

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:

«Веборієнтований інформаційний додаток новин»

Виконав студент групи ВТ-22-2
МИКИТЮК Сергій Юрійович

Керівник роботи:
МОРОЗОВ Андрій Васильович

Рецензент:
ВЛАСЕНКО Олег Васильович

Житомир – 2026

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

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії програмного забезпечення

Тетяна ВАКАЛЮК

«28» лютого 2026 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **Микитюк Сергій Юрійович.**

Керівник роботи: **Морозов Андрій Васильович.**

Тема роботи: **«Веборієнтований інформаційний додаток новин»,**
затверджена Наказом закладу вищої освіти від **«28» лютого 2026 р., №92/с.**

Вихідні дані для роботи: Об'єктом дослідження є процес розробки та впровадження веборієнтованих інформаційних систем. Предметом — архітектура, реалізація та тестування веб-додатку на стеку MERN. Метою роботи є створення, перевірка працездатності та оформлення веборієнтованого інформаційного додатку з адаптивним інтерфейсом на основі сучасних вебтехнологій.

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

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Морозов А.В.	14.02.2026р.	14.02.2026р.
2	Морозов А.В.	17.04.2026р.	18.04.2026р.
3	Морозов А.В.	01.05.2026р.	01.05.2026р.

РЕФЕРАТ

Кваліфікаційна робота бакалавра складається з веборієнтованого інформаційного додатку та пояснювальної записки. Пояснювальна записка до випускної роботи містить 93 сторінок, 36 ілюстрацій, 20 таблиць, 32 джерела та 1 додатку.

Метою роботи є розробка веборієнтованого інформаційного додатку з адаптивним інтерфейсом користувача на базі React, Tailwind CSS та Node.js/Express.js, а також створення системи адміністрування контентом і користувачами з використанням хмарної бази даних MongoDB Atlas.

У процесі виконання роботи сформульовано вимоги користувача та обґрунтовано концептуальну модель системи. Проведено аналіз аналогів програмних продуктів і обрано засоби реалізації. Розроблено структуру бази даних, обрано клієнт–серверну архітектуру та спроектовано структуру веб-додатку. Реалізовано функціонал публікації та перегляду статей, категорій і коментарів, автентифікації користувачів і адміністрування. Описано програмну реалізацію системи та наведено результати її тестування.

КЛЮЧОВІ СЛОВА: MERN, EXPRESS.JS, REACT, MONGODB ATLAS, АДМІНІСТРУВАННЯ, БАЗА ДАНИХ, ВЕБОРІЄНТОВАНИЙ ДОДАТОК, ІНФОРМАЦІЙНИЙ КОНТЕНТ, КОРИСТУВАЦЬКИЙ ІНТЕРФЕЙС.

					ІПЗ.КР.Б – 121 – 26 – ПЗ			
Зм.	Арк.	№ докум.	Підпис	Дата	Веборієнтований інформаційний додаток новин. Пояснювальна записка	Літ.	Арк.	Аркушів
Розроб.		С.Ю. Микитюк						
Керівник		А.В. Морозов					3	96
						Житомирська політехніка, група ВТ-22-2		
Рецензент		О.В.Власенко						
Зав. каф.		Т.А. Вакалюк						

ABSTRACT

The bachelor's thesis consists of a web-oriented information application and an explanatory report. The explanatory report for the thesis consists of 93 pages, 36 illustrations, 20 tables, 32 references, and 1 appendix.

The aim of the thesis is to develop a web-oriented information application (news platform) with an adaptive user interface based on React, Tailwind CSS, and Node.js/Express.js, as well as to implement an administration subsystem for content and user management using the cloud database MongoDB Atlas.

The thesis defines user requirements and justifies the development of a conceptual model. An analysis of software product analogues was conducted, and implementation tools were selected. The database structure was designed, a client–server architecture was chosen, and the system structure was developed. Functionality for publishing and viewing articles, categories, and comments, user authentication, and administration was implemented. The software implementation of the system was described, and the results of system testing were presented.

KEYWORDS: MERN, EXPRESS.JS, REACT, MONGODB ATLAS, ADMINISTRATION, DATABASE, WEBORIENTED APPLICATION, INFORMATION CONTENT, USER INTERFACE.

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 апарат- ного забезпечення.....	21
Висновки до розділу 1	23
РОЗДІЛ 2. ПРОЕКТУВАННЯ СИСТЕМИ ІНФОРМАЦІЙНОГО ДОДАТКА...25	
2.1. Визначення варіантів використання та об'єктно-орієнтованої структури системи.....	25
2.2. Розробка бази даних системи.....	33
2.3. Проектування та реалізація алгоритмів роботи системи.....	39
2.4. Реалізація функціоналу інформаційного додатка.....	42
Висновки до розділу 2	50
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З СИСТЕМОЮ ДОДАТКА.52	
3.1 Порядок встановлення та налаштування системи.....	52
3.2 Структура інтерфейсу. Інтерфейс та порядок роботи з веборіє- нтованим інформаційним додатком.....	55
3.3 Тестування роботи програмного комплексу	64
Висновки до розділу 3.....	68
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	71
ДОДАТКИ.....	74

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

JWT – JSON Web Token.

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.

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

REST – Representational State Transfer.

ВСТУП

Актуальність теми пов'язана з тим, що більшість користувачів сьогодні отримують інформацію саме через браузер — на комп'ютері чи телефоні. Новини, статті, навчальні матеріали, описи товарів, повідомлення від сервісів — усе це зручніше подавати у вигляді веборієнтованого інформаційного додатку, ніж окремої програми для кожної задачі.

Такі системи давно використовують у навчанні, торгівлі, медіа, державних і комерційних сервісах. Вони дають можливість швидко оновлювати контент, додавати нові розділи, керувати доступом користувачів і не вимагають від відвідувача встановлення додаткового програмного забезпечення.

Окремої уваги заслуговує питання достовірності та своєчасності інформації. У період пандемії, воєнного стану та інформаційного перевантаження люди частіше шукають джерела, де матеріали систематизовані, оновлюються і їх можна перевірити. Саме тому актуальні веборієнтовані інформаційні додатки з перевіреним контентом залишаються затребуваними — не лише великими медіа, а й тематичними блогами, освітніми проєктами чи корпоративними сайтами.

Практична цінність веборієнтованих додатків полягає також у зручності для тих, хто ними керує. Адміністратор може додавати і редагувати записи, працювати з категоріями, модерувати коментарі, не звертаючись щоразу до розробника. Для цього потрібна продумана структура даних, зрозумілий інтерфейс і надійне зберігання інформації — зокрема у хмарних базах даних, які дозволяють працювати з системою з різних місць.

Технології в цій сфері постійно змінюються: з'являються нові підходи до побудови інтерфейсу, швидшої роботи сторінок, безпеки облікових записів. Тому вебдодаток не створюють «раз і назавжди» — його можна поступово розширювати: додати пошук, рейтинги, особистий кабінет, адмін-панель. Це робить тему розробки таких систем актуальною і на сьогодні, і на перспективу.

Метою випускної роботи є створення веборієнтованого інформаційного додатку, орієнтованого на подачу актуального контенту та зручну роботу кори-

стувача з інтерфейсом. Розроблена система може використовуватись для публікації статей, новин, матеріалів блогу або іншої тематичної інформації, яка цікавить певну аудиторію.

Досягнення цієї мети передбачає послідовне виконання кількох взаємопов'язаних завдань. Передусім потрібно підібрати технології для розробки — порівняти можливі варіанти й обрати засоби для побудови клієнтської та серверної частин, роботи з базою даних і реалізації інтерфейсу. Поряд із цим слід з'ясувати потреби користувачів, тобто визначити, що саме має вміти система для звичайного відвідувача та для адміністратора: перегляд матеріалів, пошук, реєстрацію, публікацію контенту тощо. На основі сформульованих вимог формується концептуальна модель додатку з описом основних сутностей, зв'язків між ними та логіки роботи системи, після чого проєктується інтерфейс — зручна й адаптивна структура сторінок із продуманою навігацією та розміщенням контенту.

Центральним завданням залишається реалізація програмної частини: серверної логіки та API, роботи з базою даних і клієнтського застосунку з основним функціоналом — статтями, категоріями, коментарями, обліковими записами та адмініструванням. Окремо забезпечується базовий захист даних, що включає автентифікацію користувачів, обмеження доступу до адміністративних функцій і безпечне зберігання конфіденційної інформації. Завершальними кроками стають перевірка працездатності системи через тестування основних сценаріїв, оформлення результатів у вигляді пояснювальної записки з описом архітектури, структури бази даних і порядку використання додатку, а також оцінювання того, наскільки створений продукт відповідає поставленим вимогам і чи коректно працює його основний функціонал.

Узагальнену логічну структуру виконаної роботи — від обґрунтування актуальності теми до оцінювання готового продукту — наведено на рисунку 1.

Об'єкт дослідження процеси та інформаційні технології, за допомогою яких створюють і впроваджують веборієнтовані інформаційні додатки.

Предмет дослідження моделі, методи, алгоритми та програмні засоби побудови такого додатку, коли головна увага приділяється актуальному інформаційному контенту та зручному інтерфейсу для користувача.

У процесі виконання роботи застосовано підходи вебпрограмування для реалізації серверної частини, обміну даними між клієнтом і сервером, роботи з базою даних і основних функцій системи. Окремо використано прийоми проектування інтерфейсу, щоб сторінки були зрозумілими, адаптивними та зручними для перегляду й публікації матеріалів.

Розроблений додаток можна використовувати як робочу платформу для подачі статей, новин або іншого тематичного контенту. За потреби його структуру можна доповнити новими розділами, змінити оформлення або додати функції — наприклад, розширене адміністрування, пошук, рейтинги чи особистий кабінет користувача.

РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ ІНФОРМАЦІЙНОГО ДОДАТКА

1.1. Постановка задачі

Задача випускної кваліфікаційної роботи полягає в розробці сучасного ве-
борієнтованого інформаційного додатку новин, який забезпечуватиме ефекти-
вну публікацію, систематизацію та споживання актуального контенту. У суча-
сному інформаційному суспільстві, де більшість людей отримує новини, аналі-
тичні матеріали та тематичні статті саме через веб-інтерфейси, створення зру-
чної, швидкої, надійної та функціональної платформи є надзвичайно актуальним
завданням. Розроблюваний додаток має стати не просто набором статичних
сторінок, а повноцінною динамічною екосистемою, де відвідувачі можуть легко
знаходити цікавий контент, взаємодіяти з ним через коментарі, оцінки та збере-
ження матеріалів, а адміністратори — ефективно керувати всім процесом
наповнення, модерації та розвитку ресурсу.

Актуальність теми зумовлена стрімким зростанням обсягів інформації в
цифровому просторі, необхідністю оперативного оновлення контенту та підви-
щенням вимог користувачів до зручності інтерфейсу, швидкості завантаження
та персоналізації. Традиційні новинні сайти часто страждають від перевантаже-
ності рекламними елементами, складної навігації або обмежених можливостей
для взаємодії аудиторії. Тому розробка власного рішення, яке поєднує в собі
переваги традиційних медіа-платформ з сучасними технологіями (адаптивний
дизайн, швидке оновлення контенту, зручне адміністрування), дозволить створи-
ти продукт, що відповідає реальним потребам сучасних користувачів.

Для досягнення цієї мети необхідно вирішити низку взаємопов'язаних зав-
дань, які охоплюють весь життєвий цикл розробки програмного забезпечення.
Ці завдання можна систематизувати наступним чином (див. таблицю 1.1).

Основні етапи розробки інформаційного додатку

Етап	Основні завдання
Проектування системи	Визначення вимог користувачів та адміністратора, опис складу системи (публікації, категорії, облікові записи, коментарі, адміністрування), вибір архітектури взаємодії клієнтської та серверної частин
Розробка інтерфейсу	Проектування зрозумілої навігації, адаптивної структури сторінок, забезпечення зручності читання матеріалів та роботи з основними функціями
Розробка бази даних	Проектування структури зберігання даних, реалізація операцій CRUD, підключення до хмарної бази даних MongoDB Atlas
Реалізація програмної частини	Створення серверної частини на Node.js та Express.js з REST API, розробка клієнтської частини на React, впровадження реєстрації, авторизації, публікації матеріалів та модерації
Тестування та доопрацювання	Перевірка основних сценаріїв роботи, виявлення та усунення помилок, забезпечення відповідності системи сформульованим вимогам

У результаті має бути створено повноцінний робочий веб-додаток, у якому звичайний користувач зможе швидко знаходити потрібні публікації завдяки зручному пошуку, фільтрації за категоріями та сортуванню матеріалів. Адміністратор отримає можливість оперативно оновлювати контент, модерувати коментарі та керувати користувачами без залучення додаткових технічних спеціалістів.

Передбачено реалізацію базових механізмів захисту облікових даних, зокрема автентифікацію користувачів та обмеження доступу до адміністрати-

вних функцій. Особлива увага приділяється зручності використання на різних пристроях завдяки адаптивному дизайну.

Після завершення основної розробки система підлягає комплексній перевірці працездатності основних компонентів. Наступним етапом роботи стане детальний аналіз існуючих аналогів програмних продуктів подібного призначення. Це дозволить виявити їхні сильні та слабкі сторони, врахувати кращі практики та сформулювати конкретні вимоги до функціональності власної розробки.

1.2. Аналіз аналогів програмного продукту

Аналіз аналогічних програмних продуктів є необхідним етапом підготовки до проектування веборієнтованого інформаційного додатку. Він дозволяє систематизувати наявний досвід створення подібних систем, визначити доцільні архітектурні та інтерфейсні рішення, а також сформулювати вимоги до функціональності розроблюваного програмного засобу [1].

Для проведення аналізу були відібрані три популярні українські інформаційні ресурси, які за характером функціонування та цільовою аудиторією найближче відповідають тематиці роботи: AIN.UA, ITC.ua та Rubryka. Кожен з них розглядався комплексно — з точки зору функціональних можливостей, якості користувацького інтерфейсу, організації контенту, зручності навігації та рівня взаємодії з відвідувачами. Такий підхід дав можливість не тільки оцінити поточний стан ринку, але й визначити ті елементи, які варто запозичити, а також ті аспекти, яких слід уникати в власній реалізації.

"AIN.UA" – українське інтернет-видання, орієнтоване на публікацію матеріалів у сфері технологій, бізнесу та інновацій, наведено на рисунку 1.1. Платформа реалізована як інформаційний вебресурс із системою рубрик, структурованим поданням статей і механізмами взаємодії аудиторії з контентом. [13].

Окрім загальної контентної структури, важливим аспектом аналізу стало дослідження поведінкових факторів і засобів персоналізації, які пропонують роз-

глянуті ресурси. Так, AIN.UA активно використовує тематичні підбірки та рекомендаційні блоки «схожі матеріали», що сприяє збільшенню глибини перегляду й утриманню користувача на сайті. ІТС.ua робить акцент на багаторівневій системі коментування та інтеграції з соціальними мережами, створюючи простір для дискусій і формування спільноти навколо технічних оглядів. Rubryka, своєю чергою, вирізняється акцентом на візуальні історії та довгі читання із застосуванням інтерактивних елементів (таймлайни, галереї, вставки з даними), що підвищує емоційне залучення. Водночас жоден із цих ресурсів не пропонує гнучкого налаштування стрічки новин за інтересами користувача в реальному часі, а також не використовує адаптивних алгоритмів сортування контенту залежно від історії переглядів, що відкриває перспективу для впровадження таких механізмів у власному проєкті. Отже, проведений порівняльний аналіз дозволив не лише виявити сильні сторони кожного зразка, але й окреслити потенційні напрями вдосконалення, які можуть стати конкурентними перевагами майбутнього ве-борієнтованого додатку.

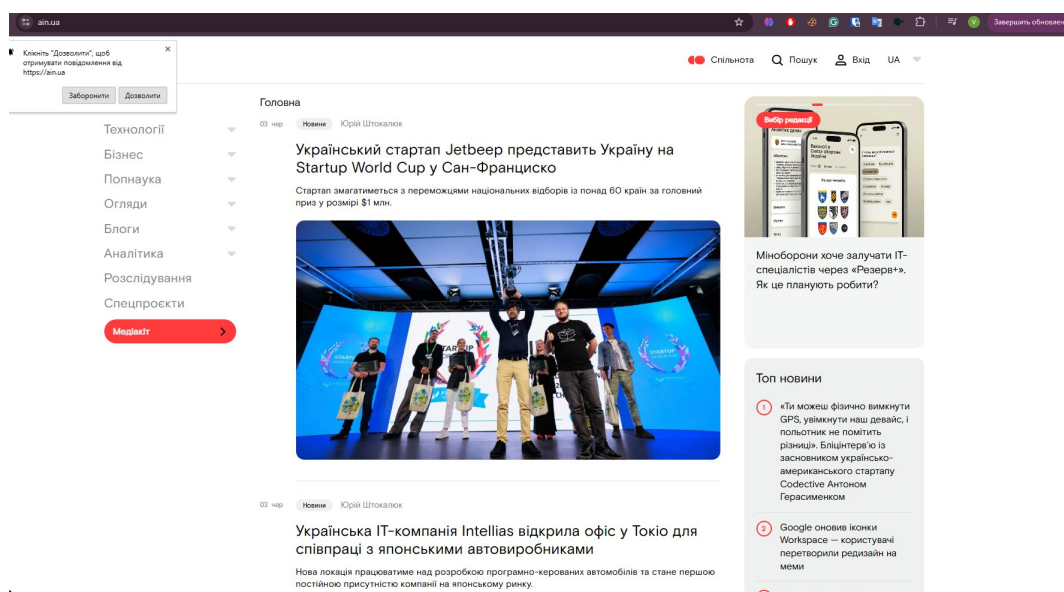


Рисунок 1.1 – Головна сторінка "AIN.UA"

Серед сильних сторін ресурсу варто виділити логічне тематичне групування контенту, зручну організацію головної сторінки з акцентом на найактуальніші та найважливіші публікації, якісну адаптивність інтерфейсу на різних при-

строях, а також регулярне оновлення матеріалів, що підтримує високу інформаційну актуальність. Крім того, видання часто публікує ексклюзивні інтерв'ю та глибокі аналітичні матеріали, що підвищує його цінність для професійної аудиторії. Водночас ресурс має ряд суттєвих недоліків. Зокрема, значна кількість рекламних блоків і банерів суттєво ускладнює сприйняття інформації та погіршує загальне враження від читання. Можливості самостійної публікації матеріалів для зареєстрованих користувачів сильно обмежені, структура сайту для нових відвідувачів може здаватися дещо складною, а модель наповнення залишається переважно редакційною, що не дає можливості широкій аудиторії брати активну участь у формуванні контенту.

ІТС.ua — ще один популярний український портал, присвячений новинам, оглядам та аналітиці в сфері інформаційних технологій наведено на рисунку 1.2. Ресурс містить категорії публікацій, сторінки окремих матеріалів і механізм коментування. [4].

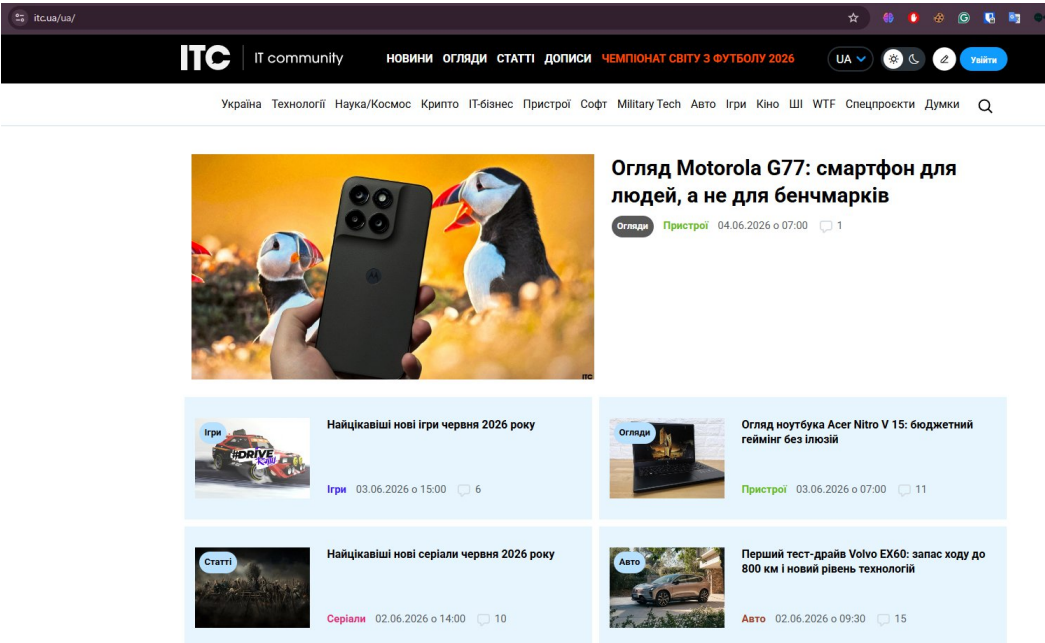


Рисунок 1.2 – Головна сторінка "ІТС.ua".

