Хмельницький національний університет
Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

Столярчука Єгора Ігоровича
Прізвище, ім'я, по батькові студента(ки)

на здобуття ступеня вищої освіти Бакалавра

Вебзастосунок для планування
Назва теми
та документування туристичних подорожей

Галузь знань 12 «Інформаційні технології»

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

Шифр КвРІПЗ.2101090.01.16.ПЗ

Виконав студент IV курсу, група ІПЗ-21-1 _____ Єгор СТОЛЯРЧУК
Підпис Ім'я, ПРІЗВИЩЕ

Керівник канд. пед. наук, доцент _____ Наталія ПРАВОРСЬКА
Науковий ступінь, звання Підпис Ім'я, ПРІЗВИЩЕ

Нормоконтролер ст. викладач _____ Ганна БЕДРАТЮК
Підпис Ім'я, ПРІЗВИЩЕ

До захисту допускаю:
Завідувач кафедри інженерії програмного забезпечення _____ Леонід БЕДРАТЮК
Підпис Ім'я, ПРІЗВИЩЕ

2 червня 2025 р.

Хмельницький 2025

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій

Кафедра Інженерії програмного забезпечення

Рівень вищої освіти Перший (бакалаврський)

Галузь знань 12 «Інформаційні технології»

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Л. П. Бедратюк
2.01 2025 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

Столярчуку Єгору Ігоровичу

Прізвище, ім'я, по батькові студента

1. Тема кваліфікаційної роботи Вебзастосунок для планування та документування туристичних подорожей

Керівник кваліфікаційної роботи Праворська Наталія Іванівна, канд. пед. наук, доцент

Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 07.02.2024 р. № 23

2. Строк подання студентом роботи на кафедру 06.06.2025 р.

3. Вихідні дані до роботи Матеріали переддипломної практики

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)
Дослідження предметної області та постановка задачі, проєктування вебзастосунку, програмна реалізація та тестування

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень)
Презентаційні матеріали (слайди, 18 шт.), діаграма варіантів використання, діаграма архітектури вебзастосунку за шаблоном MVC, спрощена ER-діаграма, ER-діаграма, загальна діаграма архітектури вебзастосунку, діаграма послідовностей для процесу автентифікації користувача діаграма послідовностей для процесу додавання історії подорожі, діаграма компонентів вебзастосунку

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Бедратюк Г. І., ст. викладач		
Антиплагіат	Форкун Ю. В., к.т.н., доцент	12.05.2025	28.05.2025

7. Дата видачі завдання «2» 01 2025р.

КАЛЕНДАРНИЙ ПЛАН

Назва етапів (розділів) кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1 Ознайомлення з тематикою дипломного проєктування, визначення та узгодження індивідуальних тем кваліфікаційних робіт (КвР)	01.12 – 31.12.2024	
2 Збір матеріалу за темою КвР; дослідження предметної області, в якій планується використання програмного забезпечення (ПЗ), визначення задач та вимог, розробка технічного завдання	01.01 – 20.02.2025	
3 Проєктування програмного забезпечення	21.02 – 20.03.2025	
4 Програмна реалізація з використанням відповідних засобів розробки. Тестування ПЗ	21.03 – 30.04.2025	
5 Написання вступу, загальних висновків, оформлення переліку джерел посилання та додатків. Оформлення пояснювальної записки КвР згідно вимог	01.05 – 25.05.2025	
6 Попередній захист КвР	Травень 2025	Згідно графіка
7 Перевірка КвР на плагіат, нормоконтроль, отримання відгуків, рецензій та інших супровідних документів. Брошурування (зшиття) пояснювальної записки.	26.05 – 30.05.2025	
8 Здача КвР на кафедру; підготовка КвР для розміщення у репозитарії ХНУ; підготовка до захисту та захист КвР	з 01.06.2025	

Студент _____ Столярчук Є. І.
Підпис Ініціали, прізвище

Керівник роботи _____ Праворська Н. І.
Підпис Ініціали, прізвище

АНОТАЦІЯ

Тема кваліфікаційної роботи: «Вебзастосунок для планування та документування туристичних подорожей».

Автор роботи: Столярчук Єгор Ігорович.

Керівник роботи: Праворська Наталія Іванівна.

Пояснювальна записка: 73 с., 22 рис., 3 табл., 4 дод., 42 джерела.

Графічна частина: 2 креслення.

ВЕБЗАСТОСУНОК, ТУРИСТИЧНІ ПОДОРОЖІ, REACT, EXPRESS.JS, NODE.JS, MONGODB, MVC

Метою роботи є розробка вебзастосунку для планування та документування туристичних подорожей, який дозволить легко та зручно організовувати та зберігати інформацію про подорожі із можливістю додання локацій у список бажаного.

У кваліфікаційній роботі було проведено аналіз предметної області, досліджено популярні сервіси, розроблено функціональні та нефункціональні вимоги, спроєктовано архітектуру системи, обґрунтовано вибір стеку технологій (MERN), реалізовано основні модулі вебзастосунку та виконано тестування.

Для реалізації клієнтської та серверної частини застосунку використано JavaScript-орієнтований MERN-стек (MongoDB, Express.js, React, Node.js). Для зберігання зображень інтегровано сервіс Cloudinary. Для автентифікації користувачів використано механізм JSON Web Token (JWT).

Запропонований вебзастосунок надає користувачам інтуїтивно зрозумілий інтерфейс, функціональність збереження та перегляду історій подорожей, можливість додавання до списку бажаного, пошуку та фільтрації контенту, а також експорту в PDF-формат.

Вебзастосунок може бути використаний всіма, хто бажає зручно організовувати власні подорожі в єдиному цифровому середовищі.

02.06.25
Дата

Підпис

ВІДОМІСТЬ ДОКУМЕНТІВ

№ рядка	Формат	Позначення документа	Найменування документа	К-сть аркушів	№ екз.	Примітка
			Текстові документи			
1	А4	КвРІПЗ.2101090.01.16.ПЗ	Пояснювальна записка	73		
2	А4		Завдання на кваліфікаційну роботу	1		
3	А4		Анотація	1		
			Графічні документи			
4	А4	КвРІПЗ.2101090.01.16.Е8	Логічна модель бази даних	1		
5	А4	КвРІПЗ.2101090.01.16.Е8	Компоненти вебзастосунку	1		

Змн.	Арк.	№ докум.	Підпис	Дата	КвРІПЗ.2101090.01.16.ВД		
Виконав		Столярчук Є. І.		02.06	Вебзастосунок для планування та документування туристичних подорожей	Літ.	Арк. 1 / Аркушів 1
Керівник		Праворська Н. І.		02.06			
Рецензент		Кашубська М. В.		02.06			
Н. Контр.		Бедратюк Г. І.		02.06			
Зав. Каф.		Бедратюк Л.П.		02.06	Пояснювальна записка	ХНУ, ІПЗ-21-1	

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ 8
ВСТУП 9
1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ 12
1.1 Змістовний аналіз предметної області, її структурних та функціональних особливостей 12
1.2 Аналіз наявного програмно-технічного забезпечення предметної області 16
1.3 Визначення функціональних та нефункціональних вимог до вебзастосунку 21
1.4 Висновки дослідження предметної області та постановки задачі 26
2 ПРОЄКТУВАННЯ ВЕБЗАСТОСУНКУ 27
2.1 Аналіз та вибір архітектури вебзастосунку 27
2.2 Проєктування структури даних та моделі бази даних 32
2.3 Проєктування інтерфейсу користувача 36
2.4 Аналіз та вибір технологій і методів реалізації вебзастосунку 41
2.5 Висновки проєктування архітектури та структури програмного забезпечення 46
3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ВЕБЗАСТОСУНКУ 47
3.1 Розроблення бази даних 47
3.2 Розроблення програмних модулів 50
3.3 Керівництво користувача 57
3.4 Технічні характеристики вебзастосунку 58
3.5 Тестування вебзастосунку 59
3.5.1 Вибір та обґрунтування методів тестування вебзастосунку 60
3.5.2 Верифікація та валідація вебзастосунку 61
3.5.3 Аналіз результатів тестування вебзастосунку 64

3.6 Висновки програмної реалізації, налагодження та тестування вебзастосунку 66
ВИСНОВКИ 67
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ 70
ДОДАТОК А Технічне завдання 74
ДОДАТОК Б Діаграми діяльності 80
ДОДАТОК В Код програми 83
ДОДАТОК Г Презентаційні слайди 145

ПЕРЕЛІК СКОРОЧЕНЬ

СКБД	–	система керування базами даних
API	–	Application Programming Interface
CDN	–	Count Delivery Network
DOM	–	Document Object Model
ER	–	Entity-Relationshop
HTTP	–	HyperText Transfer Protocol
HTTPS	–	HyperText Transfer Protocol Secure
JSON	–	JavaScript Object Notation
JWT	–	JSON Web Token
MERN	–	MongoDB, Express, React, Node.js
REST	–	Representational State Transfer
UI	–	User Interface
URI	–	Uniform Resource Identifier

ВСТУП

У сучасному світі інформаційні технології відіграють важливу роль у людському житті та у різних сферах її діяльності, включаючи подорожі та туризм. Саме подорожі – це те, що допомагає людині краще пізнати себе, вийти із зони комфорту на зустріч новим пригодам, які часто можуть бути несподіваними, але залишаються в пам’яті людей на довгий час. Вони відкривають світ незнайомих культур, які цікаво досліджувати, а також сприяють новим знайомствам. Це допомагає здобувати нові навички та розширювати світогляд [1].

Значна кількість мандрівників по всьому світу та зростання популярності соціальних мереж, що формує звичку ділитись яскравими моментами життя серед друзів, родичів та просто незнайомих людей, роблять онлайн-інструменти, які дозволяють поширювати дописи про життя, враження, локації для подорожей – незамінними для великої кількості людей [2]. Саме через це часто є потреба в пошуку ефективного інструмента для збереження спогадів. Вебсайти або мобільні застосунки, створені для цього, часто пропонують додаткові функції, наприклад, планування нових маршрутів, поширення досвіду з іншими тощо.

Подорожі завжди є актуальною сферою діяльності, адже однією з переваг такої активності є обмін культурами, що відіграє важливе значення для розвитку взаєморозуміння серед представників різних країн. Саме під час подорожей люди діляться звичаями та традиціями своєї країни, що допомагає їм краще розуміти одне одного [3].

Актуальність теми даної кваліфікаційної роботи зумовлена постійним розвитком технологій та зміною традиційних способів зберігання спогадів про подорожі у вигляді фотоальбомів на онлайн-альтернативи. Беручи до уваги активність користувачів в інтернеті, виникає потреба в зберіганні такого типу інформації в одному зручному місці, до якого можна буде отримати доступ в будь-який час.

Це робить сервіс для планування та документування туристичних подорожей важливим інструментом для людей, що люблять зберігати найкращі моменти із подорожей та цінують власний час. Користувачі зможуть публікувати історії, культурні практики, рецепти інших народів та багато іншого, що дозволить створити власний зручний архів, де кожен може зберегти свої емоції та враження під час турів. Зібрані історії підштовхуватимуть людину на усвідомлення на оцінки культурного досвіду, який вона набула за певний проміжок часу. Це також стане мотивацією для подальших подорожей, відвідавши нові місця із зануренням в інші культури [4].

Розроблений вебзастосунок буде корисним для великої кількості користувачів: як для мандрівників, які постійно подорожують і для них є необхідністю організувати свої подорожі в одному місці, так і для тих, хто майже не подорожує або хоче почати. Основною аудиторією будуть переважно професійні мандрівники, блогери, а також сімейні пари.

Сучасний ринок програмних рішень для мандрівників демонструє різні платформи для планування та документування туристичних подорожей. Однак більшість з них орієнтовані виключно на планування маршрутів або мають такий підхід, який обмежує збереження особистих історій. Також багато таких сервісів не інтегруються із соціальними мережами та не дають змоги гнучко управляти дописами користувачів. Це знижує їх популярність серед тих, хто часто подорожує.

Запропоноване рішення, яке призначене для індивідуального використання, в майбутньому може бути використане для інтеграції з туристичними агенціями, що дозволить користувачам отримувати рекомендації, базуючись на основі їх попереднього досвіду. Окрім цього, можливість інтеграції з соціальними мережами дозволить користувачам обмінюватися своїми історіями, що значно розширить базовий функціонал вебзастосунку та дозволить залучити більшу кількість людей.

Метою даної кваліфікаційної роботи є розробка вебзастосунку для планування та документування туристичних подорожей, який дозволить легко

та зручно організовувати та зберігати інформацію про подорожі із можливістю додання локацій у список бажаного. Даний вебзастосунок повинен покращити досвід користувача, пропонуючи зрозумілий інтерфейс та функціонал, використовуючи сучасні технології веброзробки.

Для досягнення поставленої мети потрібно вирішити такі завдання:

– провести аналіз наявних рішень для планування та документування туристичних подорожей;

– визначити ключові аспекти та вимоги до функціоналу;

– спроєктувати архітектуру розроблюваного продукту для забезпечення ефективної роботи сервісу;

– розробити вебзастосунок із можливістю зберігання вражень про подорожі;

– провести тестування роботи вебзастосунку на наявність помилок та загальної працездатності та продуктивності;

– проаналізувати успішність розробленого продукту та сформувати цілі для його покращення в майбутньому;

1. ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Змістовний аналіз предметної області, її структурних та функціональних особливостей

Вебсайт – це цифровий ресурс, який містить в собі інформацію, що охоплює різноманітні сфери. З технічного погляду це платформа, яка розміщується на сервері та складається з інших вебсторінок, які пов’язані між собою. Вебсайти розміщуються на серверах і для того, щоб мати до них доступ, потрібен веббраузер. Основним завданням вебсайту є надання доступу до контенту на різну тематику [5].

Вебсайти розробляються з різними цілями та поділяються певні типи. Вони можуть бути як особисті, так і комерційні, використовуючись в інформаційно-розважальних або бізнес-цілях. Визначення функціональних важливостей, що відповідатимуть очікуванням аудиторії, є важливим питанням, яке постає перед розробкою вебсайту. В контексті туризму, сервіс TripAdvisor пропонує користувачам відгуки про різноманітні розважальні заклади та готелі, Booking.com надає змогу бронювати житло, а Google Maps пропонує карти для навігації. Кожен з цих сайтів вирішує конкретні проблеми туристів.

Загалом, можна виділити наступні завдання, які виконуються вебсайтами:

– інформування: надання та знаходження корисної інформації, надання послуг тощо;

– надання послуг: можливість робити онлайн-замовлення товарів в різноманітних магазинах;

– збільшення аудиторії: залучення нових клієнтів;

– підвищення взаємодії та довіри: підтримка комунікації між людьми за допомогою інтерактивних складових (чати підтримки, форми для заповнення зі зворотним зв’язком тощо) [6].

Сформувавши цілі створення вебсайтів, можна зробити висновок, що перед розробкою вебсайту завжди потрібно чітко визначати його ключову мету та завдання.

Серед основних видів вебсайтів можна виділити наступні:

– лендінг (Landing Page): окремо створена вебсторінка, основним призначенням якої є обмеження непотрібних елементів, які можуть відволікати користувачів, що в свою чергу спонукає їх до конкретних дій та взаємодії із вебсайтом (наприклад, підписка на розсилку або можливість залишити заяву). Зазвичай це робиться в рекламних цілях та для залучення більшої кількості потенційних клієнтів, тому такі вебсайти мають чітку структуру, яка включає привабливий заголовок, невеликий опис із пропозицією, форму для введення даних тощо;

– бізнес-сайт: сайт, що робиться для компанії та відображає інформацію про неї, чим займається, послуги, що надаються, контактну інформацію для зв'язку із членами організації та інше. Така інформація сильно спрощує взаємодію та містить всю необхідну інформацію. Додатково тут можна додати секцію із відгуками інших користувачів або замовників (в залежності чим займається компанія), розділ з часто відповідями на часто поширенні запитання у форматі "Запитання-відповідь" тощо. Часто такі сайти мають інтеграцію з картами, соціальними мережами та формами для заповнення;

– інтернет-магазин (E-commerce): онлайн-торгівля є дуже поширеною, адже все більше людей відмовляється від традиційних покупок у звичайних магазинах та переходять на замовлення товарів онлайн. Очікується, що у 2025 році глобальні продажі e-commerce досягнуть 6,86 трильйона доларів, що на 8,37% більше, порівняно із 2024 роком [7]. Звичайно, категорії товарів для замовлення є дуже різноманітними, що дає змогу невеликим підприємствам працювати виключно онлайн;

– блог: ресурс, який створюється для публікації авторських статей стосовно певних подій чи ситуацій, а також для висловлювання своїх думок, роздумів, аналітики і т.д. Може бути на будь-яку тематику: спорт, технології,

кулінарія, розваги, мистецтво. Популярність блогів зросла завдяки розвитку соціальних мереж, можливостям монетизації та в зміні у способах споживань інформації;

– новинний ресурс: альтернативна телевізору та друкованій пресі є популярним способом серед людей різного віку легко отримувати актуальну інформацію у світі з будь-якого місця та в будь-який час. Вони забезпечують постійний доступ до усіх останніх подій, що трапились та дають змогу користувачам самостійно обирати цікаві для них теми;

– інформаційний ресурс: сайт, призначений для надання інформації, який може бути корисним абсолютно для кожного, адже залежно від пошукових запитів, такий сайт надає корисну інформацію, яка слугує відмінним джерелом знань і допомагає у вирішенні проблем. Такі ресурси зазвичай мають добре структуровану інформацію, що спрощує пошук необхідних даних;

У сучасному світі є звичайною практикою поєднувати декілька функцій одночасно для одного вебсайту. Наприклад, сайт для бізнесу може містити елементи форуму. Але в першу чергу завжди важливо продумувати сайт так, щоб він був максимально зручним та зрозумілим для користувачів різного віку та з різною метою використання вебзастосунку.

Вебзастосунок для планування та документування туристичних подорожей – це платформа, яка дозволяє користувачам зберігати важливі моменти з подорожей, створюючи особистий персоналізований архів дописів з подорожами. Це особливо корисно, адже кожна поїздка може бути детально описана з уточненням всіх деталей із доданням власних думок та ідей та прикріпленням відповідного фото, яке найкраще описує дану пригоду.

Основною проблемою є відсутність зручного інструменту для комплексного збереження та організації даних про подорожі. Наприклад, користувачі часто зберігають фото в Google Photos, а нотатки в Notes, що призводить до фрагментації інформації та ускладнює цілісне сприйняття досвіду.

Також існує потреба в пошуку за датами чи періодами, чого не забезпечують більшість існуючих рішень. Через це користувачі не можуть швидко знайти інформацію про конкретну поїздку, наприклад, згадати, коли саме були в певному місті.В процесі розробки будь-якого сайту завжди потрібно розуміти, що він має бути інтуїтивно зрозумілим для кінцевої аудиторії. Також важливим аспектом є наявність інтерактивних елементів, унікальних особливостей вмісту, що відрізнятимуть один продукт від іншого.

Спостерігаючи за динамікою пригод користувача, він завжди слідкуватиме за тим, коли він був більш активним та зможе візуалізувати свої подорожі. Завдяки можливості розташування записів у правильному порядку, підвищиться ефективність взаємодії користувача із вебсайтом. Тому тут важливим елементом є автоматизація процесів обробки та передачі різноманітної інформації. Має бути передбачено збереження інформації у структурованому вигляді із можливістю фільтрації даних для швидкого пошуку необхідного спогаду.

Важливо визначити основні групи кінцевих користувачів з їхніми потребами:

– люди, які часто подорожують: потреба у зберіганні великої кількості дописів, необхідність категоризації та сортування зі швидким пошуком серед записів, потреба у додатковій статистичній статистиці щодо мандрівок користувача. Прикладом може бути людина, яка кожного місяця відвідує декілька міст та їй потрібна чітка організація всіх цих подорожей;

– початківці в цій сфері: потреба у плануванні подорожей та зрозумілому інтерфейсі. В даному випадку людина може використовувати список бажаного, щоб у подальшому здійснити свою першу подорож;

– блогери: необхідність в організації записів за різними характеристиками та критеріями, експорту даних;

Процес перетворення вхідної інформації від користувача у вихідну в контексті даного вебзастосунку складається з таких етапів:

– збір первинної інформації: включає збір персональних даних користувача (ім’я, електронна пошта, пароль), інформацію про подорож (назва, дата, опис, теги, локація, фото), дані для планування (місце, приблизна дата);

– обробка даних: перевірка та валідація отриманої інформації, структурування та сортування дописів за різними характеристиками;

– генерування вихідної інформації: відображення записів з подорожами, коректне отримання результатів пошуку, візуалізація статистичних даних, як ось кількість всіх записів про подорож, а також згенеровані PDF-звіти про подорожі для друку або поширення;

На даному етапі також важливо врахувати питання, що можуть виникнути з безпекою. Адже розробка вебзастосунку для планування та документування туристичних подорожей повинна бути надійним сховищем для вражень, думок та натхнення людей. Для захисту автентифікаційних даних варто застосовувати хешування паролів із використанням відповідних алгоритмів, як ось bcrypt. Передачу даних необхідно здійснювати через HTTPS, а для автентифікації використовувати JWT-токени з обмеженим часом дії. Безпека також повинна підтримуватись через логування всіх дій та періодичні перевірки на вразливості.

Отже, важливо сформувати основні проблеми, які вирішуватиме розроблюване програмне забезпечення:

– відсутність зручного інструменту для ведення персонального щоденника подорожей;

– відсутність фільтрації та швидкого пошуку;

– відсутність списку бажаного;

– проблеми із доступністю та безпекою.

Таким чином, розробка сервісу для документування туристичних подорожей вирішить низку проблем, пов’язаних з організацією та структуруванням даних, що стосуються тематики вебсайту, із перспективою сформувати спільноту людей, які обожнюють подорожі.

1.2 Аналіз наявного програмно-технічного забезпечення предметної області

Перед розробкою вебзастосунку для планування та документування туристичних подорожей потрібно провести огляд наявного програмного забезпечення. Такий аналіз дозволить виявити оптимальні рішення та можливості для впровадження, які є актуальними на даний момент.

Різноманітні платформи для планування та документування туристичних подорожей мають схожі можливості, але мають свої переваги та недоліки.

Одним з найбільш популярних сервісів даної тематики є Travel Diaries [8]. Нижче наведено головне меню даної платформи (Рисунок 1.1):

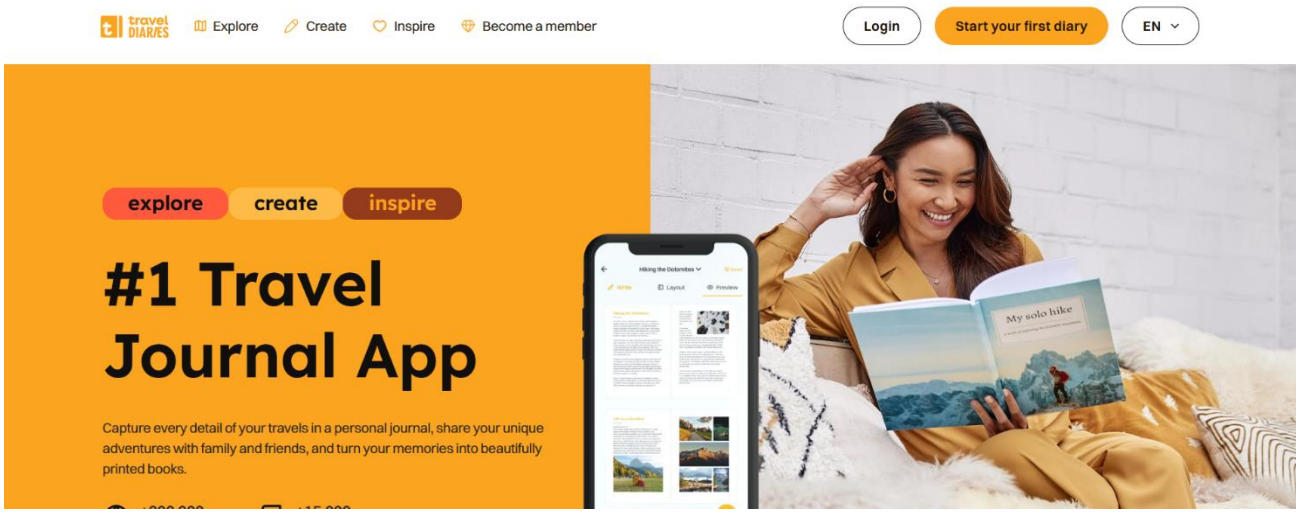


Рисунок 1.1 – Головне меню Travel Diaries

Travel Diaries – дуже популярний застосунок для створення та оформлення особистих щоденників подорожей, який пропонує зручно зберігати свої спогади, ділитись ними та перетворювати їх в друкований варіант. На даний момент існує лише онлайн версія щоденника на різних пристроях: планшети, телефони, комп'ютери. Даний сервіс підтримує додання необмеженої кількості сторінок в режимі онлайн редагування та 298 сторінок для друкованої версії книги. Вартість друкованих варіантів книг залежить від типу паперу,

кількості сторінок та обкладинки. Також тут є можливість прикріплення карти, фото, використання шаблонів, вибору різних шрифтів, зміни їхнього розміру та кольору тощо.

Окрім цього переваги Travel Diaries охоплюють зручний редактор тексту з можливістю додавати інші елементи. Також сервіс пропонує створення персоналізованих щоденників, а також їх захист за допомогою PIN-коду, щоб зробити його приватним.

Сайт оформлений у жовтих відтінках, що в свою чергу додає надихальну атмосферу для людей, що подорожують. Головна сторінка містить фото, які чітко передають концепцію сервісу, а інтерфейс зрозумілий та містить чітку навігацію, використовуючи приємні шрифти. Кнопки, що закликають до певних дій, виокремлюються за допомогою інших кольорів, що відповідає сучасним правилам дизайну та досвіду користувача.

Далі розглянуто вебсайт Daybook [9]. Нижче наведено головне меню сервісу (Рисунок 1.2):

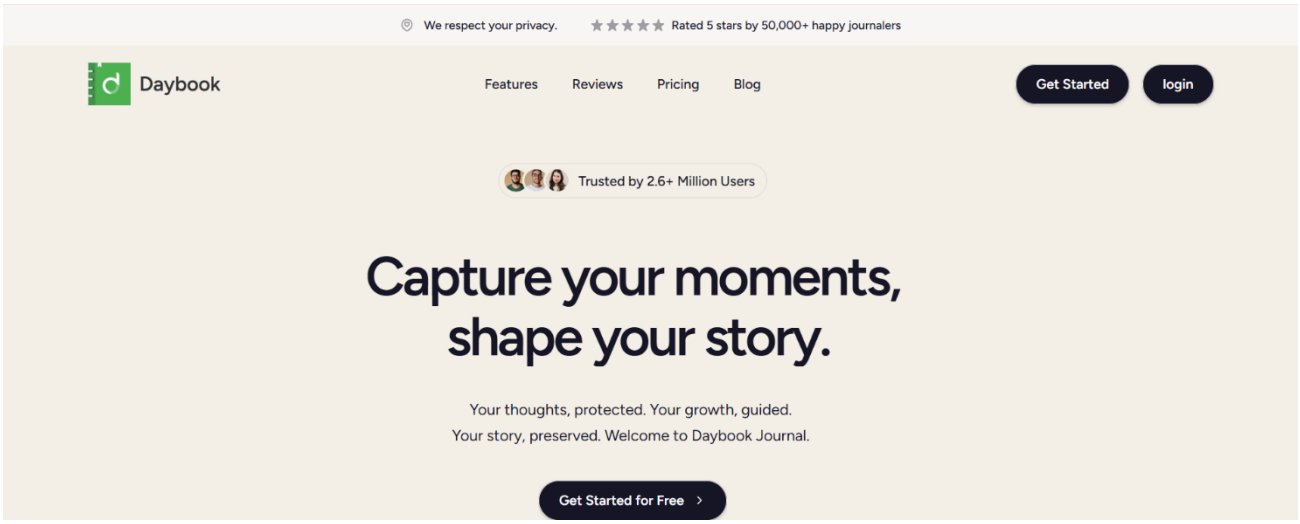


Рисунок 1.2 – Головне меню Daybook

Daybook – це популярний сервіс для документування історії подорожей, який доступний на різних платформах: Android, iOS, Web, Alexa та дозволяє синхронізувати дописи між різними пристроями. Для базового використання

достатньо безплатної версії, а для тих, хто хоче отримати максимум можливостей, Daybook пропонує преміумверсію, яка значно розширює функціонал, додаючи аналіз слів у тексті, використання ШІ, зберігання фото із високою якістю тощо. Перед купівлею преміумверсії сервіс пропонує спробувати безплатний пробний період, щоб краще ознайомитись із перевагами даної версії.

Daybook надає багато корисних функцій, щоб спростити процес ведення історії подорожей легшим. Користувач має змогу додавати нотатки вручну або за допомогою перетворення голосу в текст, що дозволяє швидко передавати свої думки без необхідності друку. Окрім цього пропонується відстеження настрою та активності відвідувача сервісу задля аналізу його емоційного стану.

Даний сервіс нараховує більше 2.6 мільйони користувачів по всьому світу та понад 50 тисяч позитивних відгуків, що свідчить про хорошу якість його роботи.

Наступним сервісом є Polarsteps [10]. Головне меню даного застосунку показано нижче (Рисунок 1.3):

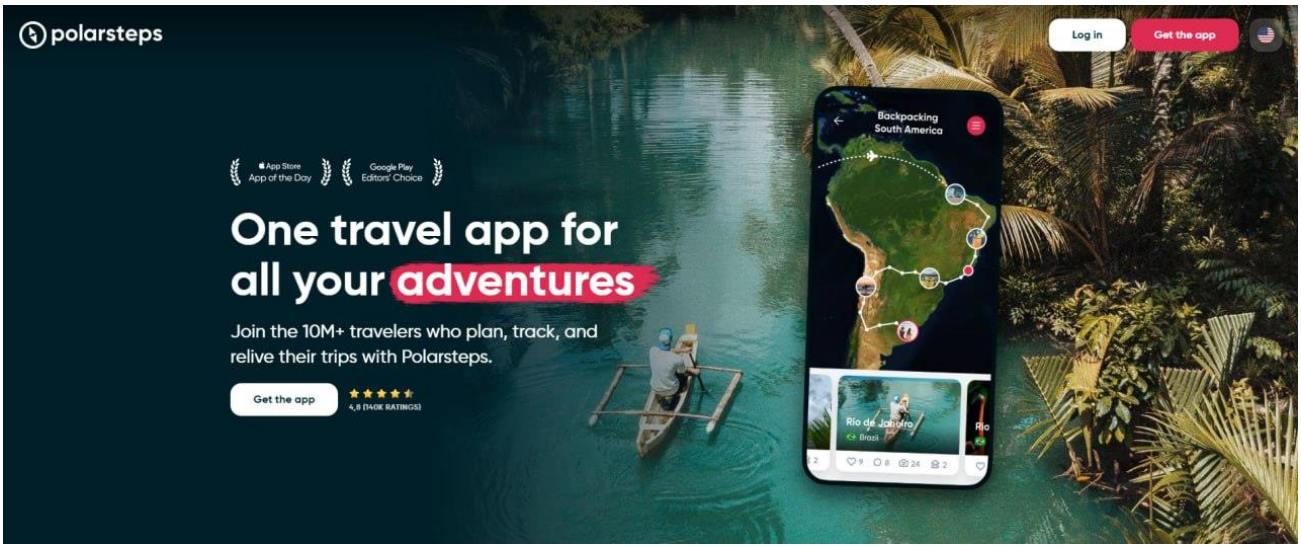


Рисунок 1.3 – Головне меню Polarsteps

Polarsteps – дуже популярний сервіс для відстеження подорожей, ключовою особливістю якого є запис маршрутів, створюючи цікаву

інтерактивну карту з місцями, де була людина. Polarsteps використовує GPS та зафіксовує людські переміщення навіть без підключення до інтернету. Як тільки з'являється стабільне підключення до інтернету, сервіс автоматично синхронізовує дані та вони стають доступними для користувача.

Polarsteps має зручний інтерфейс і дозволяє створювати персоналізовані нотатки, якими можна ділитися з іншими людьми. Важливою особливістю є можливість перегляду подорожей інших користувачів, що робить даний сервіс не лише хорошим щоденником, а й інструментом для надихання та дослідження нових місць.

Сервіс є безплатним та не містить реклами, проте має певні обмеження. Головним недоліком є те, що даний сервіс не завжди правильно відображає маршрут подорожі. Також інколи може виникнути проблема з тим, що користувач не має можливості вручну додати тип транспорту між різними точками. Хоча даний сервіс добре справляється із завданням зберігання вже здійснених поїздок, тут відсутня підтримка збереження даних про бронювання подорожей, що не підходить для планування нових пригод. Незважаючи на це, дана платформа має хороший функціонал та стабільні оновлення.

На основі описаних вище вебсайтів, нижче показано їхню порівняльну характеристику (Таблиця 1.1):

Таблиця 1.1 – Порівняльна характеристика вебсайтів для планування та документування туристичних подорожей

Сервіс	Переваги	Недоліки
Travel Diaries	Зручний редактор тексту з підтримкою шаблонів, прикріпленням карт та фото, можливість створення друкованої версії нотаток та використання PIN-коду	Обмеження у 298 сторінок при друкуванні щоденника.

Продовження таблиці 1.1

Daybook	Доступний на багатьох платформах, можливість синхронізації між пристроями та перетворення голосу в текст	Багато функцій доступні лише при купівлі преміумверсії сервісу
Polarsteps	Автоматичне відстеження маршруту користувача за допомогою GPS, робота без інтернету із синхронізацією даних при його появі, перегляд маршрутів інших людей	Можливість некоректного відображення маршрутів. Проблема з тим, що користувач не має можливості вручну додати тип транспорту між різними точками

1.3 Визначення функціональних та нефункціональних вимог до вебзастосунку

Процес аналізу вимог є дуже важливим на етапі розробки будь-якого програмного забезпечення для формування уявлення про майбутній функціонал системи. Ефективне визначення вимог допомагає усунути проблеми в майбутньому. Усі вимоги поділяються на два основні види: функціональні та нефункціональні.

Функціональні вимоги описують безпосередньо поведінку системи та описують як розроблюваний продукт має поводитись при різних обставинах. Вони формуються на сценаріях використання із детальним описом кожного аспекту взаємодії користувача та системи. При розробці таких вимог потрібно чітко розуміти як система повинна працювати та її взаємодію із кінцевим користувачем.

Функціональні вимоги є наступними:

– можливість створювати обліковий запис, використовуючи свої персональні дані;

– перевірка автентифікаційних даних при спробі входу у систему та автоматичне збереження останнього сеансу для швидкого доступу до вебзастосунку;

– можливість перегляду та редагування особистої інформації у профілі, включаючи персональні дані;

– додання нових записів про подорожі має охоплювати введення назви допису, опис, прикріплення фото, вибір дати, локації, а також встановлення тегу, що відповідає змісту доданої подорожі;

– редагування вже створених дописів з подорожами дозволятиме змінювати інформацію про відвідане місце та має бути доступним всім користувачам у будь-який час;

– реалізація системи пошуку, що дозволяє швидко знаходити потрібні подорожі за назвою або датою, використовуючи інтерактивний календар;

– використання фільтрів сортування, щоб в першу чергу отримувати записи, які помічені, як “Обране”;

– можливість зберігання певних місць у списку бажаного для планування майбутніх подорожей;

– можливість експорту записів про подорож у форматі PDF, щоб дозволити користувачам ділитися своїми мандрівками у зручному форматі з іншими користувачами.

Нефункціональні вимоги в свою чергу більше спрямовані на очікування користувачів та описують те, як саме програмне забезпечення виконуватиме свої функції, охоплюючи аспекти, які не були описані у функціональних вимогах. Ці вимоги необхідні, адже можуть покращити взаємодію між користувачем та розроблюваною системою [11].

Нефункціональні вимоги є наступними:

– час завантаження вебсайту повинен бути мінімальним;

– час перемикання між усіма сторінками не повинен перевищувати 3 секунди;

– реалізація ефективної індексації запитів та оптимізація взаємодії із базою даних прискорить отримання інформації;

– відповідання сучасним стандартам безпеки при автентифікації та управлінні доступом;

– зберігання паролів у захищеному вигляді;

– реалізація JSON Web Token (JWT) у процесі входу користувачів на вебсайт;

– розробка інтерфейсу повинна забезпечувати зрозуміле розташування кнопок та інших складових вебсайту;

– підтримка останніх версій сучасних браузерів із забезпеченням коректної працездатності функціоналу вебсайту;

– вебсайт повинен бути доступним впродовж усієї доби, за винятком часу, відведеного на технічне обслуговування.

