестування роботи додатку	25 травня 2023 – 30 травня 2023	Виконано
7	Попередній захист	5 травня 2023	Виконано

РЕФЕРАТ

Бакалаврська робота складається з веборієнтованого інформаційного додатку та пояснювальної записки. Пояснювальна записка до випускної роботи містить 76 сторінок, 27 ілюстрацій та 4 таблиці.

Метою роботи є розробка сайту веборієнтованого додатку з використанням HTML, TAILWIND CSS, EXPRESS.JS, а також розробка системи адміністрування з використанням EXPRESS.JS та використанням хмарної бази даних MongoDB Atlas.

В роботі були встановлені вимоги користувача і обґрунтовано створення концептуальної моделі. Було проведено аналіз аналогів програмних продуктів і вибрано засоби для реалізації. Розроблена структура бази даних, обрана архітектура та розроблена структура системи. Була описана програмна реалізація вебсайту і наведені результати тестування системи.

КЛЮЧОВІ СЛОВА: EXPRESS.JS, АДМІНІСТРУВАННЯ, БАЗА ДАНИХ, ВЕБСАЙТ, ІНФОРМАЦІЙНИЙ ДОДАТОК, ПРОДУКТИ.

					ІПЗ.КР.Б – 121 – 23 – ПЗ			
Зм.	Арк.	№ докум.	Підпис	Дата				
Розроб.		В.В. Троцький			Веб орієнтовний інформаційний додаток. Пояснювальна записка	Літ.	Арк.	Аркушів
Керівник		Н.М. Лобанчикова					4	60
						Житомирська політехніка, група ІПЗ-19-2		
Рецензент		А.В. Морозов						
Зав. каф.		А.В. Морозов						

ABSTRACT

The bachelor's thesis consists of a web-oriented information application and an explanatory report. The explanatory report for the thesis consists of 76 pages, 27 illustrations, and 4 tables.

The aim of the thesis is to develop a web-oriented application website using HTML, TAILWIND, EXPRESS.JS, and to develop an administration system using EXPRESS.JS with the cloud database MongoDB Atlas.

The thesis establishes user requirements and justifies the creation of a conceptual model. An analysis of software product analogues was conducted, and tools for implementation were selected. The database structure was developed, and the system architecture and structure were chosen. The implementation of the website was described, and the results of the system testing were provided.

KEYWORDS: ADMINISTRATION, DATABASE, EXPRESS.JS, INFORMATION APPLICATION, PRODUCTS, WEBSITE.

					ИПЗ.КР.Б – 121 – 23 – ИПЗ	
						5

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ.....	7
ВСТУП	8
РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ ІНФОРМАЦІЙНОГО ДОДАТКА	11
1.1. Постановка задачі	11
1.2. Аналіз аналогів програмного продукту	13
1.3. Вибір архітектури інформаційного додатка.....	18
1.4. Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення	20
Висновки до розділу 1	21
РОЗДІЛ 2. ПРОЕКТУВАННЯ СИСТЕМИ ІНФОРМАЦІЙНОГО ДОДАТКА ..	23
2.1. Визначення варіантів використання та об'єктно-орієнтованої структури системи	23
2.2. Розробка бази даних системи.....	29
2.3. Проектування та реалізація алгоритмів роботи системи.....	33
2.4. Реалізація функціоналу інформаційного додатка.....	34
Висновки до розділу 2	46
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З СИСТЕМОЮ ДОДАТКА	47
3.1 Порядок встановлення та налаштування системи.....	47
3.2 Структура інтерфейсу. Інтерфейс та порядок роботи з веборієнтованим інформаційним додатком	49
3.3 Тестування роботи програмного комплексу	57
Висновки до розділу 3	59
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	62
ДОДАТКИ.....	63

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

API – application programming interface.

CRUD – create, read, update, delete (з англ. створити, прочитати, оновити, видалити).

JS – JavaScript.

MDB – MongoDB.

БД – База даних.

ПЗ – Програмне забезпечення.

СКБД – Система керування базами даних.

ВСТУП

Актуальність теми. Тематика щодо розробки веборієнтованих інформаційних додатків є актуальною з кількох причин.

По-перше, вебдодатки стали необхідними інструментами для доступу до інформації, здійснення операцій та взаємодії з різними сервісами і продуктами в Інтернеті. Вони широко застосовуються в різних галузях, включаючи освіту, бізнес, електронну комерцію, соціальні мережі, медіа і багато інших.

По-друге, особливо, під час пандемії та війни, коли існує багато джерел ненадійної інформації або мають обмежені інформаційні ресурси. Багато людей віддають перевагу інформаційним додаткам з перевіреною і актуальною інформацією.

Також важливо зазначити, що застосування веборієнтованих інформаційних додатків широке і різноманітне. Вони можуть використовуватись у сферах освіти, бізнесу, комунікацій, електронної комерції, медицини та багатьох інших галузях. Важливо мати зручну, ефективну та надійну систему для обробки та зберігання інформації, а також забезпечити можливість адміністрування та управління додатком.

Взагалі, веборієнтований інформаційний додаток може мати потенціал для інновацій та вдосконалення. Із зростанням вимог користувачів, нових технологій та тенденцій, вебдодаток може розвиватися та адаптуватися до змін, що дозволяє підтримувати його актуальність протягом тривалого часу.

Метою випускної роботи є розробка веборієнтованого інформаційного додатку з фокусом на актуальний інформаційний контент і зручний інтерфейс користувача. Такий додаток може бути використаний для представлення новин, статей, блогів, ресурсів або будь-якої іншої інформації, що має цінність для цільової аудиторії.

Основні завдання роботи включають в себе:

1. Вибір технологій та інструментів: оцінка та вибір найбільш підходящих технологій та інструментів для реалізації веборієнтованого інформаційного додатку.

2. Аналіз потреб користувачів: визначення основних потреб та очікувань цільової аудиторії додатку, щоб забезпечити високу користувальницьку задоволеність.

3. Розробка концептуальної моделі: визначення загальної структури, функціональності та інтерфейсу додатку на основі вимог користувачів.

4. Дизайн інтерфейсу користувача: розробка привабливого та зручного дизайну, включаючи графічні елементи, кольорову схему та організацію контенту.

5. Реалізація функціональності: розробка основних функцій додатку, які відповідають вимогам користувачів, включаючи обробку даних, роботу з базою даних та інтеграцію з іншими сервісами або API.

6. Безпека та захист даних: забезпечення високого рівня безпеки даних користувачів та застосування відповідних заходів для запобігання несанкціонованому доступу до інформації.

7. Тестування та відлагодження: проведення різних тестів, виявлення помилок та усунення їх для забезпечення стабільної та надійної роботи додатку.

8. Документування: підготовка документації, включаючи пояснювальну записку, технічну документацію та інструкції з використання додатку.

9. Оцінка ефективності: аналіз та оцінка ефективності додатку, включаючи його швидкодію, доступність та задоволення потреб користувачів.

Об'єктом дослідження є процеси та інформаційні технології побудови веборієнтованих інформаційних додатків.

Предметом дослідження є моделі, методи, алгоритми та засоби побудови веборієнтованого інформаційного додатку з фокусом на актуальний інформаційний контент і зручний інтерфейс користувача.

У цій роботі використовувалися методи вебпрограмування для розробки функціональності та структури інформаційного додатку. Були також

використані методи дизайну користувацьких інтерфейсів, щоб забезпечити зручний та привабливий вигляд додатку.

Розроблена система інформаційного додатку може бути використана як практичний вебсайт, який може бути легко адаптований для будь-якого існуючого інформаційного додатку. Більше того, є можливість розширення функціональності системи шляхом додавання унікальних функцій та удосконалення властивостей.

РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ ІНФОРМАЦІЙНОГО ДОДАТКА

1.1. Постановка задачі

Постановка задачі випускної роботи полягає в розробці веборієнтованого інформаційного додатку з фокусом на актуальний інформаційний контент і зручний інтерфейс користувача. Основні задачі, поставлені в рамках цієї роботи, включають:

1. Проектування системи:

- Визначення потреб і функціональних вимог до інформаційного доповнення.
- Встановлення структури системи, включаючи сторінки постів, категорії, облікові записи користувачів та інші компоненти.
- Розробка логіки та алгоритмів роботи системи.

2. Розробка інтерфейсу:

- Створення привабливого дизайну інтерфейсу, враховуючи ергономіку та користувацький досвід.
- Забезпечення зручності навігації та інтуїтивно зрозумілих елементів керування.
- Розробка адаптивного дизайну, що забезпечує коректне відображення інтерфейсу на різних пристроях і розмірах екранів.

3. Розробка бази даних:

- Розробка механізмів для збору та аналізу даних про пости, поведінку користувачів та коментарів.
- Реалізація механізмів доступу до даних, включаючи операції додавання, редагування та видалення записів.

4. Тестування та вдосконалення:

- Ретельне тестування різних функціональних аспектів системи, включаючи функції пошуку, додавання та поширення коментарів тощо.
- Виявлення та виправлення помилок, а також забезпечення відповідності роботи системи заданим вимогам та очікуванням користувачів.
- Оптимізація продуктивності системи для швидкого перегляду сторінок та ефективного використання ресурсів сервера.

5. Аналітика та звітність:

- Розробка механізмів для збору та аналізу даних про пости, поведінку користувачів та коментарів.

Планується розробка інформаційного додатку, який матиме зручний та інтуїтивно зрозумілий інтерфейс, дозволяючи користувачам легко розміщувати потрібні публікації. У нашому додатку буде наявна система сортування постів за категоріями та зручний пошуковий механізм. Безпека та конфіденційність персональних даних користувачів є пріоритетним, тому планується використання надійних методів шифрування та захисту інформації для гарантування конфіденційності та захисту даних від несанкціонованого доступу.

Після розробки додатку планується проведення ретельного тестування, щоб переконатися в правильному функціонуванні всіх компонентів системи. Виявлені помилки будуть виправлені для забезпечення вимог коректної роботи системи, яка відповідає очікуванням користувачів. Метою є створення інформаційного додатку, який забезпечуватиме комфортний і безпечний перегляд публікацій для наших користувачів з високою якістю обслуговування та задоволення їх потреб.

На наступному етапі розробки проведено аналіз програмних продуктів даного призначення з метою виявлення переваг та недоліків кожного з них для подальшого проектування системи.

1.2. Аналіз аналогів програмного продукту

Аналіз аналогічних програмних продуктів перед розробкою веборієнтованого інформаційного додатку є важливою складовою процесу. Він дозволяє отримати корисні вказівки щодо функціональності, взаємодії з користувачем та ефективності роботи додатку.

Ось кілька кроків, які можна виконати для аналізу аналогів:

1. Проведемо огляд наявних веборієнтованих інформаційних додатків. Здійснимо пошук та вивчимо популярні існуючі додатки, щоб отримати уявлення про їх функціональність, дизайн та особливості. Проаналізуємо можливості цих додатків.

2. Визначимо основні функції, які повинен виконувати наш веборієнтований інформаційний додаток, і складемо список. Потім розглянемо, як ці функції реалізовано в аналогічних програмних продуктах.

3. Проведемо оцінку інтерфейсу користувача, проаналізуємо дизайн інтерфейсу в аналогічних програмних продуктах і розглянемо, як користувачі взаємодіють з додатком. Оцінімо зручність, інтуїтивність та зрозумілість інтерфейса.

4. Після аналізу аналогів зробимо висновки і складемо план розробки, враховуючи отримані дані. Визначимо, які елементи будуть запозичені від аналогічних програмних продуктів, а також визначимо унікальні функції, які плануємо розробити самостійно.

На основі цього складемо план розробки, враховуючи переваги та недоліки, виявлені під час аналізу аналогічних програмних продуктів.

В Україні існує декілька популярних веборієнтованих інформаційних додатків, які можуть послужити нам прикладом для аналізу ринку в контексті програмної розробки [1-14]. Ось декілька з них, разом із їхніми перевагами та недоліками.

"Фокус" – відомий український журнал "Фокус" має власний веборієнтований інформаційний додаток, (див. рис.1.1), де публікуються статті,

новини та репортажі на різноманітні теми: політика, економіка, культура, наука тощо [14].

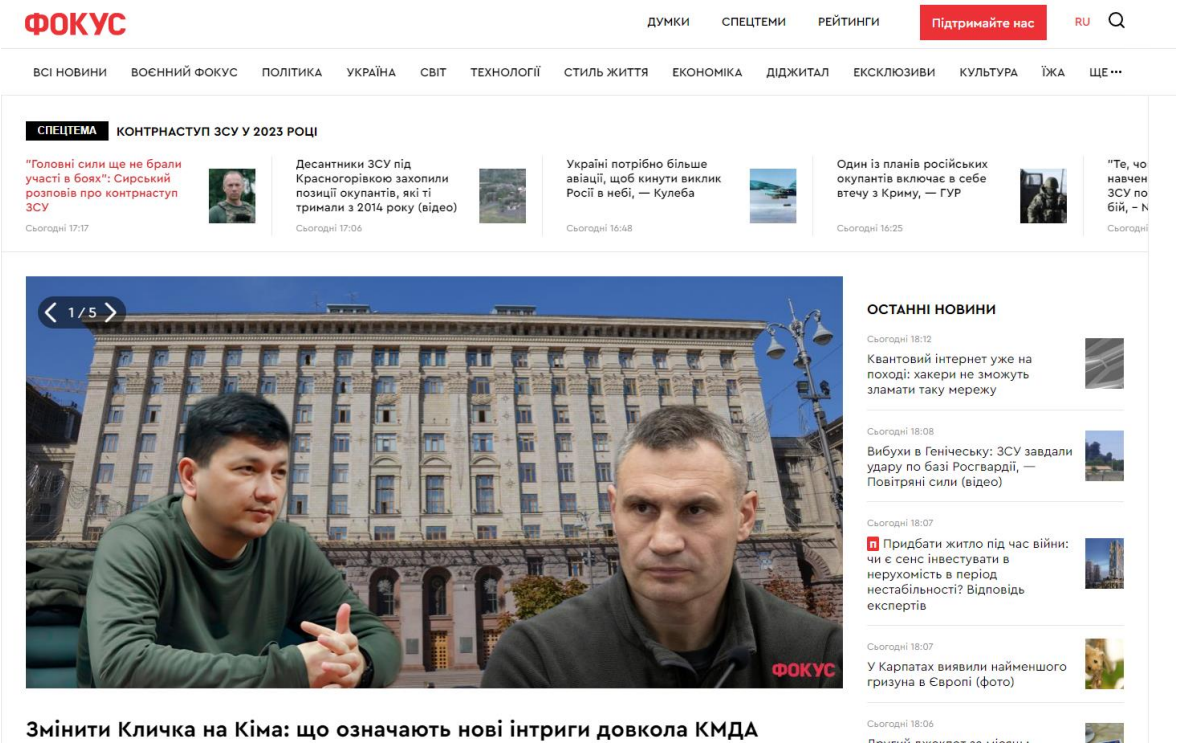


Рисунок 1.1 – Головна сторінка "Фокус"

Основними перевагами "Фокус" є:

1. Різнноманітний контент.
2. Різнноманітний контент.
3. Зручний інтерфейс: додаток має зручний та інтуїтивно зрозумілий інтерфейс.

