ХЕРСОНСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА ДИЗАЙНУ
КАФЕДРА ПРОГРАМНИХ ЗАСОБІВ І ТЕХНОЛОГІЙ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи

бакалавра
(освітньо-кваліфікаційний рівень)

на тему *«Розробка веб-застосунку для інтерактивного обговорення туристичного досвіду»*

Виконала:
здобувачка *4* року навчання
групи *4зПР*
напряму підготовки (спеціальності)
121-Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Колодій Зоряна Олександрівна
(підпис, прізвище та ініціали)
Керівник *Жарікова Марина Віталіївна*
(підпис, прізвище та ініціали)
Рецензент *к.т.н., доцент Григорова А.А.*
(прізвище та ініціали)
Нормоконтролер *Комісаров О. С.*
(підпис, прізвище та ініціали)

Хмельницький – 2026

ХЕРСОНСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРЗВТЕТ

Інститут, факультет, відділення *Інформаційних технологій та дизайну*
Кафедра, циклова комісія *Програмних засобів і технологій*
Освітньо-кваліфікаційний рівень *бакалавр*
Освітньо-професійна підготовка *Програмна інженерія*
(шифр і назва)
Спеціальність *121-Інженерія програмного забезпечення*
(шифр і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри *програмних засобів і технологій*
_____ О.Є. Огнєва
«___» _____ 2026 року

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧА

Колодій Зоряни Олександрівни
(прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) *Розробка веб-застосунку для інтерактивного обговорення туристичного досвіду*

керівник проєкту (роботи) *Жарікова Марина Віталіївна, д.т.н., професор*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від «16» січня 2026 року №84-с
2. Строк подання здобувачем проєкту(роботи) *10.06.2026*
3. Вихідні дані до проєкту (роботи) *літературні та періодичні джерела, матеріали переддипломної практики*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
Аналіз предметної області та існуючих цифрових рішень для обговорення туристичного досвіду, проєктування архітектури веб-застосунку для інтерактивного обговорення туристичного досвіду, розроблення та практична реалізація веб-застосунку для інтерактивного обговорення туристичного досвіду, тестування та впровадження веб-застосунку для інтерактивного обговорення туристичного досвіду

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
5 таблиць, 35 рисунків

6. Консультанти розділів проєкту (роботи)

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

7. Дата видачі завдання *14.01.2026*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів проєкту (роботи)	Примітка
1	Отримання завдання	14.01.2026	Виконано
2	Підбір літератури	17.01.2026-21.01.2026	Виконано
3	Аналіз предметної області	22.01.2026-26.01.2026	Виконано
4	Розробка та обґрунтування завдання	27.01.2026-12.02.2026	Виконано
5	Розробка концептуальної моделі	13.02.2026-19.02.2026	Виконано
6	Моделювання та проєктування системи	20.02.2026-02.03.2026	Виконано
7	Розробка веб-застосунку	03.03.2026-15.04.2026	Виконано
8	Тестування веб-застосунку	16.04.2026-20.04.2026	Виконано
9	Оформлення пояснювальної записки	21.04.2026-02.05.2026	Виконано
10	Захист кваліфікаційної роботи	23.06.2026	Виконано

Здобувач _____ *Колодій З.О.*
(підпис) (прізвище та ініціали)

Керівник проєкту (роботи) _____ *Жарікова М.В.*
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи бакалавра: 85 сторінок, 35 рисунків, 5 таблиць, 1 додаток, 34 джерела.

Мета роботи:

Розробка веб-застосунку, який забезпечує повноцінне інтерактивне середовище для обговорення туристичного досвіду: зберігання та відображення геолокаційних об'єктів, обмін думками та рекомендаціями між користувачами у режимі реального часу, рейтингування локацій та персоналізоване представлення маршрутів.

Об'єкт дослідження:

Процес інтерактивного обміну туристичним досвідом у цифровому просторі за допомогою спеціалізованих веб-платформ.

Предмет дослідження:

Методи, моделі та програмні засоби розроблення веб-застосунків для інтерактивного обговорення туристичного досвіду.

Інструменти та засоби розробки:

Для розробки веб-застосунку було використано сучасний стек технологій React, TypeScript, Node.js, Express.js, MongoDB, Mapbox, Socket.io.

Результати роботи:

Результатом роботи є створений повнофункціональний прототип веб-застосунку, який може бути безпосередньо розгорнуто та використано туристичними спільнотами, організаторами подорожей або як навчальний приклад побудови повностекових застосунків із підтримкою реального часу. Реалізовано функціонал реєстрації, авторизації через JWT, створення та редагування маркерів і маршрутів на інтерактивній карті, систему

коментарів та дискусій у режимі реального часу, систему рейтингування локацій.

Новизна рішення:

Систематизовано вимоги до захищених веб-систем для інтерактивного обговорення туристичного досвіду з підтримкою синхронізації даних через WebSocket, розвинуто підхід до реалізації соціальної взаємодії користувачів (коментарів, обговорень, рейтингів) безпосередньо в геопросторовому контексті на основі Mapbox, вдосконалено механізм авторизації через поєднання JWT та рольової моделі доступу в контексті туристичних платформ.

Практична цінність:

Створено прототип сучасного веб-застосунку, який може бути використаний як платформа для туристичних спільнот, обміну досвідом між мандрівниками, побудови маршрутів та інтерактивної взаємодії користувачів.

КЛЮЧОВІ СЛОВА: *ВЕБ-ЗАСТОСУНОК, КЛІЄНТ-СЕРВЕРНА АРХІТЕКТУРА, REACT, NODE.JS, MONGODB, MAPBOX, JWT, REST API, WEBSOCKET, SOCKET.IO, TYPESCRIPT, ІНТЕРАКТИВНА КАРТА, ТУРИЗМ, ОБГОВОРЕННЯ.*

АНОТАЦІЯ

Кваліфікаційна робота бакалавра складається зі вступу, чотирьох розділів, висновків, списку використаних джерел та додатків.

Роботу присвячено розробці веб-застосунку для інтерактивного обговорення туристичного досвіду.

У першому розділі проведено аналіз предметної області та сучасних цифрових туристичних платформ, визначено основні функціональні обмеження існуючих рішень та сформовано вимоги до розроблюваної системи.

У другому розділі спроєктовано клієнт-серверну архітектуру програмної системи із використанням сучасного стеку технологій React, TypeScript, Node.js, Express.js, MongoDB, Mapbox та Socket.io. Значною перевагою роботи є використання технологій реального часу та інтеграція інтерактивних картографічних сервісів. Розроблено структуру бази даних, REST API, UML-діаграми та ER-діаграму системи.

Третій розділ присвячено реалізації веб-застосунку, який забезпечує реєстрацію та авторизацію користувачів, створення туристичних публікацій, їх інтерактивне обговорення, систему коментарів, підтримку інтерактивних карт, створення туристичних маршрутів, обмін повідомленнями у режимі реального часу, адаптивний користувацький інтерфейс.

У четвертому розділі проведено тестування програмної системи, яке підтвердило коректність роботи функціональних модулів, стабільність REST API, ефективність WebSocket-з’єднань та адаптивність інтерфейсу користувача. Також виконано впровадження веб-застосунку у серверне середовище.

Результатом виконання кваліфікаційної роботи є створення сучасного веб-застосунку для організації туристичної спільноти та інтерактивного обміну туристичним досвідом між користувачами.

ABSTRACT

The bachelor's thesis consists of an introduction, four sections, conclusions, a list of sources used and appendices.

The thesis is devoted to the development of a web application for interactive discussion of tourist experience.

In the first chapter, an analysis of the subject area and modern digital tourist platforms is carried out, the main functional limitations of existing solutions are identified and the requirements for the developed system are formed.

In the second chapter, the client-server architecture of the software system is designed using a modern stack of technologies: React, TypeScript, Node.js, Express.js, MongoDB, Mapbox and Socket.io. A significant advantage of the work is the use of real-time technologies and the integration of interactive cartographic services. The database structure, REST API, UML diagrams and ER diagrams of the system are developed.

The third chapter is devoted to the implementation of a web application that provides user registration and authorization, creation of tourist publications, interactive discussion, a comment system, support for interactive maps, creation of tourist routes, real-time messaging, and an adaptive user interface.

In the fourth chapter, testing of the software system was carried out, which confirmed the correct operation of functional modules, stability of the REST API, efficiency of WebSocket connections, and adaptability of the user interface. The web application was also implemented in the server environment.

The result of the bachelor's thesis is the creation of a modern web application for organizing a tourist community and interactive exchange of tourist experience between users.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

API (Application Programming Interface) – програмний інтерфейс взаємодії між клієнтською та серверною частинами застосунку, зокрема REST API.

JWT (JSON Web Token) – відкритий стандарт для безпечної передачі інформації між учасниками у вигляді підписаного токена.

UI (User Interface) – інтерфейс користувача, що визначає спосіб взаємодії людини з веб-застосунком.

UX (User Experience) – сукупність відчуттів та вражень користувача від взаємодії з програмним продуктом.

HTTP/HTTPS – протокол передачі гіпертексту та його захищений варіант із шифруванням з'єднання TLS/SSL.

CRUD – набір базових операцій з даними: створення (Create), читання (Read), оновлення (Update), видалення (Delete).

MongoDB – документо-орієнтована нереляційна база даних (NoSQL), що застосована в проєкті для зберігання даних.

ODM (Object Document Mapping) – підхід до роботи з документною базою даних через об'єктні моделі.

Express – мінімалістичний фреймворк для побудови серверної частини застосунків на платформі Node.js.

React – JavaScript-бібліотека для побудови компонентного інтерфейсу користувача.

WebSocket – протокол двостороннього обміну даними між клієнтом і сервером у режимі реального часу.

Socket.io – бібліотека для реалізації WebSocket-з'єднань у середовищах Node.js та браузера.

TypeScript – статично типізована надмножина JavaScript, що підвищує надійність і передбачуваність коду.

Mapbox GL JS – бібліотека для відображення інтерактивних карт у браузері на основі технології WebGL.

XSS (Cross-Site Scripting) – тип кібератаки, пов'язаний із впровадженням шкідливих скриптів у веб-сторінки.

Rate-limiting – механізм обмеження кількості запитів від одного джерела для запобігання зловживанням.

Axios – HTTP-клієнт для браузера та Node.js з підтримкою перехоплення запитів та відповідей.

Vite – сучасний інструмент складання фронтенду з підтримкою миттєвого гарячого оновлення модулів.

Mongoose – ODM-бібліотека для роботи з MongoDB у середовищі Node.js.

POI (Point of Interest) – точка інтересу, геолокаційний маркер, що позначає визначне або цікаве місце.

bcrypt – алгоритм односпрямованого хешування паролів із вбудованою підтримкою солі.

ВСТУП

У сучасних умовах стрімкого розвитку інформаційних технологій веб-застосунки стали важливим інструментом комунікації, обміну інформацією та формування цифрових спільнот. Особливо актуальними є платформи, що забезпечують взаємодію користувачів у сфері туризму, оскільки туристична галузь активно використовує цифрові сервіси для планування подорожей, навігації, пошуку туристичних об’єктів та обміну враженнями.

Значне поширення мобільних пристроїв, картографічних сервісів та соціальних платформ призвело до зростання попиту на інтерактивні туристичні веб-застосунки, які дозволяють користувачам не лише переглядати інформацію, а й активно взаємодіяти між собою. Сучасні користувачі очікують від туристичних платформ можливості створення власного контенту, публікації фотографій, побудови маршрутів, обговорення подорожей у режимі реального часу та отримання персоналізованих рекомендацій.

Цифровізація туристичної галузі відкрила нові можливості для мандрівників: сьогодні пошук, планування та оцінювання подорожей усе більше здійснюється через спеціалізовані веб-платформи. Проте переважна більшість наявних сервісів зосереджена на перегляді готового контенту або залишенні статичних відгуків, тоді як потреба в живому, динамічному обговоренні між учасниками туристичної спільноти залишається значною мірою незадоволеною. Саме цей контекст визначає актуальність розробки спеціалізованого веб-застосунку, орієнтованого на інтерактивний обмін туристичним досвідом.

Існуючі цифрові туристичні платформи, такі як Tripadvisor, Google Maps та Wikiloc, мають широкий набір функціональних можливостей, проте не забезпечують повноцінного інтерактивного середовища для

спілкування користувачів і комплексного обговорення туристичного досвіду. У більшості випадків такі системи орієнтовані або на навігацію, або на відгуки, або на побудову маршрутів, що обмежує можливості створення єдиної туристичної спільноти.

У зв'язку з цим виникає необхідність розроблення сучасного веб-застосунку, який поєднуватиме функції соціальної взаємодії, інтерактивного картографування, обміну повідомленнями, створення туристичних маршрутів та публікації мультимедійного контенту.

Актуальність теми полягає у необхідності створення інтерактивної туристичної платформи, що забезпечуватиме ефективну комунікацію між користувачами, підтримку геолокаційних сервісів та сучасний адаптивний інтерфейс. Актуальність зумовлена низкою чинників.

По-перше, зростання мобільного інтернету та поширення смартфонів сформували запит на платформи, де мандрівники можуть оперативно ділитися враженнями та отримувати відповіді від інших учасників спільноти в режимі, наближеному до реального часу.

По-друге, традиційні платформи відгуків страждають від недостатньої інтерактивності: користувач може залишити коментар, але не може брати участь у повноцінній дискусії навколо конкретної локації або маршруту.

По-третє, сучасний технологічний стек – TypeScript, React, Node.js, MongoDB, Socket.io – дозволяє реалізувати подібне рішення з відносно невисокою складністю архітектури, водночас забезпечуючи масштабованість і безпеку.

По-четверте, зростаючі вимоги до захисту персональних даних та безпеки взаємодії вимагають застосування сучасних механізмів автентифікації та авторизації.

Метою дослідження є розробка веб-застосунку, який забезпечує повноцінне інтерактивне середовище для обговорення туристичного

досвіду: зберігання та відображення геолокаційних об'єктів, обмін думками та рекомендаціями між користувачами у режимі реального часу, рейтингування локацій та персоналізоване представлення маршрутів.

Для досягнення зазначеної мети необхідно вирішити такі завдання:

1. Провести систематичний аналіз чинних веб-платформ туристичного спрямування та виявити їхні ключові можливості й обмеження.
2. Дослідити потреби цільової аудиторії та сформулювати функціональні й нефункціональні вимоги до розроблюваної системи.
3. Обрати та обґрунтувати технологічний стек для реалізації клієнтської та серверної частин застосунку.
4. Розробити архітектуру програмного забезпечення з урахуванням принципів масштабованості, модульності та безпеки.
5. Реалізувати захищений механізм автентифікації та управління доступом на основі JWT.
6. Побудувати компонентний інтерфейс користувача з інтерактивною картою, системою коментарів та обговорень у реальному часі.
7. Провести комплексне тестування застосунку та виявити потенційні помилки й вразливості безпеки.
8. Розробити пропозиції щодо подальшого вдосконалення та масштабування системи.

Об'єктом дослідження є процес інтерактивного обміну туристичним досвідом у цифровому просторі за допомогою спеціалізованих веб-платформ.

Предметом дослідження є методи, моделі та програмні засоби розроблення веб-застосунків для інтерактивного обговорення туристичного досвіду.

Методи дослідження, використані в роботі:

– аналіз і синтез – для вивчення існуючих платформ та визначення архітектурних підходів;
– моделювання – для проєктування логіки клієнт-серверної взаємодії та структури бази даних;
– проєктування і програмна реалізація – для побудови прототипу та його поетапного вдосконалення;
– емпіричне тестування – для перевірки коректності роботи реалізованих модулів.

Наукова новизна дослідження полягає в такому:

– вперше систематизовано вимоги до захищених веб-систем для інтерактивного обговорення туристичного досвіду з підтримкою синхронізації даних через WebSocket;

– розвинуто підхід до реалізації соціальної взаємодії користувачів (коментарів, обговорень, рейтингів) безпосередньо в геопросторовому контексті на основі Mapbox;

– вдосконалено механізм авторизації через поєднання JWT та рольової моделі доступу в контексті туристичних платформ.

Практична цінність роботи полягає у створенні повнофункціонального прототипу веб-застосунку, який може бути безпосередньо розгорнуто та використано туристичними спільнотами, організаторами подорожей або як навчальний приклад побудови повностекових застосунків із підтримкою реального часу.