Функціональні та нефункціональні вимоги зосереджені на різних, але однаково важливих елементах системи, отже, їх потрібно грамотно поєднувати під час розробки. Це особливо важливо, коли постає питання покращення користувацького досвіду та забезпечення стабільності роботи програмного забезпечення.

Формування даних вимог є важливим аспектом, оскільки вони визначають функціональність розроблюваного програмного забезпечення та якість його роботи.

Гармонійне поєднання цих вимог дає змогу створити як ефективну, так і зручну систему. Функціональні вимоги беруть на себе виконання всіх основних завдань, таких як обробка запитів користувача, взаємодія з базою даних чи реалізація певної бізнес-логіки вебзастосунку. В свою чергу нефункціональні вимоги впливають на досвід використання того чи іншого програмного забезпечення та визначають питання безпеки, стійкості до навантаження, швидкості роботи тощо [12].

Для більш детального представлення та розуміння вимог, потрібно розробити діаграму варіантів використання. Вона дозволить краще показати та

структурувати функціонал вебсайту та взаємодію між користувачем та системою. Така діаграма має на меті відобразити лише основний функціонал, а не допоміжні деталі системи [13].

Діаграми варіантів використання є універсальним інструментом, що підходять для багатьох ситуацій, що стосуються, наприклад, проєктування систем, збору вимог, документації тощо [14].

Діаграму варіантів використання для вебзастосунку планування та документування туристичних подорожей детально зображено нижче (Рисунок 1.4):

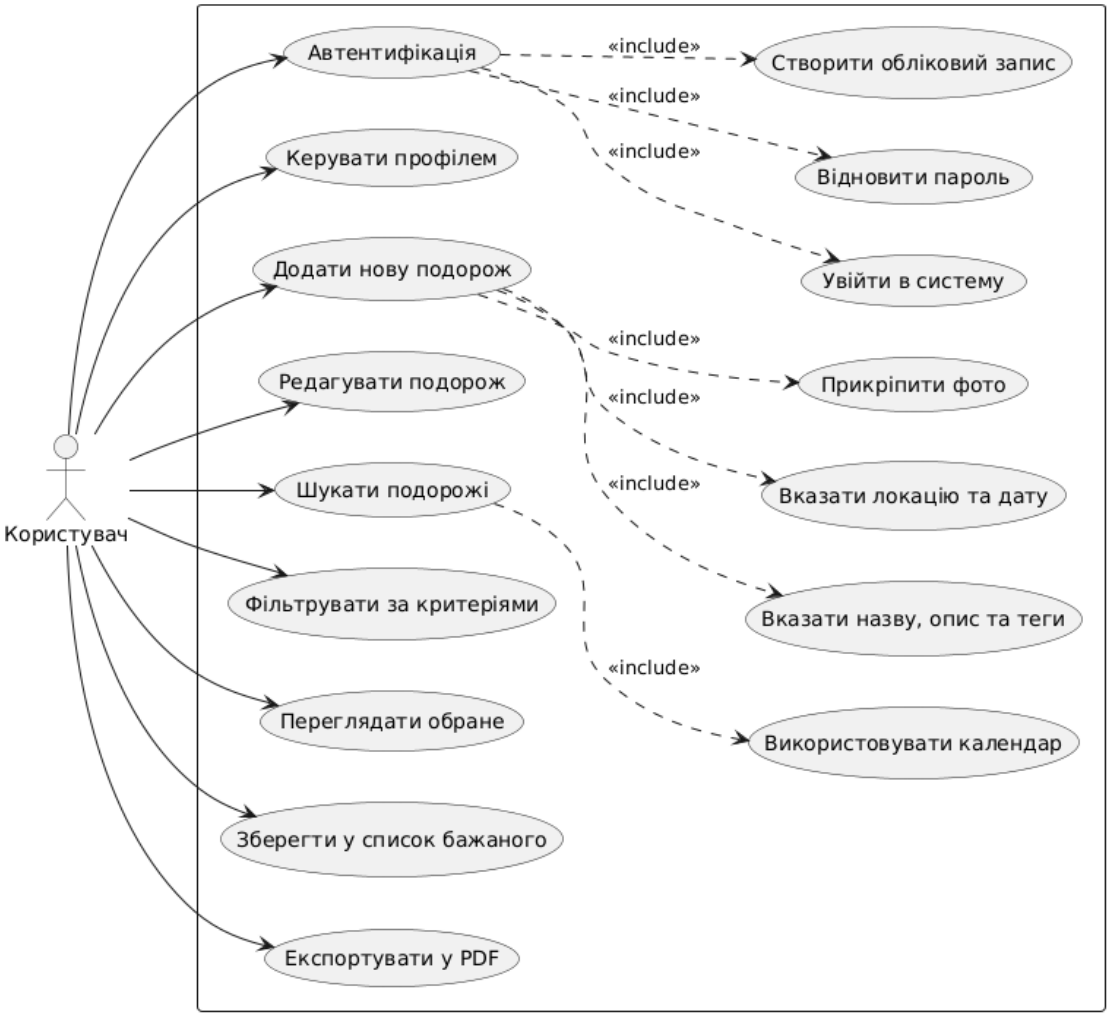


Рисунок 1.4 – Діаграма варіантів використання для вебзастосунку планування та документування туристичних подорожей

Діаграма варіантів використання демонструє функціональність вебзастосунку планування та документування туристичних подорожей, показуючи взаємодію актора (користувача) та варіантів використання (функціоналу системи).

Користувач є основним суб'єктом, що взаємодію із вебзастосунком. Він може виконувати різні дії, починаючи від автентифікації та закінчуючи додаванням або редагування записів про подорожі тощо.

Щоб мати доступ до системи, користувачу потрібно спочатку пройти етап автентифікації, що включає:

– створення облікового запису: реєстрація нового користувача із введенням всіх необхідних даних (ім'я, електронна пошта, пароль);

– вхід: введення облікових даних, що були використані при реєстрації.

Після цього користувач отримує доступ до особистого профілю та може вести історію подорожей.

Система дає змогу маніпулювати із записами про подорожі, а саме:

– додання подорожі: створення допису про подорож;

– редагування подорожі: можливість змінювати інформацію про записи з поїздками, які були створені раніше;

– видалення подорожі: видалення записів, що є вже неактуальними для користувача тощо.

Після додання подорожей, конкретні записи можна зробити обраними, щоб потім можна було легко їх знайти. Далі можна використати можливості пошуку та сортування, а саме:

– пошук подорожей: здійснення пошуку записів за назвою або використовуючи конкретну дату або діапазон дат в інтерактивному календарі;

– сортування: впорядкування туристичних подорожей за критеріями.

Додатково, даний вебзастосунок передбачає функцію збереження різних місць у списку бажаного. Це дозволить людям планувати майбутні поїздки або ж просто зберігати цікаві локації.

Таким чином, завдяки використанню діаграми варіантів використання можна отримати розуміння того, як працює система, що розроблюється. Вона дозволяє структурувати етап розробки та проєктування, виділивши основні аспекти роботи системи.

1.4 Висновки дослідження предметної області та постановки задачі

У процесі аналізу предметної області, її структурних та функціональних особливостей було описано що таке вебсайт, його основні типи та призначення. Вебсайти виконують найрізноманітніші завдання: від надання інформації стосовно певної теми до надання послуг. Аналіз існуючого програмно-технічного забезпечення продемонстрував, що сервіси з тематикою документування подорожей мають хороший функціонал для збереження та організації записів з подорожами зі своїми перевагами та недоліками.

Виявлені основні проблеми, що потребують рішення в контексті розробки вебзастосунку для планування та документування туристичних подорожей, є наступними:

– недостатня підтримка пошуку або фільтрації серед великої кількості записів;

– відсутня можливість управління списком бажаного;

– проблеми із безпекою та збереженням даних користувача.

На основі проведеного аналізу визначено основні вимоги для розроблюваного вебзастосунку:

– створити зручний та зрозумілий інтерфейс;

– розробити систему авторизації та реєстрації, використовуючи методи шифрування даних;

– додати можливість прикріпляти фото до записів із подорожами.

– розробити механізми пошуку та фільтрації;

– додати список бажаного для планування майбутніх подорожей;

2. ПРОЄКТУВАННЯ ВЕБЗАСТОСУНКУ

2.1 Аналіз та вибір архітектури вебзастосунку

Даний підрозділ є важливим для розробки програмного забезпечення, адже визначення архітектури та подальшої структури розроблюваної системи – це те, з чим часто мають справу розробники, особливо коли шукають архітектурний підхід, що спрощує виконання поставлених завдань. Ці підходи або ж патерни є набором готових правил та функціональних можливостей, які допомагають вирішити питання із загальною структурою системи, взаємозв'язком між її компонентами, а також такими аспектами, як продуктивність, безпека та інші [15].

Ключовими аспектами архітектури програмного забезпечення є:

– безпека: використання методів захисту для збереження доступу до необхідної інформації;

– продуктивність: забезпечення швидкодії системи та ефективного використання її ресурсів;

– компонентність та декомпозиція: розбиття системи на дрібніші компоненти, які незалежно один від одного можна розробляти та тестувати.

Шаблонів архітектури програмного забезпечення існує багато, але розглянуто буде найбільш популярні:

– багатошарова архітектура: характеризується підходом до проєктування програмного забезпечення, який ділить систему на рівні, які в свою чергу відповідають за певний функціонал. Це дозволяє зробити систему більш пристосованою до різних умов та зручною для підтримки. Зазвичай, даний вид архітектури включає такі рівні як: презентаційний, прикладний, рівень бізнес-логіки, та доступу до даних. Кожен з них працює незалежно та відповідає за окрему частину функціоналу програмного забезпечення. Це дозволяє взаємодіями з окремими частинами системи без впливу на інші. Однією з головних переваг даної архітектури є масштабованість, адже кожен рівень може

в певний час розширитись окремо від інших рівнів відповідно до потреб системи. Також дозволяється використовувати різні інструменти та технології на різних рівнях, не впливаючи на загальну функціональність. Через те, що є можливість розробникам паралельно працювати над окремими частинами системи, це значною мірою пришвидшує процес розробки. Серед недоліків можна виділити велику кількість рівнів, що, в свою чергу, інколи може створити проблеми або затримки під час передачі або отримання даних. Багатошарова архітектура найкраще підходить для складних рішень, де висока продуктивність є пріоритетом. Прикладами є системи управління взаємовідносинами з клієнтами (CRM), банківські системи, електронна комерція тощо;

– архітектура Модель-Вигляд-Контролер (MVC): базується на поділі системи на три взаємопов’язані компоненти: модель, вигляд та контролер. Це в першу чергу дає змогу організувати код програми, що в подальшому сприятиме зручності у підтримці програмного забезпечення. Модель є ключовим елементом та відповідає за логіку системи та керування даними. Вона взаємодіє із базою даних та обробляє різноманітну інформацію для нормального функціонування застосунку. Представлення відповідає за відображення даних та забезпечує взаємодію між користувачем та системою. Отримуючи дані від моделі, представлення відображає цю інформацію в потрібному форматі на екрані користувача. Контролер є посередником між цими двома компонентами. Отримуючи вхідні дані від користувача, він займається їх обробкою та надсилає дані моделі. Також він за необхідності оновлює представлення, щоб на екрані користувача завжди була актуальна інформація. Перевагою цієї архітектури є чіткий розподіл обов’язків між компонентами, що в результаті спрощує роботу під час різних етапів життєвого циклу програмного забезпечення. Серед недоліків є складність розуміння взаємодії між компонентами. У більшості випадків дана архітектура використовується в електронній комерції та настільних програмах [16];

– архітектура клієнт-сервер: передбачає розподіл системи на дві основні частини (клієнт – робить запити, а сервер – обробляє запити та надає на основі цього інформацію). Загальний процес роботи даного типу архітектури полягає в тому, що коли користувач (клієнт) надсилає запит на сервер, той в свою чергу повинен опрацювати запит та виконати певні дії, наприклад, звернутись до бази даних і в результаті повернути відповідь. Такий процес сприяє ефективній комунікації між пристроєм користувача та сервером. Однією з основних переваг такої архітектури є спрощене адміністрування та підтримка системи, адже загальна логіка та дані системи зосереджені на серверній частині, що підвищує рівень безпеки. Масштабованість також є важливим аспектом, оскільки можна додати додаткові сервери або оптимізувати ресурси у випадку зростання навантаження. Проте серйозним недоліком є те, що велика кількість запитів може негативно вплинути на сервер, що призведе до збоїв та уповільнення роботи. Також серед недоліків варто зазначити, що стабільність мережевого підключення впливає на надійність системи. Цей тип архітектури зазвичай використовується у різних вебзастосунках, системах електронної пошти, платформах для навчання та службах для зберігання файлів [17];

– архітектура мікросервісів: характеризується підходом до розробки програмного забезпечення, який передбачає процес розбиття системи на незалежні сервіси, кожен з яких призначений для виконання окремої функції. Це дозволяє уникнути певних обмежень, які можуть бути в інших архітектурних підходах та зробити систему більш гнучкою. Головною перевагою цього підходу є незалежність мікросервісів, оскільки кожен з них може бути розроблений, протестований та розгорнутий окремо від інших, що робить процес розробки більш ефективним. Мікросервіси взаємодіють між собою через API (Application Programming Interface) для виконання запитів, отриманих від користувача. Це дозволяє легко вирішувати, на перший погляд, складні завдання. Серед переваг можна виділити швидке впровадження необхідних функцій та масштабування окремих сервісів залежно від навантаження.

Водночас недоліками є високі вимоги до взаємодії з мережею та складність керування великою кількістю цих сервісів. Архітектура мікросервісів найчастіше використовується для потокових сервісів, фінансових платформ та електронної комерції [18];

Використання архітектурних шаблонів є хорошою практикою для підвищення зручності та ефективності в розробці програмного забезпечення. Кожен з таких шаблонів має свої переваги та області застосування, де він найкраще підходитиме під потреби та критерії того чи іншого проєкту [19].

Для розроблюваного вебзастосунку найбільш прийнятною буде архітектура Модель-Вигляд-Контролер (MVC). Як вже згадувалось, дана архітектура забезпечує чітке розмежування між компонентами системи, що дозволяє створити гнучку та легко підтримувану систему. Також вона є дуже ефективною при керуванні станом вебзастосунку. Для того, щоб вебзастосунок працював якомога ефективніше, потрібно чітко розмежувати функціонал, який виконуватимуть компоненти MVC [20]:

– модель: керування виключно даними вебзастосунку та логікою;

– вигляд: відображення виключно даних на екрані користувача;

– контролер: обробка вхідних даних користувача та оновлення моделі.

Нижче показано діаграму архітектури вебзастосунку за шаблоном MVC (Рисунок 2.1):

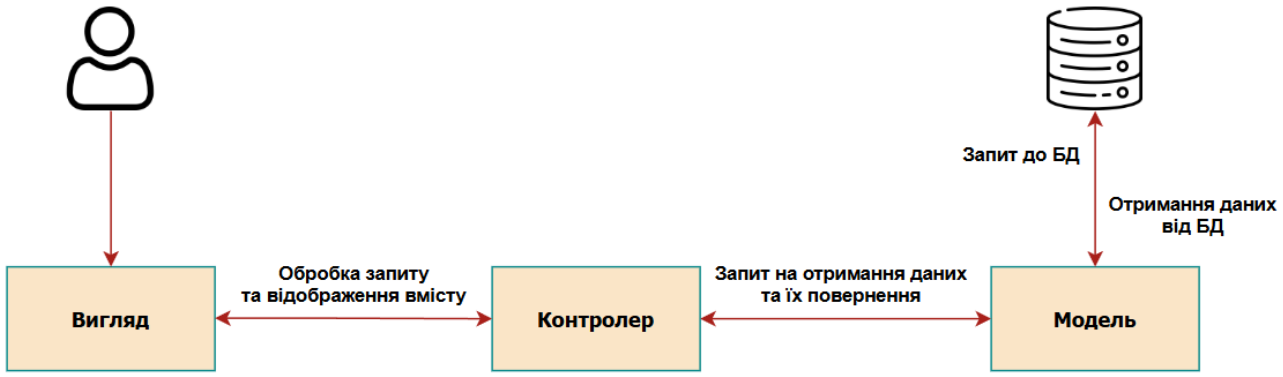


Рисунок 2.1 – Діаграма архітектури вебзастосунку за шаблоном MVC

Архітектура MVC продовжує добре пристосовуватись до сучасних вебзастосунків, особливо односторінкових. Раніше серверна частина могла керувати всім застосунком, але зараз функціонал чітко розмежовується і серверна частина відповідає за логіку та зв’язок з базою даних для збереження даних. Клієнтська частина в свою чергу відповідає за відображення контенту для користувача та обробку його дій.

Зазвичай, в контексті MVC, модель є ключовим компонентом, і відповідає за такі завдання, як: обробка логіки, зв’язок з базою даних тощо. Контролер виконує інше завдання, а саме обробка HTTP-запитів від користувачів, включаючи перевірку на наявність в них помилок та подальшу передачу даних для моделі. Для прикладу, в мові програмування JavaScript, завдяки асинхронному підходу, можна одночасно обробляти велику кількість запитів, дозволяючи іншим завданням виконуватись без жодних проблем. Щодо представлення, то в сучасних вебсистемах часто створюються динамічні інтерфейси зі швидкою відповіддю на запити користувача. При цьому кожен елемент інтерфейсу має свою роль та логіку роботи.

Архітектура MVC проводить такий цикл роботи вебзастосунку, який забезпечує високу швидкість та зручність використання. Але будь-який застосунок можна оптимізувати з точки зору особливостей різних інструментів розробки. Наприклад, можна використовувати індекси в базі даних або зменшити розмір програми за допомогою оптимізації зображень чи використанню відкладеного завантаження різних ресурсів.

Безпека також є дуже важливою складовою при розробці будь-якого сучасного вебзастосунку. Вона є важливою як для підвищення продуктивності, так і для загальної цілісності системи. Для забезпечення безпеки потрібно впроваджувати ефективні засоби автентифікації користувачів, використовувати протокол передачі даних HTTPS, бути обережним з терміном дії JWT токенів, щоб запобігти втраті даних, використовувати шифрування паролів та іншої важливої інформації тощо.

2.2 Проєктування структури даних та моделі бази даних

Логічна модель бази даних є сполучною ланкою між концептуальною та фізичною моделями. Вона чітко описує сутності, атрибути та зв'язки між ними без залежності до певної системи керування базами даних (СКБД). Головними компонентами логічної моделі є: сутності, атрибути, первинні ключі, зовнішні ключі та зв'язки між сутностями [21].

Сутності – це основні елементи системи, які мають певну інформацію. У контексті розроблюваного вебзастосунку із документуванням подорожей вони можуть бути представлені наступним чином:

– user (користувач): зберігає дані про користувачів, які зареєструвались на сайті;

– travelStory (подорож): містить інформацію про туристичні поїздки користувачів;

– location (локація): місце розташування, яке відвідав мандрівник;

– wishlist (список бажань): місця, яка користувач планує відвідати;

– tag (тег): категорії, за якими можна описати подорож.

Маючи потенційні сутності, можна зробити спрощену схему логічної моделі бази даних, яка демонструє базовий зв'язок між описаними сутностями (Рисунок 2.2):

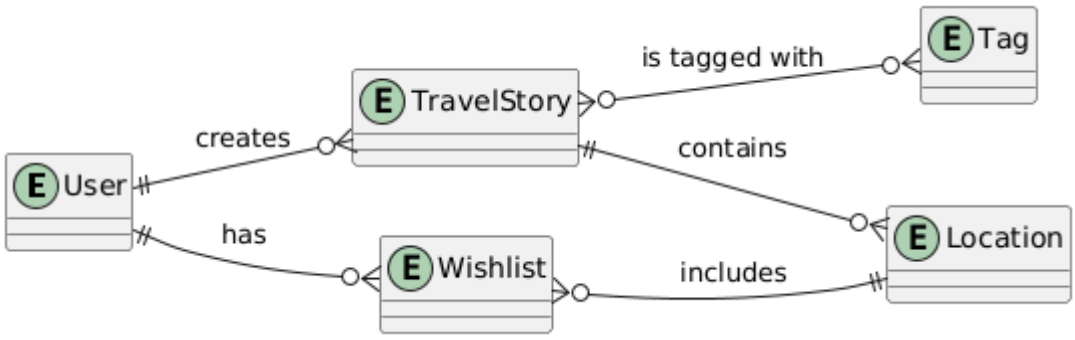


Рисунок 2.2 – Спрощена ER-діаграма

Звідси видно, що:

– користувач може створювати історії подорожей;

– історії подорожей містить теги та локації;

– користувач може користуватись списком бажань.

Атрибути – це поля, які складають характеристику сутності. Для прикладу сутність User може мати такі атрибути:

– user_id: ідентифікатор користувача;

– username: ім’я користувача;

– email: електронна пошта;

– password: пароль.

Сутність TravelStory може мати такі атрибути:

– travel_story_id: ідентифікатор подорожі;

– title: назва подорожі;

– story: опис подорожі;

– is_favourite: позначка, чи є подорож обраною;

– image_url: фото подорожі;

– user_id: ідентифікатор користувача;

– visited_date: дата подорожі.

Сутність Wishlist може мати такі атрибути:

– wishlist_id: ідентифікатор історії в списку бажаного;

– notes: опис місця подорожі;

– image_url: посилання на фото;

– rating: оцінка локації;

– user_id: ідентифікатор користувача;

– location_id: ідентифікатор локації.

Сутність Tag може мати такі атрибути:

– tag_id: ідентифікатор тегу;

– name: назва тегу.

Сутність Location може мати такі атрибути:

– location_id: ідентифікатор локації;

– name: назва локації;

– address: адресу локації;

– travel_story_id: ідентифікатор подорожі.

Нижче показано приклад зв’язку між сутностями User та TravelStory (Рисунок 2.3):

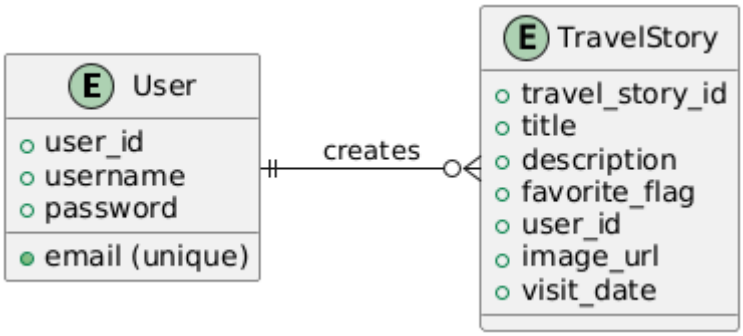


Рисунок 2.3 – Зв’язок між сутностями User та TravelStory

Первинний ключ – це унікальний ідентифікатор сутності, який відрізняє один об’єкт від іншого. В контексті даної моделі вони будуть наступними:

– user_id: унікальний ідентифікатор для кожного користувача в таблиці User;

– travel_story_id: унікальний ідентифікатор для кожного запису з історією про подорож в таблиці TravelStory;

– wishlist_id: унікальний ідентифікатор для кожного запису у списку бажаного в таблиці Wishlist;

– location_id: унікальний ідентифікатор для кожної локації;

– tag_id: унікальний ідентифікатор для кожного тегу (категорії).

Зовнішній ключ – це ключ, який поєднує сутності, встановлюючи чіткий взаємозв’язок між ними. Для прикладу:

– user_id (TravelStory) до user_id (User): становить зв’язок “один-до-багатьох”. Тобто один користувач може мати багато історій з подорожей і кожна історія в свою чергу належить лише одному користувачеві. Тут зовнішній

ключ user_id у таблиці TravelStory пов’язаний з атрибутом user_id у таблиці User;

– travel_story_id (Location) до travel_story_id (TravelStory): зв’язок “багато-до-одного”. Тобто багато локацій можуть бути пов'язані з одним записом про подорож;

– tag_id (TravelStory) до tag_id (Tag): зв’язок “багато-до-багатьох”. Тобто багато історій можуть бути пов'язані з багатьма різноманітними категоріями;

– user_id (Wishlist) до user_id (User): зв’язок “один-до-багатьох”. Тобто один користувач може додавати різні локації до свого списку бажаного;

– location_id (Wishlist) до location_id (Location): зв’язок “один-до-багатьох”. Тобто одна локація може бути в багатьох списках бажаного різних користувачів.

Всі описані вище ключі та взаємозв’язки слугують для забезпечення цілісності роботи системи та потрібні для правильної організації даних [22].

Нижче зображено ER-діаграму для розроблюваного вебзастосунку (Рисунок 2.4):

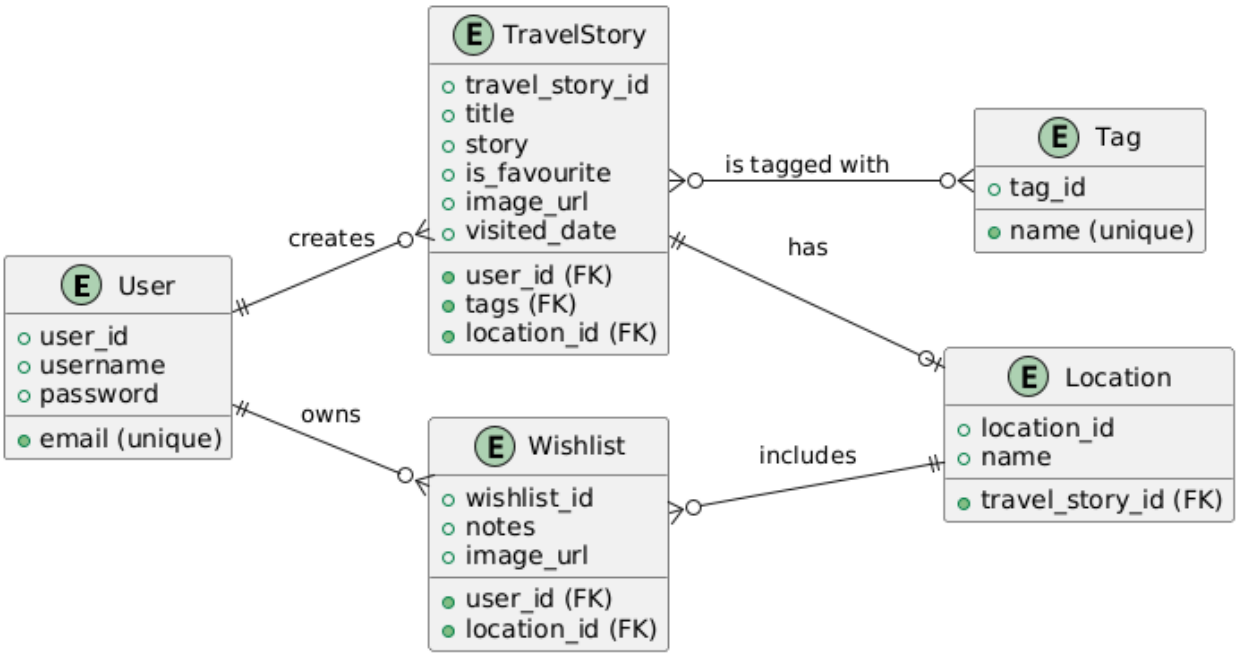


Рисунок 2.4 – ER-діаграма

2.3 Проєктування інтерфейсу користувача

У даному підрозділі потрібно описати процес проєктування інтерфейсу користувача для розроблюваного вебзастосунку. Дизайн користувацького інтерфейсу (UI) має досить велике значення для задоволення користувачів, зручності та їх подальше утримання. Сучасне програмне забезпечення повинне відповідати вимогам основним правилам дизайну, щоб створити інтуїтивно зрозумілий та привабливий інтерфейс. У процесі користування вебзастосунком, інтерфейс має бути простим та зручним у навігації [23].

Окрім цього, як згадувалось раніше, добре спроєктований інтерфейс має вплив на показники утримання користувачів, що дозволяє продукту бути кращим вибором серед конкурентів. Також це може позитивно вплинути на зниження витрат на розробку, адже при умові правильно спроєктованого інтерфейсу на початкових етапах, мінімізується кількість різноманітних проблем в майбутньому, що призведе до економії часу та зусиль команди дизайнерів та розробників [24].

Вебзастосунок для планування та документування подорожей повинен мати чітку ієрархічну структуру для забезпечення зрозумілої та зручної навігації. Система складається лише з одним рівнем доступу для користувачів – приватним, тобто для користування вебзастосунком потрібно зареєструватись, щоб мати доступ до функціоналу. Вебзастосунок складається з таких частин:

– сторінка для входу та реєстрації;

– головна сторінка з відображенням записів про подорожі;

– модальне вікно для додання, редагування та видалення історії;

– список бажаного;

– профіль користувача.

Щодо системи навігації, то вона має бути максимально зрозуміла, щоб користувач міг легко переходити між різними сторінками та складовими системи.

Основна навігація розміщена у верхній частині вебзастосунку (Хедер) та включає:

– логотип (при натисканні відбувається перехід на головну сторінку);

– пошук історій;

– список бажаного;

– профіль користувача;

– кнопка виходу.

Описавши кожну зі сторінок вебзастосунку, можна краще зрозуміти структуру розроблюваної системи. Сторінка входу (Рисунок 2.5) розроблена в мінімалістичному стилі та містить:

– поля для введення електронної пошти та паролю;

– кнопку для входу;

– кнопку із посиланням на сторінку зі створенням нового облікового запису;

– слоган вебзастосунку, що передає його загальну ідею та концепцію сервісу.

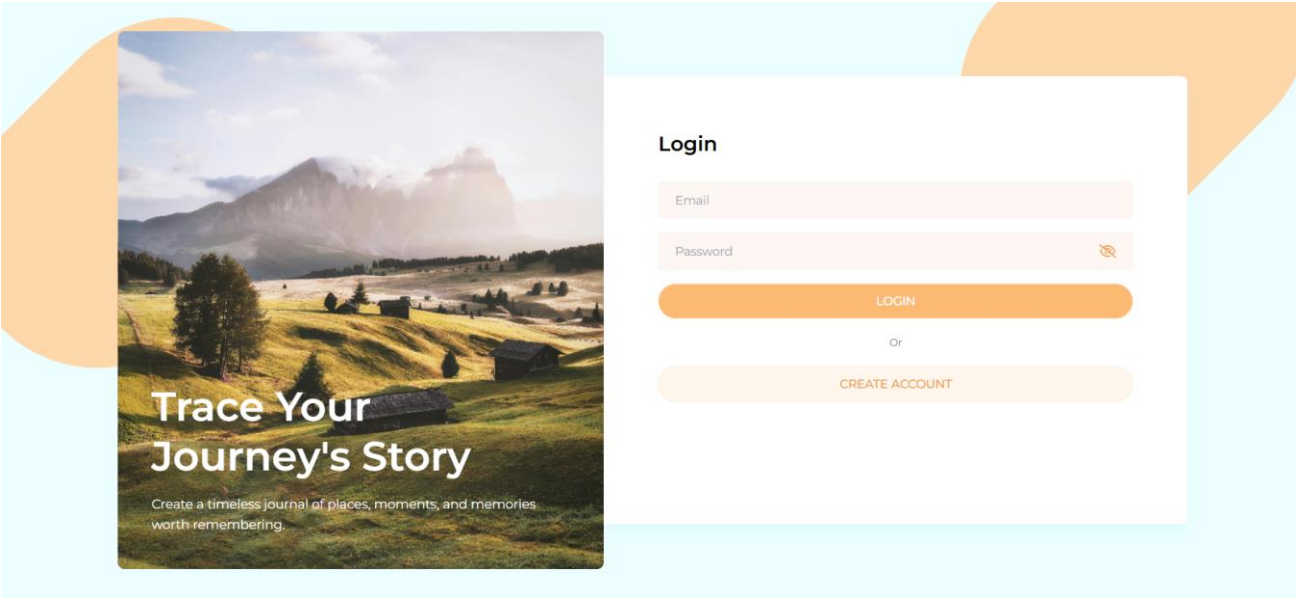


Рисунок 2.5 – Сторінка входу

Сторінка реєстрації (Рисунок 2.6) не сильно відрізняється від сторінки входу (Рисунок 2.5), окрім додання нового поля для введення імені користувача, а також зміненого альтернативного слогану.

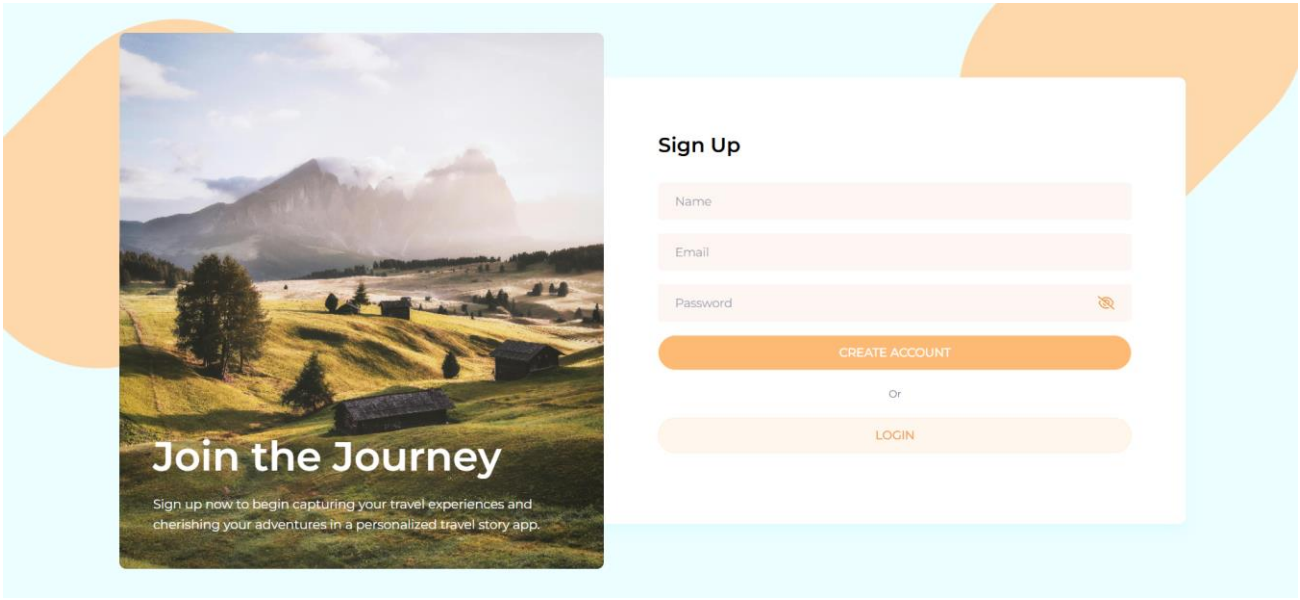


Рисунок 2.6 – Сторінка реєстрації

Головна сторінка (Рисунок 2.7) є центральною частиною вебзастосунку, де відображається список історій з подорожами користувачів. Вона розроблена таким чином, щоб можна було зручно та швидко отримати доступ до всього необхідного функціоналу вебзастосунку. Вона складається з наступних компонентів:

– верхня навігаційна панель з логотипом, пошуком, кнопкою для переходу на сторінку зі списком бажаного та ім’ям користувача з кнопкою-переходом в профіль або для виходу із системи;

– список подорожей у вигляді карток з короткою інформацією;

– поле для введення для того, щоб користувач міг швидко знайти потрібний допис;

– календар для фільтрації дописів за певною датою або періодом, що значно полегшує навігацію серед великої кількості записів;

– кнопка у вигляді символу “+” для створення нового запису про подорож, яка розташована у правому нижньому кутку.

