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

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи

бакалавра
(освітньо-кваліфікаційний рівень)

на тему *«Розробка веб-застосунку для планування подорожей»*

Виконав: здобувач *4* року навчання
групи *4ПР1*
напряму підготовки (спеціальності)
121-Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Лук'янченко Олександр Олександрович
(підпис, прізвище та ініціали)
Керівник *Доровський В.О.*
(підпис, прізвище та ініціали)
Рецензент *к.т.н., доцент Григорова А.А.*
(прізвище та ініціали)
Нормоконтролер *Комісаров О. С.*
(підпис, прізвище та ініціали)

Хмельницький – 2026

ХЕРСОНСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРЗВТЕТ

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

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

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧА

Лук'янченка Олександра Олександровича
(прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) *Розробка веб-застосунку для планування подорожей*

керівник проєкту (роботи) *Доровський Володимир Олексійович, д.т.н., професор*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від «16» січня 2026 року №82-с

2. Строк подання здобувачем проєкту(роботи) *10.06.2026*

3. Вихідні дані до проєкту (роботи) *літературні та періодичні джерела, матеріали переддипломної практики*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
Аналіз предметної області планування подорожей та постановка задачі,
проєктування веб-застосунку для планування подорожей,
програмна реалізація веб-застосунку,
тестування та впровадження веб-застосунку

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
2 таблиці, 34 рисунка

6. Консультанти розділів проєкту (роботи)

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

7. Дата видачі завдання *14.01.2026*

КАЛЕНДАРНИЙ ПЛАН

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

Здобувач _____ *Лук'янченко О.О.*
(підпис) (прізвище та ініціали)

Керівник проєкту (роботи) _____ *Доровський В.О.*
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи бакалавра: 83 сторінки, 34 рисунки, 2 таблиці, 1 додаток, 32 джерела.

Мета роботи:

Розробка веб-застосунку для організації та планування подорожей, що забезпечує автоматизацію процесів створення маршрутів, пошуку туристичних локацій та збереження персоналізованої інформації про плани подорожей користувачів.

Об'єкт дослідження:

Процес планування туристичних подорожей із використанням веб-технологій.

Предмет дослідження:

Методи, технології та програмні засоби розроблення веб-застосунків для планування подорожей.

Інструменти та засоби розробки:

Для розробки веб-застосунку було використано веб-технології Node.js та Express.js для реалізації серверної частини, MongoDB разом із Mongoose для зберігання даних, Handlebars для динамічного формування сторінок, HTML5 і CSS3 для розмітки та стилізації, Google Places API для взаємодії з геолокаційними сервісами.

Результати роботи:

Результатом роботи є веб-застосунок «Планувальник подорожей», який дозволяє вирішити задачі централізації всіх аспектів підготовки до подорожей в єдиній платформі, що суттєво скорочує витрати часу на планування, підвищує зручність роботи з туристичною інформацією та забезпечує персоналізований доступ до збережених маршрутів.

Новизна рішення:

Розроблено інтегровану платформу, яка об'єднує функції пошуку різнопланових туристичних об'єктів (готелів, музеїв, пам'ятників, торгових центрів, розважальних закладів) із персоналізованим управлінням маршрутами в єдиному середовищі.

Практична цінність:

Розроблений веб-застосунок є ефективним інструментом для туристів, що мінімізує час на планування поїздок і покращує взаємодію між учасниками мандрівки. Він може застосовуватися туристичними агентствами, корпоративними клієнтами та індивідуальними мандрівниками.

КЛЮЧОВІ СЛОВА: *ВЕБ-ЗАСТОСУНОК, ПЛАНУВАННЯ ПОДОРОЖЕЙ, JAVASCRIPT, UX/UI, МАРШРУТИ, ІНТЕРАКТИВНИЙ ІНТЕРФЕЙС, ТУРИСТИЧНІ СЕРВІСИ, NODE.JS, MONGODB, GOOGLE PLACES API..*

АНОТАЦІЯ

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

У першому розділі проведено комплексний аналіз предметної області. Було досліджено основні процеси планування туристичних подорожей, визначено проблеми існуючих програмних рішень та обґрунтовано необхідність створення спеціалізованого веб-застосунку. На основі проведеного аналізу було обґрунтовано вибір технологічного стеку Node.js, Express.js, MongoDB, Mongoose, Handlebars, HTML5, CSS3 та Google Places API для реалізації програмного забезпечення.

У другому розділі розглянуто питання проєктування веб-застосунку, визначено архітектуру системи, структуру бази даних, модель взаємодії компонентів та принципи інтеграції із зовнішніми сервісами. Запропонована архітектура забезпечує модульність системи, масштабованість, ефективну взаємодію компонентів, зручність супроводу програмного забезпечення, підтримку сучасних веб-технологій.

У третьому розділі розглянуто питання практичної реалізації веб-застосунку відповідно до спроєктованої архітектури. Розроблено серверну та клієнтську частини системи, механізми взаємодії між компонентами, інтеграцію із зовнішніми API та систему збереження даних. Розглянуто REST-архітектуру серверної частини, структуру MongoDB-моделей, шаблони Handlebars, алгоритми планування маршрутів, механізми авторизації, інтеграцію з Google Places API.

У четвертому розділі розроблено та успішно реалізовано комплексну стратегію тестування веб-застосунку для планування подорожей, яка охопила модульний, інтеграційний, функціональний рівні та перевірку користувацького інтерфейсу. Проведене тестування підтвердило працездатність програмного забезпечення, коректність взаємодії модулів та

стабільність функціонування веб-застосунку. Виконано розгортання системи із використанням хмарної бази даних MongoDB Atlas та хостингової платформи Render з налаштуванням захищених змінних оточення, що забезпечило готовність веб-застосунку до реальної експлуатації.

ABSTRACT

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

In the first chapter provides a comprehensive analysis of the subject area. The main processes of planning tourist trips were investigated, problems of existing software solutions were identified and the need for creating a specialized web application was justified. Based on the analysis, the choice of the Node.js, Express.js, MongoDB, Mongoose, Handlebars, HTML5, CSS3 and Google Places API technology stack for software implementation was justified.

The second chapter considers the issue of web application design, defines the system architecture, database structure, component interaction model, and principles of integration with external services. The proposed architecture provides system modularity, scalability, effective interaction of components, ease of software maintenance, and support for modern web technologies.

The third chapter considers the practical implementation of the web application in accordance with the designed architecture. The server and client parts of the system, mechanisms for interaction between components, integration with external APIs, and a data storage system are developed. The REST architecture of the server part, the structure of MongoDB models, Handlebars templates, route planning algorithms, authorization mechanisms, and integration with Google Places API are considered.

In the fourth chapter, a comprehensive testing strategy for a travel planning web application was developed and successfully implemented, which covered modular, integration, functional levels, and user interface testing. The conducted testing confirmed the functionality of the software, the correctness of the interaction of the modules and the stability of the functioning of the web application. The system was deployed using the MongoDB Atlas cloud database

and the Render hosting platform with the setting of protected environment variables, which ensured that the web application was ready for real operation.

