ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка web-застосунку для обміну досвідом подорожей та планування особистих маршрутів»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

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

_____ Яна ЛИСАКОВСЬКА
(підпис)

Виконала: здобувачка вищої освіти групи ПД-41

Яна ЛИСАКОВСЬКА

Керівник: Владислав ЯСКЕВИЧ
к.т.н., доцент

Рецензент: _____

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
« ____ » _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Лисаковській Яні Віталіївні

1. Тема кваліфікаційної роботи: «Розробка web-застосунку для обміну досвідом подорожей та планування особистих маршрутів»
керівник кваліфікаційної роботи к.т.н., доцент Владислав ЯСКЕВИЧ,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «24» лютого 2025 р. № 56.

2. Строк подання кваліфікаційної роботи «02» червня 2025 р.

3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості про принципи розробки вебзастосунків, опис існуючих вебзастосунків для обміну досвідом подорожей та планування особистих маршрутів, огляд обраних технологій розробки та технічна документація до використаних фреймворків та бібліотек.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Огляд та аналіз існуючих вебзастосунків у сфері обміну досвідом подорожей.
2. Проектування web-застосунку для обміну досвідом подорожей та планування особистих маршрутів.

3. Програмна реалізація та опис функціонування застосунку для обміну досвідом подорожей та планування особистих машрутів.
4. Тестування застосунку.

5. Перелік графічного матеріалу: *презентація*

1. Аналіз аналогів.
2. Вимоги до програмного забезпечення.
3. Програмні засоби реалізації.
4. Діаграма варіантів використання.
5. Карта сайту
6. Структура бази даних
7. Екранні форми.
8. Апробація результатів дослідження

6. Дата видачі завдання«25» лютого 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	25.02.-04.03.2025	
2	Аналіз та дослідження існуючих аналогів	05.03-13.03.2025	
3	Огляд існуючих вебзастосунків у сфері обміну досвідом подорожей	14.03-18.03.2025	
4	Проектування web-застосунку для обміну досвідом подорожей та планування особистих маршрутів	19.03-25.03.2025	
5	Програмна реалізація застосунку	26.03-24.04.2025	
6	Тестування застосунку	25.04-28.04.2025	
7	Оформлення роботи: вступ, висновки, реферат	29.04-11.05.2025	
8	Розробка демонстраційних матеріалів	12.05-18.05.2025	
9	Попередній захист роботи	19.05-30.05.2025	

Здобувачка вищої освіти _____ Яна ЛИСАКОВСЬКА
(підпис)

Керівник
кваліфікаційної роботи _____ Владислав ЯСКЕВИЧ
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 57 стор., 2 табл., 43 рис., 21 джерел.

Мета роботи – спрощення процесу обміну особистим досвідом подорожей та надання базових функцій збереження вподобаних місць для планування маршрутів.

Об'єкт дослідження – процес обміну особистим досвідом подорожей та планування особистих маршрутів.

Предмет дослідження – web-застосунок для обміну досвідом подорожей та планування особистих маршрутів.

Короткий зміст роботи: У роботі досліджено процес обміну особистим досвідом подорожей та планування особистих маршрутів. Проаналізовано існуючі рішення та їх функціональні можливості. Розроблено алгоритм роботи web-застосунку та програмно реалізовані ключові функціональні можливості, зокрема: створення та перегляд історій подорожей, планування маршрутів, додавання відгуків та коментарів. Проведено функціональне та модульне тестування розробленого застосунку. Використано Node.js, фреймворк Express.js та нереляційну базу даних MongoDB для створення back-end частини. Для реалізації автентифікації використано бібліотеку JSON Web Token. Front-end частина розроблена з використанням React.

Сферою використання застосунку є обмін досвідом подорожей та планування особистих маршрутів користувачами.

КЛЮЧОВІ СЛОВА: ПОДОРОЖІ, ОБМІН ІСТОРІЯМИ, ПЛАНУВАННЯ, NODE.JS, REACT, MONGODB.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API - Application Programming Interface

JSON – JavaScript Object Notation

BSON - Binary JavaScript Object Notation

DOM - Document Object Model

UI - User Interface

HTTPS - Hypertext Transfer Protocol Secure

UX - User Experience

ODM - Object Data Modeling

XML - Extensible Markup Language

HTML - HyperText Markup Language

JS – JavaScript

ВСТУП

У сучасному світі подорожі перестали бути виключно способом відпочинку, перетворившись на важливу складову саморозвитку, отримання нових знань та налагодження культурних зв’язків. Зі зростанням популярності самостійних подорожей та прагненням до автентичного досвіду, зростає потреба у зручних цифрових інструментах, що дозволяють не лише зберігати спогади, але й обмінюватися ними з іншими мандрівниками, а також отримувати натхнення та інформацію для планування майбутніх поїздок.

Мета роботи – спрощення процесу обміну особистим досвідом подорожей та надання базових функцій збереження вподобаних місць для планування маршрутів.

Об’єкт дослідження – процес обміну особистим досвідом подорожей та планування особистих маршрутів.

Предмет дослідження – web-застосунок для обміну досвідом подорожей та планування особистих маршрутів.

Для реалізації поставленої мети виділено такі задачі:

1. Провести огляд та аналіз існуючих вебзастосунків для обміну досвідом подорожей та планування особистих маршрутів, визначити їх переваги та недоліки;
2. Сформулювати вимоги до розроблюваного web-застосунку на основі аналізу предметної галузі та потреб користувачів.
3. Провести огляд та аналіз технологій та інструментів, що можуть бути використані для розробки web-застосунку;
4. Спроектувати та розробити web-застосунок з використанням обраних технологій;
5. Провести тестування розробленого web-застосунку для виявлення та усунення помилок.

Для розробки web-застосунку було обрано мову програмування JavaScript, що забезпечує універсальність розробки як клієнтської (React), так і серверної (Node.js) частин. Використання Node.js дозволяє створити високонавантажений та масштабований бекенд завдяки його неблокуючій архітектурі. Фреймворк Express.js спрощує розробку API та керування маршрутами. Бібліотека React забезпечує творення інтерактивного та динамічного користувацького інтерфейсу з компонентним підходом та ефективним оновленням DOM. Для зберігання даних було обрано NoSQL базу даних MongoDB, яка відрізняється гнучкістю схеми та зручністю роботи з даними у форматі JSON.

Очікуваним результатом роботи є web-застосунок, який надасть користувачам зручний інструмент для обміну історіями подорожей, збереження цікавих місць та базового планування особистих маршрутів, сприяючи створенню спільноти зацікавлених мандрівників.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Актуальність web-застосунку для обміну досвідом подорожей та планування особистих маршрутів

У наші дні подорожі стали не лише способом відпочинку, а й можливістю самопізнання, натхнення та культурного обміну. Усе частіше люди подорожують не просто заради фото на фоні відомої історичної пам’ятки, а заради автентичного досвіду: дегустація традиційної кухні, пізнання нової культури не лише через відвідання популярних пам’яток, а і через маловідомі місця, які можуть розкрити набагато більше, ніж ті, які заповнені туристами, участь в фестивалях чи інших місцевих святкуваннях, які дозволяють краще пізнати місцеву культуру. Таким чином, зростає потреба у зручних цифрових інструментах, що здатні не лише зберігати спогади про подорожі, а і ділитися ними з іншими мандрівниками, отримувати натхнення для майбутніх поїздок та дізнаватися про нові цікаві місця, які можна використати для планування особистих маршрутів. Окрім цього, все більшої популярності зазнає тревел-блогінг, що також масово популяризує подорожі, та є важливим джерелом інформації про цікаві місця та культурні заходи [1, 3, 4]. Люди схильні довіряти відгукам та особистим розповідям більше ніж офіційним джерелам та рекламним описам. Обмін особистим досвідом дозволяє уникнути типових помилок, дізнатися інформацію, яку зазвичай не згадують у туристичних путівниках, та відчути атмосферу місця ще до того як потрапиш туди фізично.

На сьогоднішній день існують різні способи обміну туристичним досвідом: ведення особистих блогів, сторінок у соціальних мережах, розповсюдження відгуків про відвідані місця на спеціальних платформах. Однак, соціальні мережі зазвичай зосереджені на візуальному контенті, а не на детальному описі вражень та порад і до того ж не забезпечують зручне зберігання наданої інформації для подальшого використання. Ведення особистих блогів часом потребує часу,

зусиль та технічних знань, які не завжди є в звичайного мандрівника [1]. А спеціалізовані сервіси для відгуків часто обмежуються оцінкою лише окремих локацій і не надають можливість розповісти повну і детальну розповідь про подорож.

Інфокомунікаційні технології відкривають нові можливості для створення комплексного web-застосунку, який поєднує функції публікації подорожей у формі історій з фото, відмічення відвіданих місць та можливістю надихатися досвідом інших користувачів [2]. Таке рішення дозволяє задовольнити потреби сучасної цільової аудиторії.

Цільовою аудиторією такого застосунку є: досвідчені мандрівники, які хочуть поділитися порадами, новачки, які шукають натхнення та корисну збільшення своєї аудиторії та презентації своїх історій у зручному та привабливому форматі, а також користувачі, які хочуть зберігати власні спогади у вигляді інтерактивного щоденника подорожей, та разом зі спільнотою однодумців побачити найгарніші куточки світу [3, 4].

Серед основних вимог, які висуває аудиторія до подібного застосунку, можна виділити: інтуїтивно зрозумілий інтерфейс, можливість швидкого створення та редагування історії подорожі, зручний перегляд збережених місць, швидкий пошук за назвою та локацією та соціальна взаємодія на прикладі коментарів, де можна уточнити будь яке питання.

Таким чином, з огляду на зростаючу популярність подорожей як способу самовираження та обміну цінним досвідом, є актуальним створення web-застосунку, що поєднуватиме функції публікації особистих історій, збереження цікавих локацій та соціальної взаємодії. Такий інструмент не лише спростить організацію подорожей, а й створить спільноту однодумців, мандрівників, що об’єднані спільними інтересами, цінностями та бажанням пізнати світ.

1.2 Аналіз аналогів

Для вирішення завдань, пов’язаних з обміном подорожей, нині існує велика кількість програмних засобів та інструментів, що дозволяють користувачам ділитися враженнями та планувати особисті маршрути. Однак, кожен з цих сервісів має свої особливості та підходи до реалізації функцій, що допомагають мандрівникам. В цьому контексті варто розглянути наступні засоби та оцінити їх можливості та недоліки в обміні досвідом подорожей:

- TripAdvisor;
- Wanderlog;
- Google Maps;
- Travellerspoint.

1.2.1 TripAdvisor

TripAdvisor – це один з найвідоміших у світі вебсервісів для мандрівників, який надає можливість залишати відгуки про відвіданні місця, а також бронювати житло, екскурсії та авіаквитки [5]. Також, застосунок надає можливість зберігати вподобані місця для планування подорожей.

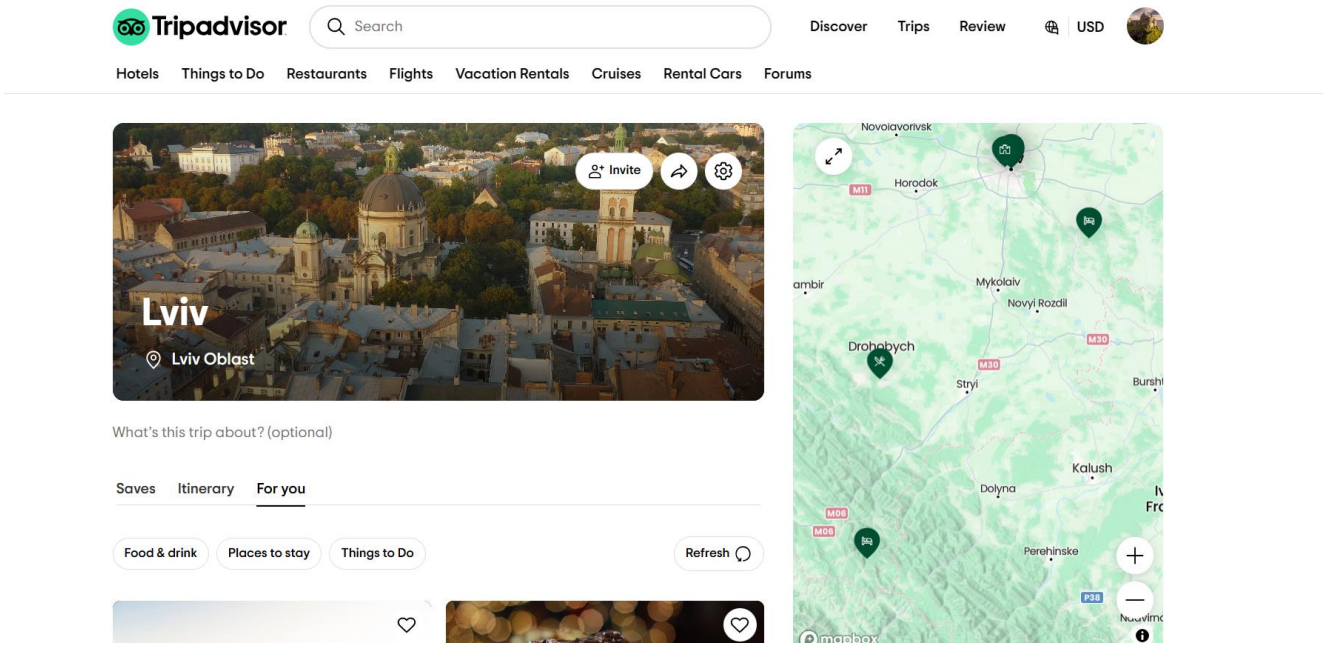


Рис. 1.1 Приклад екранної форми web-застосунку TripAdvisor

Основні переваги web-застосунку:

- Широке охоплення аудиторії – TripAdvisor має мільйони активних користувачів, що робить платформу надійною для пошуку відгуку та рекомендацій;

- Доступність контенту багатьма мовами;

- Можливість планування подорожі;

- Розділ форумів, де можна ставити питання та ділитися досвідом.

Однак, web-застосунок також має суттєві недоліки, так як немає багатьох функцій, які б хотілося бачити потенційним користувачам, серед яких можна відзначити такі:

- Фокус на відгуках про локації, а не на повноцінних розповідях про подорож;

- Відсутність соціальної складової – немає персонального профілю де б можна було зберігати хронологію мандрів;

- Інтерфейс перенасичений інформацією та рекламою, через що часом важко зрозуміти правдивість відгуку.

Таким чином, TripAdvisor є потужним інструментом для оцінювання та вибору місць під час планування подорожі. Однак з точки зору обміну досвідом у вигляді зв’язних розповідей про подорожі, він обмежений.

1.2.2 Wanderlog

Wanderlog позиціонує себе як платформу, що поєднує функції планування подорожей та обміну ідеями [6]. Це дозволяє користувачам не лише знаходити цікаві місця та складати маршрути власних подорожей, а і ділитися своїми планами та знахідками з іншими.

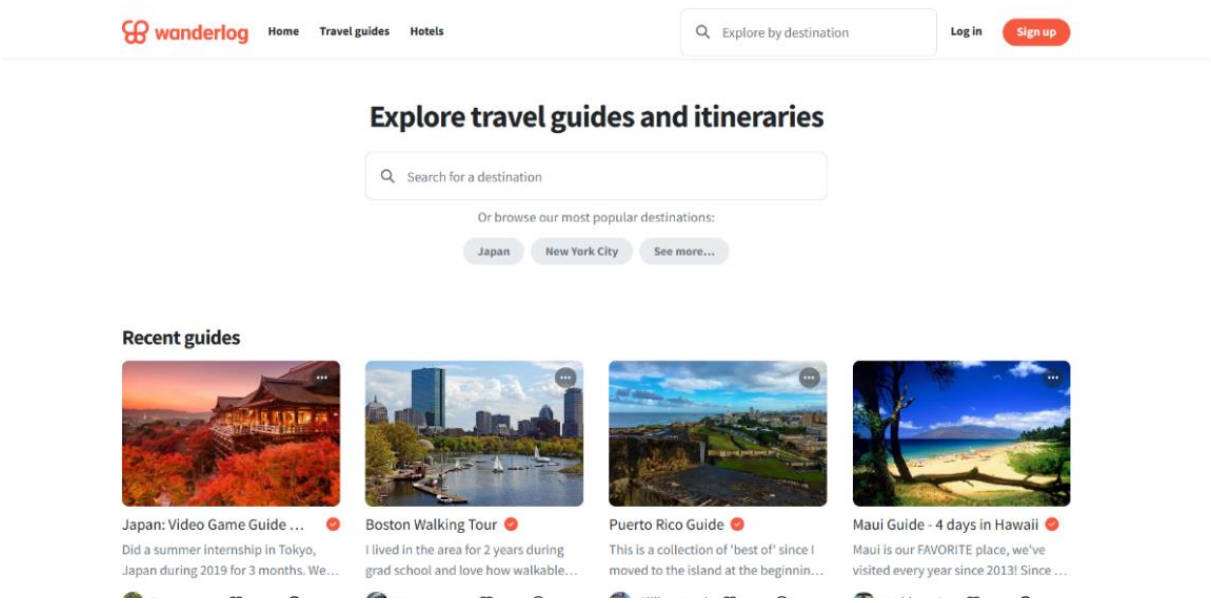


Рис. 1.2. Приклад екранної форми web-застосунку Wanderlog

Основні переваги web-застосунку:

- Wanderlog вдало поєднує інструменти для планування подорожей зі спільнотою, де користувачі мають змогу обмінюватися ідеями та рекомендаціями;

- Наявність візуалізації планів на карті, що робить процес планування більш інтуїтивно зрозумілим та легшим;

- Wanderlog пропонує також функції для спільного планування подорожей;

- Застосунок надає велику кількість інструментів для організації інформації про подорожі, такі як збереження місць, нотатки, посилання та інша корисна інформація в одному місці.

Однак, не дивлячись на всі переваги, web-застосунок має певні недоліки:

- Менший фокус на обміні історіями з подорожей. Wanderlog надає зручні можливості обміну ідей, або відвіданих місць, однак web-застосунок не орієнтований на обмін історіями з особистого досвіду подорожей. Основна увага йде на планування;

- Web-застосунок перенавантажений інструментами планування, через що, недосвідченим користувачам, чи тим що шукають ідеї для подорожей, інтерфейс може здатися менш зручним та інтуїтивно зрозумілим.

Можна зробити висновок, що Wanderlog є гарним інструментом планування подорожей для досвідчених користувачів. Однак з точки зору обміну досвідом та історіями з подорожей він є дуже обмеженим.

1.2.3 Google Maps

Google Maps є одним з найпопулярніших картографічних сервісів, що надає детальну інформацію про маршрути, місця, місцезнаходження та багато іншого [7].

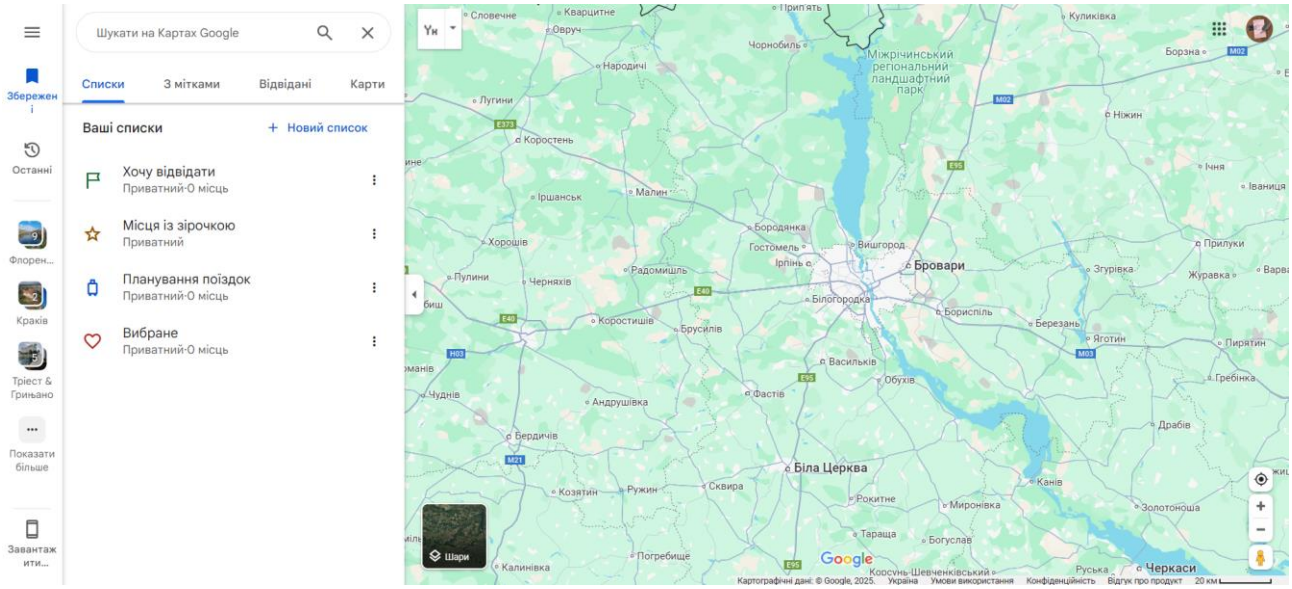


Рис. 1.3. Приклад екранної форми Google Maps

Основними перевагами вебзастосунку є:

- Величезна база даних місць. Google Maps містить інформацію про різноманітні локації по світі. Це робить його потужним інструментом для пошук та планування потенційних місць для відвідання;

- Можливість залишати відгуки та оцінки закладам. Ця можливість є дуже цінною, адже надає можливість користувачам обмінюватися досвідом того чи іншого місця;

- Візуальне представлення місць. Застосунок містить величезну кількість фотографій, що дозволяє користувачам отримати візуальне уявлення про локацію;

- Можливість створення списків та збереження місць. Ця функція є дуже корисною в плануванні подорожей, адже користувач можуть створювати та зберігати на карті власні списки цікавих місць, а також ділитися ними з іншими;

Хоча Google Maps є дуже популярним та корисним сервісом в плануванні подорожей, в контексті обміну досвідом про подорожі, він має обмеження:

- Фокус на окремих локаціях, а не на цілісних подорожах. Google Maps зосереджений саме на наданні інформації про конкретні місця, тому відгуки стосуються лише окремих закладів та пам’яток;

- Відсутність персональних профілів чи будь інструментів для хронології подорожжі;

- Великий ризик, що відгуки можуть бути суб’єктивними та інколи неправдивими;

- Інколи, результати пошуку інформації про окремі місця можуть бути перенавантажені рекламою та пропозиціями.

Таким чином, Google Maps є корисним інструментом для пошуку інформації про окремі місця, але його можливості для обміну повноцінним досвідом подорожей у вигляді розповідей, є обмеженим.

1.2.4 Travellerspoint

Travellerspoint позиціонує себе як соціальну мережу для мандрівників, що робить акцент саме на обміні досвідом подорожей та спілкуванні між користувачами [8].

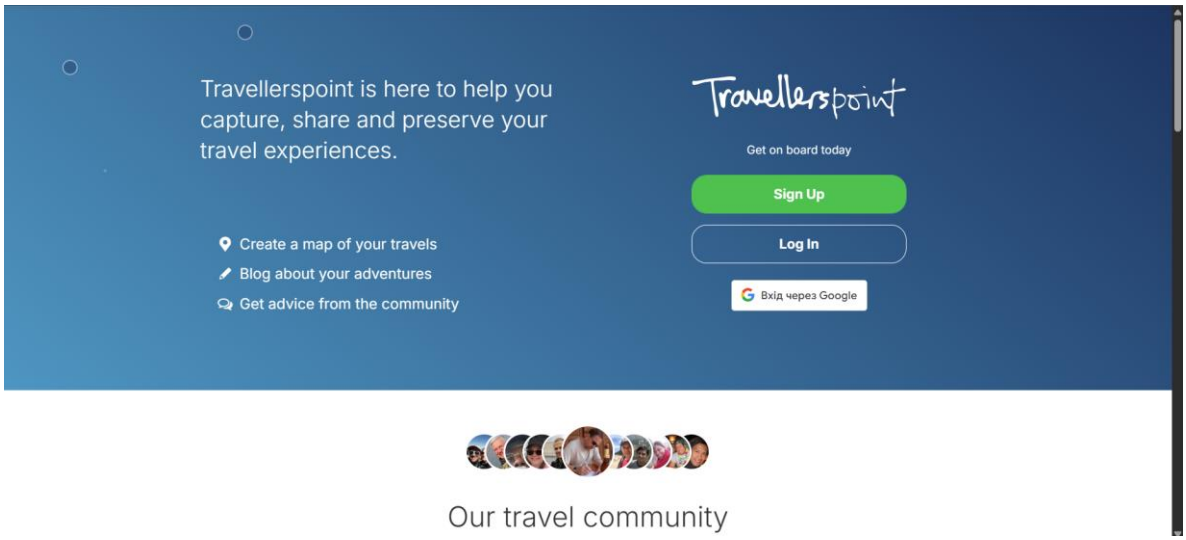


Рис. 1.4. Приклад екранної форми Travellerspoint

Основні переваги web-застосунку:

- Travellerspoint сприяє формуванню активної спільноти мандрівників, де користувачі охоче діляться своїми історіями, порадами та враженнями;

- Наявність персональних профілів з хронологією подорожей – користувачі можуть документувати свої подорожі, ведучи онлайн-щоденник мандрів;

- Travellerspoint має активні форуми та групи, де користувачі можуть ставити питання та отримувати поради.

Та все ж, Travellerspoint має також певні недоліки:

- Візуально застарілий інтерфейс, через що Travellerspoint є менш інтуїтивно зрозумілим;

- Обмін історіями з подорожей обмежується додаванням розповіді та фото;

- Обмежені можливості пошуку актуальної інформації. Нині, платформа є не такою популярною, через що активність користувачів знизилася, що знижує оперативність отримання потрібної та актуальної інформації.

У підсумку, можна сказати, що Travellerspoint є платформою, що робить сильний акцент на обміні досвідом подорожей та спілкуванні між мандрівниками. Однак, нині вона поступається сучасним сервісам у розмірі аудиторії та сучасності інтерфейсу.

1.2.5 Таблиця порівнянь аналогів

Зведені результати аналізу характеристик розглянутих web-застосунку наведено у таблиці 1.1.

Таблиця 1.1

Порівняння існуючих аналогів

Показник	**TripAdvisor**	**Wanderlog**	**Google Maps**	**Travellerspoint**	**Wayfora**
Обмін розгорнутими історіями	-	- (фокус більше в обміні коротких ідей та планів)	-	+	+
Персональні профілі з хронологією подорожей	-	+ (збереження майбутніх планів, та ідей, що публікувалися у спільноту)	-	+ (ведення онлайн-щоденника подорожей)	+
Візуалізація (наявність візуального контенту)	+	+	+	+	+
Актуальність інформації	+ (може бути перенасичення рекламою та неправдиві відгуки)	+	+ (може бути перенасичення рекламою та неправдиві відгуки)	- (залежить від активності спільноти)	+
Можливість взаємодії з контентом інших користувачів	+	-	-	- (не можна взаємодіяти в коментарях, але для обговорення є форуми)	+
Можливість планувати порожі	+	+	+ (можна створювати списки з місць, що хочеш відвідати)	-	+(можна створювати списки з місць, що хочеш відвідати)

Проведений порівняльний аналіз web-застосунків TripAdvisor, Wanderlog, Google Maps та Travellerspoint показує, що більшість сервісів роблять акцент на візуалізації та плануванні подорожей, проте можливості для обміну розгорнутими історіями та наявність персональних профілів з хронологією подорожей представлені менш широко. Взаємодія між користувачами та актуальність інформації також варіюються. Загалом, таблиця демонструє відсутність універсального рішення, що комплексно поєднує всі ключові аспекти обміну досвідом подорожей та планування.