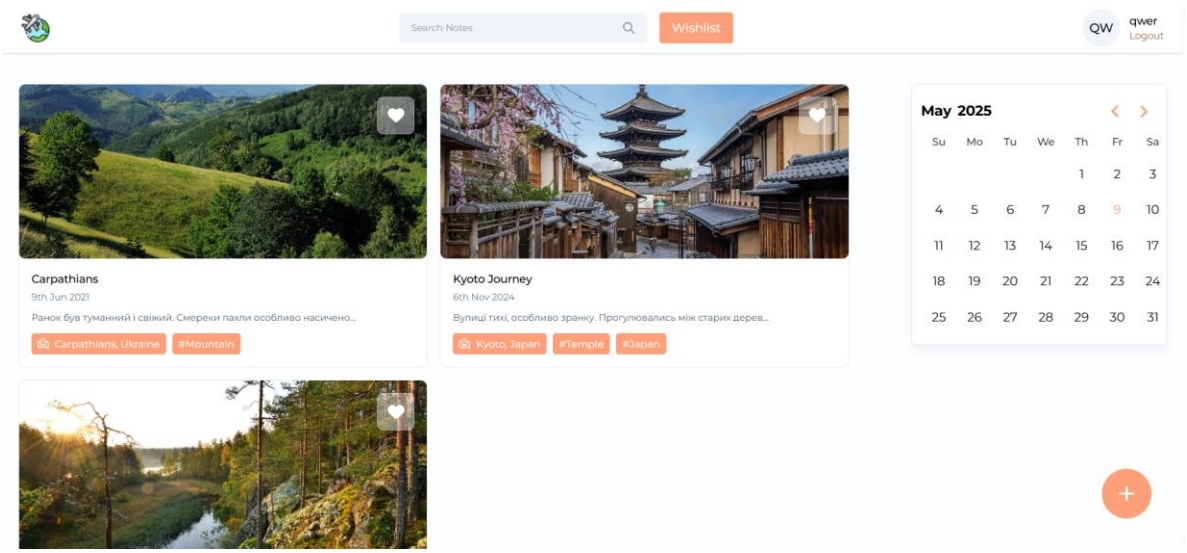


Рисунок 2.7 – Головна сторінка

Натиснувши на допис із туристичною поїздкою (Рисунок 2.8), з’явиться модальне вікно для більш детальної інформації. Тут містяться:

– кнопки для експорту в PDF формат, оновлення та видалення історії;

– назва історії, дата, місце, фото, опис та теги.

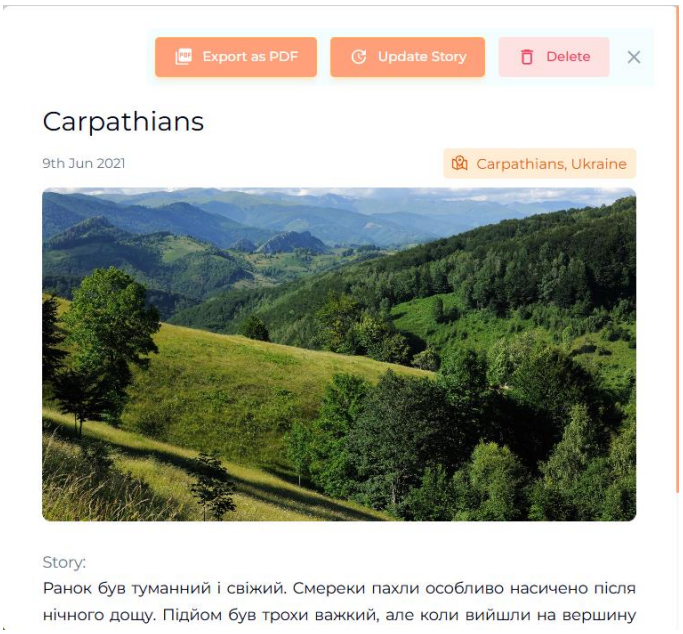


Рисунок 2.8 – Модальне вікно із детальним описом подорожі

Список бажаного (Рисунок 2.9) виглядає досить просто та складається з таких частин:

– заголовок;

– кнопка для додання нової локації;

– список відображення місць для відвідування із коротким описом про них.

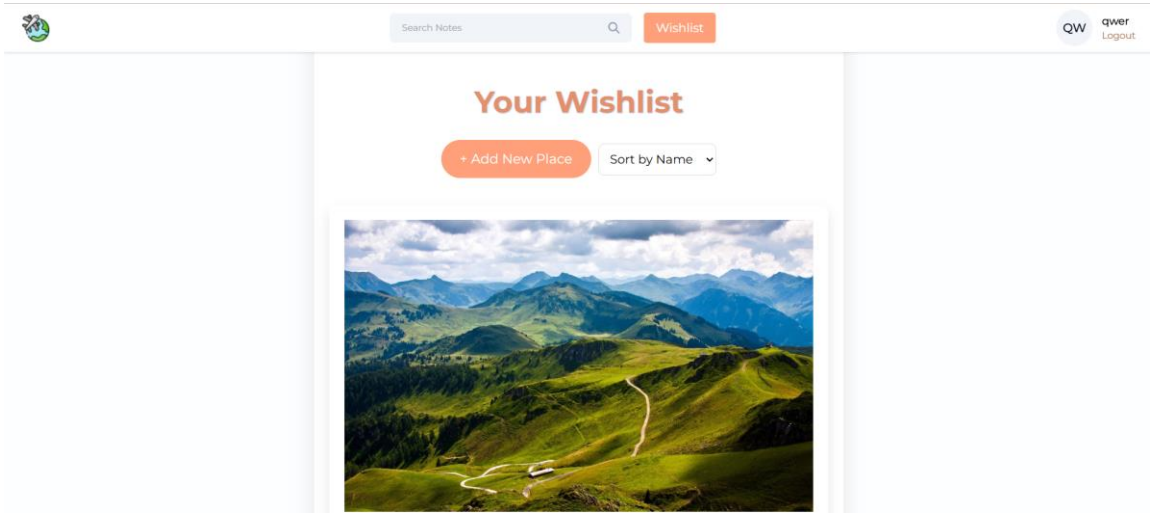


Рисунок 2.9 – Список бажаного

Профіль користувача (Рисунок 2.10) є останньою сторінкою вебзастосунку та призначений для відображення даних про користувача:

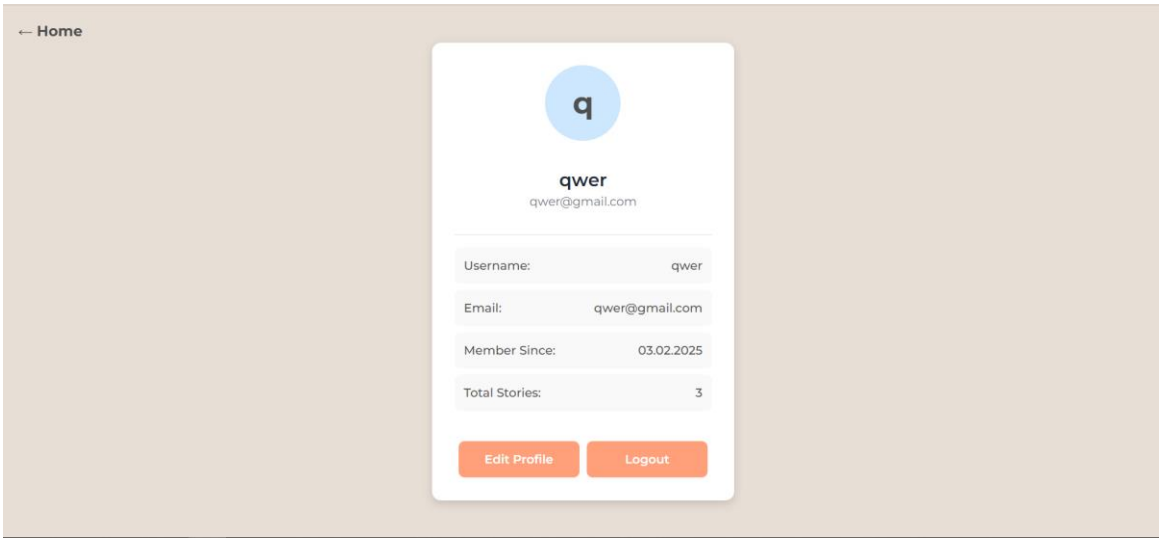


Рисунок 2.10 – Профіль користувача

Описавши інтерфейс вебзастосунку, можна зробити висновок, що все виконано у мінімалістичному дизайні з чітким структуруванням інформації. Основними кольорами є:

– білий: для фону різних сторінок та компонентів;

– чорний: для тексту, включаючи відображення дати ти чисел календаря на головній сторінці;

– помаранчевий та персиковий: відображення кнопок, посилань та фону профілю користувача.

2.4 Аналіз та вибір технологій і методів реалізації вебзастосунку

Для реалізації вебзастосунку для планування та документування туристичних подорожей вирішено обрати стек MERN, який є сучасним набором різних технологій, що дозволяють створювати вебзастосунки для різних людських потреб, на основі мови програмування JavaScript. Сам стек складається із 4 технологій, а саме: MongoDB, Express.js, React та Node.js. Перевагами даного стеку є: [25]

– єдина мова програмування: використання мови JavaScript для обох частин вебзастосунку (клієнтської та серверної) спрощує процес розробки та дає змогу розробникам мати справу одразу з усіма рівнями системи;

– висока продуктивність: вебзастосунок працює швидко завдяки асинхронній структурі Node.js та віртуальному DOM React. Така зв’язка робить відображення інтерфейсів швидким, порівняно з іншими підходами;

– гнучкість: реляційна база даних MongoDB підтримує гнучку структуру даних та дозволяє швидку обробку досить великих обсягів даних;

– розвинена екосистема: можливість додати велику кількість бібліотек та інструментів, які значно покращують процес розробки.

Найчастіше даний стек використовується в таких цілях: [26]

– соціальні мережі: обробка інформації в режимі реального часу дозволяє швидко оновлювати інтерфейс користувача, що підвищує рівень взаємодії із користувачем;

– електронна комерція: швидка обробка запитів є перевагою у сфері інтернет-магазинів;

– сервіс доставки їжі: технології даного стеку дозволяють створити сучасні платформи з доставкою їжі із забезпеченням всіх необхідних функцій та можливістю інтегрування карт розташування ресторанів тощо;

– ведення блогів: можливість створити інтерактивні та зручні вебзастосунки, які дають змогу людям створити власний блог та поширювати в ньому інформацію на різні теми.

Далі варто розглянути кожну з технологій окремо, виділити особливості, переваги та недоліки.

MongoDB – це досить популярна нереляційна база даних, що зберігає дані у форматі JSON, роблячи її зручною для розробників. Вона добре підходить для застосунків під різні платформи, які пов’язані із доступом до великих обсягів даних та потребують масштабованості. Однією з основних переваг цієї бази даних є її спосіб зберігання даних. Порівнюючи із традиційними реляційними базами даних, які для зберігання даних використовують таблиці, MongoDB використовує колекції з документами, кожен з яких має власну структуру. При розробці проєктів це дозволяє швидко змінити модель даних, що сприяє економії часу, який йде на розробку.

Серед функцій цієї бази даних, є індексування, яке робить пошук більш швидким, шардинг – для розбиття великих наборів даних на маленькі [27], конвеєр агрегації для обробки більш складних запитів тощо. Для виконання запитів використовується мова MongoDB Query. MongoDB пропонує декілька версій для використання включаючи звичайну безплатну версію для невеликих проєктів, покращену версію для комерційних проєктів, включаючи посилену безпеку та хмарний сервіс.

Незважаючи на недоліки у вигляді обмеженої підтримки транзакцій порівняно з іншими базами даних та проблем з виконанням більш складних запитів, MongoDB добре підходить для застосунків, де важливий швидкий доступ до даних [28].

Express.js – це популярний фреймворк для Node.js, який використовується для спрощення розробки серверної частини вебзастосунків. Він переважно використовується при розробці REST API та односторінкових застосунків. Express.js надає базовий функціонал для обробки запитів та відповідей, та керування маршрутизацією, яка дозволяє визначити метод відповіді застосунку на запити, що надходять від користувачів. Також він дозволяє використання проміжних функцій (middleware), які виконуються в період між відправленням даних та їх отриманням.

Даний фреймворк має широку спільноту розробників, підтримує інтеграцію з багатьма базами даних та має підтримку великої кількості додаткових бібліотек для додання нового функціоналу. Проте через обмежений набір вбудованих функцій, розробники іноді вимушені звертатись до сторонніх бібліотек для реалізації того чи іншого функціоналу [29].

Node.js – це середовище виконання вебзастосунків, які написані мовою програмування JavaScript. За допомогою нього розробники мають змогу створювати серверні та мережеві застосунки, використовуючи лише одну мову програмування як для клієнтської частини, так і для серверної [30]. Через свою неблокуючу та подієво-орієнтовану архітектуру, Node.js дозволяє зручно та швидко обробляти велику кількість паралельних запитів, не перенавантажуючи систему. Тобто через відсутність блокування потоків, забезпечується ефективне використання ресурсів та відсутні будь-які затримки, що є однозначною перевагою Node.js.

Також перевагою є широка екосистема бібліотек та пакетів, які доступні через менеджера пакетів npm, який дозволяє легко керувати різними бібліотеками. Цикл подій є ще однією важливою темою цієї технології, адже з його допомогою можна обробляти асинхронні запити без зайвих проблем у

вигляді створення додаткових потоків. Це особливо актуально при розробці відеосервісів. Окрім цього дуже часто Node.js використовується у розробці API, серверних застосунків, застосунків реального часу тощо [31].

React – це відрита бібліотека мови JavaScript, призначена для розробки динамічних користувацьких інтерфейсів. Використовуючи компонентний підхід, React дозволяє розбивати інтерфейс на менші частини, роблячи код розробника більш організованим та структурованим.

Ще однією з особливостей даної бібліотеки є концепція Virtual DOM, яка абстрактно описує звичайний DOM для оптимізації оновлення інтерфейсу шляхом фокусування лише на необхідних елементах, а не на всій сторінці. Це значно прискорює роботу вебзастосунків, оскільки зменшує навантаження на браузер. [32]

Використовуючи зовнішні бібліотеки або вбудовані функції React, такі як “Хуки”, можна зручно керувати станом вебзастосунку та динамічно обробляти різні дані [33]. React має багату екосистему, включаючи велику кількість бібліотек для використання у проєктах, а компонентний підхід допомагає зручно підтримувати код на будь-якому етапі застосунку. React має велику спільноту розробників та є основним вибором багатьох компаній при розробці програмного забезпечення [34].

Окрім звичайного стека для розробки потрібно також обрати сервіс для зручного зберігання фото історій подорожей користувачів. Для таких цілей добре підійде Cloudinary. Це хмарний сервіс для керування медіаконтентом. Серед основного функціоналу даний сервіс включає завантаження, зберігання та обробку медіа. Також він має підтримку автоматичної оптимізації для зменшення розміру файлів, не втрачаючи якість. Cloudinary добре інтегрується з різними інструментами для розробки, в тому числі зі стеком MERN. Використовуючи API сервісу, можна мати весь необхідний доступ при роботі з фото та відео файлами.

Також маючи підтримку CDN, можна швидко надсилати та отримувати велику кількість медіафайлів.

Для автентифікації та авторизації користувачів буде використовуватись JSON Web Token (JWT), який є безпечним способом передачі даних у вигляді об’єкта JSON. Він складається з трьох частин [35]:

– заголовок: містить тип токена та метод підпису (алгоритм), який забезпечує цілісність токена;

– навантаження: включає основну інформацію про користувача та згенерований токен;

– підпис: перевірка справжності токена. Якщо інформація у токені буде змінена, тоді підпис стане недійсним.

Незважаючи на недоліки використання JWT у вигляді великого розміру токенів або можливості втрати токена, якщо він не зберігається на сервері, цей спосіб є хорошим рішенням, забезпечуючи зручність використання.

Нижче показано загальну діаграму архітектури вебзастосунку (Рисунок 2.11):

Рисунок 2.11 – Загальна діаграма архітектури вебзастосунку

З діаграми видно, що стек MERN забезпечує обмін інформацією між серверною та клієнтською частинами. React взаємодіє із сервером та відображає інтерфейс користувача, Node.js та Express.js займаються обробкою запитів та

взаємодіють із MongoDB. Cloudinary зберігає зображення, а JWT забезпечує надійну автентифікацію користувачів.

2.5 Висновки проєктування архітектури та структури програмного забезпечення

У процесі проєктування архітектури та структури системи було розглянуто особливості розробки програмного забезпечення, включаючи підходи та шаблони, які спрощують розробку та визначають структуру розроблюваної системи. Для даного вебзастосунку було обрано архітектуру MVC, яка чітко розмежовує функціонал вебзастосунку між трьома ключовими компонентами: моделлю, контролером та представленням. Це робить процес розробки простішим, оскільки все добре структуровано та сприяє ефективній підтримці системи в майбутньому.

При проєктуванні логічної моделі бази даних було виділено основні сутності, атрибути та зв’язки між ними для забезпечення коректної обробки та збереження даних у системі.

У випадку з проєктуванням інтерфейсу користувача, було зосереджено увагу на сторінках вебзастосунку, описавши що вони містять та для чого призначені. Адже чітке структурування та зрозуміла навігація покращують користувацький досвід користувача.

Врешті-решт було обрано технології для розробки вебзастосунку. Стеком для впровадження всього необхідного функціоналу став MERN, який складається з таких технологій, як: MongoDB, Express.js, Node.js та React. Дана сукупність технологій забезпечує хорошу ефективність та гнучкість використання. Для зберігання фото подорожей було обрано сервіс Cloudinary, а для автентифікації користувачів використовуватиметься JWT.

3. ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ВЕБЗАСТОСУНКУ

3.1 Розроблення бази даних

У даному підрозділі описано процес реалізації бази даних для вебзастосунку, який призначений для документування та планування туристичних подорожей. На основі раніше розробленої логічної моделі (підпункт 2.2), побудовано фізичну модель бази даних із використанням MongoDB. Для ефективної взаємодії з базою даних було використано бібліотеку Mongoose, яка дозволяє будувати схеми, реалізовувати валідацію, а також створювати зв’язки між сутностями.

Усі основні сутності вебзастосунку (User, TravelStory, Location, Wishlist, Tag) реалізовані у вигляді Mongoose-схем, тобто окремих колекцій. Усі вони зберігаються в окремих файлах, що відповідає належній структурі та принципам модульності, а також полегшує підтримку та масштабування вебзастосунку. Наприклад, модель історії подорожей (TravelStory), яка зберігає інформацію про туристичні подорожі, реалізована наступним чином:

```
      const travelStorySchema = new mongoose.Schema({
        title: { type: String, required: true },
        story: { type: String, required: true },
        isFavourite: { type: Boolean, default: false },
        userId: { type: mongoose.Schema.Types.ObjectId, ref: "User", required:
true },
        imageUrl: { type: String },
        visitedDate: { type: Date, required: true },
        tags: [{ type: mongoose.Schema.Types.ObjectId, ref: "Tag", required:
true }],
        location: { type: mongoose.Schema.Types.ObjectId, ref: "Location" },
      }, { timestamps: true });
```

Зв’язки між сутностями реалізовані за допомогою типу ObjectId атрибутом ref для вказання зовнішнього посилання між різними колекціями. Це забезпечує цілісність даних та логіку взаємодії між даними. Прикладом такого зв’язку є поле “userId” з фрагмента коду вище, яке встановлює зв’язок між туристичною подорожжю та мандрівником. Подібний підхід застосовується також для реалізації взаємозв’язку між іншими колекціями вебзастосунку.

Також у контексті взаємозв’язку між колекціями було використано автоматичне отримання пов’язаних даних з різних колекцій під час виконання одного запиту. Це можна зробити за допомогою механізму populate, який надається бібліотекою Mongoose. Наприклад, під час отримання даних зі списку бажаного користувача, можна одночасно отримати інформацію про місце розташування, виконавши один запит. Це позбавляє необхідності робити лишні запити до бази даних, що в свою чергу підвищує продуктивність системи. Приклад використання наведено нижче, використовуючи фрагмент коду одного з методів вебзастосунку:

```
      try {
          const wishlist = await Wishlist.find({ userId
}).populate("locationId");
          res.status(200).json({ wishlist });
      }
```

Бібліотека Mongoose дозволяє реалізувати базову валідацію даних на рівні схем:

```
const userSchema = new mongoose.Schema({
  name: { type: String, required: true },
  email: { type: String, required: true, unique: true },
  password: { type: String, required: true, minlength: 6 },
});
```

MongoDB, як документо-орієнтована СКБД, не базується на явному створенні таблиць. Замість цього тут використовуються колекції, які створюються автоматично після збереження документа. А за допомогою своєї гнучкої структури даних, додання нової інформації не спричиняє ніяких проблем та конфліктів вже з існуючими даними.

Підключення до бази даних MongoDB виконується за допомогою окремого модуля, що дозволяє відокремити основний код вебзастосунку із логікою взаємодії з базою даних. Це забезпечує зручність та полегшує масштабування проєкту. Для збереження такої конфіденційної інформації, як URI підключення до бази даних, використовується спеціальна змінна

середовища MONGO_URI. Це відповідає принципам безпеки та дозволяє ефективно зберігати подібні дані за межами коду.

Для реалізації підключення використовується метод connect() з бібліотеки Mongoose [36], який використовує змінну MONGO_URI:

```
const connectDB = async () => {
  try {
    await mongoose.connect(process.env.MONGO_URI);
    console.log("MongoDB connected");
  } catch (error) {
    console.error("MongoDB connection error:", error);
    process.exit(1);
  }
};
```

У випадку помилки з’єднання, з’явиться відповідне повідомлення і процес буде зупинено. Нижче описано який функціонал реалізовується засобами MongoDB, а який – за допомогою мови програмування JavaScript. (Таблиця 3.1)

Таблиця 3.1 – Розподіл функціональних обов’язків між технологіями вебзастосунку

Завдання	Інструмент для реалізації
Зберігання даних	MongoDB
Опис структури даних	Mongoose
Валідація	Mongoose
Створення, оновлення та видалення даних	JavaScript (Node.js)
Авторизація та автентифікація	JWT та Node.js
Захист маршрутів	Node.js (Middleware)

MongoDB відповідає за зберігання даних та базові операції з ними. Mongoose надає об’єктний підхід для роботи з базою даних і реалізовує зв’язки між сутностями та валідацію. Вся інша програмна логіка виконується за допомогою JavaScript та Node.js. Такий поділ дозволяє чітко розділити відповідальності між базою даних та логікою вебзастосунку.

3.2 Розроблення програмних модулів

У даному підрозділі описано реалізацію програмних модулів вебзастосунку для документування та планування туристичних подорожей. Вебзастосунок складається з різних модулів, які взаємодіють між собою та забезпечують повноцінну функціональність системи. Кожен з модулів чітко виконує своє завдання, а разом, вони забезпечують зручну, ефективну та безпечну взаємодію користувача із вебзастосунком.

Основна взаємодія між клієнтською та серверною частинами вебзастосунку здійснюється через REST API, яке реалізоване на основі Node.js за допомогою фреймворку Express. Усі дані зберігаються в базі даних MongoDB, що дозволяє ефективно працювати з гнучкою структурою даних.

Одним з основних модулів є модуль авторизації та реєстрації користувачів. Він відповідає за автентифікацію користувачів, обробку запитів та реєстрацію та вхід, вихід із системи тощо. Даний модуль реалізовано за допомогою контролера authController з використанням таких технологій:

– JSON Web Token (JWT): інтернет-стандарт для безпечної передачі інформації між сервером та отримувачем (клієнтом). Використовується для того, щоб реалізувати безпечну автентифікацію користувачів;

– bcrypt: бібліотека для хешування паролів, яка забезпечує захист користувацьких даних;

– validator: бібліотека для валідації даних, зокрема перевірки правильності введення електронної пошти.

За реєстрацію відповідає метод signup, який перевіряє наявність заповнення всіх необхідних полів для реєстрації, довжину пароля, формат та унікальність електронної пошти, а також створює нового користувача в системі із хешованим паролем. Фрагмент даного методу наведено нижче:

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

```
    if (!username || !email || !password) {
      return res
        .status(400)
        .json({ error: true, message: "All fields are required" });
    }
    if (!validator.isEmail(email)) {
      return res
        .status(400)
        .json({ error: true, message: "Invalid email format" });
    }
    if (password.length < 6) {
      return res.status(400).json({
        error: true,
        message: "Password must be at least 6 characters long",
      });
    }
    const userAlreadyExists = await User.findOne({ email });
    if (userAlreadyExists) {
      return res
        .status(400)
        .json({ error: true, message: "User already exists" });
    }

    const hashedPassword = await bcrypt.hash(password, 10);
    const user = new User({
      username,
      email,
      password: hashedPassword,
    });
    await user.save();
    generateTokenAndSetCookie(res, user._id);
    res.status(201).json({
      success: true,
      message: "User created successfully",
      user: {
        ...user._doc,
        password: undefined,
      },
    });
  } catch (error) {
    res.status(500).json({ error: true, message: "Registration failed" });
  }
};
```

Іншими важливими методами даного модулю є:

– login (Вхід): перевірка користувацьких даних, автентифікація та генерація JWT токену;

– checkAuth (Перевіка автентифікації): перевірка на валідність токену та отримання даних користувача;

– checkUser (Перевірка, чи існує користувач): перевірка існування користувача за електронною поштою;

– deleteUser (Видалення користувача): видалення облікового запису користувача із бази даних;

– logout (Вихід): забезпечення виходу із систем та видалення токена з файлів cookie.

Захист маршрутів реалізований за допомогою так званої проміжної ланки (middleware), яка здійснює перевірку наявності та правильності JWT токену перед здійсненням запитів до різних ресурсів. Нижче показано діаграму послідовностей для процесу автентифікації користувача (Рисунок 3.1):

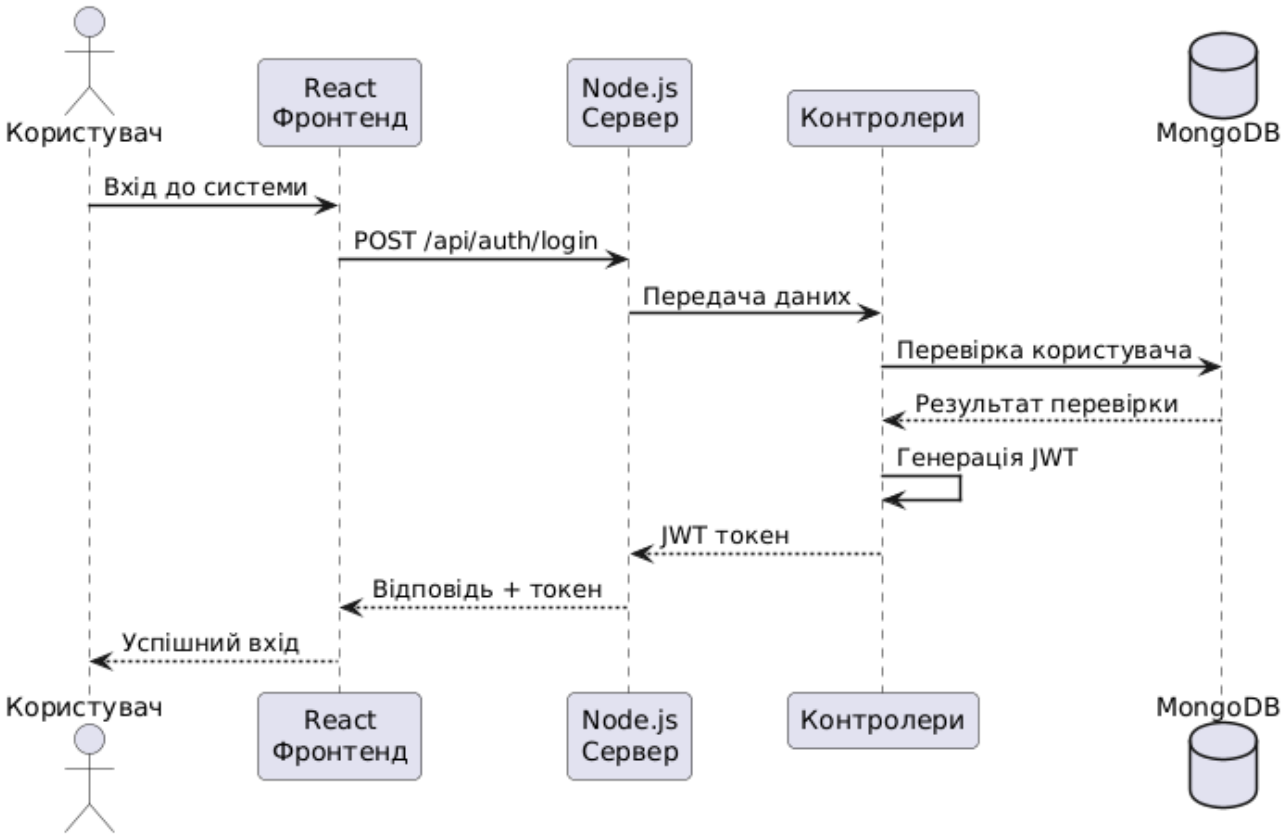


Рисунок 3.1 – Діаграма послідовностей для процесу автентифікації користувача

Модуль управління історіями подорожей реалізований за допомогою контролера travelStoryController, в якому зберігається весь необхідний функціонал для створення, редагування та видалення записів про подорож, фільтрація, пошук, оновлення статусу улюблених історій тощо. Даний модуль

також має інтеграцію із зовнішнім сервісом Cloudinary для ефективної обробки та збереження фото туристичних поїздок.

За отримання та відображення всіх дописів з подорожами відповідає метод getAllStories:

```
export const getAllStories = async (req, res) => {
  const { userId } = req.user;

  try {
    const travelStories = await TravelStory.find({ userId: userId })
      .sort({ isFavourite: -1 })
      .populate("tags")
      .populate("location");
    res.status(200).json({ success: true, stories: travelStories });
  } catch (error) {
    res.status(500).json({ success: false, message: error.message });
  }
};
```

Іншими методами є:

– addTravelStory (Додавання історії подорожі): створення нової історії із можливістю додання всієї додаткової інформації;

– uploadImage (Завантаження зображення): завантаження фото до сервісу Cloudinary;

– editTravelStory (Редагування історії подорожі): оновлення інформації про вже існуючу подорож;

– deleteTravelStory (Видалення історії подорожі): видалення існуючої подорожі з бази даних;

– deleteImage (Видалення зображення): видалення фото із сервісу Cloudinary;

– updateIsFavourite (Оновлення статусу “улюблене”): зміна статусу історії, роблячи її “улюбленою” та відповідно сортуючи її на початку сторінки з відображенням всіх подорожей, або навпаки – видаляти даний статус;

– searchTravelStories (Пошук історій подорожей): пошук історій за назвою;

– filterTravelStories (Фільтрація історій подорожей): фільтрація певних мандрівок за конкретною датою відвідування або періодом;

– getUserStoryCount (Отримання кількості історій користувача): отримання кількості дописів з подорожами користувача.

Нижче показано діаграму послідовностей для процесу додавання історії подорожі (Рисунок 3.2):

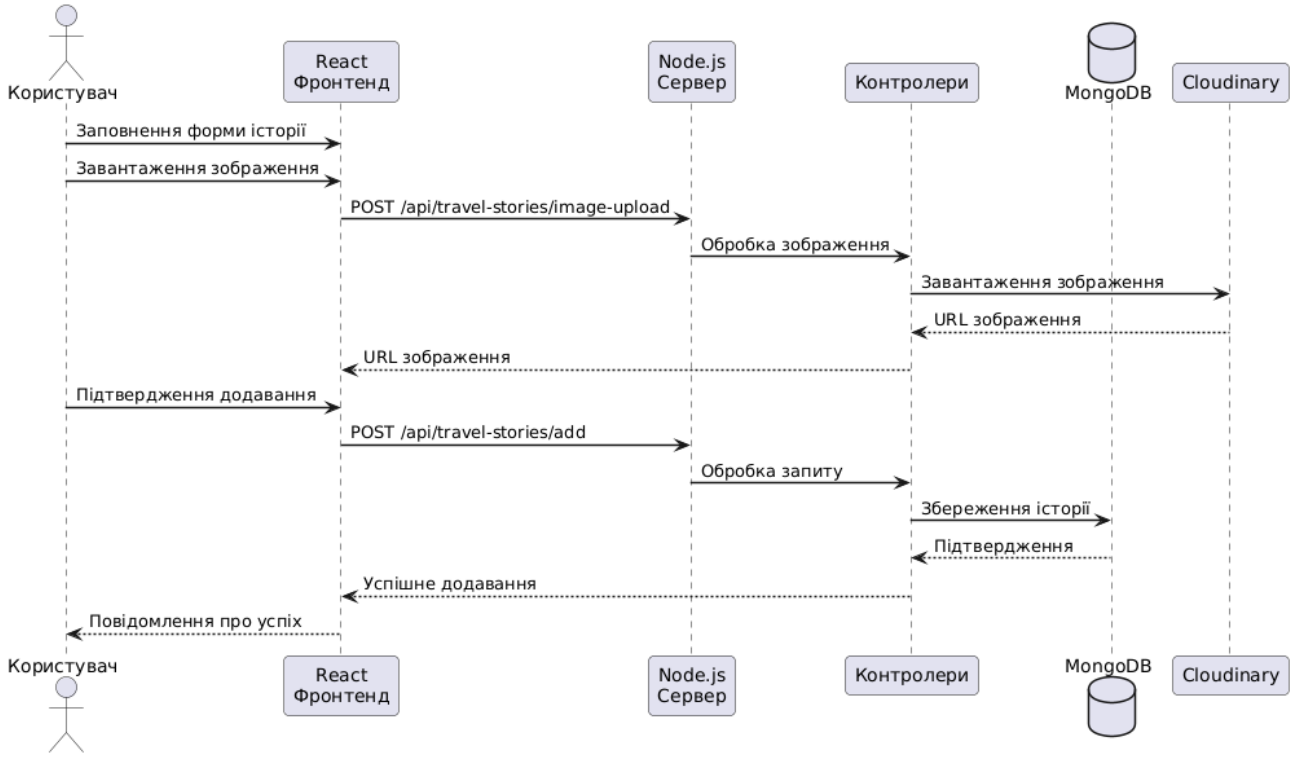


Рисунок 3.2 – Діаграма послідовностей для процесу додавання історії подорожі

Модуль списку бажаного реалізовується за допомогою контролера wishlistController, який містить необхідний функціонал для додання, редагування та видалення місць, які користувач хотів би відвідати в майбутньому. За отримання та відображення всіх бажаних місць для відвідування відповідає метод getWishlist:

```
export const getWishlist = async (req, res) => {
  const { userId } = req.user;

  try {
    const wishlist = await Wishlist.find({ userId
}).populate("locationId");
    res.status(200).json({ wishlist });
  } catch (error) {
    console.error("Error retrieving wishlist", error);
    res
```

```
            .status(500)
            .json({ error: true, message: "Couldn't retrieve wishlist" });
        }
    };
```

Іншими методами даного контролеру є:

– addWishlistItem (Додання елементу в список бажаного): створення нового місця подорожі в списку бажаного;

– uploadImage (Завантаження зображень): завантаження зображень до сервісу Cloudinary;

– deleteWishlistItem (Видалення елементу зі списку бажаного): повне видалення бажаного місця з бази даних;

– updateWishlistItem (Оновлення елементу в списку бажаного): редагування інформації про певне місце подорожі;

– deleteImage (Видалення зображення): видалення зображення з сервісу Cloudinary.

Клієнтський модуль для додання та редагування подорожей реалізований за допомогою компонента AddEditTravelStory. Даний React компонент надає форму для користувача, щоб взаємодіями з дописами. Основними можливостями є:

– введення назви та опису подорожі: надані текстові поля для введення основної інформації про подорож;

– вибір дати: використання спеціального компонента DatePicker для вибору дати;

– вибір бажаного зображення: завантаження відповідного фото, яке чітко описує туристичну поїздку;

– додання тегів: додання асоціації подорожі з відповідними тематичними мітками (наприклад “Сонячно”, “Гори” тощо);

– додання місця відвідування: вказання назви локації, яка була відвідана мандрівником.

Взаємодія з API сервера робиться з використанням бібліотеки Axios, яка призначена для виконання HTTP-запитів до сервера. Ці запити пов’язані з

додаванням, редагуванням та отриманням інформації та виконуються з використанням JWT токена.

Як вже було визначено раніше, для обробки фото використовується сервіс хмарний Cloudinary, інтеграція з яким надає функціонал зберігати фото в хорошій якості, обробляти та видаляти їх. Основними функціями даного сервісу є [37]:

– завантаження зображень: перетворення фото у формат Base64 та завантаження їх до сервісу;

– отримання URL зображень: отримання посилань на фото для їх подальшого відображення;

– видалення зображень: повне видалення обраного зображення зі сховища.

Взаємодія між модулями вебзастосунку працює за допомогою підходу REST API. В даному випадку клієнтська частина у вигляді React взаємодіє із серверною частиною у вигляді Node.js та Express за допомогою HTTP-запитів та передаючи дані у форматі JSON. Схема взаємодії між модулями працює наступним чином:

– модуль автентифікації: призначений для надання токена доступу після того, як користувач успішно входить в систему;

– модуль управління історіями подорожей: використовує токен для того, щоб перевірити, чи автентифікований користувач та подальшими операціями з даними;

– модуль списку бажаного: працює аналогічно модулю керування історіями подорожей;

– клієнтський модуль: використовує токен для того, щоб виконати запити та відобразити результати для користувача.