ЗМІСТ

ВСТУП 13

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ПЛАНУВАННЯ ПОДОРОЖЕЙ ТА ПОСТАНОВКА ЗАДАЧІ 17

1.1. Загальна характеристика предметної області 17

1.2. Аналіз процесу планування подорожей 18

1.3. Актуальність цифрових інструментів для організації туристичних подорожей 19

1.4. Порівняльний аналіз популярних онлайн-сервісів для планування подорожей 21

1.5. Обґрунтування необхідності розробки веб-застосунку 25

1.6. Методологічні підходи до розробки веб-застосунків туристичного спрямування 26

1.7. Аналіз та обґрунтування вибору технологій 28

1.8. Формування функціональних вимог до системи 29

1.9. Роль JavaScript у побудові інтерактивних веб-застосунків 30

1.10. Постановка задачі дослідження 32

Висновки до першого розділу 32

2. ПРОЄКТУВАННЯ ВЕБ-ЗАСТОСУНКУ ДЛЯ ПЛАНУВАННЯ ПОДОРОЖЕЙ 33

2.1. Засади сучасної веб-розробки 33

2.2. Огляд ключових технологій і фреймворків 34

2.3. Бібліотеки та інструменти JavaScript для туристичного планувальника 36

2.4. Принципи проєктування зручного користувацького інтерфейсу 37
2.5. Аналіз вимог до веб-застосунку 38
2.6. Проєктування архітектури веб-застосунку 39
2.7. Проєктування структури бази даних 40
2.8. Проєктування серверної частини застосунку 42
2.9. Проєктування клієнтської частини застосунку 44
2.10. Інтеграція Google Places API..... 45
2.11. UML-діаграми веб-застосунку..... 46
2.12. Проєктування безпеки системи 48
Висновки до другого розділу..... 49
3. ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБ-ЗАСТОСУНКУ 50
3.1. Загальна структура програмної реалізації веб-застосунку 50
3.2. Налаштування середовища розробки 52
3.3. Реалізація серверної частини застосунку та підключення бази даних 52
3.4. Реалізація протоколів взаємодії..... 55
3.5. Реалізація взаємодії з MongoDB 56
3.6. Реалізація клієнтської частини застосунку 57
3.7. Реалізація алгоритмів планування подорожей 60
3.8. Реалізація авторизації користувачів 61
3.9. Реалізація інтеграції з Google Places API 62
3.10. Прототипування інтерфейсу планувальника подорожей 63
3.11. Розробка візуального дизайну засобами Figma 65
Висновки до третього розділу 68

РОЗДІЛ 4. ТЕСТУВАННЯ ТА ВПРОВАДЖЕННЯ ВЕБ-ЗАСТОСУНКУ 70
4.1. Стратегія та методика тестування програмного забезпечення 70
4.2. Модульне та інтеграційне тестування компонентів 71
4.2.1. Модульне тестування бізнес-логіки та моделей 71
4.2.2. Інтеграційне тестування REST API та взаємодії з MongoDB 72
4.3. Функціональне тестування та сценарії валідації 73
4.4. Тестування користувацького інтерфейсу та адаптивності 74
4.5. Розгортання та впровадження системи 76
4.6. Забезпечення продуктивності та безпеки системи 77
Висновки до четвертого розділу 77
ВИСНОВКИ 79
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 81
ДОДАТОК А. 84

ВСТУП

У сучасних умовах стрімкого розвитку інформаційних технологій туристична галузь зазнає суттєвих змін, пов’язаних із цифровізацією сервісів та автоматизацією процесів організації подорожей. Зростання популярності онлайн-сервісів для бронювання, навігації та пошуку туристичних об’єктів сприяє підвищенню попиту на інтегровані програмні рішення, здатні забезпечити комплексне планування подорожей із використанням сучасних веб-технологій та геолокаційних сервісів.

Туристичні подорожі займають важливе місце в житті сучасної людини як спосіб пізнання світу, відпочинку та культурного збагачення. Міжнародна туристична галузь демонструє стійке зростання: за підрахунками Всесвітньої туристичної організації ООН, упродовж 2024 року кількість міжнародних поїздок наблизилась до 1,4 мільярда, тоді як сукупні доходи галузі перевищили 1,9 трильйона доларів США. Такий масштаб туристичної активності свідчить про колосальний попит на зручні та ефективні засоби організації подорожей.

Сучасна людина, плануючи поїздку, стикається з цілою низкою організаційних завдань: вибір оптимального маршруту, бронювання транспорту та житла, підбір туристичних об'єктів, розрахунок бюджету. Наразі розв'язання кожного з цих завдань потребує звернення до окремих цифрових платформ: Booking.com чи Airbnb – для підбору помешкань, Skyscanner або Rome2Rio – для пошуку транспортних сполучень, Google Maps – для прокладання маршрутів. Відсутність інтеграції між сервісами породжує фрагментацію інформації і суттєво ускладнює процес підготовки, особливо при організації групових поїздок, коли необхідно узгоджувати плани кількох учасників.

Окрема проблема полягає у збереженні та оперативному доступі до важливої інформації під час самої подорожі. Мандрівники нерідко ведуть нотатки в різних місцях – на папері, в текстових редакторах, у різних

мобільних застосунках – що підвищує ризик втрати даних та плутанини в розкладі. Крім того, переважна більшість існуючих туристичних сервісів не передбачає спільного редагування маршрутів у реальному часі, що ускладнює групове планування.

Отже, планування подорожей є складним багатокомпонентним процесом, який включає вибір туристичних напрямків, пошук локацій, побудову маршрутів, організацію переміщення, оцінку бюджету та збереження інформації про подорож. У більшості випадків користувачі змушені використовувати декілька окремих сервісів одночасно, що ускладнює процес організації подорожі та знижує зручність використання інформаційних ресурсів.

Існуючі програмні рішення часто орієнтовані переважно на комерційні аспекти туристичної діяльності, такі як бронювання готелів або продаж туристичних послуг, при цьому мають надлишковий функціонал, складний інтерфейс або обмежені можливості персоналізації маршрутів. У зв’язку з цим актуальною задачею є розробка веб-застосунку, який забезпечуватиме зручне створення, редагування та збереження маршрутів подорожей із підтримкою інтеграції геолокаційних сервісів.

Розробка веб-застосунку для планування подорожей дозволяє вирішити перелічені вище проблеми шляхом централізації всіх аспектів підготовки до поїздки в єдиній платформі. Такий інструмент здатний суттєво скоротити витрати часу на планування, підвищити зручність роботи з туристичною інформацією та забезпечити персоналізований доступ до збережених маршрутів.

Актуальність теми кваліфікаційної роботи полягає у необхідності створення сучасного веб-застосунку для планування подорожей, що поєднуватиме функції роботи з туристичними маршрутами, геолокаційними сервісами та системою збереження інформації про подорожі користувачів у єдиному інформаційному середовищі.

Об’єктом дослідження є процес планування туристичних подорожей із використанням веб-технологій.

Предметом дослідження є методи, технології та програмні засоби розроблення веб-застосунків для планування подорожей.

Метою кваліфікаційної роботи є розробка веб-застосунку для організації та планування подорожей, що забезпечує автоматизацію процесів створення маршрутів, пошуку туристичних локацій та збереження персоналізованої інформації про плани подорожей користувачів.

Для досягнення поставленої мети необхідно вирішити такі задачі:

- провести аналіз предметної області планування подорожей;
- дослідити існуючі програмні рішення;
- визначити функціональні та нефункціональні вимоги до системи;
- спроектувати архітектуру веб-застосунку;
- реалізувати серверну та клієнтську частини системи;
- реалізувати інтеграцію з Google Places API;
- створити механізми авторизації та збереження даних;
- реалізувати алгоритми формування маршрутів;
- провести тестування веб-застосунку.

Для реалізації програмного забезпечення було використано сучасний стек веб-технологій:

- Node.js та Express.js – для реалізації серверної частини;
- MongoDB та Mongoose – для організації зберігання даних;
- Handlebars – для шаблонізації веб-сторінок;
- HTML5, CSS3 та JavaScript – для створення інтерфейсу користувача;
- Google Places API – для інтеграції геолокаційних сервісів.

Результат виконання кваліфікаційної роботи полягає у створенні працездатного веб-застосунку для планування подорожей, який може бути

використаний як основа для побудови туристичних інформаційних систем, персональних планувальників маршрутів або сервісів підтримки туристичної діяльності.

Методи дослідження: аналіз науково-технічної літератури, порівняння платформ і технологій, синтез і узагальнення отриманих даних, проєктування і програмна реалізація.

Наукова новизна полягає у розробці інтегрованої платформи, яка об'єднує функції пошуку різнопланових туристичних об'єктів (готелів, музеїв, пам'ятників, торгових центрів, розважальних закладів) із персоналізованим управлінням маршрутами в єдиному середовищі.

Практична значущість: розроблений застосунок є ефективним інструментом для туристів, що мінімізує час на планування поїздок і покращує взаємодію між учасниками мандрівки. Він може застосовуватися туристичними агентствами, корпоративними клієнтами та індивідуальними мандрівниками.

Структура кваліфікаційної роботи складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків.