2 ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Вимоги до програмного забезпечення

Дослідивши існуючі аналоги було сформовано основні функціональні вимоги:

1. Реєстрація та авторизація користувачів. Система повинна надавати користувачам можливість створювати нові облікові записи та безпечно авторизуватися для доступу до функціоналу застосунку.

2. Створення, редагування та видалення публікацій про подорожі. Користувачі повинні мати можливість створювати детальні публікації про свої подорожі, включаючи текст, фотографії та відвідані місця. Також повинна бути можливість редагувати та видаляти власні публікації.

3. Перегляд стрічки публікацій. Застосунок повинен відображати стрічку з публікаціями інших користувачів.

4. Пошук публікацій за назвою та локацією. Користувачі повинні мати можливість здійснювати пошук публікацій за ключовими словами в назві або за місцем, що згадується в публікації.

5. Збереження вподобаних місць для подальшого планування маршрутів подорожі. Користувачі повинні мати можливість зберігати цікаві місця, знайдені в публікаціях для подальшого використання при плануванні власних подорожей.

6. Профіль користувача. Кожен користувач повинен мати персональний профіль, де відображатимуться його власні публікації та колекції зі збереженими раніше місцями, які можуть використовуватися користувачем для планування майбутніх подорожей.

Нефункціональні вимоги визначають якість web-застосунку, впливаючи на зручність використання, надійність та ефективність його роботи. До даних вимог належать:

1. Зручність користування. Web-застосунок повинен мати інтуїтивно зрозумілий інтерфейс та просту навігацію, що забезпечує позитивний досвід для користувачів.

2. Захист даних користувачів. Система повинна забезпечувати належний рівень безпеки для персональних даних користувачів та їхнього контенту.

3. Масштабованість. Архітектура web-застосунку повинна бути гнучкою та модульною, що дозволить легко в майбутньому додавати нові функції та вносити зміни в існуючий функціонал без необхідності повного переписування коду.

4. Сумісність. Web-застосунок повинен забезпечувати коректне відображення та функціонування на різних популярних браузерах та на різних типах пристроїв різними розмірами екранів.

2.2 Проектування варіантів використання

Варіанти використання описують способи взаємодії користувача з web-застосунком для обміну досвідом подорожей та планування особистих маршрутів з метою задоволення їх потреб. Діаграма варіантів використання є цінним інструментом для візуалізації цих взаємодій та розуміння функціональних вимог системи з точки зору користувача.

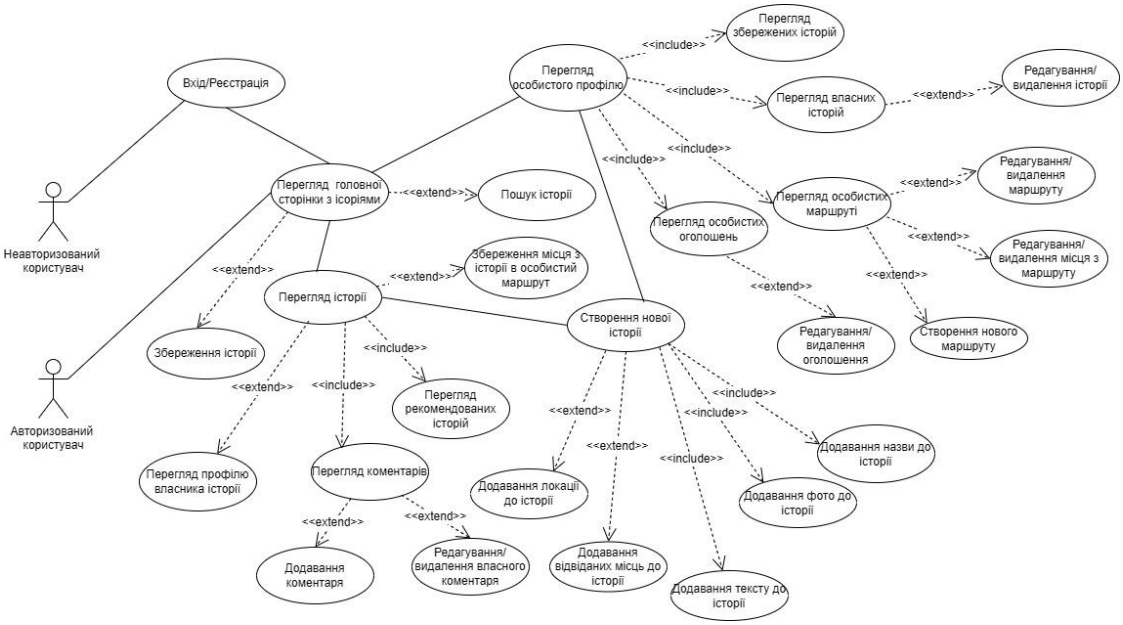


Рис. 2.1 Діаграма використання застосунку Wayfora

На даній діаграмі основним актором є користувач web-застосунку. Випадки використання включають такий перелік дій:

- Вхід/Реєстрація: Процес створення облікового запису або входу до системи для отримання доступу до функціональності платформи.
- Перегляд головної сторінки: Дозволяє авторизованому користувачеві переглядати історії подорожей інших користувачів та може розширюватися Пошуком історій.
- Перегляд історії: Дозволяє авторизованому користувачеві переглядати детальну інформацію про обрану історію та включає в себе Перегляд коментарів до історії, а також може розширюватися Збереженням історії, Переглядом рекомендованих історій, Збереження відмічених місць в особисті маршрути та Додаванням коментарів.
- Створення нової історії: Дозволяє авторизованому користувачеві створювати власні історії подорожей та включає в себе Додавання назви до історії, Додавання фото до історії та Додаванням тексту до історії, а також може розширюватися Додаванням локації до історії та Додаванням відвіданих місць до історії.
- Перегляд особистого профілю: Дозволяє авторизованому користувачеві переглядати свій профіль та включає Перегляд збережених історій, Перегляд власних історій та Перегляд особистих маршрутів, а також може розширюватися Редагуванням/видаленням власної історії, Редагуванням/видаленням маршруту та Видаленням місця з маршруту.

2.3 Архітектура web-застосунку

Web-застосунок для обміну досвідом подорожей та планування особистих маршрутів має клієнт-серверну архітектуру. Це означає, що він складається з двох основних частин, які взаємодіють між собою:

- Клієнтська частина: Розроблена з використанням бібліотеки React, ця частина відповідає за відображення користувацького інтерфейсу у web-

браузері користувача та обробку його взаємодії. Фронтенд надсилає запити до серверної частини для отримання та збереження даних, а також відображаю отриману інформацію користувачеві.

- Серверна частина: Розроблена з використанням Node.js та фреймворку Express.js, ця частина призначена для розміщення на сервері та відповідає за обробку бізнес-логіки застосунку, керування даними в базі даних MongoDB, забезпечення безпеки та надання API для взаємодії з клієнтською частиною. Обмін даними між клієнтом та сервером відбувається у форматі JSON через протокол HTTP/HTTPS.

Ця клієнт-серверна архітектура дозволяє розділити відповідальність між фронтендом та бекендом, що сприяє кращій організації коду, полегшує розробку та масштабування застосунку. React забезпечує швидкий інтерактивний інтерфейс для користувача, а Node.js/Express.js надають потужну та гнучку платформу для обробки серверних запитів та керування даними. MongoDB, як NoSQL база даних, забезпечує гнучкість у зберіганні різноманітних даних, пов’язаними з подорожами та маршрутами.

2.4 Проектування інтерфейсу користувача

При розробці інтерфейсу користувача (UI) та досвіду користувача (UX) web-застосунку для обміну досвідом подорожей та планування особистих маршрутів ключовим є створення інтуїтивно зрозумілого та простого у використанні середовища, доступного навіть для користувачів з різним рівнем технологічної підготовки. Логічна та передбачувана навігація, а також добре помітні ключові елементи є пріоритетом. Застосунок має забезпечувати зручність та ефективність у досягненні користувацьких цілей, мінімізуючи кількість необхідних кроків для обміну історіями, пошуку інформації та планування маршрутів. Візуальний стиль буде привабливим, сучасним та відповідатиме тематиці подорожей, використовуючи якісні зображення, гармонійну кольорову схему та читабельну типографіку для позитивного

сприйняття. Важливою є адаптивність інтерфейсу для коректного відображення та зручного використання на різних пристроях. В основі дизайну лежатиме орієнтація на потреби та очікування цільової аудиторії, з можливим проведенням досліджень користувачів для прийняття обґрунтованих дизайнерських рішень. Для проектування інтерфейсу користувача та досвіду користувача даного web-застосунку використовувався потужний інструмент Figma. Figma дозволяє ефективно створювати макети екранів, прототипи різного рівня деталізації, а також розробляти візуальний стиль застосунку. Використання Figma забезпечило гнучкість у процесі проектування та легкість у внесенні змін на різних етапах розробки.

Створені у Figma макети та прототипи слугували основою для візуалізації структури застосунку, розташування ключових елементів інтерфейсу та передбачуваної взаємодії користувача з системою.

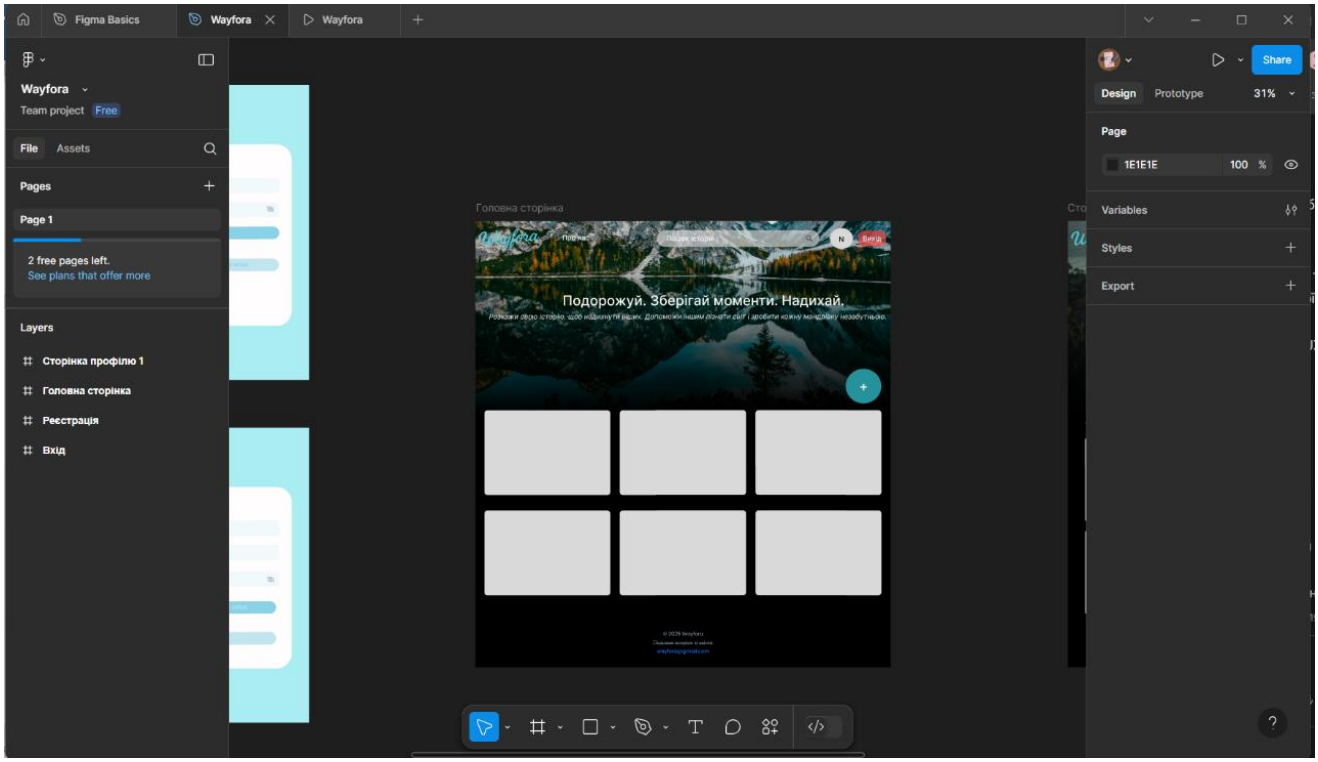


Рис. 2.2 Прототип головної сторінки у Figma

На головному екрані відображається стрічка з історіями подорожей. У верхній частині розміщено навігацію та пошук. Також є заклик до створення нової історії.

Сторінки реєстрації та авторизації надають користувачам можливість створити обліковий запис або увійти до існуючого за допомогою відповідних форм.

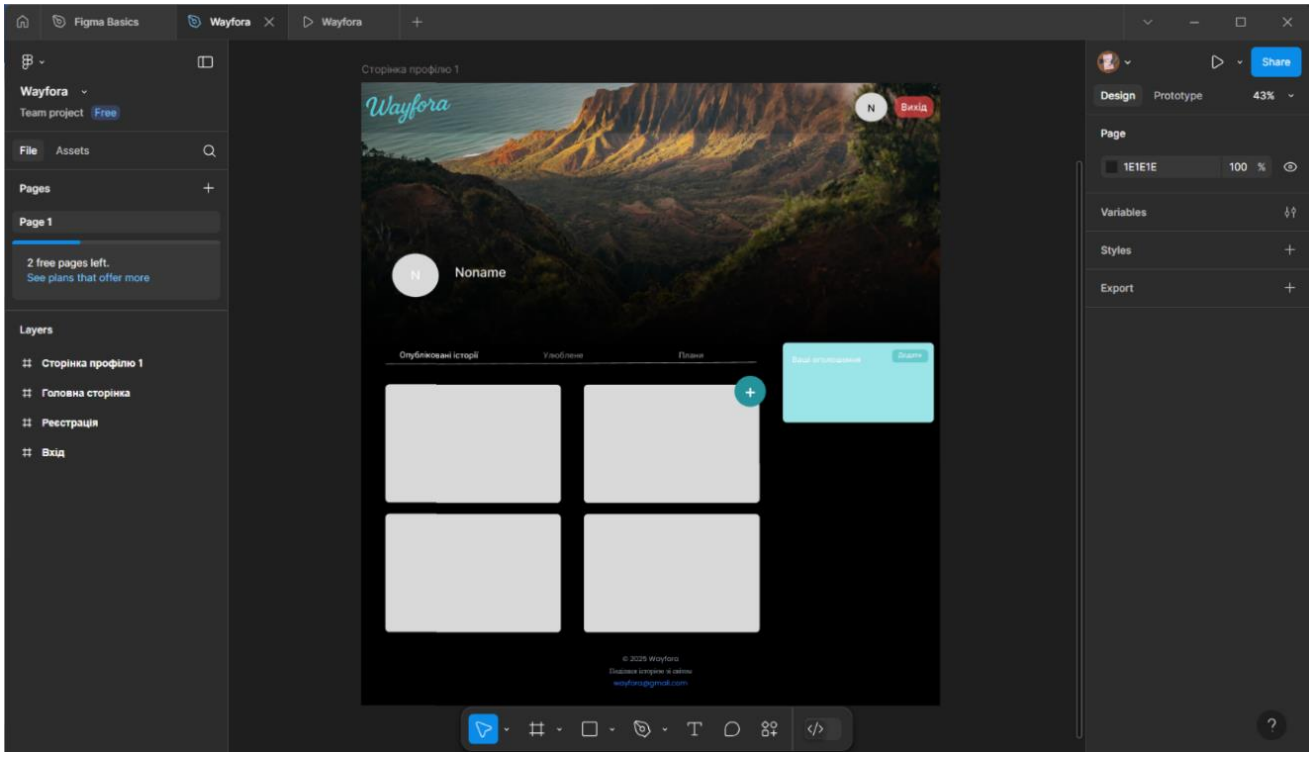


Рис. 2.3 Прототип сторінки особистого профілю у Figma

Сторінка особистого профілю містить інформацію про користувача, його опубліковані та збережені історії, а також створені маршрути.

Представлені прототипи основних екранів, розроблені у Figma, демонструють ключові елементи інтерфейсу та передбачувану взаємодію користувачів з web-застосунком. Дизайн орієнтований на забезпечення інтуїтивності, зручності та візуальної привабливості, що є важливим для залучення користувачів та сприяння активному обміну досвідом подорожей.

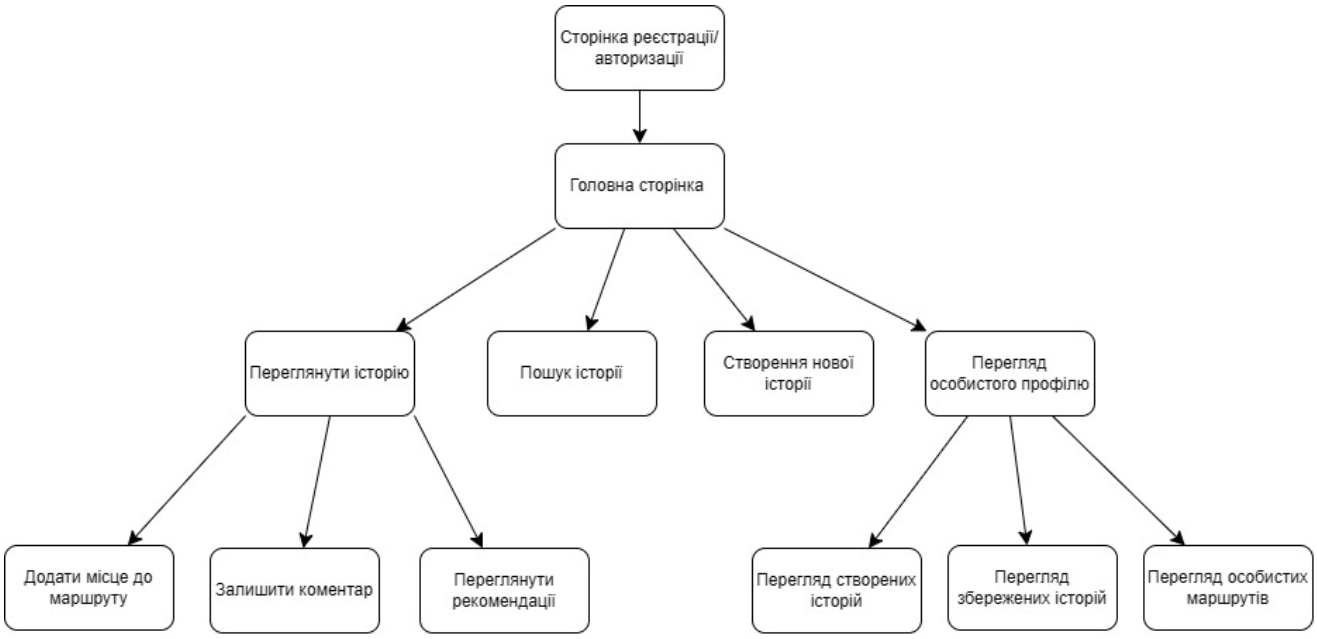


Рис. 2.4 Карта сайту

2.5 Проектування бази даних

Ефективне зберігання, впорядкування та надання доступу до інформації є критично важливим аспектом ари розробці будь-якого web-застосунку. У контексті створення платформи для обміну враженнями від подорожей та планування індивідуальних маршрутів, ключовим етапом є формування структури бази даних. Для задоволення потреб цього застосунку було визначено чотири основні колекції в обраній NoSQL базі даних MongoDB: User, Announcement, Collection та WayforaStory. Вибір саме цих колекцій обумовлений необхідністю оптимального зберігання та адміністрування відомостей, що стосуються користувачів, їхніх оголошень, створених особистих маршрутів та опублікованих історій подорожей.

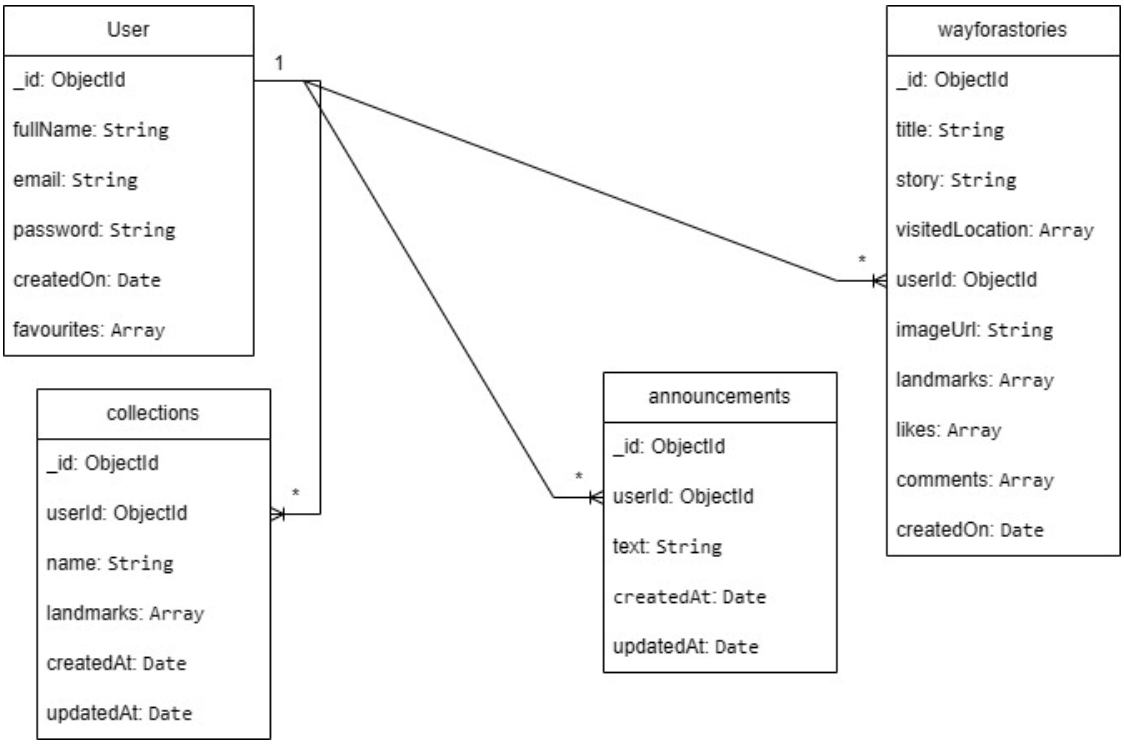


Рис. 2.5 Структура бази даних

Колекція User зберігає інформацію про користувачів додатку. Вона включає такі поля: _id (Унікальний ідентифікатор користувача, що генерується MongoDB), fullName (Повне ім’я користувача), email (Унікальна електронна пошта користувача), password (Хешований пароль користувача), createdOn (Дата та час створення облікового запису користувача), favourites (Масив ідентифікаторів історій подорожей, які користувач додав до списку улюблених). Колекція User дозволяє зберігати та керувати інформацією про користувачів, включаючи їхні облікові дані та вподобання. Існує логічний зв’язок «один до багатьох» між User та Announcement, між User та Collection, а також між User та WayforaStory.

Колекція Announcement зберігає інформацію про оголошення, які можуть створювати користувачі. Вона включає такі поля: _id (Унікальний ідентифікатор оголошення), userId (Ідентифікатор користувача, який створив оголошення), text (Текст оголошення), createdAt (Дата та час створення оголошення), updatedAt (Дата та час останнього оновлення оголошення). Колекція Announcement

забезпечує збереження та управління інформацією про оголошення, пов’язані з конкретними користувачами.

Колекція Collection зберігає інформацію про особисті маршрути, які створюють користувачі. Вона включає такі поля: _id (Унікальний ідентифікатор особистого маршруту), userId (Ідентифікатор користувача, який створив маршрут, name (Назва особистого маршруту), landmarks (Масив об’єктів, що описують окремі місця в маршруті), createdAt (Дата та час створення маршруту), updatedAt (Дата та час останнього оновлення маршруту). Колекція Collection дозволяє користувачам зберігати та організовувати інформацію про цікаві для них місця у вигляді особистих маршрутів.

Колекція WayforaStory зберігає інформацію про історії подорожей, які публікують користувачі. Вона включає такі поля: _id (Унікальний ідентифікатор подорожі), title (Заголовок історії подорожі), story (Основний текст історії подорожі), visitedLocation (Масив назв відвіданих місць), userId (Ідентифікатор користувача, який опублікував історію), imageUrl (URL головного зображення історії), landmarks (Масив об’єктів, що описують згадані в історії визначені місця), likes (Масив ідентифікаторів користувачів, які зберегли цю історію), comments (Масив об’єктів коментарів до історії), createdOn (Дата та час публікації історії подорожі). Колекція WayforaStory забезпечує збереження та управління контентом, яким діляться користувачі, включаючи текст, зображення, інформацію про місця та взаємодії з іншими користувачами.

Обрані колекції та їхні поля дозволяють ефективно зберігати та керувати інформацією, необхідною для функціонування web-застосунку для обміну досвідом подорожей та планування особистих маршрутів. Вони забезпечують чітке розділення даних за користувачами, оголошеннями, особистими маршрутами та історіями подорожей, що сприяє підвищенню продуктивності та зручності роботи з базою даних.

3 ВИБІР ЗАСОБІВ ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

3.1 Мова програмування JavaScript

JavaScript являє собою кросплатформну та динамічно типізовану мову програмування, яка є однією з фундаментальних технологій web-розробки [9, 10]. Її здатність виконуватися в різних середовищах, від браузерів до серверів, робить її надзвичайно універсальною. Динамічна типізація, де типи змінних визначаються під час виконання, , забезпечує швидкість розробки, але водночас покладає на розробника відповідальність за контроль типів для забезпечення стабільності програми.

Однією з ключових можливостей, є її підтримка асинхронного програмування. Механізм колбеків, промісів та синтаксис async/await дозволяють ефективно обробляти тривалі операції не блокуючи основний потік виконання програми. Це критично важливо для забезпечення чуйності інтерфейсу користувача та продуктивності серверних застосунків. Екосистема JavaScript є однією з найбільших та найактивніших у світі. Мільйони бібліотек та фреймворків, доступних через npm (Node Package Manager), значно спрощують та прискорюють розробку, надаючи готові рішення для широкого спектра завдань – від маніпуляцій DOM до роботи з API та базами даних.