Нижче наведено діаграму компонентів вебзастосунку (Рисунок 3.3) із взаємодією між клієнтською та серверною частинами вебзастосунку:

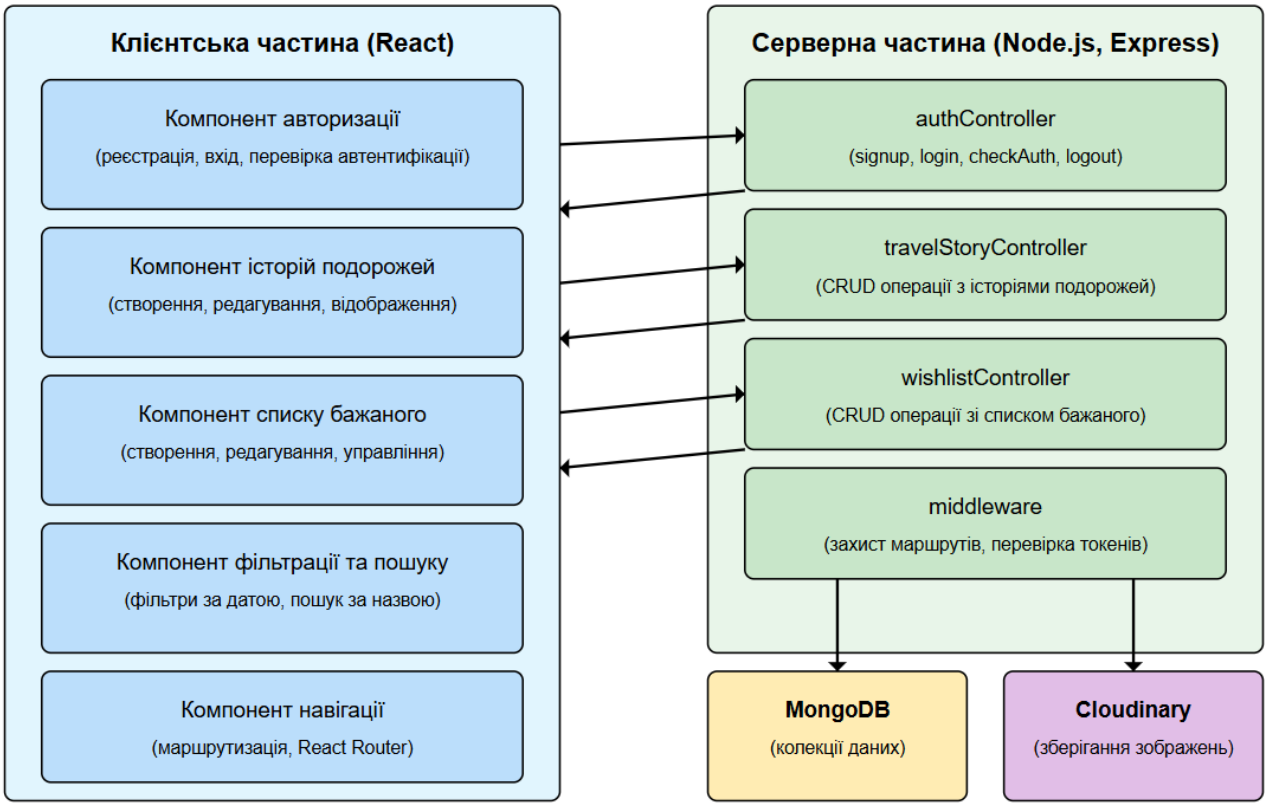


Рисунок 3.3 – Діаграма компонентів вебзастосунку

3.3 Керівництво користувача

Вебзастосунок для планування та документування туристичних подорожей дозволяє зберігати спогади з подорожей, пропонуючи зручний інтерфейс та функціонал. Для того, щоб почати користуватись вебзастосунком, спочатку як гість, потрібно зареєструватись, вказавши ім’я, пошту та пароль. Після цього вже зареєстрований користувач потрапляє на головну сторінку отримує доступ до всіх основних функцій:

– головна сторінка: вивід всіх створених подорожей, які за замовчуванням відсортовані за статусом “улюблені”;

– створення нової історії: натиснувши відповідну кнопку “+” в головному меню або “Add new place” у списку бажаного, користувач може заповнити форму з усією необхідною інформацією про подорож

– редагування та видалення історії: натиснувши на картку з історією можна відредагувати інформацію, змінивши існуючі дані, або повністю видалити запис подорожі;

– додання історії в улюблені: позначивши історію як “улюблену”, можна швидше отримувати до неї доступ;

– фільтрація та пошук: окрім стандартного сортування за улюбленими подорожами, можна сортувати записи, використовуючи календар. Також, використовуючи пошук, можна легко знаходити потрібні дописи;

– список бажаного: доступ до зберігання місць, які користувач хоче або планує відвідати.

3.4 Технічні характеристики вебзастосунку

Розроблений вебзастосунок для планування та документування туристичних подорожей побудований на сучасному MERN-стеку та забезпечує цілісність архітектури, високу гнучкість, масштабованість та зручність підтримки як серверної, так і клієнтської частин. Обраний підхід дозволяє легко розділити функціонал між ними, забезпечивши динамічну взаємодію без перешкод.

Серверна логіка реалізована за допомогою середовища Node.js, яке використовується для обробки запитів, та фреймворку Express.js, який дозволяє більш ефективно реалізовувати REST API. Особливістю серверної частини є обробка запитів від клієнта, даних, авторизації, взаємодії із базою даних тощо. Основними характеристиками серверної частини є:

– мова програмування: JavaScript (ES6+);

– середовище виконання: Node.js;

– фреймворк: Express.js;

– база даних: MongoDB;

– система автентифікації та авторизації: JSON Web Token (JWT);

– обробка зображень: Cloudinary;

– хешування паролів: bcrypt;

– інші бібліотеки та інструменти: dotenv, cors, validator, mongoose.

Уся логіка, пов’язана із цими інструментами та, яка належить до серверної частини, реалізована за допомогою окремих контролерів та маршрутів.

Клієнтську частину було реалізовано з використанням бібліотеки React, яка забезпечує використання компонентного підходу для створення інтерфейсів, та сучасних інструментів фронтенд-розробки. Основні характеристики:

– мова програмування: JavaScript (JSX, ES6+);

– бібліотека: React;

– система маршрутизації: React Router;

– менеджер стану: React Context API;

– клієнт HTTP: Axios;

– інші інструменти: Tailwind CSS, react-icons, react-tostify.

Для безперебійного функціонування вебзастосунку потрібні наступні умови:

– версія Node.js 18 або вище;

– налаштована локально база даних MongoDB або хмарно (використання MongoDB Atlas);

– наявність облікового запису в Cloudinary;

– підтримка сучасних браузерів;

– стабільне підключення до інтернету для взаємодії із сервером та базою даних.

3.5 Тестування вебзастосунку

Тестування будь-якого вебзастосунку або іншої системи є необхідною частиною життєвого циклу програмного забезпечення та відповідає за перевірку того, чи правильно функціонує система. Ефективне тестування допомагає

знайти помилки, щоб розроблюваний продукт працював правильно та відповідав поставленим вимогам та очікуванням кінцевої аудиторії. У випадку тестування React застосунків важливо гарантувати надійність компонентів та взаємозв’язку між ними. [38]

3.5.1 Вибір та обґрунтування методів тестування вебзастосунку

Для забезпечення комплексного тестування вебзастосунку було обрано методи, що охоплюють різні аспекти роботи [39]:

– функціональне тестування: перевірка різних сценаріїв використання основного функціоналу вебзастосунку (автентифікація, робота з подорожами тощо);

– інтеграційне тестування: перевірка взаємодії різних складових вебзастосунку, включаючи перевірку роботи клієнтської частини та серверної через API;

– наскрізне (E2E) тестування: перевірка взаємодії користувача із вебзастосунком шляхом імітації його дій при використання певного функціоналу, який надає система;

– тестування безпеки та продуктивності: перевірка захищеності вебзастосунку, імітуючи різні атаки та коректну роботу вебзастосунку при великому навантаженні;

– адаптивне тестування: перевірка відображення складових інтерфейсу на різних пристроях;

Для виконання тестування вебзастосунку було використано наступні інструменти:

– Postman: інструмент для тестування API, який дозволяє автоматизувати процес тестування. З його допомогою можна надсилати запити до сервера та перевіряти їхню відповідь;

– Cypress: фреймворк для автоматизованого тестування, який зазвичай використовується для наскрізного тестування вебзастосунків.

Описані вище інструменти було обрано через їхню високу ефективність та гнучкість при використанні.

3.5.2 Верифікація та валідація вебзастосунку

Для перевірки відповідності реалізацій компонентів вебзастосунку та правильне реагування на різні вхідні дані були проведені такі тести:

– реєстрація та авторизація: тестування перевірки при реєстрації та вході в систему, обробка помилок при введенні некоректних даних або реєстрації з існуючою електронною поштою;

– історії подорожей: перевірка додання, редагування та видалення історій із перевіркою формату даних;

– список бажаного: тестування додавання, редагування та видалення елементів списку бажаного. Також перевірявся діапазон рейтингу від 1 до 5;

– обробка некоректних даних: перевірка відповіді від застосунку на порожні поля, некоректний формат даних тощо;

– адаптивність: тестування відображення інтерфейсу при створенні історій та переході між сторінками на мобільних пристроях;

– продуктивність: перевірка часу завантаження вебзастосунку при завантаженні сторінки із 20 записами подорожей.

Використовуючи Cypress, було виконано автоматизоване тестування інтерфейсу [40]. Нижче наведено приклад тесту для перевірки додання нової історії подорожі через інтерфейс:

```
it("Додання історії через інтерфейс", () => {
  loginUser();
  cy.get("[data-testid='add-story-button']").click();
  cy.get("input[name='title']").type("Моя подорож до Карпат");
  cy.get("[data-testid='date-selector']").click();
  cy.get(".rdp-day").contains("15").click({ force: true });
  cy.get("input[type='file']").attachFile("test-image.jpg");
```

```
    cy.get("input[placeholder='Add Hashtag']").type("природа{enter}");
    cy.get("textarea").type("Це була незабутня подорож!");
    cy.get("input[placeholder='Add Location']").type("Карпати{enter}");
    cy.get("[data-testid='save-story-button']").click();
    cy.get(".Toastify__toast--success").should(
      "contain",
      "Story Added Successfully"
    );
    cy.get(".grid").should("contain", "Моя подорож до Карпат");
  });
```

Для кращої перевірки API також було проведено тестування запитів використовуючи Postman [41]. Наприклад, для додання нової історії, можна виконати наступний запит (Рисунок 3.4):

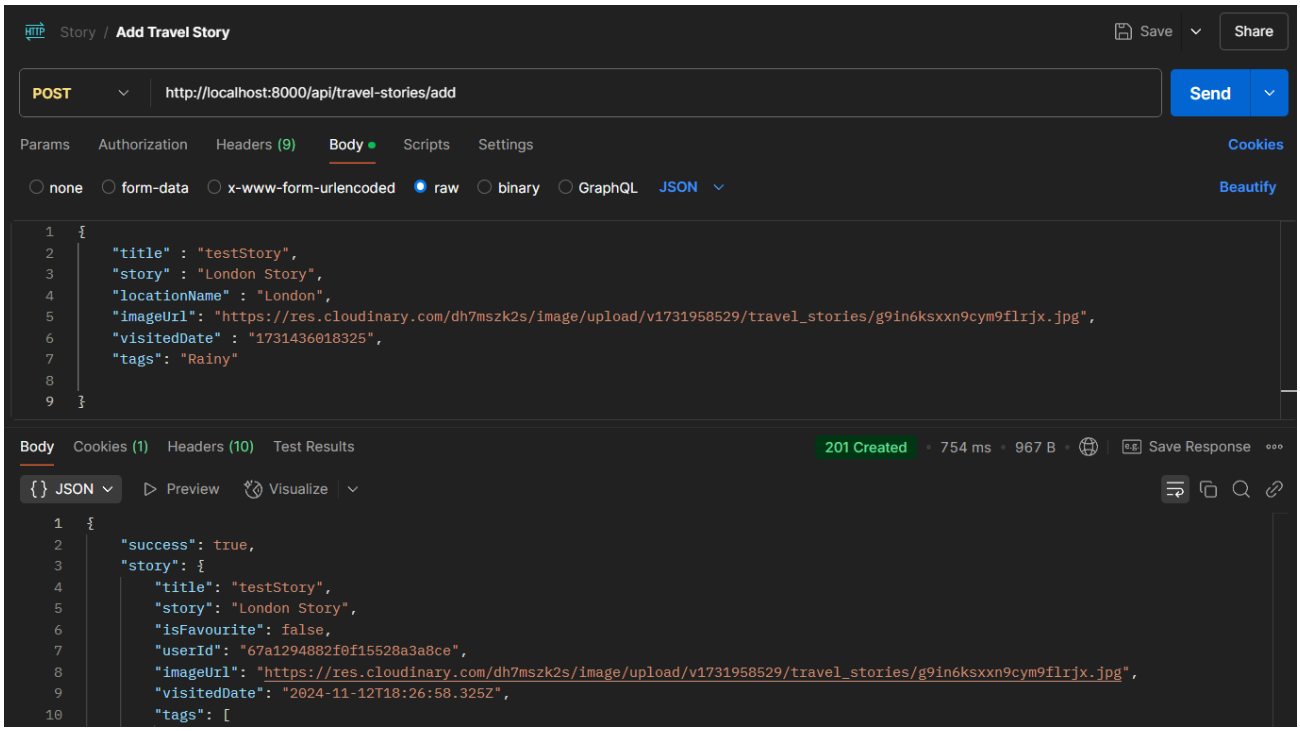


Рисунок 3.4 – Результат виконання запиту для додання нової історії подорожі

З результату видно, що запит виконано успішно. Також даний запит характеризується такими деталями:

– тип запиту: POST;

– запит: http://localhost:8000/api/travel-stories/add;

– статус відповіді: 201 OK, що свідчить про успішне виконання запиту;

– результат виконання запиту: “success”: true;

– поле “stories”: відображає деталі створеного запису про подорож.

Нижче наведено результат виконання запиту для отримання всіх подорожей (Рисунок 3.5):

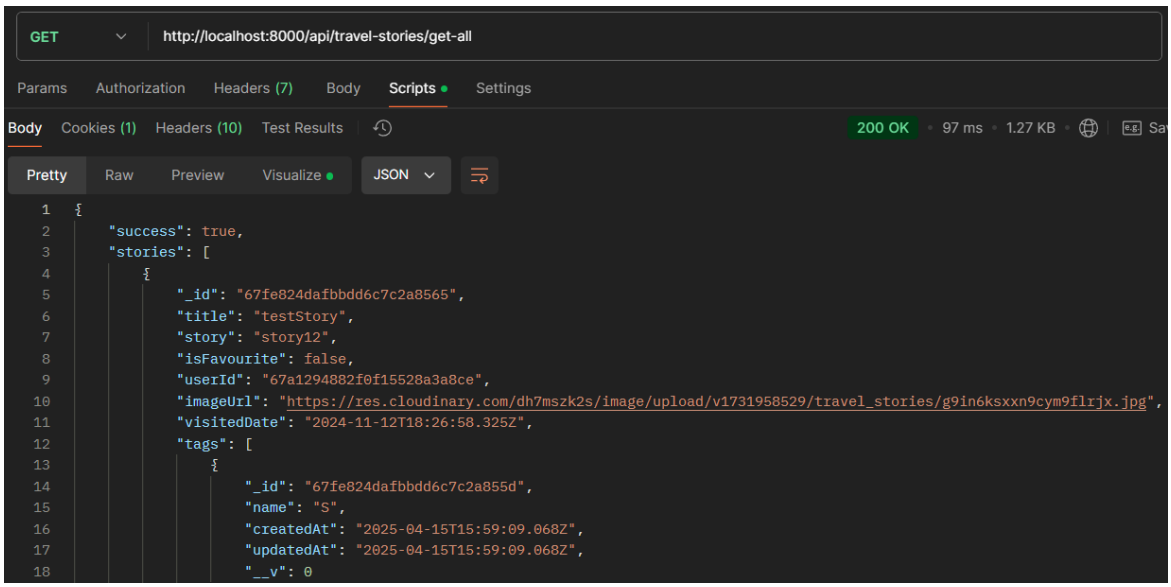


Рисунок 3.5 – Результат виконання запиту для отримання всіх історій подорожей

Для завантаження зображення використовується схожий алгоритм, але тут вказується формат даних (в даному випадку form-data), ключ та значення, яке є відповідним файлом з фото. Виконання запиту показано нижче (Рисунок 3.6):

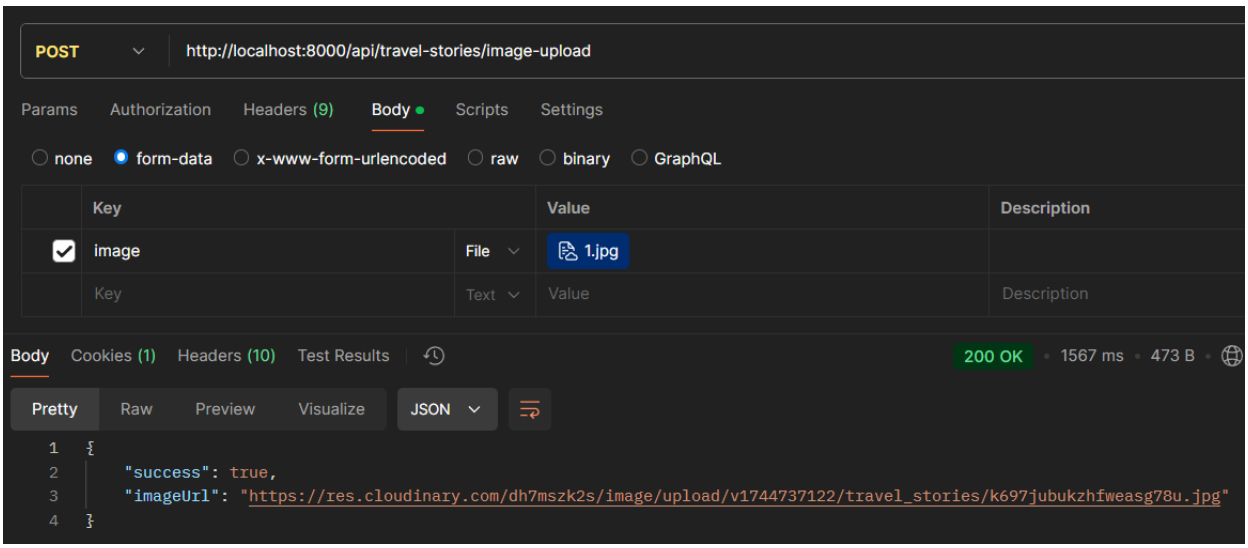


Рисунок 3.6 – Результат виконання запиту для завантаження зображення

3.5.3 Аналіз результатів тестування вебзастосунку

Тестування включало як перевірку з боку інтерфейсу, так і використання запитів API. Результати тестування з використанням фреймворку Cypress були показали успішність виконання всіх тестових випадків. Опис кожного з них наведено нижче (Таблиця 3.2):

Таблиця 3.2 – Результат тестування з використанням фреймворку Cypress

Назва	Коментар та очікуваний результат
Успішний вхід	Перевірка входу з правильними даними з переходом на головну сторінку
Відображення профілю	Перевірка відображення імені користувача (“TestUser”) після входу
Вихід	Перевірка виходу із системи та перехід на сторінку /login
Реєстрація з уже існуючим email	Перевірка наявності помилки при реєстрації з існуючою електронною поштою (код 400, повідомлення “User already exists”
Перевірка авторизації	Перевірка авторизації через запит API з поверненням коду 200 та коректними даними користувача
Отримання історій через API	Перевірка отримання всіх історій подорожей (код 200, отримання історій)
Додання історії через API	Перевірка створення нової історії подорожі (код 201, успішне додання історії)
Додання історії через інтерфейс	Перевірка створення нової історії через інтерфейс (відображення повідомлення "Story Added Successfully")
Редагування історії	Перевірка зміни інформації про подорож (повідомлення "Story Updated Successfully")
Видалення історії	Перевірка видалення історії (код 200 для API-запиту, повідомлення "Story Deleted Successfully")

Продовження таблиці 3.2

Додавання подорожі до списку бажаного	Перевірка додання нової подорожі до списку бажаного (відображення назви "Test Place")
Редагування подорожі списку бажаного	Перевірка зміни інформації про подорож (оновлення назви на "Edited Place", відображення поточних змін)
Видалення подорожі зі списку бажаного	Перевірка видалення елементу зі списку бажаного (тестовий елемент більше не відображається)
Відображення списку бажаного	Перевірка відображення двох подорожей в списку бажаного ("Test Place 1", "Test Place 2").
Сортування подорожей за назвою та рейтингом	Перевірка сортування подорожей за назвою (перший елемент 1 Place), та за рейтингом (перший елемент з рейтингом 5)
Швидке завантаження сторінки	Перевірка часу завантаження сторінки з 20 історіями (час завантаження становить менше 3 секунд)
Додавання історії на телефоні	Перевірка створення історій на мобільному телефоні (повідомлення "Story Added Successfully")
Валідація форми	Перевірка валідації форми (помилка "Please enter the title!", "Please enter the story!")

Оскільки усі описані вище тести пройшли успішно, це підтверджує правильну роботу окремих модулів вебзастосунку та його роботу в загальному. Результат тестування із переліком виконання всіх тестових сценаріїв також наведено нижче (Рисунок 3.7):

Рисунок 3.7 – Результати тестування

3.6 Висновки програмної реалізації, налагодження та тестування вебзастосунку

У цьому розділі розглянуто процес створення, налаштування та тестування вебзастосунку для планування й документування подорожей. Базу даних реалізовано на основі MongoDB із використанням бібліотеки Mongoose, що забезпечує зручну взаємодію з даними, валідацію та зв’язки між сутностями. Структура включає колекції User, TravelStory, Location, Wishlist і Tag.

Вебзастосунок було розроблено використовуючи MERN-стек, що дозволило реалізувати сучасний інтерфейс і надійну серверну логіку. Серед ключових модулів – автентифікація користувачів, управління подорожами та список бажаного. Зберігання зображень організовано через Cloudinary, а обмін даними реалізовано через REST API.

Для перевірки працездатності було застосовано функціональне, інтеграційне, наскрізне тестування, а також тести на безпеку й адаптивність. Використання Postman і Cypress дозволило ефективно перевірити як API-запити, так і взаємодію з інтерфейсом. Отримані результати демонструють надійність та відповідність вебзастосунку поставленим завданням.

ВИСНОВКИ

У процесі виконання кваліфікаційної роботи було реалізовано вебзастосунок для планування та документування туристичних подорожей, який допомагає мандрівникам зручно планувати поїздки та зберігати спогади про подорожі. Основним завданням було об'єднати в одному інструменті можливості персонального щоденника, списку бажаних місць і зручного інтерфейсу для перегляду та фільтрації записів.

На початковому етапі було проведено аналіз існуючих програмних рішень у цій сфері, зокрема Travel Diaries, Daybook та Polarsteps. Було виявлено їхні переваги та недоліки, що дозволило сформувати уявлення про те, що надають своїм користувачам дані сервіси. Далі було визначено функціональні та нефункціональні вимоги, які стали основою для подальших етапів проєктування та розробки.

На етапі проєктування було обрано архітектурну модель MVC, яка надала змогу чітко розмежувати відповідальність між окремими модулями, підвищити гнучкість та масштабованість системи. Розроблено логічну структуру бази даних із чітким визначенням сутностей (користувач, подорож, локація, список бажаного, тег) та зв'язків між ними. Дизайн інтерфейсу реалізовано у мінімалістичному стилі з акцентом на простоту, зручність навігації та адаптивність.

Програмна реалізація здійснювалась із використанням MERN-стека (MongoDB, Express.js, React, Node.js), що дозволило забезпечити високу інтегрованість компонентів, цілісність технологічного рішення та підвищити продуктивність розробки. Для зберігання зображень інтегровано сервіс Cloudinary. Було впроваджено захист персональних даних користувачів шляхом реалізації механізмів авторизації та аутентифікації на основі JSON Web Token. Усі дії з паролями проходять через хешування за допомогою бібліотеки bcrypt, що мінімізує ризики витоку конфіденційної інформації.

Для серверної частини було розроблено відповідні програмні модулі, які необхідні для правильного функціонування вебзастосунку. Це включало в себе реєстрацію користувача, створення та редагування подорожей із додаванням необхідних деталей, ведення списку бажаного, пошук та фільтрація записів. Клієнтська частина, реалізована на React, використовує REST API для динамічного завантаження та відображення контенту.

Для перевірки належної роботи вебзастосунку було проведено комплексне тестування. Застосовано інструменти Postman для перевірки API-запитів і Cypress для наскрізного та інтеграційного тестування з точки зору користувача. Окрім того, перевірено адаптивність інтерфейсу для різних пристроїв, а також стійкість системи до потенційних загроз безпеки. Результати тестування підтвердили коректну роботу вебзастосунку та відповідність функціоналу визначеним вимогам.

Розроблений вебзастосунок надає широкі можливості користування, забезпечуючи зручне та структуроване зберігання спогадів про подорожі, можливість додавання фотографій до записів, швидкий пошук та фільтрацію дописів за різними критеріями, збереження цікавих локацій у списку бажаного для планування майбутніх подорожей, а також надійний захист персональних даних.

Варто зазначити, що даний вебзастосунок має низку переваг для кінцевого користувача, серед яких: скорочення часу на організацію подорожей, зручність у збереженні даних, підвищення ефективності планування маршрутів та можливість зберігати найцінніші спогади в одному місці. Це дозволяє не лише підвищити якість особистого користувацького досвіду, а й робить застосунок корисним для широкого кола осіб, зацікавлених у туризмі.

Розробка також була опублікована у збірнику наукових праць за матеріалами XVI Всеукраїнської науково-практичної конференції «Актуальні проблеми комп’ютерних наук АПКН-2024» [42], що свідчить про її актуальність та наукову цінність.

У майбутньому передбачено подальший розвиток програмного забезпечення. Можливими напрямками є інтеграція із зовнішніми сервісами – туристичними платформами, соціальними мережами, системами геолокації та картами. Крім того, розробка мобільної версії дозволить забезпечити доступ до функціоналу з будь-якого пристрою, що розширить аудиторію користувачів і посилить практичне застосування системи.

Таким чином, поставлені завдання кваліфікаційної роботи реалізовано повною мірою, а розроблений вебзастосунок відповідає всім вимогам та має потенціал для подальшого розвитку та вдосконалення.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Essay on the Importance of Tourism Industry in Our Daily Life. URL: https://aithor.com/essay-examples/essay-on-the-importance-of-tourism-industry-in-our-daily-life (дата звернення 03.01.25)

2. Advantages and Disadvantages of Travelling Abroad in 2025. URL: https://terratern.com/blog/advantages-and-disadvantages-of-travelling-abroad/ (дата звернення 03.01.25)

3. Travel Journaling for Emotional Well-Being Benefits. URL: https://tripjive.com/travel-journaling-for-emotional-well-being-benefits/ (дата звернення 03.01.25)

4. Why You Should Document Your Travel Experiences. URL: https://www.roadsanddestinations.com/why-you-should-document-your-travel-experiences/ (дата звернення 05.01.25)

5. What is a Website? Definition, Types & Components. URL: https://www.techopedia.com/definition/5411/website (дата звернення 05.01.25)

6. What is Website: Definition and Impact on Business. URL: https://www.ramotion.com/blog/what-is-website/ (дата звернення 08.01.25)

7. 51 eCommerce Statistics In 2025 (Global and U.S. Data). URL: https://www.sellerscommerce.com/blog/ecommerce-statistics/ (дата звернення 11.01.25)

8. Travel Diaries App. URL: https://www.traveldiariesapp.com/en (дата звернення 16.01.25)

9. Daybook. URL: https://daybook.app/ (дата звернення 16.01.25)

10. Polarsteps. URL: https://www.polarsteps.com/ (дата звернення 16.01.25)

11. Writing Clear Functional and Non-functional Requirements: Tips and Best Practices from Apriorit Experts. URL: https://www.apriorit.com/dev-blog/creating-clear-functional-and-non-functional-requirements (дата звернення 19.01.25)

12. Functional vs. Non Functional Requirements. URL: https://www.geeksforgeeks.org/functional-vs-non-functional-requirements/ (дата звернення 21.01.25)

13. Use Case Diagram - Unified Modeling Language (UML). URL: https://www.geeksforgeeks.org/use-case-diagram/ (дата звернення 22.01.25)

14. What is a use case diagram? URL: https://miro.com/diagramming/what-is-a-use-case-diagram/ (дата звернення 22.01.25)

15. Types of Software Architecture Patterns. URL: https://www.geeksforgeeks.org/types-of-software-architecture-patterns/ (дата звернення 02.02.25)

16. Model-View-Controller (MVC) Architecture: Benefits and Challenges. URL: https://www.taazaa.com/mvc-architecture-benefits-and-challenges/ (дата звернення 03.02.25)

17. What Is Client-Server Architecture? URL: https://www.coursera.org/articles/client-server-architecture (дата звернення 04.02.25)

18. What are Microservices? How does Microservices architecture work? URL: https://middleware.io/blog/microservices-architecture/ (дата звернення 04.02.25)

19. 10 Types of Software Architecture Patterns. URL: https://distantjob.com/blog/software-architecture-patterns/ (дата звернення 06.02.25)

20. Building a Web Application: Understanding Business Logic in MVC Architecture. URL: https://medium.com/@devcorner/%EF%B8%8F-building-a-web-application-understanding-business-logic-in-mvc-architecture-b1b6f354780b#:~:text=Creating%20a%20web%20application%20means,%2DController%20(MVC)%20architecture. (дата звернення 08.02.25)

21. Logical Data Modeling. URL: https://medium.sqldbm.com/logical-data-modeling-60204851e829 (дата звернення 14.02.25)

22. Mastering Logical Database Models: A Comprehensive Guide. URL: https://risingwave.com/blog/mastering-logical-database-models-a-comprehensive-guide/ (дата звернення 16.02.25)

23. What is a User Interface (UI): Types & Examples. URL: https://dapth.com/insights/user-interface-ui (дата звернення 26.02.25)

24. What Are The Benefits Of Good User Interface Design? URL: https://kyledavidgroup.com/articles/good-user-interface-design/ (дата звернення 27.02.25)

25. MERN Stack for Beginners: Pros and Cons. URL: https://www.ropstam.com/mern-stack-for-beginners/ (дата звернення 04.03.25)

26. Why MERN Stack is the Future of Full-Stack Development. URL: https://medium.com/@emperorbrains/why-mern-stack-is-the-future-of-full-stack-development-bb6806ed8baa (дата звернення 04.03.25)

27. MongoDB Sharding. URL: https://www.mongodb.com/docs/manual/sharding/ (дата звернення 05.03.25)

28. What Is MongoDB? An Expert Guide. URL: https://www.oracle.com/ua/database/mongodb/ (дата звернення 05.03.25)

29. The Good and the Bad of Express.js Web Framework. URL: https://www.altexsoft.com/blog/expressjs-pros-and-cons/ (дата звернення 06.03.25)

30. What is Node js Used For? URL: https://pagepro.co/blog/what-is-node-js-used-for/ (дата звернення 06.03.25)

31. How Node.js Works: A Comprehensive Guide in 2025. URL: https://nodesource.com/blog/how-nodejs-works (дата звернення 07.03.25)

32. What is the virtual DOM in React? URL: https://blog.logrocket.com/the-virtual-dom-react/ (дата звернення 09.03.25)

33. Advantages and Benefits of React JS: The Essential Tool for Modern Developers. URL: https://maybe.works/blogs/react-js-advantages (дата звернення 10.03.25)

34. Why React JS Is So Popular: Key Benefits for Web and Mobile Development. URL: https://devsu.com/blog/why-is-why-react-js-is-so-popular-key-benefits-for-web-and-mobile-developmenteact-js-so-popular (дата звернення 13.03.25)

35. JWT Authentication: A Comprehensive Guide for Developers. URL: https://www.authgear.com/post/jwt-authentication-a-secure-scalable-solution-for-modern-applications (дата звернення 14.03.25)

36. Mongoose Connections. URL: https://mongoosejs.com/docs/connections.html (дата звернення 15.03.25)

37. Cloudinary. URL: https://cloudinary.com/ (дата звернення 17.03.25)

38. The Role of Testing in React JS Development: How to Ensure Quality And Reliability. URL: https://primathon.in/blog/the-role-of-testing-in-react-js-development-how-to-ensure-quality-and-reliability/ (дата звернення 25.03.25)

39. The Ultimate Guide to Testing React Applications. URL: https://www.angularminds.com/blog/guide-to-testing-react-applications (дата звернення 28.03.25)

40. Visual Testing in Cypress. URL: https://docs.cypress.io/app/tooling/visual-testing (дата звернення 02.04.25)

41. Postman Tutorial: How to Use Postman for API Testing. URL: https://apidog.com/blog/how-to-use-postman-for-api-testing/ (дата звернення 03.04.25)

42. Столярчук Є. І. ВЕБСАЙТ ДЛЯ ВЕДЕННЯ ІСТОРІЇ ПОДОРОЖЕЙ Актуальні проблеми комп'ютерних наук АПКН-2024 : зб. наук. пр., м. Хмельницький, 12 листоп. 2024 р. Хмельницький, 2024. С. 486–489. URL: https://kn.khmnu.edu.ua/wp-content/uploads/sites/18/apkn-2024-corpuspaper.pdf (дата звернення 10.04.25)

ДОДАТОК А

(обов’язковий)

ТЕХНІЧНЕ ЗАВДАННЯ

Введення

Ця робота здійснюється в межах проєкту зі створення вебзастосунку, призначеного для планування та документування туристичних подорожей. У сучасному світі інформаційні технології відіграють важливу роль у різних сферах, зокрема в туризмі. Подорожі допомагають розширити світогляд, познайомитися з новими культурами та набути унікального досвіду.

Умовне позначення системи: «Вебзастосунок для планування та документування туристичних подорожей».

1 Підстава для розробки

Проєкт розробляється на основі підстави «Завдання на дипломний проєкт», яке було затверджене завідувачем кафедри інженерії програмного забезпечення.

Назва розробки: Вебзастосунок для організації та збереження інформації про туристичні подорожі.

2 Призначення розробки

Вебзастосунок призначений для надання користувачам зручного та інтуїтивно зрозумілого інструменту для планування та документування туристичних подорожей, зберігаючи спогади про них. Користувачі мають змогу створювати нові історії, де описуються основні деталі про подорож. Крім того, вебзастосунок дозволяє зберігати локації, які користувачі планують відвідати в майбутньому.

Основний функціонал вебзастосунку включає:

– додавання, редагування та видалення записів із подорожами, включаючи основну інформацію;

– додання місць у список бажаного;

– пошук та фільтрація за різними параметрами;

Цей вебзастосунок призначений для людей, які хочуть зберігати свої спогади про подорожі та організувати свої туристичні поїздки в одному місці.

Вебзастосунок працює у всіх сучасних браузерах та на пристроях із доступ до Інтернету.

3 Вимоги до програмного забезпечення

3.1 Функціональні вимоги

Функціональні вимоги до вебзастосунку є наступними:

– створення облікового запису: користувач має можливість створити особистий профіль, використовуючи персональні дані, такі як: ім'я, електронна пошта та пароль. Пароль повинен мати довжину не менше 8 символів, включаючи великі та малі літери, а також цифри та спеціальні символи. Після успішної реєстрації користувач вебзастосунку може управляти плануванням та документуванням туристичних подорожей;

– додання нових записів: користувач може додавати нові записи про подорож, вказуючи назву, опис, фото, дату, локацію та теги. Всі записи повинні зберігатися у базі даних;

– редагування існуючих записів: користувач може редагувати записи в будь-який час. Внесення змін включає всі елементи, що стосуються опису історії подорожі. Всі зміни мають бути підтвердженні та відразу оновлюватись у користувача на сторінці:

– можливість пошуку: користувач може легко знаходити будь-які записи за їх назвою. Результати пошуку повинні бути згруповані за певним критерієм для зручності користувачів;

– можливість фільтрації: користувач може сортувати дописи за позначкою “Обране” та фільтрувати їх за датою, використовуючи інтерактивний календар на головній сторінці;

– список бажаного: користувач може додавати місця в особистий список бажаного. Даний список може зберігати просто цікаві локації, які вподобач

користувач або потенційні локації, де хотів би побувати мандрівник в майбутньому.