Сторінки окремих публікацій оформлені з акцентом на зручність читання — добре продумані шрифти, оптимальні відступи та логічний розподіл тексту. Однак і тут спостерігаються характерні проблеми. Переважає класичний редакційний підхід до створення контенту, через що ресурс мало залучає зовні-

шніх авторів. Окремі сторінки виглядають інформаційно перевантаженими через велику кількість додаткових новинних блоків, посилань та рекомендацій, що може розсіювати увагу читача. Можливості пошуку та фільтрації матеріалів залишаються досить обмеженими, а платформа практично не передбачає зручних інструментів для управління контентом з боку зовнішніх авторів. Усе це суттєво звужує потенціал ресурсу як відкритої платформи для обміну думками та матеріалами.

"Rubryka" – українське медіа, що спеціалізується на публікації матеріалів соціально значущої та аналітичної спрямованості наведено на рисунку 1.3. Ве-бресурс характеризується сучасним чистим дизайном, високою якістю типографічного оформлення текстів і акцентом на читабельність навіть досить об'ємних матеріалів. [9].

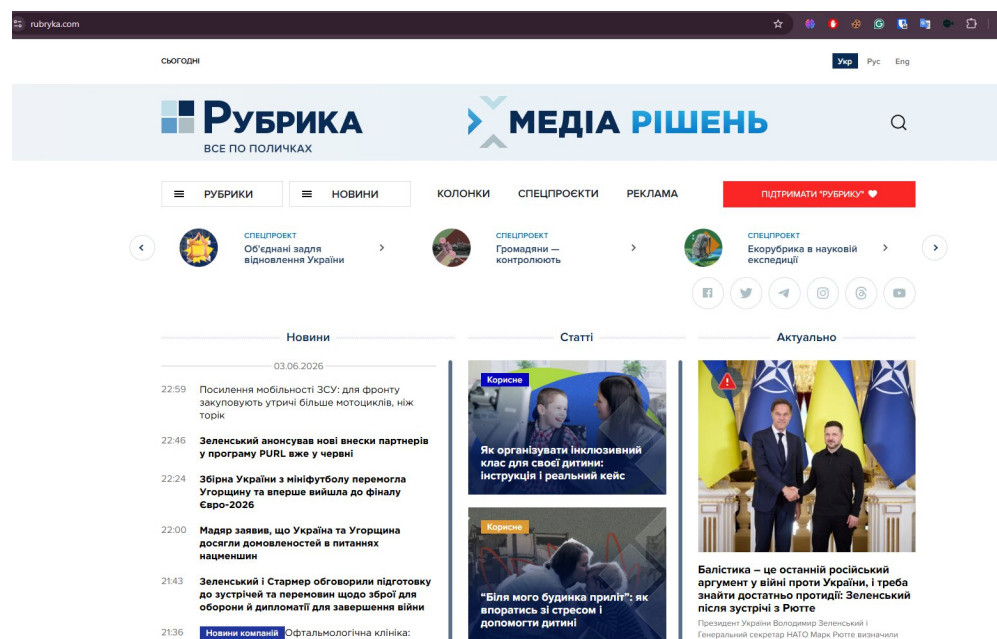


Рисунок 1.3 – Головна сторінка "Rubryka"

Тематичне групування публікацій реалізоване логічно, а адаптивність інтерфейсу на мобільних пристроях працює стабільно. Видання часто піднімає важливі суспільні теми та пропонує глибокі репортажі. Проте вузька тематична спрямованість контенту, обмежений набір інструментів для взаємодії користувачів (відсутність оцінювання, слабка система коментування), відсутність механізмів самостійної публікації матеріалів звичайними користувачами та орієнта-

ція на класичну медіа-модель значно зменшують універсальність платформи та її потенціал як повноцінного інформаційного додатку з широкою участю аудиторії.

Узагальнені результати порівняння розглянутих аналогів за основними критеріями наведено у таблиці 1.1.

Таблиця 1.2

Порівняння аналогів веборієнтованих інформаційних додатків

Критерій порівняння	AIN.UA	ITC.ua	Rubryka	Власна розробка
Тематична категоризація	Так	Так	Так	Так
Адаптивний інтерфейс	Так	Так	Так	Так
Коментування публікацій	Обмежено	Так	Ні	Так
Самостійна публікація користувачами	Ні	Ні	Ні	Так
Пошук і фільтрація матеріалів	Частково	Обмежено	Частково	Так
Адміністрування контенту	Редакційне	Редакційне	Редакційне	Адмін-панель
Реєстрація та профіль користувача	Ні	Ні	Ні	Так
Оцінювання публікацій	Ні	Ні	Ні	Так

Аналіз свідчить, що сучасні українські веборієнтовані інформаційні додатки характеризуються високим рівнем організації контенту, розвиненою системою категоризації та орієнтацією на зручність сприйняття інформації користувачем. Водночас більшість розглянутих аналогів реалізовані за редакційною моделлю і не надають користувачам повноцінних можливостей для самостійної публікації та адміністрування матеріалів. Можна зробити наступні висновки для розробки сайту наведено на рисунку 1.4.

На основі аналізу переваг і недоліків веборієнтованих інформаційних додатків переходимо до розробки власного програмного продукту. Потрібно взяти до уваги сильні сторони аналогів — зручні категорії, адаптивний дизайн і структуровану подачу матеріалів — і водночас усунути їхні слабкі місця.

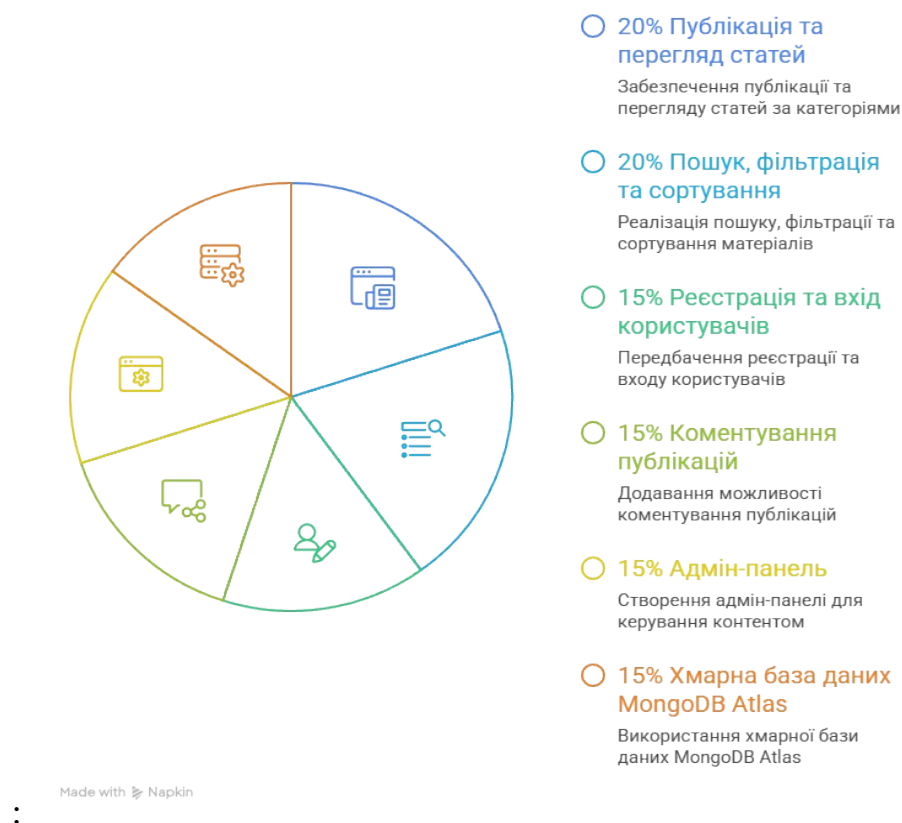


Рисунок 1.4 – Розподіл функціональності веб-сайту

1.3. Вибір архітектури інформаційного додатка

Для розробки веборієнтованого інформаційного додатку використовується фреймворк Express.js з Router для організації API-маршрутів. Порівнявши різні підходи до побудови серверної частини, визначено, що Express.js надає достатню гнучкість для реалізації обміну даними між клієнтом і сервером у контексті інформаційного додатку з публікаціями, категоріями та коментарями.

Express.js - це серверний фреймворк для Node.js, призначений для створення вебдодатків і REST API. Він дозволяє обробляти HTTP-запити, організовувати маршрути та взаємодіяти з базою даних. У розроблюваному проєкті Express.js використовується як backend-частина системи, тоді як React.js від-

повідас за клієнтський інтерфейс. Обмін між ними здійснюється через API-шляхи, які приймають і повертають дані у форматі JSON.

На сервері маршрутизація організована за допомогою Router. Для кожного шляху створюється функція-обробник, яка виконує необхідну бізнес-логіку, звертається до бази даних через Mongoose і повертає результат клієнту. Динамічні параметри в URL дозволяють працювати з конкретними об'єктами (наприклад, певною статтею чи користувачем) без зайвого дублювання коду.

Для розробки клієнтської частини використовується React.js, який дозволяє створювати компоненти для відображення користувацького інтерфейсу: головної сторінки, списку статей, сторінки публікації, форми коментарів та адміністративної панелі. Клієнтська частина працює як односторінковий веб-додаток, тобто інтерфейс оновлюється без повного перезавантаження сторінки [8].

Express.js надає можливість визначати API-шляхи за допомогою вбудованого Router. Ці шляхи визначають, які дані можна отримати або змінити на сервері. У розроблюваному додатку реалізовано маршрути для користувачів, публікацій, категорій, коментарів та адміністративних функцій. Розробники створюють API-шляхи декларативним способом, вказуючи шаблони URL та функції-обробники для кожного маршруту [2].

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 MongoDB через Mongoose та повернення результату у вигляді відповіді клієнтській частині. Наприклад, при запиті на отримання списку публікацій контролер звертається до бази даних, формує результат і повертає його у форматі JSON[2].

Маршрутизація. Router Express визначає, який обробник має спрацювати залежно від адреси запиту та HTTP-методу. Для роботи з конкретними об'єктами використовуються динамічні параметри в URL, наприклад ідентифікатор

статті чи користувача. Це дає змогу будувати зручну і зрозумілу структуру API без дублювання однотипного коду [3].

Окремі маршрути потребують підтвердження особи користувача, тому в системі застосовано JWT-автентифікацію. Після входу користувач отримує токен, який передається разом із наступними запитами до закритих endpoint-ів. Додатково middleware перевіряє права доступу, зокрема для адміністративних дій, що запобігає несанкціонованій зміні або видаленню контенту. [3].

Express підтримує middleware — проміжні функції, які виконуються до основного обробника. У проєкті вони використовуються для парсингу JSON, налаштування CORS, автентифікації та централізованої обробки помилок. Для завантаження зображень до публікацій застосовано Cloudinary, що дозволяє зберігати медіафайли окремо від текстових даних у MongoDB Atlas.

Узагальнену структуру розробленої системи та взаємодію її складових наведено на рисунку 1.5.

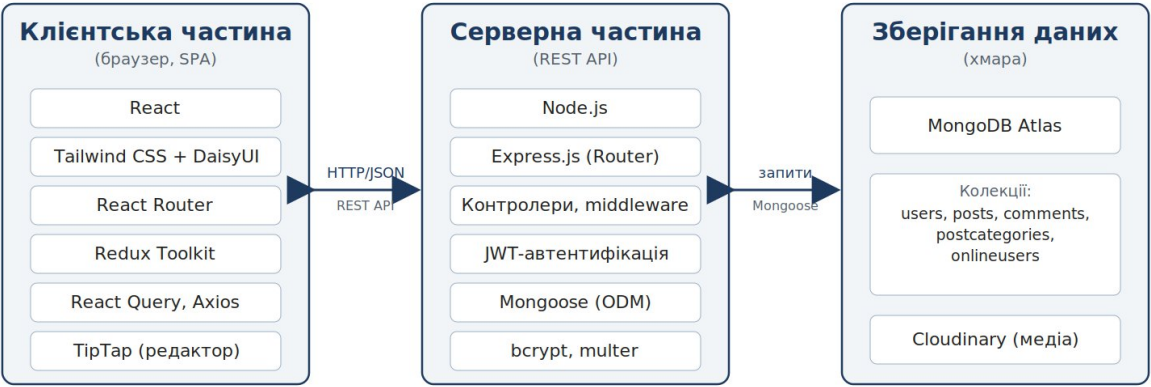


Рисунок 1.5 – Узагальнена архітектура веборієнтованого інформаційного додатку

Важливим аспектом реалізації серверної логіки є валідація вхідних даних. Перед тим, як дані потрапляють до контролера або моделі Mongoose, вони проходять перевірку за допомогою спеціалізованих бібліотек, наприклад Joi або express-validator. Це дозволяє відхиляти некоректні запити на ранньому етапі, запобігаючи помилкам бази даних та підвищуючи загальну безпеку додатку.

Крім того, така валідація є критичною для захисту від атак типу SQL-ін'єкцій (хоча MongoDB і використовує власну мову запитів, небезпечні дані все одно можуть спричинити логічні помилки) та для забезпечення цілісності даних, особливо при роботі з динамічними параметрами URL, які ідентифікують конкретні об'єкти.

Окрім базової маршрутизації та автентифікації, у системі приділено увагу оптимізації запитів до бази даних. Для зменшення навантаження на сервер та пришвидшення відповідей API використовуються такі механізми Mongoose, як `populate` для підтягування пов'язаних документів, `select` для обмеження вибірки лише необхідними полями, а також пагінація за допомогою методів `limit` та `skip` при роботі з великими списками статей чи коментарів. Для особливо важких запитів передбачено кешування результатів на рівні сервера або за допомогою зовнішніх інструментів, що дозволяє React-клієнту отримувати дані майже миттєво, особливо на головній сторінці та при відображенні списку публікацій, забезпечуючи плавну роботу односторінкового додатку.

Архітектура розроблюваного додатку базується на взаємодії React-клієнта з Express-сервером через REST API та збереженні даних у хмарній базі MongoDB. Таке поєднання технологій забезпечує зрозумілу структуру проєкту, зручність супроводу та можливість подальшого розвитку інформаційного додатку без кардинальної перебудови системи.

1.4. Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення

Веборієнтований інформаційний додаток побудовано за класичною клієнт-серверною архітектурою з чітким розділенням фронтенду та бекенду. Для клієнтської частини обрано React.js у поєднанні з Tailwind CSS і React Router, для серверної — Node.js та Express.js, а для зберігання даних — хмарну базу MongoDB Atlas через бібліотеку Mongoose.

Додатково використовуються Axios для виконання HTTP-запитів, JWT для автентифікації, Cloudinary для завантаження зображень та TipTap для текстового редактора в адмін-панелі. Для керування станом клієнтської частини застосовано Redux Toolkit [14], а для роботи із серверними даними, їх кешування та синхронізації запитів — бібліотеку React Query (TanStack Query) [15]. Для підготовки макетів використано Figma, а Visual Studio Code — як основне середовище розробки.

Під час вибору технологій враховувалися можливість швидкої розробки, зручність підтримки коду, сумісність компонентів між собою, достатня продуктивність для роботи з текстовим контентом, надійність зберігання даних і доступність інструментів для початкового рівня розробки. Обраний набір засобів дозволяє створити працездатний додаток із зрозумілою структурою проєкту та можливістю подальшого розширення функціоналу.

React.js став основою клієнтської частини, оскільки дозволяє розбивати інтерфейс на невеликі, незалежні компоненти — картки статей, форми входу, блок коментарів, адміністративні сторінки тощо. Такий підхід значно спрощує внесення змін: оновити один компонент можна без торкання решти додатка. React добре працює разом з React Router для навігації та іншими бібліотеками, використаними в проєкті [8].

Серверна частина побудована на Node.js та Express.js. Node.js забезпечує виконання JavaScript-коду на сервері, а Express дозволяє швидко організувати REST API. Саме через цей фреймворк сервер приймає запити від клієнта, обробляє їх у контролерах і повертає відповіді у форматі JSON. Express зручний завдяки простій системі маршрутів і широкій підтримці middleware [11],[2].

Для стилізації всього інтерфейсу обрано Tailwind CSS. Це utility-first підхід, коли стилі задаються прямо в розмітці через готові класи. Завдяки цьому процес оформлення сторінок йде набагато швидше, а весь додаток виглядає єдино. Додатково використовуються DaisyUI та typography-плагін, що особливо допомагає з оформленням текстового контенту статей [7].

Дані зберігаються в MongoDB Atlas — хмарному сервісі, який працює з документами. Такий формат дуже зручний для нашого додатку, оскільки публікації, користувачі, категорії та коментарі мають різну структуру і часто змінюються. Mongoose виступає як проміжний шар між Express і базою, дозволяючи описувати схеми даних і виконувати всі необхідні операції [10].

Серед додаткових інструментів варто згадати Axios для запитів з фронтенду, JWT для автентифікації, Cloudinary для роботи з зображеннями та TipTap — зручний редактор для створення статей в адмін-панелі. Стан клієнтської частини керуємо через Redux Toolkit, а для роботи з серверними даними, кешування та синхронізації використовуємо React Query. Макети інтерфейсу готувалися в Figma, а основним редактором коду став Visual Studio Code.

Для наочності основні технології та їх призначення зображено на інфографіці 1.6.

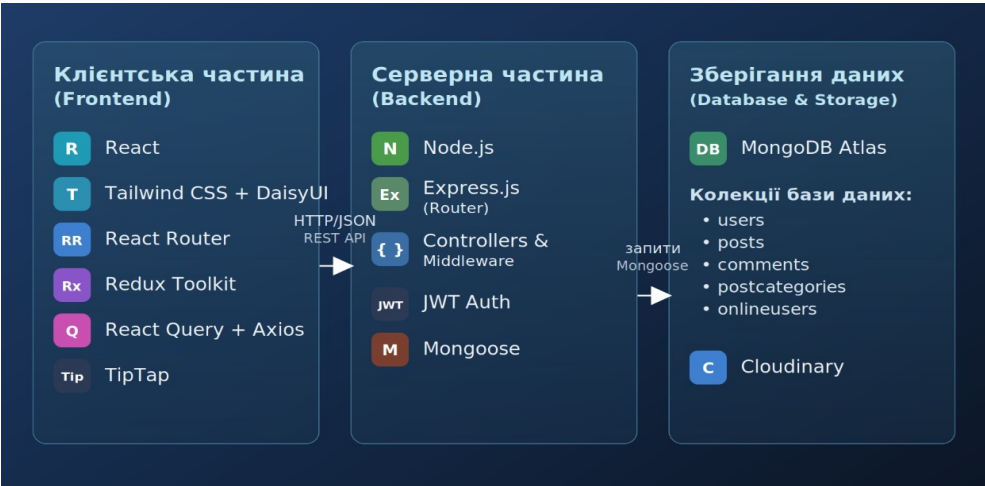


Рисунок 1.6 — Технологічний стек розробленого додатку

На схемі чітко видно три основні шари: клієнтську частину серверну частину та шар зберігання даних. Стрілки показують основні потоки даних і взаємодію між компонентами.

Висновки до розділу 1

У першому розділі було проаналізовано основні задачі створення веборієнтованого інформаційного додатку, орієнтованого на публікацію та споживання

актуального контенту. З'ясовано, що система повинна надійно зберігати дані про публікації, категорії, користувачів і коментарі, а також забезпечувати їх зручний перегляд, пошук, фільтрацію та адміністрування. Окрім цього, чітко визначено вимоги до розмежування прав доступу між звичайними користувачами та адміністраторами.

Огляд популярних українських ресурсів — AIN.UA, ITC.ua та Rubryka — показав, що для подібних платформ особливо важливими є зручна навігація, чітка категоризація матеріалів, адаптивний інтерфейс і можливості взаємодії аудиторії з контентом. Водночас більшість розглянутих аналогів працює за традиційною редакційною моделлю і майже не дає користувачам інструментів для самостійної публікації матеріалів чи активної участі в розвитку сайту. Саме тому в власній розробці було вирішено реалізувати реєстрацію, коментування, повноцінну адмін-панель, а також зручний пошук і фільтрацію контенту.

У розділі також обґрунтовано вибір архітектури та технологічного стеку. Для клієнтської частини обрано React.js, стилізацію виконано за допомогою Tailwind CSS, а серверну частину побудовано на Node.js та Express.js з REST API. Дані зберігаються в хмарній базі MongoDB Atlas. Додатково застосовано Mongoose для роботи з базою, JWT для автентифікації, Axios для запитів, Cloudinary для зображень, TipTap для редактора статей, Redux Toolkit для керування станом та React Query для роботи з серверними даними. Для створення макетів використовувалася Figma, а основним інструментом розробки став Visual Studio Code.

Обраний набір технологій повністю відповідає поставленим завданням. Він забезпечує зручність розробки, надійність роботи системи та створює добру основу для подальшої реалізації програмної частини додатку.

РОЗДІЛ 2. ПРОЕКТУВАННЯ СИСТЕМИ ІНФОРМАЦІЙНОГО ДОДАТКА

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

Розроблюваний інформаційний додаток призначений для публікації та перегляду тематичного контенту у веббраузері. Система має забезпечувати зручний доступ до статей, їх групування за категоріями, перегляд коментарів і можливість реєстрації користувача для подальшої роботи з додатком.

Система передбачає три основні ролі користувачів, кожна з яких має свій набір доступних функцій. Гість — це відвідувач, який ще не увійшов у систему. Він може використовувати лише відкриті можливості додатку наведено на рисунку 2.1. Зокрема, переглядати головну сторінку з актуальними публікаціями, список статей та окремі матеріали з заголовком, текстом, датою створення та інформацією про автора. Також гість може читати коментарі під публікаціями та проходити реєстрацію або авторизацію для отримання розширених прав.