JavaScript підтримує кілька парадигм програмування, включаючи об’єктивно-орієнтоване (через прототипне наслідування та класи, введені в ECMAScript 2015), функціональне (завдяки функціям першого класу та замиканням) та імперативне програмування. Прототипне наслідування є унікальною особливістю JavaScript, де об’єкти можуть успадковувати властивості від інших об’єктів через ланцюжок прототипів. Стандартизація JavaScript здійснюється через ECMAScript, специфікацію, що постійно розвивається та оновлюється новими можливостями, що робить мову ще потужнішою та зручнішою для розробників.

У висновку, можна сказати, що універсальність JavaScript, її потужна екосистема та підтримка асинхронності зробили її оптимальним вибором для розробки як клієнтської, так і серверної частин застосунку, забезпечуючи консистентність технологічного стеку та полегшуючи розробку.

3.2 Front-end розробка

Фронтенд частина застосунку була розроблена з використанням технологій, спрямованих на створення інтерактивного та ефективного користувацького досвіду.

React – це бібліотека JavaScript для побудови UI [11]. В основі React лежить ідея компонентів – незалежних блоків коду, які керують певними частинами інтерфейсу. Компоненти можуть бути як функціональними (прості JavaScript-функції), так і класовими (з використанням синтаксису класів ES6).

Ключовою концепцією React є віртуальний DOM. Замість безпосередньої маніпуляції з реальним DOM браузера, React створює його віртуальну копію. Коли стан компонента змінюється, React спочатку оновлює віртуальний DOM, потім порівнює його з попередньою версією та визначає мінімальну кількість змін, які необхідно внести до реального DOM. Цей процес оптимізує продуктивність та робить оновлення інтерфейсу більш ефективним.

React використовує JSX (JavaScript XML) – розширення синтаксису JavaScript, яке дозволяє писати HTML-подібну розмітку безпосередньо в коді JavaScript. JSX потім транспілюється в звичайний JavaScript, що полегшує опис структури UI. Управління станом є важливою частиною React-розробки. Компоненти можуть мати власний локальний стан, а для управління станом на рівні всього застосунку можуть використовуватися такі рішення як Context API або сторонні бібліотеки.

Можна підсумувати, що компонентна структура React та ефективний механізм оновлення DOM дозволили створити інтерактивний та швидкодіючий

інтерфейс користувача для обміну досвідом подорожей та планування маршрутів.

Для збірки фронтенду використовувався інструмент Vite, який використовує переваги нативних ES Modules у браузерах під час розробки [12]. Коли браузер запитує модуль, Vite лише трансформує та обслуговує запитуваний код «на вимогу», без необхідності попередньої збірки всього застосунку. Це призводить до надзвичайно швидкого холодного старту сервера розробки. Використання Vite значно прискорило процес розробки фронтенду завдяки швидкому запуску сервера та оптимізованій збірці, що підвищило продуктивність розробки.

Також використовувався Tailwind CSS – фреймворк CSS, що використовує утилітний підхід [13]. Замість традиційного написання CSS-класів, які описують зовнішній вигляд певного елемента, Tailwind надає набір низькорівневих класів, кожен з яких відповідає за певну властивість CSS. Комбінуючи ці класи безпосередньо в HTML/JSX, розробники можуть швидко стилізувати елементи без перемикання між файлами HTML та CSS. Tailwind CSS базується на системі дизайну, що включає палітри кольорів, типографіку, відстані тощо, які можна налаштувати відповідно до потреб проєкту. Фреймворк також підтримує адаптивний дизайн через префікси. Таким чином Tailwind CSS дозволив швидко та консистентно стилізувати інтерфейс користувача, значно прискоривши розробку візуальної частини застосунку.

Axios – HTTP-клієнт для JavaScript, який працює як у браузері, так і в Node.js. Axios надає зручний API для здійснення асинхронних HTTP-запитів до серверів. Ключові особливості Axios включають автоматичне перетворення даних JSON, можливість налаштування таймаутів запитів та підтримку перехоплення запитів та відповідей. Можна підсумувати, що Axios забезпечив надійну та зручну взаємодію між клієнтською та серверною частинами застосунку через HTTP-запити, спростивши обмін даними.

Окрім цього, використовувалася бібліотека для декларативної маршрутизації в React-застосунках, що працюють у браузері – React Router DOM.

Вона надає набір компонентів та хуків, які дозволяють визначати зв’язок між URL-адресами та компонентами, що відображаються. React Router DOM використовує історію браузера для керування навігацією, дозволяючи користувачам використовувати кнопки «назад» та «вперед» браузера. Він також підтримує вкладені маршрути, динамічні сегменти URL та lazy loading компонентів для покращення продуктивності. Отже React Router DOM забезпечив клієнтську маршрутизацію, дозволяючи користувачам переміщуватися між різними розділами застосунку без повного перезавантаження сторінки, що покращило загальний досвід користування.

3.3 Back-end розробка

Серверна частина застосунку була розроблена для обробки запитів клієнта, керування даними та забезпечення API для front-end. Перш за все було використано Node.js – середовище виконання JavaScript, яке дозволяє запускати JavaScript-код поза браузером [14,15]. Однією з ключових особливостей Node.js є його подій-орієнтована, неблокуюча модель вводу-виводу. Це означає, що замість того, щоб блокувати виконання потоку при очікуванні завершення операцій вводу-виводу, Node.js реєструє колбек, який буде виконано після завершення операції. Це дозволяє одному потоку обробляти велику кількість одночасних з’єднань, що робить Node.js добре придатним для розробки масшабованих мережевих застосунків.

Node.js має вбудовані модулі для роботи з файловою системою, мережею, шляхами, криптографією тощо. Екосистема npm надає доступ до величезної кількості сторонніх пакетів для розширення функціональності. Внутрішньо Node.js використовує Event Loop (цикл подій), реалізований у бібліотеці libuv, для керування асинхронними операціями. Коли виникає асинхронна операція, вона передається в libuv, а основний потік JavaScript продовжує виконання. Після завершення операцій, колбек поміщається в чергу подій, і Event Loop на наступній ітерації передає його на виконання у стек викликів. Node.js також

включає механізм Garbage Collection, який автоматично звільняє пам’ять, що більше не використовується програмою. Як висновок, можна сказати, що використання Node.js на сервері дозволило застосувати знання JavaScript для всього стеку розробки, а його неблокуюча модель забезпечила високу продуктивність та масштабованість серверної частини.

Також використовувався Express.js – мінімалістичний вебфреймворк для Node.js [16]. Express.js надає базову структуру для організації web-застосунку та API. Він базується на концепції middleware – функцій, які мають доступ до об’єкта запиту (req), об’єкта відповіді (res) та наступної middleware-функції в циклі запит-відповідь застосунку. Middleware можуть виконувати різноманітні завдання, такі як логування, автентифікація, обробка тіла запиту тощо. Маршрутизація в Express.js визначається за допомогою методів HTTP (GET, POST, PUT, DELETE, тощо) та шляхів URL. Розробники визначають функції-обробники для кожного маршруту, які виконуються при отриманні відповідного запиту. Отож, Express.js надав гнучкий та зручний інструментарій для створення RESTful API, необхідного для взаємодії клієнтської та сервер частини застосунку.

Для MongoDB, використовується ODM бібліотека Mongoose, що працює в середовищі Node.js. Mongoose надає абстракцію над нативним драйвером MongoDB, спрощуючи взаємодію з базою даних. Основною концепцією Mongoose є схеми, які визначають структуру документів у колекціях MongoDB, включаючи типи даних, валідацію та значення за замовчуванням. На основі схем створюються моделі, які є класами, що представляють колекції та надають API для виконання операцій з базою даних. Mongoose також підтримує middleware на рівні документів та моделей, що дозволяє виконувати певний код до або після подій, таких як збереження або видалення документів. Таким чином, Mongoose спростив роботу з базою даних MongoDB, надаючи зручні інструменти для моделювання даних та виконання запитів, що підвищило ефективність розробки серверної частини.

Для криптографічного хешування паролів використовувалася бібліотека bcrypt. Алгоритм bcrypt є адаптивним, тобто кількість ітерацій хешування може бути збільшена з часом для збереження стійкості до атак зі зростанням обчислювальної потужності. bcrypt також додає випадкову «сіль» до кожного пароля перед хешуванням, що запобігає використання попередньо обчислених хешів. Використання bcrypt забезпечило надійне та безпечне зберігання паролів користувачів у базі даних, що є критично важливим для захисту облікових даних.

Для забезпечення безпеки використовуються JSON Web Tokens (JWT) — стандартний спосіб створення токенів доступу для аутентифікації та авторизації. JWT забезпечує безпечну передачу інформації між клієнтом та сервером у вигляді JSON-об'єктів, дозволяючи перевіряти особу користувачів та надавати доступ до ресурсів без зберігання сеансів на сервері. У web-застосунку це використовується для безпечної аутентифікації. Використання JWT позитивно впливає на продуктивність та масштабованість застосунку, дозволяючи ефективно обробляти велику кількість запитів та інтегруватися з іншими сервісами. JWT легко інтегрується з різними мовами та фреймворками. Як підсумок, можна сказати, що JSON Web Tokens були використані для реалізації безпечної аутентифікації та авторизації користувачів, забезпечуючи захист доступу до ресурсів застосунку.

Також використовувався multer – middleware для Node.js, що полегшує обробку даних, що передаються у форматі multipart/form-data, який використовується для передачі файлів через HTTP-форми. При розробці web-застосунку, multer дозволив ефективно обробляти завантаження файлів користувачами, що є необхідною функціональністю для обміну історіями подорожей.

3.4 База даних

Для зберігання даних web-застосунку обрано NoSQL документо-орієнтовану базу даних MongoDB [17,18]. Вона зберігає дані у гнучких JSON-

подібних документах (BSON) в колекціях, на відміну від таблиць реляційних баз даних. Відсутність строгої схеми забезпечує гнучкість при роботі з різноманітними даними профілів, подорожей та маршрутів.

MongoDB підтримує різні типи даних, вкладені документи та масиви, що спрощує моделювання зв'язків. Для масштабованості та доступності використовуються реплікація (копіювання даних) та шардування (розподіл даних). MongoDB надає потужну мову запитів, підтримує індекси для швидкості пошуку та можливості агрегації даних для аналізу.

Можна зробити висновок, що гнучкість, масштабованість та ефективна робота з неструктурованими даними MongoDB стали оптимальним вибором для зберігання інформації про подорожі та маршрути користувачів, спостивши розробку та адаптацію структури даних.

3.5 Інструменти розробки

Для забезпечення ефективного та якісного процесу розробки web-застосунку було залучено ряд інструментів, серед яких особливу увагу заслуговують Visual Studio Code та Postman.

Visual Studio Code зарекомендував себе як потужне та водночас легке інтегроване середовище розробки, що стало основним інструментом для написання, редагування та налагодження коду [19]. VS Code вирізняється своєю кросплатформністю та надзвичайно широкою підтримкою різноманітних мов програмування, включаючи JavaScript, HTML, CSS, а також фреймворків та бібліотек, що використовується в проєкті, таких як React та Node.js. Однією з ключових переваг VS Code є його інтелектуальна система автодоповнення коду IntelliSence, яка значно прискорює процес розробки, пропонуючи релевантні підказки та допомагаючи уникнути синтаксичних помилок. Підсвічування синтаксису робить код більш читабельним та структурованим, що полегшує його розуміння та підтримку. Вбудований термінал дозволив розробникам виконувати команди операційної системи безпосередньо з IDE, спрощуючи роботу з системами контролю версій, а також з іншими інструментами

командного рядка, необхідними для збірки та запуску проєкту. Потужний налагоджувач VS Code надав можливість детально відстежувати виконання коду, встановлювати точки зупину, переглядати значення змінних та стеків викликів, що є незамінним при виявленні та усуненні помилок. Крім того завдяки величезній екосистемі розширень, VS Code можна гнучко налаштовувати під конкретні потреби проєкту, додаючи підтримку лінтерів, форматерів коду, інструментів для роботи з базами даних та багатьох інших корисних функцій. Отже, завдяки своїй потужній функціональності, гнучкості та зручності використання, VS Code став ключовим інструментом, який значно підвищив ефективність розробки клієнтської та серверної частин web-застосунку.

Для забезпечення якісної розробки та тестування API, що є критично важливим для взаємодії між front-end та back-end, активно використовувався інструмент Postman [20]. Ця платформа надала розробникам інтуїтивно зрозумілий графічний інтерфейс для створення, надсилання та аналізу HTTP-запитів до API ендпоінтів серверної частини застосунку. Postman підтримує всі основні HTTP-методів (GET, POST, PUT, DELETE тощо) та дозволяє налаштовувати різноманітні параметри запитів, включаючи заголовки, параметри URL, тіло запиту в різних форматах (JSON, XML, form-data). Отримані від сервера відповіді відображалися у зручному форматі, що дозволяло аналізувати коди статусу, заголовки та тіло відповіді. Особливо корисним виявилися можливості Postman зі створення та організації колекцій запитів, що дозволило систематизувати тестування різних частин API та ділитися цими колекціями між членами громади. Використання змінних середовища спростило тестування API в різних конфігураціях, а можливість написання автоматизованих тестів дозволила забезпечити стабільність та надійність роботи API. Крім того, Postman слугував інструментом для документування API, генеруючи зрозумілу документацію на основі створених колекцій запитів. Таким чином Postman став незамінним помічником у процесі розробки та тестування API, забезпечивши ефективну взаємодію між клієнтською та серверною частинами web-застосунку та сприяючи його стабільній роботі.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ WEB-ЗАСТОСУНКУ

4.1 Структура проєкту

Організація структури проекту є фундаментальним аспектом розробки будь-якого web-застосунку. Чітка та логічна структура не лише полегшує навігацію кодовою базою та розуміння принципів її роботи, але і сприяє підтримці та масштабуванню застосунку. Представлений web-застосунок розроблено за архітектурою full-stack, де клієнтська частина реалізована за допомогою бібліотеки React, а серверна частина – на базі середовища виконання Node.js з використанням фреймворку Express. Для зберігання та керування даними використовується NoSQL база даних MongoDB.

Основна структура проєкту розділена на дві головні частини: клієнтську (frontend) та серверну (backend), кожна з яких має свою внутрішню організацію, оптимізовану для відповідних технологій та завдань.

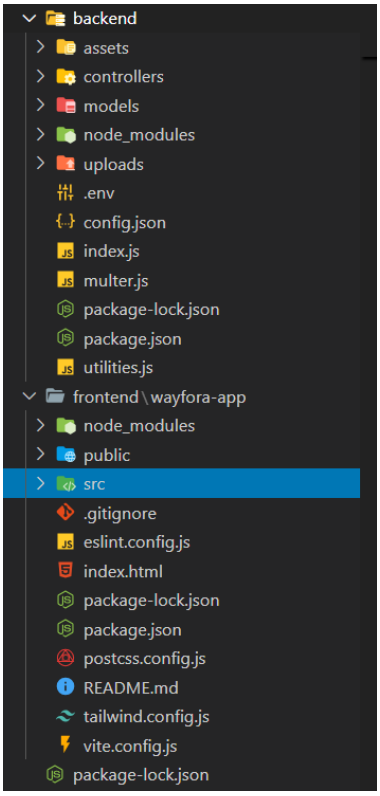


Рис. 4.1 «Wayfora» (структура web-застосунку)

Клієнтська частина проєкту розташована у директорії «frontend/wayfora-app/src» та включає наступні ключові директорії та файли.

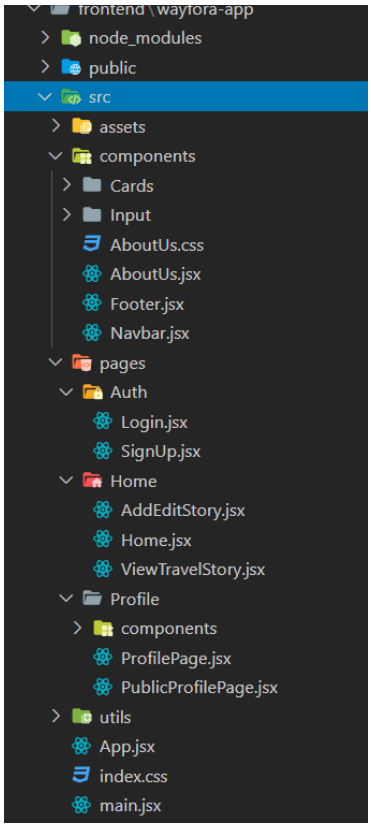


Рис. 4.2 Клієнтська частина проєкту

Директорія «components» містить багаторазові UI-компоненти, які використовуються для побудови інтерфейсу користувача. Компоненти згруповано за їхнім призначенням, наприклад у директорії «Cards» знаходяться компоненти картки для відображення інформації профілів та історій подорожей. Директорія «Input» містить компоненти для введення даних. Також в цій директорії розташовані загальні UI-елементи, такі як навігаційна панель, футер та компонент «Про нас».

Директорія «pages» містить компоненти, які відповідають за окремі сторінки web-застосунку. Для зручності їх згруповано за функціональними областями. Директорія «Auth» включає компоненти для аутентифікації користувачів. Директорія «Home» містить компоненти, пов’язані з головною

сторінкою. Директорія «Profile» містить компоненти для відображення профілю користувача та вкладені компоненти в директорії «components/» для різних секцій профілю.

У директорії «utils» зберігаються утилітарні функції та константи, що використовуються в усьому застосунку. Це включає конфігурацію екземплярів Axios для HTTP-запитів, загальні константи, допоміжні функції та функціональність для завантаження зображень. Директорія «assets» призначена для зберігання статичних ресурсів, таких як зображення та інші активи.

Файл «App.jsx» є кореневим компонентом застосунку, що визначає його загальну структуру. Файл «main.jsx» є точкою входу для React-застосунку та відповідає за його рендеринг у DOM. Також наявні файли конфігурації інструментів звірки та стилізації, глобальні стилі застосунку та стандартні файли керування проєктом.

Серверна частина проєкту розташована у кореневій директорії backend та має наступну структуру.

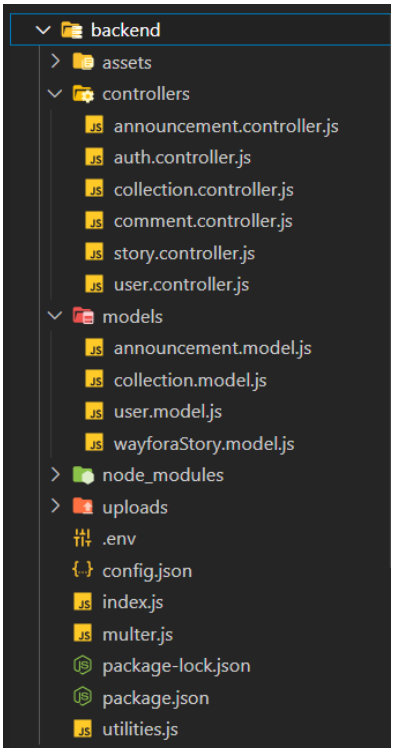


Рис. 4.3 Серверна частина проєкту

Директорія «controllers» містить контролери, які відповідають за обробку вхідних HTTP-запитів та взаємодію з моделями даних. Кожен контролер відповідає за певну сутність або групу сутностей. Директорія «models» містить схеми моделей даних, які використовуються для взаємодії з базою даних MongoDB. Тут визначено структури документів для таких сутностей, як оголошення, колекції, користувачі та історії подорожей. Файл «index.js» є головною точкою входу для серверної частини web-застосунку. Він відповідає за запуск сервера Express, підключення необхідних middleware та визначення основних маршрутів. Також тут містяться інші стандартні файли керування проєктом, також файли з різноманітними допоміжними функціями та утилітами.

Представлена структура проєкту є логічною та забезпечує чіткий поділ відповідальності між клієнтською та серверною частинами. Клієнтська частина відповідає за відображення інтерфейсу користувача та взаємодію з ним, а серверна частина обробляє бізнес-логіку, керує даними та надає API для клієнта. Такий поділ сприяє кращій організації коду, полегшує розробку та тестування окремих частин застосунку, а також забезпечує гнучкість при подальшому масштабуванні.

4.2 Розробка серверної частини

Серверна частина web-застосунку є ключовим компонентом, що відповідає за оброку запитів від клієнтської сторони, керування даними, їх зберігання в базі даних MongoDB та надання API для взаємодії з клієнтською частиною, розробленою на React. Сервер виконується на базі середовища Node.js з використанням фреймворку Express.

Для забезпечення належної функціональності серверної частини було встановлено ряд важливих бібліотек та інструментів. Серед них ключову роль відіграє бібліотека bcrypt, яка використовувалася для безпечного хешування паролів користувачів перед їхнім збереження у базі даних. Основою для створення вебсервера та API заклав потужний фреймворк express. Для реалізації

безпечної аутентифікації та авторизації користувачів застосовувалася бібліотека jsonwebtoken, що дозволяє створювати та верифікувати токени. Взаємодія з базою даних MongoDB здійснювалася за допомогою асинхронної бібліотеки mongoose, яка надає інструменти для моделювання даних та виконання запитів. Також використовувався middleware multer для обробки завантажених файлів та інструмент nodemon, який автоматично перезапускає сервер при зміні файлів коду.

Після встановлення необхідних бібліотек наступним важливим кроком є розробка структури даних. Для цього за допомогою бібліотеки Mongoose були визначені схеми моделей, які відображають організацію даних у базі даних MongoDB. Кожна модель відповідає окремій колекції у базі даних та визначає типи даних, обов’язковість полів та зв’язки між різними сутностями застосунку.

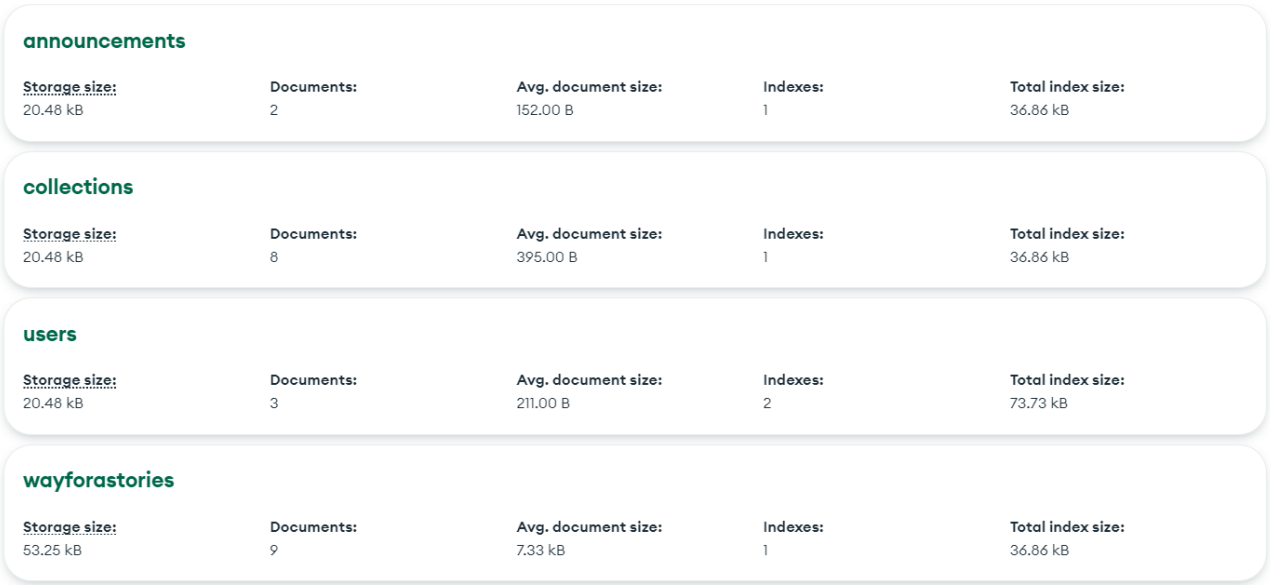


Рис. 4.4 Дані з бази даних MongoDB

Модель оголошення (Announcement) використовується для зберігання інформації про оголошення, які можуть створювати користувачі. Основні поля включають userId (послання на користувача, що створив оголошення), text (текст оголошення) та часові мітки createdAt і updatedAt.

Модель колекцій (Collection) дозволяє користувачам створювати власні колекції з визначених місць. Основні поля включають userId (посилання на

користувача, який створив колекцію), name (назва колекції) та масив landmarks, що містить інформацію про кожне збережене місце в колекції. Також присутні часові мітки createdAt і updatedAt.

Модель користувача (User), відповідає за зберігання інформації про користувачів застосунку. Основні поля включають fullName (повне ім'я), email (електронна пошта), password (хешований пароль), createdOn (дата створення облікового запису) та масив favourites, що містить посилання на улюблені історії користувача.

Модель історії подорожі (WayforaStory) є центральною для зберігання історій подорожей користувачів. Основні поля включають title (заголовок історій), story (текст історії), visitedLocation (масив відвіданих місць), userId (посилання на автора історії), imageUrl (URL головного зображення), масив вкладених об'єктів landmarks (інформація про визначні місця в історії), масив likes (посилання користувачів, які зберегли історію в улюблені), масив вкладених об'єктів comments (коментарі до історії) та createdOn (дата створення історії).

Контролери відповідають за обробку вхідних HTTP-запитів, взаємодію з моделями даних для виконання необхідних операцій та формування відповідей для клієнтського застосунку. Кожен контролер об'єднує логіку, пов'язану з певною сутністю або групою функціональностей застосунку.

Контролер оголошень (announcement.controller.js) забезпечує керування оголошеннями користувачів (див. рисунок 4.5). Він включає функціональність для створення нового оголошення, отримання всіх оголошень конкретного користувача, оновлення існуючого оголошення та видалення оголошення. Важливою особливістю є перевірка прав користувача при створені, оновленні та видаленні оголошень, щоб користувач міг маніпулювати лише власними оголошеннями.