У першому розділі проведено аналіз предметної області та існуючих програмних рішень. У другому розділі розглянуто проєктування архітектури веб-застосунку, структури бази даних та взаємодії компонентів системи. У третьому розділі описано процес розроблення програмного забезпечення, реалізацію основних модулів, інтеграцію із зовнішніми сервісами. У четвертому розділі розглянуто питання тестування та впровадження веб-застосунку.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ПЛАНУВАННЯ ПОДОРОЖЕЙ ТА ПОСТАНОВКА ЗАДАЧІ

1.1. Загальна характеристика предметної області

У сучасному світі туристична галузь є однією з найбільш динамічних сфер діяльності, що активно розвивається завдяки цифровізації та широкому використанню інформаційних технологій.

Планування подорожей є складним процесом, який включає вибір маршруту, пошук туристичних локацій, організацію транспорту, проживання, бюджету та формування графіку подорожі.

З розвитком мережі Інтернет та мобільних технологій значно зросла кількість веб-сервісів і програмних платформ, що дозволяють користувачам організовувати подорожі в онлайн-режимі. Проте більшість існуючих рішень мають обмежений функціонал або орієнтовані лише на окремі аспекти туристичного планування.

Основними задачами систем планування подорожей є:

- автоматизація процесу створення маршрутів;
- пошук туристичних місць;
- збереження інформації про подорожі;
- інтеграція картографічних сервісів;
- оптимізація туристичних маршрутів;
- управління бюджетом подорожі;
- забезпечення швидкого доступу до інформації.

Предметною областю даної роботи є процес організації та планування туристичних подорожей із використанням веб-технологій та геолокаційних сервісів.

1.2. Аналіз процесу планування подорожей

Процес планування подорожі складається з кількох взаємопов’язаних етапів.

1. Вибір напрямку подорожі.

На початковому етапі користувач визначає:

- країну або місто подорожі;
- дати поїздки;
- тип відпочинку;
- орієнтовний бюджет.

2. Пошук туристичних локацій.

Після визначення напрямку користувач здійснює пошук:

- визначних пам’яток;
- готелів;
- ресторанів;
- транспортних вузлів;
- туристичних маршрутів.

Для цього використовуються:

- геолокаційні сервіси;
- картографічні API;
- туристичні каталоги.

3. Формування маршруту.

На етапі формування маршруту виконується:

- визначення послідовності відвідування місць;
- розрахунок відстаней;
- оптимізація часу переміщення;
- оцінка витрат.

4. Збереження та редагування плану подорожі

Користувачі повинні мати можливість:

- зберігати маршрути;

- редагувати подорожі;
- додавати нові локації;
- переглядати історію подорожей.

Схема процесу планування подорожі представлена на рис. 1.1.

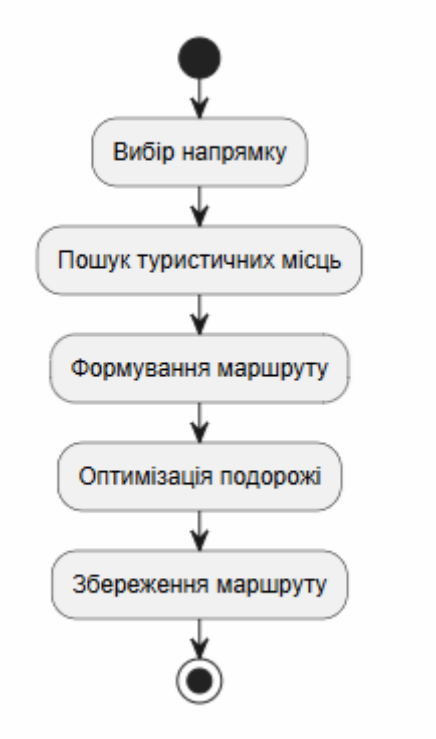

Рисунок 1.1. Схема процесу планування подорожі

1.3. Актуальність цифрових інструментів для організації туристичних подорожей

Туристична галузь належить до найбільш динамічно зростаючих секторів світової економіки. Цифровізація суспільства кардинально змінила підходи до організації подорожей: сучасні мандрівники дедалі частіше відмовляються від послуг традиційних турагентств на користь самостійного планування за допомогою онлайн-інструментів. Згідно з даними Всесвітньої туристичної організації ООН (UNWTO), у докризовому 2019 році зафіксовано близько 1,5 мільярда міжнародних туристичних прибуттів, а

вже у 2024 році показник відновився до 99 % від допандемічного рівня, що підтверджує потужний відновлювальний потенціал галузі.

Особливого значення набувають регіональні тенденції. Так, Європа прийняла понад 747 мільйонів туристів, перевищивши показники 2019 року на 1 %. Водночас регіони Близького Сходу та Африки продемонстрували зростання на 32 % відносно допандемічного рівня, що свідчить про географічне перерозподілення туристичних потоків. Доходи від міжнародного туризму у 2024 році склали 1,9 трильйона доларів США – у середньому понад 1000 доларів на одного туриста. Ці дані наочно демонструють, що туризм залишається вагомим рушієм глобального економічного розвитку.

Паралельно зі зростанням обсягів туристичної активності суттєво змінюються й очікування мандрівників щодо якості цифрових послуг. Сучасний турист прагне мати доступ до вичерпної інформації про маршрути, житло, транспортні сполучення та визначні місця в режимі реального часу, причому з будь-якого пристрою. Саме тому попит на багатофункціональні туристичні платформи з інтегрованим функціоналом постійно зростає.

Аналіз поведінки сучасних мандрівників засвідчує, що більшість з них використовує щонайменше три-чотири різні цифрові сервіси під час підготовки до однієї поїздки. Така фрагментація призводить до надмірних витрат часу, ризику втрати важливих даних і загальної незадоволеності процесом планування. За дослідженнями фахівців у сфері UX, складний і непередбачуваний процес підготовки нерідко стає вагомим чинником відмови від подорожі або зниження якості відпочинку.

На ринку вже присутні окремі спеціалізовані рішення: системи бронювання житла (Booking.com, Airbnb), пошукові системи авіаквитків (Skyscanner, Kayak), платформи мандрівничих рекомендацій (TripAdvisor).

Проте жодна з них не надає по-справжньому комплексного інструменту для планування «від ідеї до маршруту».

Наявна ринкова ніша є підґрунтям для розробки застосунку, що об'єднає пошук туристичних об'єктів різних категорій, управління персональними планами і навігацію в єдиному середовищі.

Таким чином, потреба у створенні централізованого цифрового простору для комплексного планування подорожей є цілком обґрунтованою й актуальною. Веб-застосунок, що поєднує пошук туристичних об'єктів, управління маршрутами та персоналізований доступ до збережених планів, здатний усунути ключові проблеми в організації подорожей і суттєво покращити користувацький досвід.

1.4. Порівняльний аналіз популярних онлайн-сервісів для планування подорожей

Ринок цифрових туристичних послуг є надзвичайно конкурентним і налічує десятки великих платформ.

Для визначення функціональних вимог до розроблюваного застосунку необхідно детально проаналізувати найбільш популярні з них.

На сьогодні найбільш популярними веб-сервісами для планування подорожей є:

- Airbnb;
- Booking;
- TripAdvisor;
- Google Travel;
- Sygic Travel.

Airbnb – платформа, що спеціалізується на посередництві між власниками нерухомості та мандрівниками. Головна конкурентна перевага сервісу – широкий вибір нестандартних варіантів помешкань: будиночки на

деревах, замки, плавучі будинки. Система перевірених відгуків формує довіру між сторонами угоди.

Інтерфейс платформи (рис. 1.2) вирізняється чистотою і простотою: пошуковий рядок з мінімальним набором обов'язкових параметрів і великі фотографії об'єктів. Адаптивний дизайн забезпечує однаковий досвід взаємодії на різних пристроях. Разом з тим, платформа не надає інструментів для планування маршрутів або пошуку туристичних атракцій.