Кваліфікаційна робота складається зі вступу, чотирьох розділів, висновків, списку використаних джерел та додатків.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ ЦИФРОВИХ РІШЕНЬ ДЛЯ ОБГОВОРЕННЯ ТУРИСТИЧНОГО ДОСВІДУ

1.1 Загальна характеристика предметної області

Сфера туризму є однією з найбільш динамічних галузей сучасної цифрової економіки. Активний розвиток веб-технологій, мобільних застосунків та соціальних платформ суттєво змінив підходи до планування подорожей, пошуку туристичних маршрутів та обміну досвідом між мандрівниками. Сучасні користувачі прагнуть не лише отримувати інформацію про туристичні об’єкти, але й брати участь у формуванні туристичного контенту шляхом публікації відгуків, фотографій, рекомендацій та інтерактивного спілкування.

Інтерактивне обговорення туристичного досвіду передбачає створення цифрового середовища, у якому користувачі можуть:

- обмінюватися власними враженнями від подорожей;
- оцінювати туристичні локації;
- формувати маршрути;
- взаємодіяти через коментарі та рейтинги;
- отримувати рекомендації від інших користувачів;
- переглядати мультимедійний контент.

Основними суб’єктами предметної області є:

- туристи;
- адміністратори системи;
- авторизовані користувачі;
- гості платформи;
- туристичні об’єкти;
- туристичні маршрути;
- публікації та коментарі.

Об’єктом автоматизації є процес взаємодії користувачів у середовищі туристичної веб-платформи.

Предметом дослідження є методи та програмні засоби розробки веб-застосунків для організації інтерактивного обговорення туристичного досвіду.

Сучасні туристичні сервіси повинні забезпечувати:

– високу швидкодію;
– адаптивний дизайн;
– інтеграцію з картографічними сервісами;
– підтримку мультимедійного контенту;
– персоналізацію рекомендацій;
– безпечну авторизацію користувачів;
– зручний інтерфейс взаємодії.

Таким чином, розробка веб-застосунку для інтерактивного обговорення туристичного досвіду є актуальним завданням у сфері програмної інженерії.

1.2. Огляд сучасних платформ туристичного спрямування

Ринок цифрових туристичних платформ демонструє значну різноманітність підходів до організації взаємодії між мандрівниками та накопичення геолокаційних даних. Для коректного формулювання вимог до нової системи необхідно проаналізувати найбільш поширені рішення: TripAdvisor, Google Maps та Wikiloc. Кожна з цих платформ реалізує власну модель роботи з туристичним контентом, орієнтується на різні сегменти аудиторії та застосовує відмінні технічні й архітектурні підходи.

Для визначення функціональних вимог до розроблюваної системи було проведено аналіз сучасних популярних туристичних платформ, які забезпечують обмін туристичним досвідом та взаємодію користувачів.

1.2.1. Аналіз платформи TripAdvisor

Tripadvisor є однією з найбільших у світі міжнародних платформ для пошуку інформації про туристичні об’єкти, готелі, ресторани та туристичні маршрути (рис. 1.1).

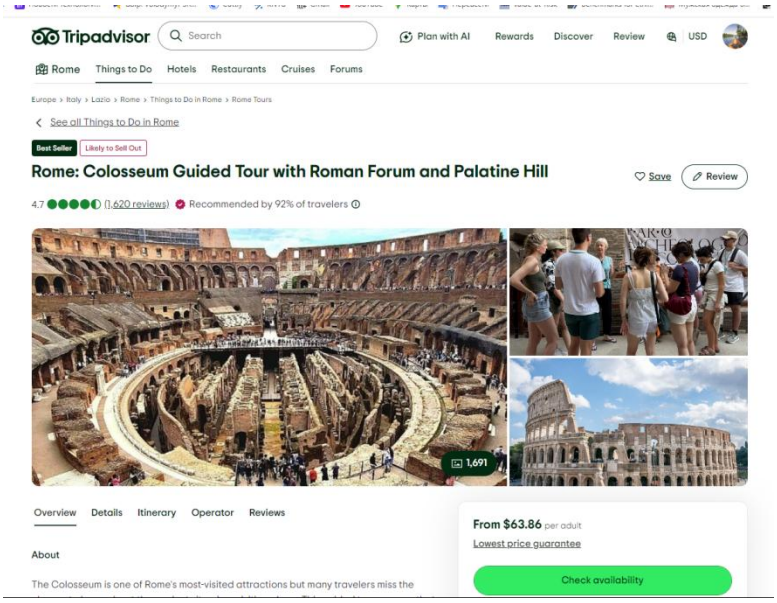

Рисунок 1.1. Інтерфейс платформи TripAdvisor

TripAdvisor є агрегатором туристичного контенту. Платформа побудована за принципом централізованого каталогу, де основним змістом виступають відгуки, рейтинги та фотографії готелів, ресторанів, атракцій і туристичних об'єктів, зібрані від мільйонів зареєстрованих користувачів.

Система передбачає розгалужену структуру фільтрації та пошуку за категоріями, географією, типом закладу та ціновим діапазоном. TripAdvisor має значний обсяг накопиченого контенту, яким забезпечується репрезентативність оцінок, а також розвинену систему ранжування з урахуванням кількох параметрів якості.

Водночас платформа має певні вади: алгоритми формування рейтингів не є прозорими та можуть залежати від рекламних угод; механізм взаємодії користувачів суттєво обмежений, оскільки відсутня повноцінна дискусія щодо конкретних туристичних об'єктів, а система картографічної прив'язки реалізована лише на базовому рівні. Крім того, переважна орієнтація на добре відомі і дуже популярні туристичні напрями

знижує інформативність платформи щодо інших, менш відомих туристичних локацій.

Узагальнимо основні можливості платформи:

– публікація відгуків та оцінок;
– система рейтингів;
– пошук туристичних об’єктів;
– завантаження фотографій;
– бронювання готелів;
– рекомендації для мандрівників.

Перевагами платформи є:

– велика база користувацьких відгуків;
– висока популярність;
– наявність геолокаційного пошуку;
– підтримка мобільних пристроїв.

Недоліки платформи:

– обмежені можливості інтерактивного спілкування;
– непрозорість алгоритми формування рейтингів;
– відсутність повноцінних тематичних обговорень;
– значна кількість рекламного контенту;
– складність модерації недостовірних відгуків.

1.2.2 Аналіз платформи Google Maps

Google Maps є популярним картографічним сервісом, який також використовується як туристична інформаційна платформа (рис. 1.2).

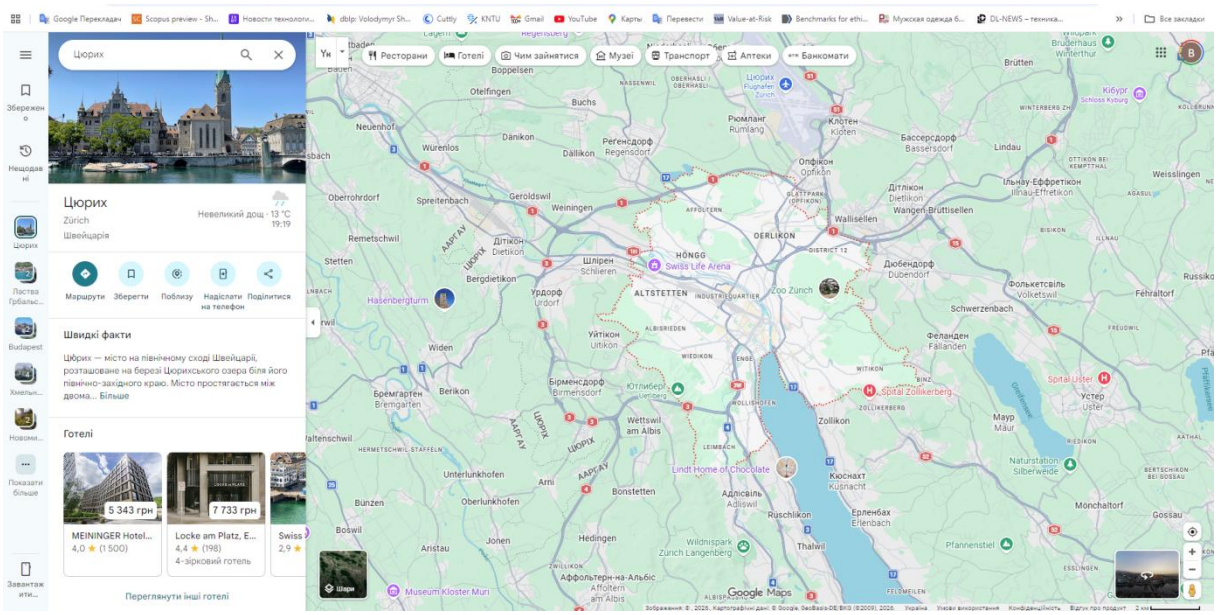

Рисунок 1.2. Інтерфейс сервісу Google Maps

Google Maps реалізує концепцію багатошарової геопросторової платформи, що поєднує навігацію, пошук точок інтересу, відгуки, фотографії та актуальні дані про трафік і графіки роботи закладів. Завдяки інтеграції з екосистемою Google, платформа отримує доступ до широкого спектра поведінкових даних, що дозволяє формувати персоналізовані рекомендації. Серед переваг платформи – висока точність карт, зручна мультимодальна навігація, підтримка панорам Street View та можливість офлайн-використання після попереднього завантаження.

Однак, Google Maps має певні обмеження: надлишок функціональних шарів і рекламних вставок призводить до когнітивного перевантаження користувача, особливо на мобільних пристроях; відгуки не проходять ефективної модерації на достовірність; можливості будь-якого обговорення між користувачами фактично відсутні. Для малих та некомерційних проєктів доступ до Google Maps API може виявитися економічно невигідним через платну модель із розрахунком за кількість запитів.

Отже, основні можливості сервісу:

– навігація та побудова маршрутів;
– перегляд туристичних об’єктів;
– користувацькі відгуки;
– фотографії локацій;
– інтеграція з мобільними сервісами;
– геолокаційний пошук.

Переваги:

– точні картографічні дані;
– висока швидкість роботи;
– інтеграція з іншими сервісами Google;
– зручний мобільний інтерфейс.

Недоліки:

– недостатній рівень соціальної взаємодії;
– відсутність тематичних туристичних спільнот;
– обмежені можливості персоналізації контенту;
– орієнтація переважно на навігацію, а не обговорення досвіду.

1.2.3. Аналіз платформи Wikiloc

Wikiloc є платформою для пошуку та публікації туристичних маршрутів, орієнтованою переважно на активний туризм (рис. 1.3).

Wikiloc спеціалізовано на обміні маршрутами активного відпочинку: пішохідними, велосипедними, гірськолижними тощо. На відміну від двох попередніх сервісів, Wikiloc орієнтований на спільноту активних мандрівників і дозволяє завантажувати, переглядати та завантажувати GPS-треки у стандартних форматах. Система надає детальні профілі маршрутів з висотними графіками, фотографіями та відгуками.

До переваг платформи належить фокусована функціональність для активного туризму та широкий вибір маршрутів.

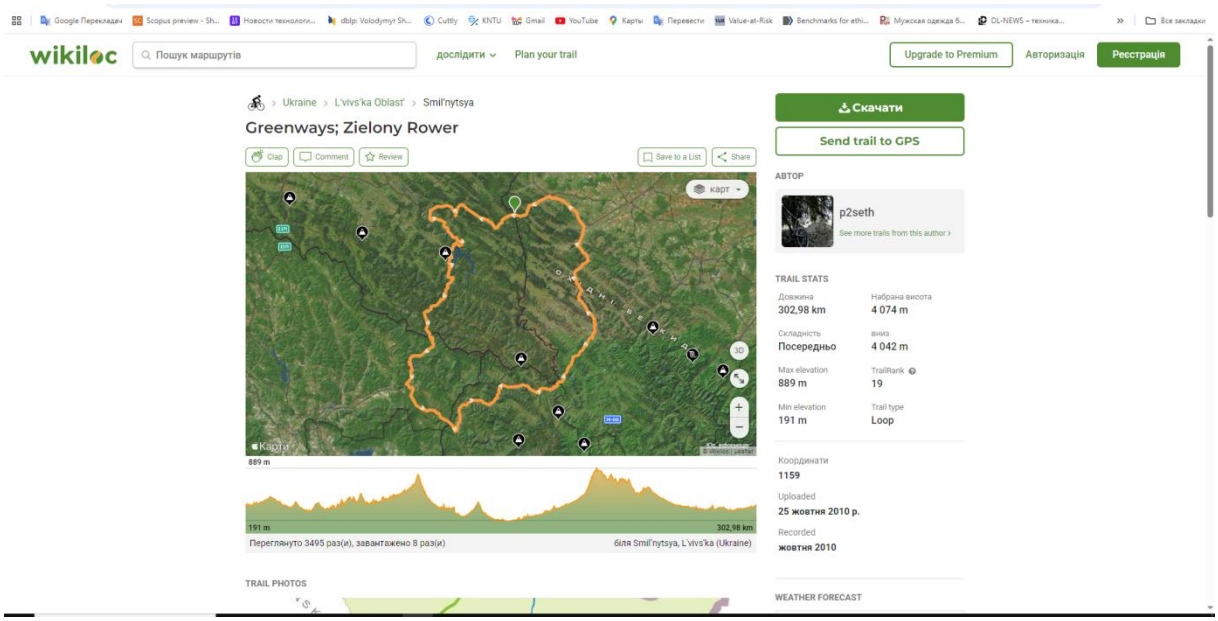

Рисунок 1.3. Інтерфейс платформи Wikiloc

Однак Wikiloc обмежений у соціальному аспекті: обговорення маршрутів відбувається у форматі коментарів без підтримки тематичних дискусій, а інтерактивність обміну думками між учасниками залишається мінімальною. Крім того, значна частина функціонала доступна лише за передплатою Premium.

Основні функції:

– створення маршрутів;
– GPS-навігація;
– публікація треків;
– обмін фотографіями;
– пошук маршрутів за категоріями;
– оцінювання маршрутів.

Переваги:

– спеціалізація на туристичних маршрутах;
– підтримка GPS-трекінгу;
– наявність активної спільноти користувачів;
– інтеграція з мобільними пристроями.

Недоліки:

– вузька спеціалізація;
– недостатньо розвинуті соціальні функції;

– складний інтерфейс для нових користувачів;
– обмежені можливості мультимедійної взаємодії.

1.3. Порівняльний аналіз існуючих рішень

Для визначення функціональних недоліків сучасних туристичних платформ доцільно провести їх порівняльний аналіз, результати якого представлені в табл. 1.1.

Таблиця 1.1. Порівняння функціональних можливостей платформ

Функція	**TripAdvisor**	**Google Maps**	**Wikiloc**
Відгуки користувачів	+	+	+
Коментарі	+	Частково	Частково
Соціальна взаємодія	Частково	Частково	Частково
Побудова маршрутів	–	+	+
Інтерактивне обговорення	–	–	Частково
Геолокація	+	+	+
Мультимедійний контент	+	+	+
Форумне спілкування	–	–	–
Персоналізовані рекомендації	Частково	+	Частково

Порівняльне дослідження означених платформ дає змогу визначити спільну для всіх трьох рішень проблему: жодна з них не забезпечує повноцінного середовища для інтерактивного обговорення туристичного досвіду безпосередньо в геопросторовому контексті. Тобто, розглянуті нами сучасні туристичні платформи мають значні функціональні можливості, однак не забезпечують повноцінного інтерактивного середовища для спілкування користувачів та обговорення туристичного досвіду.

TripAdvisor концентрується на відгуках, але не на діалозі; Google Maps надає потужні інструменти навігації, але соціальна взаємодія

реалізована поверхово, зокрема, відсутні функції інтерактивного обговорення певних туристичних локацій у вигляді чату, форуму, або тематичної дискусії; Wikiloc обслуговує вузьку нішу активного туризму без широкого охоплення локального контексту.

Ця прогалина відкриває обґрунтовану потребу у новій системі, що поєднала б картографічну візуалізацію, геопросторові маркери та розвинений функціонал обговорень із синхронізацією в реальному часі.

Результати проведеного аналізу підтверджують обґрунтованість обраного напряму дослідження. Розробка застосунку, що фокусується саме на інтерактивному обговоренні туристичного досвіду з прив'язкою до карти та оновленням у реальному часі, дозволить закрити наявну прогалину на ринку цифрових туристичних сервісів.