Недоліками є:

1. Реклама: у додатку можуть бути розміщені рекламні матеріали, які можуть вплинути на користувачів інформаційного додатку.
2. Оптимізація: у деяких випадках додаток може потребувати подальшої оптимізації для підвищення продуктивності та швидкості завантаження контенту.
3. Орієнтація на внутрішні ресурси: додаток може бути більш спрямованим на внутрішні ресурси журналу, що обмежує доступ до зовнішнього контенту та різноманітних джерел інформації.

"DOU" – це український блог-агрегатор, який збирає та публікує статті та дописи від українських IT-спеціалістів, програмістів та фахівців у галузі розробки програмного забезпечення, (див. рис.1.2). Окрім того наявна агрегація статей, коментарів та дописів [5].

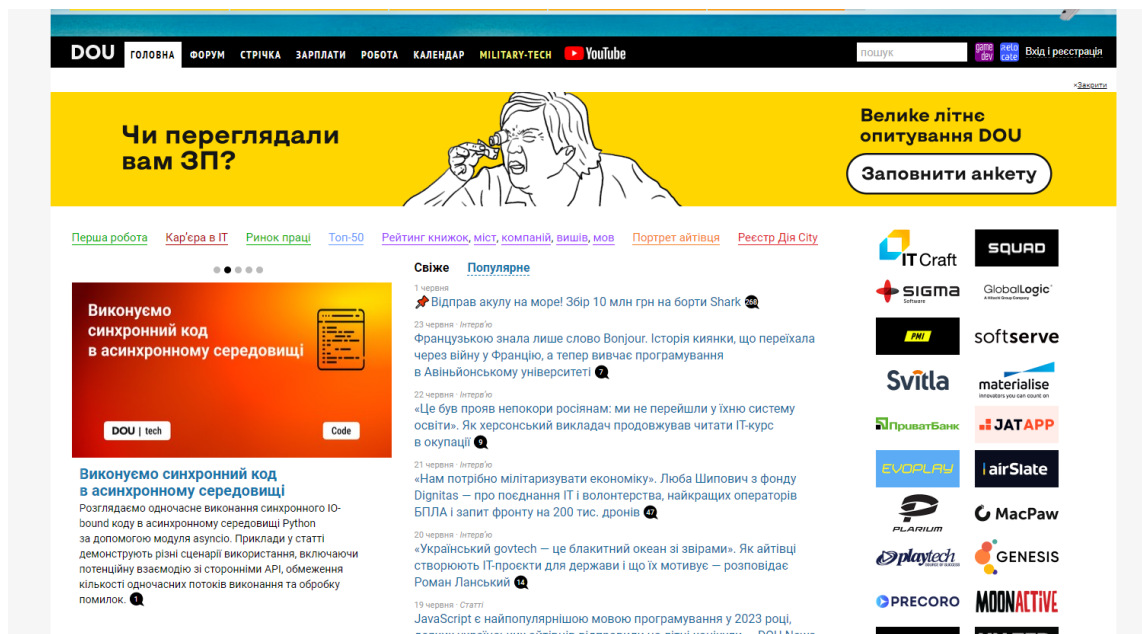


Рисунок 1.2 – Головна сторінка "DOU".

Основними перевагами DOU є:

1. Зручний та простий інтерфейс користувача. Дизайн не викликає труднощів.
2. Персоналізація. Користувачі можуть налаштувати свій профіль, вибрати підписки та налаштування спільності відповідно до своїх інтересів та потреб.
3. Зручність читання. Додаток пропонує приємний дизайн для комфортного читання статей та дописів. Текст відображатися чітко, зручний розмір шрифту та наявність відступів для зручності читання на різних пристроях.

Недоліками є:

1. Обмежений контент. Хоча DOU спеціалізується на IT-інформації, деякі користувачі можуть відчувати обмеженість тематичного покриття.
2. Відсутність мобільних додатків. DOU не має окремого мобільного додатка для iOS або Android, який може бути недоліком для користувачів, які

віддають перевагу зручній мобільній навігації та сповіщенню.

3. Іноді можуть виникати помилки або технічні проблеми, які впливають на роботу сайту та користувачів.

"GOIT" – є платформою, де IT-фахівці та експерти діляться своїми знаннями, досвідом та новинами з галузі IT, (див.рис.1.3). Тут публікуються статті про програмування, веброзробку, аналітику та інші теми [10].

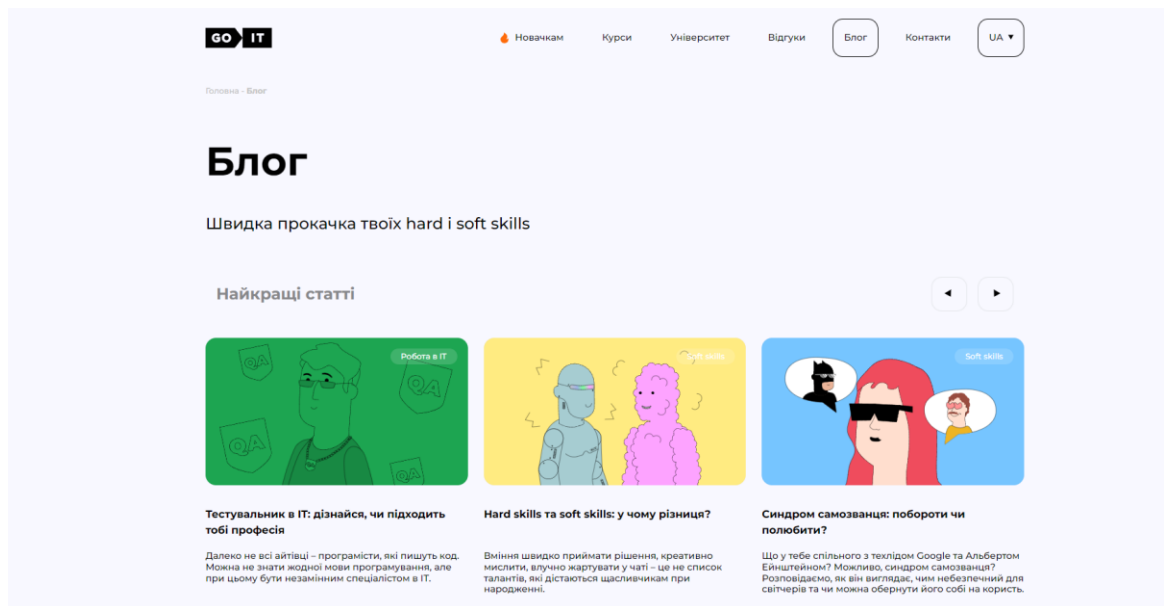


Рисунок 1.3 – Головна сторінка "GOIT"

Основними перевагами GOIT є:

1. Зручна навігація та структурованість вебсайту для легкого пошуку та доступу до різних функцій.
2. Чистий та сучасний дизайн, що забезпечує приємний візуальний досвід.
3. Інтуїтивно зрозумілі елементи керування та кнопки для простоти використання.
4. Чуйний дизайн, що адаптується до різних пристроїв, таких як комп'ютери, планшети та смартфони.
5. Чуйний дизайн, що адаптується до різних пристроїв, таких як комп'ютери, планшети та смартфони.

Недоліками є:

1. Технічні проблеми. Проблеми зі стабільністю сайту, неполадки з

доступом до матеріалів або завдань, помилки в роботі платформи.

2. Обмежені можливості. Деякі користувачі вважають, що функціональна платформа недостатньо розширена або, що вона не дає достатньо глибокого ознайомлення з певними темами.

3. Відсутність персоналізації. Якщо платформа не забезпечує відповідності можливостей налаштування навчального досвіду до індивідуальних потреб та рівні користувача, це може бути недоліком для деяких користувачів.

Висновок: після аналізу переваг та недоліків веборієнтованих інформаційних додатків можна зробити наступні висновки для розробки сайту:

1. Функціональність:

- Забезпечити широкий вибір постів і різноманітні категорії для задоволення потреб користувачів.
- Реалізувати зручний пошук та категоризацію постів для полегшення навігації.

2. Інтерфейс користувача:

- Створити простий та зручний інтерфейс із легкою навігацією.
- Розробити привабливий дизайн, який спонукатиме користувачів до взаємодії з сайтом.

3. Користувацький досвід:

- Забезпечити користувачам можливість залишати відгуки та оцінки постів, які допоможуть іншим користувачам при виборі матеріалів.

4. Швидкість та надійність:

- Оптимізувати швидкість роботи сайту та забезпечити швидке завантаження сторінок.
- Забезпечити надійність сайту, щоб користувачі мали стабільний доступ до інформації.

На основі аналізу переваг і недоліків веборієнтованих інформаційних додатків переходимо до розробки програмного продукту, враховуючи недоліки

конкурентів і підвищуючи їх переваги. Планується створення індивідуального, функціонального та ефективного додатку.

1.3. Вибір архітектури інформаційного додатка

Для розробки веборієнтованої системи використовуємо фреймворк EXPRESS.JS з API Router. Порівнявши різні підходи, визначено, що EXPRESS.JS API Router надає найкращу гнучкість для відображення даних.

EXPRESS.JS –це фреймворк розробки вебдодатків, побудований на базі React.js. Він дозволяє створювати як статичні, так і динамічні вебсайти з використанням серверного та клієнтського рендерингу. Зокрема, API Router у EXPRESS.JS дозволяє нам створювати серверні API-шляхи, які можна використовувати для отримання та збереження даних.

Вибір EXPRESS.JS API Router зумовлений його здатністю до простої та зручної реалізації роботи з даними в контексті веборієнтованого інформаційного додатку. Основні компоненти архітектури системи EXPRESS.JS API Router включають наступне:

1. Фронтенд. Для розробки фронтенду, тобто клієнтської частини, ми використовуємо React.js, який дозволяє нам створювати компоненти для відображення користувацького інтерфейсу. Використання EXPRESS.JS допомагає нам здійснювати рендеринг як на стороні сервера, так і на стороні клієнта, що дає нам можливість використовувати різні підходи до відображення сторінок. [9].

2. API шляхи. EXPRESS.JS надає можливість визначати API шляхи за допомогою вбудованого API Router. Ці шляхи визначають, які дані можна отримати або змінити на сервері.

Розробники можуть створювати API шляхи, використовуючи декларативний підхід, де вони вказують шаблони URL та функції-обробники для кожного шляху [3].

3. Для кожного API шляху в EXPRESS.JS пов'язана функція-обробник, яка виконується при отриманні запиту на цей шлях. Ця функція відповідає за обробку запитів, виконання бізнес-логіки, взаємодію з базою даних або зовнішніми службами та повернення результату у вигляді відповіді [3].

4. Маршрутизація. API Router в EXPRESS.JS дозволяє маршрутизувати запити до різних шляхів в залежності від URL-шаблонів.

Використовуємо параметри шляху для передачі додаткових даних до функцій-обробників та динамічного створення API шляхів. Це забезпечує гнучкість та можливість налаштування маршрутизації вебдодатка [3].

5. Валідація та автентифікація. EXPRESS.JS API Router надає вбудовану підтримку валідації та автентифікації запитів.

Розробники можуть використовувати різні пакети або власні методи для перевірки достовірності користувачів, перевірки вхідних даних та забезпечення безпеки API. Це дозволяє забезпечити безпеку та надійність вебдодатка шляхом контролю доступу до ресурсів та перевірки правильності переданих даних [3].

6. Додаткові функціональні можливості: EXPRESS.JS API Router дозволяє використовувати middleware для обробки запитів перед передачею їх до функцій-обробників.

Це дозволяє вставляти додаткову логіку або виконувати певні дії, наприклад, перевірку автентифікації, логування або перехоплення помилок, перед тим, як запит досягне функції-обробника.

Крім цього, EXPRESS.JS API Router підтримує WebSocket-з'єднання, що дозволяє забезпечити багаторазову взаємодію між клієнтом і сервером без необхідності постійного відправлення запитів.

Архітектура системи EXPRESS.JS API Router ґрунтується на концепції розділення клієнтської та серверної частин. Вона використовує API шляхи для взаємодії з даними, функції-обробники для обробки запитів та маршрутизації, а також надає додаткові можливості для забезпечення безпеки та розширення функціональності.

1.4. Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення

Для розробки інформаційного додатку використовуються наступні інструменти та фреймворки. React.js використовується для створення користувацького інтерфейсу, Tailwind CSS – для стилізації компонентів, Express.js – для обробки логіки на серверній стороні. Для зберігання даних використовується хмарна база даних MongoDB Atlas. Для дизайну та прототипування використовується графічний редактор Figma.

При виборі інструментальних засобів для розробки інформаційного додатку були враховані різні важливі фактори, такі як швидкодія, ефективність розробки, зручність стилізації компонентів, можливість реалізації серверної логіки, надійність зберігання даних та можливості для дизайну і прототипування. Використання зазначених інструментів дозволяє створити функціональний і привабливий інформаційний додаток зі зручним інтерфейсом і оптимальною продуктивністю.

1. React.js є фреймворком широкого використання для розробки вебінтерфейсів, який ґрунтується на JavaScript. Він дозволяє створювати ефективні вебінтерфейси з високою продуктивністю. React.js заснований на компонентній структурі, де користувацький інтерфейс розбивається на незалежні, невеликі компоненти. Цей підхід спрощує процес розробки, тестування та підтримки додатків [9].

2. Express.js є популярним фреймворком для розробки вебдодатків, який забезпечує високу продуктивність та ефективність. Він підтримує різні методи рендерингу, такі як серверний та статичний клієнтський, що сприяє швидкому завантаженню сторінок. Express.js також пропонує зручну систему маршрутизації, яка спрощує процес розробки та підтримки проектів [3].

3. Tailwind CSS – це методологія стилізації компонентів, яка використовує модульний підхід до CSS. Застосовуючи Tailwind CSS, можна створювати локальні області стилів для кожного компонента окремо, уникати конфліктів між

стилями. Основна перевага полягає в простоті перенесення та повторного використання стилів, що полегшує розробку та підтримку компонентів [8].

4. MongoDB Atlas – це керований хмарний сервіс баз даних MongoDB, який надає надійний та масштабований спосіб для зберігання та керування даними у форматі документів. MongoDB, власне, є нереляційною базою даних, що дозволяє гнучко змінювати структуру даних і додавати нові поля без необхідності в міграції баз даних. За допомогою MongoDB Atlas легко розширювати кластери та ефективно обробляти великі обсяги даних, забезпечуючи зручне та надійне збереження даних для інформаційного додатка [11].

5. Visual Studio Code (VS Code) – це інтегроване середовище розробки (IDE), розроблене компанією Microsoft. Воно спеціалізується на веброботці і надає потужні інструменти для роботи з HTML, CSS та JavaScript. VS Code містить ряд корисних функцій, таких як підсвічування синтаксису, автодоповнення коду, рефакторинг, налагодження та автоматична перевірка синтаксису.

Висновки до розділу 1

Проведений аналіз у сфері веборієнтованих інформаційних додатків показав, що важливо розробити функціонал для збереження даних про публікації, їх характеристики, а також про користувачів та коментарі. Крім того, потрібно розробити механізми для швидкого пошуку та підбору публікацій за параметрами, а також для ефективного управління публікаціями та інформації про користувача.

Огляд існуючих аналогів показав, що основними функціями системи для веборієнтованих інформаційних додатків є розміщення публікацій, пошук публікацій в Інтернеті, наявність мережевої версії та вебінтерфейсу. Важливо також відстежувати коментарі та імпортувати відредаговані дані про

користувача для зручного використання програми. Було б доцільно реалізувати додаткові функції, такі як пошук постів та фільтрація за певними параметрами.