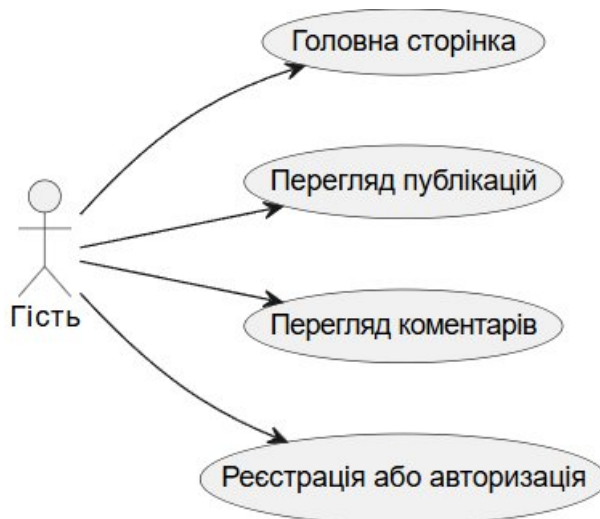


Рисунок 2.1 – Діаграма використання гостя

Авторизований користувач, який увійшов у систему після реєстрації, має доступ до всіх функцій гостя, а також до додаткових можливостей роботи з контентом і особистим профілем наведено на рисунку. Він може переглядати головну сторінку, окремі статті, коментарі та додавати власні. У особистому кабінеті

користувач бачить свої публікації, може виконувати повний цикл CRUD-операцій зі своїми постами та коментарями (створювати, переглядати, редагувати, видаляти), оновлювати персональні дані, зберігати обрані матеріали та виходити з облікового запису.

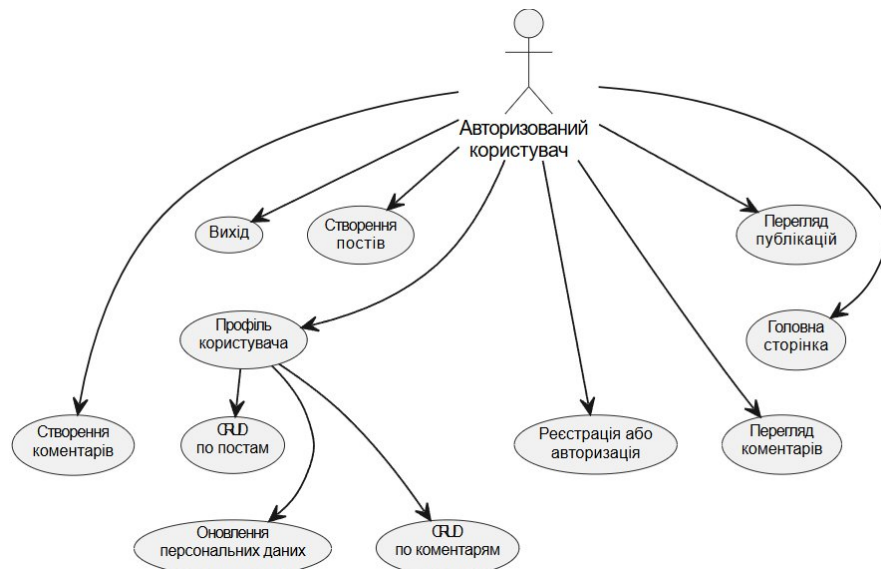


Рисунок 2.2 – Діаграма використання авторизованого користувача

Адміністратор має розширені права доступу і керує роботою всього додатку (див. рис. 2.3). Крім звичайних функцій, він працює в адміністративній панелі, де може виконувати CRUD-операції з будь-якими публікаціями, керувати користувачами (переглядати список, змінювати статус, видаляти облікові записи), модерувати коментарі, а також переглядати статистичну інформацію про проєкт.

Окрім цього, адміністратор має змогу налаштовувати глобальні параметри системи, такі як категорії публікацій, теги, ключові налаштування безпеки, параметри сповіщень та інші системні конфігурації без необхідності звертатися до розробників. Він також може ініціювати резервне копіювання бази даних, відновлювати дані з бекапу в разі збоїв, а також переглядати журнали подій (логи) для аналізу дій інших користувачів і виявлення потенційних порушень. Завдяки цим можливостям адміністратор виконує роль центрального координатора, що забезпечує стабільну, безпечну та ефективну роботу всього

додатку, оперативно реагуючи на будь-які критичні ситуації та зміни в бізнес-процесах проєкту.

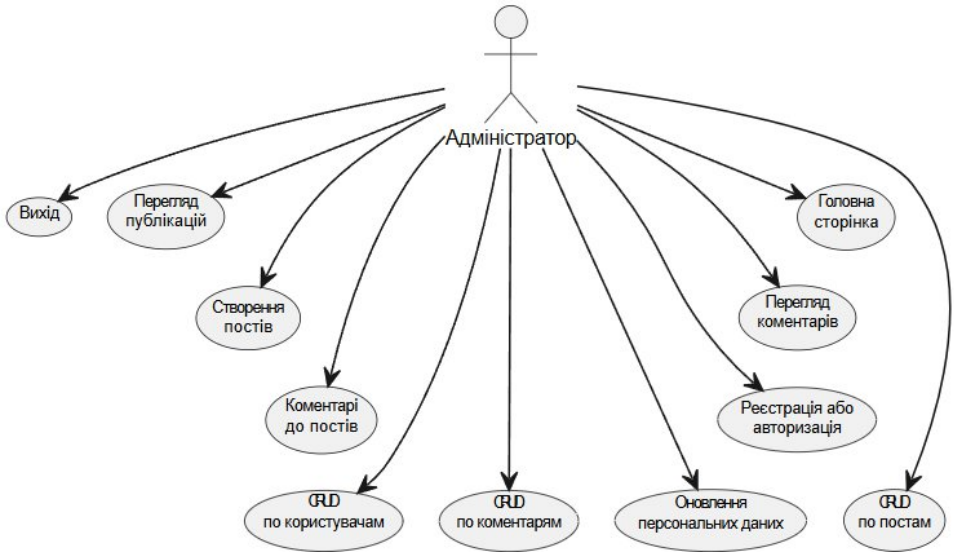


Рисунок 2.3 – Діаграма використання адміністратора

Підсумкове розмежування прав доступу за трьома ролями користувачів — гостем, авторизованим користувачем та адміністратором — наведено на рисунку 2.4. Така матриця наочно показує, що кожна наступна роль розширює можливості попередньої, а адміністративні функції відокремлено в окрему групу прав.

Можливість	Гість	Авторизований користувач	Адмін
Перегляд публікацій і коментарів	✓	✓	✓
Реєстрація та вхід у систему	✓	–	–
Коментування та оцінювання	–	✓	✓
Створення та редагування власних статей	–	✓	✓
Особистий профіль і збережені матеріали	–	✓	✓
Модерація коментарів	–	–	✓
Керування всіма публікаціями	–	–	✓
Керування користувачами та категоріями	–	–	✓

Рисунок 2.4 – Розмежування прав доступу за ролями користувачів

Функціональні вимоги до системи охоплюють кілька груп можливостей. Реєстрація та авторизація передбачають створення облікового запису, вхід і ви-

хід із додатку, причому доступ до захищених функцій здійснюється через JWT-токен, а адміністратор у своєму кабінеті переглядає користувачів і керує їхніми правами. Робота з публікаціями означає, що користувач створює, переглядає, редагує та видаляє власні статті, тоді як адміністратор керує всіма публікаціями системи через адміністративну панель. Робота з коментарями дозволяє авторизованому користувачеві додавати, редагувати й видаляти власні коментарі, а адміністраторові — переглядати та модерувати їх. Керування категоріями та контентом надає адміністраторові змогу створювати, редагувати й видаляти категорії для групування публікацій за тематикою та завантажувати зображення до статей через хмарний сервіс Cloudinary. Профіль користувача забезпечує перегляд особистого кабінету, оновлення імені, електронної пошти та пароля й керування власними публікаціями. Перегляд інформації доступний як гостям, так і зареєстрованим користувачам — це головна сторінка, список статей, окремі публікації, коментарі та службові сторінки «Про нас», «Контакти» і «FAQ», що автоматично відображають актуальні дані з бази MongoDB Atlas. Захист даних реалізовано через зберігання паролів у захищеному вигляді та обмеження доступу до адміністративних функцій перевіркою прав адміністратора [17].

Нефункціональні вимоги визначають якісні характеристики системи. Зручність використання передбачає простий і зрозумілий інтерфейс, орієнтовний час ознайомлення з основними розділами до п'яти-семи хвилин і коректне відображення на комп'ютері, планшеті та смартфоні завдяки адаптивному оформленню на базі Tailwind CSS. Продуктивність вимагає швидкого оброблення типових запитів — завантаження списку публікацій, перегляду окремої статті та роботи з коментарями — із часом відповіді до однієї-двох секунд для звичайних операцій і до трьох-п'яти секунд для складніших. Надійність означає стабільну роботу під час тривалого використання, коректне оброблення помилок з'єднання із сервером або базою даних і підтримку одночасної роботи кількох десятків користувачів без істотного погіршення швидкодії. Безпека охоплює захист облікових записів, розмежування прав звичайного користувача та

адміністратора й безпечний обмін даними між клієнтською та серверною частинами через REST API. Масштабованість і розвиток забезпечуються архітектурою клієнт–сервер, яка дозволяє окремо розширювати клієнтську та серверну частини й додавати нові функції (пошук, рейтинги, розширену модерацію) без повної перебудови системи, а хмарна база даних MongoDB Atlas дає змогу збільшувати обсяг зберігання за потреби. Супровід спрощується завдяки структуруванню коду за модулями — маршрути, контролери, моделі та React-компоненти, — що полегшує тестування, виправлення помилок і подальше оновлення програмного продукту.

На діаграмі класів відображено основні сутності розроблюваного додатку та контролери, які відповідають за обробку запитів до бази даних MongoDB через Mongoose. Структура системи побудована за об'єктно-орієнтованим підходом: кожен клас описує набір полів і методів для роботи з відповідним типом даних:

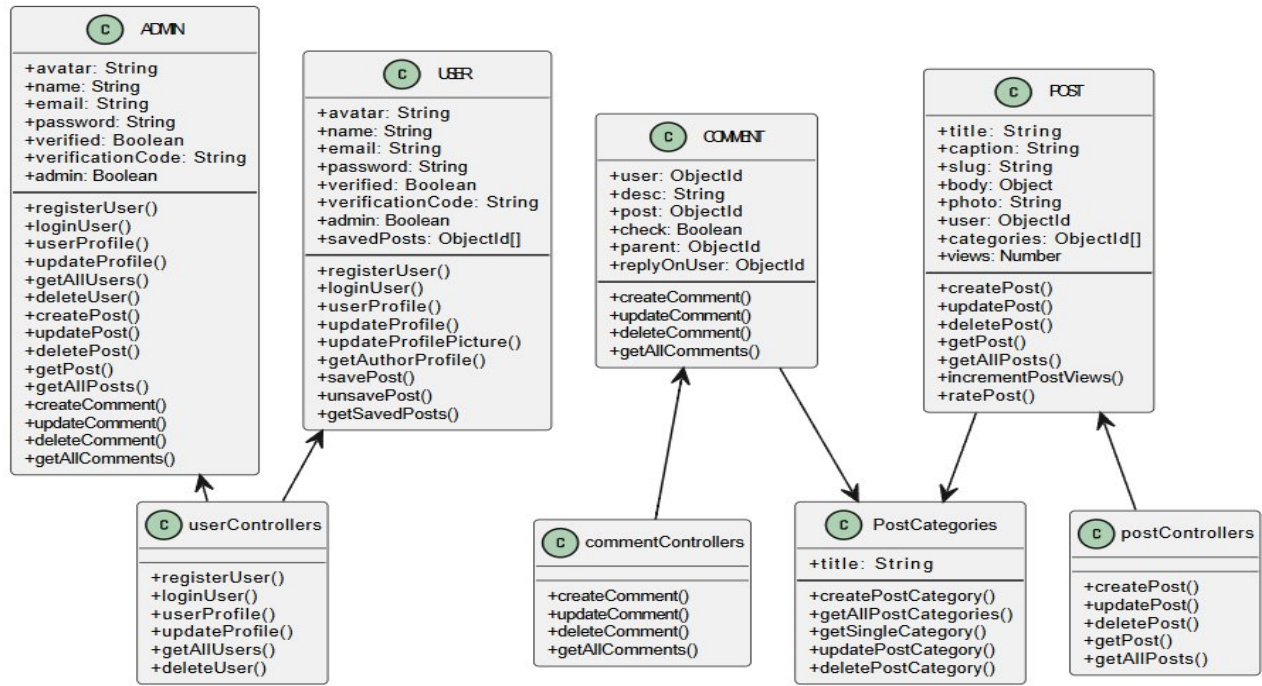


Рисунок 2.5 – Діаграма класів

Admin – клас адміністратора. У проєкті адміністратор є користувачем із полем admin: true, але на діаграмі виділено окремо, оскільки він має розширені права керування всіма сутностями додатку.

Таблиця 2.1

Структура даних користувача

avatar (String)	зображення профілю
name (String)	ім'я
email (String)	електронна пошта
password (String)	пароль
verified (Boolean)	статус підтвердження
verificationCode (String)	код перевірки
admin (Boolean)	ознака адміністратора
savedPosts (ObjectId[])	збережені публікації

У таблиці 2.1 наведено структуру, яка використовується для зберігання облікових записів у системі. Кожен користувач ідентифікується за унікальною електронною поштою (email) та має обов'язкове поле name, що відображає його ім'я. Поле password зберігає хешований пароль для автентифікації. Аватар користувача (avatar) є необов'язковим рядковим полем, що містить шлях або URL до зображення профілю.

Важливими службовими полями є verified (Boolean), яке вказує, чи підтверджено обліковий запис, та verificationCode (String) – унікальний код, що надсилається користувачеві для верифікації електронної пошти. Поле admin (Boolean) визначає рівень доступу: якщо значення true, користувач отримує розширені адміністративні права, описані в розділі 2.3. Поле savedPosts (ObjectId[]) є масивом посилань на публікації, які користувач зберіг у «обраному», що дозволяє реалізувати функцію персонального списку збережених матеріалів. Така структура забезпечує повний цикл управління користувачами – від реєстрації та верифікації до розмежування прав і збереження персональних даних.

Таблиця 2.2

Функціональні можливості API

loginUser()	вхід у систему;
registerUser()	реєстрація користувача;

userProfile()	перегляд профілю;
updateProfile()	оновлення даних;
updateProfilePicture()	зміна аватара;
getAuthorProfile()	перегляд профілю автора;
savePost() / unsavePost()	збереження публікації;
getSavedPosts()	отримання збережених матеріалів;
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.

У таблиці 2.2 систематизовано перелік методів API, які забезпечують взаємодію клієнтської частини з серверною логікою додатку. Функціонал умовно поділяється на кілька груп. Перша група стосується управління обліковими записами: ``loginUser()`` та ``registerUser()`` відповідають за автентифікацію та реєстрацію, ``userProfile()`` і ``updateProfile()`` дозволяють переглядати й редагувати особисті дані, а ``updateProfilePicture()`` – змінювати аватар користувача. Окремо виділено метод ``getAuthorProfile()`` для перегляду профілів інших авторів, що важливо для соціальної взаємодії в межах платформи.

Друга група методів реалізує роботу з публікаціями: ``createPost()``, ``updatePost()``, ``deletePost()`` забезпечують базові CRUD-операції, а ``getPost()``, ``getAllPost()`` та ``getOnePosts()`` надають можливості отримання списку постів або конкретної публікації. Також передбачено функції роботи зі збереженими

матеріалами – `'savePost()'`, `'unsavePost()'` та `'getSavedPosts()'`, що дозволяють користувачам формувати персональну добірку цікавих публікацій. Третя група стосується коментарів: методи `'createComment()'`, `'updateComment()'`, `'deleteComment()'` і `'getAllComments()'` забезпечують повноцінну систему обговорення контенту.

Таким чином, наведений у таблиці 2.2 набір методів формує єдиний інтерфейс прикладного програмування, який покриває всі ключові сценарії використання додатку – від реєстрації та авторизації до створення контенту, модерації коментарів і персоналізованого збереження матеріалів. Цей API є основою для реалізації як клієнтської частини, так і адміністративної панелі, оскільки більшість методів (крім адміністративних функцій, описаних окремо) доступні всім авторизованим користувачам із відповідним розмежуванням прав на рівні серверної логіки.

USER — клас користувача системи. Зберігає дані облікового запису та забезпечує базові операції з профілем.

Таблиця 2.3

Поля моделі користувача

avatar (String)	зображення профілю;
name (String)	ім'я;
email (String)	електронна пошта;
password (String)	пароль;
verified (Boolean)	статус підтвердження;
verificationCode (String)	код перевірки;
admin (Boolean)	ознака адміністратора;
savedPosts (ObjectId[])	збережені публікації.

Таблиця 2.4

Опис методів авторизації

loginUser()	вхід у систему;
registerUser()	реєстрація користувача;

userProfile()	перегляд профілю;
updateProfile()	оновлення даних;
updateProfilePicture()	зміна аватара;
getAuthorProfile()	перегляд профілю автора;
savePost() / unsavePost()	збереження публікації;
getSavedPosts()	отримання збережених матеріалів.

POST – клас публікації. Описує структуру статті та операції з нею.

Таблиця 2.5

Опис методів обробки профілів

title (String)	заголовок;
caption (String)	короткий опис;
slug (тип string)	унікальний ідентифікатор поста;
body (Object)	основний текст;
photo (String)	зображення;
user (ObjectId)	автор;
categories (ObjectId[])	категорії;
views (Number)	кількість переглядів;
ratings (Array)	оцінки користувачів.

Таблиця 2.5 містить перелік основних полів, що використовуються для зберігання та обробки профілів (або постів) у системі. Кожен запис ідентифікується унікальним рядковим ідентифікатором (slug), що забезпечує зручну адресацію в URL-маршрутах, поряд із текстовим заголовком (title) та коротким описом (caption). Основний вміст винесено в об'єкт body, що дозволяє гнучко структурувати текст, а поле photo відповідає за зображення-прев'ю. Зв'язок із автором реалізовано через user (ObjectId), а тематичну належність — через масив categories, що дає змогу групувати матеріали за кількома рубриками одночасно.

Окрім контентної частини, таблиця фіксує поведінкові метрики: кількість переглядів (views) та масив ratings, у якому зберігаються оцінки користувачів

(наприклад, у вигляді об'єктів із полями `userId` і `value`). Ці поля є критичними для реалізації рейтингування, сортування за популярністю та аналітики активності. Така структура підтримує як статичні дані (заголовок, опис, фото), так і динамічні, що оновлюються в процесі роботи системи, забезпечуючи повноцінний цикл управління профілями від створення до модерації та ранжування.

Таблиця 2.6

Функції опрацювання публій

<code>createPost()</code>	створення;
<code>updatePost()</code>	редагування;
<code>deletePost()</code>	видалення;
<code>getPost()</code>	отримання однієї публікації;
<code>getAllPosts()</code>	отримання списку;
<code>incrementPostViews()</code>	підрахунок переглядів;
<code>ratePost()</code>	оцінювання публікації.

`Comment` – клас коментаря. Зв'язує користувача з публікацією та підтримує вкладені відповіді.

Таблиця 2.7

Поля моделі коментаря

<code>user (ObjectId)</code>	автор коментаря;
<code>desc (String)</code>	текст;
<code>post (ObjectId)</code>	публікація;
<code>check (Boolean)</code>	статус модерації;
<code>parent (ObjectId)</code>	батьківський коментар;
<code>replyOnUser (ObjectId)</code>	користувач, якому адресована відповідь.

Таблиця 2.8

Функції опрацювання коментарів

<code>createComment()</code>	додавання;
<code>updateComment()</code>	редагування;

deleteComment()	видалення;
getAllComment()	отримання даних про всі коментарі;
getAllComments()	отримання списку коментарів.

2.2. Розробка бази даних системи

Для зберігання та обробки інформації в розроблюваному додатку використовується хмарна база даних MongoDB Atlas. Оскільки MongoDB є документоорієнтованою системою, дані зберігаються у вигляді колекцій. Кожна колекція відповідає за окрему сутність системи:

Вибір документоорієнтованої моделі зумовлений характером даних інформаційного додатку: публікації, користувачі, категорії та коментарі мають різну й гнучку структуру, що погано лягає на жорстку реляційну схему з фіксованими таблицями. На відміну від розглянутих аналогів, які працюють переважно за редакційною моделлю та зберігають однотипні матеріали, розроблювана система передбачає активну участь користувачів — створення публікацій, коментування й оцінювання, — тому модель даних має легко розширюватися без складних міграцій. Документна модель MongoDB дозволяє зберігати, наприклад, тіло публікації у вигляді структурованого об'єкта редактора TipTap, а оцінки — як вкладений масив, що було б громіздким у класичній реляційній базі даних [10], [18].

Структуру бази даних побудовано так, щоб кожна колекція відповідала за окрему сутність системи: users зберігає дані про користувачів, posts — інформацію про публікації, comments — коментарі до статей, postcategories — категорії публікацій, а onlineusers — відомості про активних користувачів, що перебувають онлайн.

Тож, база даних складається з 5 колекцій бази даних (рис. 2.6): users, posts, comments, postcategories та onlineusers.