1.4. Виявлення функціональних обмежень існуючих систем

На основі проведеного аналізу були визначені основні недоліки сучасних туристичних платформ:

1. Недостатній рівень інтерактивної взаємодії між користувачами.
2. Відсутність повноцінної системи тематичних обговорень.
3. Обмежені можливості персоналізації туристичного контенту.
4. Недостатня інтеграція соціальних механізмів комунікації.
5. Складність структурування великої кількості відгуків.
6. Обмежені засоби рекомендацій на основі користувацького досвіду.
7. Відсутність єдиного середовища для поєднання маршрутів, відгуків, фотографій та спільнот користувачів.

У зв’язку з цим виникає необхідність розробки веб-застосунку, який поєднуватиме:

– соціальні функції;
– туристичну інформацію;

– інтерактивні обговорення;
– мультимедійний контент;
– систему рекомендацій;
– зручні механізми взаємодії користувачів.

1.5. Аналіз потреб цільової аудиторії та проблематика

Успішність цифрової платформи визначається насамперед тим, наскільки точно вона відповідає реальним потребам своєї цільової аудиторії. Для розроблюваного застосунку можна виділити декілька ключових сегментів користувачів, кожен з яких формує власний набір вимог та сценаріїв використання.

Перший і найчисельніший сегмент складають незалежні мандрівники у віці від 18 до 45 років, які активно користуються мобільними пристроями та орієнтуються на самостійне планування маршрутів. Для них критично важливим є доступ до актуального досвіду реальних людей, а не відполірованих рекламних описів. Вони шукають відповіді на конкретні запитання: чи безпечний цей маршрут уночі? Чи варто відвідувати ресторан у будній день? Які місця поруч, про які мало хто знає? Саме такий тип комунікації – запитання та відповіді, підказки та попередження в контексті конкретної локації – складає основу інтерактивного обговорення.

Другий сегмент – «місцеві дослідники»: мешканці міст і регіонів, які використовують цифрові платформи не для подорожей у класичному сенсі, а для відкриття цікавих місць у власному оточенні. Для них пріоритетними є швидкість знаходження актуальної інформації, фільтрація рекомендацій за тематикою та можливість ділитися власними знахідками з такими ж допитливими людьми.

Третій сегмент – туристичні фахівці та гіди, які можуть використовувати платформу як інструмент для представлення маршрутів і відповідей на запитання потенційних клієнтів. Для цієї аудиторії важлива

можливість детального опису об'єктів і підтримки діалогу з користувачами у відкритому форматі.

Окрему категорію становлять користувачі з нестабільним або повільним інтернет-з'єднанням: мандрівники в гірських або сільських районах, де основна частина функціоналу повинна зберігатися в кешованому вигляді для забезпечення автономної роботи.

Визначивши категорії користувачів, доцільно визначити та узагальнити їх потреби (рис. 1.4).

Рисунок 1.4. Інтерфейс платформи Wikiloc

Аналіз наявних платформ у поєднанні з потребами перелічених сегментів дозволяє виділити типові проблеми, що перешкоджають ефективній взаємодії сучасних туристичних сервісів:

– *відсутність справжнього діалогу*. Більшість платформ дозволяють залишати відгуки, але не передбачають структурованого обговорення: неможливо поставити уточнювальне запитання автору відгуку або розгорнути дискусію навколо певної особливості локації;

– *централізований контроль над контентом*. Алгоритми ранжування нерідко формують непрозорі переліки рекомендацій, де комерційні об'єкти мають перевагу над нішевими або некомерційними локаціями;

– *слабка підтримка локального контенту*. Маловідомі місця не мають достатньої кількості відгуків для потрапляння до пошукових результатів, що унеможливлює їх відкриття новими мандрівниками;

– *відсутність синхронних взаємодій*. Поточні рішення не надають можливості отримати відповідь або підказку від інших мандрівників у режимі, наближеному до реального часу, під час активної подорожі;

– *перевантаженість інтерфейсів*. Надмір функцій, рекламних блоків і нерелевантної інформації знижує зручність і швидкість доступу до потрібного контенту.

Ці проблеми обґрунтовують концептуальну основу розроблюваного застосунку: система має забезпечити відкрите та швидке середовище для обміну туристичним досвідом у форматі прив'язаних до карти дискусій, де кожна геолокаційна точка може стати предметом живого обговорення між учасниками спільноти.

1.6. Формування вимог до розроблюваної системи

На підставі проведеного аналізу аналогів та потреб цільової аудиторії визначено комплекс функціональних і нефункціональних вимог до веб-застосунку для інтерактивного обговорення туристичного досвіду.

Значущість кожної з функціональних вимог визначена на основі аналізу сценаріїв використання та є підставою для визначення пріоритетів у реалізації. Проведений аналіз сценаріїв використання дозволив побудувати діаграму варіантів використання (рис. 1.5), яка відображає взаємодію користувачів із функціональними можливостями системи.

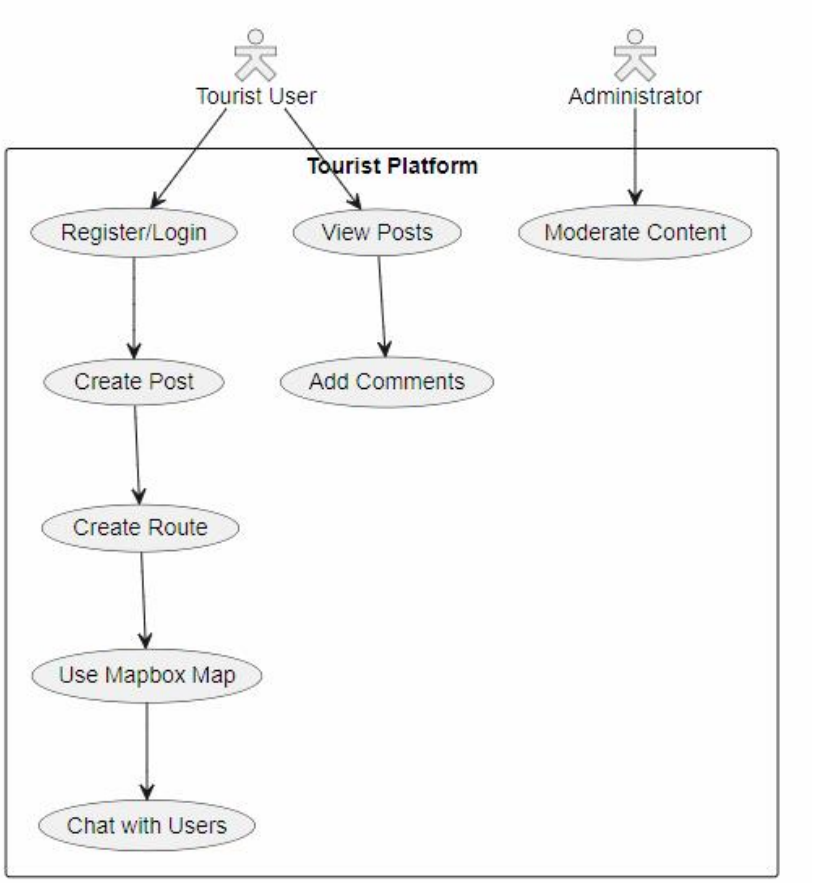

Рисунок 1.5. Діаграма варіантів використання

Аналіз сценаріїв використання показує, що реєстрація та автентифікація, інтерактивна карта й модуль обговорень становлять критичне ядро системи з найвищим пріоритетом для їх впровадження.

1.6.1. Функціональні вимоги

Функціональні вимоги охоплюють усі ключові сценарії взаємодії користувача із системою. Застосунок повинен реалізовувати повний цикл роботи з обліковими записами: реєстрацію за електронною поштою та паролем, авторизацію з видачею JWT-токена, відновлення пароля, а також управління профілем. Рольова модель має передбачати щонайменше два рівні доступу – звичайний користувач та адміністратор – з чітко визначеними дозволеними діями для кожної ролі.

Центральним функціональним компонентом є інтерактивна карта, на якій відображаються всі створені маркери з підтримкою масштабування, навігації та шарового перемикання (звичайний вигляд, супутникова зйомка, топографічний шар). Кожен авторизований користувач може додати маркер подвійним кліком на карті, вказавши назву, опис, категорію, рейтинг та фотографії. Маркери зберігаються в базі даних із підтримкою геолокаційної індексації та можуть фільтруватися за категорією, рейтингом і географічним радіусом.

Ключовою відмінністю від існуючих рішень є повноцінний модуль обговорень. Кожен маркер містить розділ коментарів, де користувачі можуть залишати запитання, відповіді, поради та враження. Система підтримує ієрархічну структуру коментарів (відповідь на конкретний коментар), лайки та позначку «корисно». Нові коментарі та оновлення маркерів відображаються у всіх підключених клієнтів без перезавантаження сторінки завдяки WebSocket-синхронізації.

Серед додаткових функціональних вимог:

– CRUD-операції для маркерів, коментарів і маршрутів – виключно в межах власного контенту для звичайного користувача;
– пошук за назвою, тегами, категорією та географічним положенням;
– рейтингування маркерів і формування топ-підбірок за регіонами;
– система сповіщень про нові коментарі під власними маркерами;
– особистий кабінет з переліком створених маркерів, статистикою активності та налаштуваннями конфіденційності;
– валідація всіх форм із відображенням інформативних повідомлень про помилки;
– захист від дублювання записів і спаму через rate-limiting.

1.6.2. Нефункціональні вимоги

Нефункціональні вимоги визначають якісні характеристики системи. Система повинна відповідати таким характеристикам:

– адаптивність інтерфейсу;

– висока швидкодія;
– безпечне зберігання даних;
– масштабованість;
– кросплатформеність;
– зручність користування;
– відмовостійкість.

В аспекті безпеки передбачено: шифрування HTTP-з'єднань через HTTPS, хешування паролів алгоритмом bcrypt із динамічною сіллю, авторизацію через JWT з контрольованим терміном дії, захист від основних вразливостей OWASP Top 10 (ін'єкції, XSS, CSRF), санітизацію вхідних даних та обмеження доступу до внутрішніх API.

Вимоги масштабованості передбачають модульну архітектуру з підтримкою горизонтального масштабування, асинхронну обробку запитів через Node.js Event Loop та можливість введення кешування у майбутньому.

Доступність застосунку охоплює адаптивний дизайн для мобільних і десктопних пристроїв, кросбраузерну сумісність (Chrome, Firefox, Safari, Edge), підтримку базового офлайн-режиму через кешування карти, а також відповідність рекомендаціям WCAG для доступності інтерфейсу.

1.7. Постановка задачі

У результаті проведеного аналізу визначено, що існуючі туристичні платформи не забезпечують комплексної підтримки інтерактивного обговорення туристичного досвіду між користувачами.

Отже, основним завданням кваліфікаційної роботи є розробка веб-застосунку, який забезпечить:

– створення туристичних публікацій;
– інтерактивне обговорення;
– формування туристичних маршрутів;
– мультимедійну взаємодію;
– персоналізовані рекомендації;
– зручний сучасний інтерфейс користувача.

Результатом роботи має стати сучасний веб-застосунок, орієнтований на створення активної туристичної спільноти та покращення взаємодії між мандрівниками.

Висновки до розділу

У першому розділі проведено комплексний огляд актуальних веб-платформ туристичного спрямування: TripAdvisor, Google Maps та Wikiloc. Аналіз засвідчив, що кожна з розглянутих систем реалізує власний підхід до організації туристичного контенту, однак жодна з них не забезпечує повноцінного інтерактивного обговорення між користувачами безпосередньо в геопросторовому контексті.

Виявлені типові проблеми – відсутність справжнього діалогу між мандрівниками, централізований непрозорий контроль над контентом, слабке охоплення локальних та маловідомих об'єктів, відсутність синхронних взаємодій і перевантаженість інтерфейсів – обґрунтовують потребу у створенні нового рішення.

Аналіз цільової аудиторії виявив стійкий запит на платформу, яка поєднує зручну картографічну візуалізацію, відкрите середовище для дискусій і оновлення даних у реальному часі. Сформульовані функціональні та нефункціональні вимоги закладають основу для проєктування системи з акцентом на безпеку, масштабованість та інтерактивність.

РОЗДІЛ 2. ПРОЄКТУВАННЯ АРХІТЕКТУРИ ВЕБ-ЗАСТОСУНКУ ДЛЯ ІНТЕРАКТИВНОГО ОБГОВОРЕННЯ ТУРИСТИЧНОГО ДОСВІДУ

2.1. Архітектура програмної системи

Архітектурну основу розроблюваного веб-застосунку становить класична клієнт-серверна модель із розширенням у вигляді WebSocket-каналу для синхронізації подій у реальному часі.

Загальна пропонована архітектура системи представлена на рис. 2.1. Така організація забезпечує чітке розмежування відповідальності: клієнтська частина відповідає за відображення інтерфейсу та обробку подій, серверна – за виконання бізнес-логіки, управління даними та безпеку.

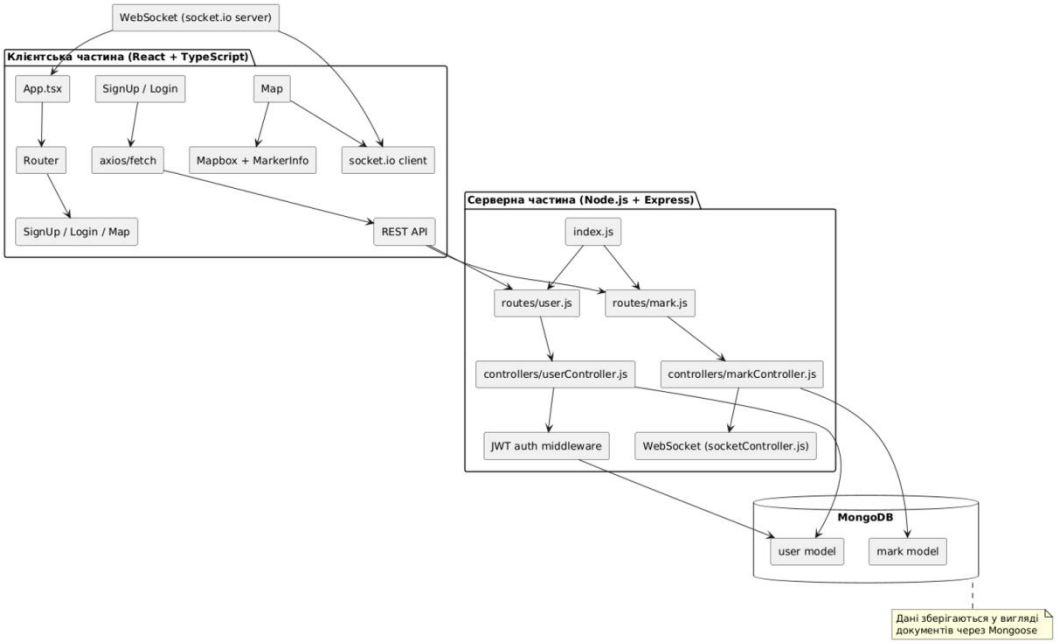

Рисунок 2.1. Загальна архітектура системи

Клієнтська частина реалізована на базі бібліотеки React у поєднанні з TypeScript. Компонентний підхід React дозволяє організувати інтерфейс у вигляді ізольованих, повторно використовуваних блоків, кожен з яких несе

відповідальність за конкретну функцію: відображення карти, форм введення, модальних вікон, панелей коментарів тощо. TypeScript надає статичну типізацію, що мінімізує категорію помилок, пов'язаних із неузгодженістю типів даних між компонентами та при взаємодії з серверним API. Клієнт не зберігає критичних даних і не виконує бізнес-логіки – вся обробка здійснюється на сервері, що унеможливлює маніпуляцію з ключовими функціями через браузерний код.

Серверна частина побудована на платформі Node.js із використанням фреймворку Express. Структура серверного коду побудована за принципом модульного розмежування відповідальностей: маршрути (routes) приймають HTTP-запити та передають їх до контролерів, контролери (controllers) реалізують бізнес-логіку та викликають сервіси, що взаємодіють із моделями даних. Окремим шаром виступають middleware-функції, які забезпечують наскрізні завдання: перевірку автентичності токенів, логування, валідацію вхідних даних, обмеження швидкості запитів та обробку помилок. Для збереження даних застосовано документо-орієнтовану базу MongoDB, доступ до якої організовано через бібліотеку Mongoose з підтримкою схем, валідації та геоіндексації.