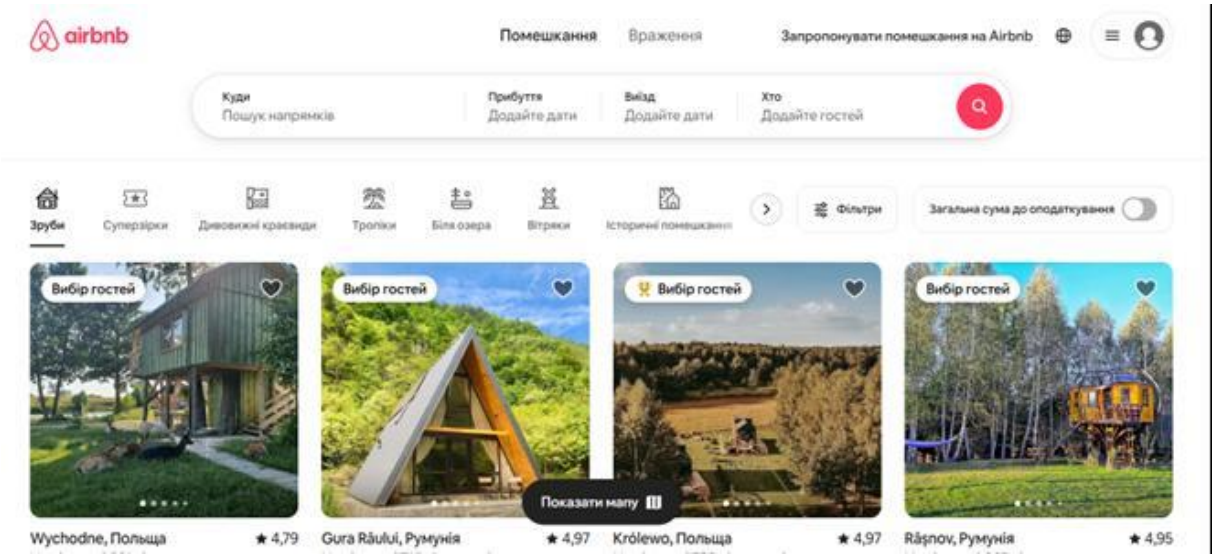

Рисунок 1.2. Інтерфейс Airbnb

Booking.com (рис. 1.3) є однією з найбільших у світі систем бронювання помешкань, що охоплює понад 28 мільйонів об'єктів різних категорій. Ключова перевага платформи – потужний механізм фільтрації з десятками параметрів (ціна, зірковість, зручності, безкоштовне скасування), що дозволяє максимально точно відповідати потребам конкретного мандрівника. Окрім бронювання житла, платформа пропонує оренду автомобілів, трансфери та авіаквитки. Проте базовий функціонал планування маршрутів реалізований менш детально.

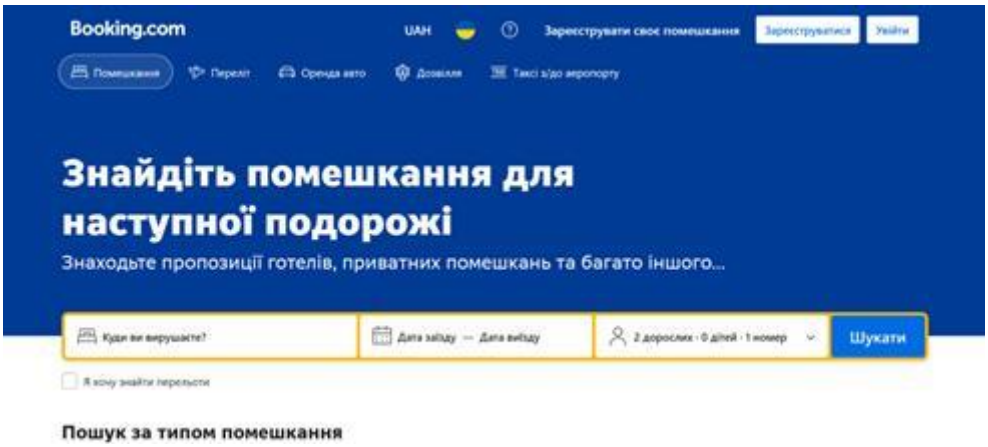

Рисунок 1.3. Інтерфейс Booking.com

TripAdvisor (рис. 1.4) позиціонується як платформа мандрівничих рекомендацій. Її головна цінність – масивна база відгуків від реальних туристів про готелі, ресторани, атракції та екскурсії. Функція побудови маршруту дозволяє скласти план відвідин ключових об'єктів у певному місті чи регіоні. Рейтингова система і фотографії від реальних відвідувачів є вагомим чинником довіри. Кількість фільтрів для пошуку житла є меншою порівняно з Booking.com.

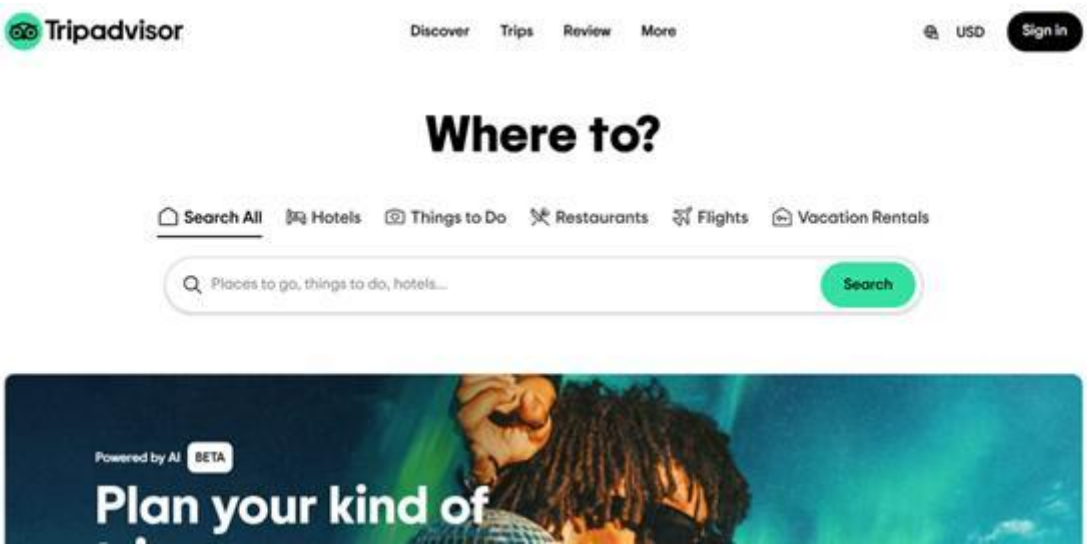

Рисунок 1.4. Інтерфейс TripAdvisor

Порівняльний аналіз цих трьох платформ виявляє спільну рису: жодна з них не є по-справжньому універсальним інструментом для планування подорожі «від початку до кінця».

Основними функціями таких систем є:

- пошук локацій;
- бронювання готелів;
- планування маршрутів;
- рекомендації місць;
- інтеграція з картографічними сервісами.

Airbnb зосереджується на помешканнях, Booking.com – на бронюванні широкого спектру об'єктів, TripAdvisor – на рекомендаціях і відгуках.

Попри широкий функціонал, ці системи мають низку недоліків:

- надлишкова складність інтерфейсу;
- велика кількість рекламного контенту;
- відсутність персоналізації;
- обмежені можливості створення власних маршрутів;
- складність інтеграції декількох сервісів;
- залежність від платних функцій.

Крім того, значна частина сервісів орієнтована переважно на:

- бронювання готелів;
- продаж туристичних послуг;
- рекламні рекомендації.

У результаті користувачі часто змушені використовувати декілька платформ одночасно.

Відсутність єдиної платформи, що органічно поєднувала б пошук місць проживання, туристичних атракцій, управління маршрутами і персоналізовані плани подорожей, є ринковою нішею для розроблюваного застосунку.

Ключові висновки з аналізу для проєктування власного застосунку: необхідно забезпечити адаптивний дизайн; пошук має підтримувати кілька категорій об'єктів; картки результатів повинні містити фото, адресу і рейтинг; персоналізована система збереження планів підвищує залученість; реєстрація і авторизація є обов'язковим мінімумом.

1.5. Обґрунтування необхідності розробки веб-застосунку

Розробка спеціалізованого веб-застосунку для планування подорожей дозволить:

- об’єднати ключові функції в єдиній системі;
- спростити процес створення маршрутів;
- забезпечити інтеграцію з геолокаційними сервісами;
- реалізувати персоналізований підхід;
- забезпечити доступність із будь-якого пристрою.

Веб-застосунок має такі переваги:

- доступ через браузер;
- відсутність необхідності встановлення;
- підтримка мобільних пристроїв;
- централізоване зберігання даних;
- можливість масштабування.