Рисунок 2.6 – Діаграма бази даних

Нижче буде наведено опис кожної з колекції бази даних наведено на рисунках 2.1 – 2.6.

Таблиця 2.10

Перелік колекцій бази даних

Назва таблиці	Призначення
users	Облікові записи користувачів
posts	Публікації
comments	Коментарі до публікацій
postcategories	Категорії матеріалів
onlineusers	Онлайн-активність

Таблиця 2.10 містить перелік основних колекцій, що формують ядро інформаційної системи. Центральне місце посідає колекція users, яка зберігає облікові записи, авторизаційні дані та персональну інформацію користувачів. Контентна складова реалізована через колекцію posts, де зберігаються всі публікації (їхні тексти, метадані, медіафайли та зв'язки з авторами). Для організації ієрархії та фільтрації матеріалів використовується postcategories, що дає змогу групувати записи за тематиками, а колекція comments забезпечує зворотний зв'язок, зберігаючи коментарі користувачів із прив'язкою до конкретних постів і авторів.

Окремої уваги заслуговує колекція onlineusers, яка, на відміну від інших, призначена не для довготривалого зберігання, а для моніторингу поточної активності - сесій, статусів "в мережі", часу останньої дії тощо. Це критично для реалізації функцій реального часу (наприклад, підрахунку відвідувачів онлайн або сповіщень). Усі колекції пов'язані між собою через зовнішні ключі (ObjectId), що забезпечує цілісність даних і дозволяє будувати складні запити (наприклад, вибірка постів із категоріями, авторами та кількістю коментарів). Така структура є збалансованою для систем із помірним навантаженням і передбачає подальше масштабування через індексацію ключових полів (userId, postId, categoryId).

Таблиця 2.11

Колекція users

Поле	Тип	Призначення
_id	ObjectId	Унікальний ідентифікатор
avatar	String	Фото профілю
name	String	Ім'я користувача
email	String	Електронна пошта
password	String	Пароль (зберігається у захищеному вигляді)
verified	Boolean	Статус підтвердження облікового запису
verificationCode	String	Код верифікації
admin	Boolean	Ознака адміністратора
savedPosts	ObjectId[]	Збережені публікації
createdAt	Date	Дата створення запису
updatedAt	Date	Дата останнього оновлення

Колекція users є центральною в системі, оскільки саме навколо облікового запису користувача будується вся логіка автентифікації, персоналізації та розмежування прав доступу. Кожен документ колекції однозначно ідентифікується

службовим полем `_id` типу `ObjectId`, яке MongoDB генерує автоматично й використовує для зв'язування користувача з його публікаціями та коментарями в інших колекціях. Основні відомості про особу зберігаються в полях `name` та `email`, причому електронна пошта є унікальною в межах колекції — це унеможливорює повторне створення того самого облікового запису й водночас дає змогу використовувати пошту як ідентифікатор під час входу.

Таблиця 2.12

Колекція `posts`

Поле	Тип	Призначення
<code>_id</code>	<code>ObjectId</code>	Унікальний ідентифікатор
<code>title</code>	<code>String</code>	Заголовок публікації
<code>caption</code>	<code>String</code>	Короткий опис
<code>slug</code>	<code>String</code>	Унікальна адреса статті
<code>body</code>	<code>Object</code>	Основний текст публікації
<code>photo</code>	<code>String</code>	Зображення до статті
<code>user</code>	<code>ObjectId</code>	Автор публікації
<code>categories</code>	<code>ObjectId[]</code>	Категорії матеріалу
<code>views</code>	<code>Number</code>	Кількість переглядів
<code>ratings</code>	<code>Array</code>	Оцінки користувачів
<code>createdAt</code>	<code>Date</code>	Дата створення
<code>updatedAt</code>	<code>Date</code>	Дата оновлення

Колекція `posts` є основним сховищем контенту додатку, адже саме в ній зберігаються всі публікації, навколо яких будується робота користувачів. Кожен запис однозначно ідентифікується службовим полем `_id` типу `ObjectId`, що дозволяє посилатися на конкретну публікацію з інших колекцій — зокрема з коментарів. Текстову основу матеріалу формують кілька полів: `title` містить заголовок, `caption` — короткий опис чи анонс, який зручно показувати у стрічці

публікацій, а body типу Object зберігає повний текст у структурованому вигляді, що його формує редактор статей у адмін-панелі. Окремо винесено поле slug — унікальну текстову адресу статті, за якою вона відкривається у браузері; завдяки йому посилання на публікацію залишаються зрозумілими для людини й стабільними навіть за зміни заголовка.

Таблиця 2.13

Колекція comments

Поле	Тип	Призначення
_id	ObjectId	Унікальний ідентифікатор
user	ObjectId	Автор коментаря
desc	String	Текст коментаря
post	ObjectId	Публікація, до якої належить коментар
check	Boolean	Статус модерації
parent	ObjectId	Батьківський коментар
replyOnUser	ObjectId	Користувач, якому адресована відповідь
createdAt	Date	Дата створення
updatedAt	Date	Дата оновлення

Колекція comments забезпечує взаємодію користувачів із публікаціями. Вона підтримує як звичайні коментарі, так і вкладені відповіді.

Таблиця 2.14

Колекція postcategories

Поле	Тип	Призначення
_id	ObjectId	Унікальний ідентифікатор
title	String	Назва категорії
createdAt	Date	Дата створення
updatedAt	Date	Дата оновлення

Колекція `postcategories` використовується для тематичного групування публікацій і полегшує навігацію по сайту.

Таблиця 2.15

Колекція `onlineusers`

Поле	Тип	Призначення
_id	ObjectId	Унікальний ідентифікатор
userId	ObjectId	Ідентифікатор користувача
lastSeen	Date	Час останньої активності
createdAt	Date	Дата створення запису
updatedAt	Date	Дата оновлення

Колекція `onlineusers` зберігає інформацію про активних користувачів. Записи автоматично видаляються, якщо користувач не проявляв активність протягом певного часу.

Між колекціями встановлено зв'язки через поля типу `ObjectId`: публікація пов'язана з користувачем і категоріями, коментар — з користувачем і публікацією. Доступ до бази даних здійснюється через бібліотеку `Mongoose`, яка описує структуру колекцій у вигляді моделей і виконує CRUD-операції.

Окрему роль відіграє колекція `onlineusers`, для якої налаштовано автоматичне видалення записів за технологією TTL-індексу: якщо користувач не виявляє активності протягом п'яти хвилин, відповідний запис вилучається, що дозволяє відстежувати кількість відвідувачів онлайн без накопичення застарілих даних. Зв'язки між колекціями реалізовано не через жорсткі зовнішні ключі, а через посилання типу `ObjectId`, які за потреби розгортаються засобами `Mongoose`; такий підхід зберігає гнучкість документної моделі й водночас забезпечує цілісність зв'язків між публікаціями, авторами, категоріями та коментарями.

2.3. Проектування та реалізація алгоритмів роботи системи

Робота інформаційного додатку побудована навколо двох основних ролей: звичайного користувача та адміністратора. Від ролі залежить, які дії доступні після входу в систему. Авторизація виконується через JWT-токен, після чого backend перевіряє права доступу до відповідних маршрутів API, що наведено на рисунку 2.7.

Після входу в обліковий запис із правами адміністратора користувач отримує доступ до адміністративного кабінету — окремого розділу, у якому зосереджено інструменти керування всім контентом системи. Тут адміністратор працює з публікаціями, виконуючи повний цикл операцій створення, перегляду, редагування та видалення статей, а також керує обліковими записами користувачів: переглядає їх список, змінює статус і за потреби видаляє окремі записи. Окремий напрям роботи становить модерація коментарів — їх перегляд, редагування та видалення, що дозволяє підтримувати якість обговорень під публікаціями. Крім того, адміністратор керує категоріями публікацій, упорядковуючи матеріали за тематикою для зручнішої навігації. Усі внесені у такий спосіб зміни не залишаються лише в інтерфейсі, а проходять через серверну частину на Express і зберігаються в базі даних MongoDB Atlas, тож результат кожної дії одразу стає доступним іншим користувачам системи.

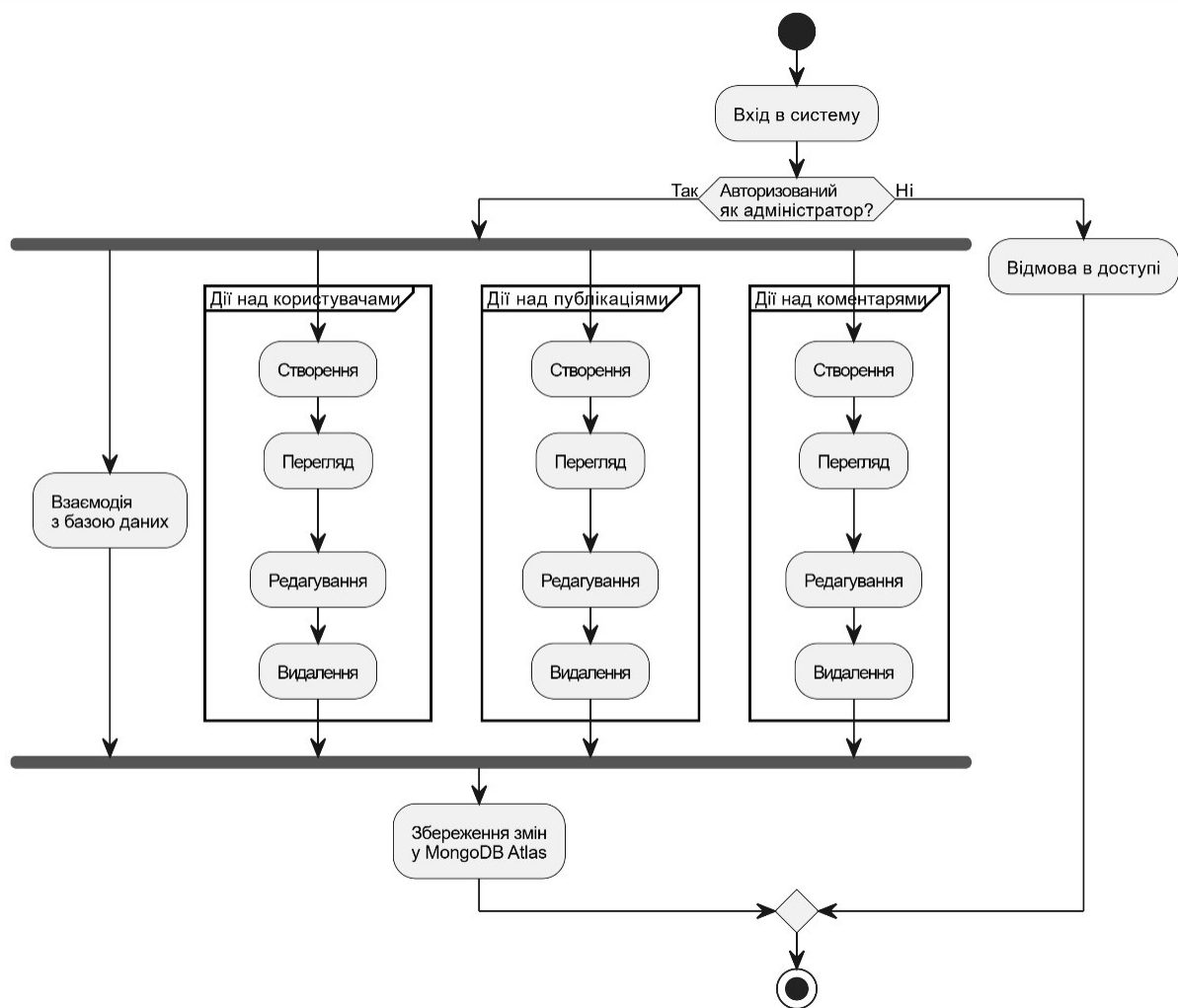


Рисунок 2.7 – Активність адміністратора

Адміністратор працює не лише з власним контентом, а з усіма сутностями системи. Усі операції проходять через REST API і контролери backend, після чого оновлені дані зберігаються в колекціях users, posts, comments та postcategories.

Алгоритм роботи авторизованого користувача показано на рисунку 2.8. Звичайний користувач після реєстрації або входу в систему працює як із відкритими, так і з особистими функціями додатку: він авторизується в обліковому записі, переглядає головну сторінку, список публікацій та окремі статті, взаємодіє з елементами сайту (коментує, оцінює, переглядає категорії), працює з особистим кабінетом, де доступні власні публікації та збережені матеріали,

створює, редагує й видаляє власні пости та коментарі, оновлює персональні дані у профілі й налаштуваннях, а виконані дії зберігаються в базі даних.

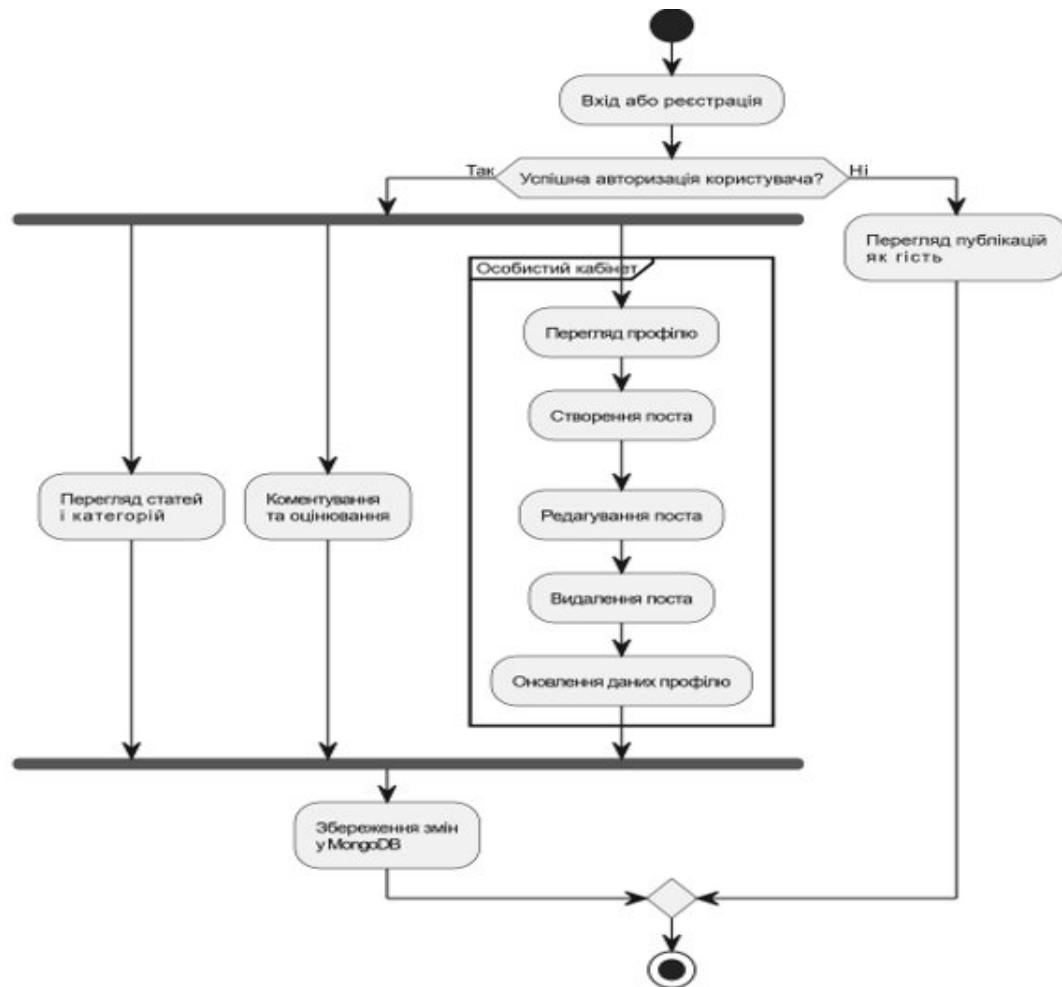


Рисунок 2.8 – Активність користувача

Якщо користувач не авторизований, він може лише переглядати контент і перейти до реєстрації або входу. Після успішної автентифікації відкривається доступ до функцій особистого кабінету.

Отже, у системі реалізовано розмежування доступу за ролями. Адміністратор керує всіма даними через адмін-панель, а авторизований користувач працює з власним контентом і профілем. Механізм авторизації визначає, які операції дозволені на кожному етапі роботи з додатком.

Незалежно від ролі користувача, оброблення будь-якого запиту в системі відбувається за єдиним узагальненим алгоритмом, наведеним на рисунку 2.9. Дія користувача в інтерфейсі ініціює HTTP-запит від клієнтської частини до серверного REST API; проміжне програмне забезпечення перевіряє коректність

запиту та, для захищених маршрутів, наявність і дійсність JWT-токена. Якщо перевірку пройдено, відповідний контролер виконує бізнес-логіку та звертається до бази даних через Mongoose, після чого сформований результат повертається клієнтові у форматі JSON і відображається в інтерфейсі; у разі помилки чи відмови в доступі користувач отримує відповідне повідомлення.

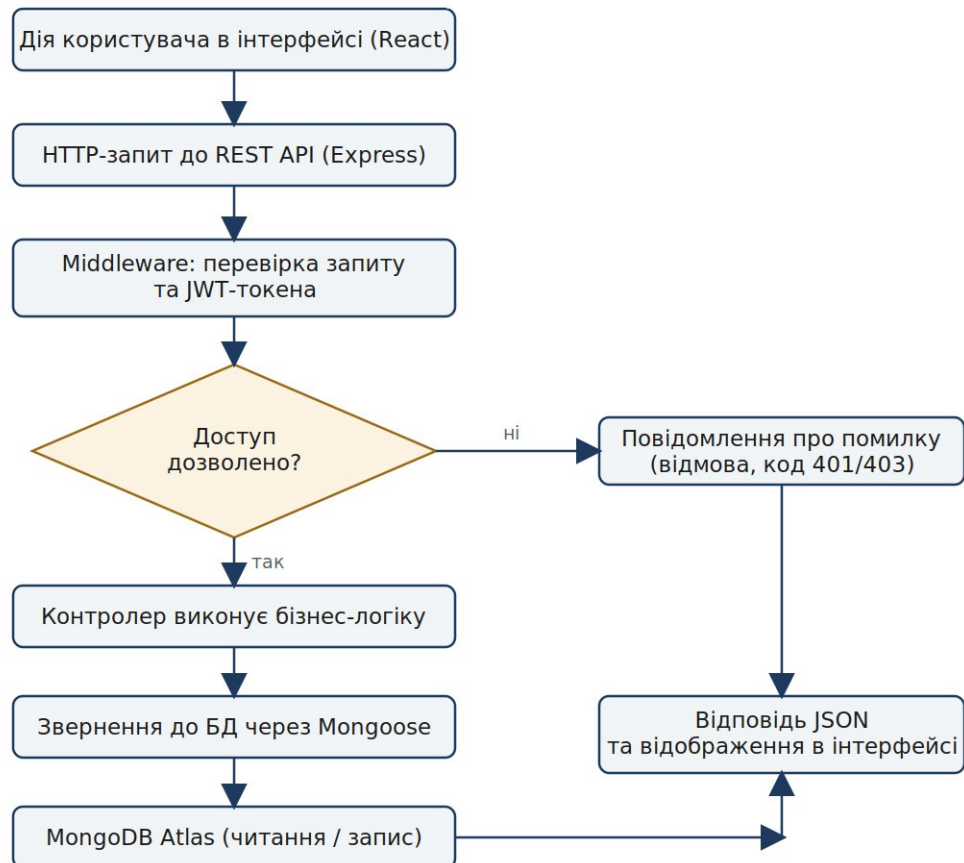


Рисунок 2.9 – Узагальнена блок-схема оброблення запиту в системі

2.4. Реалізація функціоналу інформаційного додатка

Реалізацію функціоналу розпочато з налаштування з'єднання з базою даних: функція `connectDB` у файлі `backend/config/db.js` використовує рядок підключення `DB_URI` із конфігураційного файлу `.env`, і після успішного з'єднання серверна частина готова обробляти запити (повний код модуля наведено в Додатку А). Далі у файлі `backend/server.js` сформовано структуру REST API, у

якій маршрути згруповано за функціональними модулями — користувачі (/api/users), публікації (/api/posts), коментарі (/api/comments), категорії (/api/post-categories), контактна форма та адміністративна панель; після побудови маршрутної ієрархії реалізовано контролери, що містять основну бізнес-логіку додатку [2].

Контролер користувачів `userControllers.js` відповідає за реєстрацію, авторизацію та роботу з профілем і містить функції `registerUser`, `loginUser`, `userProfile`, `updateProfile`, `updateProfilePicture`, `getAllUsers`, `deleteUser`, `getAuthorProfile`, `savePost`, `unsavePost` і `getSavedPosts`. Нижче розглянуто ключові з них із наведенням відповідних фрагментів коду.

1. `registerUser`: Реєстрація нового користувача. Перевіряє, чи не зайнята електронна пошта. Якщо ні — створює запис у колекції `users` і повертає дані разом із JWT-токеном

2. Функція `loginUser` працює схожим чином, але замість створення нового запису вона шукає існуючого користувача за `email` і перевіряє пароль. Якщо дані введені правильно, сервер також повертає токен. Якщо `email` не знайдено або пароль неправильний, користувач отримує відповідне повідомлення про помилку.