Взаємодія клієнтської, серверної частини та бази даних представлена на рис. 2.2

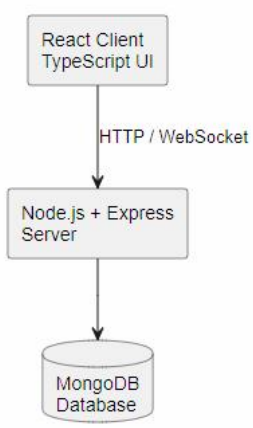

Рисунок 2.2. Взаємодія компонентів клієнт-серверної архітектури

Взаємодія між клієнтом і сервером здійснюється через два рівнозначних канали комунікації. Перший – REST API на основі HTTP-запитів типу GET, POST, PUT, DELETE – використовується для операцій із чітко визначеним результатом: реєстрація, авторизація, збереження маркерів, отримання списків коментарів. Кожна така операція підтверджується HTTP-відповіддю із відповідним кодом статусу та JSON-тілом. Другий канал – WebSocket, реалізований через бібліотеку Socket.io – забезпечує трансляцію подій між усіма підключеними клієнтами: поява нового маркера, публікація коментаря, оновлення рейтингу відображаються у всіх активних сесіях без перезавантаження сторінки. Після збереження нового ресурсу через REST API сервер додатково генерує WebSocket-подію, яка передається підключеним клієнтам для оновлення локального стану карти або списку обговорень.

У рамках описаної архітектури реалізовано принцип розмежування доступу на рівні серверних маршрутів: усі операції зі зміни або видалення ресурсів вимагають пред'явлення дійсного JWT-токена та перевірки відповідності ідентифікатора автора ресурсу ідентифікатору поточного користувача. Ця перевірка виконується у middleware до передачі запиту контролеру, що забезпечує єдину точку контролю доступу.

Такий підхід дозволяє реалізувати збалансовану архітектуру, де REST API забезпечує структурованість і контрольованість операцій зі збереженням даних, а WebSocket – високу реактивність інтерфейсу при колективній роботі з картою та обговореннями. Завдяки модульній структурі системи обидва канали можуть незалежно масштабуватися та розширюватися без значних змін у базовій логіці застосунку.

Діаграма компонентів демонструє архітектуру програмної системи (рис. 2.3).

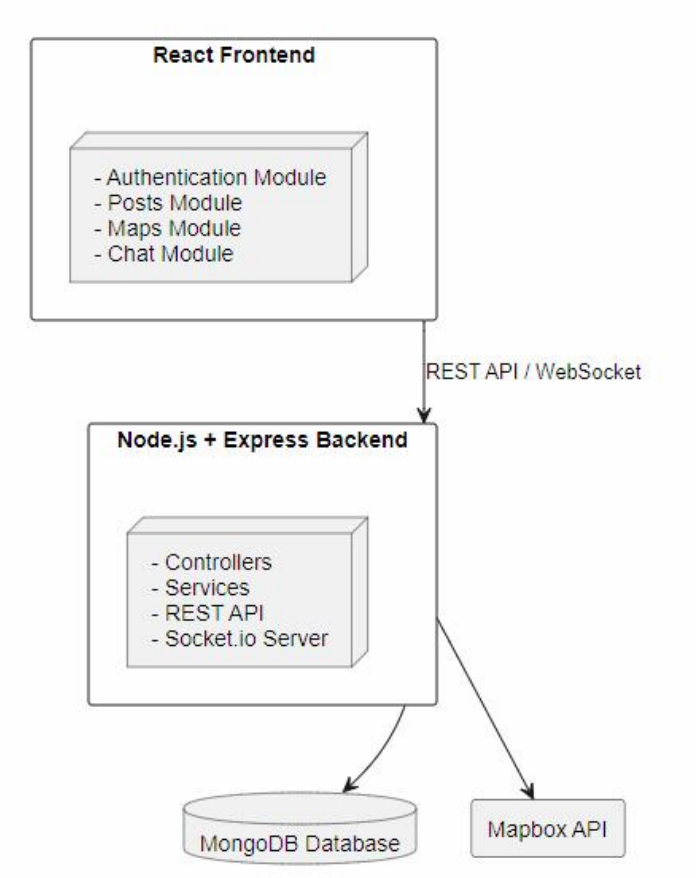

Рисунок 2.3. Діаграма компонентів

Отже, для реалізації веб-застосунку було обрано клієнт-серверну архітектуру, яка забезпечує розподіл функціональності між клієнтською та серверною частинами системи. Такий підхід дозволяє підвищити масштабованість, спростити супровід програмного забезпечення та забезпечити незалежний розвиток окремих компонентів системи.

У розроблюваному застосунку:

– клієнтська частина відповідає за взаємодію з користувачем;
– серверна частина забезпечує обробку запитів;
– база даних використовується для зберігання інформації;
– картографічний сервіс забезпечує роботу з геолокацією;
– WebSocket-з’єднання реалізують обмін повідомленнями в реальному часі.

Діаграма розгортання (рис. 2.4) демонструє фізичне розміщення компонентів системи.

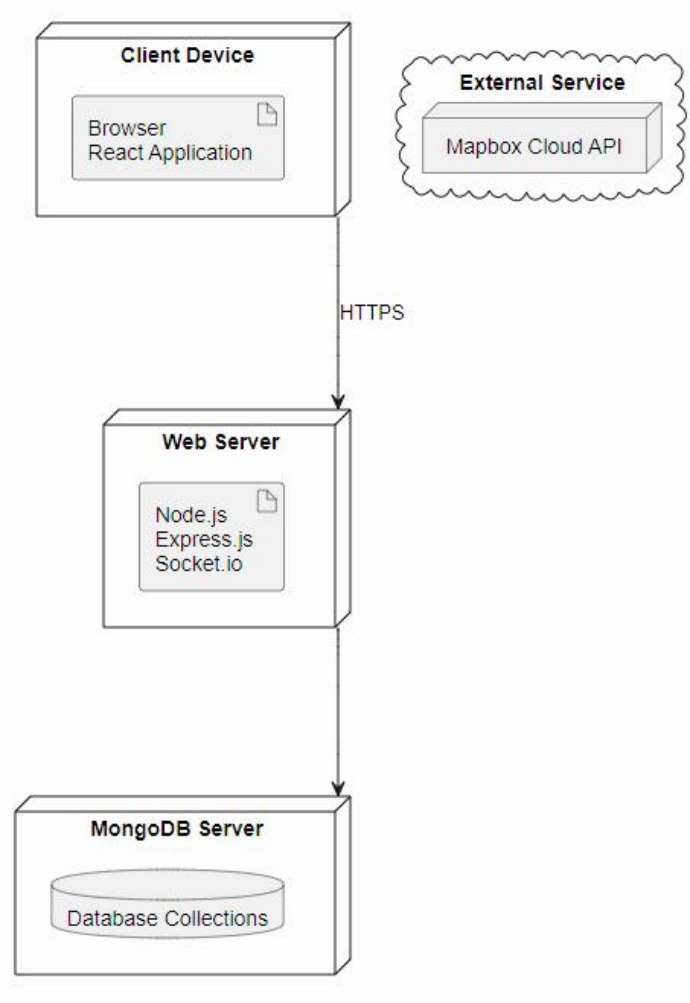

Рисунок 2.4. Діаграма розгортання

Основними перевагами клієнт-серверної архітектури є:

– централізоване зберігання даних;
– можливість масштабування;
– розподіл навантаження;
– підтримка багатокористувацької роботи;
– підвищення безпеки системи.

2.2. Технологічний стек та інструменти розробки

Вибір технологічного стеку для реалізації застосунку здійснювався з урахуванням функціональних вимог системи, необхідності підтримки реального часу, вимог безпеки, а також зрілості та поширеності відповідних рішень у середовищі веб-розробки.

Процес вибору технологій та інструментів реалізації проілюстровано на рис. 2.5.

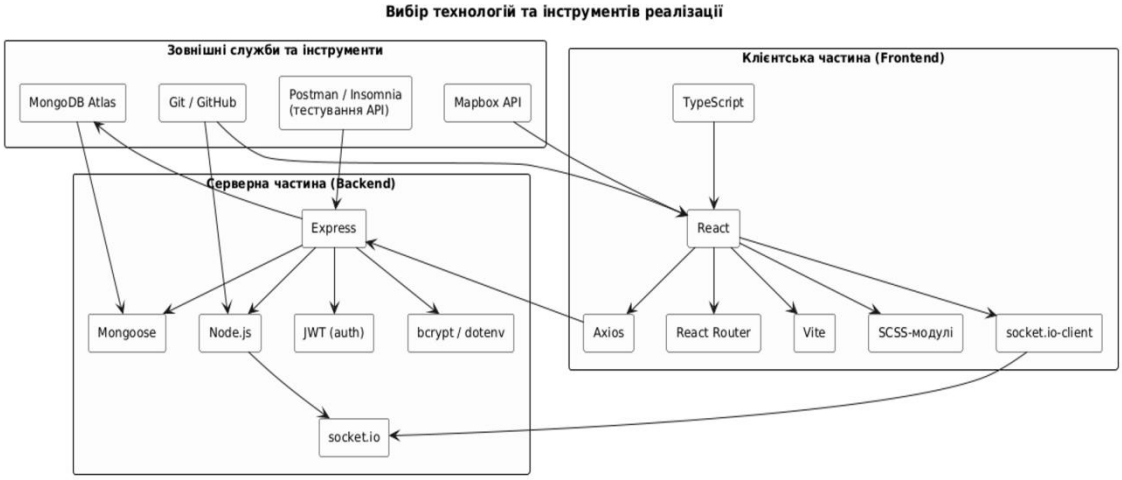

Рисунок 2.5. Вибір технологій та інструментів реалізації

Для складання фронтенду застосовано Vite – сучасний інструмент збірки на основі ESBuild, що забезпечує значно швидший цикл гарячого оновлення порівняно з традиційним Webpack. На серверній стороні для роботи з HTTP-запитами використовується Axios, що надає зручний інтерфейс для перехоплення помилок, автоматичного додавання заголовків авторизації та кешування відповідей.

Для розробки веб-застосунку було обрано сучасний стек технологій JavaScript/TypeScript, який забезпечує високу продуктивність, зручність розробки та підтримку масштабованості.

2.2.1. React

React використовується для побудови компонентного інтерфейсу. Ключовою перевагою є декларативний підхід до опису UI та ефективне оновлення DOM через механізм Virtual DOM. У застосунку React-компоненти організовано за функціональним призначенням: Navbar, MapComponent, MarkerPopup, DiscussionPanel, AddMarkerForm, CommentThread. Хуки useContext та useReducer забезпечують глобальний стан автентифікації та списку маркерів, тоді як useEffect керує підпискою на WebSocket-події та запитами до API при завантаженні компонентів.

React використовується для створення клієнтської частини веб-застосунку.

Основні переваги React:

– компонентний підхід;
– висока швидкодія завдяки Virtual DOM;
– повторне використання компонентів;
– велика екосистема бібліотек;
– підтримка SPA-архітектури.

React забезпечує створення інтерактивного користувацького інтерфейсу та динамічного оновлення даних без перезавантаження сторінки.

2.2.2. TypeScript

TypeScript обраний як мова програмування для клієнтської частини з кількох причин. По-перше, статична типізація дозволяє виявляти помилки на етапі компіляції, ще до запуску коду в браузері – особливо актуально для форм введення та об'єктів, що передаються між компонентами. По-друге, підтримка інтерфейсів і типів дозволяє точно описати структури даних, якими обмінюються клієнт і сервер (наприклад, тип Marker, User, Comment, Discussion), що значно зменшує кількість логічних помилок при роботі з API. По-третє, TypeScript суттєво покращує роботу в середовищах

розробки: автодоповнення, підказки типів і статичний аналіз прискорюють написання та рефакторинг коду.

TypeScript використовується для типізації коду та підвищення надійності програмної системи.

Переваги використання TypeScript:

– статична типізація;
– раннє виявлення помилок;
– покращення читабельності коду;
– зручність підтримки великих проєктів;
– покращена інтеграція з IDE.

2.2.3. Node.js та Express

Node.js використовується як серверне середовище виконання.

Express.js використовується для побудови REST API. Express обраний завдяки мінімалістичному дизайну, гнучкій системі middleware та широкій екосистемі.

У проєкті Express обробляє REST-запити, здійснює маршрутизацію до відповідних контролерів, а також інтегровано з Socket.io для підтримки WebSocket-з'єднань. Структура серверного коду організована за принципом MVC-розподілу: папки routes/, controllers/, services/, models/ та middleware/ чітко розмежовують відповідальність кожного шару.

Основні переваги:

– висока продуктивність;
– неблокуюча модель вводу-виводу;
– швидка обробка великої кількості запитів;
– підтримка WebSocket;
– простота інтеграції з MongoDB.

2.2.4. MongoDB

MongoDB обрана як база даних через її документо-орієнтовану модель зберігання, яка добре підходить для напівструктурованих туристичних даних: маркери можуть мати різні набори полів залежно від

категорії, а коментарі та обговорення природно описуються вкладеними документами або посиланнями. Індекс типу 2dsphere на полі coordinates дозволяє ефективно виконувати геопросторові запити, наприклад пошук маркерів у заданому радіусі від поточного місцеположення. Доступ до MongoDB організовано через Mongoose, що забезпечує схемну валідацію, хуки збереження та зручний API для запитів.

MongoDB використовується як система керування базами даних.

Переваги MongoDB:

– документоорієнтована структура;
– гнучкість схеми даних;
– висока масштабованість;
– зручна робота з JSON-документами;
– ефективне зберігання мультимедійних даних.

2.2.5. Mapbox

Для відображення карти застосовано бібліотеку Mapbox GL JS. Порівняно з альтернативами, Mapbox забезпечує плавний рендеринг карт на основі WebGL, підтримку кастомних шарів і стилів, а також зручний API для роботи з маркерами та спливаючими вікнами. Для студентських і некомерційних проєктів Mapbox надає безкоштовний рівень доступу, достатній для розробки та тестування.

Mapbox використовується для інтеграції інтерактивних карт.

Функціональні можливості:

– відображення туристичних маршрутів;
– геолокація користувачів;
– побудова маршрутів;
– відображення туристичних об’єктів;
– інтерактивна робота з картою.

2.2.6. Socket.io

WebSocket-взаємодія реалізована через Socket.io – бібліотеку, що спрощує роботу з двосторонніми з'єднаннями та автоматично обробляє

повторне підключення, передачу даних у форматі JSON і трансляцію подій до кімнат підписників. На клієнті socket.io-client підписується на події marker:created, marker:updated, comment:added, discussion:updated, що дозволяє оновлювати стан React-компонентів без ручних запитів до сервера.

Socket.IO використовується для реалізації двостороннього обміну даними у режимі реального часу.

Основні можливості:

– миттєве оновлення повідомлень;
– онлайн-чат;
– сповіщення;
– оновлення коментарів без перезавантаження сторінки;
– синхронізація клієнтів.

2.2.7. Розмежування функцій REST API та Socket.io

У розробленому веб-застосунку для організації взаємодії між клієнтською та серверною частинами використовуються два механізми обміну даними: REST API та Socket.io. Незважаючи на те, що обидві технології забезпечують передачу інформації між клієнтом і сервером, вони призначені для вирішення різних задач.

REST API використовується для виконання операцій, які не потребують миттєвого оновлення даних у всіх підключених користувачів. Взаємодія відбувається за моделлю «запит-відповідь», коли клієнт надсилає HTTP-запит, а сервер повертає результат виконання операції.

Socket.io використовується для організації двостороннього зв'язку в режимі реального часу. Після встановлення WebSocket-з'єднання сервер може самостійно надсилати повідомлення клієнтам без необхідності формування додаткових HTTP-запитів.

У розробленій системі функціональні обов'язки розподілені таким чином.

REST API відповідає за:

– реєстрацію користувачів;
– авторизацію та автентифікацію;
– отримання профілю користувача;
– створення, редагування та видалення туристичних публікацій;
– завантаження фотографій;
– отримання списку маршрутів;
– збереження коментарів;
– отримання історії повідомлень чату;
– роботу з базою даних MongoDB.