Особливо важливою є інтеграція із Google Places API, що забезпечує:

- доступ до актуальної геолокаційної інформації;
- отримання координат та карт;
- пошук туристичних об’єктів;
- відображення рейтингів та відгуків.

1.6. Методологічні підходи до розробки веб-застосунків туристичного спрямування

Вибір методології розробки безпосередньо впливає на якість кінцевого програмного продукту, терміни його виведення на ринок та здатність адаптуватися до мінливих вимог. У контексті проєктування веб-застосунків для планування подорожей особливо важливо забезпечити гнучкість системи та можливість поетапного нарощування функціоналу.

Серед методологій гнучкої розробки найбільшого поширення набули Agile, Scrum і Kanban. Agile-підхід базується на ітеративному розвитку продукту: кожна нова функціональність реалізується, перевіряється і вдосконалюється в рамках коротких циклів, що дозволяє своєчасно реагувати на зміни у вимогах. Scrum структурує роботу через спринти тривалістю від одного до чотирьох тижнів, наприкінці кожного з яких команда отримує готовий до демонстрації інкремент продукту. Kanban натомість зосереджується на безперервному потоці задач і рівномірному навантаженні виконавців, що унеможливлює накопичення незавершеної роботи.

Для туристичного планувальника гнучкі методології є особливо доречними, оскільки дозволяють поетапно нарощувати функціонал: спочатку реалізувати базовий пошук місць, а потім поступово додавати управління планами, інтеграцію з картографічними сервісами, авторизацію користувачів тощо. Водночас гнучкий підхід вимагає злагодженої командної комунікації та чіткого пріоритизування задач.

Альтернативою є каскадна модель (Waterfall), за якою розробка ведеться послідовно: аналіз вимог, архітектурне проєктування, кодування, тестування і впровадження. Ця модель підходить для проєктів із фіксованим і добре визначеним переліком вимог, забезпечуючи передбачуваність і документованість кожного етапу. Проте внесення змін після завершення попереднього етапу вимагає значних зусиль. Для планувальника подорожей

каскадна модель доцільна лише на початковому етапі, для опрацювання архітектурних рішень.

Важливим архітектурним рішенням є вибір між монолітним і мікросервісним підходами. Моноліт передбачає розміщення всіх компонентів – інтерфейсу, бізнес-логіки і рівня даних – в межах єдиного розгортуваного модуля. Така архітектура спрощує початкову розробку, скорочує накладні витрати на міжкомпонентну комунікацію і, як правило, демонструє вищу продуктивність на невеликих масштабах. Саме тому моноліт є типовим вибором для MVP-продуктів і стартапів.

Мікросервісна архітектура розбиває застосунок на автономні сервіси, кожен з яких відповідає за чітко визначену функціональну область і розгортається незалежно. Такий підхід забезпечує горизонтальну масштабованість, стійкість до збоїв і технологічну різноманітність. Проте він потребує зрілої інфраструктури і складнішого операційного управління. Для туристичного планувальника на початковому етапі доцільно використовувати моноліт із модульною внутрішньою організацією.

Невід'ємним аспектом розробки є забезпечення безпеки застосунку. Планувальник зберігає персональні дані користувачів, тому їх захист є першочерговим завданням. Основні заходи включають хешування паролів за допомогою bcrypt, передавання даних виключно через HTTPS, захист від SQL-ін'єкцій і міжсайтового скриптингу (XSS). Двофакторна автентифікація суттєво підвищує рівень захисту облікових записів.

Ключову роль у формуванні конкурентоспроможності продукту відіграє якість UX/UI-дизайну. Планування подорожі є багатоетапним процесом, тому інтерфейс застосунку повинен бути максимально зрозумілим і мінімізувати когнітивне навантаження. Принципи проєктування зручного інтерфейсу передбачають логічну ієрархію елементів, узгодженість стилів, чітку навігацію і зворотний зв'язок при

виконанні дій. Адаптивний дизайн є обов'язковою вимогою для сучасних веб-застосунків.

1.7. Аналіз та обґрунтування вибору технологій

Для реалізації веб-застосунку було обрано сучасний стек веб-технологій.

1. Node.js

Node.js використовується як серверна платформа для обробки HTTP-запитів та реалізації бізнес-логіки.

Основні переваги:

- висока продуктивність;
- асинхронна модель виконання;
- підтримка великої кількості одночасних підключень;
- використання JavaScript як на сервері, так і на клієнті.

2. Express.js

Express.js є фреймворком для Node.js, який забезпечує:

- маршрутизацію;
- middleware;
- підтримку REST API;
- модульність серверної частини.

3. MongoDB

MongoDB використовується як документоорієнтована база даних.

Переваги:

- зберігання JSON-подібних документів;
- гнучка структура даних;
- масштабованість;
- швидка інтеграція з JavaScript.

4. Mongoose

Mongoose забезпечує:

- створення схем даних;
- валідацію;
- ORM-подібну взаємодію з MongoDB;
- спрощення роботи з базою даних.

5. Handlebars

Handlebars використовується для:

- генерації динамічних HTML-сторінок;
- повторного використання шаблонів;
- формування інтерфейсу користувача.

6. HTML5 та CSS3

HTML5 і CSS3 забезпечують:

- структуру веб-сторінок;
- адаптивний дизайн;
- сучасний інтерфейс користувача.

7. Google Places API

Google Places API використовується для:

- пошуку локацій;
- роботи з картами;
- отримання інформації про туристичні об’єкти;
- побудови маршрутів.

1.8. Формування функціональних вимог до системи

На основі проведеного аналізу було сформовано функціональні вимоги до веб-застосунку.

Система повинна забезпечувати:

- реєстрацію та авторизацію користувачів;
- створення та редагування подорожей;
- пошук туристичних локацій;
- інтеграцію з Google Places API;

- формування маршрутів;
- збереження історії подорожей;
- управління списком улюблених місць;
- адаптивний інтерфейс користувача.

Нефункціональні вимоги

До нефункціональних вимог належать:

- швидкодія;
- безпечність;
- масштабованість;
- зручність використання;
- кросбраузерність;
- підтримка мобільних платформ.

1.9. Роль JavaScript у побудові інтерактивних веб-застосунків

JavaScript є однією з фундаментальних технологій сучасного вебу і єдиною мовою програмування, що виконується безпосередньо в браузері, забезпечуючи динамічність і інтерактивність веб-сторінок без необхідності їх перезавантаження. Для туристичного планувальника, де користувач постійно взаємодіє з картками місць, вкладками категорій і формами введення, JavaScript є незамінним інструментом.

Однією з ключових можливостей JavaScript є маніпуляція DOM – об'єктною моделлю документа. DOM являє собою деревоподібне представлення структури HTML-сторінки, яке JavaScript здатен читати і модифікувати в режимі реального часу. Завдяки цьому застосунок може динамічно оновлювати списки результатів пошуку, відображати або приховувати окремі секції сторінки, змінювати стилі елементів і реагувати на події – все це без звернення до сервера. У контексті туристичного планувальника цей механізм забезпечує миттєве відображення карток знайдених місць після виконання пошукового запиту.

Асинхронне програмування є ще одним критично важливим аспектом. JavaScript підтримує механізми Promises та синтаксис async/await, що дозволяють виконувати мережеві запити до зовнішніх API без блокування основного потоку виконання. Для взаємодії з Google Places API або власним серверним API застосовується вбудований Fetch API або бібліотека Axios. Вбудований Fetch API не потребує зовнішніх залежностей, тоді як Axios надає розширений функціонал: автоматичну серіалізацію JSON, перехоплювачі запитів і зручну обробку помилок.

На серверному боці JavaScript реалізується через Node.js – середовище виконання, побудоване на движку V8 від Google. Node.js застосовує подійно-орієнтовану, неблокувальну архітектуру вводу-виводу, що дозволяє ефективно обробляти велику кількість одночасних з'єднань. Express.js – мінімалістичний Node.js-фреймворк – спрощує визначення маршрутів, підключення middleware і організацію RESTful API. Для туристичного планувальника Express.js дозволяє чітко структурувати серверну логіку: окремі маршрутизатори для автентифікації, пошуку місць і управління планами. Важливим аспектом клієнтського JavaScript є обробка подій – механізм, що дозволяє реагувати на дії користувача (кліки, введення тексту, прокручування) без перезавантаження сторінки. У туристичному планувальнику обробники подій забезпечують перемикання між категоріями пошуку, додавання місць до персонального списку, асинхронне видалення елементів зі збережених планів і відображення автодоповнень при введенні назви міста.