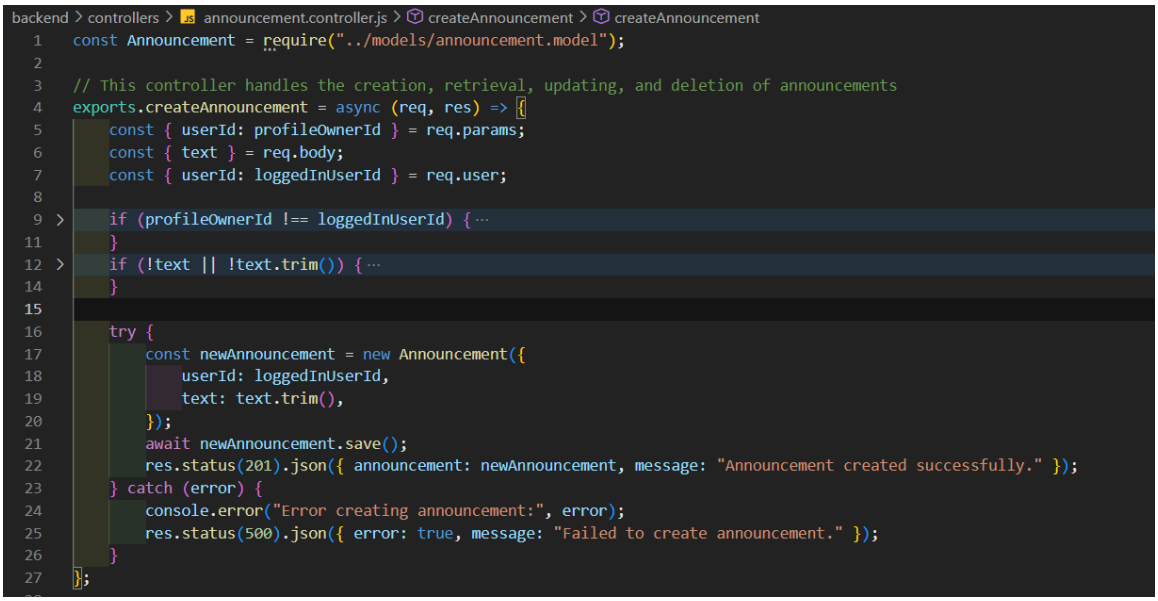
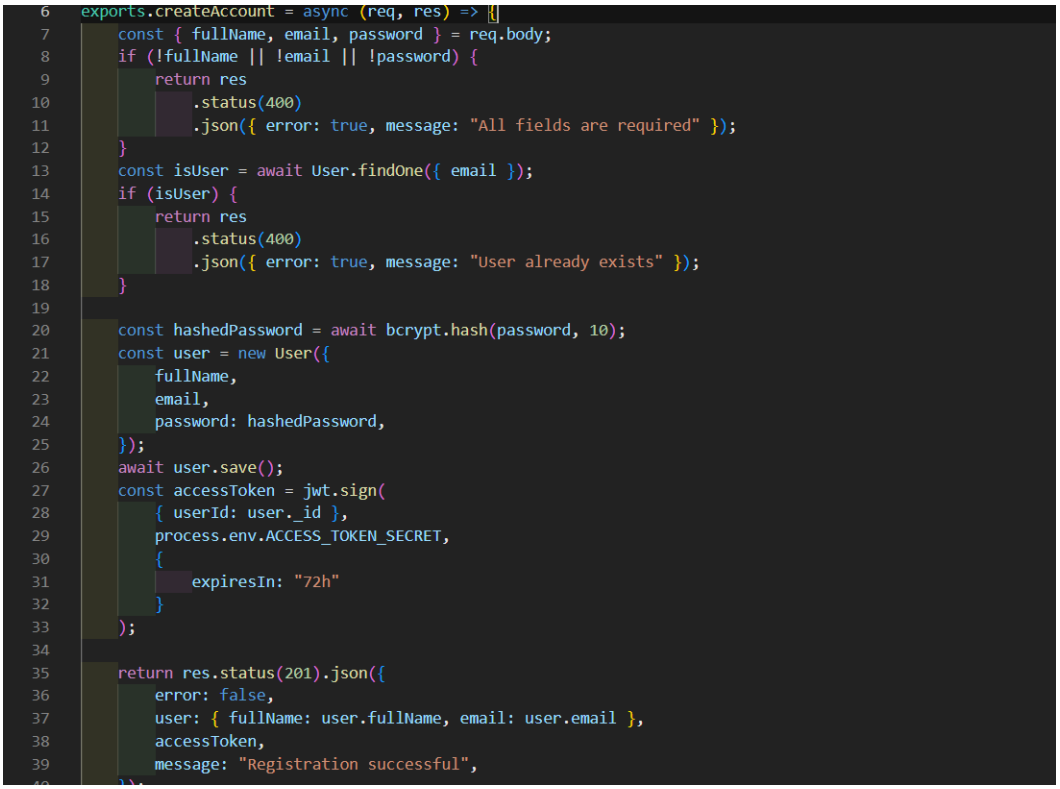


Рис. 4.5 Контролер оголошень

Контролер аутентифікації (auth.controller.js) реалізує основні процеси реєстрації та входу користувача в систему. Метод createAccount відповідає за реєстрацію нового користувача, включаючи валідацію вхідних даних, хешування пароля за допомогою бібліотеки bcrypt та збереження користувача в базі даних.

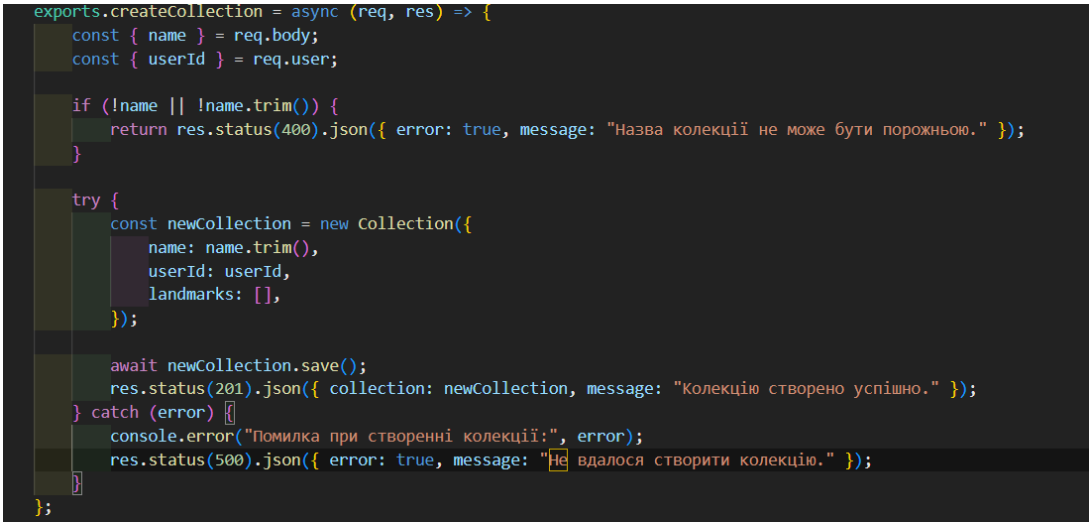
Після успішної реєстрації або входу, користувачеві виділяється JWT токен, який використовується для подальшої аутентифікації. Метод getUser дозволяє отримати інформацію про поточного авторизованого користувача на основі наданого токена.

Контролер колекцій (collection.controller.js) надає користувачам можливість організовувати визначні місця у власні колекції (див. рисунок 4.6). Він включає методи для створення нової колекції, отримання всіх колекцій авторизованого користувача, додавання нового місця до колекції, редагування назви колекції, видалення колекції, отримання всіх місць з певної колекції та видалення окремого місця з колекції. Для забезпечення цілісності даних та прав користувачів, більшість методів контролюють приналежність колекцій авторизованому користувачеві.



```
exports.createAccount = async (req, res) => {
    const { fullName, email, password } = req.body;
    if (!fullName || !email || !password) {
        return res
            .status(400)
            .json({ error: true, message: "All fields are required" });
    }
    const isUser = await User.findOne({ email });
    if (isUser) {
        return res
            .status(400)
            .json({ error: true, message: "User already exists" });
    }

    const hashedPassword = await bcrypt.hash(password, 10);
    const user = new User({
        fullName,
        email,
        password: hashedPassword,
    });
    await user.save();
    const accessToken = jwt.sign(
        { userId: user._id },
        process.env.ACCESS_TOKEN_SECRET,
        {
            expiresIn: "72h"
        }
    );

    return res.status(201).json({
        error: false,
        user: { fullName: user.fullName, email: user.email },
        accessToken,
        message: "Registration successful",
```

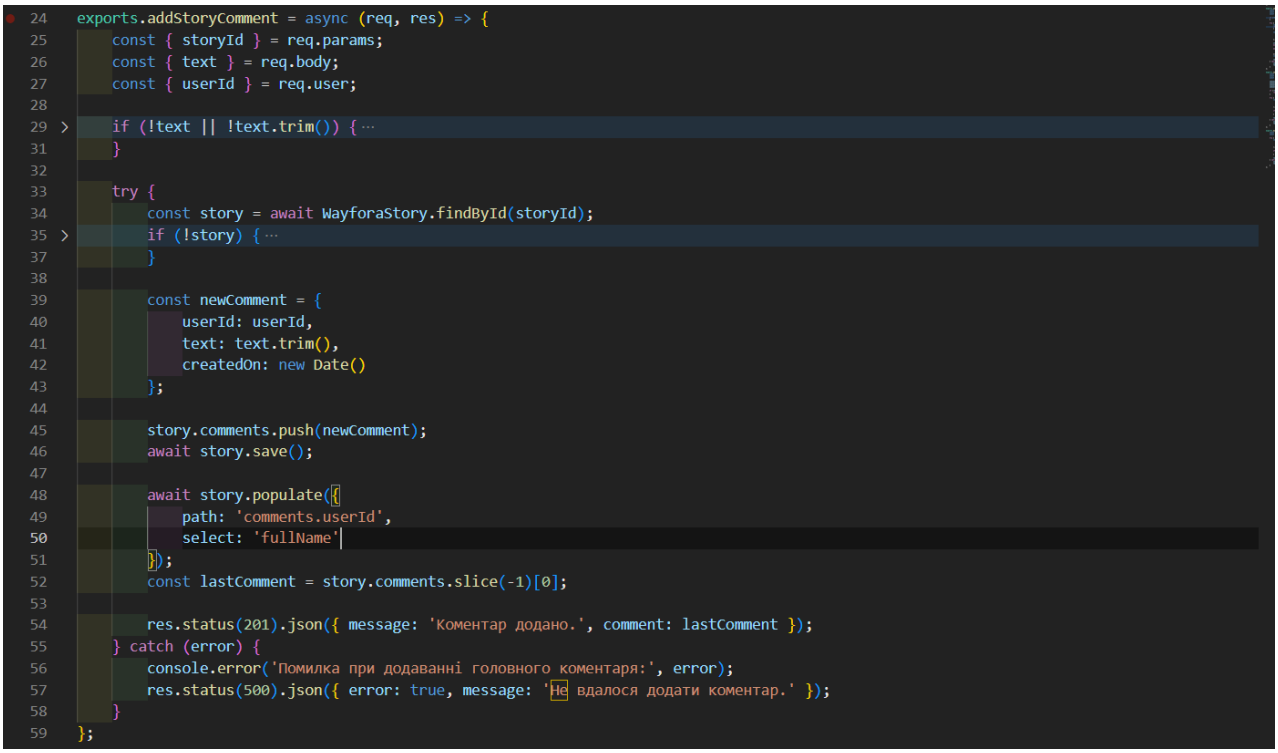
Рис. 4.6 Контролер аутентифікації



```
exports.createCollection = async (req, res) => {
    const { name } = req.body;
    const { userId } = req.user;

    if (!name || !name.trim()) {
        return res.status(400).json({ error: true, message: "Назва колекції не може бути порожньою." });
    }

    try {
        const newCollection = new Collection({
            name: name.trim(),
            userId: userId,
            landmarks: [],
        });

        await newCollection.save();
        res.status(201).json({ collection: newCollection, message: "Колекцію створено успішно." });
    } catch (error) {
        console.error("Помилка при створенні колекції:", error);
        res.status(500).json({ error: true, message: "Не вдалося створити колекцію." });
    }
};
```

Рис. 4.7 Контролер колекцій

Контролер коментарів (comment.controller.js) відповідає за функціональність коментування історій подорожей (див. рисунок 4.8). Він включає методи для отримання всіх коментарів до певної історії, додавання нового коментаря, редагування існуючого коментаря та видалення коментаря. При редагуванні та видалені коментарів також здійснюється перевірка авторства.



```
exports.addStoryComment = async (req, res) => {
    const { storyId } = req.params;
    const { text } = req.body;
    const { userId } = req.user;

    if (!text || !text.trim()) {···
    }

    try {
        const story = await WayforaStory.findById(storyId);
        if (!story) {···
        }

        const newComment = {
            userId: userId,
            text: text.trim(),
            createdOn: new Date()
        };

        story.comments.push(newComment);
        await story.save();

        await story.populate({
            path: 'comments.userId',
            select: 'fullName'
        });
        const lastComment = story.comments.slice(-1)[0];

        res.status(201).json({ message: 'Коментар додано.', comment: lastComment });
    } catch (error) {
        console.error('Помилка при додаванні головного коментаря:', error);
        res.status(500).json({ error: true, message: 'Не вдалося додати коментар.' });
    }
};
```

Рис. 4.8 Контролер коментарів

Контролер історій подорожей (story.controller.js) є одним з ключових контролерів застосунку, оскільки він забезпечує основну функціональність, пов’язану з історіями користувачів. Він включає методи для завантаження зображень та їх видалення, додавання нової історії, отримання всіх історій, отримання історій конкретного користувача, редагування існуючої історії, видалення історії та додавання або видалення історії зі списку улюблених. Також реалізовано пошук історій за ключовим словом. Для забезпечення безпеки та цілісності даних, при редагуванні та видалені історій перевіряється їх приналежність авторизованому користувачеві.

```
exports.addWayforaStory = async(req, res) => {
    const { title, story, visitedLocation, imageUrl, landmarks } = req.body;
    const { userId } = req.user;

    if (!title || !story || !visitedLocation || !imageUrl) {
        return res.status(400).json({ error: true, message: "All fields are required" });
    }

    try {
        const wayforaStory = new WayforaStory({
            title,
            story,
            visitedLocation,
            userId,
            imageUrl,
            landmarks: landmarks ? landmarks.map(landmark => ({
                name: landmark.name,
                link: landmark.link,
                note: landmark.note,
                imageUrl: landmark.imageUrl
            })) : [],
        });

        await wayforaStory.save();
        res.status(201).json({ story: wayforaStory, message: "Added successfully" });
    } catch (error) {
        res.status(400).json({ error: true, message: error.message });
    }
};
```

Рис. 4.9 Контролер історій подорожей

Контролер профілю (profile.controller.js) надає функціональність для керування профілями користувачів.

Для забезпечення взаємодії між клієнтською та серверною частинами web-застосунку було розроблено RESTful API. Файл index.js у корені серверної частини відповідає за визначення маршрутів (кінцевих точок), до яких клієнтський застосунок надсилає HTTP-запити. Кожен маршрут пов'язаний з певним контролером, який обробляє запит та повертає відповідь. Аутентифікація більшості захищених маршрутів здійснюється за допомогою JWT токенів, які передаються в заголовку Authorization.

```
backend > index.js > ...
25  app.post("/create-account", authController.createAccount);
26  app.post("/login", authController.login);
27  app.get("/get-user", authenticateToken, authController.getUser);
28
29  app.post("/image-upload", upload.single("image"), storyController.imageUpload);
30  app.delete("/delete-image", storyController.deleteImage);
31  app.post("/add-wayfora-story", authenticateToken, storyController.addWayforaStory);
32  app.get("/get-all-stories", storyController.getAllStories);
33  app.get("/get-user-stories/:userId", authenticateToken, storyController.getUserStories);
34  app.put("/edit-story/:id", authenticateToken, storyController.editStory);
35  app.delete("/delete-story/:id", authenticateToken, storyController.deleteStory);
36  app.put("/update-is-favourite/:id", authenticateToken, storyController.updateIsFavourite);
37  app.delete('/remove-favourite-story/:storyId', authenticateToken, storyController.removeFavouriteStory);
38  app.get("/get-favourite-stories/:userId", authenticateToken, storyController.getFavouriteStories);
39  app.get("/search", authenticateToken, storyController.searchStories);
40
41  app.post("/create-collection", authenticateToken, collectionController.createCollection);
42  app.get("/get-user-collections", authenticateToken, collectionController.getUserCollections);
43  app.post("/add-to-collection/:collectionId", authenticateToken, collectionController.addToCollection);
44  app.put('/edit-collection/:collectionId', authenticateToken, collectionController.editCollection);
45  app.delete('/delete-collection/:collectionId', authenticateToken, collectionController.deleteCollection);
46  app.get('/get-collection-landmarks/:collectionId', authenticateToken, collectionController.getCollectionLandmarks);
47  app.delete('/delete-landmark-from-collection/:collectionId/:landmarkId', authenticateToken, collectionController.deleteLandmarkFromColl
```

Рис. 4.10 Приклад маршрутів API серверної частини

Блок маршрутів AuthController відповідає за функціональність аутентифікації користувачів, включаючи реєстрацію нових облікових записів, вхід існуючих користувачів у систему та отримання інформації про авторизованого користувача за допомогою токена.

Маршрути, пов'язані з StoryController, забезпечують усі операції з історіями подорожей користувачів, починаючи від завантаження зображень та створення нових історій, закінчуючи їхнім переглядом (як всіх, так і окремих користувачів), редагуванням, видаленням, а також керуванням улюбленими історіями та пошуком серед них.

Функціональність керування колекціями визначних місць реалізована в блоці маршрутів CollectionController, що дозволяє користувачам створювати власні колекції, переглядати їх, додавати та видаляти місця, а також редагувати та видаляти самі колекції.

Маршрути UserController відповідають за надання інформації про профіль користувача, включаючи публічні дані та дані про відвідані й заплановані країни, а також за оновлення цієї інформації. Блок маршрутів AnnouncementController забезпечує можливість створення, перегляду, редагування та видалення оголошень користувачів. І нарешті, маршрути CommentController відповідають за функціональність коментування історій подорожей, дозволяючи отримувати, додавати, редагувати та видаляти коментарі. Окрім цього, визначено маршрути для надання доступу до статичних файлів, таких як завантажені зображення (/uploads) та інші ресурси застосунку (/assets).

Усі описані маршрути відіграють ключову роль у забезпеченні повноцінної функціональності web-застосунку, надаючи користувачам можливість взаємодіяти з платформою. Маршрути для реєстрації та авторизації, створення, редагування та видалення історій подорожей, керування колекціями та профілями користувачів, а також для взаємодії з оголошеннями та коментарями, були розроблені з метою інтеграції з клієнтською частиною застосунку.

4.3 Розробка клієнтської частини

Клієнтська частина web-застосунку визначає те, як користувач взаємодіє з додатком у своєму браузері. Вона відповідає за відображення інтерфейсу, обробку дій користувача та обмін даними з сервером. Для побудови сучасних та інтерактивних клієнтських застосунків часто використовують JavaScript-фреймворки, такі як React. React дозволяє розробникам створювати інтерфейс з окремих, багаторазових компонентів, забезпечуючи ефективне оновлення сторінки та високу продуктивність.

Кореневим компонентом застосунку є App.jsx. Він відіграє ключову роль у налаштуванні маршрутизації за допомогою React Router та визначенні загального макету сторінки. App.jsx об'єднує різні компоненти сторінок, забезпечуючи таким чином переходи між ними та відображення відповідного контенту.

```
const App = () => {
    return (
        <div>
            <Router>
                <Routes>
                    <Route path="/" exact element={<Root />} />
                    <Route path="/dashboard" exact element={<Home />} />
                    <Route path="/login" exact element={<Login />} />
                    <Route path="/signup" exact element={<SignUp />} />
                    <Route path="/profile" element={<ProfilePage />} />
                    <Route path="/user/:userId/public-profile" element={<PublicProfilePage />} />
                </Routes>
            </Router>
        </div>
    );
}
```

Рис. 4.11 Налаштування маршрутизації у App.jsx

Коли користувач вперше відкриває застосунок, він потрапляє на сторінку аутентифікації. Цей етап обробляється компонентами, розташованими в директорії Auth. На сторінці реєстрації користувач може створити новий обліковий запис. Компонент містить форму для введення особистих даних, таких як ім'я, електронна пошта та пароль. Після заповнення форми та натискання кнопки "Зареєструватися", дані відправляються на сервер для обробки.

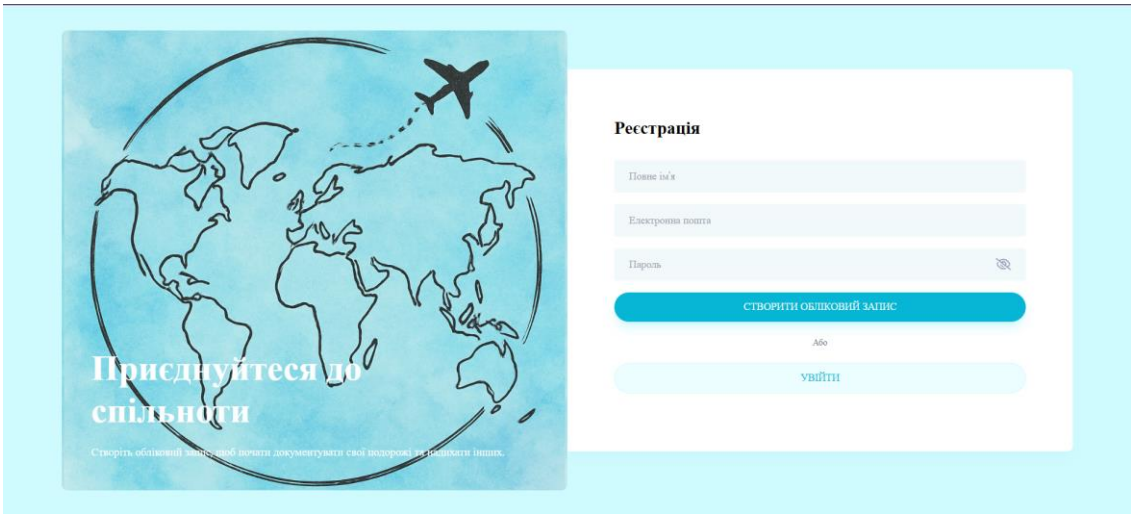


Рис. 4.12 Сторінка реєстрації користувача

Якщо користувач вже має обліковий запис, він може увійти на сайт через цю сторінку. Подібно до сторінки реєстрації, тут є форма для введення облікових даних. Після успішного входу користувач перенаправляється на головну сторінку.

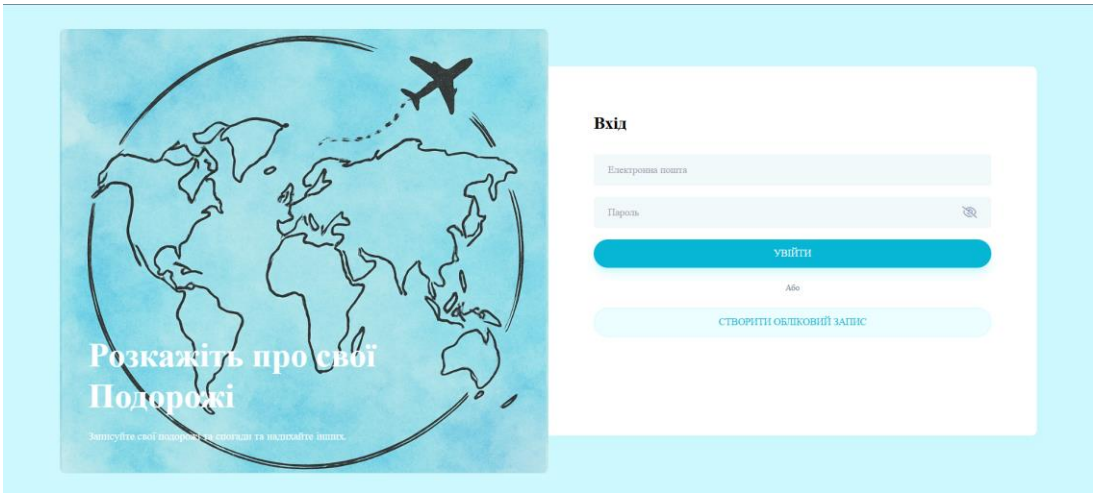


Рис. 4.13 Сторінка авторизації користувача

Після успішної аутентифікації користувач потрапляє на головну сторінку застосунку. Ця сторінка, представлена компонентом Home.jsx, є своєрідним центром контенту. Тут відображаються історії подорожей, якими діляться інші користувачі.

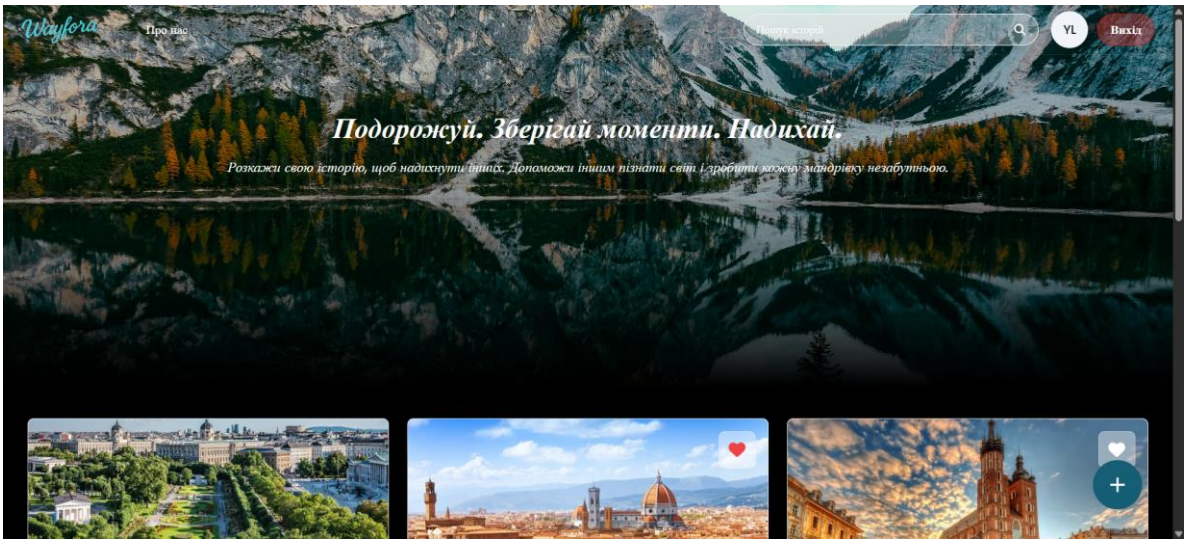


Рис. 4.14 Головна сторінка

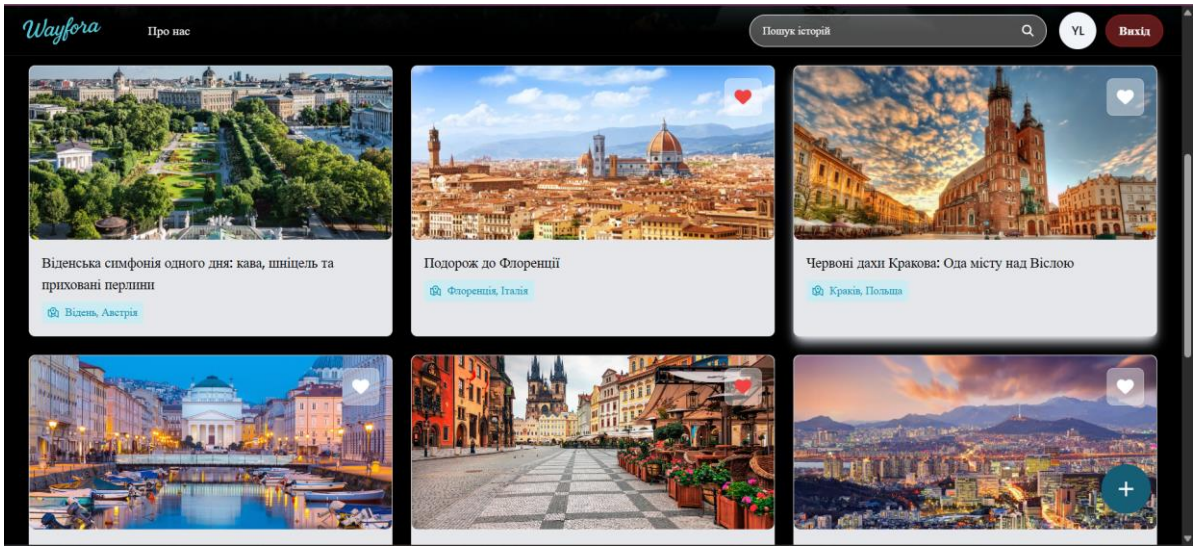


Рис. 4.15 Відображення історій на головній сторінці

Home.jsx може відображати історії у вигляді карток, забезпечуючи зручний перегляд. Кожна історія містить заголовок, локацію та зображення. З цієї сторінки користувач може перейти до детального перегляду конкретної історії, натиснувши на відповідну картку, або створити нову історію, натиснувши на кнопку праворуч. Натиснувши на кнопку на самій історії, вподобану історію можна зберегти в профілі, щоб не загубити її та мати можливість переглядати і надалі.

```
const addNewTravelStory = async () => {
    try {
        let imageUrl = "";
        if (storyImg) {
            const imgUploadsRes = await uploadImage(storyImg);
            imageUrl = imgUploadsRes.imageUrl || "";
        }

        const payload = {
            title,
            story,
            imageUrl,
            visitedLocation: visitedLocations,
            landmarks,
            createdOn: moment().toISOString(),
        };

        const response = await axiosInstance.post("/add-wayfora-story", payload);

        if (response.data && response.data.story) {
            toast.success("Історію успішно додано!");
            getAllStories();
            onClose();
        }
    } catch (error) {
        console.error("Помилка при додаванні історії:", error);
        toast.error("Не вдалося додати історію. Спробуйте ще раз.");
    }
};
```

Рис. 4.16 Реалізація функції addNewTravelStory, що додає нову історію

Також, на головній сторінці можна здійснювати пошук за назвою або локацією.