Приклади REST-запитів:

```
POST /api/auth/login
POST /api/posts
GET /api/posts
GET /api/routes
DELETE /api/posts/:id
```

Socket.io відповідає за:

– миттєву доставку повідомлень у чаті;
– оновлення списку повідомлень без перезавантаження сторінки;
– відображення статусу користувача (online/offline);
– сповіщення про нові коментарі;
– сповіщення про нові публікації;
– оновлення інформації про активність користувачів у реальному часі.

Приклади подій Socket.io:

```
socket.emit("sendMessage");
socket.on("newMessage");
socket.emit("newComment");
socket.on("commentAdded");
socket.on("userConnected");
socket.on("userDisconnected");
```

Таким чином, REST API використовується для виконання CRUD-операцій та взаємодії з базою даних, тоді як Socket.io використовується виключно для забезпечення функцій реального часу.

Такий підхід дозволяє уникнути дублювання логіки застосунку, зменшити навантаження на сервер та забезпечити ефективну взаємодію користувачів (табл. 2.1).

Таблиця 2.1. Розподіл функціональності між REST API та Socket.io

Функція	**REST API**	**Socket.io**
Реєстрація користувача	✓	
Авторизація	✓	
Отримання профілю	✓	
Створення публікації	✓	
Завантаження фото	✓	
Отримання маршрутів	✓	
Надсилання повідомлення		✓
Отримання повідомлення в реальному часі		✓
Онлайн-статус користувача		✓
Сповіщення про новий коментар		✓
Сповіщення про нову публікацію		✓
Історія повідомлень	✓	

2.3. Проєктування структури веб-застосунку

Система проєктується за модульним принципом, що дозволяє ізолювати функціональні компоненти та спростити супровід програмного забезпечення.

Основні модулі системи:

1. Модуль авторизації.
2. Модуль профілів користувачів.
3. Модуль туристичних публікацій.
4. Модуль коментарів.
5. Модуль чатів.
6. Модуль інтерактивних карт.

7. Модуль рекомендацій.
8. Адміністративний модуль.

Модульну структуру системи представлено на рис. 2.6.

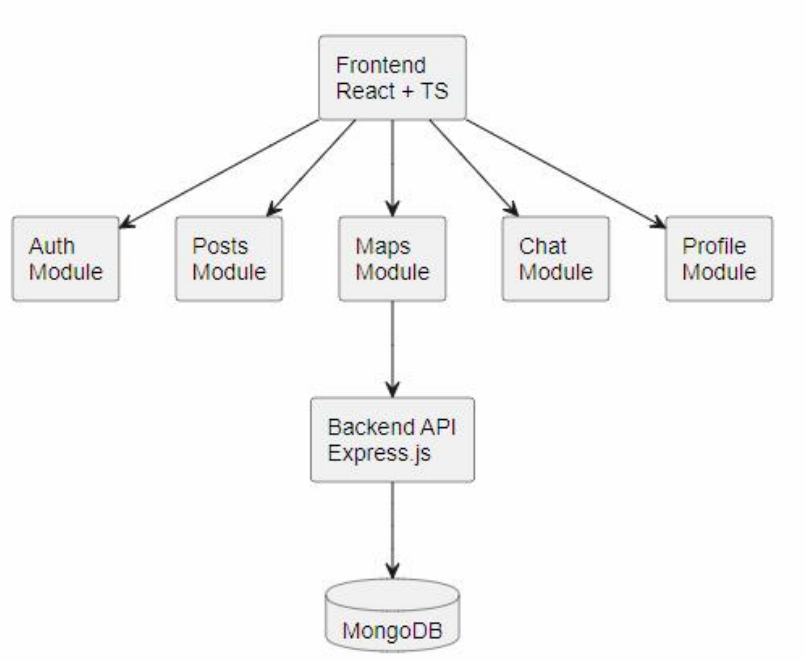

Рисунок 2.6. Модульна структура системи

Діаграма класів системи (рис. 2.7) відображає основні сутності системи та зв’язки між ними.

Для взаємодії між клієнтською та серверною частинами використовується REST API та WebSocket-з’єднання.

REST API забезпечує:

– авторизацію;
– роботу з публікаціями;
– завантаження даних;
– роботу з профілями;
– керування маршрутами.

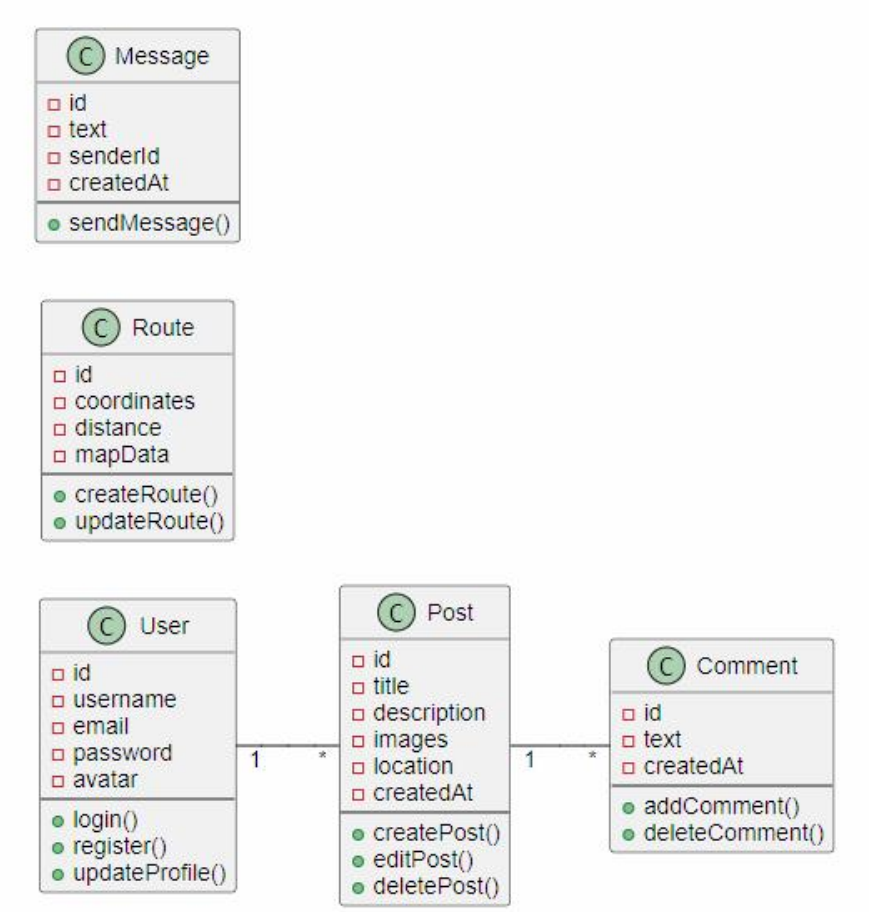

Рисунок 2.7. Діаграма класів системи

2.4. Проєктування взаємодії компонентів системи

Socket.io забезпечує:

– чат у реальному часі;
– сповіщення;
– оновлення коментарів;
– відображення онлайн-статусу користувачів.

Схему взаємодії компонентів представлено на рис. 2.8.

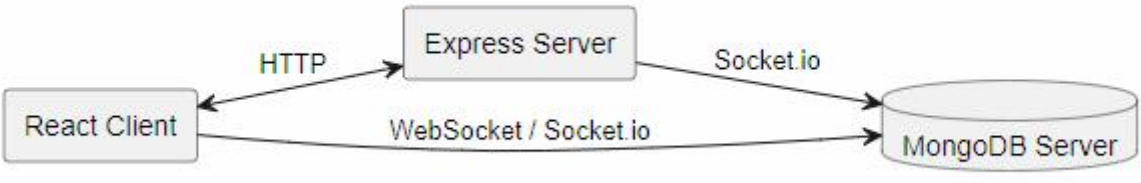

Рисунок 2.8. Схема взаємодії компонентів

На рис. 2.9 представлено діаграму діяльності щодо створення туристичної публікації.

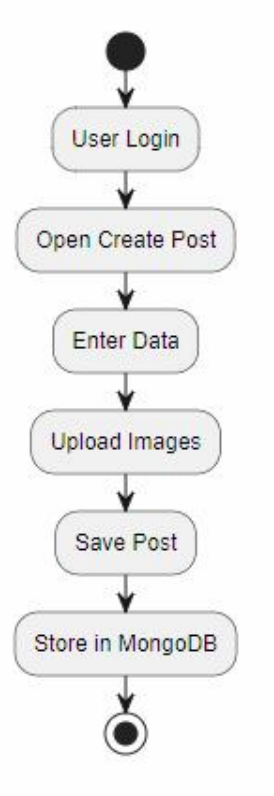

Рисунок 2.9. Діаграма діяльності

2.5. Проєктування структури бази даних

Для зберігання даних використовується документоорієнтована база даних MongoDB.

Основними колекціями є (рис. 2.10):

– Users;
– Posts;
– Comments;
– Routes;
– Messages;

– Notifications.

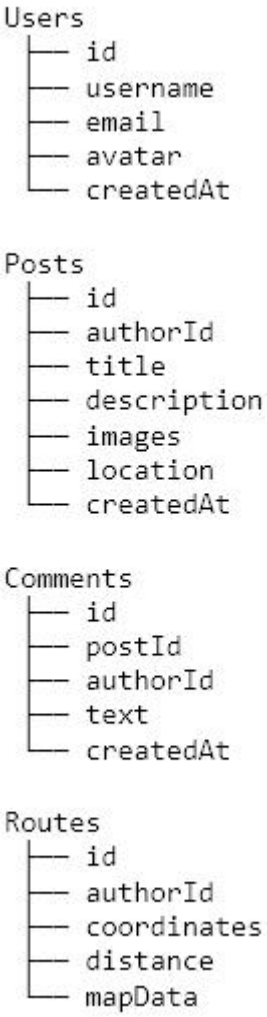

Рисунок 2.10. Логічна структура бази даних

ER-діаграма (рис. 2.11) демонструє структуру бази даних та взаємозв’язки між сутностями.

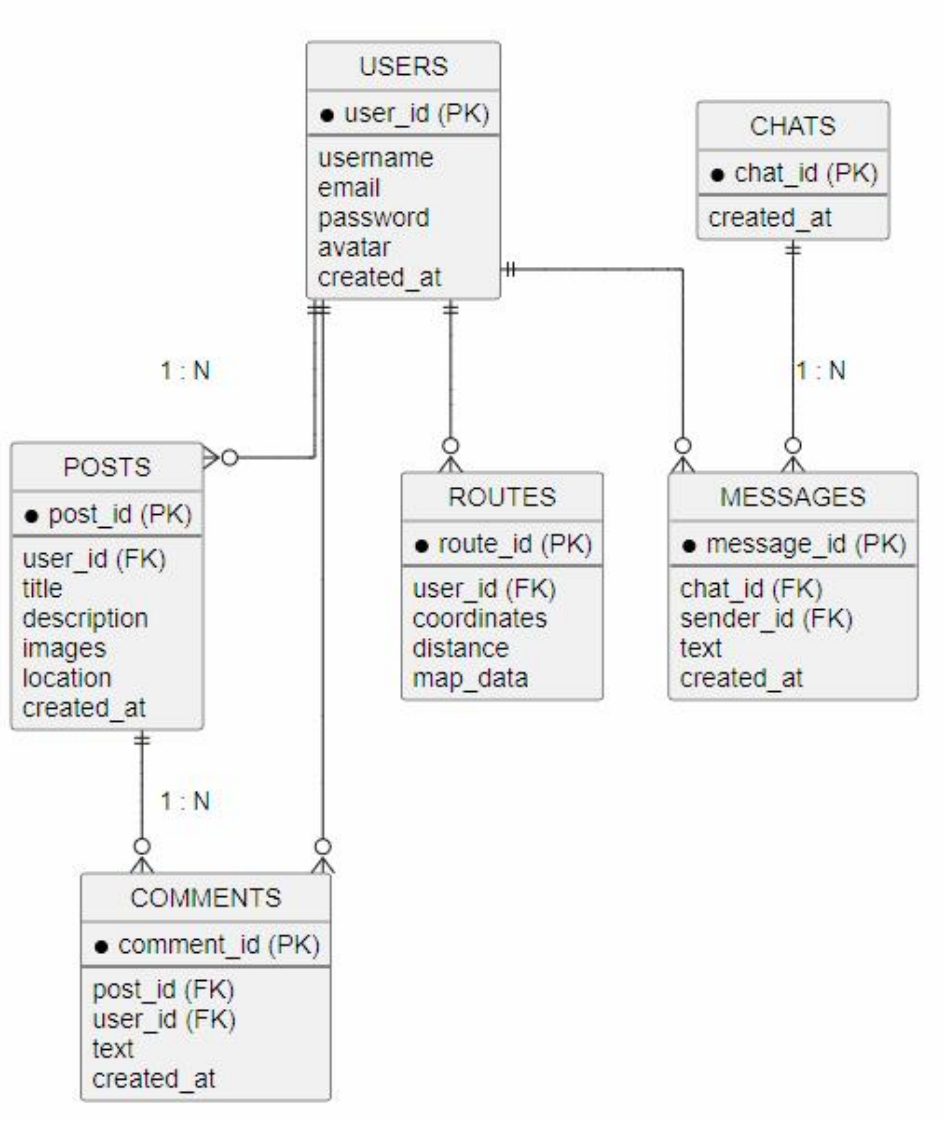

Рисунок 2.11. ER-діаграма

2.6. Проєктування користувацького інтерфейсу

Інтерфейс веб-застосунку повинен відповідати принципам usability та responsive design.

Основними сторінками системи є:

– головна сторінка;
– сторінка маршруту;
– сторінка профілю;
– сторінка обговорень;
– сторінка карти;
– адміністративна панель.

Інтерфейс проєктується адаптивним для:

– персональних комп’ютерів;
– планшетів;
– мобільних пристроїв.

2.7. Аналіз переваг обраної архітектури

Обрана архітектура має такі переваги:

– висока масштабованість;
– модульність;
– підтримка роботи в реальному часі;
– швидка взаємодія клієнта та сервера;
– гнучкість розробки;
– простота розширення функціональності;
– підтримка сучасних веб-стандартів.

Використання React та Node.js забезпечує єдину JavaScript-екосистему, що спрощує розробку та супровід програмного забезпечення.

Висновки до розділу

У другому розділі було спроєктовано архітектуру веб-застосунку для інтерактивного обговорення туристичного досвіду. Було обґрунтовано вибір технологій React, TypeScript, Node.js, Express, MongoDB, Mapbox та Socket.io.

Результатом проєктування стала модульна клієнт-серверна архітектура, яка забезпечує:

– масштабованість;
– підтримку інтерактивної взаємодії;
– роботу в реальному часі;
– інтеграцію геолокаційних сервісів;
– ефективне зберігання даних;
– адаптивний користувацький інтерфейс.

Отримані результати є основою для реалізації програмної системи у наступному розділі кваліфікаційної роботи.

Розроблені UML-діаграми та ER-діаграма дозволяють формалізувати структуру та поведінку веб-застосунку для інтерактивного обговорення туристичного досвіду.

Створені діаграми відображають:

– функціональні можливості системи;
– взаємодію користувачів;
– архітектуру компонентів;
– логіку обробки даних;
– структуру бази даних;
– процеси взаємодії між клієнтською та серверною частинами.

Отримані результати можуть бути використані під час подальшої реалізації, тестування та супроводу програмного забезпечення.

РОЗДІЛ 3. РОЗРОБЛЕННЯ ТА ПРАКТИЧНА РЕАЛІЗАЦІЯ ВЕБ-ЗАСТОСУНКУ ДЛЯ ІНТЕРАКТИВНОГО ОБГОВОРЕННЯ ТУРИСТИЧНОГО ДОСВІДУ

3.1. Загальна характеристика процесу розроблення системи

Практична реалізація веб-застосунку виконувалась відповідно до розробленої у другому розділі архітектури клієнт-серверного типу. Основною метою розроблення було створення сучасної інтерактивної платформи для обговорення туристичного досвіду, яка забезпечує:

– створення туристичних публікацій;
– обмін повідомленнями;
– взаємодію користувачів у режимі реального часу;
– роботу з інтерактивними картами;
– зберігання мультимедійного контенту.

Розроблення системи здійснювалось із використанням:

– React;
– TypeScript;
– Node.js;
– Express.js;
– MongoDB;
– Mapbox;
– Socket.io.