3.2 Нефункціональні вимоги

Нефункціональні вимоги до вебзастосунку є наступними:

– час завантаження вебзастосунку: програмне забезпечення повинно завантажуватись дуже швидко, при цьому, забезпечуючи коректну взаємодію користувача із вебсайтом. Час завантаження не повинен бути більшим за 2-3 секунди при звичайних умовах використання;

– перехід між сторінками: час на перехід між сторінками не повинен перевищувати 2 секунди;

– відповідність сучасним методам безпеки: вебзастосунок повинен використовувати HTTPS для шифрування даних при передачі між користувачем та сервером. При автентифікації потрібно застосовувати методи шифрування паролю для забезпечення захисту даних користувача;

– реалізація JSON Web Token (JWT): потреба у використанні стандарту JWT при реєстрації та вході в систему;

– розробка зрозумілого інтерфейсу: інтерфейс повинен бути інтуїтивно зрозумілим для користувача, що взаємодія із платформою. Елементи сторінки мають використовувати шрифт, який легко читається, а також містити приємну для очей кольорому гаму;

– доступність вебзастосунку: сервіс повинен бути доступним 24/7, окрім часу, виділеного та вирішення проблем.

3.3 Умови експлуатації

Вебзастосунок може використовуватись у сучасних браузерах на різних пристроях, таких як комп’ютери, планшети чи смартфони.

3.4 Вимоги до технічного забезпечення та продуктивності

Вебзастосунок призначений для роботи в браузерах та пристроях із доступом до мережі Інтернет.

Оскільки сам вебзастосунок орієнтований на веб використання, його встановлення на мобільний пристрій не є необхідним, тому доступ можна отримати використовуючи браузер.

Для коректної та стабільної роботи вебзастосунку рекомендується:

– операційна система: Windows, Linux, macOS, Android або iOS із підтримкою сучасних браузерів;

– процесор: двоядерний або краще (з частотою 1 ГГц і вище);

– оперативна пам’ять: мінімум 2 ГБ (рекомендовано 4 ГБ і вище);

– вільне місце на пристрої: не менше 500 МБ для кешування та збереження різних даних в браузері;

– інтернет-з’єднання: мінімальна швидкість 5 Мбіт/с.

3.5 Вимоги до програмної сумісності та інтеграцій

Вебзастосунок буде розроблений з використанням React для фронтенду, Node.js та Express.js для бекенду та MongoDB у якості бази даних для зберігання різної інформації. Для ефективної роботи із зберіганням зображень буде використовуватись сервіс Cloudinary.

Для розширення функціональності планується інтеграція із такими API:

– Cloudinary API: для збереження та обробки зображень;

– REST API: для взаємодії між частиною фронтенду та бекенду для забезпечення обробки запитів

3.6 Вимоги до інтерфейсу та дизайну

Вебзастосунок має наступні вимоги до дизайну:

– кольорова гама: використання білого, чорного та помаранчевого кольорів

– навігація: розташування пошуку, кнопки переходу на сторінку із списком бажань та профілю користувача у верхній частині хедера;

4 Вимоги до програмної документації

При здачі вебзастосунку для планування та документування туристичних подорожей замовнику буде наданий наступний комплект документації:

– програмний код та опис функціональних можливостей;

– інструкція користування;

– технічне завдання.

5 Стадії та етапи розробки

Стадія розробки	Етапи робіт	Зміст робіт
Технічне завдання	01.01.25 – 20.02.25	Формулювання вимог до системи та документація, призначення, опис цілей та завдань
Ескізний проєкт	21.02.25 – 20.03.25	Розробка дизайну інтерфейсу, обрання середовища програмування
Технічний проєкт	21.03.25 – 30.04.25	Опис технічних характеристик та архітектури
Робочий проєкт	01.05.25 – 25.05.25	Реалізація логіки та основних функціональних можливостей вебзастосунку. Інтеграція з базою даних та взаємодія із зовнішніми сервісами
Розробка програмної документації	Травень 2025	Оформлення документації, передбаченої технічним завдання
Тестування системи	26.05.25 – 30.05.25	Проведення різних тестів для перевірки правильності роботи функцій вебзастосунку
Впровадження	З 01.06.2025	Підготовка до розгортання системи, налаштування домену та передача вебзастосунку на експлуатацію користувачам

6 Порядок перевірки та приймання

Процес перевірки та приймання вебзастосунку здійснюється відповідно до визначених раніше вимог та містить такі етапи:

– функціональне тестування: перевірка основного функціоналу розробленого вебзастосунку;

– тестування продуктивності: оцінка часу завантаження сторінок, відповіді від сервера тощо;

– дотримання норм безпеки: перевірка захисту персональних даних, шифрування паролів, захист від атак;

– кросбраузерне тестування: перевірка правильності роботи в різних браузерах та пристроях;

– тестування інтеграції: перевірка взаємодії програмного забезпечення із сторонніми сервісами та API;

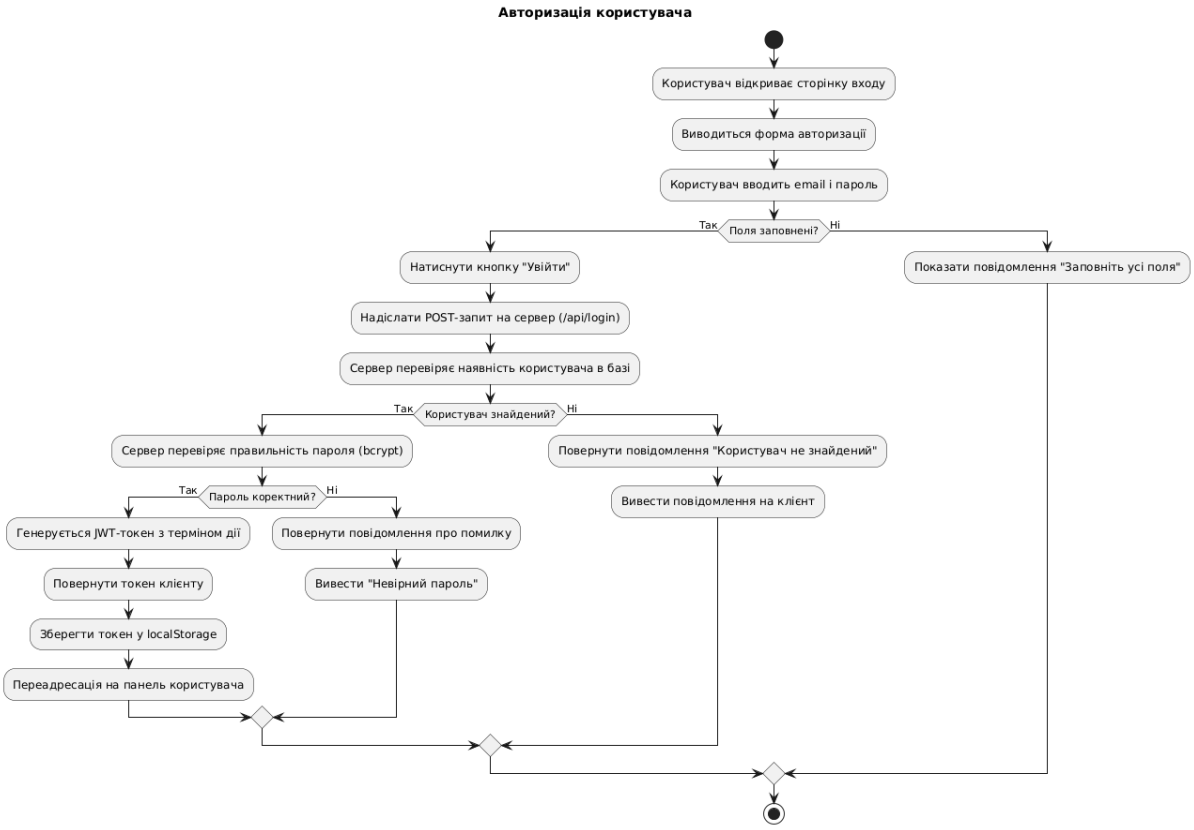
– тестування інтерфейсу: оцінка того, наскільки зручний та інтуїтивний інтерфейс різних сторінок та елементів вебзастосунку;

– відповідність документації: перевірка на відповідність встановленим вимогам та дотримання всіх інструкцій і також вирішення різноманітних проблем.

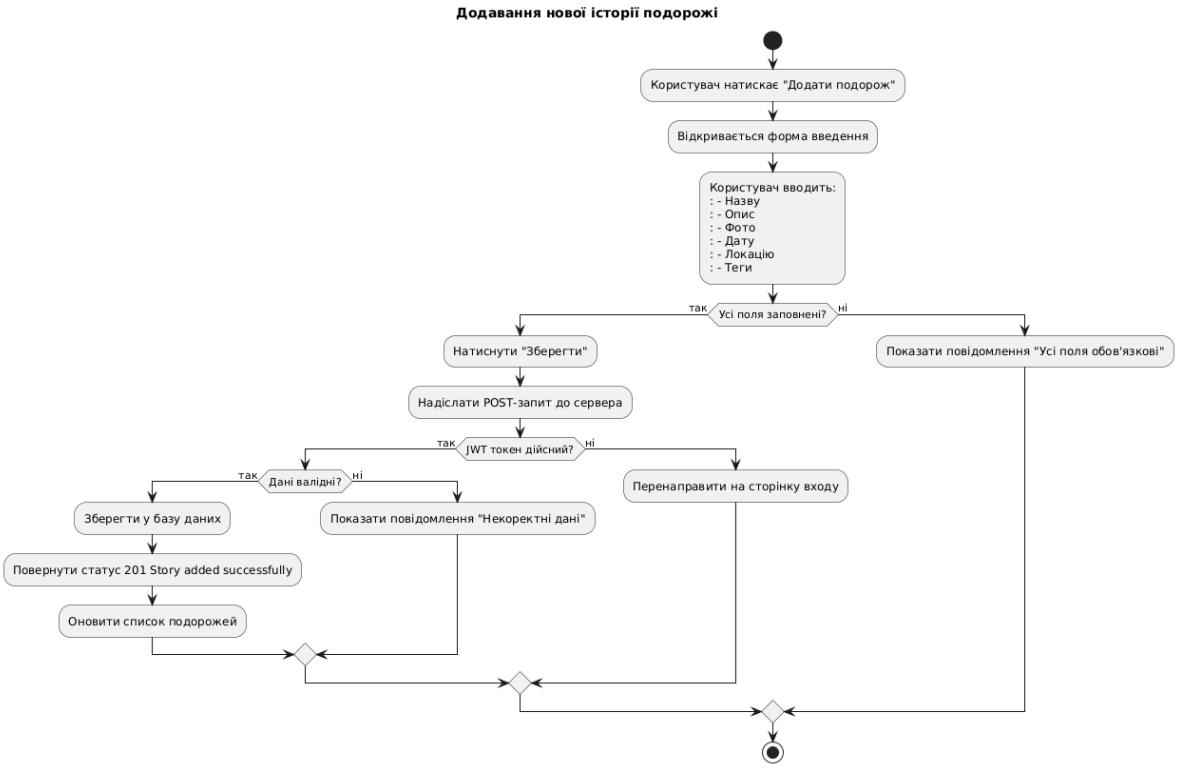
ДОДАТОК Б

(обов’язковий)

ДІАГРАМИ ДІЯЛЬНОСТІ



Додаток Б.1 – Діаграма діяльності авторизації користувача



Додаток Б.2 – Діаграма діяльності додання нової історії подорожі

ДОДАТОК В

(обов'язковий)

КОД ПРОГРАМИ

cloudinaryConfig.js:

```
import { v2 as cloudinary } from "cloudinary";
import dotenv from "dotenv";

dotenv.config();

cloudinary.config({
  cloud_name: process.env.CLOUDINARY_CLOUD_NAME,
  api_key: process.env.CLOUDINARY_API_KEY,
  api_secret: process.env.CLOUDINARY_API_SECRET,
});

export default cloudinary;
```

db.js:

```
import mongoose from "mongoose";

export const connectDB = async () => {
  try {
    await mongoose.connect(process.env.MONGO_URI);
    console.log("MongoDB connected");
  } catch (error) {
    console.error("MongoDB connection failed:", error);
    process.exit(1);
  }
};

export default connectDB;
```

authController.js:

```
import User from "../models/user.model.js";
import bcrypt from "bcrypt";
import { generateTokenAndSetCookie } from
"../utilities/generateTokenAndSetCookie.js";
import validator from "validator";

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

  try {
    if (!username || !email || !password) {
      return res
        .status(400)
        .json({ error: true, message: "All fields are required" });
    }

    if (!validator.isEmail(email)) {
      return res
        .status(400)
```

```
        .json({ error: true, message: "Invalid email format" });
    }

    if (password.length < 6) {
      return res.status(400).json({
        error: true,
        message: "Password must be at least 6 characters long",
      });
    }

    const userAlreadyExists = await User.findOne({ email });
    if (userAlreadyExists) {
      return res
        .status(400)
        .json({ error: true, message: "User already exists" });
    }

    const hashedPassword = await bcrypt.hash(password, 10);
    const user = new User({
      username,
      email,
      password: hashedPassword,
    });

    await user.save();
    generateTokenAndSetCookie(res, user._id);

    res.status(201).json({
      success: true,
      message: "User created successfully",
      user: {
        ...user._doc,
        password: undefined,
      },
    });
  } catch (error) {
    res.status(500).json({ error: true, message: "Registration failed" });
  }
};

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

  try {
    if (!email || !password) {
      return res
        .status(400)
        .json({ error: true, message: "All fields are required" });
    }

    const user = await User.findOne({ email });
    if (!user || !(await bcrypt.compare(password, user.password))) {
      return res
        .status(401)
        .json({ error: true, message: "Invalid credentials" });
    }

    generateTokenAndSetCookie(res, user._id);

    res.status(200).json({
      success: true,
      message: "Logged in successfully",
      user: {
        ...user._doc,
        password: undefined,
      },
```

```
      });
    } catch (error) {
      res.status(500).json({ error: true, message: "Login failed" });
    }
  };

  export const checkAuth = async (req, res) => {
    if (!req.user || !req.user.userId) {
      return res.status(401).json({ error: true, message: "Unauthorized" });
    }

    try {
      const user = await User.findById(req.user.userId).select("-password");
      if (!user) {
        return res.status(404).json({ error: true, message: "User not found"
});
      }

      res.status(200).json({ success: true, user });
    } catch (error) {
      console.log("Error in checkAuth:", error);
      res
        .status(500)
        .json({ error: true, message: "Failed to retrieve profile" });
    }
  };

  export const logout = async (req, res) => {
    const token = req.cookies?.token || req.headers.authorization?.split("
")[1];

    if (!token) {
      return res.status(401).json({ error: true, message: "Unauthorized" });
    }

    res.clearCookie("token", {
      httpOnly: true,
      secure: process.env.NODE_ENV === "production",
      sameSite: "strict",
    });

    res.status(200).json({ success: true, message: "Logged out successfully"
});
  };

  export const checkUser = async (req, res) => {
    const { email } = req.query;

    try {
      const user = await User.findOne({ email });
      if (user) {
        return res.status(200).json({ userExists: true });
      }
      return res.status(200).json({ userExists: false });
    } catch (error) {
      console.error("Error checking user:", error);
      return res
        .status(500)
        .json({ error: true, message: "Error checking user" });
    }
  };

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

    try {
```

```
    const user = await User.findOneAndDelete({ email });
    if (!user) {
      return res.status(404).json({ error: true, message: "User not found"
});
    }
    return res
      .status(200)
      .json({ success: true, message: "User deleted successfully" });
  } catch (error) {
    console.error("Error deleting user:", error);
    return res
      .status(500)
      .json({ error: true, message: "Failed to delete user" });
  }
};
```

travelStoryController.js:

```
import TravelStory from "../models/travelStory.model.js";
import Location from "../models/location.model.js";
import Tag from "../models/tag.model.js";
import cloudinary from "../config/cloudinaryConfig.js";

export const getAllStories = async (req, res) => {
  const { userId } = req.user;

  try {
    const travelStories = await TravelStory.find({ userId: userId })
      .sort({ isFavourite: -1 })
      .populate("tags")
      .populate("location");
    res.status(200).json({ success: true, stories: travelStories });
  } catch (error) {
    res.status(500).json({ success: false, message: error.message });
  }
};

export const addTravelStory = async (req, res) => {
  const { title, story, locationName, imageUrl, visitedDate, tags } =
req.body;
  const { userId } = req.user;

  if (!title || !story || !locationName || !visitedDate) {
    return res
      .status(400)
      .json({ success: false, message: "All fields are required" });
  }

  try {
    let location = await Location.findOne({ name: locationName });
    if (!location) {
      location = new Location({
        name: locationName,
      });
      await location.save();
    }

    let tagIds = [];
    if (tags && tags.length > 0) {
      for (const tagName of tags) {
        let tag = await Tag.findOne({ name: tagName });
        if (!tag) {
          tag = new Tag({ name: tagName });
          await tag.save();
        }
```

```
            tagIds.push(tag._id);
          }
        }

        const travelStory = new TravelStory({
          title,
          story,
          userId,
          imageUrl,
          visitedDate: new Date(parseInt(visitedDate)),
          tags: tagIds,
          location: location._id,
        });

        await travelStory.save();

        location.travelStoryId = travelStory._id;
        await location.save();

        res.status(201).json({
          success: true,
          story: travelStory,
          message: "Story added successfully",
        });
      } catch (error) {
        console.error("Failed to add story:", error);
        res.status(500).json({ success: false, message: "Failed to add story"
});
      }
    };

    export const uploadImage = async (req, res) => {
      try {
        const file =
`data:${req.file.mimetype};base64,${req.file.buffer.toString(
          "base64"
        )}`;
        const result = await cloudinary.uploader.upload(file, {
          folder: "travel_stories",
        });
        res.status(200).json({ success: true, imageUrl: result.secure_url });
      } catch (error) {
        res
          .status(500)
          .json({ success: false, message: "Failed to upload image." });
      }
    };

    export const editTravelStory = async (req, res) => {
      const { id } = req.params;
      const { title, story, locationName, imageUrl, visitedDate, tags } =
req.body;
      const { userId } = req.user;

      if (!title || !story || !locationName || !visitedDate) {
        return res
          .status(400)
          .json({ success: false, message: "All fields are required" });
      }

      try {
        const travelStory = await TravelStory.findOne({ _id: id, userId:
userId });
        if (!travelStory) {
          return res
            .status(404)
```

```
            .json({ success: false, message: "Travel story not found" });
        }

        let location = await Location.findOne({ travelStoryId: id });
        if (location) {
          location.name = locationName;
          await location.save();
        } else {
          location = new Location({
            name: locationName,
            travelStoryId: id,
          });
          await location.save();
        }

        let tagIds = [];
        if (tags && tags.length > 0) {
          for (const tagName of tags) {
            let tag = await Tag.findOne({ name: tagName });
            if (!tag) {
              tag = new Tag({ name: tagName });
              await tag.save();
            }
            tagIds.push(tag._id);
          }
        }

        travelStory.title = title;
        travelStory.story = story;
        travelStory.imageUrl = imageUrl || travelStory.imageUrl;
        travelStory.visitedDate = new Date(parseInt(visitedDate));
        travelStory.tags = tagIds;

        await travelStory.save();
        res.status(200).json({
          success: true,
          story: travelStory,
          message: "Story updated successfully",
        });
      } catch (error) {
        res.status(500).json({ success: false, message: "Failed to update
story" });
      }
    };

    export const deleteTravelStory = async (req, res) => {
      const { id } = req.params;
      const { userId } = req.user;

      try {
        const travelStory = await TravelStory.findOne({ _id: id, userId:
userId });
        if (!travelStory) {
          return res
            .status(404)
            .json({ success: false, message: "Travel story not found" });
        }

        await Location.deleteOne({ travelStoryId: id });

        await TravelStory.deleteOne({ _id: id, userId: userId });

        if (travelStory.imageUrl) {
          const publicId =
travelStory.imageUrl.split("/").pop().split(".")[0];
          await cloudinary.uploader.destroy("travel_stories/" + publicId);
```

```
    }

    res
      .status(200)
      .json({ success: true, message: "Travel story deleted successfully"
});
  } catch (error) {
    res.status(500).json({ success: false, message: error.message });
  }
};

export const deleteImage = async (req, res) => {
  const { imageUrl } = req.query;

  if (!imageUrl) {
    return res
      .status(400)
      .json({ error: true, message: "Image URL is required" });
  }

  try {
    const publicId = imageUrl.split("/").pop().split(".")[0];

    await cloudinary.uploader.destroy("travel_stories/" + publicId);

    res.status(200).json({ message: "Image deleted successfully" });
  } catch (error) {
    console.error("Error deleting image:", error);
    res.status(500).json({ error: true, message: "Failed to delete image"
});
  }
};

export const updateIsFavourite = async (req, res) => {
  const { id } = req.params;
  const { isFavourite } = req.body;
  const { userId } = req.user;

  try {
    const travelStory = await TravelStory.findOne({ _id: id, userId:
userId });
    if (!travelStory) {
      return res
        .status(404)
        .json({ success: false, message: "Travel story not found" });
    }

    travelStory.isFavourite = isFavourite;
    await travelStory.save();
    res.status(200).json({
      success: true,
      story: travelStory,
      message: "Update Successful",
    });
  } catch (error) {
    res.status(500).json({ success: false, message: error.message });
  }
};

export const searchTravelStories = async (req, res) => {
  const { query } = req.query;
  const { userId } = req.user;

  if (!query) {
    return res.status(404).json({ success: false, message: "Query not
found" });
```

```
  }

  try {
    const locations = await Location.find({
      name: { $regex: query, $options: "i" },
    });
    const locationIds = locations.map((loc) => loc.travelStoryId);

    const travelStories = await TravelStory.find({
      userId: userId,
      $or: [
        { title: { $regex: query, $options: "i" } },
        { _id: { $in: locationIds } },
      ],
    }).populate("tags");

    res.status(200).json({ success: true, stories: travelStories });
  } catch (error) {
    res.status(500).json({ success: false, message: error.message });
  }
};

export const filterTravelStories = async (req, res) => {
  const { startDate, endDate } = req.query;
  const { userId } = req.user;

  try {
    const start = new Date(parseInt(startDate));
    const end = new Date(parseInt(endDate));

    const filteredStories = await TravelStory.find({
      userId: userId,
      visitedDate: { $gte: start, $lte: end },
    })
      .sort({ isFavourite: -1 })
      .populate("tags");

    if (filteredStories.length === 0) {
      return res.status(200).json({
        success: true,
        message: "No stories found in this date range",
        stories: [],
      });
    }

    res.status(200).json({ success: true, stories: filteredStories });
  } catch (error) {
    res.status(500).json({ success: false, message: error.message });
  }
};

export const getUserStoryCount = async (req, res) => {
  const { userId } = req.user;
  try {
    const storyCount = await TravelStory.countDocuments({ userId });
    res.status(200).json({ success: true, storyCount });
  } catch (error) {
    console.error("Error fetching story count:", error);
    res.status(500).json({ error: "Failed to fetch story count" });
  }
};
```

wishlistController.js:

```
import Wishlist from "../models/wishlist.model.js";
```

```
import Location from "../models/location.model.js";
import cloudinary from "../config/cloudinaryConfig.js";

export const addWishlistItem = async (req, res) => {
  const { locationName, address, rating, notes, imageUrl } = req.body;
  const { userId } = req.user;

  if (!locationName) {
    return res
      .status(400)
      .json({ error: true, message: "Location name is required" });
  }

  try {
    let location = await Location.findOne({ name: locationName });
    if (!location) {
      location = new Location({
        name: locationName,
        address,
      });
      await location.save();
    } else {
      location.address = address;
      await location.save();
    }

    const newWishlistItem = new Wishlist({
      userId,
      locationId: location._id,
      locationName: locationName,
      rating,
      notes,
      imageUrl,
    });

    await newWishlistItem.save();
    res.status(201).json({ success: true, wishlistItem: newWishlistItem
});
  } catch (error) {
    console.error("Error adding a wishlist item", error);
    res
      .status(500)
      .json({ error: true, message: "Could not add item to wishlist" });
  }
};

export const uploadImage = async (req, res) => {
  try {
    const file =
`data:${req.file.mimetype};base64,${req.file.buffer.toString(
      "base64"
    )}`;
    const result = await cloudinary.uploader.upload(file, {
      folder: "wishlist_items",
    });
    res.status(200).json({ success: true, imageUrl: result.secure_url });
  } catch (error) {
    res
      .status(500)
      .json({ success: false, message: "Failed to upload image." });
  }
};

export const getWishlist = async (req, res) => {
  const { userId } = req.user;
```

```
        try {
          const wishlist = await Wishlist.find({ userId
}).populate("locationId");
          res.status(200).json({ wishlist });
        } catch (error) {
          console.error("Error retrieving wishlist", error);
          res
            .status(500)
            .json({ error: true, message: "Unable to retrieve wishlist" });
        }
      };

      export const deleteWishlistItem = async (req, res) => {
        const { id } = req.params;

        try {
          const wishlistItem = await Wishlist.findById(id);

          if (!wishlistItem) {
            return res
              .status(404)
              .json({ error: true, message: "Wishlist item not found" });
          }

          if (wishlistItem.imageUrl) {
            const publicId =
wishlistItem.imageUrl.split("/").pop().split(".")[0];
            await cloudinary.uploader.destroy("wishlist_items/" + publicId);
          }

          await Wishlist.findByIdAndDelete(id);

          res
            .status(200)
            .json({ success: true, message: "Item removed from wishlist" });
        } catch (error) {
          console.error("Error deleting wishlist item", error);
          res
            .status(500)
            .json({ error: true, message: "Failed to delete item from wishlist"
});
        }
      };

      export const updateWishlistItem = async (req, res) => {
        const { id } = req.params;
        const { locationName, address, rating, notes, imageUrl } = req.body;

        try {
          let updatedItem = await Wishlist.findById(id);
          if (!updatedItem) {
            return res
              .status(404)
              .json({ error: true, message: "Wishlist item not found" });
          }

          let location;
          if (locationName) {
            location = await Location.findOne({ name: locationName });
            if (!location) {
              location = new Location({
                name: locationName,
                address,
              });
              await location.save();
            } else {
```

```
            location.address = address;
            await location.save();
          }
          updatedItem.locationId = location._id;
        }

        if (address !== undefined) {
          updatedItem.address = address;
        }

        if (rating !== undefined) {
          updatedItem.rating = rating;
        }

        if (notes !== undefined) {
          updatedItem.notes = notes;
        }

        if (imageUrl) {
          updatedItem.imageUrl = imageUrl;
        }

        await updatedItem.save();

        res.status(200).json({ success: true, wishlistItem: updatedItem });
      } catch (error) {
        console.error("Error updating wishlist item", error);
        res
          .status(500)
          .json({ error: true, message: "Failed to update item in wishlist"
});
      }
    };

    export const deleteImage = async (req, res) => {
      const { imageUrl } = req.query;

      if (!imageUrl) {
        return res
          .status(400)
          .json({ error: true, message: "Image URL is required" });
      }

      try {
        const publicId = imageUrl.split("/").pop().split(".")[0];

        await cloudinary.uploader.destroy("wishlist_items/" + publicId);

        res.status(200).json({ message: "Image deleted successfully" });
      } catch (error) {
        console.error("Error deleting image:", error);
        res.status(500).json({ error: true, message: "Failed to delete image"
});
      }
    };

     verifyToken.js:
    import jwt from "jsonwebtoken"

    export const verifyToken = (req, res, next) => {
        const token = req.cookies.token;
        if (!token) return res.status(401).json({ success: false, message:
"Unauthorized - no token provided" });

        try {
            const decoded = jwt.verify(token, process.env.JWT_SECRET);
```

```
            if (!decoded) return res.status(401).json({ success: false,
message: "Unauthorized - invalid token" });

            req.user = decoded;
            next();
        } catch (error) {
            console.log("Error in verifyToken", error);
            return res.status(500).json({ success: false, message: "Server
error" });
        }
    };
```

location.model.js:

```
import mongoose from "mongoose";

const locationSchema = new mongoose.Schema(
  {
    name: {
      type: String,
      required: true,
    },
    address: {
      type: String,
    },
  },
  {
    timestamps: true,
  }
);

export const Location = mongoose.model("Location", locationSchema);
export default Location;
```

tag.model.js:

```
import mongoose from "mongoose";

const tagSchema = new mongoose.Schema(
  {
    name: {
      type: String,
      required: true,
      unique: true,
    },
  },
  { timestamps: true }
);

export const Tag = mongoose.model("Tag", tagSchema);
export default Tag;
```

travelStory.model.js:

```
import mongoose from "mongoose";

const travelStorySchema = new mongoose.Schema(
  {
    title: {
      type: String,
      required: true,
```

```
    },
    story: {
      type: String,
      required: true,
    },
    isFavourite: {
      type: Boolean,
      default: false,
    },
    userId: {
      type: mongoose.Schema.Types.ObjectId,
      ref: "User",
      required: true,
    },
    imageUrl: {
      type: String,
    },
    visitedDate: {
      type: Date,
      required: true,
    },
    tags: [
      {
        type: mongoose.Schema.Types.ObjectId,
        ref: "Tag",
        required: true,
      },
    ],
    location: {
      type: mongoose.Schema.Types.ObjectId,
      ref: "Location",
      required: true,
    },
  },
  {
    timestamps: true,
  }
);

export const TravelStory = mongoose.model("TravelStory",
travelStorySchema);
export default TravelStory;
```

user.model.js:

```
import mongoose from "mongoose";

const userSchema = new mongoose.Schema(
    {
        username: {
            type: String,
            required: true
        },
        email : {
            type: String,
            required: true,
            unique: true
        },
        password : {
            type: String,
            required: true
        }
    },
    {
      timestamps: true,
```

```
}
);

export const User = mongoose.model("User", userSchema);
export default User;
```

wishlist.model.js:

```
import mongoose from "mongoose";

const wishlistSchema = new mongoose.Schema(
  {
    userId: {
      type: mongoose.Schema.Types.ObjectId,
      ref: "User",
      required: true,
    },
    locationId: {
      type: mongoose.Schema.Types.ObjectId,
      ref: "Location",
      required: true,
    },
    rating: {
      type: Number,
      min: 1,
      max: 5,
    },
    notes: {
      type: String,
    },
    imageUrl: {
      type: String,
    },
  },
  {
    timestamps: true,
  }
);

export const Wishlist = mongoose.model("Wishlist", wishlistSchema);
export default Wishlist;
```

authRoutes.js:

```
import express from "express";
import {
  checkAuth,
  signup,
  login,
  logout,
  checkUser,
  deleteUser,
} from "../controllers/authController.js";
import { verifyToken } from "../middleware/verifyToken.js";

const router = express.Router();

router.get("/check-auth", verifyToken, checkAuth);
router.get("/check-user", checkUser);
router.post("/signup", signup);
router.post("/login", login);
router.post("/logout", logout);
router.delete("/delete-user", deleteUser);
```

```
export default router;
```

travelStoryRoutes.js:

```
import express from "express";
import { verifyToken } from "../middleware/verifyToken.js";
import {
  addTravelStory,
  uploadImage,
  getAllStories,
  editTravelStory,
  deleteTravelStory,
  deleteImage,
  updateIsFavourite,
  searchTravelStories,
  filterTravelStories,
  getUserStoryCount,
} from "../controllers/travelStoryController.js";
import { upload } from "../utilities/multer.js";

const router = express.Router();

router.post("/add", verifyToken, addTravelStory);
router.post("/image-upload", verifyToken, upload.single("image"),
uploadImage);
router.get("/get-all", verifyToken, getAllStories);
router.put("/edit/:id", verifyToken, editTravelStory);
router.delete("/delete/:id", verifyToken, deleteTravelStory);
router.put("/update-is-favourite/:id", verifyToken, updateIsFavourite);
router.get("/search", verifyToken, searchTravelStories);
router.get("/filter", verifyToken, filterTravelStories);
router.delete("/delete-image", verifyToken, deleteImage);
router.get("/user-story-count", verifyToken, getUserStoryCount);

export default router;
```

wishlistRoutes.js:

```
import express from "express";
import {
  addWishlistItem,
  getWishlist,
  updateWishlistItem,
  deleteWishlistItem,
  uploadImage,
  deleteImage,
} from "../controllers/wishlistController.js";
import { verifyToken } from "../middleware/verifyToken.js";
import { upload } from "../utilities/multer.js";

const router = express.Router();

router.get("/", verifyToken, getWishlist);
router.post("/", verifyToken, addWishlistItem);
router.post("/image-upload", verifyToken, upload.single("image"),
uploadImage);
router.delete("/delete-image", verifyToken, deleteImage);

router.put("/:id", verifyToken, updateWishlistItem);
router.delete("/:id", verifyToken, deleteWishlistItem);

export default router;
```

index.js:

```
import dotenv from "dotenv";
import express from "express";
import cors from "cors";
import connectDB from "./config/db.js";
import cookieParser from "cookie-parser";

import authRoutes from "./routes/authRoutes.js";
import travelStoryRoutes from "./routes/travelStoryRoutes.js";
import wishlistRoutes from "./routes/wishlistRoutes.js";

dotenv.config();

const app = express();

connectDB();

app.use(express.json());
app.use(cookieParser());
//app.use(cors({ origin: "*" }));
app.use(
  cors({
    origin: "http://localhost:5173",
    credentials: true,
  })
);

app.use("/api/auth", authRoutes);
app.use("/api/travel-stories", travelStoryRoutes);
app.use("/api/wishlist", wishlistRoutes);

const PORT = process.env.PORT || 8000;
app.listen(PORT, () => {
  console.log(`Server is running on port ${PORT}`);
});

export default app;
```

package.json:

```
{
  "name": "backend",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "test": "node --experimental-vm-modules node_modules/jest/bin/jest.js
--detectOpenHandles",
    "start": "nodemon index.js"
  },
  "jest": {
    "transform": {}
  },
  "type": "module",
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": "",
  "dependencies": {
    "axios": "^1.7.7",
    "bcrypt": "^5.1.1",
    "cloudinary": "^2.5.1",
```

```
    "cookie-parser": "^1.4.7",
    "cors": "^2.8.5",
    "dotenv": "^16.4.5",
    "express": "^4.21.1",
    "jest": "^29.7.0",
    "jsonwebtoken": "^9.0.2",
    "mongoose": "^8.8.0",
    "multer": "^1.4.5-lts.1",
    "multer-storage-cloudinary": "^4.0.0",
    "nodemon": "^3.1.7",
    "react-icons": "^5.3.0",
    "supertest": "^7.0.0",
    "validator": "^13.12.0"
  },
  "devDependencies": {
    "chai": "^5.1.2",
    "mocha": "^11.1.0",
    "mongodb-memory-server": "^10.1.3",
    "sinon": "^19.0.2"
  }
}
```

EmptyCard.jsx:

```
import React from 'react'

const EmptyCard = ({ imgSrc, message }) => {
  return (
    <div className='flex flex-col items-center justify-center mt-20'>
      <img src={imgSrc} alt='No notes' className='w-24'/>

      <p className='w-1/2 text-sm font-medium text-slate-700 text-center
leading-7 mt-5'>
        {message}
      </p>
    </div>
  )
}

export default EmptyCard
```

FilterInfoTitle.jsx:

```
import React from "react";
import moment from "moment";
import { MdOutlineClose } from "react-icons/md";

const FilterInfoTitle = ({ filterType, filterDates, onClear }) => {
  const DateRangeChip = ({ date }) => {
    const startDate = date?.from ? moment(date?.from).format("Do MMM
YYYY") : "N/A";
    const endDate = date?.to ? moment(date?.to).format("Do MMM YYYY") :
"N/A";

    return (
      <div className="flex items-center gap-2 bg-slate-100 px-3 py-2
rounded">
        <p className="text-xs font-medium ">
          {startDate} - {endDate}
        </p>

        <button onClick={onClear}>
          <MdOutlineClose />
```

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

      return (
        filterType && (
          <div className="mb-5">
            {filterType === "search" ? (
              <h3 className="text-lg font-medium">Search Results</h3>
            ) : (
              <div className="flex items-center gap-2">
                <h3 className="text-lg font-medium">Travel Stories from</h3>