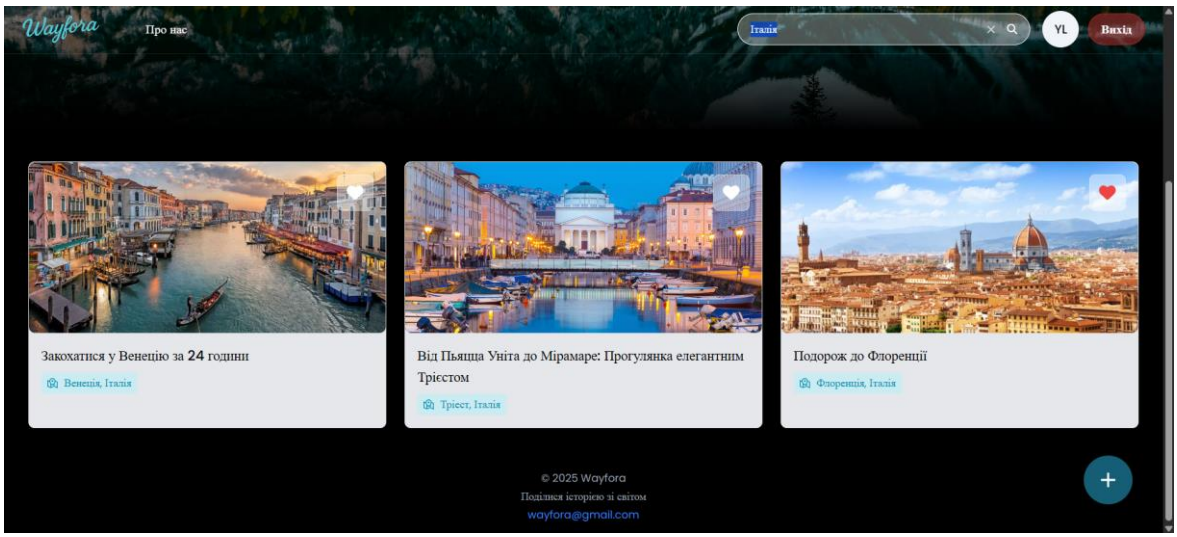


Рис. 4.17 Демонстрація роботи пошуку

Коли користувач вибирає конкретну історію на головній сторінці, відкривається сторінка детального перегляду. Компонент ViewTravelStory.jsx

відповідає за відображення повного змісту історії, включаючи заголовок та текст історії, зображення, інформація про автора, дату публікації, місце подорожі та додані місця. Цей компонент забезпечує занурення в історію, дозволяючи користувачам отримати максимум інформації про подорож.

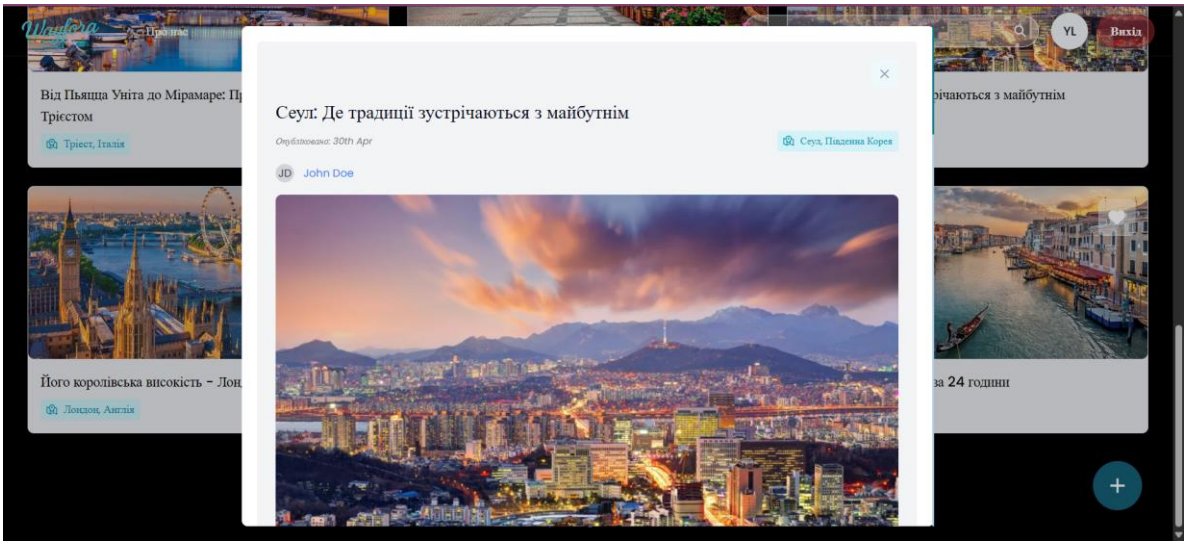


Рис. 4.18 Перегляд обраної історії

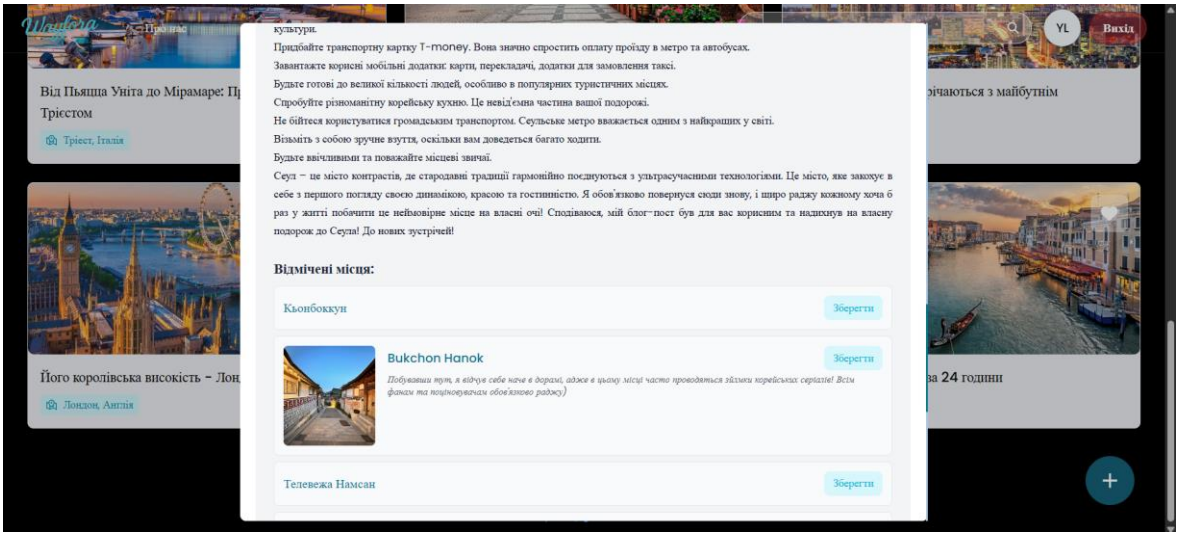


Рис. 4.19 Перегляд відмічених місць в історії

Якщо користувачу сподобалося відмічене місце і він хоче зберегти його для планування маршруту майбутньої подорожі, то достатньо натиснути кнопку «Зберегти» поряд з бажаним місцем. Користувач може зберегти місце до існуючої колекції місць або створити нову.

```
const saveLandmarkToCollection = async (collectionId) => {
    try {
        const response = await axiosInstance.post(`/add-to-collection/${collectionId}`, {
            landmark: selectedLandmark,
        });
        if (response.data && response.data.message) {
            toast.success(`Мітку "${selectedLandmark.name}" додано до колекції!`);
            closeCollectionModal();
        }
    } catch (error) {
        console.error('Error saving landmark to collection:', error);
        toast.error('Не вдалося додати мітку до колекції.');
    }
};
```

Рис. 4.20 Реалізація функції, що додає обране місце до колекції

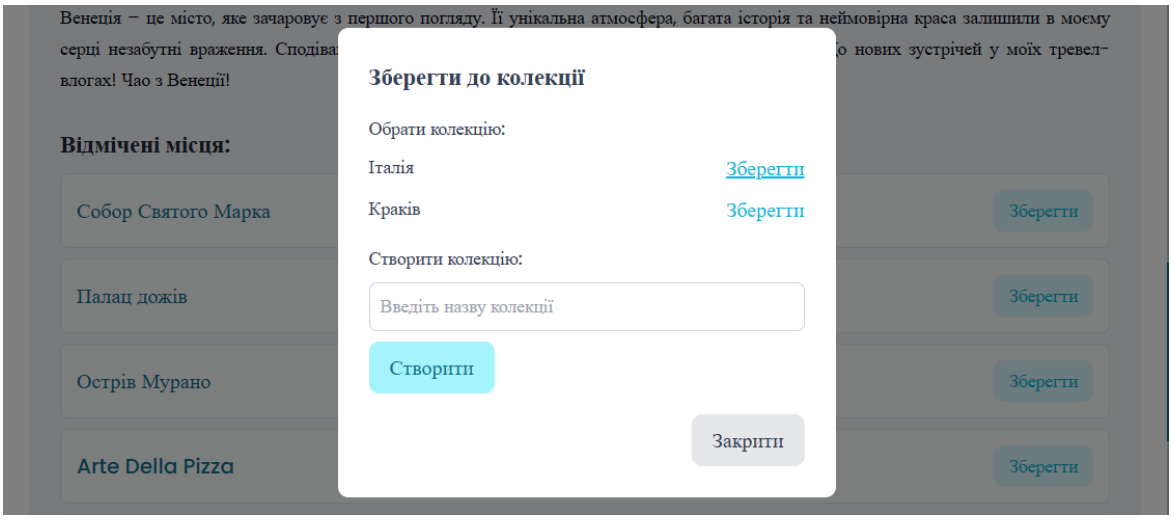


Рис. 4.21 Демонстрація функції збереження місць до колекції

Користувачі також можуть взаємодіяти між собою залишаючи коментарі з питаннями до автора, або просто обговорюючи історію.

```
const handleAddComment = async () => {
    if (newComment.trim() && storyInfo?._id && currentUserId) {
        try {
            const response = await axiosInstance.post(`/api/stories/${storyInfo._id}/comments`, {
                text: newComment,
                userId: currentUserId,
            });
            if (response.data?.comment) {
                setComments(prevComments => [...prevComments, response.data.comment]);
                setNewComment('');
            }
        } catch (error) {
            console.error('Помилка при додаванні коментаря:', error);
            toast.error('Не вдалося додати коментар.');
        }
    }
};
```

Рис. 4.22 Реалізація функції для додавання коментаря до історії

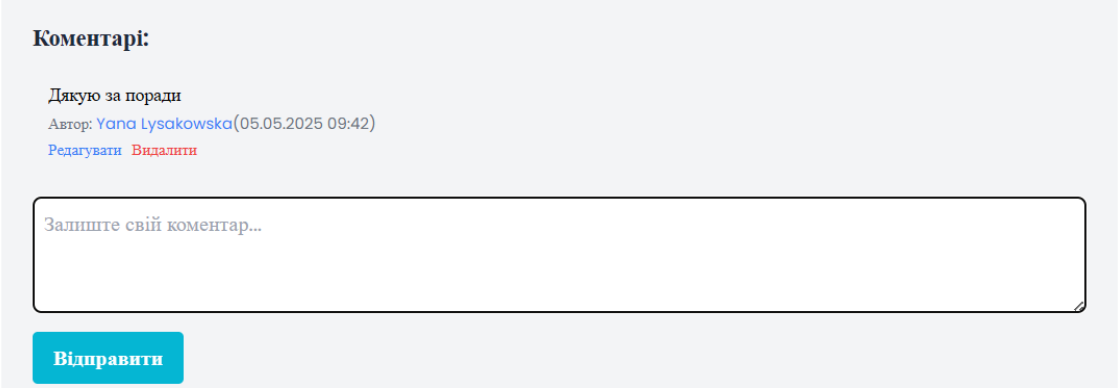


Рис. 4.23 Демонстрація можливості додавання коментарів

На основі схожих локацій, також реалізовані рекомендації подібних історій, які можуть сподобатися користувачам.

```
const getRecommendedStories = (currentStory, allStories, currentUserId) => {
    if (!currentStory || !allStories || !currentUserId) {
        return [];
    }

    const { visitedLocation } = currentStory;
    if (!visitedLocation || visitedLocation.length === 0) {
        return [];
    }

    return allStories.filter(story =>
        story._id !== currentStory._id &&
        story.userId !== currentUserId &&
        story.visitedLocation.some(location => visitedLocation.includes(location))
    ).slice(0, 3);
};
```

Рис. 4.24 Реалізація функції для відображення рекомендацій

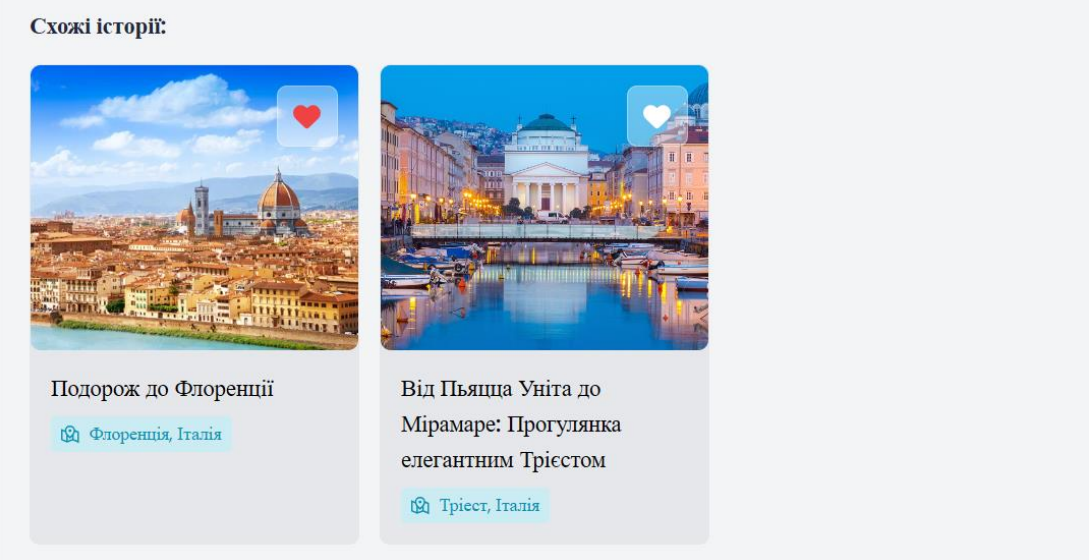


Рис. 4.25 Демонстрація відображення рекомендованих історій

Додати нову історію можна як з головної сторінки, так і з сторінки особистого профілю. Натискаючи кнопку Додати історію користувачу надають поля для вводу всіх необхідних відомостей про історію.

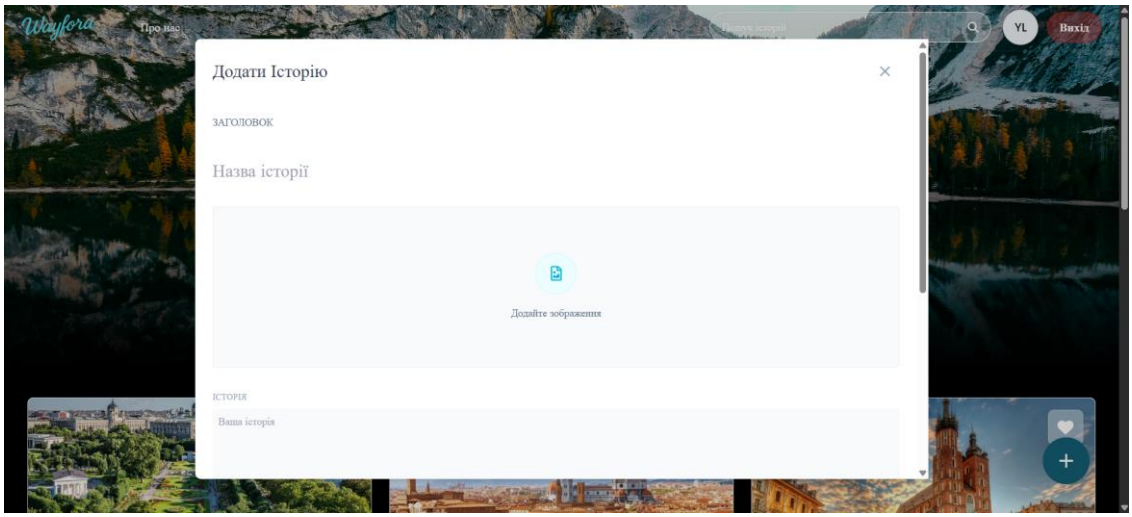


Рис. 4.26 Вікно для додавання нової історії

Особистий профіль користувача є централізованим місцем для кожного користувача web-застосунку, де вони можуть керувати своєю присутністю на платформі. Ця сторінка надає користувачам можливість створювати оголошення, переглядати та редагувати всі свої опубліковані історії подорожей, а також переглядати власні заплановані маршрути на основі збережених місць.

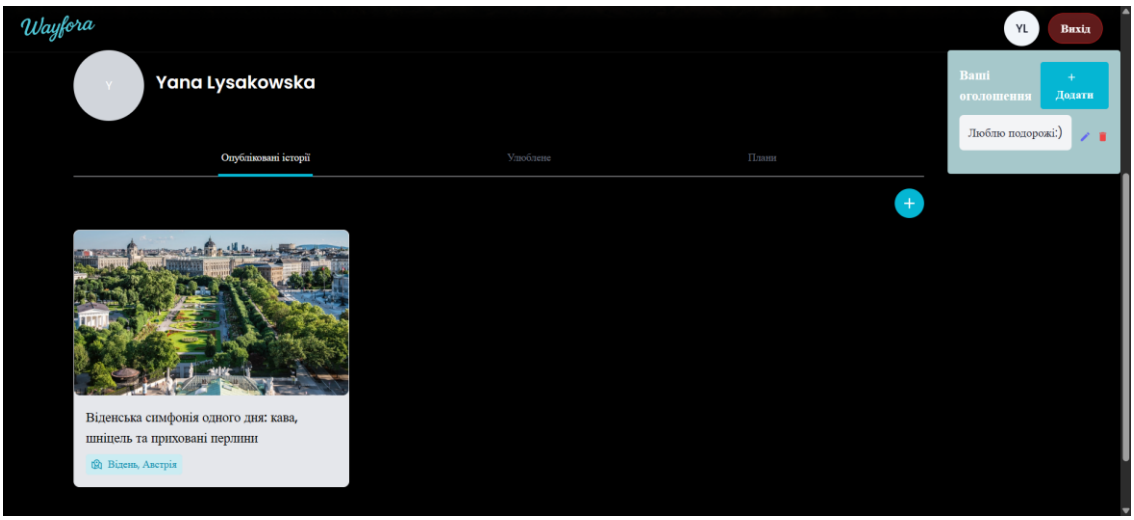


Рис. 4.27 Особистий профіль користувача

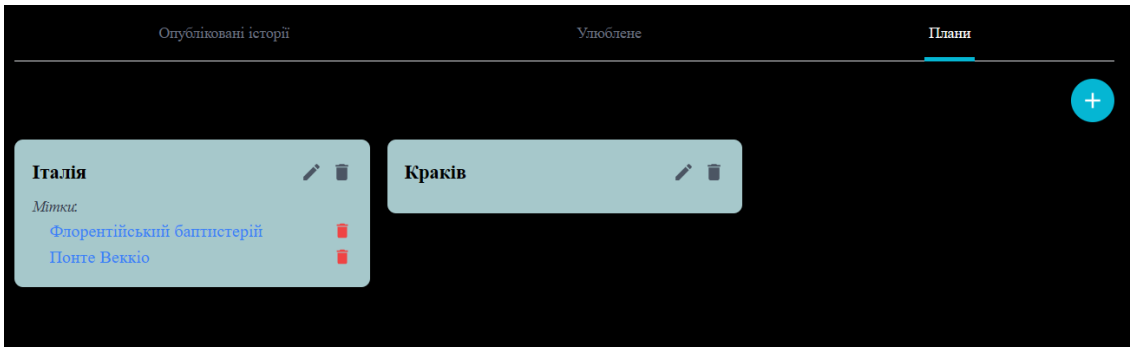


Рис. 4.28 Відображення запланованих маршрутів

За бажанням, під час прочитання історій, користувачі можуть також переглянути профіль автора історії, де відображаються всі опубліковані історії та оголошення автора.

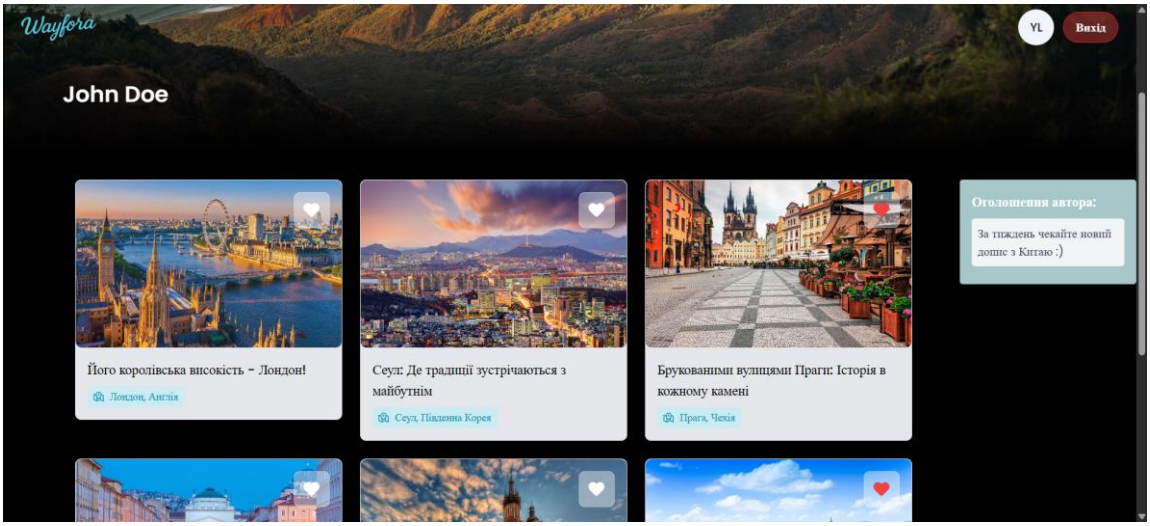


Рис. 4.29 Перегляд профілю інших користувачів

4.4 Тестування web-застосунку

Кросбраузерне тестування є критично важливим для web-застосунку. Оскільки користувачі використовують різні браузери, до прикладу такі як Chrome, Firefox та Edge, кожен з яких може по-різному інтерпретувати HTML, CSS та JavaScript, особливо при використанні React, можуть виникати візуальні та функціональні проблеми. Навіть при використанні React, який мінімізує

багато проблем, відмінності у відображенні все ще можливі. Тому, якщо застосунок не протестований належним чином у різних браузерах, це може призвести до негативного досвіду користувачів. Для web-застосунку, де важлива взаємодія з контентом, кросбраузерне тестування гарантує, що всі користувачі матимуть консистентний досвід, забезпечуючи широку доступність та зручність використання.