Процес реалізації складався з таких етапів:

1. Створення структури проєкту.
2. Розроблення серверної частини.
3. Реалізація клієнтського інтерфейсу.
4. Інтеграція картографічного сервісу.
5. Реалізація системи реального часу.
6. Підключення бази даних.
7. Тестування функціональності.

3.2. Реалізація серверної частини застосунку

Серверна частина (рис. 3.1) була реалізована на платформі Node.js із використанням фреймворку Express.js.

Основними функціями серверної частини є:

– обробка HTTP-запитів;
– авторизація користувачів;
– взаємодія з базою даних;
– робота з REST API;
– підтримка WebSocket-з’єднань;
– модерація контенту.

Рисунок 3.1. Структура серверної частини

Для організації серверної логіки було використано модульний підхід, який дозволяє розділити бізнес-логіку, маршрути та моделі даних.

3.3. Реалізація REST API

Для взаємодії клієнтської та серверної частин було реалізовано REST API (рис. 3.2).

Основні API-маршрути:

– /api/auth – авторизація користувачів;
– /api/users – робота з профілями;
– /api/posts – туристичні публікації;
– /api/comments – коментарі;
– /api/routes – туристичні маршрути;
– /api/messages – повідомлення.

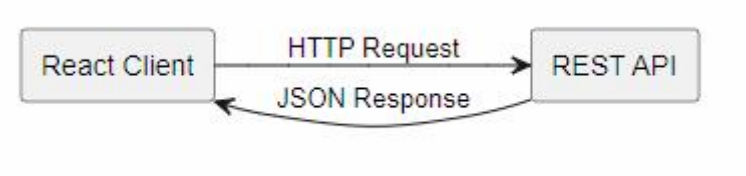

Рисунок 3.2. Схема роботи REST API

Для передачі даних використовується формат JSON.

3.4. Реалізація системи авторизації

Для забезпечення безпеки веб-застосунку було реалізовано систему реєстрації та авторизації користувачів (рис. 3.3).

Функціональні можливості:

– реєстрація нового користувача;
– вхід до системи;
– хешування паролів;
– автентифікація через JWT;
– перевірка прав доступу.

Паролі користувачів зберігаються у зашифрованому вигляді за допомогою алгоритму bcrypt.

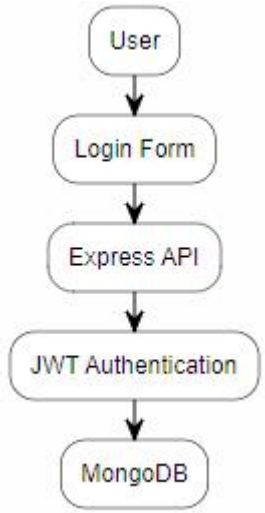

Рисунок 3.3. Схема авторизації користувача

Безпека користувацьких даних є одним із ключових нефункціональних пріоритетів системи. Реалізована модель автентифікації та авторизації базується на стандарті JWT у поєднанні з bcrypt-хешуванням паролів і рольовою моделлю доступу.

Процес реєстрації ініціюється відправкою POST-запиту з клієнтської форми на серверний ендпоінт /api/auth/register. Дані проходять двоетапну валідацію: на клієнті перевіряється формат email та відповідність підтвердження паролю, на сервері – виконується додаткова перевірка засобами express-validator, а також перевіряється унікальність email у колекції користувачів MongoDB. Пароль перед збереженням хешується бібліотекою bcrypt з автоматичною генерацією унікальної солі для кожного запису, що унеможливлює відновлення пароля навіть у разі компрометації бази даних. Після успішного збереження нового користувача сервер генерує JWT-токен, підписаний секретним ключем із файлу змінних середовища, та повертає його клієнту у тілі відповіді.

Процедура входу в систему реалізована схожим чином: клієнт відправляє email і пароль на ендпоінт /api/auth/login, сервер перевіряє наявність облікового запису та за допомогою bcrypt.compare() зіставляє введений пароль із збереженим хешем. У разі успіху видається новий JWT-токен. У разі будь-яких невідповідностей сервер повертає відповідний код статусу (401 або 403) без розкриття деталей внутрішньої логіки, що унеможливлює перебір облікових даних шляхом аналізу повідомлень про помилки.

JWT-токен містить у зашифрованому вигляді ідентифікатор користувача, його email та роль, а також обмежений термін дії. На клієнті токен зберігається у localStorage та автоматично додається до заголовка Authorization кожного HTTP-запиту через налаштований екземпляр Axios. На сервері реалізовано middleware-функцію authMiddleware, яка перехоплює захищені маршрути, декодує та верифікує токен, прикріплює

об'єкт користувача до об'єкта запиту req.user та передає управління контролеру. Відсутність або недійсність токена призводить до негайного відхилення запиту з кодом 401.

Рольова модель доступу передбачає два рівні привілеїв. Звичайний користувач може створювати, редагувати та видаляти виключно власний контент – маркери, коментарі та відповіді на дискусії. Адміністратор має додатковий доступ до панелі модерації: може видаляти будь-який контент, що порушує правила, та переглядати список зареєстрованих користувачів. Перевірка прав доступу реалізована на двох рівнях: middleware перевіряє роль для загальнодоступних адміністративних маршрутів, а контролери виконують порівняння authorId === req.user.id для операцій із конкретними ресурсами.

Моделі даних MongoDB побудовані з урахуванням принципів мінімальності та безпеки. Поле password у схемі User визначено з атрибутом select: false, що виключає його з будь-яких відповідей API за замовчуванням. Хук pre('save') виконує хешування пароля лише у разі його фактичної зміни, запобігаючи повторному хешуванню при оновленні інших полів профілю. Схема Marker містить геоіндексоване поле coordinates типу 2dsphere, що дозволяє виконувати просторові запити на кшталт «знайти всі точки в радіусі 10 км від заданих координат». Схема Comment та Discussion містить посилання на Marker і User через ObjectId-референси, що забезпечує цілісність даних та можливість populate-запитів для отримання повної інформації за одним запитом.

Для захисту від загроз типу NoSQL-ін'єкцій усі параметри запитів до бази формуються через типізовані Mongoose-методи без прямого рядкового підставлення. Вхідні дані перевіряються за типом, форматом і довжиною перед будь-якою операцією з базою. Крім того, реалізовано обмеження кількості запитів на ендпоінти авторизації через rate-limiting middleware, що ускладнює атаки підбору паролів.

Діаграма послідовності авторизації користувача (рис 3.4) демонструє процес авторизації користувача у системі.

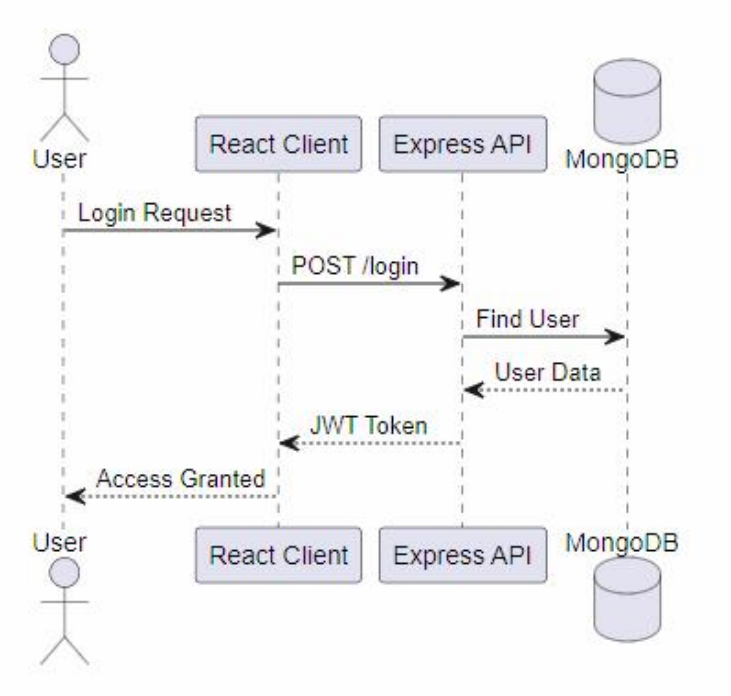

Рисунок 3.4. Діаграма послідовності

Оскільки для реалізації механізму автентифікації користувачів використано JWT-токени, у розробленому прототипі токен автентифікації може зберігатися у сховищі браузера (localStorage), що спрощує реалізацію клієнтської частини та забезпечує збереження сеансу між перезавантаженнями сторінки.

Водночас такий підхід має певні недоліки з точки зору інформаційної безпеки, оскільки у випадку успішної XSS-атаки зловмисник потенційно може отримати доступ до токена.

У професійних системах для зниження зазначених ризиків доцільно використовувати механізм зберігання токенів у httpOnly Secure Cookies, які є недоступними для JavaScript-коду на стороні клієнта. В рамках же даної кваліфікаційної роботи прийняте рішення щодо механізму автентифікації є цілком достатнім.

Додатковими заходами захисту можуть бути використання Content Security Policy (CSP), валідація та санітизація користувацького вводу, обмеження часу життя токенів та механізм їх періодичного оновлення (Refresh Token Rotation).

3.5. Реалізація клієнтської частини застосунку

Клієнтська частина (рис. 3.5) була реалізована за допомогою React та TypeScript.

Основні компоненти інтерфейсу:

– Navbar;
– Sidebar;
– PostCard;
– CommentSection;
– ChatWindow;
– MapView;
– UserProfile.

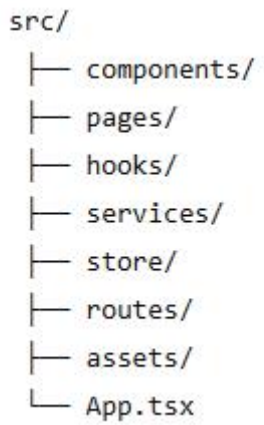

Рисунок 3.5. Структура клієнтської частини

Для керування станом застосунку використовувались React Hooks та Context API. Інтерфейс застосунку побудований за принципом мінімалізму та функціональної зрозумілості: кожен елемент екрана несе конкретне навантаження, не перевантажуючи користувача інформацією (табл. 3.1).

Таблиця 3.1. Основні компоненти інтерфейсу та їх призначення

Компонент	**Призначення**	**Технології**
MapPage	Головний контейнер карти та стану	React, Mapbox GL JS, Socket.io
Navbar	Навігація, стан авторизації	React, useContext, SCSS
MapMarker	Відображення POI на карті	Mapbox Marker, React.memo
MarkerPopup	Деталі маркера, кнопки дій	Mapbox Popup, React
DiscussionPanel	Ієрархічні коментарі та обговорення	React, Axios, Socket.io
AddMarkerForm	Форма створення нового маркера	React, useState, Axios
UpdateMarkerForm	Форма редагування маркера	React, useState, Axios
CommentThread	Окремий коментар із відповідями	React, рекурсивний рендеринг
ProfilePage	Профіль, активність, налаштування	React Router, Axios

Архітектура UI реалізована на основі компонентного підходу React із використанням SCSS-модулів для ізольованого стилізування та медіа-запитів для адаптивності.

Головний екран застосунку займає компонент MapPage, що є контейнером для карти, навігаційної панелі та бічних панелей. При першому завантаженні компонент виконує GET-запит до /api/markers, отримує список усіх маркерів та відображає їх на карті Mapbox. Одночасно встановлюється WebSocket-з'єднання, і компонент підписується на події marker:created, marker:updated, marker:deleted, comment:added та discussion:updated. Кожна з цих подій оновлює відповідну частину локального стану через setState-хуки, що забезпечує реактивне відображення змін без перезавантаження.

Компонент Navbar відображається у верхній частині екрана та динамічно адаптується залежно від стану авторизації. Для неавторизованих

користувачів він містить кнопки «Увійти» та «Зареєструватися», для авторизованих – ім'я або аватар, посилання на профіль та кнопку виходу. На мобільних пристроях навбар перетворюється на компактне «гамбургер»-меню з бічним висувним списком.

Взаємодія з маркерами реалізована через Mapbox Marker та Popup. При кліці на маркер з'являється спливаюче вікно з назвою, описом, категорією, рейтингом у вигляді зіркового індикатора, датою та автором. Авторизовані користувачі, які є авторами маркера, бачать додаткові кнопки «Редагувати» та «Видалити». Кнопка «Переглянути обговорення» відкриває бічну панель DiscussionPanel з повним ланцюжком коментарів.

Модуль обговорень є ключовою відмінністю розроблюваного застосунку від існуючих аналогів. Компонент DiscussionPanel відображає дерево коментарів для конкретного маркера, завантажених через GET-запит до /api/markers/:id/comments. Кожен коментар містить текст, ім'я автора, час публікації та кнопки «Відповісти» і «Подобається». Кнопка «Відповісти» відкриває вбудовану форму для введення відповіді з відступом, що позначає вкладеність. Після публікації коментаря через POST-запит сервер зберігає його в базі та транслює подію comment:added усім підключеним клієнтам, що перегукуються з цим маркером. Це дозволяє кільком користувачам, які одночасно переглядають один маркер, спостерігати нові коментарі в реальному часі без будь-яких ручних дій.

Компонент AddMarkerForm відкривається після подвійного кліку на карту та містить поля для назви, опису, категорії (з випадаючого списку), рейтингу та завантаження фотографій. Координати заповнюються автоматично на основі місця кліку, але можуть бути уточнені вручну. Форма використовує контрольований підхід через useState із вбудованою валідацією: обов'язковість полів, обмеження довжини, перевірка типу завантажуваних файлів та їх розміру. Після успішного збереження маркера

форма закривається, і новий маркер миттєво з'являється на карті для всіх підключених користувачів завдяки WebSocket-сповіщенню.

Оптимізація рендерингу карти реалізована через React.memo для компонентів маркерів, useCallback для обробників подій та обмеження кількості одночасно відображуваних маркерів на основі поточного вікна перегляду. Це дозволяє підтримувати плавну взаємодію навіть при значній кількості точок на карті.

3.6. Реалізація інтерактивної карти

Для реалізації картографічного функціоналу було використано сервіс Mapbox (рис. 3.6).

Основні можливості модуля:

– відображення туристичних об’єктів;
– побудова маршрутів;
– визначення координат;
– інтерактивна навігація;
– додавання геоміток.

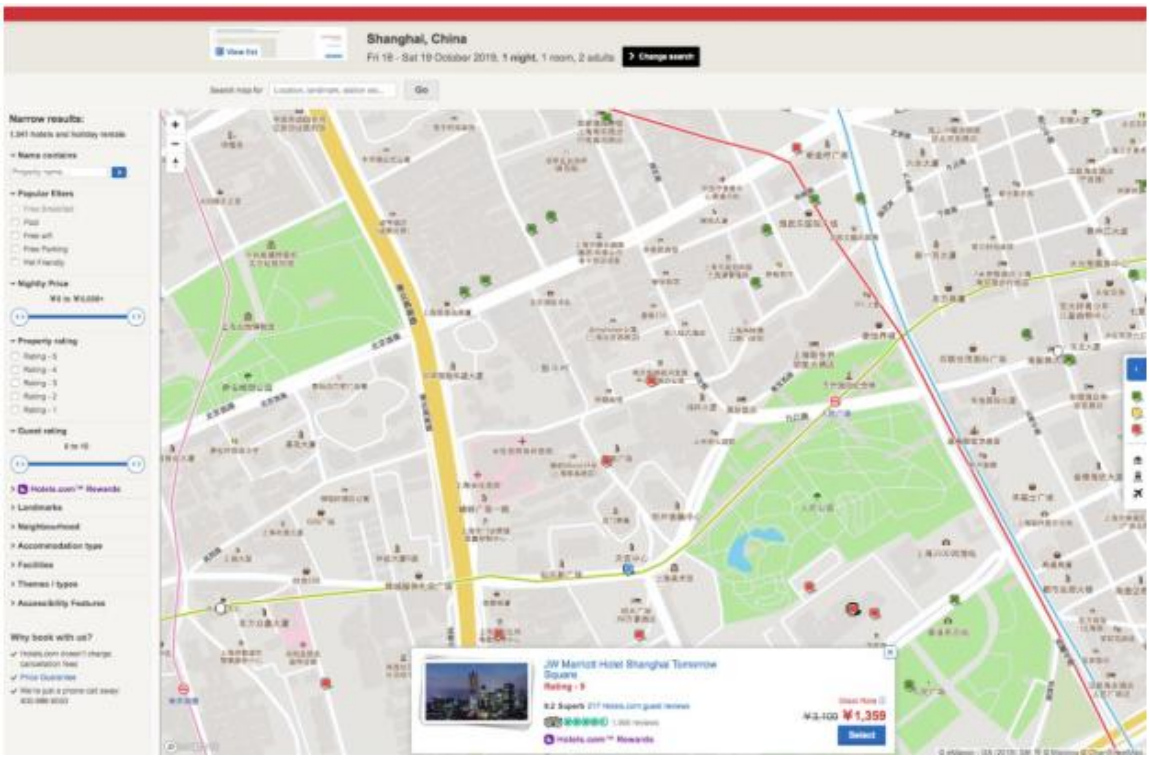

Рисунок 3.6. Інтерактивна карта Mapbox у веб-застосунку

Інтерактивна карта дозволяє користувачам додавати туристичні маршрути та переглядати геолокації інших користувачів.