Окрім цього, у контролері реалізовано функції для перегляду профілю, оновлення особистих даних, зміни аватара, отримання списку всіх користувачів для адміністратора, видалення користувача, а також збереження і видалення улюблених публікацій.

3. `updateProfile`. Оновлення даних профілю користувача.

Звичайний користувач може змінити лише свої дані, а адміністратор може редагувати будь-якого користувача, а також змінювати поля `admin` і `verified`. Функція також перевіряє, щоб новий пароль був не коротший за 6 символів.

4. `getAllUsers`. Отримання списку всіх користувачів. Ця функція доступна тільки адміністратору. Підтримує пошук за `email`, пагінацію і повертає кори-

стувачів посторінково. Додаткова інформація про кількість записів передається в заголовках відповіді.

5. `updateProfilePicture`. Зміна аватара користувача. Функція завантажує нове зображення через `middleware uploadPicture`, зберігає посилання на нього в полі `avatar` і повертає оновлені дані користувача. Якщо файл не передано, аватар очищується.

6.`deleteUser`. Видалення користувача з системи. Функція доступна адміністратору. Спочатку видаляються всі пости цього користувача, коментарі до них і пов'язані файли, після чого видаляється сам обліковий запис.

7. `userProfile`. Отримання даних профілю поточного авторизованого користувача. Функція знаходить користувача за ідентифікатором з JWT-токена і повертає його основну інформацію без пароля.

4.Контролер `postControllers.js` відповідає за роботу з публікаціями в системі. Нижче описані всі функції цього модуля з фрагментами коду з проєкту.Основні функції, які виконує цей модуль, описані нижче:

1. `createPost`. Створення нової публікації. Функція приймає заголовок, опис, текст, категорії і за потреби зображення. Дані надходять у полі `document` у форматі JSON, а фото завантажується через `middleware uploadPicture`. Якщо `slug` не передано, він генерується автоматично з заголовка. Після перевірки категорій пост зберігається в колекції `posts`.

2. `updatePost`. Редагування існуючої публікації. Функція знаходить пост за `slug`, після чого оновлює його поля. Можна змінити заголовок, опис, текст, категорії і зображення. Якщо передано нове фото, воно замінює старе. Якщо `removePhoto` дорівнює `"true"`, зображення видаляється.

3. `deletePost`. Видалення публікації. Функція знаходить пост за `slug`, видаляє його з бази даних, а також усі коментарі, які були прив'язані до цієї публікації.

4. `getPost`. Отримання однієї публікації для сторінки перегляду. Функція знаходить пост за `slug` і повертає його разом з даними автора, категоріями та

схваленими коментарями. Вкладені відповіді на коментарі також підтягуються через populate.

5. getAllPosts. Отримання списку публікацій. Ця функція використовується на головній сторінці і в блозі. Підтримує пошук за заголовком, фільтрацію за категоріями, фільтрацію за автором, пагінацію і сортування. Можна сортувати за датою створення або за популярністю. Для підрахунку середнього рейтингу використовується aggregation pipeline MongoDB.

6. incrementPostViews. Підрахунок переглядів публікації. Кожного разу, коли користувач відкриває статтю, лічильник views збільшується на одиницю. Це потрібно для відображення популярності матеріалів.

7. ratePost. Оцінювання публікації користувачем. Функція дозволяє поставити оцінку від 1 до 5. Якщо користувач вже оцінював цей пост, його оцінка оновлюється. Якщо ні, додається новий запис у масив ratings. Після збереження повертається середній рейтинг і оцінка поточного користувача.

5. Контролер commentControllers.js відповідає за роботу з коментарями під публікаціями. У модулі чотири функції: createComment, updateComment, deleteComment та getAllComments.

1.createComment. Додавання нового коментаря до публікації. Функція отримує текст коментаря, slug поста, а також за потреби parent і replyOnUser для вкладених відповідей. Спочатку знаходить публікацію за slug, потім створює запис у колекції comments і прив'язує його до поста та авторизованого користувача.

2. updateComment. Редагування існуючого коментаря. Функція знаходить коментар за commentId, після чого може змінити його текст (desc) або статус модерації (check). Поле check використовується для того, щоб адміністратор міг схвалювати або приховувати коментарі перед показом на сайті.

3. deleteComment. Видалення коментаря. Функція знаходить коментар за ідентифікатором і видаляє його з бази даних. Разом з ним видаляються і всі дочі-

рні відповіді, у яких цей коментар є батьківським (parent). Це потрібно, щоб після видалення головного коментаря не залишались «висячі» відповіді.

4. getAllComments. Отримання списку всіх коментарів для адміністративної панелі. Функція доступна тільки адміністратору. Підтримує пошук за текстом коментаря, пагінацію і повертає додаткову інформацію про автора, публікацію, батьківський коментар і користувача, якому адресована відповідь.

5.Бібліотека Axios. На клієнтській частині додатку для звернення до серверного API використовується бібліотека Axios. Вона надсилає HTTP-запити з React-компонентів на backend Express і отримує відповідь у форматі JSON. Frontend напряду з базою даних не працює: спочатку йде запит на API, потім сервер звертається до MongoDB Atlas і повертає результат.

6.Схема користувача для моделі користувача у базі даних MongoDB за допомогою пакету Mongoose. Там зберігається ім'я, електронна пошта, пароль, фото профілю, а також додаткова інформація: чи підтверджений акаунт, чи є людина адміністратором, які публікації вона зберегла собі. Кожен користувач має свій унікальний email, тому одну й ту саму пошту не можна використати двічі.

Пароль у базу не записується у звичайному вигляді. Перед збереженням він спеціально перетворюється, щоб його не можна було просто прочитати. Це робиться для безпеки. Коли людина входить у систему, введений пароль порівнюється з тим, що вже є в базі. Після успішного входу або реєстрації користувач отримує спеціальний токен. Він потрібен, щоб система розуміла, хто саме звертається до неї, коли людина, наприклад, створює пост, змінює профіль або залишає коментар.

7. Схема публікацій. Усі статті в додатку зберігаються за допомогою окремої моделі поста. Вона знаходиться у файлі backend/models/Post.js і описує, з чого складається одна публікація в базі даних.

Кожен пост має заголовок, короткий опис, основний текст і за потреби зображення. Також у ньому є унікальна адреса slug, за якою можна відкрити

конкретну статтю в браузері. Наприклад, замість довгого посилання з ідентифікатором використовується зрозуміліша назва.

Окремо зберігається інформація про автора поста і категорії, до яких він належить. Завдяки цьому на сайті можна показувати, хто написав статтю, і групувати матеріали за темами. Також у пості ведеться кількість переглядів і оцінки від користувачів. Це потрібно, щоб бачити, які статті популярніші. Коментарі до поста напряму в цій моделі не зберігаються, але вона пов'язана з ними. Тобто коли відкривається стаття, система може підтягнути всі коментарі, які належать саме до цієї публікації.

8. Автентифікації користувача вона використовує дві функції: `authGuard` і `adminGuard`. `authGuard` перевіряє, чи користувач увійшов у систему. Без правильного токена доступ до захищених дій закритий. Якщо токен присутній і має правильний формат він розшифровується за допомогою секретного ключа `process.env.JWT_SECRET`. Після розшифрування отримується ідентифікатор користувача, який використовується для пошуку користувача в базі даних.

`adminGuard`: `adminGuard` перевіряє, чи користувач є адміністратором. Без цього він не може керувати користувачами, категоріями та коментарями в адмін-панелі. Якщо ж умови не виконуються, генерується помилка авторизації зі статусом 401 Unauthorized.

9. Маршрутизація. У проєкті для кожної частини системи створено окремі файли маршрутів у папці `routes`. Це адреси, за якими `frontend` звертається до сервера. В `server.js` ці маршрути підключаються до головного сервера.

У файлі `userRoutes.js` описано маршрути для роботи з користувачами. Реєстрація і вхід доступні без авторизації. Перегляд профілю, зміна даних, збереження статей і робота з аватаром — лише після входу в акаунт. Перегляд усіх користувачів і видалення акаунта доступне тільки адміністратору. Перед виконанням захищених дій `middleware` перевіряє, чи користувач авторизований, а для адмінських функцій — чи має він відповідні права.

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

Окрему увагу в системі приділено механізмам автентифікації та авторизації, які захищають частину маршрутів від несанкціонованого доступу. Автентифікація з'ясовує, хто саме звертається до системи, а авторизація визначає, які дії цьому користувачеві дозволені; обидва механізми реалізовано на серверній частині, тому приховування елементів інтерфейсу на клієнті не є єдиним рівнем захисту.

Паролі користувачів ніколи не зберігаються у відкритому вигляді. У моделі User (файл backend/models/User.js) визначено проміжний обробник pre(«save»), який перед збереженням документа перевіряє, чи змінювалося поле пароля, і за потреби хешує його функцією bcrypt із десятима раундами солі. Перевірка введеного пароля під час входу виконується методом comparePassword, що порівнює введене значення зі збереженим хешем засобами bcrypt, не відновлюючи початковий пароль [20].

Після успішної реєстрації або входу сервер видає користувачеві токен JSON Web Token. Метод generateJWT підписує токен, який містить ідентифікатор користувача, секретним ключем JWT_SECRET зі строком дії тридцять днів. Цей токен клієнтська частина зберігає й додає до заголовка Authorization кожного наступного запиту у форматі Bearer, завдяки чому сервер ідентифікує користувача без повторного введення облікових даних [17].

Перевірку автентифікації інкапсульовано у проміжному програмному забезпеченні authGuard (файл backend/middleware/authMiddleware.js). Воно зчитує заголовок Authorization, переконується, що той починається з ключового слова Bearer, виокремлює токен і перевіряє його методом jwt.verify із тим самим секретним ключем. У разі успіху за ідентифікатором із токена завантажується відповідний користувач без поля пароля й додається до об'єкта запиту як req.user, після чого керування передається далі; якщо ж токен відсутній або недійсний, клієнт отримує помилку з кодом 401.

Авторизацію адміністративних дій забезпечує проміжне програмне забезпечення adminGuard, яке виконується після authGuard і пропускає запит лише тоді, коли в завантаженого користувача встановлено ознаку admin. Завдяки цьому захищені маршрути оформлюються лаконічно: наприклад, видалення користувача описується як послідовність authGuard, adminGuard і функції deleteUser, де перший обробник підтверджує особу, другий — її права, а третій виконує саму операцію. Такий поділ відповідальності робить логіку доступу прозорою та придатною для повторного використання на будь-якому маршруті.

Узагальнену послідовність автентифікації та авторизації запиту до захищеного маршруту наведено на рисунку 2.10.



Рисунок 2.10 – Послідовність автентифікації та авторизації на основі JWT

Описаний підхід поєднує надійне зберігання паролів, перевірку особи за токеном і розмежування прав на рівні маршрутів, що загалом забезпечує достатній рівень захисту інформаційного додатку за порівняно простої та зрозумілої реалізації.

Висновки до розділу 2

У розділі описано структуру додатку та принцип його роботи. Визначено три ролі користувачів: гість, авторизований користувач і адміністратор. Кожна

роль має свій набір дій — перегляд контенту, створення публікацій, коментування або керування системою через адмін-панель.

Побудовано діаграми використання та діаграму класів, які показують основні сутності додатку і зв'язки між ними.

У пункті 2.4 описано програмну частину: підключення до бази даних, маршрути API, контролери та роботу frontend через Axios. Розглянуто захист маршрутів через authGuard і adminGuard.

Окрему увагу приділено механізмам автентифікації та авторизації: вхід користувача підтверджується JWT-токеном, який клієнтська частина додає до кожного наступного запиту, паролі зберігаються виключно у захищеному (хешованому) вигляді, а проміжне програмне забезпечення authGuard і adminGuard перевіряє відповідно особу користувача та його права перед виконанням захищених операцій. Такий підхід дає змогу розмежовувати дії звичайного користувача й адміністратора безпосередньо на рівні серверної логіки, а не лише приховуванням елементів інтерфейсу.

Спроектowana структура додатку відображає принцип розподілу відповідальності: клієнтська частина відповідає за відображення даних і взаємодію з користувачем, серверна — за бізнес-логіку, оброблення запитів і захист маршрутів, а хмарна база даних MongoDB Atlas — за надійне збереження інформації. Завдяки документоорієнтованій моделі бази даних і реалізації зв'язків через ідентифікатори ObjectId систему можна порівняно легко розширювати новими сутностями та функціями без суттєвої перебудови наявної архітектури.

Загалом у розділі сформовано цілісне уявлення про структуру, базу даних і програмну реалізацію додатку, що стало основою для подальшого опису інтерфейсу та тестування системи в наступному розділі.

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З СИСТЕМОЮ ДОДАТКА

3.1 Порядок встановлення та налаштування системи

Веборієнтований інформаційний додаток є звичайним вебсайтом. Щоб ним користуватися, достатньо відкрити його в браузері на комп'ютері, планшеті або телефоні. Окремо встановлювати програму на пристрій не потрібно. Для нормальної роботи сайту достатньо сучасного пристрою з доступом до інтернету, операційної системи Windows, macOS або Android і будь-якого популярного браузера, наприклад Google Chrome, Mozilla Firefox, Microsoft Edge або Safari. Також бажано мати не менше 2 ГБ оперативної пам'яті, щоб сторінки завантажувались без затримок.

Для звичайного користувача не потрібне спеціальне обладнання чи додаткові налаштування. Достатньо перейти за адресою сайту і почати роботу: переглядати статті, читати коментарі, реєструватися або вводити в акаунт. Якщо ж мова йде про запуск додатку під час розробки, тоді на комп'ютері потрібно встановити Node.js, завантажити файли проєкту, виконати `npm install` у папках `frontend` і `backend`, налаштувати файл `.env` з підключенням до MongoDB Atlas і запустити обидві частини: сервер на порту 5000 і сайт на порту 3000.

Розгортання середовища розробки складається з кількох послідовних кроків. Спершу на комп'ютер встановлюють середовище виконання Node.js разом із менеджером пакетів `npm`, після чого завантажують вихідні файли проєкту, що містять окремі каталоги клієнтської (`frontend`) та серверної (`backend`) частин. У кожному з каталогів виконують команду `npm install`, яка завантажує всі залежності, перелічені у файлі `package.json`, — зокрема React, Redux Toolkit і React Query для клієнта та Express, Mongoose і пакети роботи з JWT для сервера [11], [2].

Перед першим запуском налаштовують файл конфігурації `.env` у серверній частині. У ньому зберігаються рядок підключення до хмарної бази даних MongoDB Atlas, секретний ключ для підписування токенів, а також облікові дані

сервісу Cloudinary для завантаження зображень. Винесення цих параметрів в окремий файл середовища дозволяє не зберігати конфіденційні дані безпосередньо в коді та змінювати налаштування без його редагування, що є важливим і з погляду безпеки, і з погляду зручності супроводу [18].

Після налаштування конфігурації обидві частини додатку запускають окремо: серверну — командою запуску Node.js-застосунку, що піднімає REST API на порту 5000, а клієнтську — командою розробницького сервера, доступного на порту 3000. Під час розробки клієнт звертається до сервера за вказаною адресою, а зміни в коді автоматично відображаються в браузері завдяки механізму гарячого перезавантаження. Такий поділ на дві незалежні частини спрощує одночасну роботу над інтерфейсом і логікою та полегшує подальше розгортання системи.

Далі на рисунку 3.1 показано, як у цілому влаштована система. Користувач працює через браузер на своєму комп'ютері. Браузер відображає сторінки сайту: список постів, окрему статтю, профіль і коментарі. Сам додаток обробляє ці дії і за потреби звертається до бази даних, де зберігаються публікації, користувачі та коментарі. Це допомагає зрозуміти, що користувач не працює з базою напряму, а всі дані проходять через додаток.

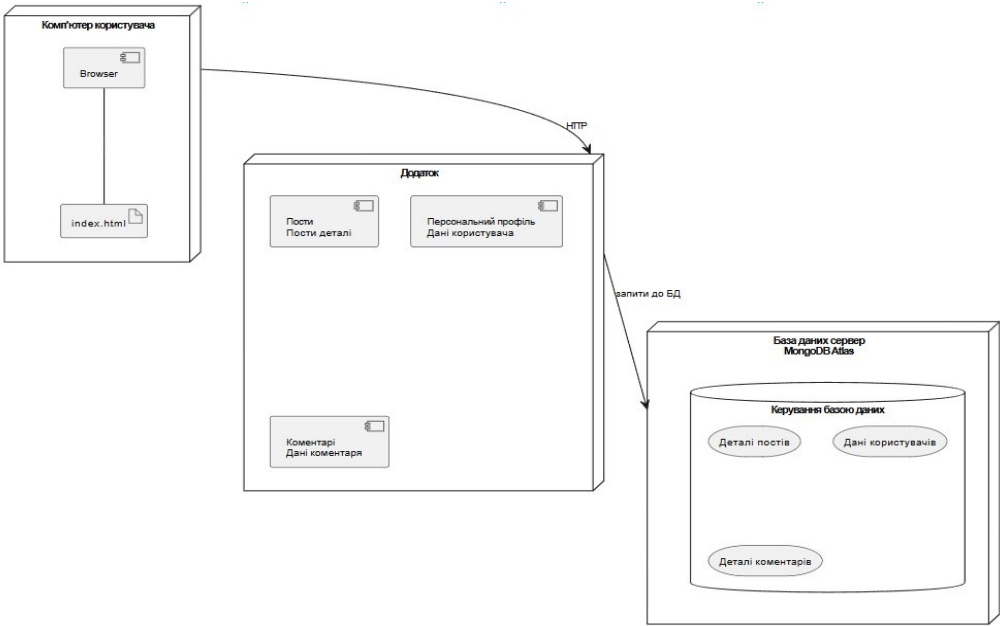


Рисунок 3.1 – Діаграма розгортання

Діаграма розгортання показує, з яких частин складається система і як вони пов'язані між собою. На ній видно три основні блоки: комп'ютер користувача, сам додаток і база даних. Нижче коротко описано кожен з них у контексті розроблюваного інформаційного додатку:

Діаграма розгортання показує, з яких частин складається система та як вони пов'язані між собою; на ній виокремлено три основні блоки — комп'ютер користувача, сам додаток і сервер бази даних. Вузол у нотації розгортання позначає окрему частину системи, що виконує певну роботу, тож вузли показують, де саме відбувається оброблення запитів, відображення сторінок і збереження даних. Вузол «Комп'ютер користувача» — це пристрій (телефон, планшет або ноутбук), з якого людина через браузер відкриває сторінки додатку, переглядає статті, читає коментарі, реєструється чи входить в обліковий запис; саме тут відбувається робота з інтерфейсом, проте дані на пристрої постійно не зберігаються, а надходять із серверної частини. Вузол «Додаток» є основною програмною частиною, яка відповідає за показ публікацій, перегляд окремої статті, особистий профіль, коментарі та інші функції: будь-яка дія користувача спочатку потрапляє сюди, після чого додаток обробляє її та за потреби звертається до бази даних. Вузол «Сервер бази даних» — це хмарна база MongoDB Atlas, у якій зберігаються записи про користувачів, публікації, коментарі, категорії та іншу інформацію; саме звідти додаток бере дані для відображення списку статей і туди записує нові коментарі чи оновлення профілю, завдяки чому інформація не втрачається після закриття браузера та доступна під час наступного входу.

Описана структура наочно демонструє, що користувач ніколи не працює з базою даних напряму: усі дані проходять через додаток, який виконує роль посередника між інтерфейсом і сховищем. Завдяки цьому забезпечується контроль доступу, перевірка прав і коректне оброблення помилок на рівні серверної логіки, а клієнтська частина залишається відповідальною лише за відображення інформації та взаємодію з користувачем.

Для промислового розгортання клієнтську частину збирають у статичний набір файлів, який розміщують на вебсервері або у спеціалізованому хостингу для статичних застосунків, тоді як серверну частину розгортають на хостингу з підтримкою Node.js. Хмарні сервіси MongoDB Atlas і Cloudinary при цьому залишаються незмінними, оскільки вони не залежать від конкретного середовища розгортання та доступні через мережу. Такий підхід спрощує масштабування: за зростання навантаження клієнтську та серверну частини можна розвивати й розміщувати незалежно одна від одної, а обсяг хмарного сховища збільшувати за потреби без зміни архітектури системи.

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

Сайт відкривається в браузері за адресою <http://localhost:3000/>. У верхній частині сторінки є шапка з логотипом і меню: «Головна», «Блог», «Сторінки» (Про нас, Зв'язатись з нами), «FAQ». Якщо користувач не увійшов у систему, справа показується кнопка «Увійти». Після входу з'являється меню «Акаунт», звідки можна перейти в профіль, налаштування, адмін-панель (для адміністратора) або вийти з облікового запису. У шапці сайту є посилання для швидкого переходу до інших сторінок. Щоб отримати доступ до особистих налаштувань і додаткових функцій — потрібно увійти в акаунт. Демонстрація роботи сайту проводиться від імені адміністратора, який має повний доступ до керування контентом.

На головній сторінці (див. рис. 3.2) також показані найновіші публікації, тож користувач одразу бачить свіжий контент сайту. Це дає змогу швидко ознайомитися з останніми матеріалами, не переходячи спочатку в інші розділи (див. рис. 3.3). Нові пости можуть містити статті на різні теми, огляди, корисні матеріали та іншу інформацію, яка може зацікавити відвідувача. Тому головна сторінка фактично стає місцем, де зосереджена актуальна інформація, що полегшує роботу з сайтом і робить його зручнішим для користувача.