В роботі обґрунтовано вибір архітектури для реалізації системи:

- REACT.JS як front-end;
- TAILWIND CSS для стилізації компонентів;
- EXPRESS.JS як back-end;
- хмарне середовище керування базами даних MongoDB Atlas;
- комп'ютер користувача виконує роль сервера.

Буде використано такі програмні продукти та інтегровані середовища розробки як: графічний редактор Figma, среда розробки VS Code.

РОЗДІЛ 2. ПРОЕКТУВАННЯ СИСТЕМИ ІНФОРМАЦІЙНОГО ДОДАТКА

2.1. Визначення варіантів використання та об'єктно-орієнтованої структури системи

Інформаційний додаток має за мету поширення актуального, інформативного та перевіреного контенту та розповсюдження інформація.

Вимоги користувачів. Гість має доступ до наступних функцій (рис.2.1):

1. до інформації з головної сторінки (постів, інформацію про організацію, перегляд коментарів);
2. до перегляду окремого поста та інформації про неї (дату створення, ким створена);
3. до коментарів під окремого коментаря;
4. до перегляду всіх постів та їх детальну інформацію;
5. до реєстрації або авторизації на сайті.

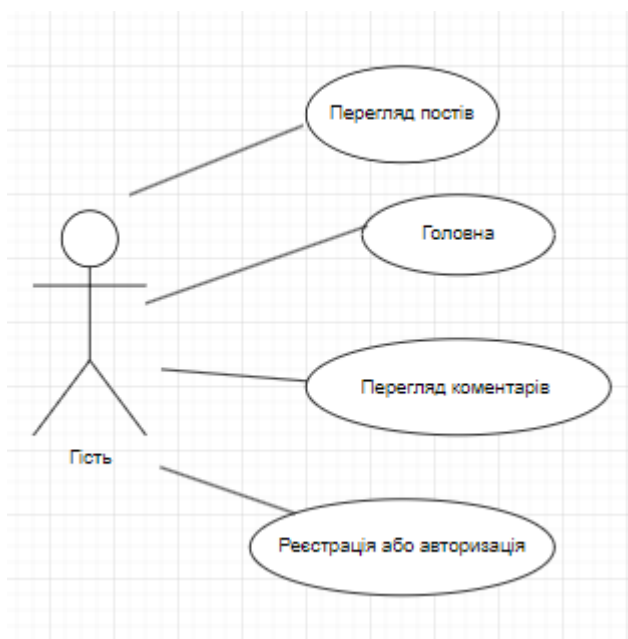


Рисунок 2.1 – Діаграма використання гостя

Авторизований користувач має доступ до наступних функцій (рис.2.2):

1. До інформації з головної сторінки (постів, інформацію про організацію, перегляд коментарів);

2. до перегляду окремого поста та інформації про неї (дату створення, ким створена);
3. до коментарів окремого поста та їх створенню;
4. до особистого кабінету;
5. до CRUD по постам;
6. до CRUD з коментарями;
7. до виходу с облікового запису;
8. до оновлення облікових даних в профілю користувача.



Рисунок 2.2 – Діаграма використання авторизованого користувача

Адміністратор має доступ до наступних функцій (рис.2.3):

1. до адмінкабінету;
2. до CRUD з постів;
3. до CRUD з користувачів;
4. до інформації з головної сторінки (пости, інформацію про організацію);
5. до перегляду окремого поста та інформації про неї (дату створення, ким створений);

6. до коментарів окремої публікації та їх створенню.



Рисунок 2.3 – Діаграма використання адміністратора

Функціональні вимоги:

1. Авторизація та реєстрація. Для користувачів необхідна можливість реєстрації. Необхідна реалізація можливості додавання користувачів з адміністративного кабінету.
2. Дозволити користувачам створювати облікові записи та виконувати вхід до системи.
3. Створення, редагування, перегляд та видалення публікацій, коментарів та користувачів з адмінкабінету.
4. Забезпечити зберігання конфіденційності користувачів.
5. Система повинна коректно та автоматично відображати необхідну інформацію.
6. Дозволити користувачам переглядати та редагувати свій профіль.

Нефункціональні вимоги:

1. Сприйняття:

– Час, потрібний для ознайомлення з інструментами інформаційного

додатку - 6 хв.

- Час відповіді системи для звичайних запитів не повинен перевищувати 0.490 мс, а для складних запитів - 0.960 мс.
- Інтерфейс та адаптивна його версія повинен бути зрозуміла для користувачів.

2. Надійність:

- Середній час безперервної роботи - 48 год.
- Максимальна норма помилок та дефектів в роботі програми - 1 помилка на 500 запитів користувача.
- Система повинна мати можливість підтримувати 100 користувачів одночасно.

3. Швидкість. Система повинна швидко виконувати запити користувачів.

4. Можливості використання:

- Масштабування – система повинна мати можливість збільшувати свою потужність та продуктивність.
- Оновлення версій – здійснювати відповідно до вподобань користувачів та розширення переліку надання послуг.

Після аналізу функціональних вимог, встановлено наявність ключових елементів, які забезпечать створення програмного комплексу для даної системи. На діаграмі класів, яка представлена на рис. 2.4, показані контролери системи. Детальна структура представлень буде розглянута в третьому розділі.

На даному рисунку можна виділити наступні класи:

Admin – клас, який містить методи для роботи зі всіма сутностями додатка.

Поля:

- avatar (тип bigint) – унікальний ідентифікатор;
- name (тип string) – ім'я користувача;
- email (тип string) – е-мейл користувача;
- password (тип string) – пароль користувача;
- verified (тип boolean) – перевірка користувача;

- verificationCode (тип string) – код перевірки користувача;
- admin (тип boolean) – роль користувача.

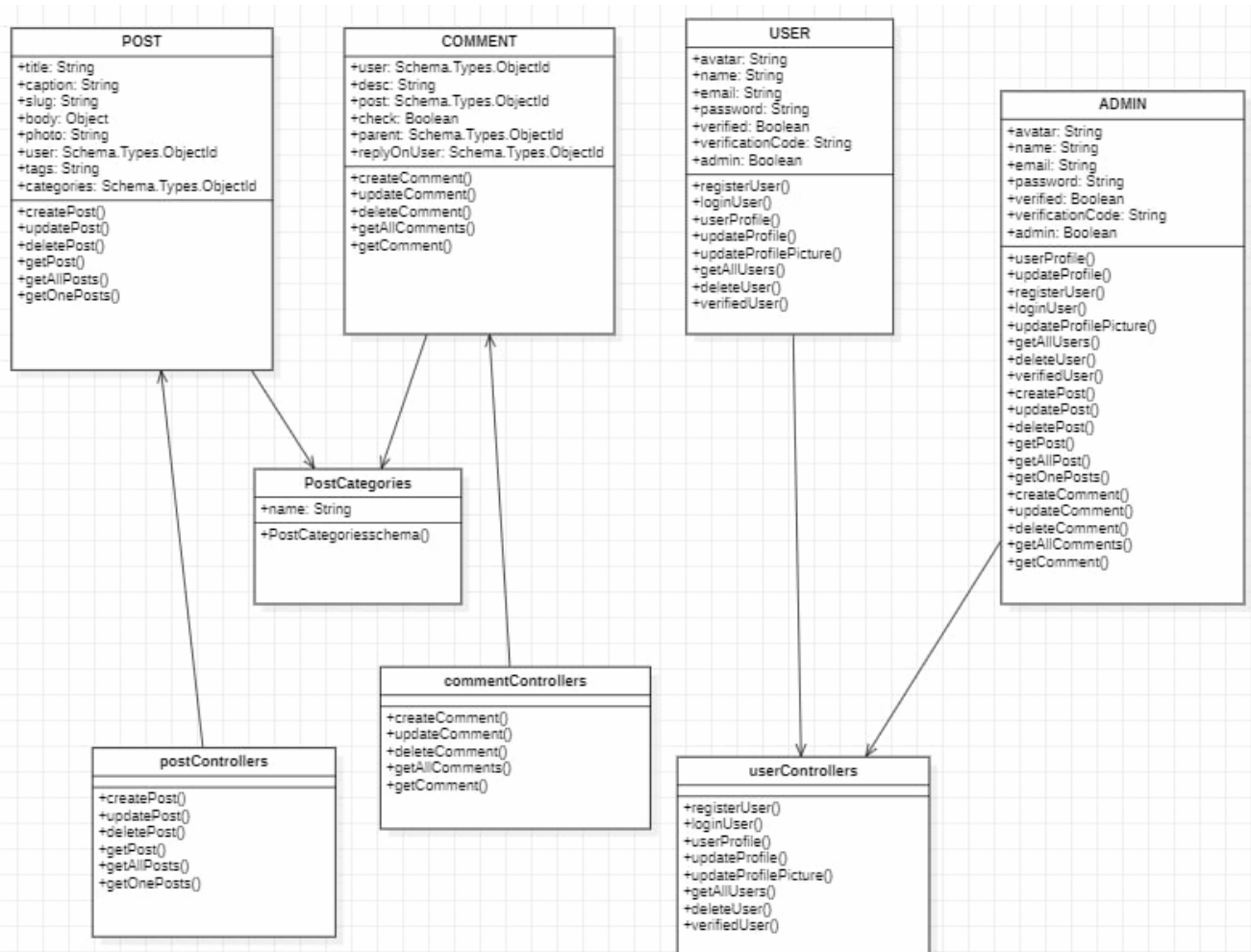


Рисунок 2.4 – Діаграма класів

Методи:

- userProfile() – профіль користувача;
- updateProfile() – оновлення профілю користувача;
- registerUser() – реєстрація користувача;
- loginUser() – авторизація користувача;
- updateProfilePicture() – оновлення фото профілю користувача;
- getAllUsers() – отримання даних всіх користувачів;
- deleteUser() – видалення користувача;
- verifiedUser() – перевірка користувача;

- createPost() – створення поста;
- updatePost() – оновлення поста;
- deletePost() – видалення поста;
- getPost() – отримання даних про пости;
- getAllPost () – отримання даних про всі пости;
- getOnePosts() – отримання даних про конкретний пост;
- createComment() – створення коментарю;
- updateComment() – оновлення коментарю;
- deleteComment() – видалення коментарю;
- getAllComments() – отримання даних про всі коментарі;
- getComment() – отримання даних про коментар.

POST – клас, який містить методи для роботи із постами.

Поля:

- title (тип string) – заголовок поста;
- caption (тип string) – опис поста;
- slug (тип string) – унікальний ідентифікатор поста;
- body (тип object) – контент поста;
- photo (тип string) – фото поста;
- user (тип Schema.Types.ObjectId) – ідентифікатор користувача;
- tags (тип string) – теги;
- categories (тип Schema.Types.ObjectId) – категорія.

Методи:

- createPost() – створити пост;
- updatePost() – оновити пост;
- deletePost() – видалити пост;
- getPost() – отримання даних про конкретний пост;
- getAllPost() – отримання всіх постів з бази даних;
- getOnePost() – отримання конкретної публікації з бази даних.

Comment – клас, який містить методи для роботи з коментарями.

Поля:

- user (тип Schema.Types.ObjectId) – ідентифікатор користувача;
- desc(тип string) – опис коментаря;
- post (тип Schema.Types.ObjectId) – ідентифікатор поста;
- check (тип boolean) – статус коментаря;
- parent (тип Schema.Types.ObjectId) – батьківський ідентифікатор;
- replyOnUser (тип Schema.Types.ObjectId) – батьківський ідентифікатор.

Методи:

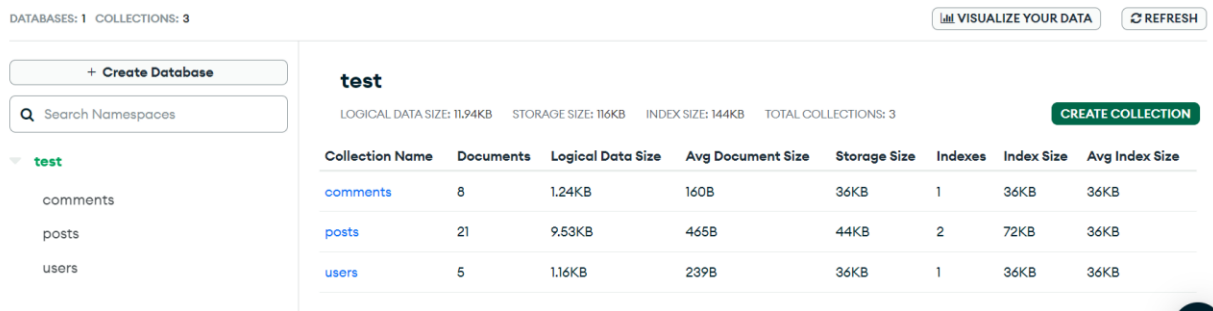
- createComment() – створення коментаря;
- updateComment() – оновлення коментаря;
- deleteComment() – видалення коментаря;
- getAllComment() – отримання даних про всі коментарі;
- getComment() – отримання інформації про окремий коментарь.

2.2. Розробка бази даних системи

Для збереження та подальшого використання даних було розроблено таблиці бази даних MongoDB Atlas для кожного зі змінюваних сутностей, таких як:

1. users – таблиця в якій знаходиться інформація про користувачів;
2. posts – таблиця в якій знаходиться інформація про програми допомоги;
3. comments – таблиця в якій знаходиться інформація про коментарі;

Тож, база даних складається з 3 таблиць (рис.2.5): користувачі, постів та коментарів.



Collection Name	Documents	Logical Data Size	Avg Document Size	Storage Size	Indexes	Index Size	Avg Index Size
comments	8	1.24KB	160B	36KB	1	36KB	36KB
posts	21	9.53KB	465B	44KB	2	72KB	36KB
users	5	1.16KB	239B	36KB	1	36KB	36KB

Рисунок 2.5 – Діаграма бази даних

Нижче буде наведено опис кожної з таблиць бази даних (див. табл. 2.1 - 2.4).

Таблиця 2.1

Назва таблиці	Призначення
users	Користувачі
posts	Публікації
comments	Коментарі

Таблиця 2.2

users		
Поле	Тип	Призначення
id	ObjectId	Унікальний ідентифікатор
avatar	String	Фото користувача
name	String	Ім'я користувача
password	String	Пароль користувача
verified	Boolean	Перевірка користувача
admin	Boolean	Статус користувача
createAt	Date	Дата та час створення
updateAt	Date	Дата та час видалення

Колекція "Users" використовується для зберігання і керування даними користувачів, які мають доступ до функцій інформаційного додатку. Основна мета цієї таблиці – забезпечити контроль доступу та аутентифікацію користувача.

Таблиця 2.3

comments		
Поле	Тип	Призначення
Id	ObjectId	Унікальний ідентифікатор
users	ObjectId	Унікальний ідентифікатор користувача
desc	String	Контент коментаря
post	ObjectId	Ідентифікатор поста
check	Boolean	Статус коментаря
parent	ObjectId	Батьківський ідентифікатор
replyOnUser	ObjectId	Ідентифікатор користувача
createAt	Date	Дата та час створення
updateAt	Date	Дата та час видалення

Колекція "Comments" використовується для зберігання і керування даними коментарів. Основна мета цієї таблиці - забезпечити контроль доступу та взаємодії з коментарями.