Загалом, JavaScript є системоутворюючою технологією для туристичного планувальника: на клієнтському боці він відповідає за динамічне формування інтерфейсу, обробку подій і взаємодію з API, а на серверному – за бізнес-логіку, маршрутизацію і доступ до бази даних. Використання єдиної мови на обох рівнях скорочує перемикання контексту і спрощує обмін даними між клієнтом і сервером.

1.10. Постановка задачі дослідження

Метою кваліфікаційної роботи є розробка веб-застосунку для планування подорожей із використанням Node.js, Express.js, MongoDB, Handlebars та Google Places API.

Для досягнення поставленої мети необхідно вирішити такі задачі:

- провести аналіз предметної області;
- дослідити існуючі програмні рішення;
- визначити вимоги до системи;
- спроектувати архітектуру веб-застосунку;
- реалізувати серверну та клієнтську частини;
- реалізувати інтеграцію із Google Places API;
- створити механізми авторизації користувачів;
- реалізувати алгоритми формування маршрутів;
- провести тестування програмного забезпечення.

Висновки до першого розділу

У результаті аналізу предметної області було досліджено основні процеси планування туристичних подорожей, визначено проблеми існуючих програмних рішень та обґрунтовано необхідність створення спеціалізованого веб-застосунку.

Було встановлено, що сучасні веб-технології дозволяють реалізувати ефективну систему планування подорожей із підтримкою геолокаційних сервісів, адаптивного інтерфейсу та персоналізованих маршрутів.

На основі проведеного аналізу було обґрунтовано вибір технологічного стеку Node.js, Express.js, MongoDB, Mongoose, Handlebars, HTML5, CSS3 та Google Places API для реалізації програмного забезпечення.

2. ПРОЕКТУВАННЯ ВЕБ-ЗАСТОСУНКУ ДЛЯ ПЛАНУВАННЯ ПОДОРОЖЕЙ

2.1. Засади сучасної веб-розробки

Веб-розробка є міждисциплінарною сферою, що охоплює проєктування, програмування і тестування програмних продуктів, доступних через браузер. Традиційно вона поділяється на клієнтську (frontend) і серверну (backend) частини, хоча сучасні підходи все частіше передбачають їх тісну інтеграцію в рамках єдиного технологічного стеку.

HTML (HyperText Markup Language) є базовою мовою розмітки, що описує структуру та семантику веб-документа. HTML5 – остання стандартна версія – значно розширила можливості попередніх версій: додала нативну підтримку мультимедіа через елементи <video> і <audio>, нові семантичні теги (<header>, <nav>, <main>, <article>, <section>, <footer>), а також API для роботи з геолокацією і локальним сховищем. Семантична розмітка покращує доступність документа для пошукових систем і допоміжних технологій, а також підвищує зрозумілість коду для розробників.

CSS (Cascading Style Sheets) відповідає за візуальне оформлення HTML-документів. Сучасний CSS3 надає розробникам потужний інструментарій: медіа-запити для реалізації адаптивного дизайну, Flexbox і CSS Grid для гнучкого управління розміщенням елементів, кастомні властивості (змінні CSS) для централізованого управління дизайн-токенами, трансформації і анімації для збагачення інтерактивності. Принцип «Mobile First» є рекомендованим підходом: спочатку визначається базовий стиль для мобільних пристроїв, а потім поступово розширюється для більших екранів.

JavaScript виконує роль «клею», що об'єднує структуру (HTML) і стиль (CSS) в живий, інтерактивний веб-застосунок. Версія стандарту ES6+ суттєво збагатила мову: стрілкові функції, деструктурування, шаблонні

рядки, класи, модулі, Promises і async/await зробили код більш лаконічним і зручним для підтримки.

Серверна частина веб-застосунків забезпечує обробку запитів, реалізацію бізнес-логіки та взаємодію з базами даних. Node.js дозволяє використовувати JavaScript на серверній стороні. PHP залишається широко використовуваною мовою завдяки низькому порогу входу, тоді як Python із фреймворками Django і Flask є популярним вибором для застосунків з розширеними вимогами до обробки даних.

Бази даних поділяються на реляційні і нереляційні. Реляційні системи (PostgreSQL, MySQL) зберігають дані у вигляді таблиць із жорсткою схемою і підтримують SQL. Нереляційні бази даних, зокрема MongoDB, зберігають дані у вигляді документів у форматі BSON, що забезпечує гнучкість схеми і горизонтальну масштабованість. Для туристичного планувальника MongoDB є природним вибором: схема даних може еволюціонувати разом із розвитком продукту, а структура документів добре відповідає об'єктній моделі JavaScript.

2.2. Огляд ключових технологій і фреймворків

Серед фреймворків для клієнтської розробки домінує тріада: React, Angular і Vue.js. React – бібліотека від Meta – реалізує компонентний підхід і використовує Virtual DOM для мінімізації маніпуляцій з реальним DOM-деревом. Angular від Google є комплексним фреймворком із вбудованими інструментами для маршрутизації, управління станом і тестування. Vue.js поєднує простоту освоєння з гнучкістю і є популярним для проєктів середнього масштабу. Для туристичного планувальника клієнтський JavaScript відповідає за динамічне оновлення контенту без перезавантаження сторінок.

Express.js – мінімалістичний Node.js-фреймворк – є де-факто стандартом для побудови RESTful API. Його основні переваги: незначний

overhead, велика гнучкість у виборі архітектурних рішень і багата екосистема middleware-компонентів. Express дозволяє організувати маршрутизацію через окремі Router-об'єкти, що сприяє модульності коду. Для туристичного планувальника Express.js є ідеальним вибором: він не нав'язує зайвої складності, при цьому надає всі необхідні можливості для серверної логіки.

Mongoose – ODM-бібліотека для роботи з MongoDB у Node.js-середовищі – дозволяє визначати схеми колекцій, додавати методи до моделей і виконувати валідацію даних на рівні застосунку. Рівень абстракції суттєво спрощує взаємодію з базою даних. Для туристичного планувальника Mongoose дозволяє чітко визначити схеми користувачів і планів подорожей, а також додати зручні методи безпосередньо до моделі користувача.

Handlebars – мова шаблонів для серверного рендерингу HTML-сторінок – дозволяє вбудовувати динамічні дані в HTML-структуру через прості вирази. Використання шаблонів Handlebars спрощує розділення логіки і представлення, а механізм partials дозволяє виносити повторювані елементи (header, footer) в окремі файли. Для туристичного планувальника Handlebars забезпечує серверний рендеринг стартових сторінок, що покращує початковий час завантаження.

Google Places API надає доступ до бази даних Google, що налічує понад 200 мільйонів точок інтересу по всьому світу. Text Search API дозволяє виконувати пошук місць за текстовим запитом і отримувати структуровані результати: назва, адреса, рейтинг, кількість відгуків, фотографія. Photo API надає можливість завантажувати фотографії за посиланнями з результатів пошуку. Для туристичного планувальника цей API є основним джерелом даних про туристичні об'єкти.

2.3. Бібліотеки та інструменти JavaScript для туристичного планувальника

Для роботи з картографічними даними широко застосовуються спеціалізовані бібліотеки. Leaflet – легковагова бібліотека з відкритим кодом – дозволяє інтегрувати інтерактивні карти на базі OpenStreetMap без комерційних ліцензій. Google Maps JavaScript API забезпечує доступ до повного функціоналу картографічного сервісу Google: маршрутизацію, Street View, геокодинг, Places Autocomplete. Mapbox SDK надає можливість створювати кастомні карти з унікальними стилями.

Для туристичного планувальника найбільш доцільним є поєднання Google Places API (для пошуку і даних про місця) та Google Maps JavaScript API (для відображення карти і побудови маршрутів). Такий вибір забезпечує синергію між пошуковим і картографічним компонентами, оскільки обидва API використовують єдину базу даних Google.