3.7. Реалізація системи обміну повідомленнями

Для підтримки взаємодії у режимі реального часу (рис. 3.7) було використано бібліотеку Socket.IO.

Socket.io забезпечує:

– миттєвий обмін повідомленнями;
– оновлення коментарів;
– онлайн-сповіщення;
– синхронізацію користувачів.

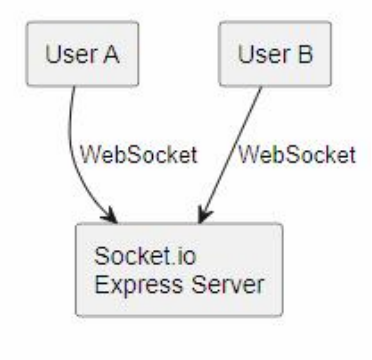

Рисунок 3.7. Схема роботи Socket.io

Завдяки використанню WebSocket-з’єднань забезпечується миттєва передача даних без необхідності оновлення сторінки.

3.8. Реалізація бази даних

Для зберігання інформації використовується MongoDB.

У системі реалізовано такі колекції (рис. 3.8):

– Users;
– Posts;
– Comments;
– Chats;
– Messages;
– Routes.

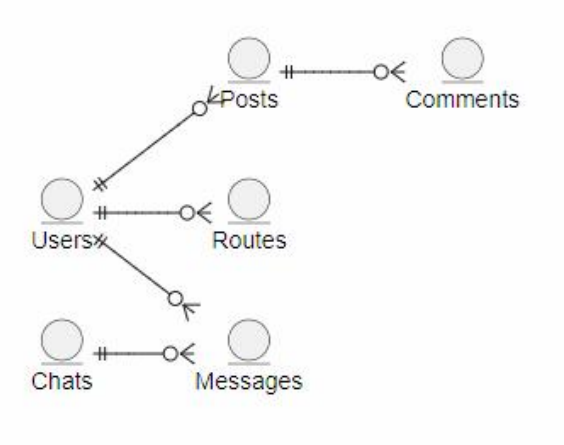

Рисунок 3.8. Взаємозв’язок колекцій MongoDB

MongoDB забезпечує ефективне зберігання JSON-документів та високу швидкість роботи із великими обсягами даних.

3.9. Реалізація адаптивного інтерфейсу

Однією з важливих вимог до системи була підтримка адаптивного дизайну.

Для цього використовувались:

– CSS Flexbox;
– CSS Grid;
– Media Queries;
– адаптивні React-компоненти.

Інтерфейс оптимізовано для:

– персональних комп’ютерів;
– планшетів;

У процесі реалізації веб-застосунку були створені такі функціональні можливості:

– створення публікацій;
– коментування записів;
– оцінювання туристичних місць;
– створення маршрутів;
– обмін повідомленнями;
– додавання фотографій;
– система повідомлень;
– профілі користувачів.

У результаті розроблення було створено працездатний веб-застосунок, який забезпечує:

– інтерактивну взаємодію користувачів;
– підтримку туристичних маршрутів;
– систему реального часу;
– інтеграцію картографічних сервісів;
– зручний сучасний інтерфейс.

Під час тестування було підтверджено:

– коректність роботи REST API;
– стабільність WebSocket-з’єднань;
– правильність роботи бази даних;
– адаптивність інтерфейсу;
– швидкодію системи.

Висновки до розділу

У третьому розділі було здійснено практичну реалізацію веб-застосунку для інтерактивного обговорення туристичного досвіду.

Було реалізовано:

– серверну частину на Node.js та Express;
– клієнтський інтерфейс на React і TypeScript;
– інтеграцію MongoDB;
– картографічний модуль Mapbox;
– систему обміну повідомленнями Socket.io.

Результатом роботи став сучасний веб-застосунок із підтримкою інтерактивної взаємодії, геолокації та обміну туристичним досвідом у режимі реального часу.

РОЗДІЛ 4. ТЕСТУВАННЯ ТА ВПРОВАДЖЕННЯ ВЕБ-ЗАСТОСУНКУ ДЛЯ ІНТЕРАКТИВНОГО ОБГОВОРЕННЯ ТУРИСТИЧНОГО ДОСВІДУ

4.1. Мета та завдання тестування програмної системи

Після завершення етапу розроблення веб-застосунку було проведено комплексне тестування системи з метою перевірки:

– коректності функціонування програмних модулів;
– стабільності роботи серверної частини;
– правильності взаємодії клієнтського та серверного застосунків;
– продуктивності системи;
– адаптивності користувацького інтерфейсу;
– безпечності обробки даних користувачів.

Основними завданнями тестування були:

1. Виявлення програмних помилок.
2. Перевірка відповідності функціональним вимогам.
3. Аналіз швидкодії веб-застосунку.
4. Перевірка роботи системи реального часу.
5. Тестування інтеграції картографічного сервісу.
6. Оцінювання зручності користувацького інтерфейсу.

4.2. Види тестування веб-застосунку

У процесі перевірки працездатності системи були використані різні види тестування.

4.2.1. Модульне тестування

Модульне тестування (рис. 4.1) використовувалось для перевірки окремих компонентів системи:

– API-маршрутів;
– функцій авторизації;
– React-компонентів;
– модулів роботи з базою даних;

– сервісів Socket.io.

Для тестування серверної частини використовувались:

– Jest;
– Supertest.

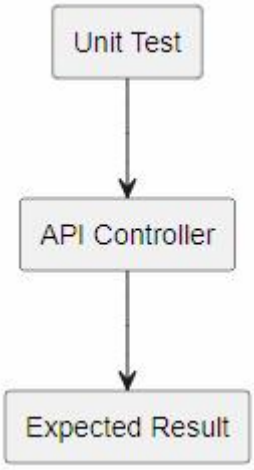

Рисунок 4.1. Схема модульного тестування

4.2.2. Інтеграційне тестування

Інтеграційне тестування (рис. 4.2) проводилось для перевірки взаємодії між компонентами системи:

– React ↔ REST API;
– Express ↔ MongoDB;
– Socket.io ↔ клієнт;
– Mapbox ↔ Frontend.

Основною метою було забезпечення коректного обміну даними між модулями системи.

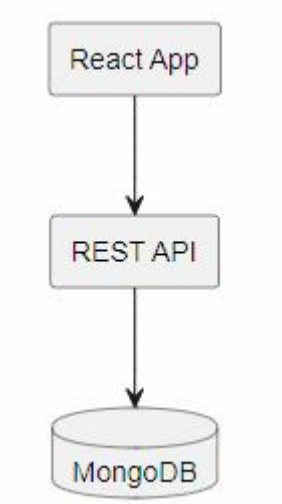

Рисунок 4.2. Схема інтеграційного тестування

4.2.3. Функціональне тестування

Функціональне тестування дозволило перевірити відповідність системи функціональним вимогам.

Було протестовано:

– реєстрацію користувачів;
– авторизацію;
– створення публікацій;
– додавання коментарів;
– створення маршрутів;
– роботу інтерактивної карти;
– чат у режимі реального часу.

Результати функціонального тестування веб-застосунку приведено в табл. 4.1.

Таблиця 4.1. Результати функціонального тестування

№	Функція	Очікуваний результат	Результат
1	Реєстрація	Створення акаунта	Успішно
2	Авторизація	Вхід у систему	Успішно
3	Створення публікації	Збереження запису	Успішно
4	Коментарі	Відображення коментаря	Успішно
5	Чат	Надсилання повідомлення	Успішно
6	Mapbox	Відображення карти	Успішно

4.3. Тестування користувацького інтерфейсу

Для перевірки адаптивного інтерфейсу користувача проводилось UI-тестування (рис. 4.3).

Основна увага приділялась:

– коректному відображенню компонентів;
– адаптивності інтерфейсу;
– навігації між сторінками;
– зручності користування;
– швидкості завантаження сторінок.

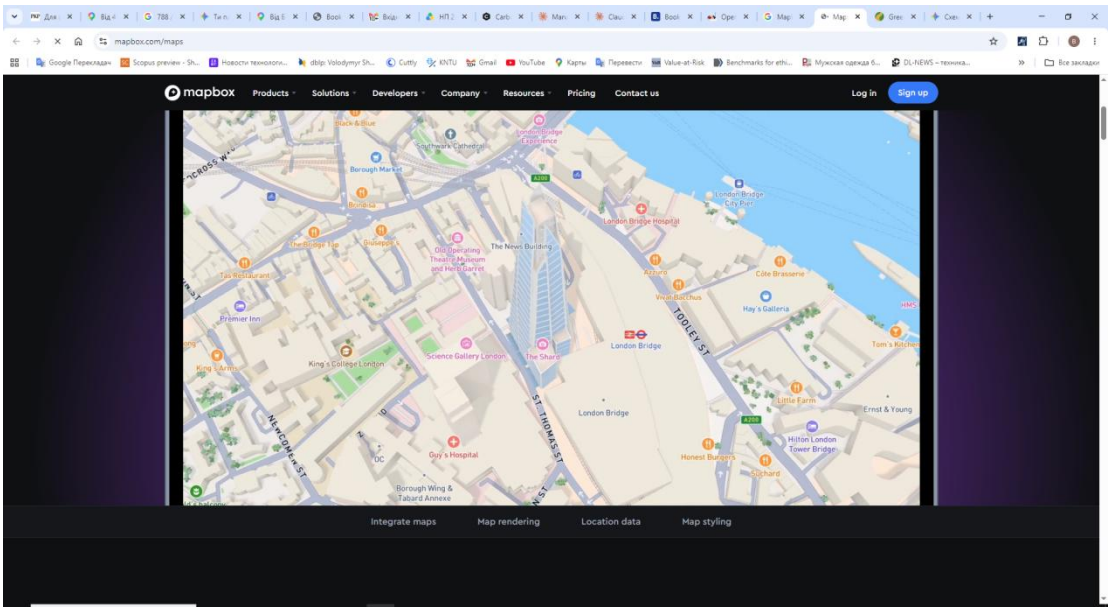

Рисунок 4.3. Тестування адаптивного інтерфейсу

Тестування проводилось для:

– Google Chrome;
– Mozilla Firefox;
– Microsoft Edge;
– мобільних браузерів Android.

4.4. Тестування системи реального часу

Особлива увага приділялась перевірці роботи Socket.io (рис. 4.4).

Було протестовано:

– миттєву передачу повідомлень;
– оновлення коментарів;
– роботу сповіщень;
– стабільність WebSocket-з’єднання.

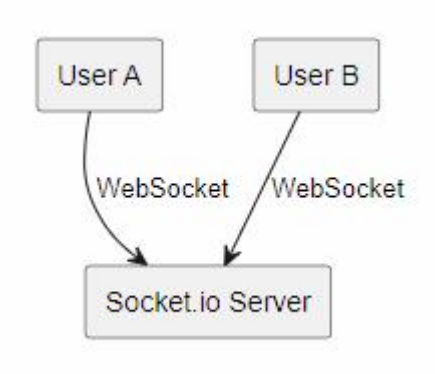

Рисунок 4.4. Тестування WebSocket-з’єднань

У результаті тестування встановлено, що затримка передачі повідомлень є мінімальною та не впливає на комфорт використання системи.

4.5. Тестування продуктивності системи

Для оцінювання швидкодії веб-застосунку проводилось тестування продуктивності.

Перевірялись:

– час відповіді REST API;
– швидкість завантаження сторінок;
– обробка одночасних запитів;

– використання пам’яті;
– стабільність роботи сервера.

Результати тестування продуктивності системи представлені в табл. 4.2, отримані кінцеві метрики представлено на рис. 4.5.

Таблиця 4.2. Результати тестування продуктивності

Параметр	**Значення**
Середній час відповіді API	120 мс
Завантаження головної сторінки	1.4 с
Час авторизації	0.8 с
Кількість одночасних користувачів	100+
Середнє використання CPU	38%

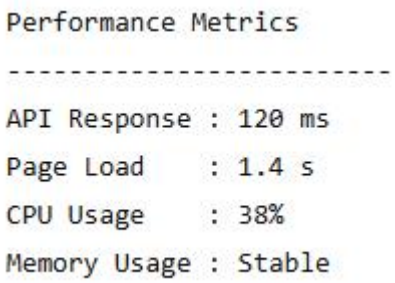

Рисунок 4.5. Аналіз продуктивності системи

4.6. Тестування безпеки

Під час тестування безпеки перевірялись:

– захист авторизації;
– шифрування паролів;
– перевірка JWT-токенів;
– захист REST API;
– перевірка прав доступу.

Для забезпечення безпеки були реалізовані:

– bcrypt-хешування;
– JWT-аутентифікація;
– middleware перевірки токенів;
– валідація даних.

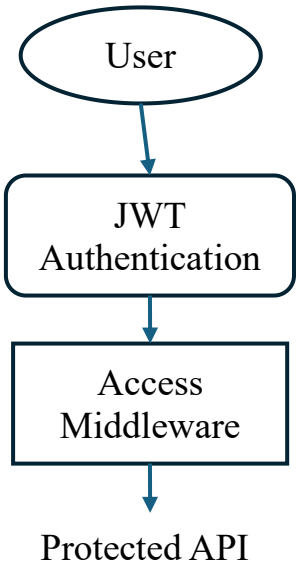

Рисунок 4.6. Схема безпеки системи

4.7. Впровадження веб-застосунку

Після завершення тестування було виконано впровадження системи.

Основними етапами впровадження були:

1. Підготовка серверного середовища.
2. Розгортання Node.js-застосунку.
3. Налаштування MongoDB.
4. Підключення домену.
5. Налаштування HTTPS.
6. Конфігурація Socket.io.
7. Запуск системи у робочому режимі.

Схема розгортання системи представлена на рис. 4.7.

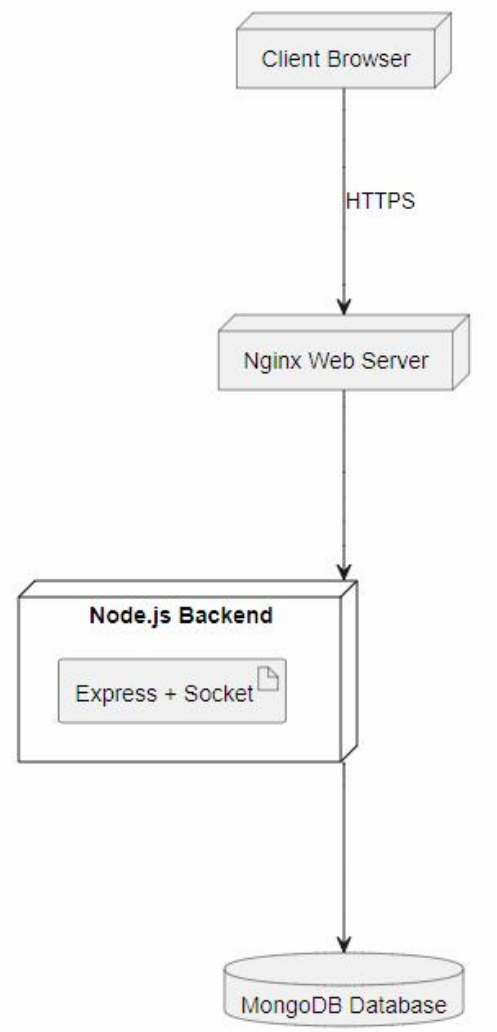

Рисунок 4.7. Схема розгортання системи

4.8. Характеристика середовища впровадження

Для розгортання веб-застосунку використовувалось таке програмне забезпечення:

– Ubuntu Server;
– Node.js;
– MongoDB;
– Nginx;
– PM2.