Таблиця 2.4

posts		
Поле	Тип	Призначення
Id	ObjectId	Унікальний ідентифікатор
title	String	Заголовок
caption	String)	Опис
slug	String	Ідентифікатор поста
body	String	Контент публікації
photo	String	Фото поста
user	ObjectId	Ідентифікатор користувача
tags	Array	Теги
categories	Array	Категорія
createAt	Date	Дата та час створення
updateAt	Date	Дата та час видалення

Колекція "Posts" використовується для зберігання і керування даними постів. Основна мета цієї таблиці - забезпечити контроль доступу та взаємодії над постами.

Реалізована база даних відповідає всім вимогам, і забезпечує доступ і взаємодію для користувача та адміністратора. База даних складається з 3 основних колекцій: "Posts", "Users", "Comments". Дані колекції виконують важливу роль у збереженні та організації роботи додатку.

2.3. Проектування та реалізація алгоритмів роботи системи

Головними компонентами в додатку є взаємодії зі статтями автора (адміністратором) та дії зі статтями зі сторони користувачів. Розглянемо алгоритм роботи адміністратора (рис.2.6):

- Автентифікація в обліковому записі з роллю Адміністратор;
- створення постів, користувачів, коментарів;
- збереження створених сутностей у базу даних;
- перегляд, редагування, видалення певних елементів;
- збереження оновленої інформації;
- Керування правами доступу.

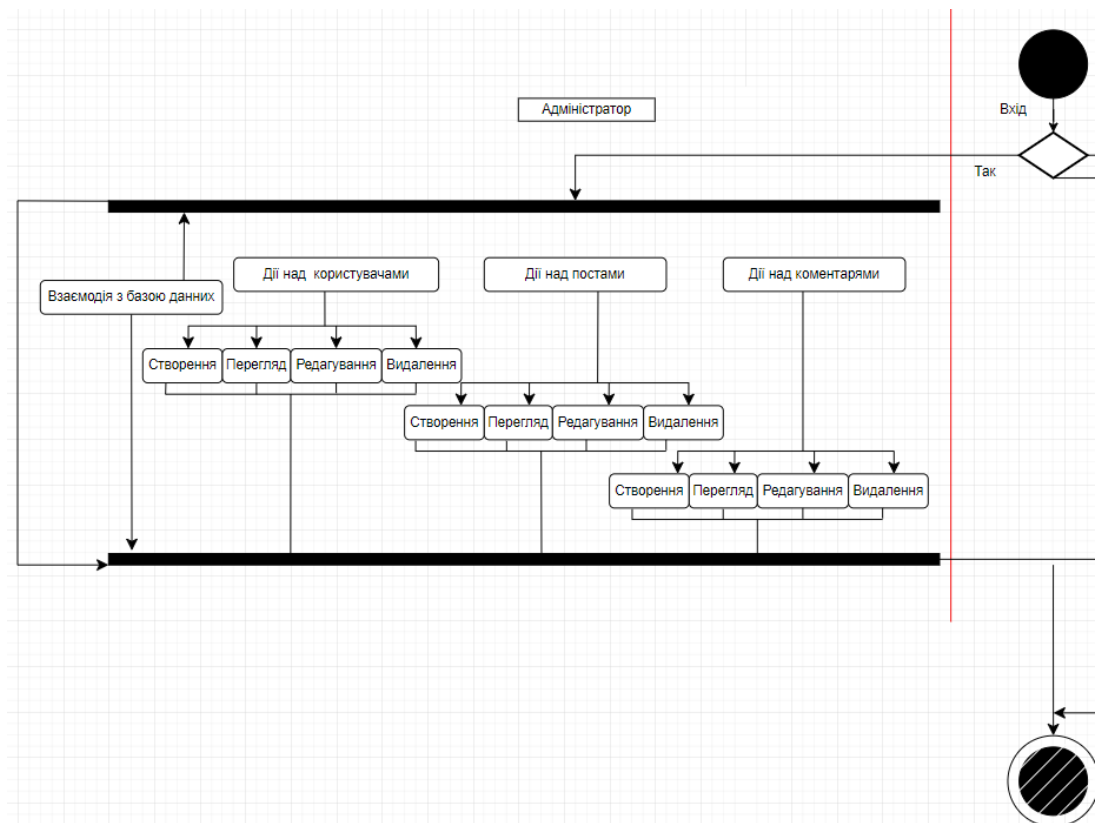


Рисунок 2.6 – Активність адміністратора

Розглянемо алгоритм роботи користувача (рис.2.7):

- Автентифікація в обліковому записі з роллю користувача;
- взаємодія зі постами(перегляд, коментарі);
- збереження інформації про виконані дії у базу даних;
- перегляд, редагування, видалення коментарів;

- збереження оновленої інформації;
- налаштування особистого кабінету.

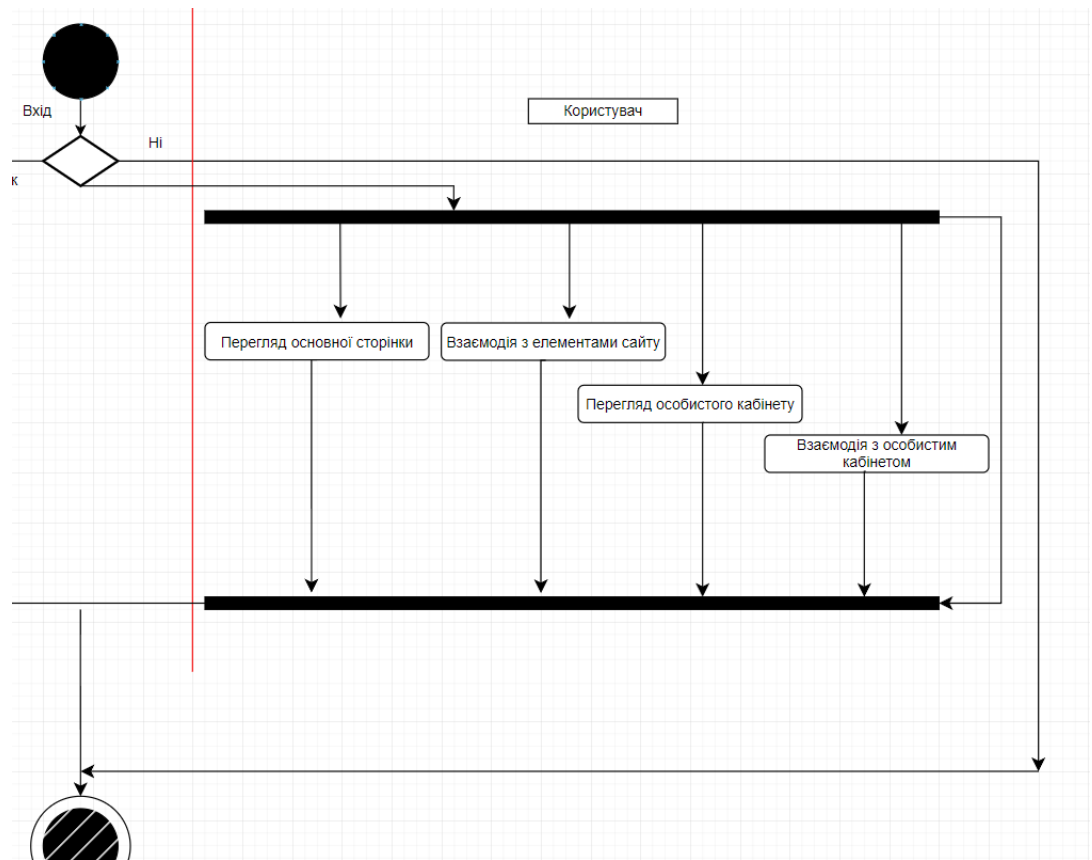


Рисунок 2.7– Активність користувача

Отже, було розглянути можливі варіанти доступу до функціоналу додатку та функціональних можливостей залежно від ролі користувача. Система авторизації, надає відповідний набір прав та функцій.

2.4. Реалізація функціоналу інформаційного додатка

1.Була реалізована функція, яка відповідає за підключення до бази даних. Ця функція використовує параметри, зазначені в конфігураційному файлі .env, для налаштування з'єднання та його параметрів.

2.Першим кроком до розробки додатка буде реалізація маршрутів. Код представлено у Додатку А. Побудувавши ієрархію маршрутів переходимо до створення контролерів. Розглянемо контролери користувача.

3.Контролер `UserController` представляє собою модуль, який містить функції для реєстрації, аутентифікації та управління користувачами в додатку.

Основні функції включають реєстрацію користувача, вхід у систему, отримання профілю користувача, оновлення профілю, зміну аватарки, отримання списку всіх користувачів, видалення користувача та підтвердження облікового запису користувача.

1.registerUser: Функція для реєстрації нового користувача. Перевіряє, чи існує користувач з вказаною електронною поштою. Якщо користувач не знайдений, створює новий запис користувача та повертає його дані та JWT-токен.

```
const registerUser = async (req, res, next) => {
  try {
    const {name, email, password} = req.body;
    let user = await User.findOne({email});
    if(user) {
      throw new Error("Користувач вже зареєструвався");
    }
    user = await User.create({
      name,
      email,
      password,
    });
    return res.status(201).json({
      _id: user._id,
      avatar: user.avatar,
      name: user.name,
      email: user.email,
      verified: user.verified,
      admin: user.admin,
      token: await user.generateJWT(),
    });
  }
  catch (error) {
    next(error);
  }
};
```

2. loginUser: Функція для аутентифікації користувача. Перевіряє, чи існує користувач з вказаною електронною поштою. Якщо користувач знайдений, перевіряє введений пароль. Якщо пароль вірний, повертає дані користувача та JWT-токен.

```

const loginUser = async (req, res, next) => {
  try {
    const {email, password} = req.body;
    let user = await User.findOne({email});
    if(!user){
      throw new Error("Електронна пошта не знайдена");
    }
    if (await user.comparePassword(password)) {
      return res.status(201).json({
        _id: user._id,
        avatar: user.avatar,
        name: user.name,
        email: user.email,
        verified: user.verified,
        admin: user.admin,
        token: await user.generateJWT(),
      });
    } else {
      throw new Error("Невірна адреса електронної пошти або пароль");
    }
  } catch (error) {
    next(error);
  }
};

```

3. updateProfilePicture: Функція для зміни аватарки користувача. Завантажує нову аватарку користувача, зберігає її у файловій системі та оновлює посилання на аватарку користувача в базі даних. Повертає оновлені дані користувача та JWT-токен.

```

const updateProfile = async (req, res, next) => {
  try {
    let user = await User.findById(req.user._id);

    if(!user) {
      throw new Error("Користувач не знайдений");
    }
    user.name = req.body.name || user.name;
    user.email = req.body.email || user.email;

    if(req.body.password && req.body.password.length < 6) {
      throw new Error("Пароль має бути не менше 6 символів");
    } else if (req.body.password) {
      user.password = req.body.password;
    }
    const updatedUserProfile = await user.save();
    res.json({
      _id: updatedUserProfile._id,

```

```

        avatar: updatedUserProfile.avatar,
        name: updatedUserProfile.name,
        email: updatedUserProfile.email,
        verified: updatedUserProfile.verified,
        admin: updatedUserProfile.admin,
        token: await updatedUserProfile.generateJWT(),
    });

    } catch (error) {
        next(error);
    }
};

```

4. **verifiedUser**: Функція для підтвердження облікового запису користувача. За допомогою ідентифікатора користувача знаходить користувача в базі даних та оновлює його статус підтвердження. Повертає оновлені дані користувача.

```

const verifiedUser = async (req, res, next) => {
    try {
        const {verified} = req.body;
        const user = await User.findById({_id: req.params._id});

        if(!user) {
            const error = new Error("Користувача не знайдено");
            return next(error);
        }
        user.verified = verified || user.verified;
        const verifiedUser = await user.save();
        return res.json(verifiedUser);
    } catch (error) {
        next(error)
    }
};

```

4.Контролер postController відповідає за роботу зі постами, їх створення, редагування, перегляд та видалення. Цей модуль для створення, оновлення, видалення та отримання постів в блозі. Основні функції, які виконує цей модуль, описані нижче:

1.createPost: Ця функція створює новий пост в блозі. Вона отримує дані запиту, такі як заголовок, підпис, зображення, текст та ідентифікатор користувача. Функція спочатку ініціалізує об'єкт Post з отриманими даними. Далі відбувається завантаження зображення, яке прикріплене до поста, за допомогою uploadPicture. Якщо завантаження успішне, зображення зберігається у відповідне

поле `post.photo`, і пост зберігається в базі даних. Якщо завантаження не вдалося, генерується помилка.

```
const createPost = async (req, res, next) => {
  try {

    const upload = uploadPicture.single("photo");
    console.log(req.body)
    const post = new Post({
      title: req.body.title,
      caption: req.body.caption,
      slug: uuidv4(),
      body: req.body.body,
      photo: "" ,
      user: req.user._id,
    });

    const handleUpdatePostData = async (data) => {
      console.log(post)
      const createdPost = await post.save();
      return res.json(createdPost);
    };
    upload(req, res, async function (err) {
      if (err) {
        const error = new Error(
          "Під час завантаження сталася невідома помилка" + err.message
        );
        next(error);
      } else {
        if (req.file) {
          let filename;

          filename = post.photo;
          if (filename) {
            fileRemover(filename);
          }
          post.photo = req.file.filename;
          handleUpdatePostData(req.body.document);
        } else {
          let filename;

          filename = post.photo;
          post.photo = "";
          console.log(post.photo);
          fileRemover(filename);
          handleUpdatePostData(req.body.document);
        }
      }
    });
  } catch (error) {
```

```

    console.log(error);
    next(error);
  }
};

```

2.updatePost: Ця функція оновлює існуючий пост в блозі. Вона отримує ідентифікатор поста, знаходить відповідний пост у базі даних і оновлює його дані з отриманими даними запиту. Аналогічно до createPost, функція також може завантажувати нове зображення для оновлення. Оновлений пост зберігається в базі даних.

```

const updatePost = async (req, res, next) => {
  try {
    const post = await Post.findOne({ _id: req.params.slug });
    console.log(post);
    if (!post) {
      const error = new Error("Повідомлення не знайдено");
      next(error);
      return;
    }

    const upload = uploadPicture.single("postPicture");
    const handleUpdatePostData = async (data) => {
      console.log(post);
      console.log(11)
      const { title, caption, slug, body, tags, categories } = JSON.parse(data);
      post.title = title || post.title;
      post.caption = caption || post.caption;
      post.slug = slug || post.slug;
      post.body = body || post.body;
      post.tags = tags || post.tags;
      post.categories = categories || post.categories;
      const newPost = { ...post };
      const updatedPost = await Post.findByIdAndUpdate(
        req.params.slug,
        newPost
      );
      return res.json(updatedPost);
    };
    upload(req, res, async function (err) {
      if (err) {
        const error = new Error(
          "Під час завантаження сталася невідома помилка" + err.message
        );
        next(error);
      } else {
        if (req.file) {
          let filename;

```

```

        filename = post.photo;
        if (filename) {
            fileRemover(filename);
        }
        post.photo = req.file.filename;
        console.log(post);
        handleUpdatePostData(req.body.document);
    } else {
        let filename;

        filename = post.photo;
        post.photo = "";
        fileRemover(filename);
        handleUpdatePostData(req.body.document);
    }
    });
} catch (error) {
    next(error);
}
};