Управління сесіями реалізується через бібліотеку express-session у поєднанні з connect-mongodb-session, яка зберігає дані сесій у MongoDB. Такий підхід забезпечує персистентність: після перезапуску сервера користувачі залишаються авторизованими. Хешування паролів здійснюється за допомогою bcryptjs – JavaScript-реалізації алгоритму bcrypt, що є галузевим стандартом безпечного зберігання паролів. Bcrypt застосовує адаптивну функцію виведення ключа з налаштовуваним коефіцієнтом складності, що підвищує стійкість до brute-force атак.

Dotenv – бібліотека для управління змінними середовища – дозволяє зберігати конфіденційні налаштування (URI бази даних, ключі API, секрет сесії) в окремому файлі .env, який не додається до системи контролю версій. Це є стандартною практикою безпечного управління конфігурацією застосунку.

2.4. Принципи проєктування зручного користувацького інтерфейсу

Проєктування користувацького інтерфейсу (UI) і забезпечення якісного досвіду взаємодії (UX) є взаємопов'язаними і однаково важливими аспектами розробки. UX охоплює весь спектр відчуттів і вражень від взаємодії з продуктом, тоді як UI зосереджується на конкретних візуальних і інтерактивних елементах.

Процес UX-проєктування зазвичай починається з дослідження користувачів: інтерв'ю, опитувань, аналізу конкурентів. На основі зібраних даних формуються персони – узагальнені образи цільових користувачів – і визначаються ключові сценарії використання. Для туристичного планувальника основний сценарій: «Як мандрівник, я хочу знайти цікаві місця для відвідування в обраному місті і зберегти їх у своєму списку планів, щоб не забути під час поїздки».

Ієрархія інформації є фундаментальним принципом проєктування. Елементи, найважливіші для ключових завдань, повинні бути найпомітнішими. Принцип Hick's Law стверджує, що час прийняття рішення зростає зі збільшенням кількості варіантів, тому інтерфейс туристичного планувальника свідомо обмежує кількість одночасно відображуваних категорій.

Gestalt-принципи сприйняття – близькість, подібність, замкнутість – активно використовуються при проєктуванні карток місць. Елементи однієї картки групуються разом і візуально відокремлюються від сусідніх, що дозволяє користувачеві сприймати кожну картку як цілісну одиницю інформації.

Зворотний зв'язок є обов'язковим елементом якісного UX. Кожна дія повинна супроводжуватися видимою реакцією: при додаванні місця – підтверджувальне повідомлення, при видаленні – картка зникає без

перезавантаження, при пошуку – індикатор завантаження. Відсутність зворотного зв'язку породжує невизначеність і знижує довіру до продукту.

Figma є стандартним інструментом для UI-проєктування. Вона надає можливості для створення wireframe-макетів, інтерактивних прототипів і систем дизайну. Компонентна система Figma дозволяє визначити повторювані елементи один раз і перевикористовувати їх на всіх екранах, забезпечуючи узгодженість дизайну.

2.5. Аналіз вимог до веб-застосунку

Веб-застосунок для планування подорожей призначений для автоматизації процесів створення туристичних маршрутів, пошуку локацій, організації поїздок та збереження інформації про подорожі користувачів.

Основною метою системи є надання користувачам зручного інструменту для:

- створення маршрутів подорожей;
- пошуку туристичних місць;
- перегляду геолокацій;
- планування витрат;
- формування списків місць для відвідування;
- збереження історії подорожей.

До функціональних вимог системи належать:

- реєстрація та авторизація користувачів;
- створення та редагування маршрутів;
- інтеграція з Google Places API;
- відображення інформації про туристичні об’єкти;
- збереження планів подорожей;
- формування персоналізованих маршрутів;
- управління списками улюблених місць.

Нефункціональні вимоги:

- адаптивний інтерфейс;
- швидкодія системи;
- безпечне зберігання даних;
- масштабованість;
- кросбраузерність;
- підтримка мобільних пристроїв.

Для реалізації застосунку було обрано:

- Node.js та Express.js – серверна частина;
- MongoDB та Mongoose – база даних;
- Handlebars – шаблонізатор;
- HTML5 та CSS3 – інтерфейс користувача;
- Google Places API – геолокаційні сервіси.

2.6. Проєктування архітектури веб-застосунку

Архітектура системи побудована за клієнт-серверним принципом із використанням багаторівневого підходу.

Основними рівнями системи є:

1. Клієнтський рівень (Frontend);
2. Серверний рівень (Backend);
3. Рівень бізнес-логіки;
4. Рівень доступу до даних;
5. Зовнішні API-сервіси.

Архітектурна схема системи представлена на рис. 2.1.

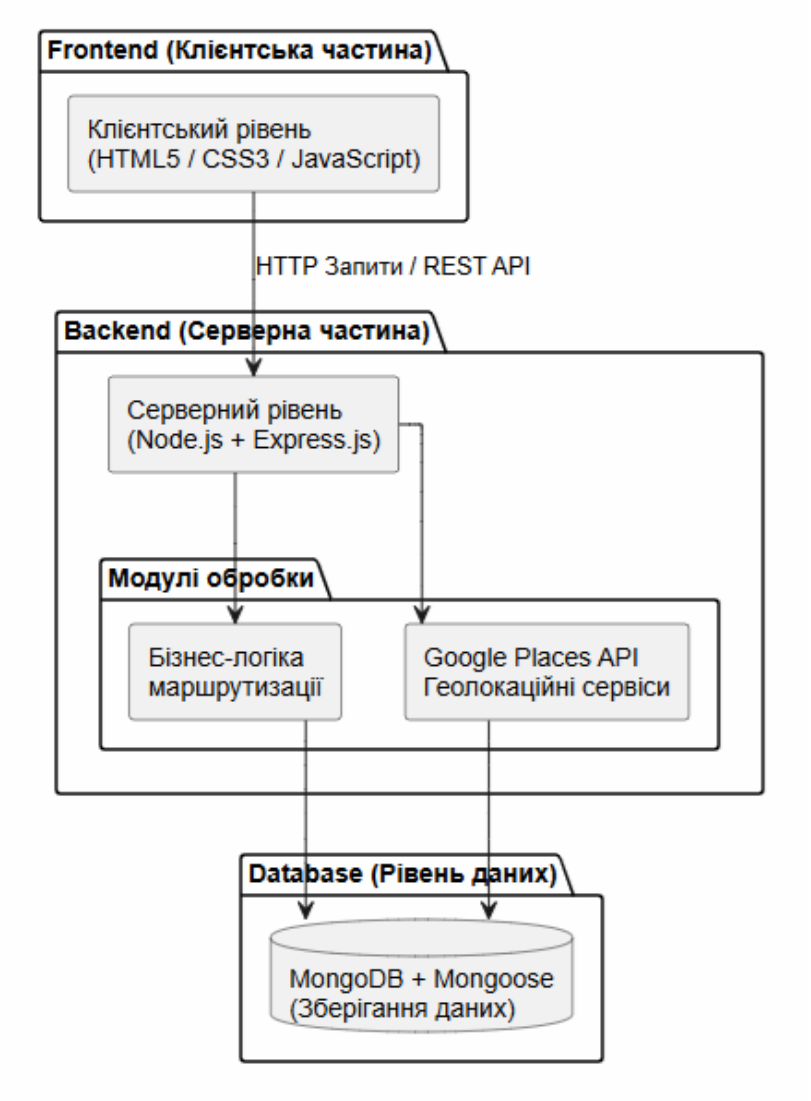

Рисунок 2.1. Архітектурна схема системи

Основними перевагами такої архітектури є:

- модульність;
- масштабованість;
- розділення відповідальності між компонентами;
- можливість інтеграції зовнішніх сервісів.

2.7. Проєктування структури бази даних

Для зберігання інформації використовується документоорієнтована база даних MongoDB.

Основними сутностями системи є:

- User – користувач;
- Trip – подорож;
- Destination – туристична локація;
- Route – маршрут;
- FavoritePlace – улюблені місця.

Структура колекцій MongoDB представлена на рис. 2.2.