Основні характеристики сервера:

– процесор: 4 ядра;
– оперативна пам’ять: 8 ГБ;
– SSD-накопичувач: 100 ГБ;

– операційна система: Ubuntu Server 22.04.

4.9. Аналіз результатів впровадження

Під час впровадження виконувались експериментальні тестові запуски розробленого веб-застосунку.

Головний екран веб-застосунку представлено на рис. 4.8.

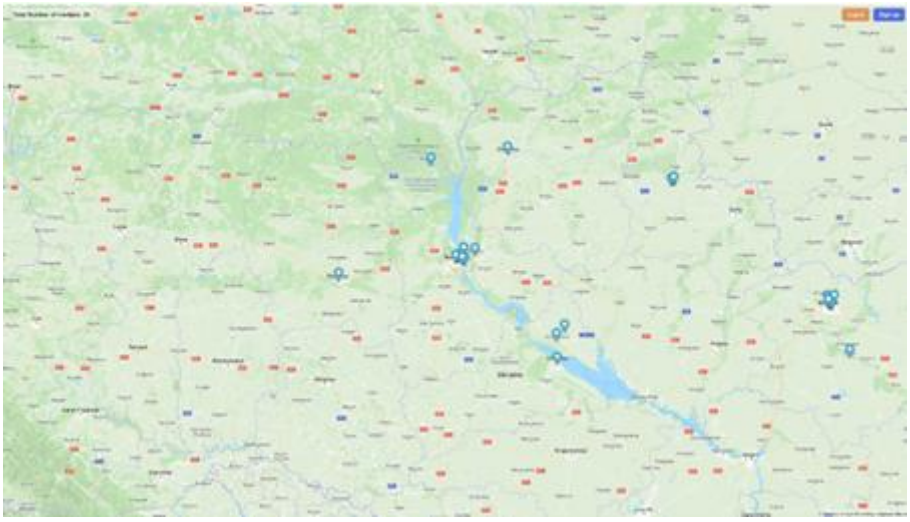

Рисунок 4.8. Головний екран веб-застосунку

Вікно перегляду інформації про встановлені маркери геолокації туристичних об’єктів представлено на рис. 4.9.

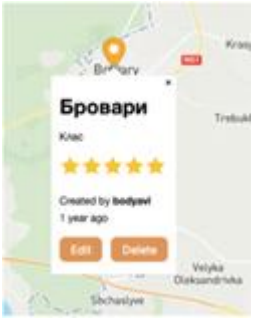

Рисунок 4.9. Вікно перегляду інформації про маркери

Вікно створення нового маркера для туристичних об’єктів представлено на рис. 4.10.

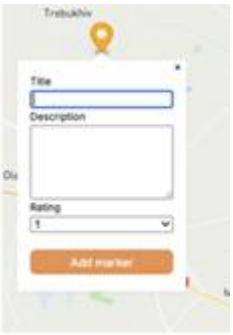

Рисунок 4.10. Вікно створення нового маркера

Контексте вікно маркера для перегляду інформації про туристичні об’єкти представлено на рис. 4.11.

Рисунок 4.11. Контекстне вікно маркера

У результаті проведеного тестування встановлено:

– коректність роботи функціональних модулів;
– стабільність REST API;
– надійність WebSocket-з’єднань;
– адаптивність інтерфейсу;
– відповідність функціональним вимогам.

Веб-застосунок успішно пройшов етап впровадження та готовий до практичного використання.

Висновки до розділу

У четвертому розділі було проведено комплексне тестування та впровадження веб-застосунку для інтерактивного обговорення туристичного досвіду.

Було виконано:

– модульне тестування;
– інтеграційне тестування;
– функціональне тестування;
– тестування продуктивності;
– тестування безпеки;
– впровадження системи у серверне середовище.

Результати тестування підтвердили працездатність, стабільність та ефективність розробленого веб-застосунку. Система забезпечує швидку взаємодію користувачів, підтримку роботи в реальному часі та інтеграцію геолокаційних сервісів, що дозволяє використовувати її як сучасну платформу для обговорення туристичного досвіду.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи розроблено функціональний та захищений веб-застосунок для інтерактивного обговорення туристичного досвіду, який забезпечує створення туристичних публікацій, взаємодію користувачів у режимі реального часу, підтримку геолокаційних сервісів та інтеграцію мультимедійного контенту.

Досягнута мета дослідження – створення системи, що забезпечує повноцінне середовище для обміну враженнями, думками та рекомендаціями між мандрівниками безпосередньо в геопросторовому контексті, з оновленням даних у режимі реального часу.

У ході виконання роботи було проведено комплексний аналіз предметної області та сучасних цифрових туристичних платформ, зокрема Tripadvisor, Google Maps і Wikiloc. Проведений порівняльний аналіз дозволив визначити основні переваги та функціональні обмеження існуючих рішень. Аналіз виявив, що всі три системи мають спільну суттєву ваду – відсутність повноцінної можливості для інтерактивного діалогу між користувачами навколо конкретних туристичних об'єктів. Було встановлено, що сучасні туристичні сервіси недостатньо підтримують інтерактивне спілкування користувачів та не забезпечують єдиного середовища для обговорення туристичного досвіду, побудови маршрутів і соціальної взаємодії. Виявлена прогалина обґрунтувала доцільність розробки нового рішення.

На основі проведеного аналізу було сформовано функціональні та нефункціональні вимоги до розроблюваної програмної системи. Основними вимогами стали підтримка адаптивного інтерфейсу, інтерактивних карт, системи реального часу, безпечної авторизації користувачів та масштабованої клієнт-серверної архітектури.

У процесі проєктування було розроблено архітектуру веб-застосунку на основі сучасного технологічного стеку: клієнтська частина – TypeScript і React, серверна – Node.js із Express, база даних – MongoDB із Mongoose, картографія – Mapbox GL JS, реальний час – Socket.io. Таке поєднання забезпечило суцільне JavaScript-середовище з єдиними принципами розробки на клієнті та сервері, підтримку статичної типізації через TypeScript, гнучке зберігання напівструктурованих туристичних даних у MongoDB та ефективну двосторонню синхронізацію через WebSocket.

Було спроєктовано структуру клієнтської та серверної частин системи, REST API, логічну структуру бази даних, UML-діаграми та ER-діаграму, що дозволило формалізувати процеси взаємодії компонентів програмної системи.

Реалізовано захищений механізм автентифікації: хешування паролів через bcrypt із динамічною сіллю, авторизація через JWT-токени з обмеженим терміном дії, рольова модель доступу (User/Admin), middleware-перевірка прав доступу до кожного захищеного ендпоінта, захист від NoSQL-ін'єкцій та XSS. Принцип «авторство = право редагування» реалізовано на рівні контролерів через зіставлення authorId з ідентифікатором поточного користувача з JWT.

Ключовою функціональною інновацією є компонент DiscussionPanel – модуль ієрархічних обговорень, прив'язаних до конкретних геолокаційних маркерів. Завдяки WebSocket-синхронізації нові коментарі та відповіді моментально з'являються у всіх підключених клієнтів, що переглядають той самий маркер. Такий підхід принципово відрізняє розроблювану систему від існуючих аналогів, де взаємодія є виключно асинхронною.

Для підтримки інтерактивної взаємодії між користувачами було реалізовано WebSocket-з’єднання за допомогою бібліотеки Socket.IO, що

забезпечило миттєвий обмін повідомленнями та оновлення даних без перезавантаження сторінки.

У роботі було проведено комплексне тестування програмної системи, яке включало модульне тестування, інтеграційне тестування, функціональне тестування, тестування продуктивності, тестування безпеки.

Результати тестування підтвердили стабільність роботи системи, коректність функціонування REST API, ефективність WebSocket-з'єднань, адаптивність інтерфейсу, відповідність функціональним вимогам.

Було виконано впровадження веб-застосунку у серверне середовище з використанням Node.js, MongoDB, Nginx та PM2. Система успішно функціонує у режимі клієнт-серверної взаємодії та готова до практичного використання.

Практичне значення роботи полягає у створенні сучасної веб-платформи для туристичних спільнот, яка може бути використана для обміну туристичним досвідом, створення маршрутів, комунікації між мандрівниками, організації туристичних обговорень, інтерактивної взаємодії користувачів.

Отримані результати можуть бути використані для подальшого розвитку системи, зокрема для впровадження системи рекомендацій на основі штучного інтелекту, інтеграції мобільного застосунку, підтримки офлайн-карт, додавання системи рейтингів і досягнень, інтеграції соціальних мереж.

Таким чином, поставлена мета кваліфікаційної роботи досягнута, а всі визначені завдання успішно виконані. Розроблена система є готовим до розгортання рішенням, що закриває реальну прогалину на ринку туристичних цифрових платформ і може бути використана як туристичними спільнотами, так і організаторами тематичних заходів, гідами та дослідниками локального туризму.

ПЕРЕЛІК ПОСИЛАНЬ

1. React Documentation. Meta. URL: https://react.dev/learn

2. TypeScript Documentation. Microsoft. URL: https://www.typescriptlang.org/docs

3. Node.js Documentation. OpenJS Foundation. URL: https://nodejs.org/en/docs

4. Express Documentation. OpenJS Foundation. URL: https://expressjs.com

5. MongoDB Documentation. MongoDB, Inc. URL: https://www.mongodb.com/docs

6. Mongoose Documentation. Mongoose ODM. URL: https://mongoosejs.com/docs

7. Socket.IO Documentation. URL: https://socket.io/docs/v4

8. Mapbox GL JS Documentation. URL: https://docs.mapbox.com/mapbox-gl-js

9. JSON Web Token Introduction. JWT.io (Auth0). URL: https://jwt.io/introduction

10. OWASP Top Ten. OWASP Foundation. URL: https://owasp.org/www-project-top-ten

11. Vite Documentation. URL: https://vitejs.dev/guide

12. React Router Documentation. Remix Software. URL: https://reactrouter.com

13. Axios Documentation. URL: https://axios-http.com/docs/intro

14. bcrypt npm package. URL: https://www.npmjs.com/package/bcrypt

15. Ibrahim M. How to use MERN Stack: A Complete Guide. MongoDB. URL: https://www.mongodb.com/developer/languages/javascript/mern-stack-tutorial

16. Ahmad K. MERN TypeScript Setup Guide. DEV Community. URL: https://dev.to/kafeel_ahmad/mern-typescript-setup-guide-4lbf

17. Aboalganam K. M., AlFraihat S. F. The Impact of User-Generated Content on Tourist Visit Intentions // Administrative Sciences. 2025. 15(4): 117.

18. Cross-Origin Resource Sharing (CORS). MDN Web Docs. URL: https://developer.mozilla.org/docs/Web/HTTP/CORS

19. Helmet.js – Express security middleware. URL: https://helmetjs.github.io

20. SQL vs NoSQL: 5 Critical Differences. Integrate.io Blog. URL: https://www.integrate.io/blog/sql-vs-nosql-differences

21. Don't Block the Event Loop. Node.js Official Docs. URL: https://nodejs.org/en/learn/asynchronous-work/dont-block-the-event-loop

22. Importance of User-Generated Content in Tourism Events // Journal of Business. 2021. URL: https://openaccessojs.com/JBReview/article/view/4546

23. Using promises – JavaScript. MDN Web Docs. URL: https://developer.mozilla.org/docs/Learn/JavaScript/Asynchronous/Promises

24. Error Handling in Express.js. GeeksforGeeks. URL: https://www.geeksforgeeks.org/error-handling-in-express-js

25. Getting Started with MERN Stack. GeeksforGeeks. URL: https://www.geeksforgeeks.org/getting-started-with-mern-stack

26. Balaji D. JWT Authentication in Node.js: A Practical Guide. Medium. URL: https://dvmhn07.medium.com/jwt-authentication-in-node-js-a-practical-guide-c8ab1b432a49

27. Бéjar R., Umer M. A Mobile Application to Share Georeferenced Tourist Experiences on a Discrete Global Grid // GEOProcessing 2020. 2020. URL: https://thinkmind.org/articles/geoprocessing_2020

28. Express + Node.js Handbook. freeCodeCamp. URL: https://www.freecodecamp.org/news/express-js-and-node-js-handbook

29. Travel Journal App with MERN Stack. GeeksforGeeks. URL: https://www.geeksforgeeks.org/travel-journal-app-with-mern-stack-with-api

30. DOMPurify Documentation. URL: https://github.com/cure53/DOMPurify

31. When to Use NoSQL and MongoDB. Fivetran Blog. URL: https://www.fivetran.com/blog/when-to-use-nosql-and-mongodb

32. React Router Documentation. URL: https://reactrouter.com/en/main

33. Asmare E. Setting up Express, TypeScript, MongoDB. Medium. URL: https://medium.com/@it.ermias.asmare/part-one-setting-up-express-typescript-mongodb

34. Express.js and MongoDB REST API Tutorial. MongoDB. URL: https://www.mongodb.com/languages/express-mongodb-rest-api-tutorial

ДОДАТОК А. Фрагменти програмного коду

SignUp.tsx

```
import s from './SignUp.module.scss'
import {useState, FormEvent} from 'react'
import { useNavigate } from 'react-router-dom'
import { signup } from '../../services';
const SignUp = () => {
const navigate = useNavigate();
const [email, setEmail] = useState('');
const [username, setUsername] = useState('');
const [password, setPassword] = useState('');
const [isLoading, setIsLoading] = useState(false);
async
function
handleSubmit(e:
FormEvent<HTMLFormElement>) {
setIsLoading(true);
e.preventDefault();
let res = await signup(email, password, username);
if(res.status === 400) {
alert("User with this username already exists!")
setIsLoading(false);
return;
}
let result = await res.json();
localStorage.setItem('accessToken',
result.accessToken)
localStorage.setItem('username', result.username);
localStorage.setItem('refreshToken',
result.refreshToken)
setIsLoading(false);
navigate('/')
}
return (
<>
<form
onSubmit={handleSubmit}
className={s.formContainer}>
<label
htmlFor="email"
className={s.formLabel}>Email</label>
<input
type="email"
id='email'
className={s.formTextInput} name='email' onChange={e
=>
setEmail(e.target.value)}
disabled={isLoading}
required/>
<label
htmlFor="username"
className={s.formLabel}>Username</label>
<input
type="text"
id='username'
className={s.formTextInput}
name='username'
onChange={e
```

```
=>
setUsername(e.target.value)}
disabled={isLoading} required/>
<label
htmlFor="password"
className={s.formLabel}>Password</label>
<input
type="password"
id='password'
className={s.formTextInput}
name='password'
onChange={e
=>
setPassword(e.target.value)}
disabled={isLoading} required/>
<button
className={s.formButton}
disabled={isLoading}>Sign up</button>
</form>
</>
)
}
export default SignUp
Login.tsx
import s from './Login.module.scss'
import { FormEvent, useState } from 'react'
import { Link, useNavigate } from 'react-router-dom';
import { login } from '../../services';
const Login = () => {
const navigate = useNavigate();
const [email, setEmail] = useState('');
const [password, setPassword] = useState('');
const [isLoading, setIsLoading] = useState(false);
async
function
handleSubmit(e:
FormEvent<HTMLFormElement>) {
setIsLoading(true);
e.preventDefault();
let res = await login(email, password)
if(res.status === 500) {
alert("User with this email doesn't exist!")
setIsLoading(false);
return;
}
else if(res.status === 403) {
alert("Wrong password!")
setIsLoading(false);
return;
}
let result = await res.json();
localStorage.setItem('accessToken',
result.accessToken)
localStorage.setItem('username', result.username);
localStorage.setItem('refreshToken',
result.refreshToken)
setIsLoading(false);
navigate('/');
}
return (
<>
```

```
<form
onSubmit={handleSubmit}
className={s.formContainer}>
<label
htmlFor="email"
className={s.formLabel}>Email</label>
<input
type="email"
id='email'
className={s.formTextInput} name='email' onChange={e
=>
setEmail(e.target.value)}
disabled={isLoading}
required/>
<label
htmlFor="password"
className={s.formLabel}>Password</label>
<input
type="password"
id='password'
className={s.formTextInput}
name='password'
onChange={e
=>
setPassword(e.target.value)}
disabled={isLoading} required/>
<button
className={s.formButton}
disabled={isLoading}>Login</button>
</form>
<Link to={'/forgotPassword'}></Link>
</>
)
}
```