```

3.getAllPosts: Ця функція отримує всі пости з блогу. Вона приймає параметри сторінки і кількості постів на сторінці, і повертає список постів з додатковою інформацією про користувача, який створив кожен пост, а також загальну кількість постів у блозі.

```

const getAllPosts = async (req, res, next) => {
    try {
        const page = req.query.page || 1;
        const total = req.query.total || 10;
        const offset = page * total - total;
        const posts = await Post.find({}).skip(offset).limit(total).populate([
            {
                path: "user",
                select: ["avatar", "name", "verified"],
            },
        ]);
        const count = await Post.countDocuments();

        res.json({posts:posts,count:count});
    } catch (error) {
        next(error);
    }
};

```

5. Контролер commentController відповідає за зміни у таблиці бази даних comments. Відповідний модуль для створення, оновлення, видалення та

					ІПЗ.КР.Б – 121 – 23 – ПЗ	
						40

отримання коментарів в блозі. Основні функції, які виконує цей модуль, описані нижче:

1.createComment: Ця функція створює новий коментар до поста в блозі. Вона отримує дані запиту, такі як опис коментаря, ідентифікатор поста, ідентифікатор батьківського коментаря та користувача, на якого було залишено відповідь. Спочатку функція перевіряє існування поста за його ідентифікатором. Якщо пост не знайдений, генерується помилка. Далі створюється новий об'єкт коментаря з отриманими даними, і цей коментар зберігається в базі даних. Після успішного збереження коментаря повертається відповідь з збереженим коментарем.

```
const createComment = async (req,res,next) => {
  try {
    const {desc, slug, parent, replyOnUser} = req.body;

    const post = await Post.findOne({slug: slug});

    if(!post) {
      const error = new Error("Повідомлення не знайдено");
      return next(error);
    }

    const newComment = new Comment({
      user: req.user._id,
      desc,
      post: post._id,
      parent,
      replyOnUser,
    });

    const savedComment = await newComment.save();
    return res.json(savedComment);
  } catch (error) {
    next(error)
  }
};
```

2.updateComment: Ця функція оновлює існуючий коментар в блозі. Вона отримує ідентифікатор коментаря та дані запиту, такі як опис коментаря та статус коментаря. Функція знаходить коментар за його ідентифікатором у базі даних, оновлює його дані з отриманими даними запиту та зберігає оновлений

коментар в базі даних. Після успішного оновлення коментаря повертається відповідь з оновленим коментарем.

```
const updateComment = async (req,res,next) => {
  try {
    const {desc,check} = req.body;

    const comment = await Comment.findById(req.params.commentId);

    if(!comment) {
      const error = new Error("Коментар не знайдено");
      return next(error);
    }

    comment.desc = desc || comment.desc;

    comment.check = check || comment.check;

    const updatedComment = await comment.save();
    return res.json(updatedComment);

  } catch (error) {
    next(error)
  }
};
```

3.deleteComment: Ця функція видаляє існуючий коментар в блозі. Вона отримує ідентифікатор коментаря, знаходить коментар за його ідентифікатором у базі даних та видаляє його. Крім того, видаляються всі коментарі, що мають цей коментар як батьківський. Після успішного видалення коментаря повертається відповідь з повідомленням про успішне видалення.

```
const deleteComment = async (req,res,next) => {
  try {

    const comment = await Comment.findByIdAndDelete(req.params.commentId);
    await Comment.deleteMany({parent: comment._id});

    if(!comment) {
      const error = new Error("Коментар не знайдено");
      return next(error);
    }

    return res.json({
      message: "Коментар успішно видалено",
    });
  }
};
```

```

    } catch (error) {
      next(error)
    }
  };

```

6.Бібліотека Axios. Також, для роботи з базою даних ми часто використовуємо бібліотеку Axios. Для прикладу можемо розглянути отримання даних для відображення списку постів на сторінці. Інші запити до бази даних мають аналогічний вигляд.

```

import axios from "axios";

export const getAllPosts = async (page=1) => {
  try {
    const {data} = await axios.get("/api/post",{
      params: {
        page:page
      }
    });

    return data;
  } catch (error) {
    console.log(error)
    if (error.response && error.response.data.message)
      throw new Error(error.response.data.message);
    throw new Error(error.message);
  }
};

```

7.Схема користувача (UserSchema) для моделі користувача у базі даних MongoDB за допомогою пакету Mongoose. Нижче наведений код створює схему та модель користувача для зберігання та обробки даних про користувачів у базі даних MongoDB. Він також надає додаткові методи для генерації JWT токена та перевірки пароля користувача. А також використовується бібліотека bcryptjs використовуються для забезпечення безпеки паролів, зокрема при реєстрації користувачів, зберіганні паролів у базі даних та перевірці введеного користувачем пароля при аутентифікації.

```

import {Schema,model} from "mongoose";
import {hash,compare} from "bcryptjs";
import {sign} from "jsonwebtoken";

const UserSchema = new Schema({
  avatar: {type: String, default: ""},

```

```

    name: {type: String, required: true},
    email: {type: String, required: true},
    password: {type: String, required: true},
    verified: {type: Boolean, default: false},
    verificationCode: {type: String, required: false},
    admin: {type: Boolean, default: false},
  },
  {timestamps: true}
);
UserSchema.pre("save", async function (next) {
  if(this.isModified("password")) {
    this.password = await hash(this.password, 10);
    return next();
  }
  return next();
});

UserSchema.methods.generateJWT = async function() {
  return await sign({id:this._id}, process.env.JWT_SECRET, {
    expiresIn: '30d',
  });
};

UserSchema.methods.comparePassword = async function (enteredPassword) {
  return await compare(enteredPassword, this.password);
};

const User = model("User", UserSchema);
export default User;

```

8. Автентифікації користувача вона використовує дві функції: `authGuard` і `adminGuard`. `authGuard` виконує перевірку автентифікації користувача. Вона перевіряє наявність токена авторизації у заголовках запиту. Якщо токен присутній і має правильний формат (починається з "Bearer"), то він розшифровується за допомогою секретного ключа `process.env.JWT_SECRET`. Після розшифрування отримується ідентифікатор користувача, який використовується для пошуку користувача в базі даних.

`adminGuard`: Ця функція виконує перевірку прав доступу адміністратора. Вона перевіряє, чи присутній об'єкт `req.user` (попередньо автентифікований користувач) і чи має він поле `admin` зі значенням `true`. Якщо ці умови виконуються, то управління передається до наступної функції з `middleware`.

Якщо ж умови не виконуються, генерується помилка авторизації зі статусом 401 Unauthorized.

```
import {verify} from "jsonwebtoken";
import User from "../models/User";
export const authGuard = async (req, res, next) => {
  if (
    req.headers.authorization &&
    req.headers.authorization.startsWith("Bearer")
  ) {
    try {
      const token = req.headers.authorization.split(" ")[1];
      const { id } = verify(token, process.env.JWT_SECRET);
      req.user = await User.findById(id).select("-password");
      next();
    } catch (error) {
      let err = new Error(" Не авторизовано, маркер не вдається");
      err.statusCode = 401;
      next(err);
    }
  } else {
    let error = new Error("Не авторизовано, немає маркера");
    error.statusCode = 401;
    next(error);
  }
};
export const adminGuard = (req, res, next) => {
  if (req.user && req.user.admin) {
    next();
  } else {
    let error = new Error("Не авторизований як адміністратор");
    error.statusCode = 401;
    next(error);
  }
};
```

9.Маршрутизація: Імпортуючи модуль express створюється об'єкт роутера. Це дозволяє групувати різні маршрути в один об'єкт. Після цього імпортуються функції з інших модулів, які будуть використовуватись для обробки запитів користувачів. Зокрема, імпортується registerUser, loginUser, userProfile, updateProfile, updateProfilePicture, getAllUsers, deleteUser та verifiedUser з модулів userControllers і authMiddleware. Далі визначаються маршрути за допомогою методу router.post, router.get, router.put і router.route. Далі визначаються маршрути за допомогою методу router.post, router.get, router.put і router.route. Загалом, цей

код налаштовує маршрути для реєстрації користувачів, входу в систему, отримання та оновлення профілю користувача, завантаження та оновлення зображення профілю, отримання списку всіх користувачів, видалення користувача та підтвердження користувача.

```
import express from "express";
const router = express.Router();
import { registerUser, loginUser, userProfile , updateProfile ,
updateProfilePicture, getAllUsers, deleteUser, verifiedUser} from
"./controllers/userControllers";
import { adminGuard, authGuard } from "../middleware/authMiddleware";

router.post('/register', registerUser);
router.post('/login', loginUser);
router.get('/profile', authGuard, userProfile);
router.put('/updateProfile', authGuard, updateProfile);
router.put('/updateProfilePicture', authGuard, updateProfilePicture);
router.route("/").get(authGuard,getAllUsers);
router.route('/:id').delete( authGuard, adminGuard,deleteUser).put(authGuard,
adminGuard,verifiedUser)
export default router;
```

Також передбачена автентифікація та авторизація для захисту деяких маршрутів від несанкціонованого доступу.

Висновки до розділу 2

Описаний веборієнтований інформаційний додаток надає ряд функцій, таких як можливість додавання постів, коментарів і збереження даних про користувачів. Для цього використовується набір API-шляхів (api routes), які відповідають за обробку запитів для постів, коментарів і облікових записів користувачів. В додатку також існує база даних, яка відповідає вимогам і забезпечує збереження та спільний доступ до інформації.

База даних додатку складається з трьох колекцій, які зберігають різні типи даних, необхідних для правильної роботи додатку. У описі коду основна увага приділена функціям, які відповідають за збереження даних у колекціях постів і коментарів, а також функціям, які забезпечують підключення до бази даних.

В пункті 2.4 детально розглядається частина функціонала, точніше контролери які відповідають за створення сутностей

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З СИСТЕМОЮ ДОДАТКА

3.1 Порядок встановлення та налаштування системи

Веборієнтований інформаційний додаток є вебсайтом, який розроблено для використання у різних браузерях. Щоб користуватись цим сайтом, необхідно мати телефон, планшет або ноутбук з наступними мінімальними вимогами:

- Операційна система: Windows 8 і вище, macOS 10.11 і вище.
- Процесор: 1,3-3,8 ГГц.
- Android версія 6.0 і вище.
- 2 ГБ оперативної пам'яті (ОЗУ).
- Будь-який веббраузер.

Для розробленого додатку не потрібне спеціалізоване обладнання або додаткові налаштування для розгортання. Достатньо стандартного набору інструментів, які використовуються для більшості сучасних вебсайтів на будь-якій операційній системі. Це робить використання додатку зручним і покращує досвід користувачів.

Переходячи до наступного етапу - розробки діаграми розгортання, яка описує архітектуру програмного комплексу. Діаграма розгортання включає зображення вузлів виконання програмних компонентів у реальному часі, процесів та об'єктів.

Діаграма розгортання використовується для візуалізації загальної конфігурації та топології розподіленої програмної системи. Вона надає інформацію про розташування компонентів на окремих вузлах системи. Крім того, діаграма розгортання показує фізичні з'єднання - шляхи передачі інформації між апаратними пристроями, які використовуються для реалізації системи. Ця діаграма спрямована на візуалізацію елементів і компонентів програми, які існують лише під час її виконання. На діаграмі розгортання відображаються лише компоненти програми, які є виконуваними файлами або динамічними бібліотеками. Компоненти, які не використовуються під час

виконання, не зображуються на цій діаграмі. Це означає, що компоненти, що складаються з початкових текстів програм, можуть бути присутніми лише на діаграмі компонентів, але не на діаграмі розгортання.

Діаграма розгортання включає графічні зображення процесорів, пристроїв, процесів та їх зв'язків. Вона надає повну картину про реалізацію системи в цілому, відображаючи всі її особливості, і відрізняється від діаграм логічного представлення. Діаграма розгортання є останнім етапом специфікації моделі ООП для конкретної програмної системи. Результат побудованої схеми показано на рисунку 3.1.

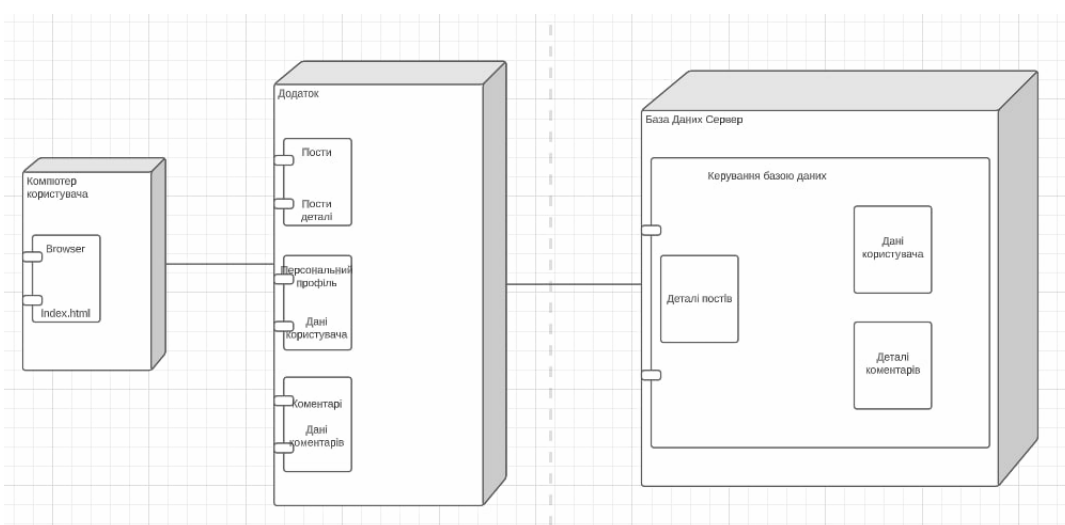


Рисунок 3.1 – Діаграма розгортання

Діаграма розгортання відображає структуру та зв'язки між різними вузлами (фізичними або логічними компонентами) у системі. Вона надає візуальне уявлення про конфігурацію та топологію системи, що допомагає зрозуміти, як компоненти взаємодіють між собою та зовнішніми елементами. На діаграмі розгортання можна побачити такі елементи:

1. Вузол (node) є ключовим компонентом системи, який відповідає за обробку запитів від користувачів і надання їм доступу до вебсторінок та інших ресурсів. Вузол включає в себе програмне забезпечення, яке здійснює обробку HTTP-запитів та забезпечує функціональність вебсайтів та додатків. Він може включати сервери, вебсервери, аплікаційні сервери або інші компоненти, які забезпечують взаємодію з користувачами.

2. Вузол "Користувач" представляє обчислювальні пристрої, такі як телефони, планшети або ноутбуки, які використовуються користувачами для доступу до системи. Ці пристрої дозволяють користувачам переглядати і взаємодіяти з вебінтерфейсом інформаційного додатку. Вони служать інтерфейсом між користувачем і системою, де користувачі можуть вводити дані, переглядати вміст, виконувати дії та отримувати відповіді від системи.

3. Вузол "Додаток" включає програмний компонент, що виконується на вузлі і надає функціональні можливості додатку. Він включає в себе виконуваний файли або динамічні бібліотеки, які забезпечують роботу додатку та його інтерфейс. Цей компонент виконує необхідні операції, обробляє дані та взаємодіє з іншими компонентами системи для забезпечення функціональності додатку. Вузол "Додаток" може включати додатки, сервіси або інші програмні компоненти, що виконуються на вузлі.