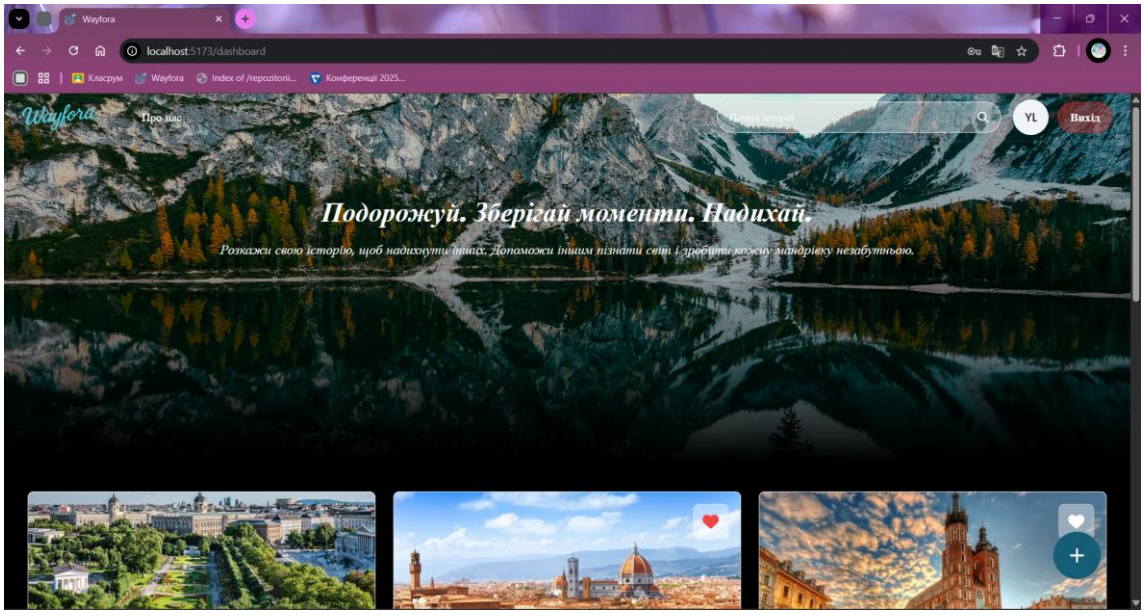


Рис. 4.30 Google Chrome

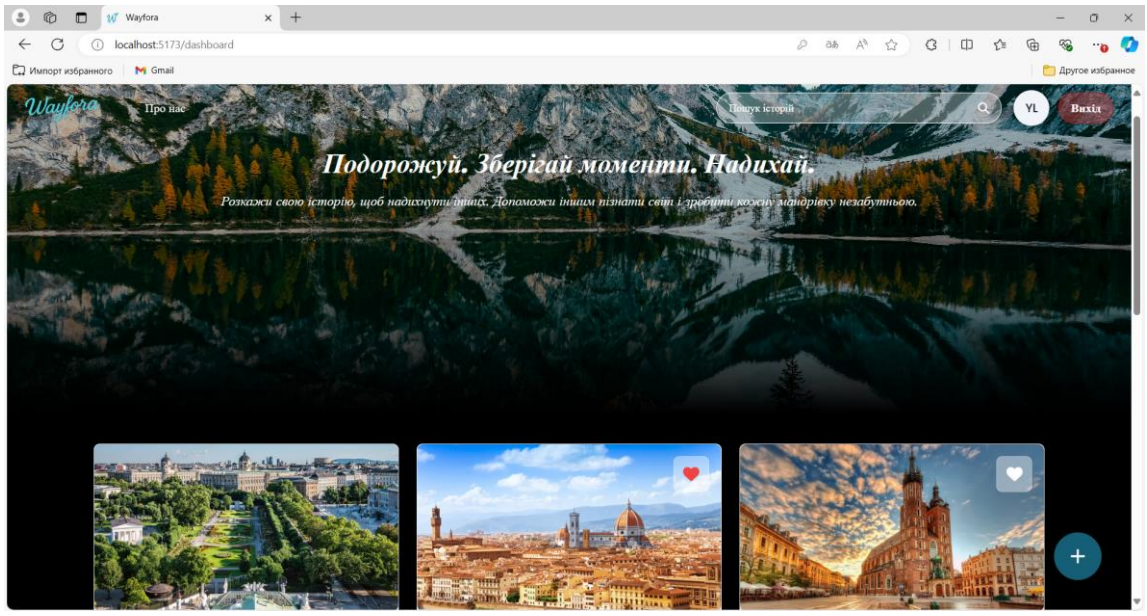


Рис. 4.31 Microsoft Edge

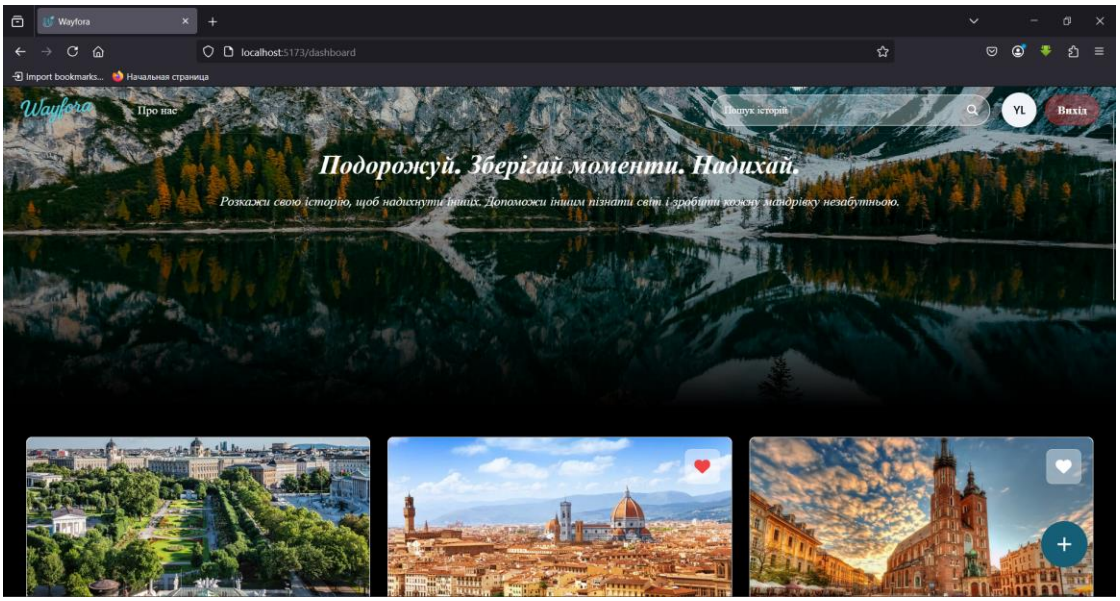


Рис. 4.32 Mozilla Firefox

Адаптивність є базовою вимогою до сучасних вебсайтів. Вона означає здатність web-застосунку адаптуватися до різних розмірів екранів пристроїв, забезпечуючи позитивний досвід користувачів та збільшуючи цільову аудиторію. Адаптивний дизайн передбачає гнучкість макету, динамічну зміну розташування та розмірів елементів, адаптивну типографіку для зручного читання тексту, адаптивні зображення з різною роздільною здатністю та зручність використання на сенсорних екранах.

Для web-застосунку, де користувачі можуть переглядати історії подорожей та взаємодіяти з контентом на різних пристроях, адаптивність гарантує, що кожен користувач, незалежно від пристрою, отримає оптимальний досвід, роблячи застосунок доступним та зручним у використанні.

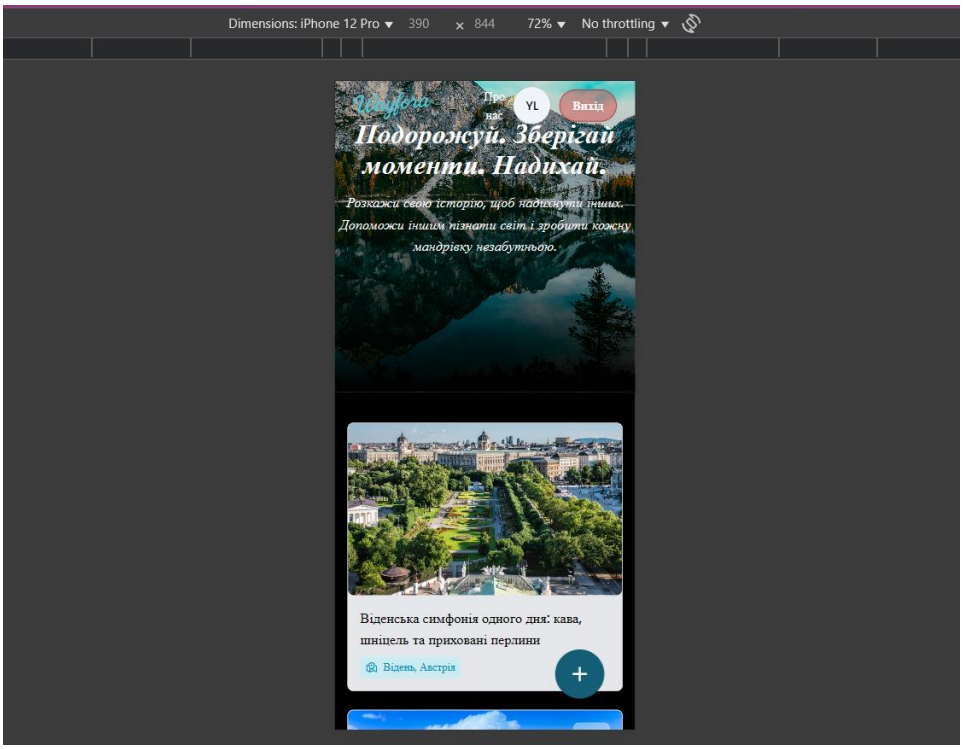


Рис. 4.33 Вигляд сайту на екрані з роздільною здатністю 390 x 844

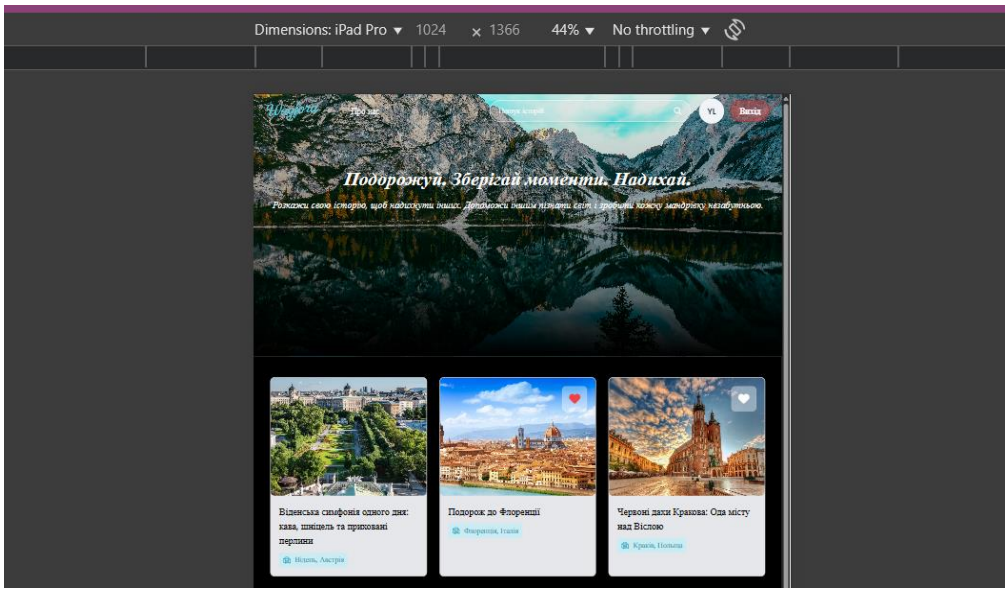


Рис. 4.34 Вигляд сайту на екрані з роздільною здатністю 1024 x 1366

Тестові сценарії, розроблені для web-застосунку, мають на меті всебічну перевірку його функціональності відповідно до встановлених вимог. Ці сценарії поділяються на позитивні, що використовуються для підтвердження правильної роботи функцій з валідними даними, та негативні, спрямовані на виявлення помилок та перевірку стійкості системи до некоректних вхідних даних.

Таблиця 4.1

Тест-кейси для функціонального тестування

№ тест-кейсу	OK/NOK	Дії	Очікуваний результат
1	OK	1. Відкрити головну сторінку.	1. Відображається список історій подорожей інших користувачів.
2	OK	1. Ввести ключове слово у поле "Пошук" та натиснути на значок пошуку.	1. Відображаються історії подорожей, що відповідають введеному ключовому слову
3	OK	1. Натиснути на картку історії подорожі на головній сторінці	1. Відкривається сторінка перегляду обраної історії подорожі з відображенням всіх елементів.
4	OK	1. На сторінці детального перегляду історії натиснути кнопку "Зберегти" біля відміченого місця. 2. Обрати існуючу колекцію або створити нову.	1. Відмічене місце додається до обраної колекції користувача.
5	OK	1. На сторінці перегляду історії ввести текст у поле коментаря та натиснути кнопку "Додати".	1. Створений коментар відображається під історією подорожі.
6	OK	1. Натиснути кнопку "Додати історію" на головній сторінці або в особистому профілі. 2. Заповнити форму для створення нової історії. 3. Натиснути кнопку "Опублікувати".	1. Відкривається форма для заповнення інформації про нову історію подорожі. 2. Всі поля для вводу доступні. 3. Нова історія успішно додається до системи та відображається у списку історій на головній сторінці та в профілі користувача.

Продовження таблиці 4.1

Тест-кейси для функціонального тестування

7	OK	1. Перейти до особистого профілю. 2. Натиснути кнопку "Редагувати" біля однієї зі своїх історій. 3. Змінити дані в формі редагування. 4. Натиснути кнопку "Зберегти зміни".	1. Відкривається форма редагування обраної історії з попередньо заповненими даними. 2. Всі поля для редагування доступні. 3. Змінені дані успішно зберігаються та відображаються на сторінці історії та в профілі.
8	OK	1. Перейти до особистого профілю. 2. Ввести текст оголошення у відповідне поле. 3. Натиснути кнопку "Створити оголошення".	1. Створене оголошення відображається у розділі оголошень в особистому профілі користувача.
9	OK	1. Перейти до особистого профілю. 2. У розділі "Заплановані маршрути" переглянути збережені колекції місць.	1. Відображається список створених користувачем колекцій місць з їх назвами та кількістю збережених місць.

ВИСНОВКИ

У даній кваліфікаційній роботі проведено аналіз предметної галузі онлайн-інструментів для обміну досвідом подорожей та планування особистих маршрутів, що є актуальним у контексті зростаючої популярності самостійних подорожей та обміну враженнями в соціальних мережах.

На початковому етапі роботи було розглянуто основні аспекти та актуальність вебзастосунків даного типу, що дозволяють користувачам ділитися своїми подорожами та планувати майбутні. Проведено аналіз існуючих рішень на ринку, виявлено їх ключові функціональні можливості, переваги та недоліки, що допомогло окреслити вимоги до розроблюваного застосунку. Визначено ключові задачі, необхідні для успішної реалізації проєкту, включаючи проєктування архітектури, розробку бази даних та реалізацію основної функціональності.

Досліджено та описано принципи роботи технологій, що були використані для розробки web-застосунку. Описано переваги використання обраних інструментів розробки, включаючи їхню ефективність у створенні сучасних вебзастосунків. Розглянуто підходи до розробки серверної та клієнтської частин, а також управління базою даних.

Розроблено архітектуру web-застосунку, що включає клієнтську та серверну частини, а також основні сторінки, які забезпечують логічну структуру та зручність використання. Описано структуру бази даних, яка включає колекції для зберігання інформації про користувачів, історії подорожей, колекції збережених місць та оголошення. Представлено загальну структуру проєкту та основні файли, необхідні для його функціонування.

Реалізовано основні функціональні можливості web-застосунку, зокрема створення та перегляд історій подорожей з можливістю додавання мультимедійного контенту, планування особистих маршрутів на основі збережених місць, здійснення пошуку історій за різними критеріями, перегляд та

редагування особистого профілю користувача, а також додавання коментарів до історій інших користувачів.

Проведено тестування розробленого web-застосунку, яке підтвердило його відповідність визначеним функціональним та нефункціональним вимогам, а також продемонструвало стабільну роботу основних компонентів та зручність використання для кінцевих користувачів.

Отже, виконана кваліфікаційна робота підтвердила ефективність розробленого web-застосунку як комплексного рішення для обміну досвідом подорожей та планування особистих маршрутів. Поєднання соціальної взаємодії з практичною функціональністю надає користувачам зручний та інформативний інструмент, що свідчить про успішне досягнення поставленої мети та практичну цінність розробленого рішення.

Робота пройшла апробацію. За її результатами було опубліковано наступні тези доповідей:

1. Лисаковська Я.В., Яскевич В.О. Актуальність розробки вебзастосунку для обміну подорожніми досвідами: Матеріали Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в ІКТ» 24 квітня 2025 р., Київ Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2025. С. 33-35.

2. Лисаковська Я.В., Яскевич В.О. Визначення вимог до вебзастосунку для обміну досвідом подорожнім досвідом: Матеріали Всеукраїнської науково-технічної конференції «Сучасний стан та перспективи розвитку IoT» 15 квітня 2025 р., Київ Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2025. (подано до друку).

ПЕРЕЛІК ПОСИЛАНЬ

1. Машіка Г. В., Грабар М. В. Тревел-блогінг як інноваційний інструмент розвитку туристичного бізнесу. Економіка та суспільство. 2024. № 60. С. 60-80. URL: https://dspace.uzhnu.edu.ua/jspui/handle/lib/60999

2. Кондра А., Лойко Н., Пасічник С., Кунанець Н., Машіка Г. Рекомендаційна система для туристичної галузі. Вісник Хмельницького національного університету. Серія: Технічні науки. 2023. Том 329, № 6. С. 172-180. URL: https://dsim.khmnu.edu.ua/index.php/dsim/article/download/136/135/306

3. Rancati E., & d'Agata A. (2025). How do Travel Bloggers Influence Generation Z's Travel Decisions? An Exploratory Study through Five Italian Famous Travel Bloggers. *European Scientific Journal, ESJ*, 21(4), С. 17-34 URL: https://www.researchgate.net/publication/389433825_How_do_Travel_Bloggers_Influence_Generation_Z%27s_Travel_Decisions_An_Exploratory_Study_through_Five_Italian_Famous_Travel_Bloggers

4. Bosangit C., Hibbert S., McCabe S. "If I was going to die I should at least be having fun": Travel blogs, meaning and tourist experience. Annals of Tourism Research. 2015. Vol. 55. С. 1-14 URL: https://www.sciencedirect.com/science/article/pii/S0160738315001024

5. Tripadvisor URL: https://www.tripadvisor.com/ (дата звернення 14.03.2024).

6. Wanderlog URL: https://wanderlog.com/ (дата звернення 15.03.2024).

7. Google Maps URL: https://www.google.com.ua/maps/ (дата звернення 16.03.2024).

8. Travellerspoint URL: https://www.travellerspoint.com/ (дата звернення 17.03.2024).

9. JavaScript документація URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript (дата звернення 22.03.2024).

10. Сучасний підручник JavaScript URL: https://uk.javascript.info/ (дата звернення 22.03.2024).

11. React документація URL: https://react.dev/ (дата звернення 22.03.2024).

12. Why Vite URL: https://vite.dev/guide/why (дата звернення 22.03.2024).
13. Tailwind CSS документація URL: https://tailwindcss.com/docs/installation/using-vite (дата звернення 22.03.2024).
14. Node.js документація URL: https://nodejs.org/en (дата звернення 23.03.2024).
15. Node.js Tutorial URL: https://www.geeksforgeeks.org/nodejs/ (дата звернення 23.03.2024).
16. Express.js документація URL: https://expressjs.com/ (дата звернення 23.03.2024).
17. MongoDB документація URL: https://www.mongodb.com/ (дата звернення 23.03.2024).
18. MongoDB Tutorial URL: https://www.w3schools.com/mongodb/index.php (дата звернення 23.03.2024).
19. Visual Studio Code документація URL: https://code.visualstudio.com/docs (дата звернення 23.03.2024).
20. Postman документація URL: https://www.postman.com/product/what-is-postman/ (дата звернення 23.03.2024).
21. Лисаковська Я.В. Актуальність розробки вебзастосунку для обміну подорожніми досвідами: Матеріали Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в ІКТ» 24 квітня 2025 р., Київ Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2025. С. 32-34.

ДОДАТОК А ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка web-застосунку для обміну досвідом подорожей та планування особистих маршрутів

Виконала студентка 4 курсу

групи ПД-41

Лисаковська Яна Віталіївна

Керівник роботи

к.т.н, доц, доцент кафедри ІПЗ Яскевич Владислав Олександрович

Київ – 2025

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

- **Мета роботи** - спрощення процесу обміну особистим досвідом подорожей та планування особистих маршрутів.
- **Об'єкт дослідження -** процес обміну особистим досвідом подорожей та планування особистих маршрутів.
- **Предмет дослідження** – web-застосунок Wayfora для обміну досвідом подорожей та планування особистих маршрутів.

ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1. Провести огляд та аналіз існуючих web-застосунків для обміну досвідом подорожей та планування особистих маршрутів, визначити їх переваги та недоліки;
2. Сформулювати вимоги до розроблюваного web-застосунку на основі аналізу предметної галузі та потреб користувачів.
3. Провести огляд та аналіз технологій та інструментів, що можуть бути використані для розробки web-застосунку;
4. Спроектувати та розробити web-застосунок з використанням обраних технологій;
5. Провести тестування розробленого web-застосунку для виявлення та усунення помилок.

3

АНАЛІЗ АНАЛОГІВ

Показник	TripAdvisor	Wanderlog	Google Maps	Travellerspoint	Wayfora
Обмін розгорнутими історіями	-	- (фокус більше в обміні коротких ідей та планів)	-	+	+
Персональні профілі з хронологією подорожей	-	+ (збереження майбутніх планів, та ідей, що публікувалися у спільноту)	-	+ (ведення онлайн-щоденника подорожей)	+
Візуалізація (наявність візуального контенту)	+	+	+	+	+
Актуальність інформації	+ (може бути перенасичення рекламою та неправдиві відгуки)	+(залежить від активності спільноти)	+ (може бути перенасичення рекламою та неправдиві відгуки)	- (залежить від активності спільноти)	+
Можливість взаємодії з контентом інших користувачів	+	-	-	- (не можна взаємодіяти в коментарях, але для обговорення є форуми)	+
Можливість планувати порожі	+	+	+ (можна створювати списки з місць, що хочеш відвідати)	-	+(можна створювати списки з місць, що хочеш відвідати)

4

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

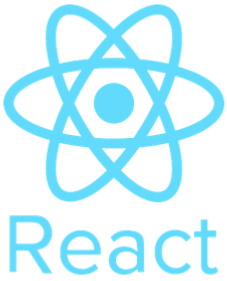
Функціональні вимоги:

1. Реєстрація та авторизація.
2. Створення історій: Інструменти для написання, редагування та видалення користувацьких публікацій про подорожі з фото та місцями.
3. Перегляд стрічки: Відображення оновлень та історій від інших користувачів.
4. Пошук: Функція пошуку публікацій за назвою та локацією
5. Збереження місць: Можливість додавати цікаві локації до особистих списків для майбутніх подорожей.
6. Особистий профіль: Розділ користувача з його публікаціями та збереженими місцями.

Нефункціональні вимоги:

1. Зручність користування: Інтуїтивно зрозумілий інтерфейс та легка навігація.
2. Захист даних: Забезпечення безпеки особистої інформації користувачів та їхнього контенту.
3. Сумісність: Коректна робота на різних браузерах та пристроях з різними розмірами екранів.