                <DateRangeChip date={filterDates} />
              </div>
            )}
          </div>
        )
      );
    };

    export default FilterInfoTitle;
```

ProfileInfo.jsx:

```
    import React from "react";
    import { getInitials } from "../../../utils/helper.js";
    import { useNavigate } from "react-router-dom";

    const ProfileInfo = ({ userInfo, onLogout }) => {
      const navigate = useNavigate();

      if (!userInfo) return null;

      const handleProfileClick = () => {
        navigate("/profile");
      };

      return (
        <div className="flex items-center gap-3">
          <div
            className="w-12 h-12 flex items-center justify-center rounded-full
text-slate-950 font-medium bg-slate-100 cursor-pointer"
            onClick={handleProfileClick}
          >
            {getInitials(userInfo.username)}
          </div>

          <div className="flex flex-col">
            <p
              onClick={handleProfileClick}
              className="text-sm font-medium cursor-pointer hover:text-
[#E5906D]"
            >
              {userInfo.username || "User"}
            </p>

            <button
              className="cursor-pointer"
              data-testid="logout-button"
              style={{ color: "#a2674a", fontSize: "0.775rem" }}
              onClick={onLogout}
            >
              Logout
```

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

    export default ProfileInfo;
```

TravelStoryCard.jsx:

```
    import React from "react";
    import moment from "moment";
    import { FaHeart } from "react-icons/fa6";
    import { GrMapLocation } from "react-icons/gr";

    const TravelStoryCard = ({
      imgUrl,
      title,
      date,
      story,
      location,
      tags,
      isFavourite,
      onFavouriteClick,
      onClick,
    }) => {
      const getLocationName = () => {
        if (!location) return null;

        if (typeof location === "object" && location.name) {
          return location.name;
        }

        if (typeof location === "string") {
          return location;
        }

        return null;
      };

      const locationName = getLocationName();

      return (
        <div className="story-item relative border rounded-lg overflow-hidden
bg-white hover:shadow-lg hover:shadow-slate-200 transition-all ease-in-out
cursor-pointer">
          {imgUrl && (
            <img
              src={imgUrl}
              alt={title}
              className="w-full h-56 object-cover rounded-lg"
              onClick={onClick}
            />
          )}

          <button
            className="w-12 h-12 flex items-center justify-center bg-white/40
rounded-lg border border-white/30 absolute top-4 right-4"
            onClick={(e) => {
              e.stopPropagation();
              onFavouriteClick();
            }}
          >
            <FaHeart
```

```
            className={`icon-btn ${isFavourite ? "text-red-500" : "text-
white"}`}
          />
        </button>

        <div className="p-4" onClick={onClick}>
          <div className="flex items-center gap-3">
            <div className="flex-1">
              <h6 className="text-sm font-medium">{title}</h6>
              <span className="text-xs text-slate-500">
                {date ? moment(date).format("Do MMM YYYY") : "-"}
              </span>
            </div>
          </div>

          <p className="text-xs text-slate-600 mt-2 whitespace-normal break-
words">
            {story?.slice(0, 60)}
            {story?.length > 60 && "..."}
          </p>

          <div className="flex flex-wrap gap-2 mt-3">
            {locationName && (
              <div
                className="inline-flex items-center gap-2 text-[13px]
rounded px-2 py-1"
                style={{ color: "#FFFFFF", backgroundColor: "#FFA07A" }}
              >
                <GrMapLocation className="text-sm" />
                <span>{locationName}</span>
              </div>
            )}

            {tags &&
              tags.length > 0 &&
              tags.map((tag, index) => (
                <span
                  key={typeof tag === "object" ? tag._id : index}
                  className="inline-flex items-center gap-2 text-[13px]
rounded px-2 py-1"
                  style={{
                    color: "#FFFFFF",
                    backgroundColor: "#FFA07A",
                  }}
                >
                  #{typeof tag === "object" ? tag.name : tag}
                </span>
              ))}
          </div>
        </div>
      </div>
    );
  };

  export default TravelStoryCard;
```

DateSelector.jsx:

```
import React, { useState } from "react";
import { MdOutlineDateRange, MdClose } from "react-icons/md";
import { DayPicker } from "react-day-picker";
import moment from "moment";

const DateSelector = ({ date, setDate }) => {
  const [openDatePicker, setOpenDatePicker] = useState(false);
```

```
        const handleDaySelect = (selectedDay) => {
          setDate(selectedDay);
          setOpenDatePicker(false);
        };

        return (
          <div data-testid="date-selector">
            <button
              className="inline-flex items-center gap-2 text-[13px] font-medium
text-orange-600 bg-orange-100 hover:bg-orange-200 rounded px-2 py-1 cursor-
pointer"
              onClick={() => {
                setOpenDatePicker(true);
              }}
            >
              <MdOutlineDateRange className="text-lg" />
              {date
                ? moment(date).format("Do MMM YYYY")
                : moment().format("Do MMM YYYY")}
            </button>

            {openDatePicker && (
              <div className="overflow-y-scroll p-5 bg-orange-50/80 rounded-lg
relative pt-9 absolute z-10">
                <button
                  className="w-10 h-10 rounded-full flex items-center justify-
center bg-orange-100 hover-bg-orange-100 absolute top-2 right-2"
                  onClick={() => {
                    setOpenDatePicker(false);
                  }}
                >
                  <MdClose className="text-xl text-orange-600" />
                </button>

                <DayPicker
                  captionLayout="dropdown-buttons"
                  mode="single"
                  selected={date}
                  onSelect={handleDaySelect}
                  pagedNavigation
                  classNames={{
                    day_selected: "bg-orange-600 text-white rounded",
                    day_today: "bg-orange-100 text-orange-600 rounded",
                  }}
                />
              </div>
            )}
          </div>
        );
      };

      export default DateSelector;
```

ImageSelector.jsx:

```
      import React, { useEffect, useRef, useState } from "react";
      import { FaRegFileImage } from "react-icons/fa";
      import { MdDeleteOutline } from "react-icons/md";

      const ImageSelector = ({ image, setImage, handleDeleteImg }) => {
        const inputRef = useRef(null);
        const [previewUrl, setPreviewUrl] = useState(null);

        const handleImageChange = (event) => {
```

```
    const file = event.target.files[0];
    if (file) {
      setImage(file);
    }
  };

  const onChooseFile = (e) => {
    e.preventDefault();
    e.stopPropagation();
    inputRef.current.click();
  };

  const handleRemoveImage = (e) => {
    e.preventDefault();
    e.stopPropagation();
    setImage(null);
    handleDeleteImg();
  };

  useEffect(() => {
    if (typeof image === "string") {
      setPreviewUrl(image);
    } else if (image) {
      setPreviewUrl(URL.createObjectURL(image));
    } else {
      setPreviewUrl(null);
    }

    return () => {
      if (
        previewUrl &&
        typeof previewUrl === "string" &&
        !previewUrl.startsWith("http") &&
        !image
      ) {
        URL.revokeObjectURL(previewUrl);
      }
    };
  }, [image]);

  return (
    <div onClick={(e) => e.stopPropagation()}>
      <input
        type="file"
        accept="image/*"
        ref={inputRef}
        onChange={handleImageChange}
        className="hidden"
      />

      {!image ? (
        <button
          type="button"
          className="w-full h-[220px] flex flex-col items-center justify-
center gap-4 bg-slate-50 rounded border border-slate-200/50"
          onClick={onChooseFile}
        >
          <div className="w-14 h-14 flex items-center justify-center bg-
cyan-50 rounded-full border border-cyan-100">
            <FaRegFileImage className="text-xl text-[#FFA07A]" />
          </div>

          <p className="text-sm text-slate-500">Browse image files to
upload</p>
        </button>
      ) : (
```

```
              <div className="w-full relative" onClick={(e) =>
e.stopPropagation()}>
                <img
                  src={previewUrl}
                  alt="Selected"
                  className="w-full h-[300px] object-cover rounded-lg"
                />

                <button
                  type="button"
                  className="btn-small-image btn-delete-image absolute top-2
right-2"
                  onClick={handleRemoveImage}
                >
                  <MdDeleteOutline className="text-lg" />
                </button>
              </div>
            )}
          </div>
        );
      };

      export default ImageSelector;
```

PasswordInput.jsx:

```
      import React, { useState } from "react";
      import { FaRegEye, FaRegEyeSlash } from "react-icons/fa6";

      const PasswordInput = ({ value, onChange, placeholder }) => {
        const [isShowPassword, setIsShowPassword] = useState(false);

        const toggleShowPassword = () => {
          setIsShowPassword(!isShowPassword);
        };

        return (
          <div className="flex items-center bg-orange-600/5 px-5 rounded mb-3">
            <input
              name="password"
              value={value}
              onChange={onChange}
              placeholder={placeholder || "Password"}
              type={isShowPassword ? "text" : "password"}
              className="w-full text-sm bg-transparent py-3 mr-3 rounded
outline-none"
            />

            {isShowPassword ? (
              <FaRegEye
                size={22}
                className="text-primary cursor-pointer"
                onClick={() => toggleShowPassword()}
              />
            ) : (
              <FaRegEyeSlash
                size={22}
                className="text-primary cursor-pointer"
                onClick={() => toggleShowPassword()}
              />
            )}
          </div>
        );
      };
```

```
export default PasswordInput;
```

SearchBar.jsx:

```
import React from "react";
import { FaMagnifyingGlass } from "react-icons/fa6";
import { IoMdClose } from "react-icons/io";

const SearchBar = ({ value, onChange, handleSearch, onClearSearch }) => {
  return (
    <div className="w-80 flex items-center px-4 bg-slate-100 rounded-md">
      <input
        type="text"
        placeholder="Search Notes"
        className="w-full text-xs bg-transparent py-[11px] outline-none"
        value={value}
        onChange={onChange}
        data-testid="search-input"
      />

      {value && (
        <IoMdClose
          className="text-xl text-slate-500 cursor-pointer hover:text-
black mr-3"
          onClick={onClearSearch}
        />
      )}

      <FaMagnifyingGlass
        className="text-slate-400 cursor-pointer hover:text-black"
        onClick={handleSearch}
        data-testid="search-button"
      />
    </div>
  );
};

export default SearchBar;
```

TagInput.jsx:

```
import React, { useState } from "react";
import { MdAdd, MdClose } from "react-icons/md";
import { GrMapLocation } from "react-icons/gr";

const TagInput = ({ tags, setTags, inputType, maxTags = Infinity }) => {
  const [inputValue, setInputValue] = useState("");

  const addNewTag = () => {
    if (
      inputValue.trim() &&
      !tags.includes(inputValue.trim()) &&
      tags.length < maxTags
    ) {
      setTags([...tags, inputValue.trim()]);
      setInputValue("");
    }
  };

  const handleInputChange = (e) => {
    setInputValue(e.target.value);
  };
```

```
        const handleKeyDown = (e) => {
          if (e.key === "Enter") {
            e.preventDefault();
            addNewTag();
          }
        };

        const handleRemoveTag = (tagToRemove) => {
          setTags(tags.filter((tag) => tag !== tagToRemove));
        };

        return (
          <div>
            {tags.length > 0 && (
              <div className="flex items-center gap-2 flex-wrap mt-2">
                {tags.map((tag, index) => (
                  <span
                    key={index}
                    className="flex items-center gap-2 text-sm text-orange-600
bg-orange-100 px-3 py-1 rounded"
                    data-testid="tag"
                  >
                    {inputType === "hashtag" ? (
                      <span className="text-sm">#</span>
                    ) : (
                      <GrMapLocation className="text-sm" />
                    )}
                    {tag}
                    <button
                      onClick={() => handleRemoveTag(tag)}
                      data-testid="remove-tag-button"
                    >
                      <MdClose />
                    </button>
                  </span>
                ))}
              </div>
            )}

            <div className="flex items-center gap-4 mt-3">
              <input
                type="text"
                name={inputType === "location" ? "locationName" : undefined}
                value={inputValue}
                className="text-sm bg-transparent border px-3 py-2 rounded
outline-none"
                placeholder={inputType === "hashtag" ? "Add Hashtag" : "Add
Location"}
                onChange={handleInputChange}
                onKeyDown={handleKeyDown}
              />

              <button
                className="w-8 h-8 flex items-center justify-center rounded
border border-[#FFA07A] hover:bg-[#FFA07A]"
                onClick={addNewTag}
              >
                <MdAdd className="text-2xl text-[#FFA07A] hover:text-white" />
              </button>
            </div>
          </div>
        );
      };

      export default TagInput;
```

Navbar.jsx:

```
import React from "react";
import logo from "../assets/images/logo.png";
import ProfileInfo from "./Cards/ProfileInfo";
import { useNavigate } from "react-router-dom";
import SearchBar from "./Input/SearchBar";
import { axiosInstance } from "../../utils/axiosInstance";

const Navbar = ({
  userInfo,
  searchQuery,
  setSearchQuery,
  onSearchNote,
  handleClearSearch,
}) => {
  const navigate = useNavigate();

  const onLogout = async () => {
    try {
      await axiosInstance.post("/api/auth/logout");
      navigate("/login");
    } catch (error) {
      console.error("Logout failed:", error);
    }
  };

  const handleSearch = () => {
    if (searchQuery) {
      onSearchNote(searchQuery);
    }
  };

  const onClearSearch = () => {
    handleClearSearch();
    setSearchQuery("");
  };

  return (
    <div className="bg-white flex items-center justify-between px-6 py-2
drop-shadow sticky top-0 z-10">
      <div className="cursor-pointer" onClick={() =>
navigate("/dashboard")}>
        <img src={logo} alt="logo" className="h-9" />
      </div>

      {userInfo && (
        <>
          <div className="flex items-center gap-4">
            <SearchBar
              value={searchQuery}
              onChange={({ target }) => {
                setSearchQuery(target.value);
              }}
              handleSearch={handleSearch}
              onClearSearch={onClearSearch}
              data-testid="search-input"
            />
            <button
              className="bg-[#FFA07A] hover:bg-[#E5906D] text-white px-4
py-2 rounded focus:outline-none"
              onClick={() => (window.location.href = "/wishlist")}
            >
              Wishlist
```

```
            </button>
          </div>

          <ProfileInfo userInfo={userInfo} onLogout={onLogout} />
        </>
      )}
    </div>
  );
};

export default Navbar;
```

Login.jsx:

```
import React, { useState } from "react";
import PasswordInput from "../../components/Input/PasswordInput";
import { useNavigate } from "react-router-dom";
import { validateEmail } from "../../../utils/helper";
import { axiosInstance } from "../../../utils/axiosInstance";

const Login = () => {
  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");
  const [error, setError] = useState(null);

  const navigate = useNavigate();

  const handleLogin = async (e) => {
    e.preventDefault();

    if (!validateEmail(email)) {
      setError("Please enter a valid email address.");
      return;
    }

    if (!password) {
      setError("Please enter the password.");
      return;
    }

    setError("");

    try {
      const response = await axiosInstance.post("/api/auth/login", {
        email: email,
        password: password,
      });

      if (response.data && response.data.success) {
        navigate("/dashboard");
      }
    } catch (error) {
      if (
        error.response &&
        error.response.data &&
        error.response.data.message
      ) {
        setError(error.response.data.message);
      } else {
        setError("An unexpected error occurred. Please try again.");
      }
    }
  };

  return (
```

```
          <div className="h-screen bg-cyan-50 overflow-hidden relative">
            <div className="login-ui-box bg-orange-200 right-10 -top-40" />
            <div className="login-ui-box" />

            <div className="container h-screen flex items-center justify-center
px-20 mx-auto">
              <div className="w-5/12 h-[90vh] flex flex-col justify-end bg-
login-bg-image bg-cover bg-center rounded-lg p-10 z-50">
                <h4 className="text-5xl text-white font-semibold leading-
[58px]">
                  Trace Your <br /> Journey's Story
                </h4>
                <p className="text-[15px] text-white leading-6 pr-7 mt-4">
                  Create a timeless journal of places, moments, and memories
worth
                  remembering.
                </p>
              </div>

              <div className="w-2/4 h-[75vh] bg-white rounded-r-lg relative p-16
shadow-lg shadow-cyan-200/20">
                <form onSubmit={handleLogin}>
                  <h4 className="text-2xl font-semibold mb-7">Login</h4>

                  <input
                    type="text"
                    name="email"
                    placeholder="Email"
                    className="input-box"
                    value={email}
                    onChange={({ target }) => {
                      setEmail(target.value);
                    }}
                  />

                  <PasswordInput
                    value={password}
                    onChange={({ target }) => {
                      setPassword(target.value);
                    }}
                  />

                  {error && <p className="text-red-700 text-xs pb-
1">{error}</p>}

                  <button
                    type="submit"
                    className="btn-primary"
                    data-testid="login-button"
                  >
                    LOGIN
                  </button>

                  <p className="text-xs text-slate-500 text-center my-4">Or</p>

                  <button
                    type="submit"
                    className="btn-primary btn-light"
                    onClick={() => {
                      navigate("/signup");
                    }}
                  >
                    CREATE ACCOUNT
                  </button>
                </form>
              </div>
```

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

export default Login;
```

SignUp.jsx:

```
import React, { useState } from "react";
import PasswordInput from "../../components/Input/PasswordInput";
import { useNavigate } from "react-router-dom";
import { validateEmail } from "../../../utils/helper";
import { axiosInstance } from "../../../utils/axiosInstance";

const SignUp = () => {
  const [name, setName] = useState("");
  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");
  const [error, setError] = useState(null);

  const navigate = useNavigate();

  const handleSignUp = async (e) => {
    e.preventDefault();

    if (!name) {
      setError("Please enter your name");
      return;
    }

    if (!validateEmail(email)) {
      setError("Please enter a valid email address.");
      return;
    }

    if (!password) {
      setError("Please enter the password");
      return;
    }

    setError("");

    try {
      const response = await axiosInstance.post("/api/auth/signup", {
        username: name,
        email: email,
        password: password,
      });

      if (response.data && response.data.success) {
        navigate("/dashboard");
      }
    } catch (error) {
      if (
        error.response &&
        error.response.data &&
        error.response.data.message
      ) {
        setError(error.response.data.message);
      } else {
        setError("An unexpected error occurred. Please try again.");
      }
    }
  };
```

```
        return (
          <div className="h-screen bg-cyan-50 overflow-hidden relative">
            <div className="login-ui-box bg-orange-200 right-10 -top-40" />
            <div className="login-ui-box" />

            <div className="container h-screen flex items-center justify-center
px-20 mx-auto">
              <div className="w-5/12 h-[90vh] flex flex-col justify-end bg-
signup-bg-image bg-cover bg-center rounded-lg p-10 z-50">
                <h4 className="text-5xl text-white font-semibold leading-
[58px]">
                  Join the Journey
                </h4>
                <p className="text-[15px] text-white leading-6 pr-7 mt-4">
                  Sign up now to begin capturing your travel experiences and
                  cherishing your adventures in a personalized travel story app.
                </p>
              </div>

              <div className="w-2/4 h-[75vh] bg-white rounded-r-lg relative p-16
shadow-lg shadow-cyan-200/20">
                <form onSubmit={handleSignUp}>
                  <h4 className="text-2xl font-semibold mb-7">Sign Up</h4>

                  <input
                    type="text"
                    placeholder="Name"
                    className="input-box"
                    value={name}
                    onChange={({ target }) => {
                      setName(target.value);
                    }}
                  />

                  <input
                    type="text"
                    placeholder="Email"
                    className="input-box"
                    value={email}
                    onChange={({ target }) => {
                      setEmail(target.value);
                    }}
                  />

                  <PasswordInput
                    value={password}
                    onChange={({ target }) => {
                      setPassword(target.value);
                    }}
                  />

                  {error && <p className="text-red-700 text-xs pb-
1">{error}</p>}

                  <button type="submit" className="btn-primary">
                    CREATE ACCOUNT
                  </button>

                  <p className="text-xs text-slate-500 text-center my-4">Or</p>

                  <button
                    type="submit"
                    className="btn-primary btn-light"
                    onClick={() => {
                      navigate("/login");
```

```
                }}
              >
                LOGIN
              </button>
            </form>
          </div>
        </div>
      </div>
    );
  };

  export default SignUp;
```

AddEditTravelStory.jsx:

```
  import React, { useState, useEffect } from "react";
  import { MdAdd, MdDeleteOutline, MdUpdate, MdClose } from "react-
icons/md";
  import DateSelector from "../../components/Input/DateSelector";
  import ImageSelector from "../../components/Input/ImageSelector";
  import TagInput from "../../components/Input/TagInput";
  import { toast } from "react-toastify";
  import { axiosInstance } from "../../../utils/axiosInstance";
  import moment from "moment";

  const AddEditTravelStory = ({
    storyInfo,
    type,
    onClose,
    getAllTravelStories,
  }) => {
    const [title, setTitle] = useState(storyInfo?.title || "");
    const [storyImg, setStoryImg] = useState(storyInfo?.imageUrl || null);
    const [story, setStory] = useState(storyInfo?.story || "");

    const [locationName, setLocationName] = useState(
      storyInfo?.locationName || ""
    );

    const [locationTags, setLocationTags] = useState(
      storyInfo?.locationName ? [storyInfo.locationName] : []
    );

    const [tags, setTags] = useState(
      storyInfo?.tags
        ? Array.isArray(storyInfo.tags)
          ? storyInfo.tags.map((tag) =>
              typeof tag === "object" ? tag.name : tag
            )
          : []
        : []
    );

    const [visitedDate, setVisitedDate] = useState(
      storyInfo?.visitedDate || null
    );

    const [error, setError] = useState("");

    useEffect(() => {
      if (locationTags.length > 0) {
        setLocationName(locationTags[0]);
      } else {
        setLocationName("");
      }
```

```
  }, [locationTags]);

  const uploadImage = async (file) => {
    const formData = new FormData();
    formData.append("image", file);
    try {
      const response = await axiosInstance.post(
        "/api/travel-stories/image-upload",
        formData
      );
      return response.data.imageUrl;
    } catch (error) {
      console.error("Failed to upload image", error);
      throw error;
    }
  };

  const addNewTravelStory = async () => {
    try {
      let imageUrl = storyImg;
      if (storyImg && typeof storyImg === "object") {
        imageUrl = await uploadImage(storyImg);
      }

      const response = await axiosInstance.post("/api/travel-stories/add",
{
        title,
        story,
        imageUrl,
        locationName,
        visitedDate: visitedDate
          ? moment(visitedDate).valueOf()
          : moment().valueOf(),
        tags,
      });

      if (response.data && response.data.story) {
        toast.success("Story Added Successfully");
        getAllTravelStories();
        onClose();
      }
    } catch (error) {
      const errorMessage =
        error.response?.data?.message ||
        error.message ||
        "An unexpected error occurred. Please try again.";
      console.error("Error adding story:", errorMessage);
      setError(errorMessage);
    }
  };

  const updateTravelStory = async () => {
    try {
      let imageUrl = storyInfo.imageUrl;

      if (storyImg && typeof storyImg === "object") {
        imageUrl = await uploadImage(storyImg);
      }

      const storyId = storyInfo._id;
      const response = await axiosInstance.put(
        `/api/travel-stories/edit/${storyId}`,
        {
          title,
          story,
          imageUrl,
```

```
        locationName,
        visitedDate: visitedDate
          ? moment(visitedDate).valueOf()
          : moment().valueOf(),
        tags,
      }
    );

    if (response.data && response.data.story) {
      toast.success("Story Updated Successfully");
      getAllTravelStories();
      onClose();
    }
  } catch (error) {
    const errorMessage =
      error.response?.data?.message ||
      error.message ||
      "An unexpected error occurred. Please try again.";
    console.error("Error updating story:", errorMessage);
    setError(errorMessage);
  }
};

const handleAddOrUpdateClick = () => {
  if (!title) {
    setError("Please enter the title!");
    return;
  }

  if (!story) {
    setError("Please enter the story!");
    return;
  }

  if (!locationName) {
    setError("Please enter the location!");
    return;
  }

  setError("");

  if (type === "edit") {
    updateTravelStory();
  } else {
    addNewTravelStory();
  }
};

const handleDeleteStoryImg = async () => {
  try {
    const deleteImgRes = await axiosInstance.delete(
      "/api/travel-stories/delete-image",
      {
        params: {
          imageUrl: storyInfo.imageUrl,
        },
      }
    );

    if (deleteImgRes.data) {
      const storyId = storyInfo._id;
      const postData = {
        title,
        story,
        locationName,
        visitedDate: moment().valueOf(),
```

```
            imageUrl: "",
          };

          await axiosInstance.put(
            `/api/travel-stories/edit/${storyId}`,
            postData
          );
          setStoryImg(null);
        }
      } catch (error) {
        const errorMessage =
          error.response?.data?.message ||
          error.message ||
          "An unexpected error occurred. Please try again.";
        console.error("Error deleting the image:", errorMessage);
        setError(errorMessage);
      }
    };

    return (
      <div className="relative">
        <div className="flex items-center justify-between">
          <h5 className="text-xl font-medium text-slate-700">
            {type === "add" ? "Add Story" : "Update Story"}
          </h5>

          <div>
            <div className="flex items-center gap-3 p-2 rounded-l-lg">
              {type === "add" ? (
                <button
                  data-testid="save-story-button"
                  className="btn-small"
                  onClick={handleAddOrUpdateClick}
                >
                  <MdAdd className="text-lg" /> ADD STORY
                </button>
              ) : (
                <button
                  data-testid="save-story-button"
                  className="btn-small"
                  onClick={handleAddOrUpdateClick}
                >
                  <MdUpdate className="text-lg" /> Update Story
                </button>
              )}
              <button className="" onClick={onClose}>
                <MdClose className="text-xl text-slate-400" />
              </button>
            </div>

            {error && (
              <p className="text-red-500 text-xs pt-2 text-
right">{error}</p>
            )}
          </div>
        </div>

        <div>
          <div className="flex-1 flex flex-col gap-2 pt-4">
            <label className="input-label">TITLE</label>
            <input
              type="text"
              name="title"
              className="text-2xl text-slate-900 outline-none"
              placeholder="A Journey to the Past"
              value={title}
```

```
                onChange={({ target }) => setTitle(target.value)}
              />
              <div classNam="my-3">
                <DateSelector date={visitedDate} setDate={setVisitedDate} />
              </div>
              <ImageSelector
                image={storyImg}
                setImage={setStoryImg}
                handleDeleteImg={handleDeleteStoryImg}
              />
              <label className="input-label mt-3">TAGS</label>
              <TagInput tags={tags} setTags={setTags} inputType="hashtag" />
              <div className="flex flex-col gap-2 mt-4">
                <label className="input-label">STORY</label>
                <textarea
                  type="text"
                  className="text-sm text-slate-950 outline-none bg-slate-50
p-2 rounded"
                  placeholder="Your Story"
                  rows={10}
                  value={story}
                  onChange={({ target }) => setStory(target.value)}
                />
              </div>
              <div className="pt-3">
                <label className="input-label">VISITED LOCATIONS</label>
                <TagInput
                  tags={locationTags}
                  setTags={setLocationTags}
                  inputType="location"
                  maxTags={1}
                />
              </div>
            </div>
          </div>
        </div>
      );
    };

    export default AddEditTravelStory;
```

Home.jsx:

```
    import React, { useState, useEffect } from "react";
    import Navbar from "../../components/Navbar.jsx";
    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.jsx";
    import AddEditTravelStory from "./AddEditTravelStory";
    import { ToastContainer, toast } from "react-toastify";
    import ViewTravelStory from "./ViewTravelStory.jsx";
    import EmptyCard from "../../components/Cards/EmptyCard.jsx";
    import { getEmptyCardImg, getEmptyCardMessage } from
"../../../utils/helper.js";
    import { DayPicker } from "react-day-picker";
    import moment from "moment";
    import FilterInfoTitle from "../../components/Cards/FilterInfoTitle.jsx";
    import "react-toastify/dist/ReactToastify.css";

    const Home = () => {
      const navigate = useNavigate();

      const [userInfo, setUserInfo] = useState(null);
```

```
        const [allStories, setAllStories] = useState([]);

        const [searchQuery, setSearchQuery] = useState("");
        const [filterType, setFilterType] = useState("");

        const [dateRange, setDateRange] = useState({ from: null, to: null });

        const [openAddEditModal, setOpenAddEditModal] = useState({
          isShown: false,
          type: "add",
          data: null,
        });

        const [openViewModal, setOpenViewModal] = useState({
          isShown: false,
          data: null,
        });

        const getUserInfo = async () => {
          try {
            const response = await axiosInstance.get("/api/auth/check-auth");
            if (response.data && response.data.user) {
              setUserInfo(response.data.user);
            }
          } catch (error) {
            if (error.response && error.response.status === 401) {
              navigate("/login");
            }
          }
        };

        const getAllTravelStories = async () => {
          try {
            const response = await axiosInstance.get("/api/travel-stories/get-
all");
            if (response.data && response.data.stories) {
              setAllStories(response.data.stories);
            }
          } catch (error) {
            console.log("An unexpected error occurred. Please try again.");
          }
        };

        const getFilteredStories = () => {
          let filteredStories = allStories;

          if (showFavoritesOnly) {
            filteredStories = filteredStories.filter((story) =>
story.isFavourite);
          }

          if (dateRange.from && dateRange.to) {
            const start = moment(dateRange.from).startOf("day").valueOf();
            const end = moment(dateRange.to).endOf("day").valueOf();
            filteredStories = filteredStories.filter((story) =>
              moment(story.visitedDate).isBetween(start, end, undefined, "[]")
            );
          }

          return filteredStories;
        };

        const handleEdit = (data) => {
          setOpenAddEditModal({ isShown: true, type: "edit", data: data });
        };
```

```
        const handleViewStory = (data) => {
          setOpenViewModal({ isShown: true, data });
        };

        const updateIsFavourite = async (storyData) => {
          const storyId = storyData._id;

          try {
            const response = await axiosInstance.put(
              `/api/travel-stories/update-is-favourite/${storyId}`,
              { isFavourite: !storyData.isFavourite }
            );

            if (response.data && response.data.story) {
              toast.success("Story Updated Successfully");
              getAllTravelStories();
            }
          } catch (error) {
            console.error("An unexpected error occurred:", error);
            toast.error("Failed to update the story. Please try again.");
          }
        };

        const deleteTravelStory = async (data) => {
          const storyId = data._id;

          try {
            const response = await axiosInstance.delete(
              `/api/travel-stories/delete/${storyId}`
            );

            if (response.data && !response.data.error) {
              toast.success("Story Deleted Successfully");
              setOpenViewModal((prevState) => ({ ...prevState, isShown: false
}));
              getAllTravelStories();
            }
          } catch (error) {
            console.error("An unexpected error occurred:", error);
            toast.error("Failed to delete the story. Please try again.");
          }
        };

        const onSearchStory = async (query) => {
          try {
            const response = await axiosInstance.get("/api/travel-
stories/search", {
              params: {
                query,
              },
            });

            if (response.data && response.data.stories) {
              setFilterType("search");
              setAllStories(response.data.stories);
            }
          } catch (error) {
            console.log("An unexpected error occurred. Please try again.");
          }
        };

        const handleClearSearch = () => {
          setFilterType("");
          getAllTravelStories();
        };
```

```
        const filterStoriesByDate = async (day) => {
          try {
            const startDate = day.from ? moment(day.from).valueOf() : null;
            const endDate = day.to ? moment(day.to).valueOf() : null;

            if (startDate && endDate) {
              const response = await axiosInstance.get("/api/travel-
stories/filter", {
                params: { startDate, endDate },
              });

              if (response.data && response.data.stories) {
                setFilterType("date");
                setAllStories(response.data.stories);
              }
            }
          } catch (error) {
            console.log("An unexpected error occurred. Please try again.");
          }
        };

        const handleDayClick = (day) => {
          setDateRange(day);
          filterStoriesByDate(day);
        };

        const resetFilter = () => {
          setDateRange({ from: null, to: null });
          setFilterType("");
          getAllTravelStories();
        };

        useEffect(() => {
          getUserInfo();
          getAllTravelStories();

          return () => {};
        }, []);

        return (
          <>
            <Navbar
              userInfo={userInfo}
              searchQuery={searchQuery}
              setSearchQuery={setSearchQuery}
              onSearchNote={onSearchStory}
              handleClearSearch={handleClearSearch}
            />

            <div className="container mx-auto py-10 px-5">
              <FilterInfoTitle
                filterType={filterType}
                filterDates={dateRange}
                onClear={() => {
                  resetFilter();
                }}
              />

              <div className="flex gap-10">
                <div className="flex-1 pr-5">
                  {allStories.length > 0 ? (
                    <div className="grid grid-cols-2 gap-4">
                      {allStories.map((item) => {
                        return (
                          <TravelStoryCard
                            key={item._id}
```

```
                            imgUrl={item.imageUrl}
                            title={item.title}
                            story={item.story}
                            date={item.visitedDate}
                            location={item.location}
                            tags={item.tags}
                            isFavourite={item.isFavourite}
                            onClick={() => handleViewStory(item)}
                            onFavouriteClick={() => updateIsFavourite(item)}
                          />
                        );
                      })}
                    </div>
                  ) : (
                    <EmptyCard
                      imgSrc={getEmptyCardImg(filterType)}
                      message={getEmptyCardMessage(filterType)}
                    />
                  )}
                </div>

                <div className="w-[350px] pl-5">
                  <div className="bg-white border border-slate-200 shadow-lg
shadow-slate-200/60 rounded-lg">
                    <div className="p-3">
                      <DayPicker
                        captionLayout="dropdown-buttons"
                        mode="range"
                        selected={dateRange}
                        onSelect={handleDayClick}
                        pagedNavigation
                      />
                    </div>
                  </div>
                </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,
                },
              }}
              appElement={document.getElementById("root")}
              className="model-box"
              shouldCloseOnOverlayClick={true}
              shouldCloseOnEsc={true}
            >
              <AddEditTravelStory
                type={openAddEditModal.type}
                storyInfo={openAddEditModal.data}
                onClose={() => {
                  setOpenAddEditModal({ isShown: false, type: "add", data: null
});

                }}
                getAllTravelStories={getAllTravelStories}
              />
            </Modal>

            <Modal
```

```
            isOpen={openViewModal.isShown}
            onRequestClose={() => {}}
            style={{
              overlay: {
                backgroundColor: "rgba(0,0,0,0.2)",
                zIndex: 999,
              },
            }}
            appElement={document.getElementById("root")}
            className="model-box"
            shouldCloseOnOverlayClick={true}
            shouldCloseOnEsc={true}
          >
            <ViewTravelStory
              storyInfo={openViewModal.data || null}
              onClose={() => {
                setOpenViewModal((prevState) => ({ ...prevState, isShown:
false }));
              }}
              onEditClick={() => {
                setOpenViewModal((prevState) => ({ ...prevState, isShown:
false }));
                handleEdit(openViewModal.data || null);
              }}
              onDeleteClick={() => {
                deleteTravelStory(openViewModal.data || null);
              }}
            />
          </Modal>

          <button
            data-testid="add-story-button"
            className="w-16 h-16 flex items-center justify-center rounded-full
hover:bg-opacity-80 fixed right-10 bottom-10"
            style={{ backgroundColor: "#FFA07A" }}
            onClick={() => {
              setOpenAddEditModal({ isShown: true, type: "add", data: null });
            }}
          >
            <MdAdd className="text-[32px] text-white" />
          </button>

          <ToastContainer />
        </>
      );
    };

    export default Home;
```

ViewTravelStory.jsx:

```
    import React, { useRef, useState } from "react";
    import { GrMapLocation } from "react-icons/gr";
    import {
      MdDeleteOutline,
      MdUpdate,
      MdClose,
      MdPictureAsPdf,
    } from "react-icons/md";
    import moment from "moment";
    import jsPDF from "jspdf";

    const ViewTravelStory = ({
      storyInfo,
      onClose,
```

```
  onEditClick,
  onDeleteClick,
}) => {
  const contentRef = useRef(null);
  const [showDeleteConfirm, setShowDeleteConfirm] = useState(false);

  const exportToPDF = async () => {
    try {
      const pdf = new jsPDF("p", "mm", "a4");
      const pageWidth = pdf.internal.pageSize.getWidth();
      const margin = 20;
      const contentWidth = pageWidth - margin * 2;
      let yPos = margin;

      pdf.setFont("helvetica", "bold");
      pdf.setFontSize(18);
      pdf.text(storyInfo?.title || "Travel Story", margin, yPos);
      yPos += 10;

      pdf.setFont("helvetica", "normal");
      pdf.setFontSize(12);
      pdf.setTextColor(100, 100, 100);
      const dateText = storyInfo?.visitedDate
        ? moment(storyInfo.visitedDate).format("Do MMM YYYY")
        : "-";
      pdf.text(`Date: ${dateText}`, margin, yPos);
      yPos += 8;

      if (storyInfo?.location) {
        const locationText =
          typeof storyInfo.location === "object"
            ? storyInfo.location.name
            : storyInfo.location;
        pdf.text(`Location: ${locationText}`, margin, yPos);
        yPos += 8;
      }

      if (storyInfo?.imageUrl) {
        const img = new Image();
        img.src = storyInfo.imageUrl;

        await new Promise((resolve) => {
          img.onload = () => {
            const imgRatio = img.width / img.height;
            const imgWidth = contentWidth;
            const imgHeight = imgWidth / imgRatio;

            pdf.addImage(
              storyInfo.imageUrl,
              "JPEG",
              margin,
              yPos,
              imgWidth,
              imgHeight
            );
            yPos += imgHeight + 10;
            resolve();
          };
          img.onerror = () => {
            console.warn("Could not load image for PDF");
            resolve();
          };
        });
      }

      if (storyInfo?.tags?.length > 0) {
```

```
            const formattedTags = storyInfo.tags
              .map(
                (tag) =>
                  `#${typeof tag === "object" && tag.name ? tag.name :
String(tag)}`
              )
              .join(", ");
            pdf.text(`Tags: ${formattedTags}`, margin, yPos);
            yPos += 10;
          }

          pdf.setTextColor(0, 0, 0);
          pdf.setFontSize(12);
          pdf.text("Story:", margin, yPos);
          yPos += 6;

          const storyText = storyInfo?.story || "";
          const splitText = pdf.splitTextToSize(storyText, contentWidth);

          if (
            yPos + splitText.length * 5 >
            pdf.internal.pageSize.getHeight() - margin
          ) {
            pdf.addPage();
            yPos = margin;
          }
          pdf.text(splitText, margin, yPos);

          const filename = storyInfo?.title
            ? `${storyInfo.title.replace(/\s+/g, "_")}_travel_story.pdf`
            : "travel_story.pdf";

          pdf.save(filename);
        } catch (error) {
          console.error("Error generating PDF:", error);
        }
      };

      const handleDeleteClick = () => {
        setShowDeleteConfirm(true);
      };

      const confirmDelete = () => {
        onDeleteClick();
        setShowDeleteConfirm(false);
      };

      const cancelDelete = () => {
        setShowDeleteConfirm(false);
      };

      return (
        <div className="relative story-item">
          <div className="flex items-center justify-end">
            <div className="flex items-center bg-cyan-50/50 gap-3 p-2 rounded-
l-lg">
              <button className="btn-small" onClick={exportToPDF}>
                <MdPictureAsPdf className="text-lg" /> Export as PDF
              </button>
              <button
                data-testid="edit-story-button"
                className="btn-small"
                onClick={onEditClick}
              >
                <MdUpdate className="text-lg" /> Update Story
              </button>
```

```
                <button
                  data-testid="delete-story-button"
                  className="btn-small btn-delete"
                  onClick={handleDeleteClick}
                >
                  <MdDeleteOutline className="text-lg" /> Delete
                </button>
                <button onClick={onClose}>
                  <MdClose className="text-xl text-slate-400" />
                </button>
              </div>
            </div>

            {showDeleteConfirm && (
              <div className="fixed inset-0 bg-black bg-opacity-50 flex items-
center justify-center z-50">
                <div className="bg-white p-6 rounded-lg shadow-lg max-w-md w-
full">
                  <h3 className="text-xl font-medium mb-4">Delete Story</h3>
                  <p className="mb-6">
                    Are you sure you want to delete this story? This action
cannot be
                    undone.
                  </p>
                  <div className="flex justify-end gap-4">
                    <button
                      className="px-4 py-2 bg-gray-200 rounded"
                      onClick={cancelDelete}
                    >
                      Cancel
                    </button>
                    <button
                      data-testid="confirm-delete-button"
                      className="px-4 py-2 bg-red-500 text-white rounded"
                      onClick={confirmDelete}
                    >
                      Delete
                    </button>
                  </div>
                </div>
              </div>
            )}

            <div
              ref={contentRef}
              className="flex-1 flex flex-col gap-2 py-4 bg-white p-4 rounded-
lg"
            >
              <h1 className="text-2xl text-slate-950">{storyInfo?.title}</h1>
              <div className="flex items-center justify-between gap-3">
                <span className="text-xs text-slate-500">
                  {storyInfo && moment(storyInfo.visitedDate).format("Do MMM
YYYY")}
                </span>
                {storyInfo?.location && (
                  <div className="inline-flex items-center gap-2 text-[13px]
text-orange-600 bg-orange-100 rounded px-2 py-1">
                    <GrMapLocation className="text-sm" />
                    <span className="block">
                      {typeof storyInfo.location === "object"
                        ? storyInfo.location.name
                        : storyInfo.location}
                    </span>
                  </div>
                )}
              </div>
```

```
            {storyInfo?.imageUrl && (
              <img
                src={storyInfo.imageUrl}
                alt="Selected"
                className="w-full h-[300px] object-cover rounded-lg"
              />
            )}

            <div className="mt-4">
              <span className="text-sm text-slate-500">Story: </span>
              <p className="text-sm text-slate-950 leading-6 text-justify
whitespace-pre-line">
                {storyInfo.story}
              </p>
            </div>

            {storyInfo?.tags?.length > 0 && (
              <div className="mt-4">
                <span className="text-sm text-slate-500">Tags: </span>
                <div className="flex flex-wrap gap-2 mt-2">
                  {storyInfo.tags.map((tag, index) => (
                    <span
                      key={typeof tag === "object" ? tag._id : index}
                      className="text-orange-600 bg-orange-100 text-xs px-2
py-1 rounded-full"
                    >
                      #
                      {typeof tag === "object" && tag.name ? tag.name :
String(tag)}
                    </span>
                  ))}
                </div>
              </div>
            )}
          </div>
        </div>
      );
    };

    export default ViewTravelStory;
```

Wishlist.jsx:

```
    import React, { useEffect, useState } from "react";
    import { axiosInstance } from "../../../utils/axiosInstance";
    import "../../../src/pages/Wishlist/Wishlist.css";
    import Navbar from "../../components/Navbar";
    import ImageSelector from "../../components/Input/ImageSelector";

    const Wishlist = () => {
      const [wishlist, setWishlist] = useState([]);
      const [error, setError] = useState(null);
      const [userInfo, setUserInfo] = useState(null);
      const [isAdding, setIsAdding] = useState(false);
      const [placeName, setPlaceName] = useState("");
      const [address, setAddress] = useState("");
      const [rating, setRating] = useState(0);
      const [notes, setNotes] = useState("");
      const [searchQuery, setSearchQuery] = useState("");
      const [loading, setLoading] = useState(false);
      const [image, setImage] = useState(null);
      const [sortOption, setSortOption] = useState("name");

      const [isEditing, setIsEditing] = useState(false);
```

```
const [editItemId, setEditItemId] = useState(null);
const [editPlaceName, setEditPlaceName] = useState("");
const [editAddress, setEditAddress] = useState("");
const [editRating, setEditRating] = useState(0);
const [editNotes, setEditNotes] = useState("");
const [editImage, setEditImage] = useState(null);

const fetchUserInfo = async () => {
  try {
    const response = await axiosInstance.get("/api/auth/check-auth");
    setUserInfo(response.data.user);
  } catch (error) {
    console.error("Failed to load user info");
  }
};

const fetchWishlist = async () => {
  setLoading(true);
  try {
    const response = await axiosInstance.get("/api/wishlist");
    const formattedWishlist = response.data.wishlist.map((item) => ({
      ...item,
      placeName:
        item.locationName ||
        (item.locationId && item.locationId.name
          ? item.locationId.name
          : "Unnamed Place"),
    }));
    setWishlist(formattedWishlist);
  } catch (err) {
    setError("Failed to load wishlist items");
  } finally {
    setLoading(false);
  }
};

useEffect(() => {
  fetchUserInfo();
  fetchWishlist();
}, []);

const uploadImage = async (file) => {
  const formData = new FormData();
  formData.append("image", file);
  try {
    const response = await axiosInstance.post(
      "/api/wishlist/image-upload",
      formData
    );
    return response.data.imageUrl;
  } catch (error) {
    console.error("Failed to upload image", error);
    throw error;
  }
};

const addWishlistItem = async (e) => {
  e.preventDefault();
  try {
    let imageUrl = null;

    if (image && typeof image === "object") {
      imageUrl = await uploadImage(image);
    }
    const response = await axiosInstance.post("/api/wishlist", {
      locationName: placeName,
```

```
      address,
      rating,
      notes,
      imageUrl,
    });

    const newItem = {
      ...response.data.wishlistItem,
      placeName: response.data.wishlistItem.locationName,
    };

    setWishlist((prev) => [...prev, newItem]);

    fetchWishlist();

    setIsAdding(false);
    setPlaceName("");
    setAddress("");
    setRating(0);
    setNotes("");
    setImage(null);
  } catch (error) {
    console.error(error);
    setError("Failed to add wishlist item");
  }
};

const deleteWishlistItem = async (id) => {
  try {
    await axiosInstance.delete(`/api/wishlist/${id}`);
    setWishlist((prev) => prev.filter((item) => item._id !== id));
  } catch (error) {
    setError("Failed to delete wishlist item");
  }
};

const startEditing = (item) => {
  setIsEditing(true);
  setEditItemId(item._id);
  setEditPlaceName(item.locationName || item.placeName || "");
  setEditAddress(item.locationId?.address || "");
  setEditRating(item.rating || 0);
  setEditNotes(item.notes || "");
  setEditImage(item.imageUrl || null);
};

const handleDeleteImage = async (imageUrl) => {
  try {
    await axiosInstance.delete("/api/wishlist/delete-image", {
      params: { imageUrl },
    });
    return "";
  } catch (error) {
    console.error("Error deleting image:", error);
    setError("Failed to delete image");
    return imageUrl;
  }
};

const updateWishlistItem = async (e) => {
  e.preventDefault();
  try {
    let imageUrl = editImage;

    if (editImage && typeof editImage === "object") {
      imageUrl = await uploadImage(editImage);
```

```
            }

            const response = await
axiosInstance.put(`/api/wishlist/${editItemId}`, {
              locationName: editPlaceName,
              address: editAddress,
              rating: editRating,
              notes: editNotes,
              imageUrl,
            });

            fetchWishlist();

            setIsEditing(false);
            setEditItemId(null);
            setEditPlaceName("");
            setEditAddress("");
            setEditRating(0);
            setEditNotes("");
            setEditImage(null);
          } catch (error) {
            console.error(error);
            setError("Failed to update wishlist item");
          }
        };

        const renderError = () => {
          if (error) {
            return <div className="error-message">{error}</div>;
          }
          return null;
        };

        const filteredWishlist = wishlist.filter((item) =>
          (item.placeName ||
"").toLowerCase().includes(searchQuery.toLowerCase())
        );

        const sortedWishlist = [...filteredWishlist].sort((a, b) => {
          if (sortOption === "name") {
            return (a.placeName || "").localeCompare(b.placeName || "");
          } else if (sortOption === "rating") {
            return (b.rating || 0) - (a.rating || 0);
          }
          return 0;
        });

        return (
          <div className="wishlist-page">
            {userInfo && (
              <header className="w-full fixed top-0 z-10">
                <Navbar
                  userInfo={userInfo}
                  searchQuery={searchQuery}
                  setSearchQuery={setSearchQuery}
                  onSearchNote={() => {}}
                  handleClearSearch={() => setSearchQuery("")}
                />
              </header>
            )}
            <div className="wishlist-container pt-[80px]">
              <h2 className="wishlist-title">Your Wishlist</h2>

              {renderError()}

              {loading ? (
```

```
                <p>Loading...</p>
              ) : (
                <>
                  <div className="wishlist-controls">
                    <button
                      className="add-item-btn"
                      onClick={() => setIsAdding(true)}
                    >
                      + Add New Place
                    </button>
                    <select
                      className="sort-select"
                      value={sortOption}
                      onChange={(e) => setSortOption(e.target.value)}
                    >
                      <option value="name">Sort by Name</option>
                      <option value="rating">Sort by Rating</option>
                    </select>
                  </div>

                  {isAdding && (
                    <div className="wishlist-modal">
                      <div className="wishlist-modal-content">
                        <h3>Add New Place</h3>
                        <form onSubmit={addWishlistItem} className="wishlist-
form">
                          <input
                            type="text"
                            placeholder="Place Name"
                            value={placeName}
                            onChange={(e) => setPlaceName(e.target.value)}
                            required
                          />
                          <input
                            type="text"
                            placeholder="Address"
                            value={address}
                            onChange={(e) => setAddress(e.target.value)}
                            required
                          />
                          <input
                            type="number"
                            placeholder="Rating (1-5)"
                            value={rating}
                            onChange={(e) => setRating(e.target.value)}
                            min="1"
                            max="5"
                            required
                          />
                          <textarea
                            placeholder="Notes"
                            value={notes}
                            onChange={(e) => setNotes(e.target.value)}
                          />
                          <div className="mb-4">
                            <ImageSelector
                              image={image}
                              setImage={setImage}
                              handleDeleteImg={() => setImage(null)}
                            />
                          </div>
                          <div className="form-buttons">
                            <button type="submit" className="submit-btn">
                              Add
                            </button>
                            <button
```

```
                        type="button"
                        className="cancel-btn"
                        onClick={() => {
                          setIsAdding(false);
                          setImage(null);
                        }}
                      >
                        Cancel
                      </button>
                    </div>
                  </form>
                </div>
              </div>
            )}

            {isEditing && (
              <div className="wishlist-modal">
                <div className="wishlist-modal-content">
                  <h3>Edit Place</h3>
                  <form onSubmit={updateWishlistItem} className="wishlist-
form">
                    <input
                      type="text"
                      placeholder="Place Name"
                      value={editPlaceName}
                      onChange={(e) => setEditPlaceName(e.target.value)}
                      required
                    />
                    <input
                      type="text"
                      placeholder="Address"
                      value={editAddress}
                      onChange={(e) => setEditAddress(e.target.value)}
                      required
                    />
                    <input
                      type="number"
                      placeholder="Rating (1-5)"
                      value={editRating}
                      onChange={(e) => setEditRating(e.target.value)}
                      min="1"
                      max="5"
                      required
                    />
                    <textarea
                      placeholder="Notes"
                      value={editNotes}
                      onChange={(e) => setEditNotes(e.target.value)}
                    />
                    <div className="mb-4">
                      <ImageSelector
                        image={editImage}
                        setImage={setEditImage}
                        handleDeleteImg={() => {
                          if (typeof editImage === "string") {
                            handleDeleteImage(editImage);
                          }
                          setEditImage(null);
                        }}
                      />
                    </div>
                    <div className="form-buttons">
                      <button type="submit" className="submit-btn">
                        Update
                      </button>
                      <button
```

```
                            type="button"
                            className="cancel-btn"
                            onClick={() => {
                              setIsEditing(false);
                              setEditImage(null);
                            }}
                          >
                            Cancel
                          </button>
                        </div>
                      </form>
                    </div>
                  </div>
                )}

                {sortedWishlist.length === 0 ? (
                  <p className="empty-message">No items in wishlist.</p>
                ) : (
                  <ul className="wishlist-list">
                    {sortedWishlist.map((item) => (
                      <li key={item._id} className="wishlist-item">
                        {item.imageUrl && (
                          <div className="item-image-container">
                            <img
                              src={item.imageUrl}
                              alt={item.placeName || item.locationName ||
"Place"}
                              className="item-image"
                            />
                          </div>
                        )}
                        <h3 className="item-title">
                          {item.locationName || item.placeName || "Unnamed
Place"}
                        </h3>
                        <p className="item-address">
                          Address: {item.locationId?.address || "N/A"}{" "}
                          {/* Відображаємо адресу */}
                        </p>
                        <p className="item-rating">
                          Rating: {item.rating || "N/A"}
                        </p>
                        <p className="item-notes">
                          Notes: {item.notes || "No notes added"}
                        </p>
                        <button
                          onClick={() => startEditing(item)}
                          className="edit-btn"
                          data-testid="edit-wishlist-item"
                        >
                          Edit
                        </button>
                        <button
                          onClick={() => deleteWishlistItem(item._id)}
                          className="delete-btn"
                        >
                          Remove
                        </button>
                      </li>
                    ))}
                  </ul>
                )}
              </>
            )}
          </div>
        </div>
```

```
  );
};

export default Wishlist;
```

App.jsx:

```
import {
  BrowserRouter as Router,
  Routes,
  Route,
  Navigate,
} from "react-router-dom";
import React, { useEffect, useState } from "react";

import Login from "./pages/Auth/Login";
import SignUp from "./pages/Auth/SignUp";
import Home from "./pages/Home/Home";
import UserProfile from "./pages/UserProfile/UserProfile";
import Wishlist from "./pages/Wishlist/Wishlist";
import { axiosInstance } from "../utils/axiosInstance";

const App = () => {
  const [isAuthenticated, setIsAuthenticated] = useState(null);

  useEffect(() => {
    const checkAuth = async () => {
      try {
        const response = await axiosInstance.get("/api/auth/check-auth");
        setIsAuthenticated(response.data.success);
      } catch (error) {
        setIsAuthenticated(false);
      }
    };
    checkAuth();
  }, []);

  if (isAuthenticated === null) return <div>Loading...</div>;

  return (
    <div>
      <Router>
        <Routes>
          <Route
            path="/"
            element={<Root isAuthenticated={isAuthenticated} />}
          />
          <Route path="/login" element={<Login />} />
          <Route path="/signUp" element={<SignUp />} />
          <Route
            path="/dashboard"
            element={
              <ProtectedRoute isAuthenticated={isAuthenticated}>
                <Home />
              </ProtectedRoute>
            }
          />
          <Route
            path="/wishlist"
            element={
              <ProtectedRoute isAuthenticated={isAuthenticated}>
                <Wishlist />
              </ProtectedRoute>
            }
          />
```

```
          <Route
            path="/profile"
            element={
              <ProtectedRoute isAuthenticated={isAuthenticated}>
                <UserProfile />
              </ProtectedRoute>
            }
          />
        </Routes>
      </Router>
    </div>
  );
};

const ProtectedRoute = ({ isAuthenticated, children }) => {
  return isAuthenticated ? children : <Navigate to="/login" />;
};

const Root = ({ isAuthenticated }) => {
  return isAuthenticated ? (
    <Navigate to="/dashboard" />
  ) : (
    <Navigate to="/login" />
  );
};

export default App;
```

UserProfile.jsx:

```
import React, { useEffect, useState } from "react";
import { useNavigate } from "react-router-dom";
import { axiosInstance } from "../../../utils/axiosInstance";
import "./UserProfile.css";

const UserProfile = () => {
  const [userInfo, setUserInfo] = useState(null);
  const [storyCount, setStoryCount] = useState(0);
  const navigate = useNavigate();

  useEffect(() => {
    const fetchUserInfo = async () => {
      try {
        const response = await axiosInstance.get("/api/auth/check-auth");
        if (response.data && response.data.user) {
          setUserInfo(response.data.user);
        }
      } catch (error) {
        console.log("Error fetching user info:", error);
      }
    };

    const fetchStoryCount = async () => {
      try {
        const response = await axiosInstance.get(
          "/api/travel-stories/user-story-count"
        );
        if (response.data && response.data.storyCount !== undefined) {
          setStoryCount(response.data.storyCount);
        }
      } catch (error) {
        console.log("Error fetching story count:", error);
      }
    };
```

```
          fetchUserInfo();
          fetchStoryCount();
        }, []);

        if (!userInfo) return <p>Loading...</p>;

        const memberSinceDate = userInfo.createdAt
          ? new Date(userInfo.createdAt)
          : null;
        const isValidDate =
          memberSinceDate instanceof Date && !isNaN(memberSinceDate);

        return (
          <div className="user-profile-wrapper">
            <button className="back-button" onClick={() => navigate("/")}>
              &#8592; Home
            </button>
            <div className="user-profile-container">
              <div className="flex flex-col items-center">
                <div className="user-profile-avatar">
                  {userInfo.username ? userInfo.username[0] : "U"}
                </div>
                <h2 className="mt-4 text-2xl font-semibold text-gray-800">
                  {userInfo.username}
                </h2>
                <p className="text-gray-500">{userInfo.email}</p>
              </div>

              <div className="user-profile-details">
                <div className="user-profile-detail-item">
                  <span className="font-medium">Username:</span>
                  <span>{userInfo.username || "User123"}</span>
                </div>
                <div className="user-profile-detail-item">
                  <span className="font-medium">Email:</span>
                  <span>{userInfo.email}</span>
                </div>
                <div className="user-profile-detail-item">
                  <span className="font-medium">Member Since:</span>
                  <span>
                    {isValidDate
                      ? memberSinceDate.toLocaleDateString()
                      : "Invalid Date"}
                  </span>
                </div>
                <div className="user-profile-detail-item">
                  <span className="font-medium">Total Stories:</span>
                  <span>{storyCount}</span>
                </div>
              </div>

              <div className="user-profile-buttons">
                <button className="user-profile-button user-profile-button-
edit">
                  Edit Profile
                </button>
                <button className="user-profile-button user-profile-button-
logout">
                  Logout
                </button>
              </div>
            </div>
          </div>
        );
      };
```

```
export default UserProfile;
```

package.json:

```
{
  "name": "frontend",
  "private": true,
  "version": "0.0.0",
  "type": "module",
  "scripts": {
    "dev": "vite",
    "build": "vite build",
    "lint": "eslint .",
    "preview": "vite preview"
  },
  "dependencies": {
    "axios": "^1.7.7",
    "html2canvas": "^1.4.1",
    "html2pdf.js": "^0.10.2",
    "jspdf": "^3.0.0",
    "react": "^18.3.1",
    "react-day-picker": "^9.3.2",
    "react-dom": "^18.3.1",
    "react-icons": "^5.3.0",
    "react-modal": "^3.16.1",
    "react-moment": "^1.1.3",
    "react-router-dom": "^6.27.0",
    "react-toastify": "^10.0.6",
    "tailwind-scrollbar": "^3.1.0"
  },
  "devDependencies": {
    "@eslint/js": "^9.13.0",
    "@types/react": "^18.3.12",
    "@types/react-dom": "^18.3.1",
    "@vitejs/plugin-react": "^4.3.3",
    "autoprefixer": "^10.4.20",
    "cypress": "^14.0.1",
    "cypress-file-upload": "^5.0.8",
    "eslint": "^9.13.0",
    "eslint-plugin-react": "^7.37.2",
    "eslint-plugin-react-hooks": "^5.0.0",
    "eslint-plugin-react-refresh": "^0.4.14",
    "globals": "^15.11.0",
    "postcss": "^8.4.47",
    "tailwindcss": "^3.4.14",
    "vite": "^5.4.10"
  }
}
```

allTests.cy.js:

```
describe("Тести для вебзастосунку", () => {
  const email = `test_${Date.now()}@example.com`;

  const createTestUser = () => {
    cy.request({
      method: "GET",
      url: "http://localhost:8000/api/auth/check-user",
      qs: { email },
    }).then((response) => {
      if (response.body.userExists) {
        cy.request("DELETE", `http://localhost:8000/api/auth/delete-user`,
{
```

```
            email,
          });
        }
      });

      cy.request({
        method: "POST",
        url: "http://localhost:8000/api/auth/signup",
        body: {
          username: "TestUser",
          email,
          password: "password123",
        },
      }).then((response) => {
        expect(response.status).to.eq(201);
        const token = response.body.token;
        window.localStorage.setItem("token", token);
      });
    };

    const removeTestUser = () => {
      cy.request("DELETE", `http://localhost:8000/api/auth/delete-user`, {
        email,
      });
      window.localStorage.removeItem("token");
    };

    const loginUser = () => {
      cy.visit("/login");
      cy.get("input[name='email']").type(email);
      cy.get("input[name='password']").type("password123");
      cy.get("[data-testid='login-button']").click();
      cy.url().should("include", "/dashboard");
    };

    const createStoryViaUI = (title, content, location, tag) => {
      cy.get("[data-testid='add-story-button']").click();
      cy.get("input[name='title']").type(title);
      cy.get("[data-testid='date-selector']").click();
      cy.get(".rdp-day").contains("15").click({ force: true });
      cy.get("input[type='file']").attachFile("test-image.jpg");
      cy.get("input[placeholder='Add Hashtag']").type(`${tag}{enter}`);
      cy.get("textarea").type(content);
      cy.get("input[placeholder='Add
Location']").type(`${location}{enter}`);
      cy.get("[data-testid='save-story-button']").click();
      cy.get(".Toastify__toast--success").should(
        "contain",
        "Story Added Successfully"
      );
      cy.get(".grid").should("contain", title);
    };

    describe("Профіль", () => {
      beforeEach(createTestUser);
      afterEach(removeTestUser);

      it("Успішний вхід", () => {
        loginUser();
      });

      it("Відображення профілю", () => {
        loginUser();
        cy.visit("/profile");
        cy.contains("TestUser").should("be.visible");
      });
```

```
  it("Вихід", () => {
    loginUser();
    cy.get("[data-testid='logout-button']").click();
    cy.url().should("include", "/login");
    cy.get("h4").should("contain", "Login");
  });

  it("Реєстрація з уже існуючим email", () => {
    cy.request({
      method: "POST",
      url: "http://localhost:8000/api/auth/signup",
      body: {
        username: "AnotherUser",
        email,
        password: "password123",
      },
      failOnStatusCode: false,
    }).then((response) => {
      expect(response.status).to.eq(400);
      expect(response.body.message).to.eq("User already exists");
    });
  });

  it("Перевірка авторизації через API", () => {
    cy.request({
      method: "GET",
      url: "http://localhost:8000/api/auth/check-auth",
      headers: {
        Authorization: `Bearer ${window.localStorage.getItem("token")}`,
      },
    }).then((response) => {
      expect(response.status).to.eq(200);
      expect(response.body.user.username).to.eq("TestUser");
      expect(response.body.user.email).to.eq(email);
    });
  });
});

describe("Історії подорожей", () => {
  beforeEach(createTestUser);
  afterEach(removeTestUser);

  it("Отримання історій через API", () => {
    cy.request({
      method: "GET",
      url: "http://localhost:8000/api/travel-stories/get-all",
      headers: {
        Authorization: `Bearer ${window.localStorage.getItem("token")}`,
      },
    }).then((response) => {
      expect(response.status).to.eq(200);
      expect(response.body.stories).to.be.an("array");
    });
  });

  it("Додання історії через API", () => {
    cy.request({
      method: "POST",
      url: "http://localhost:8000/api/travel-stories/add",
      headers: {
        Authorization: `Bearer ${window.localStorage.getItem("token")}`,
      },
      body: {
        title: "Нова історія подорожі",
        story: "Це тестова історія подорожі.",
```

```
            locationName: "Тестове місце",
            visitedDate: Date.now().toString(),
            tags: ["тест", "подорож"],
          },
        }).then((response) => {
          expect(response.status).to.eq(201);
          expect(response.body.story.title).to.eq("Нова історія подорожі");
        });
      });

      it("Додання історії через інтерфейс", () => {
        loginUser();
        createStoryViaUI(
          "Моя подорож до Карпат",
          "Це була незабутня подорож!",
          "Карпати",
          "природа"
        );
      });

      it("Редагування історії", () => {
        loginUser();
        createStoryViaUI(
          "Моя подорож до Львова",
          "Подорож у місто.",
          "Львів",
          "архітектура"
        );
        cy.contains("Моя подорож до Львова").click();
        cy.get("[data-testid='edit-story-button']").click();
        cy.get("input[name='title']").clear().type("Оновлена подорож до
Львова");
        cy.get("textarea").clear().type("Оновлена історія про Львів.");
        cy.get("[data-testid='tag']").each(($tag) => {
          cy.wrap($tag)
            .find("[data-testid='remove-tag-button']")
            .click({ force: true });
        });
        cy.get("input[placeholder='Add Hashtag']").type("історія{enter}");
        cy.get("input[placeholder='Add Location']")
          .clear()
          .type("Львів, Україна{enter}");
        cy.get("[data-testid='save-story-button']").click();
        cy.get(".Toastify__toast--success").should(
          "contain",
          "Story Updated Successfully"
        );
      });

      it("Видалення історії", () => {
        loginUser();
        createStoryViaUI(
          "Тестова подорож для видалення",
          "Для видалення",
          "Тест",
          "тест"
        );
        cy.contains("Тестова подорож для видалення").click();
        cy.intercept("DELETE", "/api/travel-
stories/delete/*").as("deleteStory");
        cy.get("[data-testid='delete-story-button']").click({ force: true
});
        cy.get("body").then((body) => {
          if (body.find("[data-testid='confirm-delete-button']").length > 0)
{
            cy.get("[data-testid='confirm-delete-button']").click({
```

```
                force: true,
              });
            }
          });
          cy.wait("@deleteStory").its("response.statusCode").should("eq",
200);
          cy.get(".Toastify__toast--success").should(
            "contain",
            "Story Deleted Successfully"
          );
        });
      });

      describe("Тести для списку бажаного", () => {
        const email = `test_${Date.now()}@example.com`;

        const createTestUser = () => {
          cy.request({
            method: "GET",
            url: "http://localhost:8000/api/auth/check-user",
            qs: { email },
          }).then((response) => {
            if (response.body.userExists) {
              cy.request("DELETE", `http://localhost:8000/api/auth/delete-
user`, {
                email,
              });
            }
          });

          cy.request({
            method: "POST",
            url: "http://localhost:8000/api/auth/signup",
            body: {
              username: "TestUser",
              email,
              password: "password123",
            },
          }).then((response) => {
            expect(response.status).to.eq(201);
            const token = response.body.token;
            window.localStorage.setItem("token", token);
          });
        };

        const removeTestUser = () => {
          cy.request("DELETE", `http://localhost:8000/api/auth/delete-user`, {
            email,
          });
          window.localStorage.removeItem("token");
        };

        const loginUser = () => {
          cy.visit("/login");
          cy.get("input[name='email']").type(email);
          cy.get("input[name='password']").type("password123");
          cy.get("[data-testid='login-button']").click();
          cy.url().should("include", "/dashboard");
        };

        const addWishlistItemViaUI = (placeName, address, rating, notes) => {
          cy.get(".add-item-btn").click();
          cy.get("input[placeholder='Place Name']").type(placeName);
          cy.get("input[placeholder='Address']").type(address);
          cy.get("input[placeholder='Rating (1-5)']").type(rating);
          cy.get("textarea[placeholder='Notes']").type(notes);
```

```
    cy.get("input[type='file']").attachFile("test-image.jpg");
    cy.get(".submit-btn").click();
  };

  const editWishlistItemViaUI = (placeName, address, rating, notes) => {
    cy.get("[data-testid='edit-wishlist-item']").first().click();
    cy.get("input[placeholder='Place Name']").clear().type(placeName);
    cy.get("input[placeholder='Address']").clear().type(address);
    cy.get("input[placeholder='Rating (1-5)']").clear().type(rating);
    cy.get("textarea[placeholder='Notes']").clear().type(notes);
    cy.get(".submit-btn").click();
  };

  beforeEach(createTestUser);
  afterEach(removeTestUser);

  it("Додавання подорожі до списку бажаного", () => {
    loginUser();
    cy.visit("/wishlist");
    addWishlistItemViaUI("Test Place", "Test Address", 5, "Test Notes");
    cy.contains("Test Place").should("be.visible");
  });

  it("Редагування подорожі списку бажаного", () => {
    loginUser();
    cy.visit("/wishlist");
    addWishlistItemViaUI("Test Place", "Test Address", 5, "Test Notes");
    editWishlistItemViaUI(
      "Edited Place",
      "Edited Address",
      4,
      "Edited Notes"
    );
    cy.contains("Edited Place").should("be.visible");
  });

  it("Видалення подорожі зі списку бажаного", () => {
    loginUser();
    cy.visit("/wishlist");
    addWishlistItemViaUI("Test Place", "Test Address", 5, "Test Notes");
    cy.get(".delete-btn").first().click();
    cy.contains("Test Place").should("not.exist");
  });

  it("Відображення подорожей списку бажаного", () => {
    loginUser();
    cy.visit("/wishlist");
    addWishlistItemViaUI("Test Place 1", "Test Address 1", 5, "Test
Notes 1");
    addWishlistItemViaUI("Test Place 2", "Test Address 2", 4, "Test
Notes 2");
    cy.contains("Test Place 1").should("be.visible");
    cy.contains("Test Place 2").should("be.visible");
  });

  it("Сортування подорожей списку бажаного за назвою", () => {
    loginUser();
    cy.visit("/wishlist");
    addWishlistItemViaUI("B Place", "B Address", 5, "B Notes");
    addWishlistItemViaUI("A Place", "A Address", 4, "A Notes");
    cy.get(".sort-select").select("name");
    cy.get(".wishlist-item").first().should("contain", "A Place");
  });

  it("Сортування подорожей списку бажаного за рейтингом", () => {
    loginUser();
```

```
            cy.visit("/wishlist");
            addWishlistItemViaUI(
              "Low Rating Place",
              "Low Rating Address",
              2,
              "Low Rating Notes"
            );
            addWishlistItemViaUI(
              "High Rating Place",
              "High Rating Address",
              5,
              "High Rating Notes"
            );
            cy.get(".sort-select").select("rating");
            cy.get(".wishlist-item").first().should("contain", "High Rating
Place");
          });
        });
        describe("Тести продуктивності", () => {
          beforeEach(createTestUser);
          afterEach(removeTestUser);

          it("Швидке завантаження сторінки з великою кількістю історій", () => {
            for (let i = 0; i < 20; i++) {
              cy.request({
                method: "POST",
                url: "http://localhost:8000/api/travel-stories/add",
                headers: {
                  Authorization: `Bearer
${window.localStorage.getItem("token")}`,
                },
                body: {
                  title: `Тестова історія ${i}`,
                  story: `Це тестова історія номер ${i}`,
                  locationName: `Локація ${i}`,
                  visitedDate: Date.now().toString(),
                  tags: ["тест", `${i}`],
                },
              });
            }

            cy.visit("/dashboard", {
              onBeforeLoad(win) {
                win.performance.mark("start-loading");
              },
            });

            cy.get(".grid")
              .should("be.visible")
              .then(() => {
                cy.window().then((win) => {
                  win.performance.mark("end-loading");
                  win.performance.measure(
                    "page-load",
                    "start-loading",
                    "end-loading"
                  );
                  const measure = win.performance.getEntriesByName("page-
load")[0];
                  cy.log(`Час завантаження: ${measure.duration}ms`);
                  expect(measure.duration).to.be.lessThan(3000);
                });
              });
          });
        });
```