4. Вузол "База даних Сервер" представляє фізичну або віртуальну систему, що використовується для зберігання та управління базою даних інформаційного додатку. Цей вузол включає необхідне програмне забезпечення та обладнання, включаючи сервер бази даних, системи управління базами даних (СУБД) та засоби забезпечення безпеки. Вузол "База даних Сервер" виконує функцію надійного зберігання та обробки даних, що використовуються в інформаційному додатку. Це дозволяє забезпечити доступ до даних, виконувати запити до бази даних, зберігати та оновлювати інформацію про пости, дані користувачів, а також забезпечувати безпеку даних.

3.2 Структура інтерфейсу. Інтерфейс та порядок роботи з веборієнтованим інформаційним додатком

Головна сторінка сайту розташована за адресою <http://localhost:5000/> і містить перелік основних посилань у шапці. Користувач може легко орієнтуватися та переходити до різних розділів сайту. На головній сторінці розміщена інформація, яка відображає загальний зміст сайту та важливі новини.

Крім того, у шапці присутні посилання, які дозволяють користувачу швидко перейти до інших важливих сторінок. Для отримання розширених можливостей і доступу до особистих налаштувань, користувач може увійти до свого облікового запису. У даний момент демонстрація роботи сайту проводиться з облікового запису адміністратора, що надає повний доступ до управління контентом та іншими функціями.

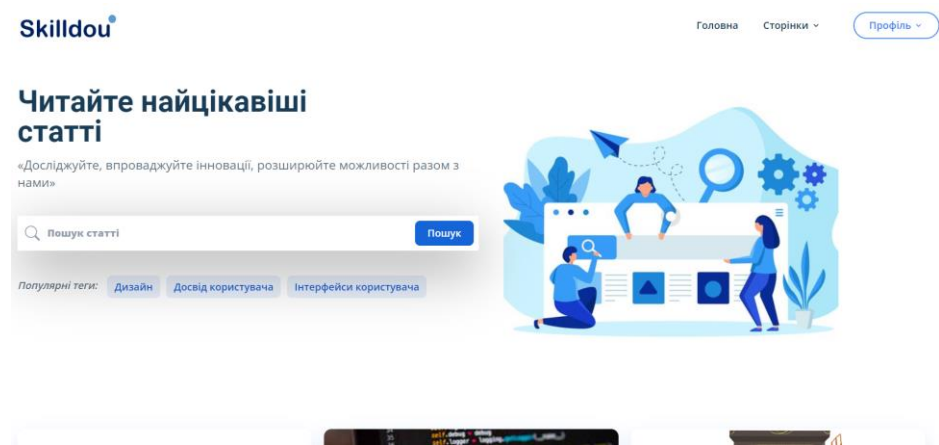


Рисунок 3.2 – Головна сторінка

На головній сторінці (див. рис.3.2) також представлені найновіші пости, що дозволяють користувачам бути в курсі останніх трендів і актуальних новин. Ця функція надає змогу зручно ознайомитися зі свіжим контентом без необхідності переходу на інші сторінки (див. рис.3.3). Нові пости можуть містити інформацію про програмні продукти, IT-івенти, статті або інші цікаві матеріали, які можуть зацікавити користувача. Таким чином, головна сторінка стає центром актуальної інформації, що сприяє залученню користувачів і забезпечує зручний досвід використання сайту.

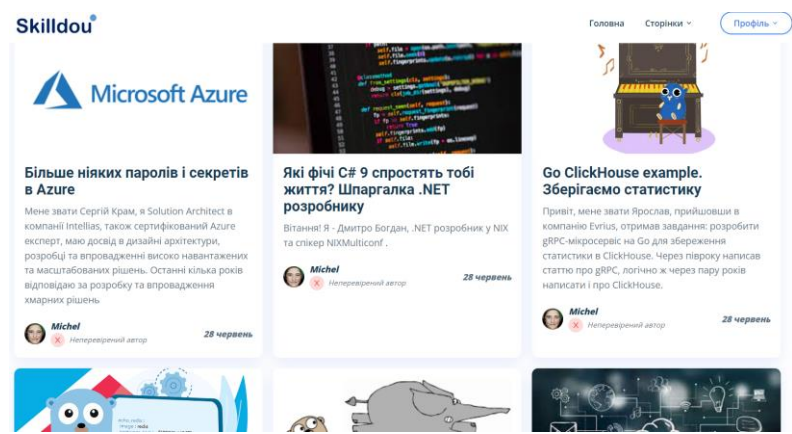


Рисунок 3.3 – Публікації

Крім того, користувач має можливість переглянути обрану публікацію та детально ознайомитись з її контентом (див. рис.3.4). Після вибору певної публікації на головній сторінці, користувач може клікнути на неї, щоб перейти до повної версії публікації. Там він зможе прочитати всі деталі, опис, коментарі і, в деяких випадках, відповісти на неї або поділитись з іншими користувачами.

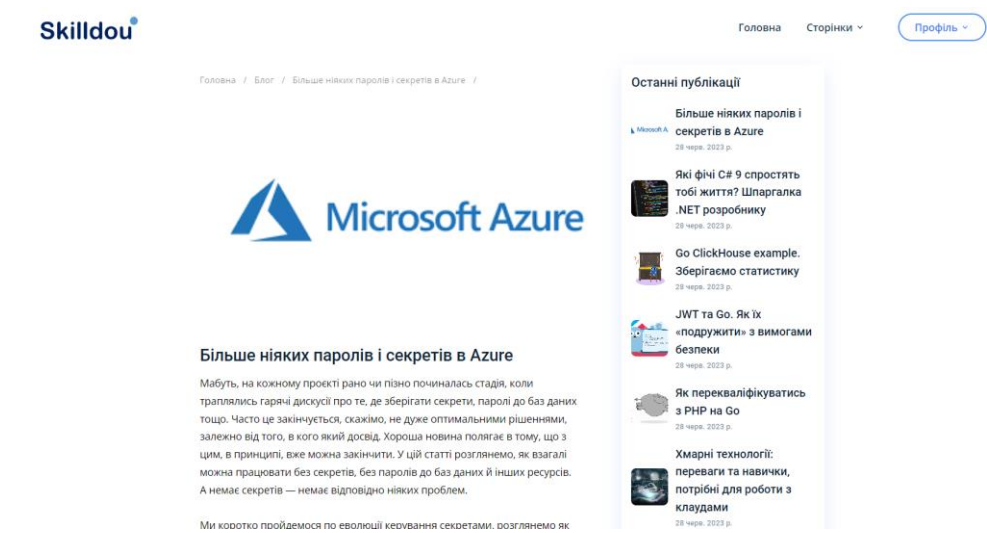


Рисунок 3.4 – Детальний зміст публікації

Прогорнувши пост до кінця, користувач побачить секцію з коментарями, (рис.3.5). Ця секція дозволяє користувачам висловлювати свої думки, задавати питання або виражати зацікавлення щодо даної публікації. Всі коментарі відображаються у хронологічному порядку, надаючи можливість легко слідкувати за обговоренням та брати участь у ньому.

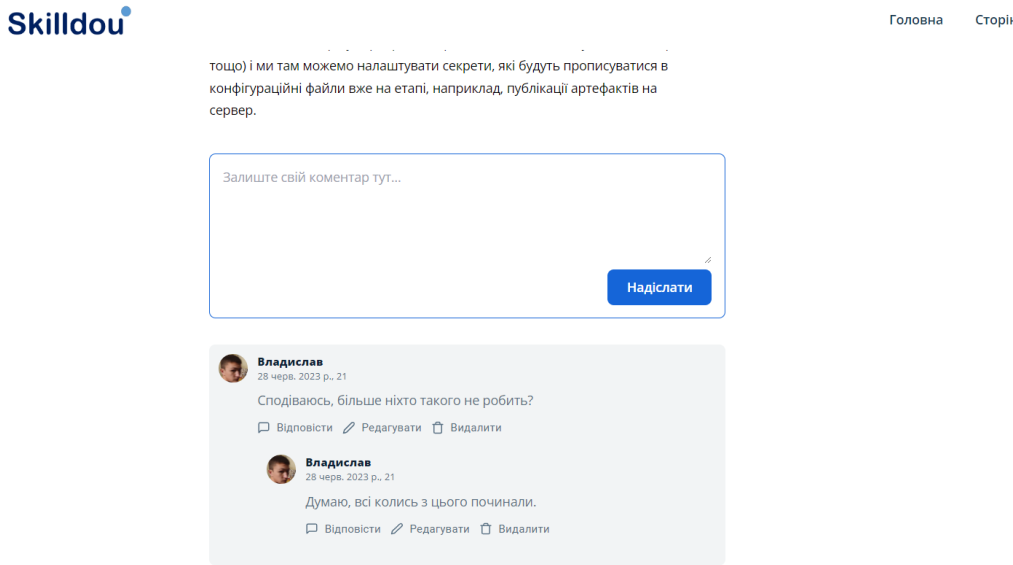


Рисунок 3.5 – Секція з коментарями

У боковому меню також присутня можливість переглянути останні публікації (див. рис. 3.6). Це дозволяє користувачеві швидко отримати доступ до найсвіжіших матеріалів, які були опубліковані на сайті. В цьому розділі користувач може переглянути заголовки або короткі огляди останніх публікацій та вибрати ту, яка зацікавила його найбільше. Це забезпечує зручний спосіб ознайомитись з останніми матеріалами, необхідними новинами або цікавим контентом, що допомагає користувачеві бути в тренді та знайти цікаву інформацію швидко та ефективно. Перегляд останніх публікацій в боковому меню сприяє полегшенню навігації та робить процес пошуку цікавого контенту більш доступним для користувача.

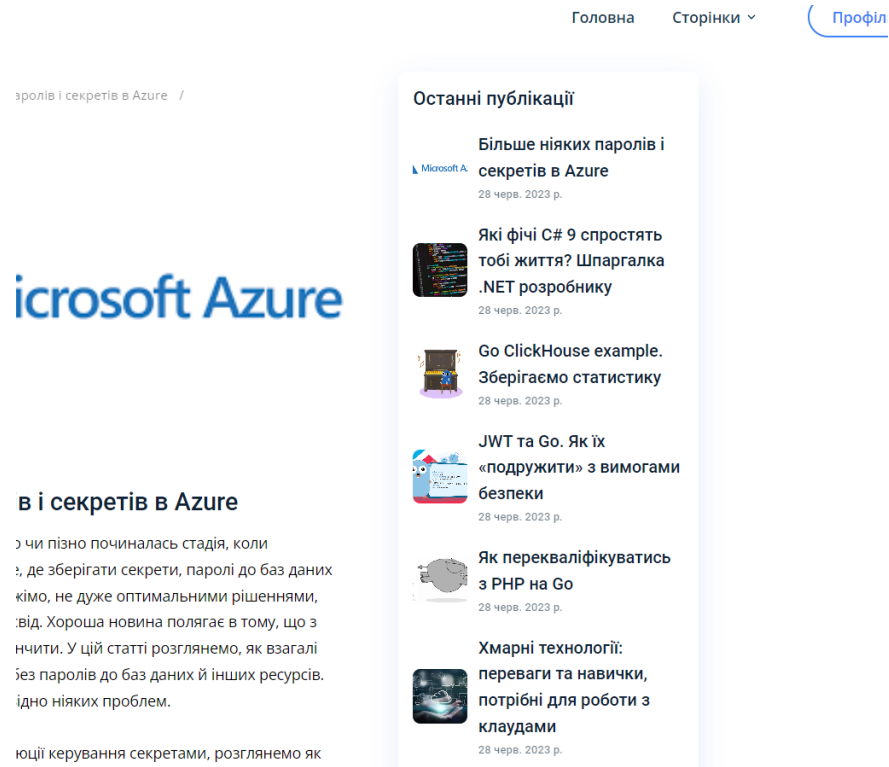


Рисунок 3.6 – Останні публікації

На сторінці допомоги користувач має можливість детально переглянути заклик про допомогу. Також користувач може вибрати заклики за категорією чи містом. Розглянемо детально заклик. Користувач бачить основну інформацію про створення заклика (час та дата), інформацію про проблему, автора та його контакти.

Перейшовши на вкладку “Про нас” користувач перейде за адресою <http://localhost:5000/about>, де він отримає детальнішу інформацію про

розробника сайту (див. рис.3.7). На сторінці "Про нас" користувач зможе дізнатись про мету та цілі інформаційного додатку.

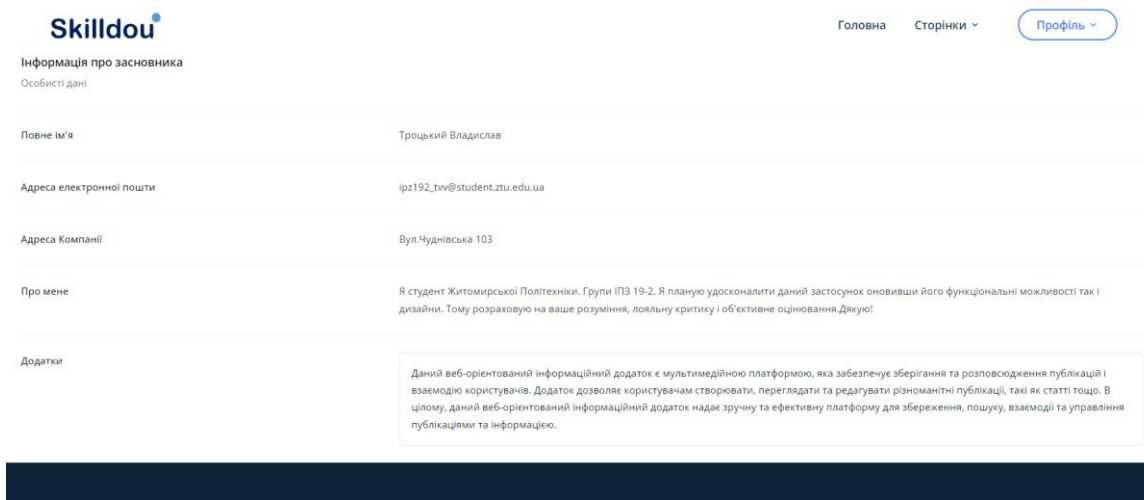


Рисунок 3.7 – Сторінка “Про нас”

Також є система пошуку публікацій (див. рис.3.8, рис.3.9). При введенні назви публікації, завантажується сторінка окремої публікації.