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

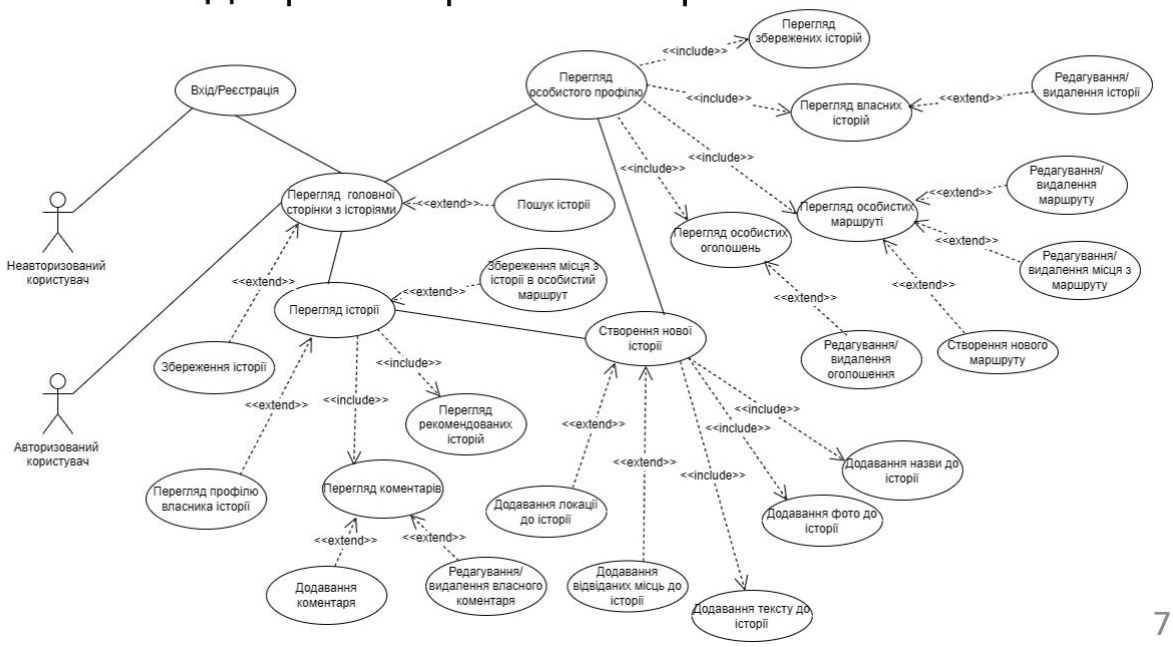


mongoDB

express

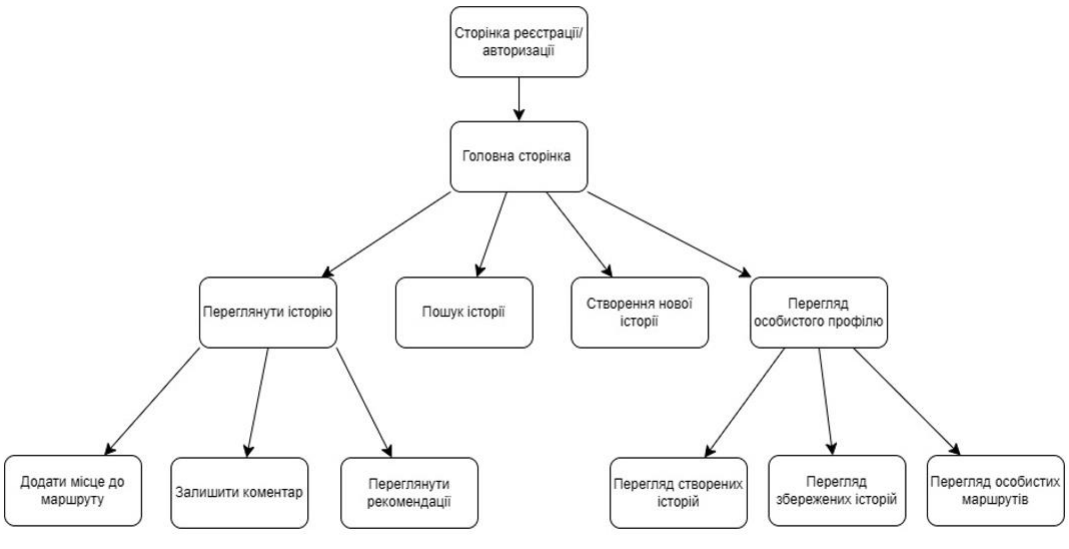
POSTMAN

Діаграма варіантів використання



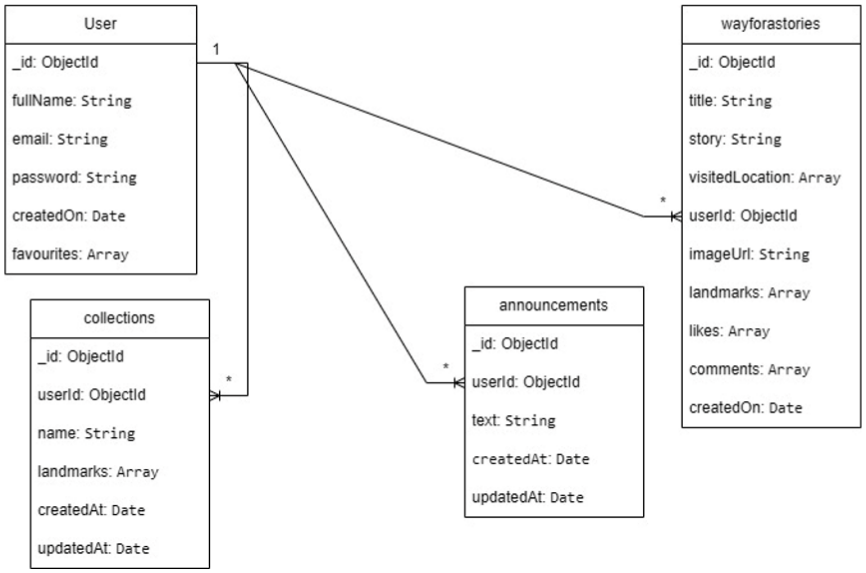
7

Карта сайту



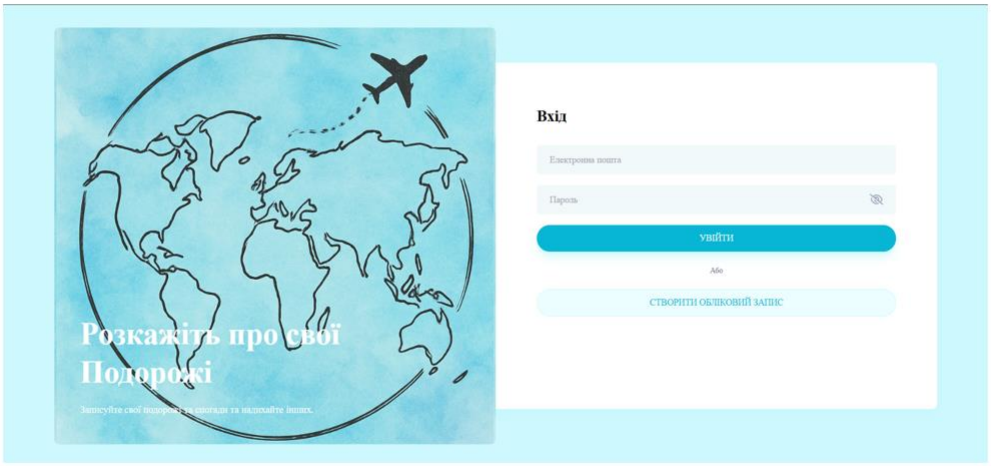
8

Структура бази даних



9

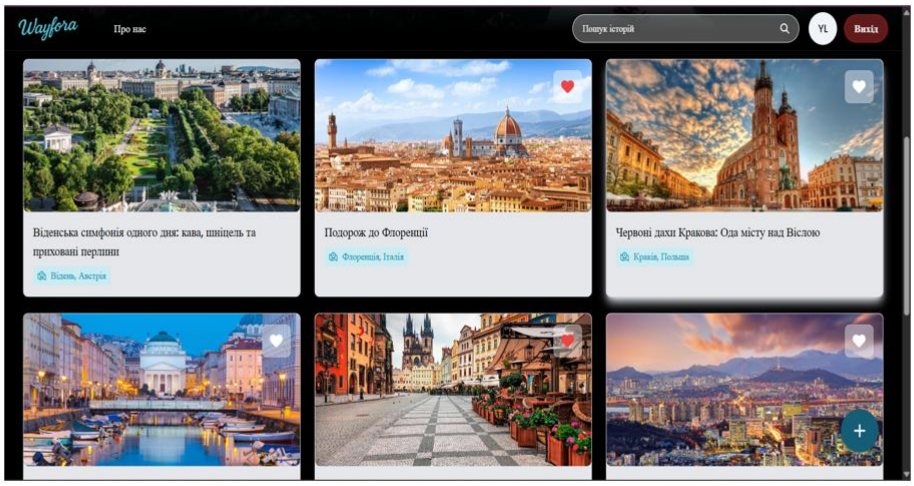
ЕКРАННІ ФОРМИ



Сторінка авторизації користувача

10

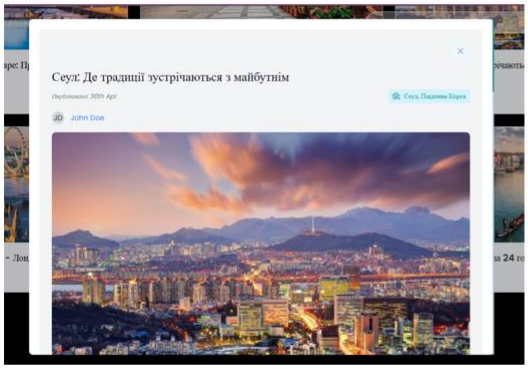
ЕКРАННІ ФОРМИ

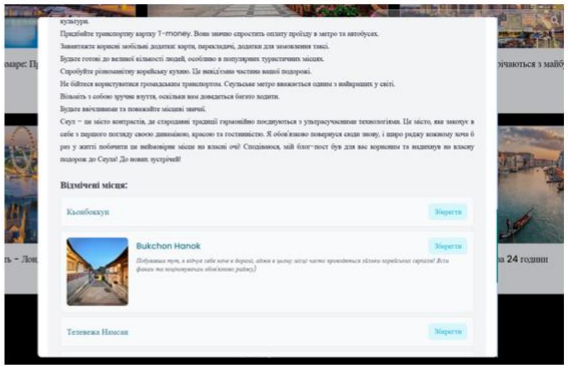


Відображення історій подорожей на головній сторінці

11

ЕКРАННІ ФОРМИ

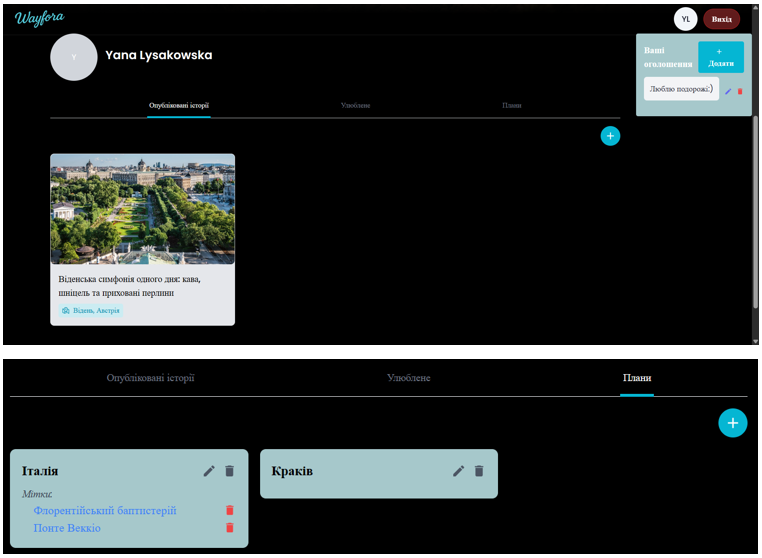




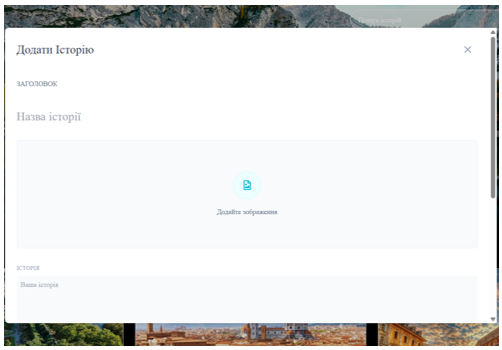
Перегляд історій та відмічених місць

12

ЕКРАННІ ФОРМИ



Сторінка особистого профілю



Вікно додавання нової історії

13

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Лисаковська Я.В., Яскевич В.О. Актуальність розробки вебзастосунку для обміну подорожніми досвідами: Матеріали Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в ІКТ» 24 квітня 2025 р., Київ Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2025. С. 33-35.
2. Лисаковська Я.В., Яскевич В.О. Визначення вимог до вебзастосунку для обміну досвідом подорожнім досвідом: Матеріали Всеукраїнської науково-технічної конференції «Сучасний стан та перспективи розвитку IoT» 15 квітня 2025 р., Київ Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2025. (подано до друку).

14

ВИСНОВКИ

1. Аналіз існуючих рішень на ринку визначив ключові можливості та обмеження, що дозволило сформувати конкурентоздатний функціонал для web-застосунку.
2. Визначені на основі потреб користувачів функціональні та нефункціональні вимоги стали чітким орієнтиром для розробки.
3. Обраний стек технологій (JavaScript, React, Node.js, Express.js, MongoDB) продемонстрував свою ефективність та відповідність задачам проєкту.
4. Реалізовано основні функціональні можливості web-застосунку забезпечують користувачам необхідний інструментарій для обміну досвідом та планування подорожей.
5. Проведене тестування підтвердило функціональну придатність розробленого web-застосунку та його відповідність визначеним вимогам.

15

ДОДАТОК Б. ЛІСТИНГИ ОСНОВНИХ МОДУЛІВ

Клієнтська частина:
Home.jsx

```
import React, { useEffect, useState, useRef } from "react";
import Navbar from "../../components/Navbar";
import Footer from "../../components/Footer";
import { useNavigate } from "react-router-dom";
import axiosInstance from "../../utils/axiosInstance";
import { MdAdd } from "react-icons/md";
import Modal from "react-modal";
import TravelStoryCard from "../../components/Cards/TravelStoryCard";
import AddEditStory from "./AddEditStory";
import ViewTravelStory from "./ViewTravelStory";
import AboutUs from "../../components/AboutUs";
import { ToastContainer, toast } from 'react-toastify';
import 'react-toastify/dist/ReactToastify.css';
import backgroundImage from '../../assets/images/bg1.jpg';
const ITEMS_PER_PAGE = 12;
const Home = () => {
    const navigate = useNavigate();
    const [userInfo, setUserInfo] = useState(null);
    const [allStories, setAllStories] = useState([]);
    const [searchQuery, setSearchQuery] = useState('');
    const [filterType, setFilterType] = useState('');
    const [displayedStories, setDisplayedStories] = useState([]);
    const [loadingMore, setLoadingMore] = useState(false);
    const [isAboutUsModalOpen, setIsAboutUsModalOpen] = useState(false);
    const aboutUsModalRef = useRef(null);
    const [currentViewedStory, setCurrentViewedStory] = useState(null);
    const getRecommendedStories = (currentStory, allStories, currentUserId) => {
      if (!currentStory || !currentStory.visitedLocation || !allStories) {
        return [];
      }
      const recommended = allStories.filter(story => {
        if (story._id === currentStory._id) {
          return false;
        }
        if (String(story.userId) === String(currentUserId)) {
          return false;
        }
        if (!story.visitedLocation || story.visitedLocation.length === 0) {
          return false;
        }
        return currentStory.visitedLocation.some(location => story.visitedLocation.includes(location)
        );
      });
      const shuffled = [...recommended].sort(() => 0.5 - Math.random());
      return shuffled.slice(0, 3);
    };
    const [openAddEditModal, setOpenAddEditModal] = useState({
      isShown: false,
      type: "add",
      data: null,
    });
    const [openViewModal, setOpenViewModal] = useState({
      isShown: false,
      data: null,
    });
    const openAboutUsModal = () => {
      setIsAboutUsModalOpen(true);
    };
    const closeAboutUsModal = () => {
      setIsAboutUsModalOpen(false);
    };
    const getUserInfo = async () => {
      try {
        const response = await axiosInstance.get("/get-user");
        if (response.data && response.data.user) {
          setUserInfo(response.data.user);
        }
      } catch (error) {
        if (error.response && error.response.status === 401) {
          localStorage.clear();
          navigate("/login");
        }
```

```
    }
  };
  const getAllStories = async () => {
    try {
      const response = await axiosInstance.get("/get-all-stories");
      if (response.data && response.data.stories) {
        const sortedStories = response.data.stories.sort((a, b) => {
          const dateA = new Date(a.createdOn);
          const dateB = new Date(b.createdOn);
          return dateB.getTime() - dateA.getTime();
        });
        setAllStories(sortedStories);
        setDisplayedStories(sortedStories.slice(0, ITEMS_PER_PAGE));
      }
    } catch (error) {
      console.log("An unexpected error occurred. Please try again");
    }
  };
  const handleEdit = (data) => {
    setOpenAddEditModal({
      isShown: true,
      type: "edit",
      data: data
    });
  };
  const handleOpenViewStoryModal = (data) => {
    setOpenViewModal({ isShown: true, data });
    setCurrentViewedStory(data);
  };
  const updateIsFavourite = async (storyData) => {
    if (!storyData || !storyData._id) {
      console.error("Invalid story data");
      return;
    }
    const storyId = storyData._id;
    try {
      const response = await axiosInstance.put(`/update-is-favourite/${storyId}`);
      if (response.data && response.data.story) {
        setAllStories((prevStories) =>
          prevStories.map((story) =>
            story._id === storyId ? { ...story, likes: response.data.story.likes } : story
          ).sort((a, b) => {
            const aIsLiked = response.data.story.likes?.includes(userInfo?._id);
            const bIsLiked = response.data.story.likes?.includes(userInfo?._id);
            if (aIsLiked && !bIsLiked) return -1;
            if (!aIsLiked && bIsLiked) return 1;
            return 0;
          })
        );
      }
    } catch (error) {
      console.error("Помилка при збереженні історії:", error);
      toast.error("Не вдалося зберегти історію. Спробуйте ще раз.");
    }
  };
  const deleteTravelStory = async (data) => {
    const storyId = data._id;
    try {
      const response = await axiosInstance.delete("/delete-story/" + storyId);
      if (response.data && !response.data.error) {
        toast.error("Історію видалено");
        setOpenViewModal((prevState) => ({ ...prevState, isShown: false }));
        setCurrentViewedStory(null);
        getAllStories();
      }
    } catch {
      console.log("An unexpected error occurred. Please try again.");
    }
  };
  const onSearchStory = async (query) => {
    try {
      const response = await axiosInstance.get("/search", {
        params: {
          query,
        },
      });
      if (response.data && response.data.stories) {
        const filteredStories = response.data.stories.filter((story) => story.title.toLowerCase().includes(query.toLowerCase()) ||
          (Array.isArray(story.visitedLocation) && story.visitedLocation.some((location) =>
```

```
location.toLowerCase().includes(query.toLowerCase())
                    )));
                setFilterType("search");
                setAllStories(filteredStories);
                console.log("Search Results:", filteredStories);
              }
            } catch (error) {
              console.log("An unexpected error occurred. Please try again.");
            }
          };
          const handleClearSearch = () => {
            setSearchQuery("");
            setFilterType('');
            getAllStories();
          };
          const openRecommendedStory = (story) => {
            setOpenViewModal({ isShown: false, data: null });
            setTimeout(() => {
            setOpenViewModal({ isShown: true, data: story });
            setCurrentViewedStory(story);
          }, 0);
          };
          useEffect(() => {
            getAllStories();
            getUserInfo();
          }, [userInfo?._id]);
          useEffect(() => {
        setDisplayedStories(allStories.slice(0, ITEMS_PER_PAGE));
            setLoadingMore(false);
          }, [allStories]);
          const loadMore = () => {
            setLoadingMore(true);
            setTimeout(() => {
              const nextStories = allStories.slice(displayedStories.length, displayedStories.length + ITEMS_PER_PAGE);
      setDisplayedStories([...displayedStories, ...nextStories]);
              setLoadingMore(false);
            }, 500);
          };
          return (
            <>
              <Navbar
                userInfo={userInfo}
                searchQuery={searchQuery}
                setSearchQuery={setSearchQuery}
                onSearchNote={onSearchStory}
                handleClearSearch={handleClearSearch}
                className="absolute top-0 left-0 right-0 z-20 bg-transparent border-b border-white/10"
                onOpenAboutUs={openAboutUsModal}
              />
              <div className="hero relative bg-black mt-[-68px]" style={{ minHeight: '500px' }}>
                <div className="absolute inset-0 bg-image" style={{ backgroundImage: `url('${backgroundImage}')`, backgroundSize: 'cover', backgroundPosition: 'center' }}></div>
                <div className="absolute inset-0 bg-gradient-to-b from-transparent via-black/30 via-70% to-black opacity-100"></div>
                <div className="container mx-auto relative z-10 pt-36 pb-16 text-center text-white italic">
                  <h1 className="text-4xl font-bold mb-4">Подорожуй. Зберігай моменти. Надихай.</h1>
                  <p className="text-lg mb-8">Розкажи свою історію, щоб надихнути інших. Допоможи іншим пізнати світ і зробити кожну мандрівку незабутньою.</p>
                </div>
              </div>
              <div className="bg-black py-10 px-4 sm:px-6 lg:px-8">
                <div className="container mx-auto">
                  <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 gap-6">
                    {displayedStories.length > 0 ? (
                      displayedStories.map((item) => (
                        <TravelStoryCard
                          key={item._id}
                          imgUrl={item.imageUrl}
                          title={item.title}
                          story={item.story}
                          date={item.visitedDate}
                          visitedLocation={item.visitedLocation}
                          isFavourite={item.likes?.includes(userInfo?._id)}
                          onClick={() => handleOpenViewStoryModal(item)}
                          onFavouriteClick={() => updateIsFavourite(item)}
                          authorId={item.userId}
                          currentUserId={userInfo?._id}
                        />
                      ))
                    ) : (
                      <div className="text-center text-gray-500 col-span-full">
```

```
                        {searchQuery ? "Не знайдено історій за вашим запитом." : "Немає жодної історії. Натисніть кнопку + щоб додати першу історію!"}
                        </div>
                      )}
                    </div>
                    {displayedStories.length < allStories.length && (
                      <div className="flex justify-center mt-8">
                        <button
                          className="bg-primary hover:bg-cyan-400 text-white font-bold py-2 px-4 rounded"
                          onClick={loadMore} disabled={loadingMore} >
                          {loadingMore ? "Завантаження..." : "Завантажити ще"}
                        </button>
                      </div> )}
                  </div></div>
                <Modal isOpen={openAddEditModal.isShown}
                  onRequestClose={() => setOpenAddEditModal({ isShown: false, type: "add", data: null })}
                  style={{
                    overlay: {
                      backgroundColor: "rgba(0, 0, 0, 0.2)",
                      zIndex: 999,
                      display: "flex",
                      alignItems: "center",
                      justifyContent: "center",
                      padding: "20px"
                    },
                    content: {
                      position: "relative",
                      width: "100%",
                      maxWidth: "1000px",
                      height: "auto",
                      minHeight: "600px",
                      margin: "0",
                      padding: "0",
                      border: "none",
                      borderRadius: "16px",
                      background: "#fff",
                      boxShadow: "0 4px 6px -1px rgba(0, 0, 0, 0.1), 0 2px 4px -1px rgba(0, 0, 0, 0.06)",
                      inset: "auto",
                      overflow: "visible"
                    }
                  }}
                  appElement={document.getElementById("root")}
                >
                  <AddEditStory
                    type={openAddEditModal.type}
                    storyInfo={openAddEditModal.data}
                    onClose={() => {
                      setOpenAddEditModal({ isShown: false, type: "add", data: null });
                    }}
                    getAllStories={getAllStories}
                  />
                </Modal>
                <Modal isOpen={openViewModal.isShown}
                  onRequestClose={() => {
                    setOpenViewModal((prevState) => ({ ...prevState, isShown: false }));
                    setCurrentViewedStory(null);
                  }}
                  style={{
                    overlay: {
                      backgroundColor: "rgba(0, 0, 0, 0.2)",
                      zIndex: 999,
                      display: "flex",
                      alignItems: "center",
                      justifyContent: "center",
                      padding: "60px",},
                    content: {
                      height: "650px",
                      width: "900px",
                      marginBottom: "50px",
                    }}}

                  appElement={document.getElementById("root")}
                  className="model-box"
                >
                  <ViewTravelStory
                    storyInfo={openViewModal.data || null}
                    onClose={() => {
                      setOpenViewModal((prevState) => ({ ...prevState, isShown: false }));
                      setCurrentViewedStory(null);
                    }}
                    oneEditClick={() => {
                      setOpenViewModal((prevState) => ({ ...prevState, isShown: false }));
                      setCurrentViewedStory(null);
                      handleEdit(openViewModal.data || null);
                    }}
                    oneDeleteClick={() => {
                      deleteTravelStory(openViewModal.data || null);
                      setCurrentViewedStory(null);
                    }}
                    currentUserId={userInfo?._id}
                    authorId={openViewModal.data?.userId}
                    allStories={allStories}
                    openRecommendedStory={openRecommendedSt
```

```
ory}
onUpdateFavourite={updateIsFavourite}
        />
                </Modal>
                <button
                  className="w-16 h-16 flex items-center justify-center rounded-full bg-cyan-800 hover:bg-cyan-700 fixed right-10 bottom-10 transition-all duration-300"
                  onClick={() => {
                    setOpenAddEditModal({ isShown: true, type: "add", data: null });
                  }}
                >
                  <MdAdd className="text-[32px] text-white" />
                </button>
                {isAboutUsModalOpen && (
                  <AboutUs isOpen={isAboutUsModalOpen} onClose={closeAboutUsModal} />
                )}
                <ToastContainer />
                <Footer />
              </>
            );
        };
        export default Home;
```