Читайте найцікавіші статті

Місце, де можна читати, писати та поглиблювати своє розуміння

 Пошук статті

Пошук

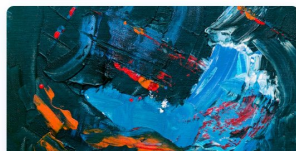
Популярні пошукові запити:

Дизайн

Користувачський досвід

Користувачський інтерфейс

Нові ▾



Звew



Чарівний вокзал Києва



Нульові викиди: Мрія чи

Рисунок 3.2 – Головна сторінка



Звew

www

★ 0 2

зробив
Верифікований автор

23 квітня



Чарівний вокзал Києва

побував в чарівному місті

★ 4.5 30

Девід
Верифікований автор

28 травня

Нульові викиди: Мрія чи
реальність

ЕкоUkraine: крок до рівноваги

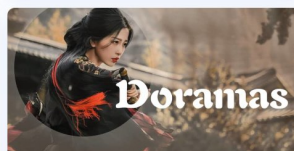
★ 4.0 20

Весельська Ангеліна
Верифікований автор

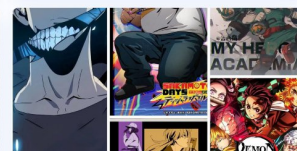
27 травня



Синдром самозванця



Дорами: Феномен, що захопив



Топ-5 найпопулярніших аніме

Рисунок 3.3 – Публікації

Користувач може відкрити будь-яку публікацію і прочитати її повністю (див. рис. 3.4). Для цього достатньо натиснути на картку статті на головній сторінці або в розділі «Блог». На сторінці публікації показуються заголовок, основний текст, фото, автор, категорії, кількість переглядів і оцінка. Якщо користувач увійшов у систему, він також може залишити коментар або поставити оцінку.

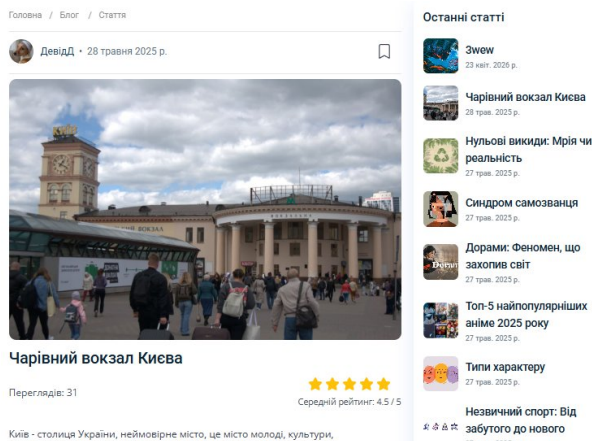


Рис. 3.4 – Детальний зміст публікації

Якщо прокрутити сторінку вниз, з’явиться блок із коментарями (див. рис. 3.5). Тут користувачі можуть залишати свої думки, ставити запитання або відповідати на інші коментарі. Усі коментарі показуються під публікацією, тому з ними зручно працювати. Додати коментар може лише той, хто увійшов у свій акаунт.

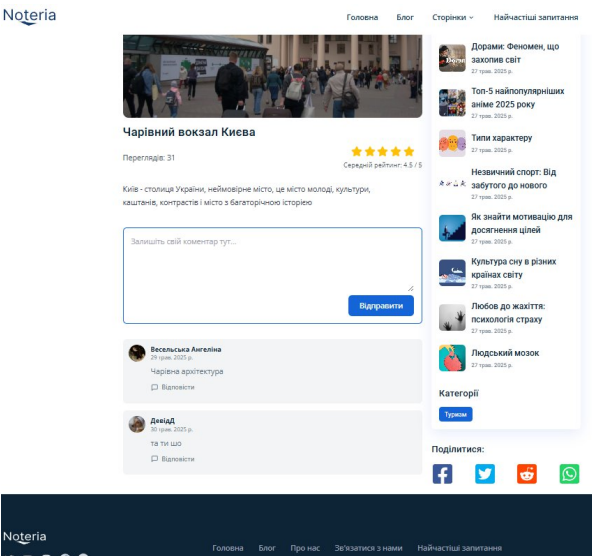


Рисунок 3.5 – Секція з коментарями

Окремо варто розглянути сторінку «Блог» (<http://localhost:3000/blog>) (див. рис. 3.6). Тут показано всі публікації сайту, а не лише частину, як на головній. Користувач може переглянути повний список матеріалів, обрати потрібну категорію, відсортувати записи і перейти на наступну сторінку, якщо статей багато. Це зручно, коли потрібно знайти конкретну тему або переглянути весь контент сайту.

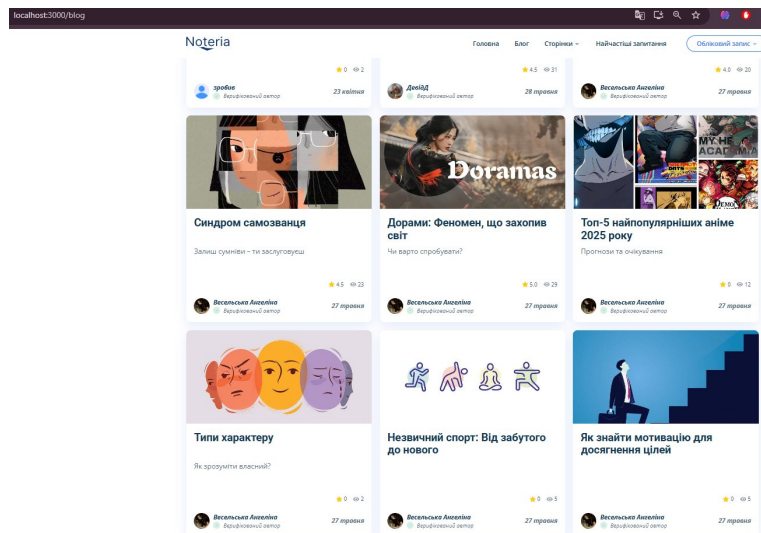


Рисунок 3.6 – Сторінка «Блог»

Через меню «Сторінки» користувач може перейти на «Про нас» за адресою <http://localhost:3000/about> (див. рис. 3.7). Там розміщена інформація про сам проєкт, його мету і призначення. Окремо на сайті є сторінки «Зв'язатись з нами» (/contact) і «FAQ» (/faq), де можна прочитати відповіді на поширені питання або надіслати повідомлення через форму.

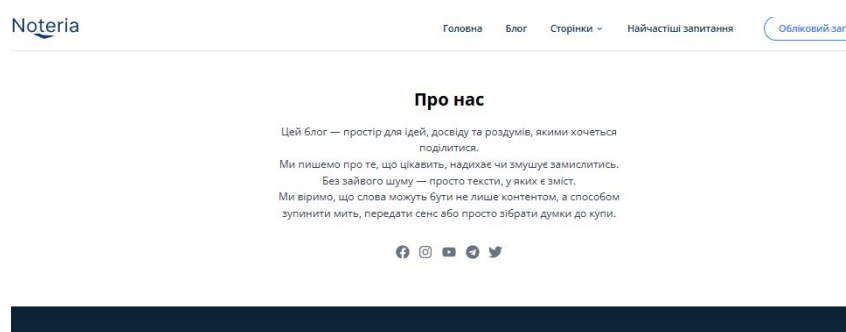


Рисунок 3.7 – Сторінка “Про нас”

На сторінці «Блог» реалізовано пошук публікацій (див. рис. 3.8). Користувач вводить назву або частину назви статті, після чого список оновлюється і показує лише відповідні результати (див. рис. 3.9). Це допомагає швидко знайти потрібний матеріал, коли на сайті вже є багато публікацій.

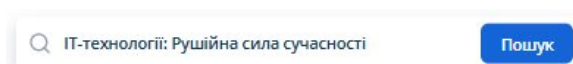


Рисунок 3.8 – Секція пошуку

«Акаунт» → «Профіль» можна швидко перейти на цю сторінку. Також у профілі видно, які статті вже створив користувач.

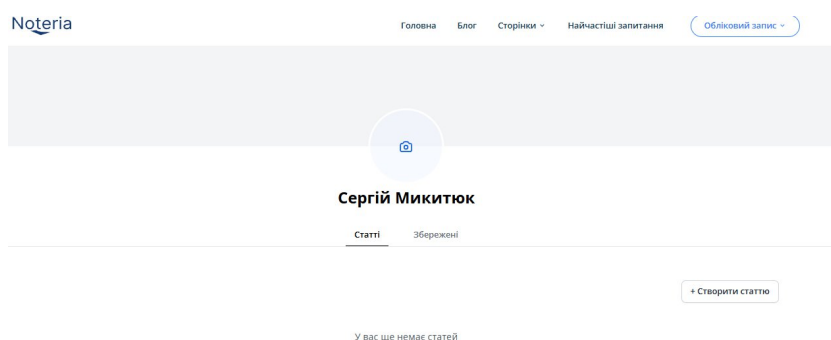


Рисунок 3.11 – Профіль користувача

Оновити особисті дані можна в налаштуваннях (див. рис. 3.12). Там користувач змінює ім'я, email, пароль і фото профілю. Ця сторінка відкривається через меню «Акаунт» → «Налаштування».

Налаштування


Видалити

Ім'я

Електронна пошта

Пароль

Оновити профіль

Рисунок 3.12 – Налаштування профілю

Зареєстрований користувач також може створити власну публікацію (див. рис. 3.13). Для цього потрібно перейти на сторінку <http://localhost:3000/create>, ввести заголовок, опис, текст, обрати категорії і за потреби додати зображення. Після збереження стаття з'являється на сайті.

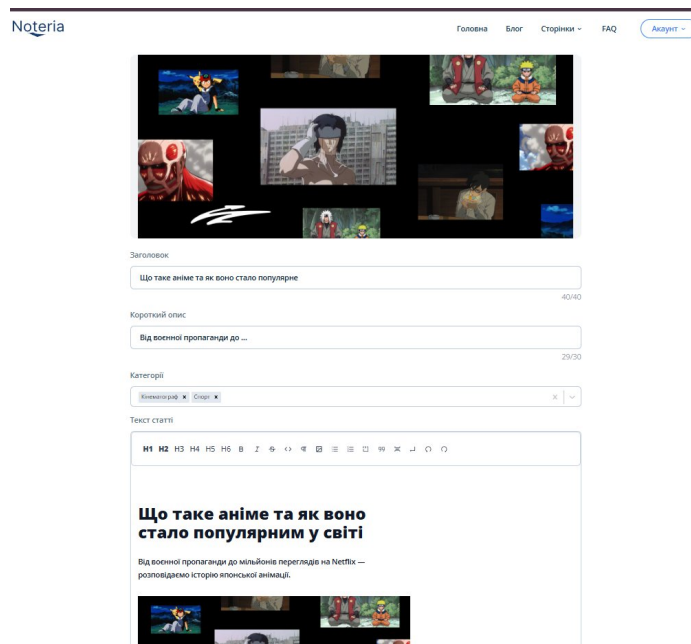


Рисунок 3.13 – Сторінка створення публікації

Після створення публікацію можна одразу переглянути на сайті (див. рис. 3.14). Вона з'являється в блозі, на головній сторінці і в профілі автора. Користувач може перевірити, чи правильно відображаються заголовок, текст, фото і категорії.

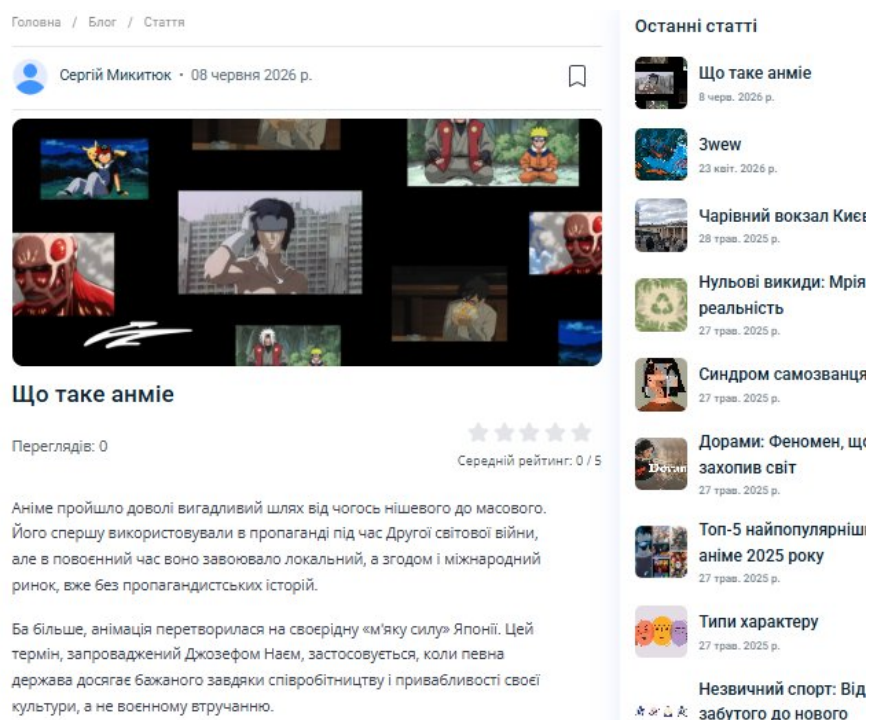


Рисунок 3.14 – Сторінка створеної публікації

Окремо варто розглянути адміністративний кабінет (див. рис. 3.15). Адміністратор заходить у нього через меню «Акаунт» → «Адмін панель» або за адресою <http://localhost:3000/admin>. На головній сторінці панелі показується статистика: скільки є користувачів, публікацій і хто зараз онлайн.

Noteria Статистика

Користувачів: 8 Статті: 41 Онлайн зараз: 1

★ Top-10 постів за рейтингом

№	Назва статті	Автор	Середній рейтинг	К-сть оцінок	Перегляди	Дата публікації
1	Дорамки: Феномен, що захопив світ	Веселыська Ангеліна	5	2	29	12.04.2026
2	Топ 5 фін про які має знати кожен один два	mila	5	2	18	17.04.2026
3	манора	mila	5	1	13	21.05.2026
4	«Четверте крило»	Веселыська Ангеліна	5	1	5	16.04.2026
5	Любов до життя: психологія страху	Веселыська Ангеліна	5	1	3	31.05.2026
6	Чарівний вокзал Києва	DavidD	4.5	2	33	06.05.2026
7	Синдром самозванця	Веселыська Ангеліна	4.5	2	23	09.04.2026
8	Нульові викиди: Мрія чи реальність	Веселыська Ангеліна	4	1	22	10.05.2026
9	sample title	djfk	4	1	7	18.05.2026
10	kiskisatmesomscow	mila	0	0	12	31.05.2026

Рисунок 3.15 – Головна сторінка панелі адміністратора

Розглянемо декілька можливих варіантів роботи з адміністративною панеллю. Наприклад, додавання та видалення “Коментарів” (див. рис. 3.16). У розділі «Коментарі» адміністратор бачить усі коментарі на сайті (див. рис. 3.16). Він може їх переглянути, відредагувати, схвалити або видалити. Це потрібно для модерації обговорень під публікаціями.

Керування коментарями

Коментарі

Пошук коментарів... **Фільтр**

АВТОР	КОМЕНТАР	СТАТТЯ	СТВОРЕНО В	
DavidD	Відповідь на твір не слагася	манора	21.05.26, 09:00	Не підтвердити Видалити
Веселыська Ангеліна	Чарівна архітектура	Чарівний вокзал Києва	31.05.26, 20:26	Не підтвердити Видалити
DavidD	Відповідь на твір я не дозволю додивлятися дораму.....	Дорамки: Феномен, що захопив світ	12.04.26, 13:47	Не підтвердити Видалити
mila	з	Синдром самозванця	16.05.26, 03:12	Не підтвердити Видалити
Веселыська Ангеліна	Відповідь на твір Mi amore.	манора	11.04.26, 08:19	Не підтвердити Видалити
Веселыська Ангеліна	Відповідь на твір Але як знайти на то час :)	Дорамки: Феномен, що захопив світ	04.05.26, 02:09	Не підтвердити Видалити
Веселыська Ангеліна	Відповідь на DavidD Фу, абсурд	Дорамки: Феномен, що захопив світ	04.04.26, 17:51	Не підтвердити Видалити
djfk	to	sample title	11.04.26, 03:51	Не підтвердити Видалити
kg	ooo	Знаєш	06.04.26, 10:19	Не підтвердити Видалити
Веселыська Ангеліна	Відповідь на твір 👍👍	«Четверте крило»	09.05.26, 00:50	Не підтвердити Видалити

Рисунок 3.16 – Сторінка “Коментарі”

Натиснувши кнопку “Користувачі”, Адміністратор може змінити статус користувача, надати права адміністратора, підтвердити акаунт або видалити користувача. Це допомагає керувати доступом до системи. (див. рис. 3.17).

Керування користувачами

Користувачі

Email користувача... **Фільтр**

ІМ'Я	EMAIL	СТВОРЕНО	ВЕРИФІКОВАНИЙ	АДМІН	
Сергій Мухомов	vs222_mysu@student.znu.edu.ua	Jun 8, 2026	☐	☒	Delete
mila	test@gmail.com	May 15, 2026	☒	☒	Delete
Тасоля	borciostan@gmail.com	Jun 5, 2026	☒	☒	Delete
Васильська Ангеліна	angelinavassilisa659@gmail.com	Jun 4, 2026	☒	☒	Delete
qjrk	test1@gmail.com	May 3, 2026	☒	☒	Delete
DavidO	9705@nupat.net	Apr 9, 2026	☒	☒	Delete
lkr	test02@gmail.com	Apr 28, 2026	☒	☒	Delete
Test User	testuser@example.com	Apr 8, 2026	☐	☐	Delete

Рисунок 3.17 – 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 своїх публікаціях.

У розділі «Статті» адміністратор керує всіма публікаціями на сайті (див. рис. 3.18). Тут можна переглянути список статей, відкрити потрібну для редагування або видалити публікацію. Так адміністратор контролює весь контент сайту.

Noteria

Керування статтями

Статті

Назва статті... **Фільтр**

НАЗВА	КАТЕГОРІЯ	СТВОРЕНО	ТЕГИ	
sample title	No categories	9 черв. 2026 р.		Видалити Редагувати статтю
Програмування як мистецтво	ШІ, IT-технології, Мистецтво	3 черв. 2026 р.		Видалити Редагувати статтю
IT-технології: Рушійна сила сучасності	IT-технології, ШІ	2 черв. 2026 р.		Видалити Редагувати статтю
kisalikameovmeow	No categories	31 трав. 2026 р.		Видалити Редагувати статтю
Любов до життя: психологія страхи	Психологія	31 трав. 2026 р.		Видалити Редагувати статтю
Мінималізм	Мистецтво	28 трав. 2026 р.		Видалити Редагувати статтю
asad	Кінематограф	28 трав. 2026 р.		Видалити Редагувати статтю

Рисунок 3.18 – Сторінка “Статті”

У розділі «Категорії» (див. рис. 3.19). Тут можна додавати нові категорії, змінювати їхні назви або видаляти непотрібні. Це потрібно для зручного групування публікацій за темами.

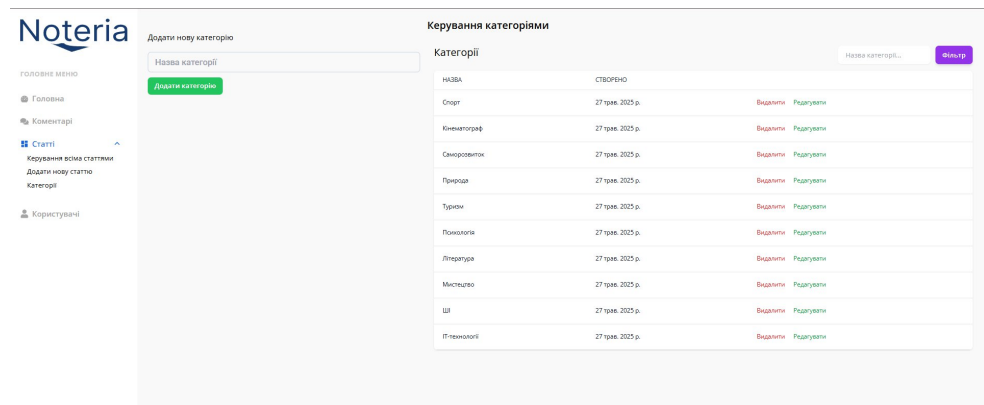


Рисунок 3.19 – Сторінка “ Категорії ”

3.3 Тестування роботи програмного комплексу

Під час розробки веборієнтованого інформаційного додатку на React, Express і MongoDB важливо перевірити, чи всі основні функції працюють коректно. Тестування допомагає знайти помилки, переконатися, що користувач може нормально користуватися сайтом, і оцінити загальну якість готового продукту. У цьому підрозділі описані тест-кейси для ключових сценаріїв роботи системи, а також наведені фактичні результати перевірки, які підтверджують стабільність і зручність розробленого програмного комплексу.