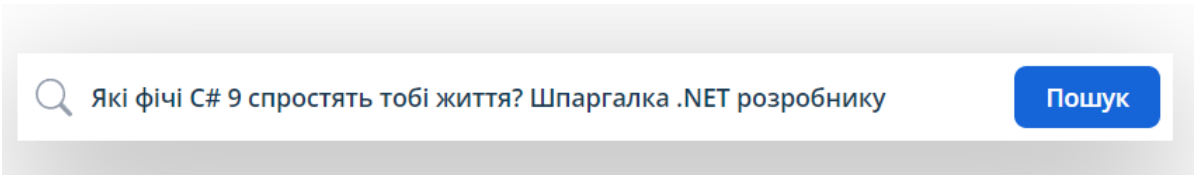


Рисунок 3.8 – Секція пошуку

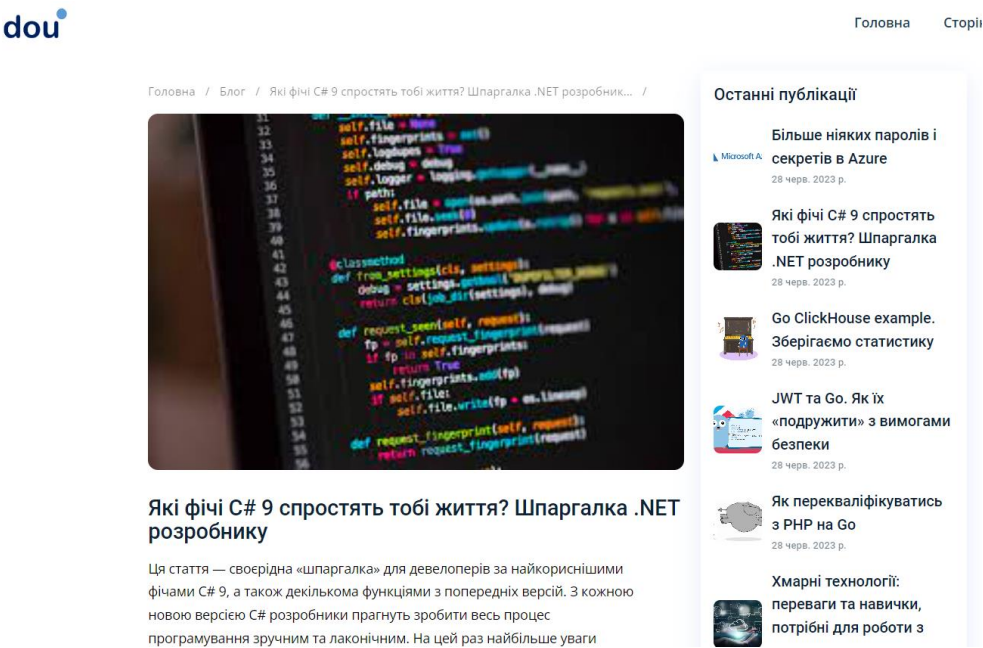


Рисунок 3.9 – Результат пошуку

Для успішної взаємодії з функціями додатку користувачу необхідно зареєструватися на сайті, (рис.3.10).

Зареєструватися

Ім'я

Електронна пошта

Пароль

Підтвердити пароль

Зареєструватися

У вас є обліковий запис?[Увійдіть зараз](#)

Рисунок 3.10 – Сторінка з формою реєстрації

При успішній авторизації користувач має можливість перейти до власного профілю (див. рис.3.11).

Видалити

Ім'я

Електронна пошта

Новий пароль

Оновити

Рисунок 3.11 – Профіль користувача

Користувач має власний профіль, до якого він може потрапити за посиланням у шапці сайту. На головній сторінці користувач оновлювати свої дані: фото профілю, ім'я, електронна адреса та пароль.

Також зареєстрований користувач має можливість створити публікацію, (див. рис.3.12).

Рисунок 3.12 – Сторінка створення публікації

Перейшовши на головну сторінку, користувач має можливість переглянути щойно створену публікацію (див. рис.3.13).

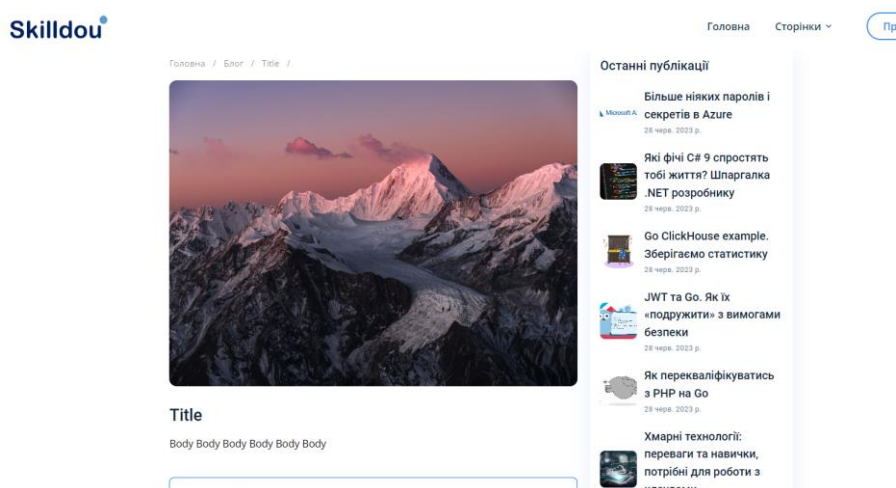


Рисунок 3.13 – Сторінка щойно створеної публікації

Перейшовши у вкладку “Допомога” користувач бачить перелік створених ним закликів про допомогу та кнопку для створення. Натиснувши на неї користувач буде відправлений на форму зі створенням заклику про допомогу.

Розглянемо можливості та функціонал адміністративного кабінету, (рис.3.14). При переході за посиланням з шапки сайту або за адресою <http://localhost:5000/admin>, буде відкрита головна сторінка адміністративного

кабінету. Головна сторінка відображає статистику постів, зареєстрованих користувачів та коментарі.

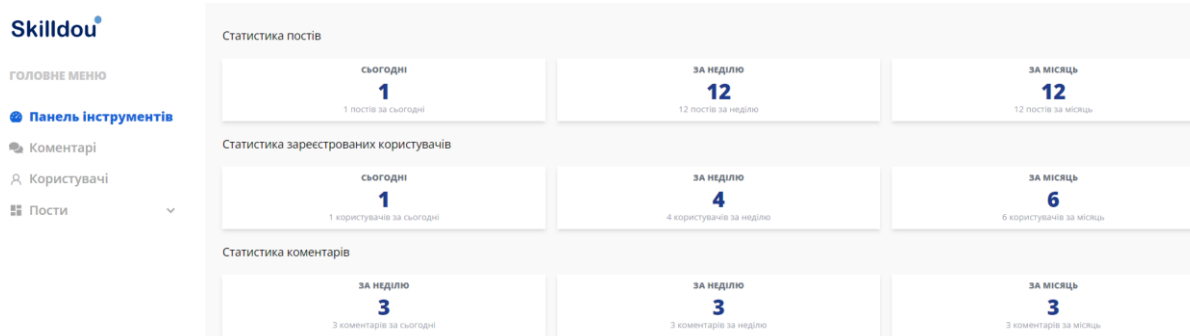


Рисунок 3.14 – Головна сторінка адміністративного кабінету

Розглянемо декілька можливих варіантів роботи з адміністративною панеллю. Наприклад, додавання та видалення “Коментарів” (див. рис.3.15). Адміністратор при переході на вкладку “Коментарі” бачить перелік всіх коментарів. Має можливість перевірити та додати , а також видалити коментар.

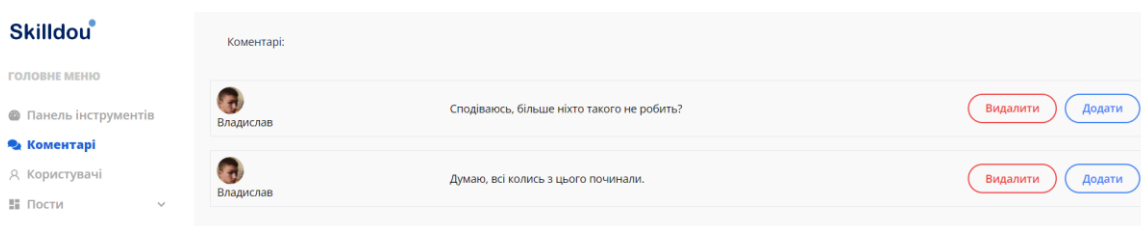


Рисунок 3.15 – Сторінка “Коментарі”

Натиснувши кнопку “Користувачі”, здійсниться перехід на сторінку зі всіма користувачами (див. рис.3.16).

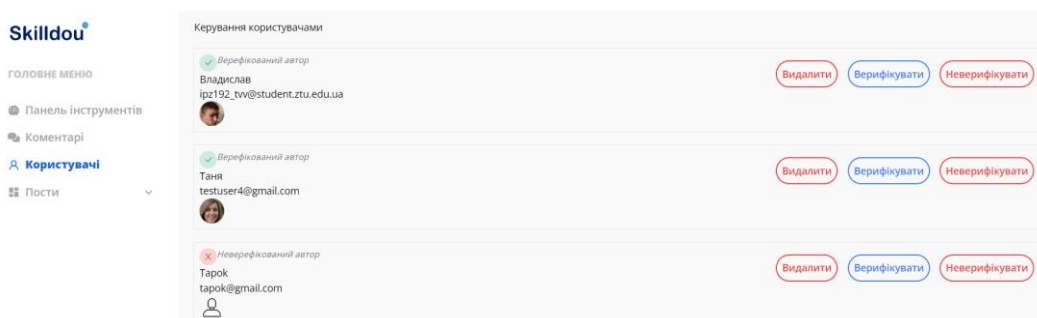


Рисунок 3.16 – Сторінка “Користувачів”

У додатку реалізовано видалення та верифікація користувачів де адміністратор надає статус користувачу. Статус користувача надає можливість

іншим користувачам переконатись, що користувач надає перевірену і актуальну інформацію у свої публікаціях.

Розглянемо елемент адміністративної панелі “Пости” (див. рис.3.17).

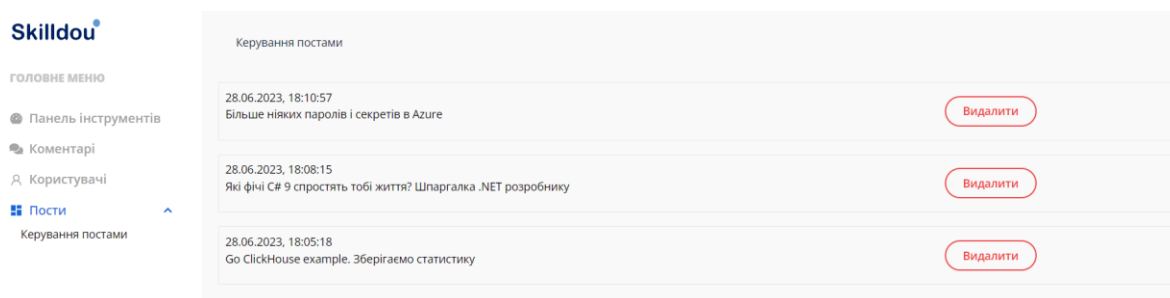


Рисунок 3.17 – Сторінка “Пости”

Реалізовано видалення публікацій на головній сторінці.

3.3 Тестування роботи програмного комплексу

Під час розробки та впровадження програмного продукту, велику роль відіграє успішне тестування. Цей етап має на меті виявлення помилок, перевірку функціональності та оцінку загальної якості та ефективності продукту. В цьому розділі будуть розглянуті тест-кейси для основних функцій та методів системи, а також надані результати загального тестування функціоналу, щоб забезпечити його надійну та зручну роботу.

1. Тест-кейс: реєстрація нового користувача:

передумови: вебдодаток відкритий у браузері;

кроки: користувач натискає на кнопку "Реєстрація", користувач вводить свої дані (ім'я, електронну пошту, пароль), користувач підтверджує реєстрацію;

очікуваний результат: користувач успішно зареєстрований і перенаправлений на головну сторінку;

фактичний результат: користувач успішно зареєстрований і перенаправлений на головну сторінку.

2. Тест-кейс: Пошук інформації:

передумови: користувач авторизований та знаходиться на головній

сторінці;

кроки: користувач вводить ключове слово в поле пошуку, користувач натискає кнопку "Пошук";

очікуваний результат: показані результати пошуку, що відповідають ключовому слову;

фактичний результат: показані результати пошуку, що відповідають ключовому слову.

3. Тест-кейс: Додавання коментаря до публікації

передумови: користувач авторизований та переглядає публікацію;

кроки: користувач прокручує до секції коментарів, користувач вводить текст коментаря у відповідне поле, користувач натискає кнопку "Додати коментар";

очікуваний результат: коментар успішно додано і відображено на сторінці публікації;

фактичний результат: коментар успішно додано і відображено на сторінці публікації.

4. Тест-кейс: Зміна особистих налаштувань:

передумови: користувач авторизований та знаходиться у власному профілі;

кроки: користувач клікає на посилання "Налаштування профілю", користувач змінює потрібні налаштування, користувач зберігає зміни;

очікуваний результат: зміни налаштувань успішно збережені.

фактичний результат: зміни налаштувань успішно збережені.

5. Тест-кейс: Перегляд деталей публікації:

передумови: користувач авторизований та знаходиться на головній сторінці зі списком публікацій;

кроки: користувач клікає на заголовок або зображення певної публікації;

очікуваний результат: користувач переходить на сторінку з детальною інформацією про публікацію;

фактичний результат: користувач переходить на сторінку з детальною інформацією про публікацію.

Висновки до розділу 3

У третьому розділі нашого дослідження детально розглянуто інтерфейс та процес роботи з веборієнтованим інформаційним додатком. Представлено аналіз кроків, необхідних для встановлення та налаштування параметрів системи. Підтверджено, що належне налаштування системи відіграє важливу роль у забезпеченні оптимальної роботи додатку і задоволення потреб користувачів.

Розглянуто структуру інтерфейсу та представлено основні елементи, які присутні в веборієнтованому інформаційному додатку. Продемонстровано, що належна організація інтерфейсу є ключовим аспектом, який допомагає користувачам легко орієнтуватися в додатку, знаходити потрібну інформацію та взаємодіяти з коментарями без непотрібних перешкод.

Проведено докладне дослідження процесу налаштування системи, структури інтерфейсу та протестували роботу нашого веборієнтованого інформаційного додатку. Результати показали, що правильний порядок дій та належне налаштування грають важливу роль у створенні функціонального та зручного додатку для користувачів. Тестування програмного комплексу ефективно допомогло запобігти можливим помилкам та недолікам, які могли б вплинути на користувачів. Всі виявлені помилки були виправлені, і на даний момент, сайт працює коректно та виконує всі поставлені завдання.

ВИСНОВКИ

У процесі виконання кваліфікаційної роботи бакалавра зосереджено увагу на проектуванні та реалізації веборієнтованого інформаційного додатку. Завдання включали аналіз схожих розробок, таких як Фокус, GOIT, DOU, для визначення основного функціоналу, такого як публікація постів, пошук, коментування, реєстрація користувачів та профілі. Вирішено додати до додатку додаткові функції, такі як сортування публікацій за тегами та перегляд за категоріями.

Для реалізації веборієнтованого інформаційного додатку використано такі інструменти: React.js для швидкої та ефективної розробки з покращеним користувацьким досвідом; Tailwind CSS для чистого та структурованого коду; CSS для стилізації інтерфейсу; MongoDB Atlas для надійного збереження даних про пости, коментарі та користувачів.

Проведено аналіз функціональних та нефункціональних вимог до додатку і побудовано діаграму варіантів використання для демонстрації основних функцій та можливостей системи. Розроблено діаграму класів для представлення структури системи та зв'язків між класами.

Створено базу даних у MongoDB для забезпечення збереження та колективного доступу до необхідної для функціонування додатку інформації. База даних складається з трьох колекцій, які відображають різні аспекти системи та зберігають відповідні дані.

Розроблено інтерфейс системи, який включає адміністраторське та користувацьке представлення. Адміністраторський інтерфейс дозволяє адміністраторам керувати системою, налаштовувати верифікацію користувачів, переглядати статистику та здійснювати інші адміністративні дії. Користувацький інтерфейс призначений для звичайних користувачів, які можуть переглядати публікації, реєструватися, оновлювати профіль, залишати коментарі та взаємодіяти з іншими функціями системи.