ViewTravelStory.jsx

```
import moment from 'moment';
import React, { useState, useEffect } from 'react';
import { GrMapLocation } from 'react-icons/gr';
import { MdDeleteOutline, MdUpdate, MdClose } from 'react-icons/md';
import axiosInstance from '../../utils/axiosInstance';
import { toast } from 'react-toastify';
import { Link, useNavigate } from 'react-router-dom';
import TravelStoryCard from "../../components/Cards/TravelStoryCard";
const ViewTravelStory = ({ storyInfo, onClose, oneEditClick, oneDeleteClick, allStories, openRecommendedStory, onUpdateFavourite }) => {
  const [isOwner, setIsOwner] = useState(false);
  const [currentUserId, setCurrentUserId] = useState(null);
  const [isCollectionModalOpen, setIsCollectionModalOpen] = useState(false);
  const [selectedLandmark, setSelectedLandmark] = useState(null);
  const [collections, setCollections] = useState([]);
  const [newCollectionName, setNewCollectionName] = useState("");
  const [comments, setComments] = useState([]);
  const [newComment, setNewComment] = useState('');
  const [editingCommentId, setEditingCommentId] = useState(null);
  const [editedCommentText, setEditedCommentText] = useState('');
  const navigate = useNavigate();
  const [recommendedStories, setRecommendedStories] = useState([]);
  const decodeToken = (token) => {
    try {
      const payload = token.split('.')[1];
      const decoded = JSON.parse(atob(payload));
      return decoded;
    } catch (error) {
      console.error('Failed to decode token:', error);
      return null;
    }
  };
  useEffect(() => {
    const token = localStorage.getItem('token');
    if (token) {
      const decoded = decodeToken(token);
      if (decoded && decoded.userId) {
        setCurrentUserId(decoded.userId);
      } else {
       console.warn("userId not found in token");
      }
    } else {
      console.warn("Token not found in localStorage");
    }
  }, []);
  useEffect(() => {
    if (storyInfo?.userId && currentUserId) {
      const ownerCheck = String(storyInfo.userId) === String(currentUserId);
      setIsOwner(ownerCheck);
    }
  }, [storyInfo?.userId, currentUserId]);
  const fetchCollections = async () => {
    try {
      const response = await axiosInstance.get('/get-user-collections');
      if (response.data && response.data.collections) {
        setCollections(response.data.collections);
      }
    } catch (error) {
      console.error('Error fetching collections:', error);
      toast.error('Failed to load collections.');
    }
  };
```

```
  const createNewCollection = async () => {
    try {
      const response = await axiosInstance.post('/create-collection', { name: newCollectionName });
      if (response.data && response.data.collection) {
        setCollections([...collections, response.data.collection]);
        setNewCollectionName("");
        toast.success('Колекцію створено!');
      }
    } catch (error) {
      console.error('Error creating collection:', error);
      toast.error('Failed to create collection.');
    }
  };
  const saveLandmarkToCollection = async (collectionId) => {
    try {
      const response = await axiosInstance.post(`/add-to-collection/${collectionId}`, {
        landmark: selectedLandmark,
      });
      if (response.data && response.data.message) {
        toast.success(`Мітку "${selectedLandmark.name}" додано до колекції!`);
        closeCollectionModal();
      }
    } catch (error) {
      console.error('Error saving landmark to collection:', error);
      toast.error('Не вдалося додати мітку до колекції.');
    }
  };
  const openCollectionModal = (landmark) => {
    setSelectedLandmark(landmark);
    setIsCollectionModalOpen(true);
    fetchCollections();
  };
  const closeCollectionModal = () => {
    setSelectedLandmark(null);
    setIsCollectionModalOpen(false);
  };
  useEffect(() => {
    fetchComments();
  }, [storyInfo?._id]);
  const fetchComments = async () => {
    if (storyInfo?._id) {
      try {
        const response = await axiosInstance.get(`/api/stories/${storyInfo._id}/comments`);
        if (response.data?.comments) {
          setComments(response.data.comments);
          console.log('Отримані коментарі:', response.data.comments);
        }
      } catch (error) {
        console.error('Помилка при завантаженні коментарів:', error);
        toast.error('Не вдалося завантажити коментарі.');
      }
    }
  };
  const handleAddComment = async () => {
    if (newComment.trim() && storyInfo?._id && currentUserId) {
      try {
        const response = await axiosInstance.post(`/api/stories/${storyInfo._id}/comments`, {
          text: newComment,
          userId: currentUserId,
        });
        if (response.data?.comment) {
          setComments(prevComments => [...prevComments, response.data.comment]);
          setNewComment('');
        }
      } catch (error) {
        console.error('Помилка при додаванні коментаря:', error);
        toast.error('Не вдалося додати коментар.');
      }
    }
  };
  const handleSaveEditComment = async (commentId, isReply = false, parentCommentId = null) => {
    if (editedCommentText.trim()) {
      try {
        const apiUrl = `/api/comments/${commentId}`;
        const updateData = { text: editedCommentText };
        if (isReply && parentCommentId) {
          updateData.parentCommentId = parentCommentId;
        }
        await axiosInstance.put(apiUrl, updateData);
        setComments(prevComments =>
          prevComments.map(comment => {
```

```
                if (comment._id === parentCommentId) {
                  return {
                 ...comment,
                    replies: comment.replies.map(reply =>
                      reply._id === commentId ? { ...reply, text: editedCommentText } : reply
                    ),
                  };
                }
                if (!isReply && comment._id === commentId) {
                  return { ...comment, text: editedCommentText };
                }
                return comment;
              })
            );
            setEditingCommentId(null);
            setEditedCommentText('');
            toast.success('Коментар оновлено!');
          } catch (error) {
            console.error('Помилка при редагуванні коментаря:', error);
            toast.error('Не вдалося редагувати коментар.');
          }
        }
    };
    const handleDeleteComment = async (commentId) => {
      try {
        await axiosInstance.delete(`/api/comments/${commentId}`);
        setComments(prevComments =>
          prevComments
            .map(comment => ({
              ...comment,
              replies: comment.replies ? comment.replies.filter(reply => reply._id !== commentId) : [],
            }))
            .filter(comment => comment._id !== commentId)
        );
        toast.success('Коментар видалено!');
      } catch (error) {
        console.error('Помилка при видаленні коментаря:', error);
        toast.error('Не вдалося видалити коментар.');
      }
    };
    const navigateToUserProfile = (userId) => {
      if (currentUserId === userId) {
        navigate('/profile');
      } else {
        navigate(`/user/${userId}/public-profile`);
      }
    };
    const renderComment = (comment, level = 0) => {
      const authorName = comment.userId?.fullName || 'Невідомо';
      const authorId = comment.userId?._id;
      const isMyComment = currentUserId === comment.userId?._id;
      return (
        <li key={comment._id} className={`bg-gray-${level > 0 ? '200' : '100'} p-3 rounded-md ${level > 0 ? 'ml-8 border-l-2 border-gray-400 relative' : ''}`}>
          {level > 0 && <div className="text-xs text-gray-600 absolute -mt-2 ml-1">Відповідь:</div>}
          <div className={`${level > 0 ? 'mt-2' : ''}`}>
            <p className="text-sm">
              {editingCommentId === comment._id ? (
                <textarea
                  className="w-full p-2 border rounded-md text-black text-sm"
                  value={editedCommentText}
                  onChange={(e) => setEditedCommentText(e.target.value)}
                />
              ) : (
                comment.text
              )}
            </p>
            <p className="text-xs text-gray-500 mt-1">
              Автор: {authorId ? (
                <button onClick={() => navigateToUserProfile(authorId)} className="text-blue-500 hover:underline cursor-pointer">
                  {authorName}
                </button>
              ) : (
                authorName
              )}
({moment(comment.createdOn).format('DD.MM.YYYY HH:mm')})
            </p>
            {isMyComment && editingCommentId !== comment._id && (
              <div className="flex space-x-2 mt-1 text-xs">
                <button
                  className="text-blue-500"
```

```
                        onClick={() => {
setEditingCommentId(comment._id);
setEditedCommentText(comment.text);
                        }} >
                        Редагувати
                      </button>
                      <button
                        className="text-red-500"
                        onClick={() => handleDeleteComment(comment._id)}
                      >
                        Видалити
                      </button>
                    </div>
                  )}
                  {editingCommentId === comment._id && (
                    <div className="mt-2">
                      <button
                        className="bg-green-500 hover:bg-green-700 text-white font-bold py-1 px-2 rounded text-xs"
                        onClick={() => handleSaveEditComment(comment._id, false)}
                      >
                        Зберегти
                      </button>
                      <button
                        className="bg-gray-300 hover:bg-gray-400 text-gray-800 font-bold py-1 px-2 rounded ml-2 text-xs"
                        onClick={() => setEditingCommentId(null)}
                      >
                        Скасувати
                      </button>
                    </div>
                  )}
                </div>
              </li>
            );
          };

          const getRecommendedStories = (currentStory, allStories, currentUserId) => {
            if (!currentStory || !allStories || !currentUserId) {
              return [];
            }
            const { visitedLocation } = currentStory;
            if (!visitedLocation || visitedLocation.length === 0) {
              return [];
            }
            return allStories.filter(story =>
              story._id !== currentStory._id &&
              story.userId !== currentUserId &&
              story.visitedLocation.some(location => visitedLocation.includes(location))
            ).slice(0, 3);
          };
          useEffect(() => {
            if (storyInfo && allStories && currentUserId) {
              const recommended = getRecommendedStories(storyInfo, allStories, currentUserId);
              setRecommendedStories(recommended);
              console.log('Рекомендовані історії:', recommended);
            } else {
              console.log('Недостатньо даних для отримання рекомендацій:', { storyInfo, allStories, currentUserId });
            }
          }, [storyInfo, allStories, currentUserId]);
          return (
            <div className='relative bg-gray-100 p-6 rounded-md'>
              <div className='flex items-center justify-end'>
                <div>
                  <div className='flex items-center gap-3 bg-cyan-50/50 p-2 rounded-l-lg'>
                    {currentUserId && isOwner && (
                      <button className='btn-small' onClick={oneEditClick}>
                        <MdUpdate className="text-lg" /> UPDATE STORY
                      </button>
                    )}
                    {currentUserId && isOwner && (
                      <button className='btn-small btn-delete' onClick={oneDeleteClick}>
                        <MdDeleteOutline className="text-lg" /> DELETE
                      </button>
                    )}
                    <button className="" onClick={onClose}>
                      <MdClose className='text-xl text-slate-400' />
                    </button>
                  </div>
                </div>
              </div>
              <div>
                <div className='flex-1 flex flex-col gap-2 py-4'>
                  <h1 className='text-2xl text-slate-950'>
                    {storyInfo && storyInfo.title}
```

```
            </h1>
            <div className='flex items-center justify-between gap-3'>
              {storyInfo?.createdOn && (
                <span className='text-xs text-gray-500 italic'>
                  Опубліковано: {moment(storyInfo.createdOn).format("Do MMM")}
                </span>
              )}
              <div className='inline-flex items-center gap-2 text-[13px] text-cyan-600 bg-cyan-200/40 rounded px-2 py-1'>
                <GrMapLocation className='text-sm' />
                {storyInfo && storyInfo.visitedLocation && storyInfo.visitedLocation.map((item, index) =>

storyInfo.visitedLocation.length === index + 1 ? `${item}` : `${item}, `
                )}
              </div>
            </div>
            {storyInfo?.author && (
              <div className="flex items-center gap-2 mt-2 text-sm text-gray-700">
                <div className="rounded-full bg-gray-300 w-6 h-6 flex items-center justify-center mr-1">
                  {storyInfo.author.initials ? storyInfo.author.initials.toUpperCase() : (storyInfo.author.username ? storyInfo.author.username.charAt(0).toUpperCase() : '')}
                </div>
                <button onClick={() => navigateToUserProfile(storyInfo.userId)} className="hover:underline text-blue-500 cursor-pointer">

<span>{storyInfo.author.username || storyInfo.author.displayName || 'Автор'}</span>
                </button>
              </div>
            )}
          </div>
          <img
            src={storyInfo && storyInfo.imageUrl}
            alt="Selected"
            className='w-full h-[500px] object-cover rounded-lg'
          />
          <div className='mt-4'>
            <p className='text-sm text-slate-950 leading-6 text-justify whitespace-pre-line'>
              {storyInfo && storyInfo.story}
            </p>
          </div>
          <div className='mt-6'>
            <h3 className='text-lg font-semibold text-slate-800'>Відмічені місця:</h3>
            {storyInfo?.landmarks && storyInfo.landmarks.length > 0 ? (
              <ul className='mt-2 space-y-2'>
                {storyInfo.landmarks.map((landmark, index) => (
                  <li key={index} className='p-3 rounded-md border border-gray-200 bg-gray-50'>
                    <div className='flex items-start gap-4'>
                      {landmark.imageUrl && (
                        <div className='w-1/4 max-w-[120px] h-[130px] overflow-hidden rounded-md'>
                          <img
                            src={landmark.imageUrl}
                            alt={landmark.name}
                            className='w-full h-full object-cover'
                          />
                        </div>
                      )}
                      <div className='flex-1'>
                        <div className='flex items-center justify-between'>
                          <Link
                            to={landmark.link}
                            target="_blank"
                            rel="noopener noreferrer"
                            className='text-cyan-700 hover:underline font-medium'
                          >
                            {landmark.name}
                          </Link>
                          <button
                            className='bg-cyan-200/40 hover:bg-cyan-200 text-cyan-500 px-3 py-1.5 rounded-lg ml-2 text-sm transition-all duration-300'
                            onClick={() => openCollectionModal(landmark)}
                          >
                            Зберегти
                          </button>
                        </div>
                        {landmark.note && (
                          <p className='text-xs text-gray-600 italic mt-1'>{landmark.note}</p>
                        )}
```

```
                        </div>
                      </div>
                    </li>
                  ))}
                </ul>
              ) : (
                <p className='text-slate-500 '>Немає відмічених місць.</p>
              )}
            </div>
          </div>
          {isCollectionModalOpen && selectedLandmark && (
            <div className="fixed inset-0 bg-black/50 flex items-center justify-center z-50">
              <div className="bg-white rounded-lg p-6 w-96">
                <h5 className="text-lg font-semibold text-slate-700 mb-4">Зберегти до колекції</h5>
                <div className="space-y-4">
                  <div>
                    <h6 className="text-sm font-medium text-slate-700 mb-2">Обрати колекцію:</h6>
                    <ul className="space-y-2">
                      {collections.map((collection) => (
                        <li key={collection._id} className="flex items-center justify-between">
                          <span
                            className="text-sm text-slate-700">{collection.name}</span>
                          <button
                            className="text-cyan-500 hover:underline"
                            onClick={() => saveLandmarkToCollection(collection._id)}
                          >
                            Зберегти
                          </button>
                        </li>
                      ))}
                    </ul>
                  </div>
                  <div>
                    <h6 className="text-sm font-medium text-slate-700 mb-2">Створити колекцію:</h6>
                    <input
                      type="text"
                      className="w-full border border-slate-300 rounded-md p-2 text-sm text-slate-950 outline-none"
                      placeholder="Введіть назву колекції"
                      value={newCollectionName}
                      onChange={(e) => setNewCollectionName(e.target.value)}
                    />
                    <button
                      className="mt-2 bg-cyan-200 hover:bg-cyan-300 text-cyan-700 px-4 py-2 rounded-lg"
                      onClick={createNewCollection}>
                      Створити
                    </button>
                  </div>
                </div>
                <div className="flex justify-end mt-4">
                  <button
                    className="bg-gray-200 hover:bg-gray-300 text-gray-700 px-4 py-2 rounded-lg"
                    onClick={closeCollectionModal}
                  >
                    Закрити
                  </button>
                </div>
              </div>
            </div>
          )}
          <div className="mt-6">
            <h3 className="text-lg font-semibold mb-2 text-slate-800">Коментарі:</h3>
            {comments.length > 0 ? (
              <ul className="space-y-4">
                {comments.map(comment => renderComment(comment))}
              </ul>
            ) : (
              <p className="text-gray-500">Немає коментарів.</p>
            )}
          </div>
          <div className="mt-4">
            <textarea
              className="w-full p-2 border rounded-md text-black"
              rows="3"
              placeholder="Залиште свій коментар..."
              value={newComment}
              onChange={(e) => setNewComment(e.target.value)}
            />
            <button
              className="bg-primary hover:bg-cyan-400 text-white font-bold py-2 px-4 rounded mt-2"
              onClick={handleAddComment}
            >
              Відправити
```

```
            </button>
          </div>
          {recommendedStories.length > 0 && (
    <div className="mt-8">
      <h3 className="text-lg font-semibold mb-4 text-slate-800">Схожі історії:</h3>
      <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 gap-4">
      {recommendedStories.map(story => (
    <TravelStoryCard
      key={story._id}
      imgUrl={story.imageUrl}
      title={story.title}
      createdOn={story.createdOn}
      visitedLocation={story.visitedLocation}
      onClick={() => {
        openRecommendedStory(story);
      }}
      authorId={story.userId}
      currentUserId={currentUserId}
      onFavouriteClick={() => onUpdateFavourite(story)}

isFavourite={story.likes?.includes(currentUserId)}
    />
))}
      </div>
    </div>
)}
          <button
            className="mt-6 bg-gray-300 hover:bg-gray-400 text-gray-800 font-bold py-2 px-4 rounded"
            onClick={onClose}
          > Закрити </button>
      </div>
    );};
export default ViewTravelStory;
```

ProfilePage.jsx

```
import React, { useState, useEffect } from 'react';
import axiosInstance from '../../utils/axiosInstance';
import Navbar from "../../components/Navbar";
import Footer from "../../components/Footer";
import ProfileHeader from './components/ProfileHeader';
import ProfileNavigation from './components/ProfileNavigation';
import PublishedStoriesSection from './components/PublishedStoriesSection';
import CollectionsSection from './components/CollectionsSection';
import FavouriteStoriesSection from './components/FavouriteStoriesSection';
import backgroundImage from '../../assets/images/bg.jpg';
import AuthorAnnouncements from './components/AuthorAnnouncements';
const ProfilePage = () => {
 const [profileData, setProfileData] = useState(null);
 const [loading, setLoading] = useState(true);
 const [error, setError] = useState('');
 const [activeTab, setActiveTab] = useState('stories');
 const [announcements, setAnnouncements] = useState([]);
 const handleLogout = () => {
  localStorage.clear();
 };
 const fetchAnnouncements = async () => {
  try {
   const response = await axiosInstance.get(`/api/users/${profileData?._id}/announcements`);
setAnnouncements(response.data.announcements;
  } catch (err) {
   console.error('Помилка завантаження оголошень:', err);
  }
 };
 useEffect(() => {
  const fetchProfileData = async () => {
   setLoading(true);
   setError('');
   try {
    const response = await axiosInstance.get('/get-user');
     if (response.data && response.data.user) {
      setProfileData(response.data.user);
     } else {
      setError('Не вдалося завантажити дані профілю.');
     }
   } catch (error) {
    console.error('Помилка при отриманні даних профілю:', error);
     setError('Сталася помилка при завантаженні даних профілю.');
   } finally {
    setLoading(false);
   }
  };
  fetchProfileData();
 }, []);
 useEffect(() => {
  if (profileData?._id) {
   fetchAnnouncements();
  }
 }, [profileData?._id]);
 const handleTabChange = (tab) => {
```

```
  setActiveTab(tab);
  console.log("ProfilePage: Активна вкладка:", tab);
 };
 const handleAnnouncementUpdated = () => {
  fetchAnnouncements();
 };

 if (loading) {
  return <div className="container mx-auto py-10">Завантаження профілю...</div>;
 }
 if (error) {
  return <div className="container mx-auto py-10 text-red-500">{error}</div>;
 }
 if (!profileData) {
  return <div className="container mx-auto py-10">Профіль не знайдено.</div>;
 }
 return (
  <div className="bg-black text-white">
   <Navbar
    userInfo={profileData}
    onLogout={handleLogout}
    className="absolute top-0 left-0 right-0 z-20 bg-transparent border-b border-white/10"
   />
   <div className="hero relative bg-black mt-[-68px]" style={{ minHeight: '400px' }}>
    <div
     className="absolute inset-0 bg-image"
     style={{ backgroundImage: `url('${backgroundImage}')`, backgroundSize: 'cover', backgroundPosition: 'center' }}
    ></div>
    <div className="absolute inset-0 bg-gradient-to-b from-transparent via-black/30 via-70% to-black opacity-100"></div>
    <div className="container mx-auto relative z-10 py-10">
     <h1 className="text-3xl font-semibold mb-6">{profileData?.firstName} {profileData?.lastName}</h1>
    </div>
   </div>
   <div className="container mx-auto py-10">
    <div className="flex">
     <div className="w-1/12 pr-8 hidden lg:block">
     </div>
     <div className="flex-grow">
      <ProfileHeader userInfo={profileData} />
      <ProfileNavigation activeTab={activeTab} onTabChange={handleTabChange} />
      <div>
       {activeTab === 'stories' && <PublishedStoriesSection userId={profileData?._id} />}
       {activeTab === 'favourites' && <FavouriteStoriesSection userId={profileData?._id} />}
       {activeTab === 'collections' && <CollectionsSection />}
      </div>
     </div>
     <div className="w-1/4 pl-8 hidden lg:block">
      <div className="bg-cyan-100/80 p-4 rounded">
       {profileData?._id && <AuthorAnnouncements userId={profileData?._id} onAnnouncementUpdated={handleAnnouncementUpdated} />}
      </div>
     </div>
    </div>
   </div>
   <Footer />
  </div>
 );
};

export default ProfilePage;
```

Серверна частина:
story.controller.js

```
const WayforaStory = require("../models/wayforaStory.model");
const path = require("path");
const fs = require("fs");
// This controller handles the creation, retrieval, updating, and deletion of stories
exports.imageUpload = async (req, res) => {
  try {
    if (!req.file) {
      return res
        .status(400)
        .json({ error: true, message: "No image uploaded" });
    }
    const imageUrl = `http://localhost:8000/uploads/${req.file.filename}`;
    res.status(201).json({ imageUrl});
  } catch (error) {
    res.status(500).json({ error: true, message: error.message });
  }
};
exports.deleteImage = async (req, res) => {
```

```
  const { imageUrl } = req.query;
  if (!imageUrl) {
    return res
      .status(400)
      .json({ error: true, message: "imageUrl parameter is required" });
  }

  try{
    const filename = path.basename(imageUrl);
    const filePath = path.join(__dirname, "../uploads", filename);
    if (fs.existsSync(filePath)) {
      fs.unlinkSync(filePath);
      return res.status(200).json({ message: "Image deleted successfully" });
    } else {
      return res.status(404).json({ error: true, message: "Image not found" });
    }
  } catch (error) {
    return res.status(500).json({ error: true, message: error.message });
  }
};
exports.addWayforaStory = async(req, res) => {
  const { title, story, visitedLocation, imageUrl, landmarks } = req.body;
  const { userId } = req.user;
  if (!title || !story || !visitedLocation || !imageUrl) {
    return res.status(400).json({ error: true, message: "All fields are required" });
  }
  try {
    const wayforaStory = new WayforaStory({
      title,
      story,
      visitedLocation,
      userId,
      imageUrl,
      landmarks: landmarks ? landmarks.map(landmark => ({
        name: landmark.name,
        link: landmark.link,
        note: landmark.note,
        imageUrl: landmark.imageUrl
      })) : [],
    });
    await wayforaStory.save();
    res.status(201).json({ story: wayforaStory, message: "Added successfully" });
  } catch (error) {
    res.status(400).json({ error: true, message: error.message });
  }
};
exports.getAllStories = async (req, res) => {
  try {
    const stories = await WayforaStory.find()
      .sort({ isFavourite: -1 })
      .populate('userId', 'fullName');
    const storiesWithAuthorInfo = stories.map(story => ({
      ...story.toObject(),
      author: {
        username: story.userId ? story.userId.fullName : 'Невідомий автор',
        initials: story.userId && story.userId.fullName ? story.userId.fullName.split(' ').map(n=>n[0]).join('').toUpperCase() : '',
        displayName: story.userId ? story.userId.fullName : 'Невідомий автор',
      },
      authorId: story.userId._id,
      userId: story.userId._id
    }));
    res.status(200).json({ stories: storiesWithAuthorInfo });
  } catch (error) {
    console.error("Помилка при отриманні всіх історій:", error);
    res.status(500).json({ error: true, message: "Failed to fetch stories." });
  }
};
exports.getUserStories = async (req, res) => {
  const { userId } = req.params;
  const authenticatedUserId = req.user.userId;
  if (userId !== authenticatedUserId) {
    return res.status(403).json({ error: true, message: "Forbidden: You can only access your own stories." });
  }
  try {
    const stories = await WayforaStory.find({ userId: userId }).sort({ createdAt: -1 });
    res.status(200).json({ stories });
  } catch (error) {
    console.error("Error fetching user's stories:", error);
    res.status(500).json({ error: true, message: "Failed to fetch user's stories." });
  }
};
exports.editStory = async (req, res) => {
  const { id } = req.params;
  const { title, story, visitedLocation, imageUrl, landmarks, visitedDate } = req.body;
  const { userId } = req.user;

  if (!title || !story || !visitedLocation || !imageUrl) { return res
```

```
        .status(400)
        .json({ error: true, message: "All fields are required" });
    }
    try {
      const wayforaStory = await WayforaStory.findOneAndUpdate(
        { _id: id, userId: userId },
        {
          title,
          story,
          visitedLocation,
          imageUrl,
          landmarks: landmarks ? landmarks.map(landmark => ({
            name: landmark.name,
            link: landmark.link,
            note: landmark.note,
            imageUrl: landmark.imageUrl
          })) : [],
          visitedDate: visitedDate || null,
        },
        { new: true }
      );
      if (!wayforaStory) {
        return res.status(404).json({ error: true, message: "Story not found" });
      }
      res.status(200).json({ story: wayforaStory, message: "Story updated successfully" });
    } catch (error) {
      console.error("Error updating story:", error);
      res.status(500).json({ error: true, message: error.message });
    }
};
exports.deleteStory = async (req, res) => {
    const { id } = req.params;
    const { userId } = req.user;
    try {
      const wayforaStory = await WayforaStory.findOne({ _id: id, userId: userId });
      if (!wayforaStory) {
        return res
          .status(404)
          .json({ error: true, message: "Story not found" });
      }
      const imageUrl = wayforaStory.imageUrl;
      if (imageUrl) {
        const filename = path.basename(imageUrl);
        const filePath = path.join(__dirname, "../uploads", filename);
        if (fs.existsSync(filePath)) {
          fs.unlinkSync(filePath);
        } else {
          console.warn("File not found:", filePath);
        }
      }
      await WayforaStory.deleteOne({ _id: id, userId: userId });
      res.status(200).json({ message: "Story deleted successfully" });
    } catch (error) {
      console.error("Error deleting story:", error);
      res.status(500).json({ error: true, message: error.message });
    }
};
exports.updateIsFavourite = async (req, res) => {
    const { id } = req.params;
    const { userId } = req.user;
    try {
      const wayforaStory = await WayforaStory.findById(id);
      if (!wayforaStory) {
        return res.status(404).json({ error: true, message: "Story not found" });
      }

      const isLiked = wayforaStory.likes.includes(userId);
      if (isLiked) {
        wayforaStory.likes = wayforaStory.likes.filter(
          (likeUserId) => likeUserId.toString() !== userId.toString()
        );
      } else {
        wayforaStory.likes.push(userId);
      }
      await wayforaStory.save();
      res.status(200).json({ story: wayforaStory, message: "Favourite status updated successfully" });
    } catch (error) {
      res.status(500).json({ error: true, message: error.message });
    }
};
exports.removeFavouriteStory = async (req, res) => {
    const { storyId } = req.params;
    const { userId } = req.user;

    console.log("Запит на видалення улюбленої історії:", { storyId, userId });
    try {
      const user = await User.findById(userId);
      if (!user) {
        console.log("Користувача не знайдено:", userId);
```

```
        return res.status(404).json({ message: 'Користувача не знайдено.' });
      }
      console.log("Поточні улюблені історії користувача:", user.favourites);

      user.favourites = user.favourites.filter(
        (favouriteId) => String(favouriteId) !== String(storyId)
      );
      console.log("Оновлений список улюблених історій:", user.favourites);
      await user.save();
      res.status(200).json({ message: 'Історію успішно видалено з улюбленого.' });
    } catch (error) {
      console.error('Помилка при видаленні історії з улюбленого:', error);
      res.status(500).json({ message: 'Не вдалося видалити історію з улюбленого.' });
    }
};
exports.getFavouriteStories = async (req, res) => {
    const { userId: requestedUserId } = req.params;
    const { userId: loggedInUserId } = req.user;
    if (requestedUserId.toString() !== loggedInUserId.toString()) {
      return res.status(403).json({ error: true, message: "You are not authorized to view these favourite stories." });
    }
    try {
      const favouriteStories = await WayforaStory.find({ likes: loggedInUserId }).populate('userId', 'firstName lastName username');
      res.status(200).json({ favourites: favouriteStories });
    } catch (error) {
      res.status(500).json({ error: true, message: error.message });
    }
};
exports.searchStories = async (req, res) => {
    const { query } = req.query;
    if (!query) {
      return res.status(404).json({ error: true, message: "Query is required" });
    }
    try {
      const searchResults = await WayforaStory.find({
        $or: [
          { title: { $regex: query, $options: "i" } },
          { story: { $regex: query, $options: "i" } },
          { visitedLocation: { $elemMatch: { $regex: query, $options: "i" } } },
        ],
      }).sort({ isFavorite: -1 });

      res.status(200).json({ stories: searchResults });
    } catch (error) {
      res.status(500).json({ error: true, message: error.message });
    }
};
```

index.js

```
require("dotenv").config();
const config = require("./config.json");
const mongoose = require("mongoose");
const express = require("express");
const cors = require("cors");
const path = require("path");
const { authenticateToken } = require("./utilities");
const authController = require("./controllers/auth.controller");
const storyController = require("./controllers/story.controller");
const collectionController = require("./controllers/collection.controller");
const userController = require("./controllers/user.controller");
const announcementController = require("./controllers/announcement.controller");
const commentController = require("./controllers/comment.controller");
const upload = require("./multer");
mongoose.connect(config.connectionString);
const app = express();
app.use(express.json());
app.use(cors({ origin: "*" }));
app.post("/create-account", authController.createAccount);
app.post("/login", authController.login);
app.get("/get-user", authenticateToken, authController.getUser);
app.post("/image-upload", upload.single("image"), storyController.imageUpload);
app.delete("/delete-image", storyController.deleteImage);
app.post("/add-wayfora-story", authenticateToken, storyController.addWayforaStory);
app.get("/get-all-stories", storyController.getAllStories);
app.get("/get-user-stories/:userId", authenticateToken, storyController.getUserStories);
app.put("/edit-story/:id", authenticateToken, storyController.editStory);
```

```
app.delete("/delete-story/:id", authenticateToken, storyController.deleteStory);
app.put("/update-is-favourite/:id", authenticateToken, storyController.updateIsFavourite);
app.delete('/remove-favourite-story/:storyId', authenticateToken, storyController.removeFavouriteStory);
app.get("/get-favourite-stories/:userId", authenticateToken, storyController.getFavouriteStories);
app.get("/search", authenticateToken, storyController.searchStories);
app.post("/create-collection", authenticateToken, collectionController.createCollection);
app.get("/get-user-collections", authenticateToken, collectionController.getUserCollections);
app.post("/add-to-collection/:collectionId", authenticateToken, collectionController.addToCollection);
app.put('/edit-collection/:collectionId', authenticateToken, collectionController.editCollection);
app.delete('/delete-collection/:collectionId', authenticateToken, collectionController.deleteCollection);
app.get('/get-collection-landmarks/:collectionId', authenticateToken, collectionController.getCollectionLandmarks);
app.delete('/delete-landmark-from-collection/:collectionId/:landmarkId', authenticateToken, collectionController.deleteLandmarkFromCollection);
app.get("/user/:userId/public-profile", userController.getPublicProfile);
app.get("/users/:userId/travel-data", authenticateToken, userController.getTravelData);
app.post("/users/:userId/travel-data", authenticateToken, userController.postTravelData);

app.post("/api/users/:userId/announcements", authenticateToken, announcementController.createAnnouncement);
app.get("/api/users/:userId/announcements", announcementController.getUserAnnouncements);
app.put("/api/announcements/:announcementId", authenticateToken, announcementController.updateAnnouncement);
app.delete("/api/announcements/:announcementId", authenticateToken, announcementController.deleteAnnouncement);
app.get('/api/stories/:storyId/comments', commentController.getStoryComments);
app.post('/api/stories/:storyId/comments', authenticateToken, commentController.addStoryComment);
app.put('/api/comments/:commentId', authenticateToken, commentController.editComment);
app.delete('/api/comments/:commentId', authenticateToken, commentController.deleteComment);
app.use("/uploads", express.static(path.join(__dirname, "uploads")));
app.use("/assets", express.static(path.join(__dirname, "assets")));
app.listen(8000);
module.exports = app;
```