Для перевірки додатку обрано метод ручного функціонального тестування (так зване тестування «чорної скриньки»), за якого кожен сценарій перевіряється з позиції користувача — без аналізу внутрішнього коду, лише за відповідністю фактичної поведінки системи очікуваній. Такий підхід дозволяє оцінити саме ті аспекти, які безпосередньо впливають на роботу з додатком: коректність реєстрації та входу, збереження й відображення даних, реакцію інтерфейсу на дії користувача та зрозумілість повідомлень. Для тестування сформовано набір тест-кейсів, що охоплюють найважливіші сценарії роботи звичайного користувача та адміністратора.

Результати перевірки основних сценаріїв роботи системи методом ручного функціонального тестування наведено у таблиці 3.1.

Результати тестування основних сценаріїв роботи системи

№	Сценарій	Кроки	Очікуваний результат	Фактичний результат	Статус
1	Реєстрація нового користувача	Перехід на сторінку входу → «Зареєструватися», заповнення полів «Ім'я», «Електронна пошта», «Пароль», «Підтвердження пароля», натискання «Зареєструватися»	Обліковий запис створено, дані збережено в колекції users, користувач автоматично авторизований і перенаправлений на головну сторінку	Користувач успішно зареєстрований і перенаправлений на головну сторінку	Пройдено
2	Пошук статей	На сторінці «Блог» ввести ключове слово у поле «Пошук статті», натиснути «Пошук»	Відображено публікації, у заголовку або тексті яких є ключове слово; за відсутності збігів — повідомлення	Показано результати пошуку, що відповідають ключовому слову	Пройдено
3	Додавання коментаря	Авторизований користувач прокручує до блоку коментарів, вводить текст, натискає «Відправити»	Коментар збережено, показано повідомлення; після підтвердження адміністратором коментар відображається під статтею	Коментар відправлено, показано повідомлення про модерацію; після схвалення з'являється під публікацією	Пройдено
4	Зміна налаштувань профілю	«Акаунт» → «Налаштування», зміна імені, пошти або пароля, натискання «Оновити профіль»	Зміни збережено в базі даних, оновлені дані відображаються в профілі, повідомлення «Профіль оновлено»	Зміни налаштувань успішно збережені	Пройдено
5	Перегляд деталей публікації	На головній сторінці або в розділі «Блог» натиснути заголовок чи зображення публікації	Відкривається сторінка статті з повним текстом, датою, автором, рейтингом, блоком коментарів і списком схожих матеріалів	Користувач переходить на сторінку з детальною інформацією про публікацію	Пройдено

Як видно з таблиці 3.1, усі перевірені сценарії завершилися успішно: фактичні результати збіглися з очікуваними, що підтверджує коректність роботи ключових функцій системи. Під час реєстрації перевірялося не лише створення облікового запису, а й оброблення помилок — спроба зареєструватися з уже наявною електронною поштою коректно відхиляється з відповідним повідомленням, а пароль, коротший за встановлену мінімальну довжину, не приймається. Аналогічно під час входу система розрізняє ситуації неправильного пароля та неіснуючої електронної пошти, повертаючи зрозуміле повідомлення замість загальної помилки.

Окрему увагу приділено перевірці розмежування прав доступу: спроба звернутися до адміністративних функцій без відповідних прав або без дійсного токена завершується відмовою на рівні серверної логіки, що підтверджує дієвість механізмів authGuard та adminGuard [17]. Перевірено також роботу з коментарями, зокрема їх модерацію: доданий коментар спочатку отримує статус очікування й відображається під публікацією лише після підтвердження адміністратором.

Окрім функціонального тестування, проведено перевірку зручності використання та сумісності: інтерфейс протестовано в кількох сучасних браузерях (Google Chrome, Mozilla Firefox, Microsoft Edge) і на пристроях із різними розмірами екрана, що підтвердило коректність адаптивного оформлення на базі Tailwind CSS. Орієнтовний час відгуку для типових операцій — завантаження списку публікацій, відкриття окремої статті, додавання коментаря — залишався в межах однієї-двох секунд, що відповідає сформульованим нефункціональним вимогам.

Виявлені під час тестування окремі недоліки — зокрема неточності у відображенні повідомлень про помилки та в поведінці форм за некоректного введення — було усунуто безпосередньо в процесі перевірки. Загалом результати тестування свідчать, що розроблений програмний комплекс працює стабільно, коректно обробляє як типові, так і помилкові сценарії та відповідає поставленим

функціональним і нефункціональним вимогам, а отже, придатний для подальшого використання та вдосконалення.

Висновки до розділу 3

У третьому розділі розглянуто практичну реалізацію веборієнтованого інформаційного додатку. Описано встановлення та налаштування програмного комплексу — розгортання клієнтської частини на React, серверної на Node.js з Express і підключення до хмарної бази даних MongoDB Atlas; підтверджено, що коректне налаштування середовища, змінних оточення та взаємодії між фронтом і бекендом є важливою умовою стабільної роботи системи.

Розглянуто структуру інтерфейсу та основні елементи, з якими взаємодіє користувач: головну сторінку, розділ «Блог», сторінку статті, форми реєстрації й авторизації, профіль, налаштування облікового запису та адміністративну панель. Логічна навігація, єдиний стиль оформлення та зрозуміле розміщення функцій дають змогу швидко знаходити публікації, коментувати, керувати профілем і виконувати адміністративні дії. Проведене тестування основних сценаріїв — реєстрації, пошуку статей, додавання коментарів, зміни налаштувань профілю та перегляду публікації — показало, що ключові функції працюють відповідно до задуманої логіки, а виявлені недоліки усунуто. Розроблений додаток працює коректно, забезпечує основні інформаційні та соціальні можливості й відповідає поставленим завданням кваліфікаційної роботи.

ВИСНОВКИ

У процесі виконання кваліфікаційної роботи бакалавра зосереджено увагу на проектуванні та реалізації веборієнтованого інформаційного додатку для публікації та поширення статей. На початковому етапі проаналізовано схожі інформаційні ресурси, зокрема AIN.UA, ITC.ua та Rubryka, що дозволило визначити основний функціонал майбутньої системи: публікацію постів, пошук матеріалів, коментування, реєстрацію користувачів, особисті профілі та адміністративне керування контентом. За результатами аналізу до розробленого додатку додано функції фільтрації та сортування публікацій за категоріями, оцінювання статей, модерації коментарів і збереження обраних матеріалів у профілі користувача.

Для реалізації програмного комплексу використано сучасний технологічний стек. Клієнтську частину створено на React.js, що забезпечує швидку розробку інтерфейсу та зручну взаємодію з користувачем. Для оформлення застосовано Tailwind CSS, який сприяє створенню адаптивного та структурованого дизайну. Серверну логіку реалізовано на Node.js з фреймворком Express, а для обміну даними між клієнтом і сервером використано Axios. Збереження інформації організовано в хмарній базі даних MongoDB Atlas, де зберігаються дані про користувачів, публікації, коментарі, категорії та онлайн-активність. Для завантаження зображень застосовано сервіс Cloudinary, а для захисту облікових записів — механізм JWT-авторизації.

Проведено аналіз функціональних та нефункціональних вимог до додатку. Для наочного представлення можливостей системи побудовано діаграму варіантів використання, яка відображає дії гостя, авторизованого користувача та адміністратора. Розроблено діаграму класів для опису структури програмного забезпечення та зв'язків між основними сутностями, а також діаграми діяльності, що демонструють послідовність виконання ключових операцій у системі.

Створено базу даних у MongoDB, яка забезпечує надійне збереження та колективний доступ до інформації, необхідної для функціонування додатку. База даних складається з п'яти колекцій: users, posts, comments, postcategories та onlineusers, кожна з яких відповідає за певний аспект роботи системи та зберігає відповідні дані.

Розроблено інтерфейс системи, який включає адміністраторське та користувацьке представлення. Адміністраторський інтерфейс дозволяє керувати публікаціями, категоріями, коментарями та користувачами, переглядати статистику на панелі керування і виконувати інші адміністративні дії. Користувацький інтерфейс призначений для звичайних відвідувачів і зареєстрованих користувачів, які можуть переглядати статті, шукати потрібні матеріали, реєструватися, оновлювати профіль, залишати коментарі, оцінювати публікації та користуватися додатковими можливостями додатку.

Проведено тестування програмного комплексу, зокрема перевірку основних сценаріїв роботи: реєстрації користувача, пошуку статей, додавання коментарів, зміни налаштувань профілю та перегляду деталей публікації. Виявлені в процесі розробки та тестування помилки були усунені, що підтвердило працездатність додатку та коректне виконання поставлених завдань.

Практичне значення роботи полягає у створенні програмного комплексу, який може бути використаний як веборієнтований інформаційний додаток для публікації актуальних матеріалів різної тематики. Розроблений веборієнтований інформаційний додаток забезпечує зручний доступ до контенту, можливість взаємодії між користувачами через коментарі та ефективне адміністрування системи, що робить його придатним для подальшого використання та вдосконалення.

Серед перспективних напрямів подальшого розвитку додатку — упровадження повнотекстового пошуку та системи рекомендацій публікацій на основі вподобань користувачів, додавання сповіщень про нові коментарі й відповіді, а також розширення аналітики на панелі адміністратора.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. React : офіційна документація : вебсайт. URL: <https://react.dev/> (дата звернення: 01.06.2026).
2. Express : офіційна документація фреймворку : вебсайт. URL: <https://expressjs.com/> (дата звернення: 01.06.2026).
3. Axios : офіційна документація HTTP-клієнта : вебсайт. URL: <https://axios-http.com/docs/intro> (дата звернення: 01.06.2026).
4. ІТС.ua : український інформаційний портал про інформаційні технології : вебсайт. URL: <https://its.ua/ua/> (дата звернення: 01.06.2026).
5. MongoDB : офіційна документація : вебсайт. URL: <https://www.mongodb.com/docs/> (дата звернення: 01.06.2026).
6. Mongoose : офіційна документація ODM-бібліотеки : вебсайт. URL: <https://mongoosejs.com/docs/> (дата звернення: 01.06.2026).
7. Tailwind CSS : офіційна документація : вебсайт. URL: <https://tailwindcss.com/docs> (дата звернення: 01.06.2026).
8. React Router : офіційна документація бібліотеки маршрутизації : вебсайт. URL: <https://reactrouter.com/> (дата звернення: 01.06.2026).
9. Rubryka : українське рішення-орієнтоване медіа : вебсайт. URL: <https://rubryka.com/> (дата звернення: 01.06.2026).
10. MongoDB Atlas : офіційна документація хмарного сервісу баз даних : вебсайт. URL: <https://www.mongodb.com/docs/atlas/> (дата звернення: 01.06.2026).
11. Node.js : офіційна документація середовища виконання : вебсайт. URL: <https://nodejs.org/en/docs/> (дата звернення: 01.06.2026).
12. npm : офіційна документація менеджера пакетів : вебсайт. URL: <https://docs.npmjs.com/> (дата звернення: 01.06.2026).
13. АІН : українське інтернет-видання про технології та бізнес : вебсайт. URL: <https://ain.ua/> (дата звернення: 01.06.2026).

- 14.Redux Toolkit : офіційна документація бібліотеки керування станом : вебсайт. URL: <https://redux-toolkit.js.org/> (дата звернення: 01.06.2026).
- 15.TanStack Query (React Query) : офіційна документація : вебсайт. URL: <https://tanstack.com/query/latest> (дата звернення: 01.06.2026).
- 16.DaisyUI : документація компонентної бібліотеки для Tailwind CSS : вебсайт. URL: <https://daisyui.com/> (дата звернення: 01.06.2026).
- 17.JSON Web Token (JWT) : специфікація та довідкові матеріали : вебсайт. URL: <https://jwt.io/> (дата звернення: 01.06.2026).
- 18.Cloudinary : документація сервісу зберігання та обробки медіафайлів : вебсайт. URL: <https://cloudinary.com/documentation> (дата звернення: 01.06.2026).
- 19.TipTap : документація текстового редактора : вебсайт. URL: <https://tiptap.dev/docs> (дата звернення: 01.06.2026).
- 20.bcrypt.js : репозиторій та документація бібліотеки хешування : вебсайт. URL: <https://github.com/dcodeIO/bcrypt.js> (дата звернення: 01.06.2026).
- 21.MDN Web Docs : довідник з вебтехнологій : вебсайт. URL: <https://developer.mozilla.org/> (дата звернення: 01.06.2026).
- 22.ДСТУ 3008:2015. Інформація та документація. Звіти у сфері науки і техніки. Структура та правила оформлювання. Київ : ДП «УкрНДНЦ», 2016. 26 с.
- 23.ДСТУ 8302:2015. Інформація та документація. Бібліографічне посилання. Загальні положення та правила складання. Київ : ДП «УкрНДНЦ», 2016. 17 с.
- 24.ДСТУ ISO/IEC/IEEE 15288:2016. Інженерія систем і програмного забезпечення. Процеси життєвого циклу систем. Київ : ДП «УкрНДНЦ», 2016.
- 25.ДСТУ ISO/IEC 2382:2017. Інформаційні технології. Словник термінів. Київ : ДП «УкрНДНЦ», 2017.
- 26.Banks A., Porcello E. Learning React: Modern Patterns for Developing React Apps. 2nd ed. Sebastopol : O'Reilly Media, 2020. 308 p.

27. Wieruch R. The Road to React: The React.js with Hooks in JavaScript Book. Independently published, 2023. 320 p.
28. Brown E. Web Development with Node and Express: Leveraging the JavaScript Stack. 2nd ed. Sebastopol : O'Reilly Media, 2019. 356 p.
29. Bradshaw S., Brazil E., Chodorow K. MongoDB: The Definitive Guide. 3rd ed. Sebastopol : O'Reilly Media, 2019. 514 p.
30. Flanagan D. JavaScript: The Definitive Guide. 7th ed. Sebastopol : O'Reilly Media, 2020. 706 p.
31. Hahn E. Express in Action: Writing, building, and testing Node.js applications. Shelter Island : Manning Publications, 2016. 256 p.
32. Simpson K. You Don't Know JS Yet: Get Started. 2nd ed. Independently published, 2020. 143 p.

ДОДАТКИ

					ПЗ.КР.Б – 121 – 26 – ПЗ	
						74

Підключення до бази даних

```
import mongoose from "mongoose";

const connectDB = async () => {
  try {
    await mongoose.connect(process.env.DB_URI);
    console.log("Database is connected...");
  } catch (error) {
    console.log(`Error: ${error.message}`);
    process.exit(1);
  }
};

export default connectDB;
```

SERVER.JS

```
import express from "express";
import dotenv from "dotenv";
import connectDB from "./config/db.js";
import cors from "cors";

// Routes
import userRoutes from "./routes/userRoutes.js";
import postRoutes from "./routes/postRoutes.js";
import commentRoutes from "./routes/commentRoutes.js";
import postCategoriesRoutes from "./routes/postCategoriesRoutes.js";
import contactRoutes from "./routes/contactRoutes.js";
import dashboardRoutes from "./routes/dashboard.js";

dotenv.config();
connectDB();
const app = express();

// CORS configuration
app.use(cors({
  origin: 'http://localhost:3000',
  credentials: true,
  methods: ['GET', 'POST', 'PUT', 'DELETE', 'OPTIONS'],
  allowedHeaders: ['Content-Type', 'Authorization']
})));

// Basic middleware
app.use(express.json());
app.use(express.urlencoded({ extended: true }));
```

```

// Simple test route
app.get("/", (req, res) => {
  res.json({ message: "Server is running" });
});

// API routes
app.use("/api/users", userRoutes);
app.use("/api/posts", postRoutes);
app.use("/api/comments", commentRoutes);
app.use("/api/post-categories", postCategoriesRoutes);
app.use("/api/contact", contactRoutes);
app.use("/api/dashboard", dashboardRoutes);

// Test API route
app.get("/api/test", (req, res) => {
  res.json({ message: "Test API route is working" });
});

// Error handling
app.use((err, req, res, next) => {
  console.error("Error:", err);
  res.status(err.statusCode || 500).json({
    message: err.message || "Internal server error",
    stack: process.env.NODE_ENV === "production" ? null : err.stack,
  });
});

// 404 handler
app.use((req, res) => {
  console.log("404 Not Found:", req.originalUrl);
  res.status(404).json({
    message: "Route not found",
    path: req.originalUrl,
  });
});

const PORT = process.env.PORT || 5000;

app.listen(PORT, () => {
  console.log(`Server is running on port ${PORT}`);
});

```

Dashboard.jsx

```
import React, { useEffect } from "react";
import { useQuery } from "@tanstack/react-query";
import { useSelector } from "react-redux";
import { getDashboardStats } from "../../services/index/dashboard";
import { FaUsers, FaNewspaper, FaUserClock } from "react-icons/fa";
import axios from "axios";

const Dashboard = () => {
  const userState = useSelector((state) => state.user);

  const { data: stats, isLoading, refetch } = useQuery({
    queryFn: () => getDashboardStats({ token: userState.userInfo.token }),
    queryKey: ["dashboard-stats"],
    refetchInterval: 30000, // Оновлюємо кожні 30 секунд
  });

  // Оновлюємо статус користувача кожні 30 секунд
  useEffect(() => {
    const updateUserStatus = async () => {
      try {
        await axios.post(
          "/api/users/update-status",
          {},
          {
            headers: {
              Authorization: `Bearer ${userState.userInfo.token}`,
            },
          },
        );
      } catch (error) {
        console.error("Error updating user status:", error);
      }
    };

    const interval = setInterval(() => {
      updateUserStatus();
    }, 30000);

    // Оновлюємо статус при першому завантаженні
    updateUserStatus();

    return () => clearInterval(interval);
  }, [userState.userInfo.token]);
```

```

if (isLoading) {
  return (
    <div className="w-full h-screen flex justify-center items-center">
      <h3 className="text-2xl text-slate-700">Завантаження...</h3>
    </div>
  );
}

return (
  <div className="p-4">
    <h1 className="text-2xl font-semibold mb-6">Статистика</h1>

    <div className="grid grid-cols-1 md:grid-cols-3 gap-6">
      {/* Users Card */}
      <div className="bg-white rounded-lg shadow-md p-6">
        <div className="flex items-center">
          <div className="p-3 rounded-full bg-blue-100 text-blue-600">
            <FaUsers className="w-8 h-8" />
          </div>
          <div className="ml-4">
            <h2 className="text-gray-600 text-sm">Користувачі</h2>
            <p className="text-2xl font-semibold text-gray-800">{stats?.totalUsers || 0}</p>
          </div>
        </div>
      </div>

      {/* Posts Card */}
      <div className="bg-white rounded-lg shadow-md p-6">
        <div className="flex items-center">
          <div className="p-3 rounded-full bg-green-100 text-green-600">
            <FaNewspaper className="w-8 h-8" />
          </div>
          <div className="ml-4">
            <h2 className="text-gray-600 text-sm">Статті</h2>
            <p className="text-2xl font-semibold text-gray-800">{stats?.totalPosts || 0}</p>
          </div>
        </div>
      </div>

      {/* Online Users Card */}
      <div className="bg-white rounded-lg shadow-md p-6">
        <div className="flex items-center">
          <div className="p-3 rounded-full bg-purple-100 text-purple-600">
            <FaUserClock className="w-8 h-8" />
          </div>
          <div className="ml-4">
            <h2 className="text-gray-600 text-sm">Онлайн зараз</h2>

```

```

        <p className="text-2xl font-semibold text-gray-800">{stats?.onlineUsers || 0}</p>
      </div>
    </div>
  </div>
</div>

{/* Таблиця топ-10 постів за рейтингом */}
<div className="mt-10">
  <div className="flex items-center mb-4 gap-2">
    <svg xmlns="http://www.w3.org/2000/svg" className="h-6 w-6 text-yellow-400" fill="currentColor" viewBox="0 0 20 20"><path d="M9.049 2.927c.3-.921 1.603-.921 1.902 0l1.286 3.967a1 1 0 0 0 .95.69h4.175c.969 0 1.371 1.24.588 1.811-3.38 2.455a1 1 0 0 0 -.364 1.118l1.287 3.966c.3.922-.755 1.688-1.54 1.118l-3.38-2.454a1 1 0 0 0 -1.175 0l-3.38 2.454c-.784.57-1.838-.196-1.54-1.118l1.287-3.966a1 1 0 0 0 -.364-1.118l2.05 9.394c-.783-.57-.38-1.811.588-1.811h4.175a1 1 0 0 0 .95-.69l1.286-3.967z"/></svg>
    <h2 className="text-xl font-semibold">Топ-10 постів за рейтингом</h2>
  </div>
  <div className="overflow-x-auto rounded-lg shadow-lg">
    <table className="min-w-full bg-white">
      <thead>
        <tr>
          <th className="px-4 py-2 border-b text-center font-bold">№</th>
          <th className="px-4 py-2 border-b text-center font-bold">Назва
статті</th>
          <th className="px-4 py-2 border-b text-center font-bold">Автор</th>
          <th className="px-4 py-2 border-b text-center font-bold">Середній
рейтинг</th>
          <th className="px-4 py-2 border-b text-center font-bold">К-сть оці-
нок</th>
          <th className="px-4 py-2 border-b text-center font-bold">Пере-
гляди</th>
          <th className="px-4 py-2 border-b text-center font-bold">Дата пу-
блікації</th>
        </tr>
      </thead>
      <tbody>
        {stats?.topRatedPosts?.length > 0 ? (
          stats.topRatedPosts.map((post, idx) => (
            <tr key={post._id} className={"text-center transition-colors
hover:bg-blue-50"}>
              <td className="px-4 py-2 border-b">{idx + 1}</td>
              <td className="px-4 py-2 border-b font-medium text-
left">{post.title}</td>
              <td className="px-4 py-2 border-b">{post.author?.name}</td>
              <td className="px-4 py-2 border-b">{post.averageRating}</td>
              <td className="px-4 py-2 border-b">{post.ratingsCount}</td>
              <td className="px-4 py-2 border-b">{post.views}</td>
            </tr>
          )
        ) : null}
      </tbody>
    </table>
  </div>