Проведено тестування додатку, включаючи функціональне, адаптивне, кроссбраузерне, безпекове та тестування на продуктивність. Виявлені помилки були виправлені та доведено працездатність додатку.

Практичне значення роботи полягає у розробці програмного комплексу, який може бути успішно використаний для публікації актуальних та перевірених постів, надаючи надійну та ефективну інформаційну платформу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Vice R. Mastering React / R. Vice, A. Horton., 2016. – 260 с. – (Packt Publishing)
2. Eric Elliott. Programming JavaScript Applications — O'Reilly Media, 254 p
3. Офіційна документація Express [Електронний ресурс] – Режим доступу до ресурсу: <https://expressjs.com/>
4. Офіційна документація Axios [Електронний ресурс] – Режим доступу до ресурсу: <https://axios-http.com/docs/intro>
5. Інформаційний додаток 'DOU' [Електронний ресурс] – Режим доступу до ресурсу: <https://dou.ua/forums/>
6. Документація MongoDB [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.mongodb.com/>
7. Документація Mongoose [Електронний ресурс] – Режим доступу до ресурсу: <https://mongoosejs.com/docs/>
8. Офіційна документація Tailwind CSS [Електронний ресурс] – Режим доступу до ресурсу : <https://tailwindcss.com/docs>
9. Офіційна документація React [Електронний ресурс] – Режим доступу до ресурсу: <https://reactjs.org/docs/>
10. Інформаційний додаток 'GOIT' [Електронний ресурс] – Режим доступу до ресурсу: <https://goit.global/ua/blog/>
11. Офіційна документація MongoDB Atlas [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.atlas.mongodb.com/>
12. Офіційна документація Node.js [Електронний ресурс] – Режим доступу до ресурсу: <https://nodejs.org/en/docs/>
13. Офіційна документація npm [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.npmjs.com/>
14. Інформаційний додаток 'Фокус' [Електронний ресурс] – Режим доступу до ресурсу: <https://focus.ua/>

ДОДАТКИ

					ПЗ.КР.Б – 121 – 23 – ПЗ	
						63

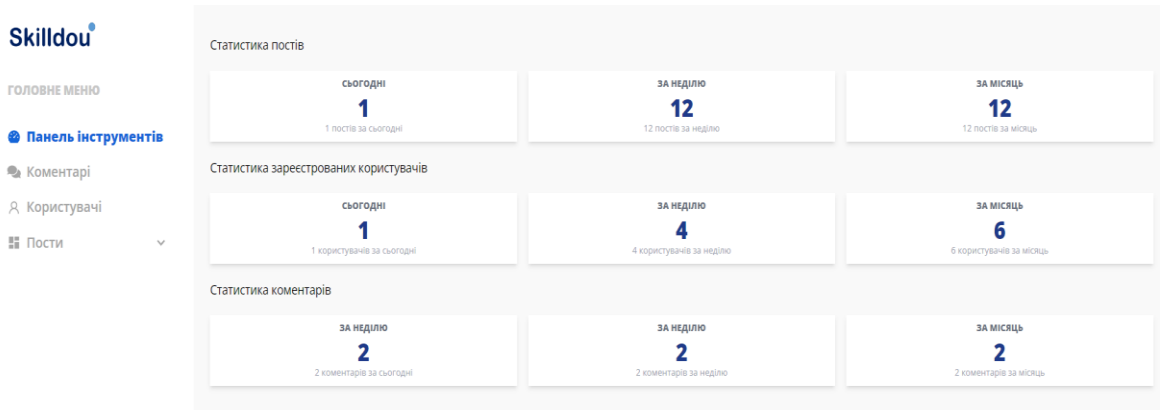


Рисунок А.1 – Сторінка зі статистикою.

commentControllers.js

```

import Comment from "../models/Comment";
import Post from "../models/Post";

const createComment = async (req, res, next) => {
  try {
    const {desc, slug, parent, replyOnUser} = req.body;

    const post = await Post.findOne({slug: slug});

    if(!post) {
      const error = new Error("Повідомлення не знайдено");
      return next(error);
    }

    const newComment = new Comment({
      user: req.user._id,
      desc,
      post: post._id,
      parent,
      replyOnUser,
    });

    const savedComment = await newComment.save();
    return res.json(savedComment);
  } catch (error) {
    next(error)
  }
};

const updateComment = async (req, res, next) => {
  try {
    const {desc, check} = req.body;

    const comment = await Comment.findById(req.params.commentId);

    if(!comment) {
      const error = new Error("Коментар не знайдено");
      return next(error);
    }

    comment.desc = desc || comment.desc;

    comment.check = check || comment.check;
  }
};

```

```

        const updatedComment = await comment.save();
        return res.json(updatedComment);

    } catch (error) {
        next(error)
    }
};

const deleteComment = async (req, res, next) => {
    try {

        const comment = await Comment.findByIdAndDelete(req.params.commentId);
        await Comment.deleteMany({parent: comment._id});

        if(!comment) {
            const error = new Error("Коментар не знайдено");
            return next(error);
        }

        return res.json({
            message: "Коментар успішно видалено",
        });

    } catch (error) {
        next(error)
    }
};

const getAllComments = async( req, res, next) => {
    try {
        const comments = await Comment.find().populate([
            {
                path: "user",
                select: ["avatar", "name", "verified"],
            },
        ]);
        res.json(comments);
    } catch (error) {
        next(error);
    }
};

const getComment = async (req, res, next) => {
    try {
        const comment = await Comment.findOne({commentId: req.query.id})
        if(!comment) {
            const error = new Error("Повідомлення не знайдено");

```

```

        return next(error);
    }

    return res.json(comment);
} catch (error) {
    next(error);
}
};

export {createComment, updateComment, deleteComment,getAllComments,getComment};

```

userControllers

```

import { uploadPicture } from "../middleware/uploadPicture";
import User from "../models/User";
import { fileRemover } from "../utils/fileRemover";

const registerUser = async (req, res, next) => {
    try {
        const {name, email, password} = req.body;
        let user = await User.findOne({email});
        if(user) {
            throw new Error("Користувач вже зареєструвався");
        }
        user = await User.create({
            name,
            email,
            password,
        });
        return res.status(201).json({
            _id: user._id,
            avatar: user.avatar,
            name: user.name,
            email: user.email,
            verified: user.verified,
            admin: user.admin,
            token: await user.generateJWT(),
        });
    }
    catch (error) {
        next(error);
    }
};

const loginUser = async (req, res, next) => {
    try {

```

```

    const {email, password} = req.body;
    let user = await User.findOne({email});
    if(!user){
        throw new Error("Електронна пошта не знайдена");
    }
    if (await user.comparePassword(password)) {
    return res.status(201).json({
        _id: user._id,
        avatar: user.avatar,
        name: user.name,
        email: user.email,
        verified: user.verified,
        admin: user.admin,
        token: await user.generateJWT(),
    });
    } else {
        throw new Error("Невірна адреса електронної пошти або пароль");
    }

    } catch (error) {
        next(error);
    }
};

const userProfile = async (req,res, next) => {
    try {
        let user = await User.findById(req.user._id);

        if(user) {
            return res.status(201).json({
                _id: user._id,
                avatar: user.avatar,
                name: user.name,
                email: user.email,
                verified: user.verified,
                admin: user.admin,
            });
        } else {
            let error = new Error("Користувач не знайдений");
            error.statusCode = 404;
            next(error);
        }
    } catch (error) {
        next(error);
    }
};

const updateProfile = async (req, res,next) => {
    try {
        let user = await User.findById(req.user._id);

        if(!user) {

```

```

        throw new Error("Користувач не знайдений");
    }
    user.name = req.body.name || user.name;
    user.email = req.body.email || user.email;

    if(req.body.password && req.body.password.length < 6) {
        throw new Error("Пароль має бути не менше 6 символів");
    } else if (req.body.password) {
        user.password = req.body.password;
    }
    const updatedUserProfile = await user.save();
    res.json({

        _id: updatedUserProfile._id,
        avatar: updatedUserProfile.avatar,
        name: updatedUserProfile.name,
        email: updatedUserProfile.email,
        verified: updatedUserProfile.verified,
        admin: updatedUserProfile.admin,
        token: await updatedUserProfile.generateJWT(),
    });

} catch (error) {
    next(error);
}
};

const updateProfilePicture = async (req,res,next) => {
    try {
        const upload = uploadPicture.single("profilePicture");
        upload(req,res,async function (err) {
            if(err){
                const error = new Error("Під час завантаження сталася невідома помилка" + err.message);
                next(error);
            } else {
                // якщо все пройшло добре
                if(req.file) {
                    let filename;
                    let updatedUser = await User.findById(req.user._id);
                    filename = updatedUser.avatar;
                    if(filename){
                        fileRemover(filename);
                    }
                    updatedUser.avatar = req.file.filename;
                    await updatedUser.save();
                    res.json({
                        _id: updatedUser._id,
                        avatar: updatedUser.avatar,

```

```

        name: updatedUser.name,
        email: updatedUser.email,
        verified: updatedUser.verified,
        admin: updatedUser.admin,
        token: await updatedUser.generateJWT(),

    });
} else {
    let filename;
    let updatedUser = await User.findById(req.user._id);
    filename = updatedUser.avatar;
    updatedUser.avatar = "";
    await updatedUser.save();
    fileRemover(filename);
    res.json({
        _id: updatedUser._id,
        avatar: updatedUser.avatar,
        name: updatedUser.name,
        email: updatedUser.email,
        verified: updatedUser.verified,
        admin: updatedUser.admin,
        token: await updatedUser.generateJWT(),

    });
}
}
});

} catch (error) {
    next(error);
}
};

const getAllUsers = async (req, res, next) => {
    try {
        const users = await User.find();
        res.json(users);
    } catch (error) {
        next(error);
    }
};

const deleteUser = async (req, res, next) => {
    try {
        const user = await User.findByIdAndDelete({_id: req.params._id});
        if(!user) {
            const error = new Error("Коментар не знайдено");
            return next(error);
        }
    }
};

```

```

        return res.json({
            message: "Коментар успішно видалено",
        });
    } catch (error) {
        next(error)
    }
};

const verifiedUser = async (req, res, next) => {
    try {
        const {verified} = req.body;
        console.log(verified)
        const user = await User.findById({_id: req.params._id});

        if(!user) {
            const error = new Error("Користувача не знайдено");
            return next(error);
        }
        user.verified = verified ;
        console.log(user.verified)
        const verifiedUser = await user.save();
        return res.json(verifiedUser);
    } catch (error) {
        next(error)
    }
};

export {
    registerUser,
    loginUser,
    userProfile,
    updateProfile,
    updateProfilePicture,
    getAllUsers,
    deleteUser,
    verifiedUser
};

```

postControllers

```

import { uploadPicture } from "../middleware/uploadPicture";
import Post from "../models/Post";
import Comment from "../models/Comment";
import { fileRemover } from "../utils/fileRemover";

import { v4 as uuidv4 } from "uuid";

const createPost = async (req, res, next) => {
    try {
        const upload = uploadPicture.single("photo");

```

					ІІЗ.КР.Б – 121 – 23 – ІІЗ	
						71

```

console.log(req.body)
const post = new Post({
  title: req.body.title,
  caption: req.body.caption,
  slug: uuidv4(),
  body: req.body.body,
  photo: "" ,
  user: req.user._id,
});

const handleUpdatePostData = async (data) => {
  console.log(post)
  const createdPost = await post.save();
  return res.json(createdPost);
};

upload(req, res, async function (err) {
  if (err) {
    const error = new Error(
      "Під час завантаження сталася невідома помилка" + err.message
    );
    next(error);
  } else {
    if (req.file) {
      let filename;

      filename = post.photo;
      if (filename) {
        fileRemover(filename);
      }
      post.photo = req.file.filename;
      handleUpdatePostData(req.body.document);
    } else {
      let filename;

      filename = post.photo;
      post.photo = "";
      console.log(post.photo);
      fileRemover(filename);
      handleUpdatePostData(req.body.document);
    }
  }
});
} catch (error) {
  console.log(error);
  next(error);
}
};

const updatePost = async (req, res, next) => {
  try {

```

```

const post = await Post.findOne({ _id: req.params.slug });
console.log(post);
if (!post) {
  const error = new Error("Повідомлення не знайдено");
  next(error);
  return;
}

const upload = uploadPicture.single("postPicture");
const handleUpdatePostData = async (data) => {
  console.log(post);
console.log(11)
  const { title, caption, slug, body, tags, categories } = JSON.parse(data);
  post.title = title || post.title;
  post.caption = caption || post.caption;
  post.slug = slug || post.slug;
  post.body = body || post.body;
  post.tags = tags || post.tags;
  post.categories = categories || post.categories;
  const newPost = { ...post };
  const updatedPost = await Post.findByIdAndUpdate(
    req.params.slug,
    newPost
  );
  return res.json(updatedPost);
};
upload(req, res, async function (err) {
  if (err) {
    const error = new Error(
      "Під час завантаження сталася невідома помилка" + err.message
    );
    next(error);
  } else {
    if (req.file) {
      let filename;

      filename = post.photo;
      if (filename) {
        fileRemover(filename);
      }
      post.photo = req.file.filename;
      console.log(post);
      handleUpdatePostData(req.body.document);
    } else {
      let filename;

      filename = post.photo;
      post.photo = "";
      fileRemover(filename);
      handleUpdatePostData(req.body.document);
    }
  }
});

```

```

    }
  }
});
} catch (error) {
  next(error);
}
};

const deletePost = async (req, res, next) => {
  try {
    console.log(req.params);
    const post = await Post.findOneAndDelete({ _id: req.params.slug });

    if (!post) {
      const error = new Error("Повідомлення не знайдено");
      return next(error);
    }
    await Comment.deleteMany({ post: post._id });

    return res.json({
      message: "Post is successfully deleted",
    });
  } catch (error) {
    next(error);
  }
};

const getPost = async (req, res, next) => {
  try {
    const post = await Post.findOne({ slug: req.params.slug }).populate([
      {
        path: "user",
        select: ["avatar", "name"],
      },
      {
        path: "comments",
        match: {
          check: true,
          parent: null,
        },
        populate: [
          {
            path: "user",
            select: ["avatar", "name"],
          },
          {
            path: "replies",
            match: {
              check: true,
            },
          },
        ],
      },
    ]);
  }
};

```

```

        populate: [
          {
            path: "user",
            select: ["avatar", "name"],
          },
        ],
      },
    ],
  },
]);

if (!post) {
  const error = new Error("Повідомлення не знайдено");
  return next(error);
}

return res.json(post);
} catch (error) {
  next(error);
}
};

const getAllPosts = async (req, res, next) => {
  try {
    const page = req.query.page || 1;
    const total = req.query.total || 12;
    const offset = page * total - total;
    const posts = await Post.find({}).skip(offset).limit(total).populate([
      {
        path: "user",
        select: ["avatar", "name", "verified"],
      },
    ]);
    const count = await Post.countDocuments();

    res.json({posts:posts.reverse(),count:count});
  } catch (error) {
    next(error);
  }
};

const getOnePosts = async (req, res, next) => {
  try {
    console.log(1);
    const posts = await Post.findOne({
      title: req.query.title,
    });
    res.json(posts);
  } catch (error) {
    next(error);
  }
};

```

```
export {  
  createPost,  
  updatePost,  
  deletePost,  
  getPost,  
  getAllPosts,  
  getOnePosts,  
};
```