```
describe("Адаптивність для мобільних пристроїв", () => {
  beforeEach(() => {
    createTestUser();
    loginUser();

    createStoryViaUI(
      "Мобільна історія",
      "Тестова історія для перевірки на телефоні",
      "Мобільна локація",
      "мобільний"
    );
  });

  afterEach(removeTestUser);

  it("Додавання історії на телефоні", () => {
    cy.viewport("iphone-x");
    cy.visit("/dashboard");

    cy.get("[data-testid='add-story-button']").click();
    cy.get("input[name='title']").type("Мобільна історія 2");
    cy.get("[data-testid='date-selector']").click();
    cy.get(".rdp-day").contains("15").click({ force: true });
    cy.get("textarea").type("Історія створена з мобільного");
    cy.get("input[placeholder='Add Location']").type(
      "Мобільна локація 2{enter}"
    );
    cy.get("[data-testid='save-story-button']").click();

    cy.get(".Toastify__toast--success").should(
      "contain",
      "Story Added Successfully"
    );
  });
});

describe("Безпека та обробка помилок", () => {
  beforeEach(createTestUser);
  afterEach(removeTestUser);

  it("Захист від XSS атак", () => {
    loginUser();

    const maliciousTitle = "<script>alert('XSS')</script>Небезпечна
назва";

    createStoryViaUI(
      maliciousTitle,
      "Звичайний текст історії",
      "Безпечна локація",
      "тест"
    );

    cy.get(".grid").should("contain", maliciousTitle);
    cy.window().then((win) => {
      const alertStub = cy.stub(win, "alert");
      expect(alertStub).not.to.be.called;
    });
  });

  it("Валідація форми при додаванні історії", () => {
    loginUser();
    cy.get("[data-testid='add-story-button']").click();

    cy.get("[data-testid='save-story-button']").click();
```

```
            cy.get(".text-red-500").should("contain", "Please enter the
title!");

            cy.get("input[name='title']").type("Тестова історія");
            cy.get("[data-testid='save-story-button']").click();

            cy.get(".text-red-500").should("contain", "Please enter the
story!");
          });
        });
      });
```

ДОДАТОК Г

(обов’язковий)

ПРЕЗЕНТАЦІЙНІ СЛАЙДИ

Хмельницький національний університет
Кафедра інженерії програмного забезпечення

Кваліфікаційна робота на тему:

«Вебзастосунок для планування та документування туристичних подорожей»

Виконав: студент IV курсу, групи ІПЗ – 21 – 1 Столярчук Є. І.

Керівник: канд. пед. наук, доцент Праворська Н. І.

Актуальність теми

У сучасному світі подорожі стали не лише способом відпочинку, а й важливою частиною особистого розвитку, самовираження та комунікації. З розвитком цифрових технологій зросла потреба в ефективних інструментах для організації мандрівок та збереження пов’язаних із ними вражень. У цьому контексті тема кваліфікаційної роботи є особливо актуальною:

- Сучасні користувачі активно використовують різноманітні цифрові сервіси - нотатки, карти, блоги, фотогалереї тощо.
- Ці сервіси зазвичай існують окремо, що створює незручності під час планування чи збереження спогадів про подорожі.
- Виникає потреба в універсальному інструменті, який би об'єднував декілька функцій в одному середовищі.
- Вебзастосунок, що поєднує щоденник подорожей, планувальник та фотогалерею дозволяє зробити досвід мандрівок більш цілісним та персоналізованим.

2

Мета та Завдання

Метою роботи є розробка вебзастосунку для планування та документування туристичних подорожей, який дозволить легко та зручно організовувати та зберігати інформацію про подорожі із можливістю додання локацій у список бажаного.

Завдання, необхідні для досягнення мети:

- Проаналізувати існуючі сервіси для планування та фіксації подорожей
- Визначити функціональні та нефункціональні вимоги до майбутнього вебзастосунку
- Обґрунтувати вибір архітектурного підходу та стеку технологій
- Спроєктувати структуру бази даних і логіку взаємодії компонентів
- Реалізувати клієнтську та серверну частини вебзастосунку
- Забезпечити безпечну автентифікацію користувачів
- Реалізувати зберігання зображень із подорожей
- Протестувати систему на відповідність функціональним вимогам
- Провести аналіз результатів і визначити шляхи подальшого розвитку застосунку

Змістовний аналіз предметної області, її структурних та функціональних особливостей

Планування подорожей і збереження пов'язаних з ними даних давно вийшло за межі традиційних щоденників і блокнотів. Сучасні мандрівники все частіше шукають цифрові інструменти, які дозволяють фіксувати маршрути, робити нотатки, прикріплювати фотографії, вести хронологію поїздок та повертатись до спогадів у будь-який момент.

Предметна область охоплює кілька ключових функцій: ведення персонального журналу подорожей, можливість зберігати важливі місця «на потім», перегляд маршрутів та фотоархіву, а також планування майбутніх поїздок. Ці функції мають бути реалізовані з використанням простого та зручного інтерфейсу.

Аналіз наявного програмно-технічного забезпечення

Характеристика	Travel Diaries	Daybook	Polarsteps
Відстеження маршруту	Так	Обмежено	Автоматично
Друк фотокниг	Немає	Немає	Так
Планування	Обмежено	Так	Так
Експорт даних	Обмежено	Так	Обмежено
Пошук та фільтрація	Ні	Так	Так
Статистика подорожей	Так	Ні	Так
Додавання медіа	Так	Так	Так
Безкоштовна версія	З обмеженнями	З обмеженнями	З обмеженнями

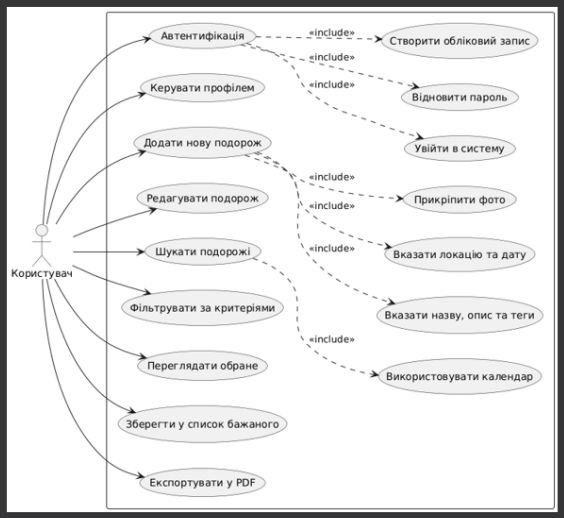
Визначення функціональних та нефункціональних вимог до програмного забезпечення

Функціональні вимоги:

- Створення облікового запису
- Створення, редагування й видалення записів
- Прикріплення фото, вибір дати, місця, тегів
- Ведення списку бажаних локацій;
- Пошук та фільтрація записів
- Позначати подорожі як «Обрані »
- Експорт подорожей у PDF

Нефункціональні вимоги:

- Швидке завантаження сторінок
- Безпечне зберігання паролів
- Стабільна робота
- Підтримка останніх версій популярних браузерів
- Інтуїтивно зрозумілий інтерфейс
- Захищена обробка запитів і робота з базою даних



Діаграма варіантів використання

Вибір типу архітектури та шаблонів проєктування

Для створення вебзастосунку було обрано архітектурний шаблон MVC (Model–View–Controller), який дозволяє чітко розділити логіку обробки даних, відображення інформації та керування діями користувача.

Діаграма архітектури вебзастосунку за шаблоном MVC

Опис декомпозиції, залежностей, інтерфейсів

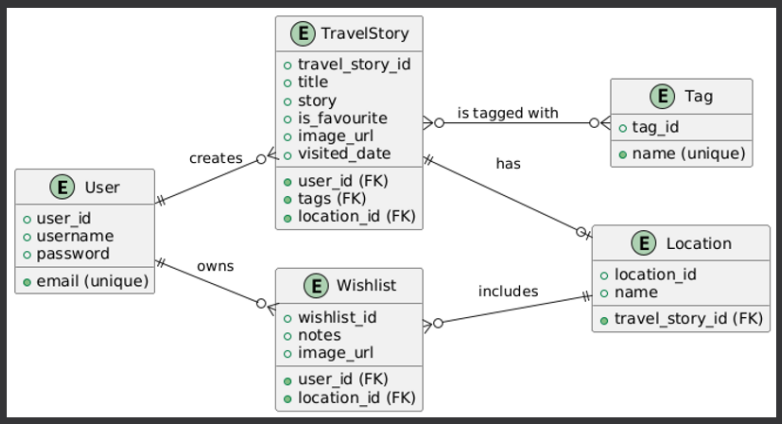
Система була декомпозована на логічно незалежні модулі, кожен з яких відповідає за окрему частину функціоналу. Це дозволяє легше керувати кодом, перевіряти його окремими частинами та масштабувати проєкт у майбутньому.

Основні компоненти:

- Аутентифікація: модуль авторизації з використанням JWT
- Користувачі: реєстрація, редагування профілю, сесії
- Подорожі: створення, редагування, видалення записів
- Список бажаного: додавання і перегляд локацій
- Фільтри та пошук: реалізація запитів за назвою, тегами, датою
- Медіа: інтеграція з Cloudinary для збереження фото

Проєктування модулів і даних

Структура бази даних була розроблена з урахуванням логіки використання застосунку та типових запитів користувачів. Основні сутності в MongoDB представлені у вигляді документів у відповідних колекціях.



ER - діаграма

Аналіз та вибір технологій

Для реалізації вебзастосунку було обрано стек MERN, який складається з чотирьох сучасних і взаємопов'язаних технологій: MongoDB, Express.js, React, Node.js. Усі ці компоненти базуються на JavaScript, що забезпечує єдність мови як на клієнтській, так і на серверній частині.

01

MongoDB

NoSQL база даних із гнучким зберіганням даних

02

Express.js

Фреймворк для Node.js, який спрощує створення RESTful API.

03

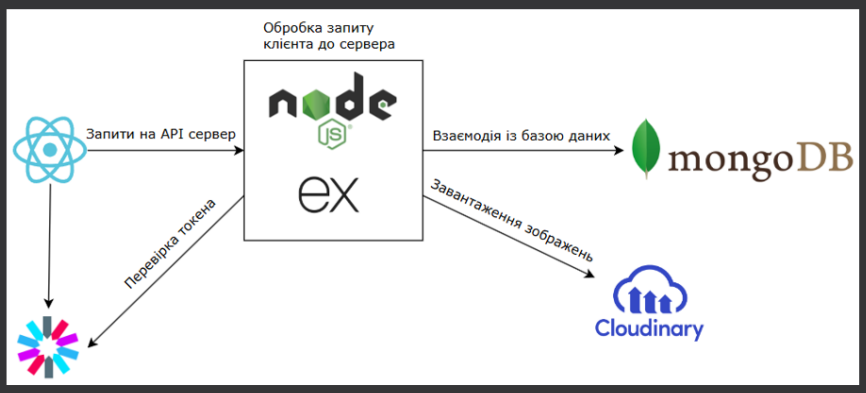
React

Бібліотека для створення інтерфейсів користувача

04

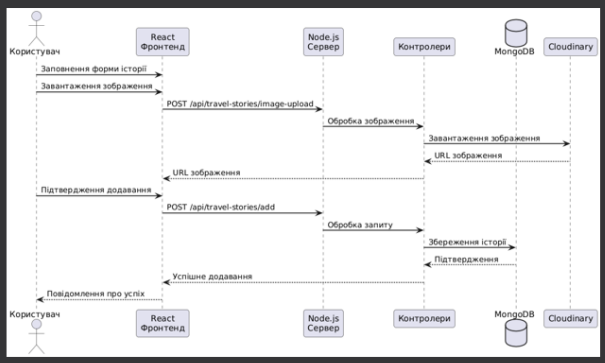
Node.js

Середовище виконання JavaScript для швидкої обробки багатьох запитів одночасно.

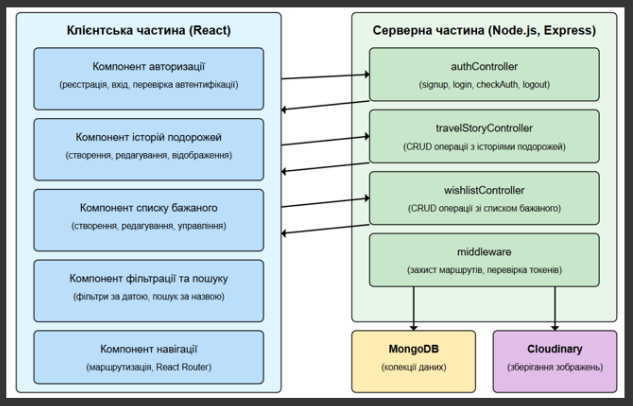


Загальна діаграма архітектури вебзастосунку

Реалізація модулів і база даних

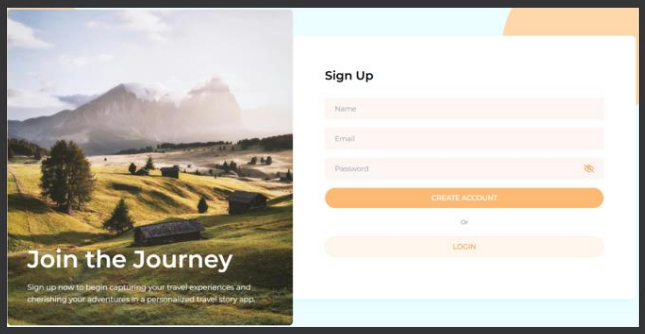


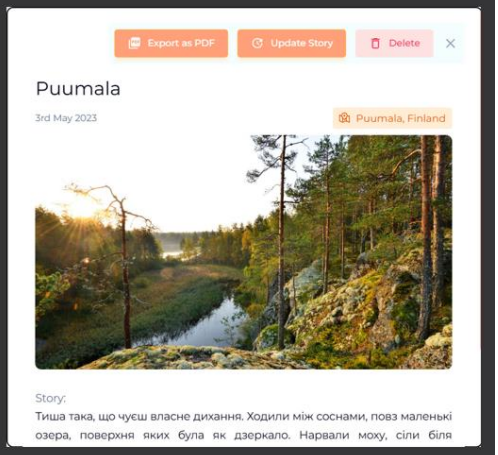
Діаграма послідовностей для додавання історії подорожі

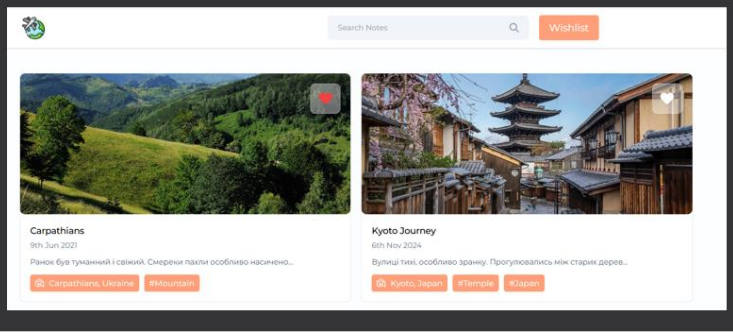


Діаграма компонентів вебзастосунку

Реалізація інтерфейсу вебзастосунку







13

Вимоги до технічного та програмного забезпечення

Програмні вимоги

Сервер:

- Node.js (18+ версії)
- Express.js, Mongoose
- MongoDB (локально / Atlas)
- JWT, bcrypt, dotenv, cors, validator
- Cloudinary API

Клієнт:

- React, React Router, Axios
- Tailwind CSS, React Icons, Toastify

Технічні вимоги

Локальний запуск:

- ПК з ОС Windows / Linux / macOS
- 2+ ГБ оперативної пам'яті
- Встановлений Node.js та npm
- Стабільне підключення до Інтернету

Для користувачів:

- Сучасний браузер (Chrome, Firefox, Edge, Safari)
- Підтримка JavaScript (ES6+)

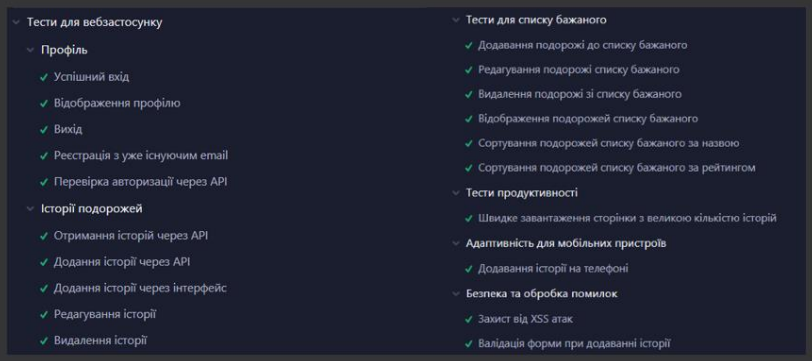
14

Тестування програмного забезпечення

Для перевірки працездатності та відповідності функціоналу вебзастосунку було проведено комплексне тестування.

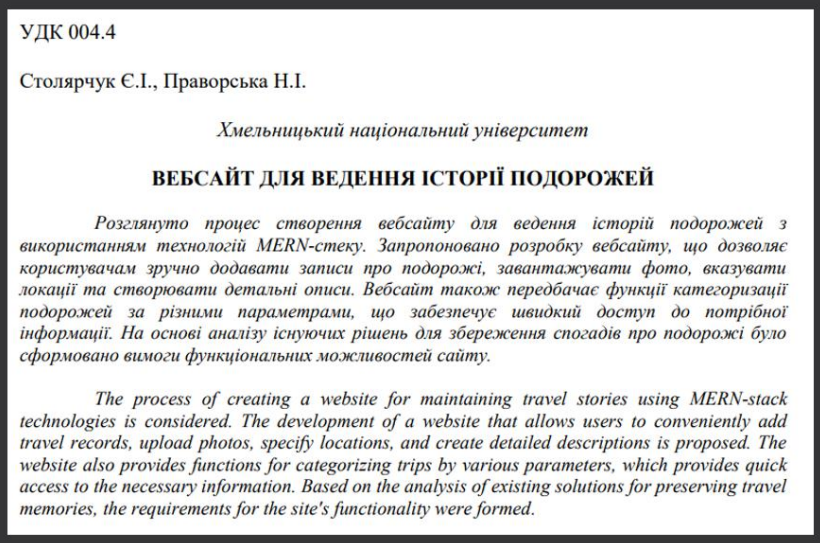
Інструменти тестування:

- Postman: перевірка REST API запитів
- Cypress: наскрізне та інтеграційне тестування



Аналіз результатів тестування

Роботу було опубліковано в Збірнику наукових праць за матеріалами XVI Всеукраїнської науково-практичної конференції «Актуальні проблеми комп'ютерних наук АПКН-2024».



УДК 004.4

Столярчук Є.І., Праворська Н.І.

Хмельницький національний університет

ВЕБСАЙТ ДЛЯ ВЕДЕННЯ ІСТОРІЇ ПОДОРОЖЕЙ

Розглянуто процес створення вебсайту для ведення історій подорожей з використанням технологій MERN-стеку. Запропоновано розробку вебсайту, що дозволяє користувачам зручно додавати записи про подорожі, завантажувати фото, вказувати локації та створювати детальні описи. Вебсайт також передбачає функції категоризації подорожей за різними параметрами, що забезпечує швидкий доступ до потрібної інформації. На основі аналізу існуючих рішень для збереження спогадів про подорожі було сформовано вимоги функціональних можливостей сайту.

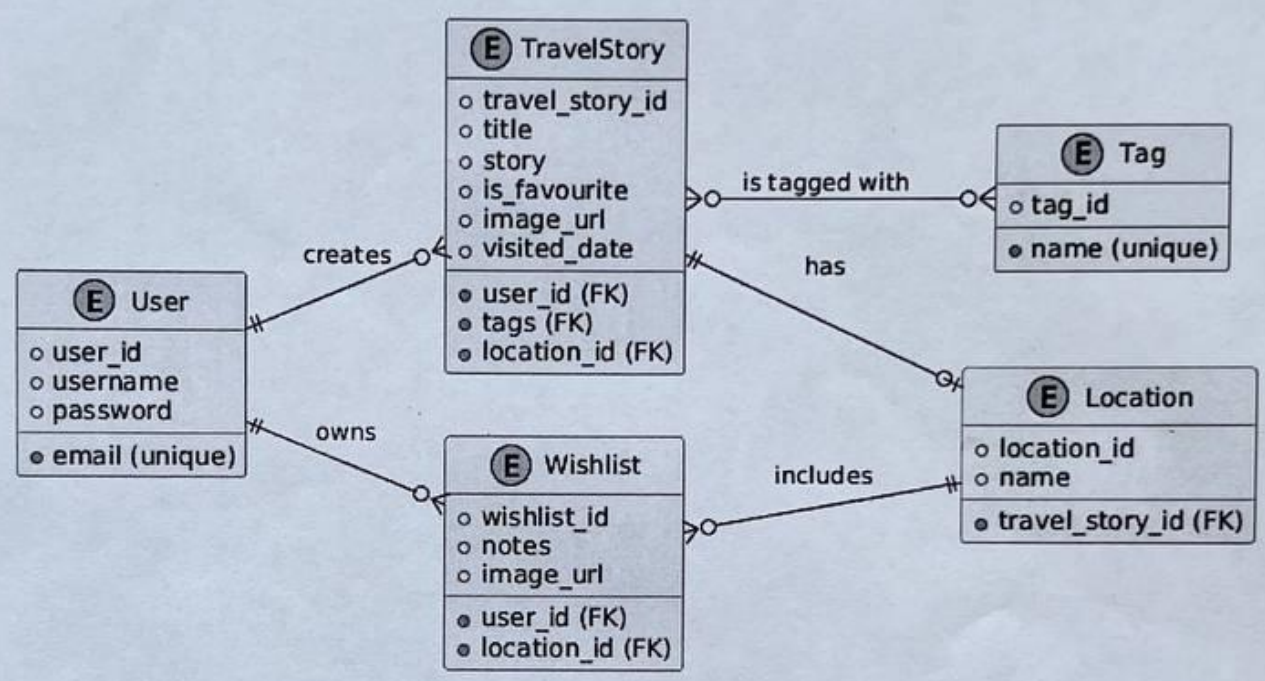
The process of creating a website for maintaining travel stories using MERN-stack technologies is considered. The development of a website that allows users to conveniently add travel records, upload photos, specify locations, and create detailed descriptions is proposed. The website also provides functions for categorizing trips by various parameters, which provides quick access to the necessary information. Based on the analysis of existing solutions for preserving travel memories, the requirements for the site's functionality were formed.

Висновки

Поставлене завдання	Результат виконання
Проаналізувати існуючі рішення	Проведено огляд платформ Travel Diaries, Daybook та Polarsteps, визначено їхні переваги та недоліки
Визначити функціональні та нефункціональні вимоги	Сформульовано перелік вимог, що охоплює основний та допоміжний функціонал системи
Обґрунтувати вибір архітектури та стеку технологій	Обрано шаблон MVC та стек MERN як найбільш оптимальні для задачі
Спроєктувати структуру БД і логіку системи	Розроблено структуру на основі MongoDB, визначено основні сутності та зв'язки
Реалізувати клієнтську та серверну частини	Застосунок реалізовано повністю з використанням React (UI) та Node.js/Express (сервер) з використанням Mongo DB
Забезпечити захист персональних даних	Реалізовано JWT-автентифікацію, хешування паролів через bcrypt, перевірку доступу
Реалізувати функцію зберігання зображень	Інтегровано хмарний сервіс Cloudinary для завантаження та зберігання фото
Провести тестування функціоналу	Проведено тестування, використовуючи Postman та Cypress, підтверджено правильність роботи системи
Проаналізувати результати та визначити напрями розвитку	Визначено можливості інтеграції з картами, соцмережами, а також розробку мобільної версії

Дякую за увагу!

ГРАФІЧНА ЧАСТИНА



					КвРІПЗ. 2101090.01.16.Е8			
Змн.	Арк.	№ докум.	Підпис	Дата	Вебзастосунок для планування та документування туристичних подорожей	Літера	Маса	Масштаб
Розробив		Столярчук Є. І.		02.06				
Керівник		Праворська Н. І.		02.06				
Консультант		Кацубін М. В.		02.06	Логічна модель бази даних	Аркуш 1	Аркушів 2	
Н. Контр.		Бедратюк Г. І		02.06		ХНУ, ІПЗ-21-1		
Зав. каф.		Бедратюк Л. П.		02.06				

					КвРІПЗ.2101090.01.16.Е8		
Змн.	Арк.	№ докум.	Підпис	Дата	Вебзастосунок для планування та документування туристичних подорожей	Літера	Маса / Масштаб
Розробив		Столярчук Є. І.		02.06			
Керівник		Праворська Н. І.		02.06			
Консультант		Бондарєв М.В.		02.06		Аркуш 2	Аркушів 2
Н. Контр.		Бедратюк Г. І		02.06	Компоненти вебзастосунку	ХНУ, ІПЗ-21-1	
Зав. каф.		Бедратюк Л. П.		02.06			

СУПРОВІДНІ ДОКУМЕНТИ

Завідувачу кафедри інженерії програмного
забезпечення проф. Леоніду БЕДРАТЮКУ
здобувача вищої освіти
Столярчука Єгора Ігоровича
факультет ІТ, IV курс, група ІПЗ-21-1

ЗАЯВА

З правилами чинного Положення про систему забезпечення академічної доброчесності в Хмельницькому національному університеті, згідно з яким виявлення академічного плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту і застосування заходів дисциплінарної та академічної відповідальності, ознайомлений. Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на наявність академічного плагіату оповіщений та надаю свою згоду на обробку й збереження університетом моєї роботи в інституційному репозитарії Хмельницького національного університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-обчислювального комплексу StrikePlagiarism та/або програмно-технічного засобу AntiPlagiarism і використання роботи для виявлення академічного плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення текстових збігів у роботах.

Робота надається для перевірки в електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

02.06.25
дата

підпис

Протокол аналізу звіту подібності науковим керівником

Заявляю, що я ознайомився (-лась) з Повним звітом подібності, який був згенерований Системою виявлення і запобігання плагіату щодо роботи:

Автор: Столярчук Єгор

Співавтор:

Назва: БКР_Вебзастосунок для планування та документування туристичних подорожей

Науковий керівник:

Підрозділ: Кафедра інженерії програмного забезпечення

Коефіцієнт подібності 1:4.8%

Коефіцієнт подібності 2:1.7%

Мікропробіли: 0

Заміна букв: 0

Інтервали: 0

Білі знаки: 4

Дата створення звіту: 2025-05-28 20:50:30.0

Після аналізу Звіту подібності констатую наступне:

☑ **Запозичення, виявлені в роботі є законними і не є плагіатом. Рівень подібності не перевищує допустимої межі. Таким чином робота незалежна і приймається.**

☐ **Запозичення не є плагіатом, але перевищено граничне значення рівня подібностей. Таким чином робота повертається на доопрацювання.**

☐ **Виявлено запозичення і плагіат або навмисні текстові спотворення (маніпуляції), як передбачувані спроби укриття плагіату, які роблять роботу невідповідною вимогам законодавства (Ст. 32. ЗУ Про вищу освіту, пункт 3.1, Ст. 42. ЗУ Про освіту) та вимог НАЗЯВО (Критерій 5), а також кодексу етики і процедурам. Таким чином робота не приймається.**

Обгрунтування:

Дата експерт

28.05.2025

Anti-Plagiarism v-15.274 Educational

The maximum coincidence with one document 5.0%

Dictionaries check: en_US, ru_RU, ua_UA. **Errors in the documents: 13%**

	Document		Sum coincidence on the DB	
ID: 241889 Title: ДП_Вебзастосунок для планування та документування туристичних подорожей Added in a DB: 2025-05-26 Authors: Єгор СТОЛЯРЧУК Heads: канд. пед. наук, доцент Наталія ПРАВОРСЬКА Consultants: Opponents:	Symbols	Lexemes	Symbols	Lexemes
	81249	1252	5299 (7%)	70 (6%)

Plagiarism sources

ID	Description	Plagiarism presence in the document	
		Symbols	Lexemes

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

освітнього ступеня «Бакалавр»

Дипломник Столярчук Єгор Ігорович

Тема Вебзастосунок для планування та документування туристичних подорожей

Спеціальність 121 – Інженерія програмного забезпечення

Обсяг кваліфікаційної роботи:

Кількість листів креслень 2 ; кількість сторінок записки 73 .

1.Короткий зміст пояснювальної записки та прийнятих рішень У кваліфікаційній роботі розглянуто створення вебзастосунку для планування та документування туристичних подорожей. Проведено аналіз предметної області, сформульовано вимоги, обґрунтовано архітектурні та технологічні рішення. Реалізовано функціональний застосунок із можливістю збереження, фільтрації та експорту інформації про подорожі. Проведено тестування, що підтвердило коректну роботу програмного забезпечення.

2. Висновок про відповідність роботи поставленому завданню Кваліфікаційна робота виконана відповідно до поставленого завдання та з дотриманням всіх вимог.

3. Характеристика виконання кожного розділу роботи, ступінь використання останніх досягнень науки і техніки та передових методів роботи У вступі обґрунтовано актуальність теми, сформульовано мету, завдання, об’єкт і предмет дослідження. У першому розділі проведено змістовний аналіз предметної області. Проаналізовано сучасні сервіси, їх функціонал, переваги та недоліки. Виділено ключові проблеми, які вирішує розроблюваний вебзастосунок, сформульовано функціональні й нефункціональні вимоги. У другому розділі спроєктовано архітектуру вебзастосунку на основі шаблону MVC та клієнт-серверної моделі. Розроблено структуру бази даних, спрощену та повну ER-діаграму, структуру взаємодії між компонентами. Обґрунтовано вибір стеку технологій MERN. Створено інтерфейс користувача з урахуванням зручності використання. У третьому розділі реалізовано серверну і клієнтську частини, створено базу даних, функціональні модулі для обліку подорожей, списку бажаного, експорту в PDF. Проведено тестування вебзастосунку та проаналізовано результати, що підтверджують відповідність функціоналу поставленим вимогам.

4. Позитивні сторони роботи Актуальність обраної теми полягає у потребі зручного цифрового інструменту для збереження спогадів про подорожі. Робота відзначається глибоким аналізом аналогів, якісним проєктуванням, застосуванням сучасних технологій (MongoDB, Express.js, React, Node.js, Cloudinary, JWT), а також орієнтацією на зручність користувача.

5. Негативні сторони роботи Обмежена реалізація фільтрації подорожей – доцільно було б додати пошук за тегами або місцями. Також варто передбачити можливість спільного редагування подорожей або інтеграцію із соціальними мережами.

6. Оцінка графічного оформлення та пояснювальної записки Графічне оформлення відповідає тематиці роботи та містить діаграми архітектури, послідовностей, компонентів тощо. Пояснювальна записка оформлена згідно вимог і має логічну структуру викладення матеріалу.

7. Відгук про кваліфікаційну роботу в цілому Кваліфікаційна робота є завершеною, цілісною та технічно грамотною. У пояснювальній записці чітко прослідковується логіка побудови рішення – від аналізу проблеми до її практичної реалізації. Застосовані сучасні технології та архітектурні підходи забезпечили створення функціонального та зручного сервісу. Робота добре структурована та наповнена ілюстративним матеріалом.

8. Інші зауваження

9. Оцінка кваліфікаційної роботи Кваліфікаційна робота виконана у повному обсязі, відповідає поставленій задачі та заслуговує на оцінку «відмінно».

РЕЦЕНЗЕНТ к.т.н., доцент кафедри КІК, Кацуба Марія Вікторівна

"1" червня 2025 р. _____
(підпис)

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

першого освітнього рівня «Бакалавр»

Студента: Столярчука Єгора Ігоровича
Група: ІП3-21-1

Тема: *«Вебзастосунок для планування та документування туристичних подорожей»*

Спеціальність: 121 – Інженерія програмного забезпечення

Короткий зміст пояснювальної записки

У вступі кваліфікаційної роботи акцентується на важливості інформаційних технологій у сфері подорожей та туризму, підкреслюючи актуальність створення вебзастосунку для планування та документування туристичних подорожей. Мета роботи полягає у розробці зручного інструменту для організації та збереження інформації про подорожі, що відповідає потребам

сучасних мандрівників. Завдання включають аналіз існуючих рішень, визначення вимог до функціоналу, проектування архітектури, розробку вебзастосунку та його тестування.

Відповідність отриманих результатів роботи поставленим завданням

Завдання, сформульовані у вступі, включають: 1. Аналіз наявних рішень для планування та документування туристичних подорожей. 2. Визначення ключових аспектів та вимог до функціоналу. 3. Проектування архітектури розроблюваного продукту. 4. Розробка вебзастосунку із можливістю зберігання вражень про подорожі. 5. Проведення тестування роботи вебзастосунку. 6. Аналіз успішності розробленого продукту.

Отримані результати в цілому відповідають поставленим завданням. Вебзастосунок розроблений з урахуванням аналізу ринку, визначено функціональні та нефункціональні вимоги, проведено тестування. Оцінка відповідності: **в цілому відповідають**.

Оцінка розділів

Розділ 1

У розділі представлено змістовний аналіз предметної області, включаючи класифікацію вебсайтів та їх функціональні особливості. Описано популярні сервіси, що вирішують проблеми туристів, та важливість створення зручного інтерфейсу. Однак, аналіз є загальним і не містить достатньої глибини, а також відсутні приклади сценаріїв використання. Рекомендується додати детальніший опис функціональних особливостей вебсайтів.

Розділ 2

У цьому розділі розглянуто вибір архітектури, проектування структури даних та інтерфейсу користувача. Описано переваги архітектури MVC, логічну модель бази даних та структуру інтерфейсу. Проте, відсутні порівняння альтернативних архітектур та деталі про фізичне зберігання даних. Рекомендується додати порівняння архітектур та опис фізичної моделі бази даних.

Розділ 3

У розділі детально описано реалізацію бази даних, програмних модулів та тестування. Описано використання MongoDB, REST API та методів тестування. Однак, відсутні графіки для візуалізації результатів тестування та деталі про тригери. Рекомендується додати графіки, діаграми та уточнити інформацію про тригери.

Позитивні сторони

Кваліфікаційна робота демонструє оригінальність у підході до розробки вебзастосунку, що відповідає сучасним потребам мандрівників. Висока якість рішень, зокрема, використання архітектури MVC та MERN-стеку, забезпечує гнучкість та масштабованість системи. Зручний інтерфейс, що проектується, підвищує користувацький досвід.

Недоліки

Серед недоліків можна відзначити недостатню глибину аналізу предметної області, відсутність детальних порівнянь альтернативних рішень та недостатню візуалізацію результатів тестування. Також, деякі аспекти реалізації потребують уточнення, зокрема, інформація про тригери та фізичну модель бази даних.

Відгук в цілому

Кваліфікаційна робота є актуальною та практично значущою, оскільки відповідає сучасним потребам у сфері туризму. Зміст роботи відповідає темі та завданням, а новизна ідей та рішень підкреслює інноваційний підхід. Об'єктивність та обґрунтованість матеріалу в цілому на високому рівні, хоча є можливості для покращення.

Оцінка кваліфікаційної роботи

Кваліфікаційна робота заслуговує оцінки **добре**. Вона виконана в повному обсязі з дотриманням основних вимог, проте має деякі недоліки, які потребують доопрацювання.

Рекомендації

Рекомендується доопрацювати роботу, зокрема, уточнити деталі реалізації, додати візуалізації та порівняння альтернативних рішень. Це підвищить якість та зрозумілість матеріалу, а також підкреслить потенціал до впровадження розробленого вебзастосунку.

OpenAI API-асистент
Session ID: cc5e6f95-02a9-4889-91a1-f2d81e769579
Підписано автоматично, модель gpt-4o-mini
Дата: 26.05.2025

РІШЕННЯ ЕКСПЕРТНОЇ КОМІСІЇ КАФЕДРИ ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Назва кваліфікаційної роботи Вебзастосунок для планування та документування туристичних подорожей

Автор Столярчук Єгор Ігорович

Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

Рівень вищої освіти Перший (бакалаврський)

Спеціальність 121 «Інженерія програмного забезпечення»

Науковий керівник Праворська Наталія Іванівна, канд. пед. наук, доцент

На основі аналізу кваліфікаційної роботи на дотримання вимог академічної доброчесності (у т.ч. відсутності ознак академічного плагіату) з урахуванням результатів перевірки роботи спеціалізованим програмним засобом(ами) комісія зробила такий висновок:

№	Висновок	Позначка про відповідність
1	Ознаки академічного плагіату	
1.1	Запозичення, виявлені в роботі, є законними і не є академічним плагіатом (далі – зазначаються підстави віднесення запозичень до правомірних, якщо потрібно). Робота приймається до захисту.	**відповідає**
1.2	Виявлені запозичення не є академічним плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи (далі – зазначаються детальні та аргументовані підстави віднесення запозичень до правомірних). Робота приймається до захисту, але має бути відкоригована.	
1.3	Виявлені запозичення не є академічним плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. Робота може бути допущена до захисту після того як буде відкоригована та доопрацьована і успішно пройде повторну перевірку на академічний плагіат.	
1.4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття текстових запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	
2	Інші види порушень академічної доброчесності	

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

1) у тексті кваліфікаційної роботи системами перевірки на плагіат виявлено схожість з деякими документами в частині загальновживаних обов'язкових словосполучень у стандартних бланках (титулка, завдання, анотація, відомість документів), у структурі змісту, назвах розділів/підрозділів тощо, у назвах публікацій та у переліку джерел посилання;

2) в якості запозичень системою було зафіксовано деякі послідовності вихідного коду і посиланання на бібліотеки, які є стандартними мовними конструкціями програмування та не можуть розглядатися як об'єкт авторських прав і, відповідно, їх порушення;

3) усі запозичення є фрагментарними або мають належним чином оформленні посилання;

4) виявлені модифікації тексту не впливають на відсоток схожості.

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів ідентичності/ схожості, складає 4,8%, що, з урахуванням наведених обґрунтувань, відповідає характеру теми і свідчить на користь кваліфікаційної роботи.

Дата 02.06.25

Завідувач кафедри — Підпис — Леонід БЕДРАТЮК (Ім'я, ПРІЗВИЩЕ)

Гарант освітньої програми — Підпис — Леонід БЕДРАТЮК (Ім'я, ПРІЗВИЩЕ)

Керівник кваліфікаційної роботи — Підпис — Наталія ПРАВОРСЬКА (Ім'я, ПРІЗВИЩЕ)