```

```

        <td className="px-4 py-2 border-b">{new
Date(post.createdAt).toLocaleDateString()}</td>
      </tr>
    ))
  ) : (
    <tr>
      <td colSpan="7" className="px-4 py-2">Немає даних</td>
    </tr>
  )}
</tbody>
</table>
</div>
</div>
</div>
);
};

export default Dashboard;

```

AdminLayout.jsx

```

import { Outlet, useNavigate } from "react-router-dom";
import Header from "../components/header/Header";
import { useQuery } from "@tanstack/react-query";
import { getUserProfile } from "../../services/index/users";
import { useSelector } from "react-redux";
import { toast } from "react-hot-toast";

const AdminLayout = () => {
  const navigate = useNavigate();
  const userState = useSelector((state) => state.user);

  const {
    data: profileData,
    isLoading: profileIsLoading,
    error: profileError,
  } = useQuery({
    queryFn: () => {
      return getUserProfile({ token: userState.userInfo.token });
    },
    queryKey: ["profile"],
    onSuccess: (data) => {
      if (!data?.admin) {
        navigate("/");
        toast.error("Ви не маєте доступу до адмін панелі");
      }
    },
    onError: (err) => {
      console.log(err);
    }
  });

```

```

        navigate("/");
        toast.error("Ви не маєте доступу до адмін панелі");
    },
  });

  if (profileIsLoading) {
    return (
      <div className="w-full h-screen flex justify-center items-center">
        <h3 className="text-2xl text-slate-700">Завантаження...</h3>
      </div>
    );
  }

  return (
    <div className="flex flex-col h-screen lg:flex-row">
      <Header />
      <main className="bg-[#F9F9F9] flex-1 p-4 lg:p-6">
        <Outlet />
      </main>
    </div>
  );
};

export default AdminLayout;

```

User.jsx

```

import React from "react";
import { useMutation } from "@tanstack/react-query";
import { toast } from "react-hot-toast";
import { useDataTable } from "../../../hooks/useDataTable";
import {
  deleteUser,
  getAllUsers,
  updateProfile,
} from "../../../services/index/users";
import DataTable from "../../../components/DataTable";
import { images, stables } from "../../../constants";

const Users = () => {
  const {
    userState,
    currentPage,
    searchKeyword,
    data: usersData,
    isLoading,
    isFetching,
    isLoadingDeleteData,
  } = useDataTable({
    queryKey: ["users"],
    queryFn: getAllUsers,
    mutationFn: deleteUser,
    mutationOptions: {
      onSuccess: () => toast.success("Пользователь удален"),
    },
    columns: [
      {
        id: 1,
        label: "Имя",
        type: "text",
        width: 200,
      },
      {
        id: 2,
        label: "Фамилия",
        type: "text",
        width: 200,
      },
      {
        id: 3,
        label: "Email",
        type: "text",
        width: 200,
      },
      {
        id: 4,
        label: "Действия",
        type: "button",
        width: 100,
      },
    ],
  });

  return (
    <div>
      <Header />
      <main>
        <DataTable
          data={usersData}
          isLoading={isLoading}
          isFetching={isFetching}
          isLoadingDeleteData={isLoadingDeleteData}
        />
      </main>
    </div>
  );
};

```

```

queryClient,
searchKeywordHandler,
submitSearchKeywordHandler,
deleteDataHandler,
setCurrentPage,
} = useDataTable({
  dataQueryFn: () =>
    getAllUsers(userState.userInfo.token, searchKeyword, currentPage),
  dataQueryKey: "users",
  deleteDataMessage: "User is deleted",
  mutateDeleteFn: ({ slug, token }) => {
    return deleteUser({
      slug,
      token,
    });
  },
});

const { mutate: mutateUpdateUser, isLoading: isLoadingUpdateUser } =
  useMutation({
    mutationFn: ({ isAdmin, isVerified, userId }) => {
      return updateProfile({
        token: userState.userInfo.token,
        userData: { admin: isAdmin, verified: isVerified },
        userId,
      });
    },
    onSuccess: (data) => {
      queryClient.invalidateQueries(["users"]);
      toast.success("User is updated");
    },
    onError: (error) => {
      toast.error(error.message);
      console.log(error);
    },
  });

const handleAdminCheck = (event, userId) => {
  const initialCheckValue = !event.target.checked;
  const user = usersData.data.find(user => user._id === userId);

  if (
    window.confirm("Do you want to change the admin status of this user?")
  ) {
    mutateUpdateUser({
      isAdmin: event.target.checked,
      isVerified: user.verified,
      userId
    });
  }
};

```

```

    } else {
      event.target.checked = initialCheckValue;
    }
  };

const handleVerificationCheck = (event, userId) => {
  const initialCheckValue = !event.target.checked;
  const user = usersData.data.find(user => user._id === userId);

  if (
    window.confirm("Do you want to change the verification status of this user?")
  ) {
    mutateUpdateUser({
      isVerified: event.target.checked,
      isAdmin: user.admin,
      userId
    });
  } else {
    event.target.checked = initialCheckValue;
  }
};

return (
  <DataTable
    pageTitle="Керування користувачами"
    dataListName="Користувачі"
    searchInputPlaceholder="Email користувача..."
    searchKeywordOnSubmitHandler={submitSearchKeywordHandler}
    searchKeywordOnChangeHandler={searchKeywordHandler}
    searchKeyword={searchKeyword}
    tableHeaderTitleList=[
      "Ім'я",
      "Email",
      "Створено",
      "Верифікований",
      "Адмін",
      "",
    ]
    isLoading={isLoading}
    isFetching={isFetching}
    data={usersData?.data}
    setCurrentPage={setCurrentPage}
    currentPage={currentPage}
    headers={usersData?.headers}
    userState={userState}
  >
    {usersData?.data.map((user) => (
      <tr key={user._id}>
        <td className="px-5 py-5 text-sm bg-white border-b border-gray-200">

```

```

<div className="flex items-center">
  <div className="flex-shrink-0">
    <a href="/" className="relative block">
      <img
        src={
          user?.avatar
            ? stables.UPLOAD_FOLDER_BASE_URL + user?.avatar
            : images.userImage
        }
        alt={user.name}
        className="mx-auto object-cover rounded-lg w-10 aspect-square"
      />
    </a>
  </div>
  <div className="ml-3">
    <p className="text-gray-900 whitespace-no-wrap">{user.name}</p>
  </div>
</div>
</td>
<td className="px-5 py-5 text-sm bg-white border-b border-gray-200">
  <p className="text-gray-900 whitespace-no-wrap">{user.email}</p>
</td>
<td className="px-5 py-5 text-sm bg-white border-b border-gray-200">
  <p className="text-gray-900 whitespace-no-wrap">
    {new Date(user.createdAt).toLocaleDateString("en-US", {
      day: "numeric",
      month: "short",
      year: "numeric",
    })}
  </p>
</td>
<td className="px-5 py-5 text-sm bg-white border-b border-gray-200">
  <input
    type="checkbox"
    className="d-checkbox disabled:bg-orange-400 disabled:opacity-100
checked:bg-[url('../public/images/check.png')] bg-cover checked:disabled:bg-none"
    defaultChecked={user.verified}
    onChange={(event) => handleVerificationCheck(event, user._id)}
    disabled={isLoadingUpdateUser}
  />
</td>
<td className="px-5 py-5 text-sm bg-white border-b border-gray-200">
  <input
    type="checkbox"
    className="d-checkbox disabled:bg-orange-400 disabled:opacity-100
checked:bg-[url('../public/images/check.png')] bg-cover checked:disabled:bg-none"
    defaultChecked={user.admin}
    onChange={(event) => handleAdminCheck(event, user._id)}
    disabled={isLoadingUpdateUser}
  />
</td>

```

```

        />
      </td>
      <td className="px-5 py-5 text-sm bg-white border-b border-gray-200 space-
x-5">
        <button
          disabled={isLoadingDeleteData}
          type="button"
          className="text-red-600 hover:text-red-900 disabled:opacity-70
disabled:cursor-not-allowed"
          onClick={() => {
            deleteDataHandler({
              slug: user?._id,
              token: userState.userInfo.token,
            });
          }}
        >
          Delete
        </button>
      </td>
    </tr>
  )}}
</DataTable>
);
};

```

export default Users;

Comment.jsx

```

import React from "react";
import { useMutation } from "@tanstack/react-query";
import { toast } from "react-hot-toast";
import { useDataTable } from "../../hooks/useDataTable";
import {
  deleteComment,
  getAllComments,
  updateComment,
} from "../../services/index/comments";
import DataTable from "../../components/DataTable";
import { images, stables } from "../../constants";
import { Link } from "react-router-dom";

```

```

const Comments = () => {
  const {
    userState,
    currentPage,
    searchKeyword,
    data: commentsData,
    isLoading,
    isFetching,

```

```

    isLoadingDeleteData,
    queryClient,
    searchKeywordHandler,
    submitSearchKeywordHandler,
    deleteDataHandler,
    setCurrentPage,
  } = useDataTable({
    dataQueryFn: () =>
      getAllComments(userState.userInfo.token, searchKeyword, currentPage),
    dataQueryKey: "comments",
    deleteDataMessage: "Коментар видалено",
    mutateDeleteFn: ({ slug, token }) => {
      return deleteComment({
        commentId: slug,
        token,
      });
    },
  });

const {
  mutate: mutateUpdateCommentCheck,
  isLoading: isLoadingUpdateCommentCheck,
} = useMutation({
  mutationFn: ({ token, check, commentId }) => {
    return updateComment({ token, check, commentId });
  },
  onSuccess: (data) => {
    queryClient.invalidateQueries(["comments"]);
    toast.success(
      data?.check ? "Коментар підтверджений" : "Коментар не підтверджений"
    );
  },
  onError: (error) => {
    toast.error(error.message);
    console.log(error);
  },
});

return (
  <DataTable
    pageTitle="Керування коментарями"
    dataListName="Коментарі"
    searchInputPlaceholder="Пошук коментарів..."
    searchKeywordOnSubmitHandler={submitSearchKeywordHandler}
    searchKeywordOnChangeHandler={searchKeywordHandler}
    searchKeyword={searchKeyword}
    tableHeaderTitleList=[
      "Автор",
      "Коментар",

```

```

        "Стаття",
        "Створено в",
        "",
    ]}
    isFetching={isFetching}
    isLoading={isLoading}
    data={commentsData?.data}
    setCurrentPage={setCurrentPage}
    currentPage={currentPage}
    headers={commentsData?.headers}
>
{commentsData?.data.map((comment) => (
    <tr key={comment._id}>
        <td className="px-5 py-5 text-sm bg-white border-b border-gray-200">
            <div className="flex items-center">
                <div className="flex-shrink-0">
                    <a href="/" className="relative block">
                        <img
                            src={
                                comment?.user?.avatar
                                ? stables.UPLOAD_FOLDER_BASE_URL + comment?.user?.avatar
                                : images.userImage
                            }
                            alt={comment?.user?.name}
                            className="mx-auto object-cover rounded-lg w-10 aspect-square"
                        />
                    </a>
                </div>
                <div className="ml-3">
                    <p className="text-gray-900 whitespace-no-wrap">
                        {comment?.user?.name}
                    </p>
                </div>
            </div>
        </td>
        <td className="px-5 py-5 text-sm bg-white border-b border-gray-200">
            {comment?.replyOnUser !== null && (
                <p className="text-gray-900 whitespace-no-wrap">
                    Відповідь на{" "}
                    <Link
                        to={` /blog/${comment?.post?.slug} /#comment-${comment?._id}`}
                        className="text-blue-500"
                    >
                        {comment?.replyOnUser?.name}
                    </Link>
                </p>
            )}
            <p className="text-gray-900 whitespace-no-wrap">{comment?.desc}</p>
        </td>
    </tr>
)}

```

```

<td className="px-5 py-5 text-sm bg-white border-b border-gray-200">
  <p className="text-gray-900 whitespace-no-wrap">
    <Link
      to={` /blog/${comment?.post?.slug}`}
      className="text-blue-500"
    >
      {comment?.post?.title}
    </Link>
  </p>
</td>
<td className="px-5 py-5 text-sm bg-white border-b border-gray-200">
  <p className="text-gray-900 whitespace-no-wrap">
    {new Date(comment.createdAt).toLocaleDateString("uk-UA", {
      day: "2-digit",
      month: "2-digit",
      year: "2-digit",
      hour: "numeric",
      minute: "numeric",
    })}
  </p>
</td>
<td className="px-5 py-5 text-sm bg-white border-b border-gray-200 space-
x-5">
  <button
    disabled={isLoadingDeleteData}
    type="button"
    className={` ${
      comment?.check
        ? "text-yellow-600 hover:text-yellow-900"
        : "text-green-600 hover:text-green-900"
    } disabled:opacity-70 disabled:cursor-not-allowed`}
    onClick={() => {
      mutateUpdateCommentCheck({
        token: userState.userInfo.token,
        check: comment?.check ? false : true,
        commentId: comment._id,
      });
    }}
  >
    {comment?.check ? "Не підтвердити" : "Підтвердити"}
  </button>
  <button
    disabled={isLoadingDeleteData}
    type="button"
    className="text-red-600 hover:text-red-900 disabled:opacity-70
disabled:cursor-not-allowed"
    onClick={() => {
      deleteDataHandler({
        slug: comment?._id,

```

```

        token: userState.userInfo.token,
      });
    }}
  >
    Видалити
  </button>
</td>
{ /* <td className="px-5 py-5 text-sm bg-white border-b border-gray-200">
  <p className="text-gray-900 whitespace-no-wrap">
    {post.categories.length > 0
      ? post.categories.map((category, index) => (
        <span key={category._id} className="text-blue-500">
          {category.title}
          {post.categories.length - 1 !== index && ", "}
        </span>
      ))
      : "No categories"}
    </p>
  </td> */}

{ /* <td className="px-5 py-5 text-sm bg-white border-b border-gray-200">
  <div className="flex gap-x-2">
    {post.tags.length > 0
      ? post.tags.map((tag, index) => (
        <p>
          {tag}
          {post.tags.length - 1 !== index && ", "}
        </p>
      ))
      : "No tags"}
    </div>
  </td>
  <td className="px-5 py-5 text-sm bg-white border-b border-gray-200 space-
x-5">
    <button
      disabled={isLoadingDeleteData}
      type="button"
      className="text-red-600 hover:text-red-900 disabled:opacity-70
disabled:cursor-not-allowed"
      onClick={() => {
        deleteDataHandler({
          slug: post?.slug,
          token: userState.userInfo.token,
        });
      }}
    >
      Delete
    </button>
    <Link

```

```

        to={`admin/posts/manage/edit/${post?.slug}`}
        className="text-green-600 hover:text-green-900"
      >
        Edit
      </Link>
    </td> */}
  </tr>
))}
</DataTable>
);
};

```

export default Comments;

ManagePosts.jsx

```

import { images, stables } from "../../../constants";
import { deletePost, getAllPosts } from "../../../services/index/posts";
import Pagination from "../../../components/Pagination";
import { toast } from "react-hot-toast";
import { Link } from "react-router-dom";
import { useDataTable } from "../../../hooks/useDataTable";
import DataTable from "../../../components/DataTable";

```

```

const ManagePosts = () => {
  const {
    userState,
    currentPage,
    searchKeyword,
    data: postsData,
    isLoading,
    isFetching,
    isLoadingDeleteData,
    queryClient,
    searchKeywordHandler,
    submitSearchKeywordHandler,
    deleteDataHandler,
    setCurrentPage,
  } = useDataTable({
    dataQueryFn: () => getAllPosts(searchKeyword, currentPage),
    dataQueryKey: "posts",
    deleteDataMessage: "Стаття видалена",
    mutateDeleteFn: ({ slug, token }) => {
      return deletePost({
        slug,
        token,
      });
    },
  });
};

```

```

return (
  <DataTable
    pageTitle="Керування статтями"
    dataListName="Статті"
    searchInputPlaceHolder="Назва статті..."
    searchKeywordOnSubmitHandler={submitSearchKeywordHandler}
    searchKeywordOnChangeHandler={searchKeywordHandler}
    searchKeyword={searchKeyword}
    tableHeaderTitleList=[["Назва", "Категорія", "Створено", "Теги", ""]]
    isLoading={isLoading}
    isFetching={isFetching}
    data={postsData?.data}
    setCurrentPage={setCurrentPage}
    currentPage={currentPage}
    headers={postsData?.headers}
    userState={userState}
  >
    {postsData?.data.map((post) => (
      <tr key={post._id}>
        <td className="px-5 py-5 text-sm bg-white border-b border-gray-200">
          <div className="flex items-center">
            <div className="flex-shrink-0">
              <a href="/" className="relative block">
                <img
                  src={
                    post?.photo
                      ? stables.UPLOAD_FOLDER_BASE_URL + post?.photo
                      : images.samplePostImage
                  }
                  alt={post.title}
                  className="mx-auto object-cover rounded-lg w-10 aspect-square"
                />
              </a>
            </div>
            <div className="ml-3">
              <p className="text-gray-900 whitespace-no-wrap">{post.title}</p>
            </div>
          </div>
        </td>
        <td className="px-5 py-5 text-sm bg-white border-b border-gray-200">
          <p className="text-gray-900 whitespace-no-wrap">
            {post.categories.length > 0
              ? post.categories.map((category, index) => (
                  <span key={category._id} className="text-blue-500">
                    {category.title}
                    {post.categories.length - 1 !== index && ", "}
                  </span>
                ))
            }
          </p>
        </td>
      </tr>
    ))
  )

```

```

        : "No categories"}
      </p>
    </td>
    <td className="px-5 py-5 text-sm bg-white border-b border-gray-200">
      <p className="text-gray-900 whitespace-no-wrap">
        {new Date(post.createdAt).toLocaleDateString("uk-UA", {
          day: "numeric",
          month: "short",
          year: "numeric",
        })}
      </p>
    </td>
    <td className="px-5 py-5 text-sm bg-white border-b border-gray-200">
    </td>
    <td className="px-5 py-5 text-sm bg-white border-b border-gray-200 space-
x-5">
      <button
        disabled={isLoadingDeleteData}
        type="button"
        className="text-red-600 hover:text-red-900 disabled:opacity-70
disabled:cursor-not-allowed"
        onClick={() => {
          deleteDataHandler({
            slug: post?.slug,
            token: userState.userInfo.token,
          });
        }}
      >
        Видалити
      </button>
      <Link
        to={` /admin/posts/manage/edit/${post?.slug}`}
        className="text-green-600 hover:text-green-900"
      >
        Редагувати статтю
      </Link>
    </td>
  </tr>
</tbody>
</table>
</DataTable>
);
};

export default ManagePosts;

```

EditCategories.jsx

```
import { useMutation, useQuery, useQueryClient } from "@tanstack/react-query";
```

```

import { useState } from "react";
import { toast } from "react-hot-toast";
import { useParams, useNavigate } from "react-router-dom";
import { useSelector } from "react-redux";
import {
  getSingleCategory,
  updateCategory,
} from "../../services/index/postCategories";

const EditCategories = () => {
  const queryClient = useQueryClient();
  const [categoryTitle, setCategoryTitle] = useState("");
  const navigate = useNavigate();
  const { slug } = useParams();
  const userState = useSelector((state) => state.user);

  const { isLoading, isError } = useQuery({
    queryFn: () => getSingleCategory({ slug }),
    queryKey: ["categories", slug],
    onSuccess: (data) => {
      setCategoryTitle(data?.title);
    },
    refetchOnWindowFocus: false,
  });

  const { mutate: mutateUpdateCategory, isLoading: isLoadingUpdateCategory } =
    useMutation({
      mutationFn: ({ title, slug, token }) => {
        return updateCategory({
          title,
          slug,
          token,
        });
      },
      onSuccess: (data) => {
        queryClient.invalidateQueries(["categories", slug]);
        toast.success("Категорія оновлена");
        navigate(`/admin/categories/manage/edit/${data._id}`, {
          replace: true,
        });
      },
      onError: (error) => {
        toast.error(error.message);
        console.log(error);
      },
    });

  const handleUpdateCategory = () => {
    if (!categoryTitle) return;
  }

```

```

mutateUpdateCategory({
  title: categoryTitle,
  slug,
  token: userState.userInfo.token,
});
});

return (
  <div className="col-span-4 py-8">
    <h4 className="text-lg leading-tight">Оновити категорію</h4>
    <div className="d-form-control w-full mt-6">
      <input
        value={categoryTitle}
        className="d-input d-input-bordered border-slate-300 !outline-slate-300
text-xl font-medium font-roboto text-dark-hard"
        onChange={(e) => setCategoryTitle(e.target.value)}
        placeholder="Назва категорії"
      />
      <button
        disabled={isLoadingUpdateCategory || isLoading || isError}
        type="button"
        onClick={handleUpdateCategory}
        className="w-fit mt-3 bg-green-500 text-white font-semibold rounded-lg
px-4 py-2 disabled:cursor-not-allowed disabled:opacity-70"
      >
        Оновити категорію
      </button>
    </div>
  </div>
);
};

export default EditCategories;

```