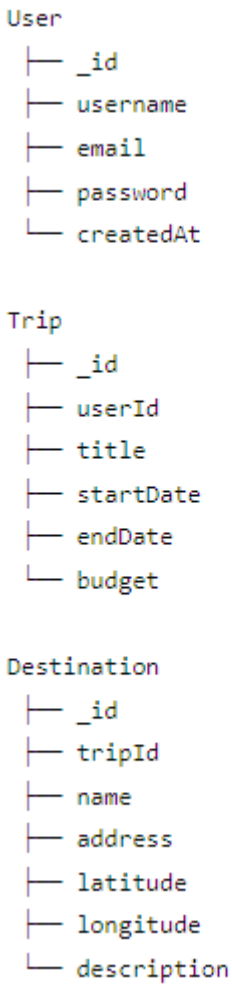

Рисунок 2.2. Архітектурна схема системи

ER-діаграма системи представлена на рис. 2.3.

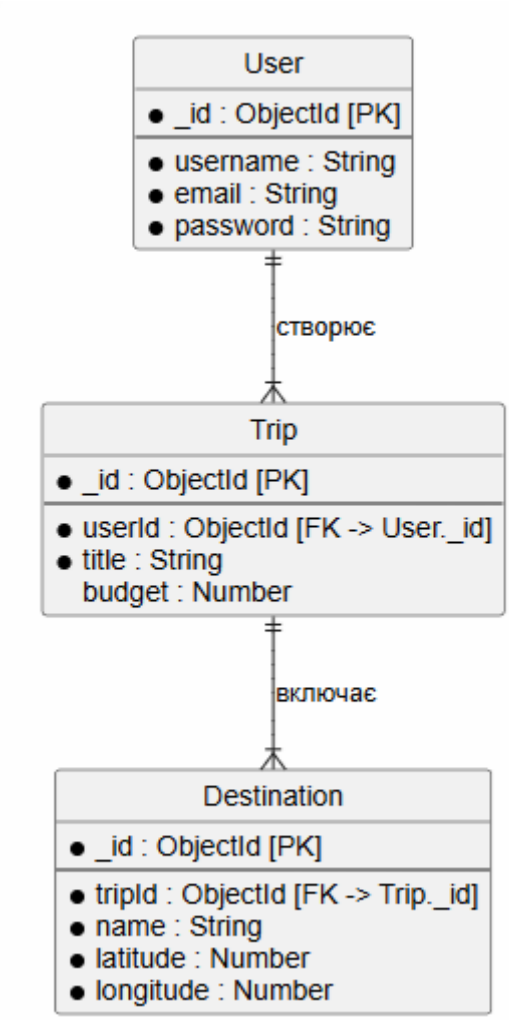

Рисунок 2.3. ER-діаграма

MongoDB забезпечує:

- гнучкість структури даних;
- швидку обробку JSON-документів;
- зручну інтеграцію з JavaScript-середовищем.

2.8. Проєктування серверної частини застосунку

Серверна частина реалізується на платформі Node.js із використанням фреймворку Express.js.

Основними задачами серверної частини є:

- обробка HTTP-запитів;
- маршрутизація;
- робота з базою даних;
- взаємодія з Google Places API;
- авторизація користувачів;
- генерація HTML-сторінок.

Структура серверної частини представлена на рис. 2.4.

```
server/
│
├── app.js
├── routes/
│   ├── authRoutes.js
│   ├── tripRoutes.js
│   └── apiRoutes.js
│
├── controllers/
│   ├── authController.js
│   ├── tripController.js
│   └── placesController.js
│
├── models/
│   ├── User.js
│   ├── Trip.js
│   └── Destination.js
│
├── middleware/
│
└── services/
```

Рисунок 2.4. Структура серверної частини веб-застосунку

Схема маршрутизації Express.js представлена на рис. 2.5.

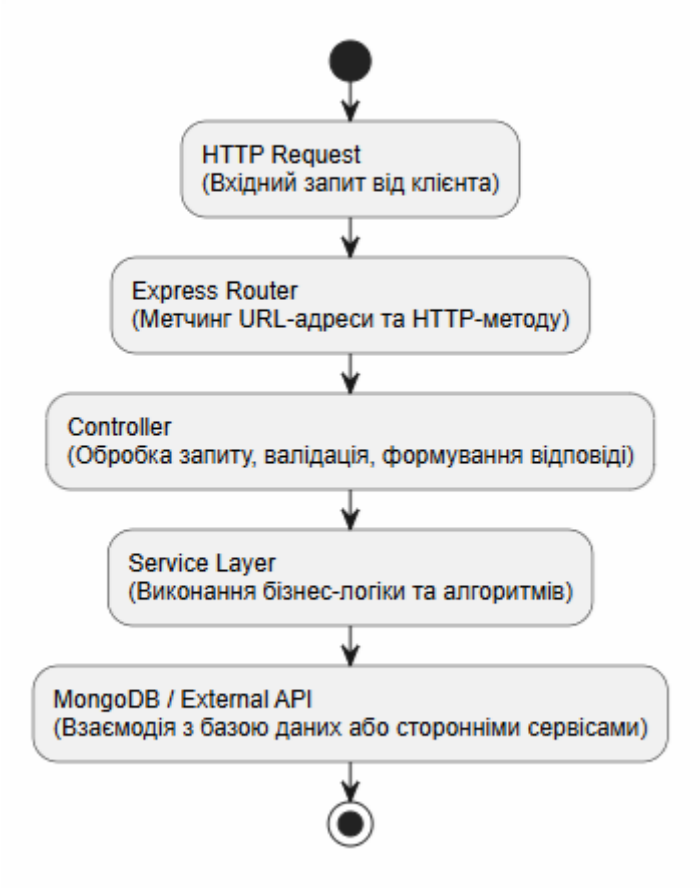

Рисунок 2.5. Схема маршрутизації Express.js

2.9. Проєктування клієнтської частини застосунку

Клієнтська частина веб-застосунку реалізується засобами:

- HTML5;
- CSS3;
- JavaScript;
- Handlebars.

Handlebars використовується для динамічного формування сторінок та шаблонізації інтерфейсу.

Основними сторінками системи є:

- головна сторінка;
- сторінка авторизації;

- сторінка створення маршруту;
- сторінка перегляду подорожі;
- сторінка карти та локацій.

Структура інтерфейсу користувача зображена на рис. 2.6.

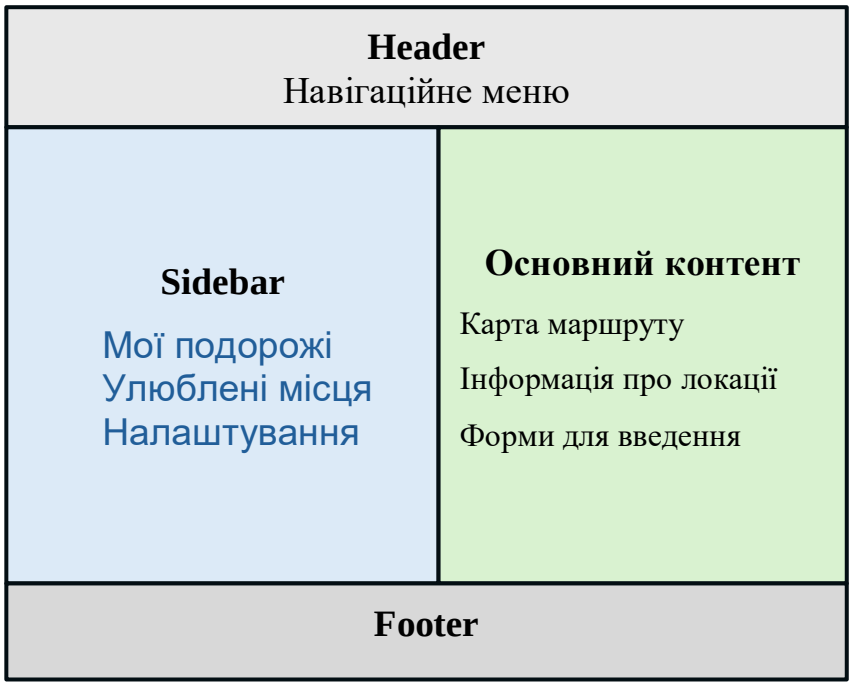

Рисунок 2.6. Структура інтерфейсу користувача

Інтерфейс адаптований для роботи на:

- персональних комп’ютерах;
- планшетах;
- мобільних пристроях.

2.10. Інтеграція Google Places API

Для отримання інформації про туристичні об’єкти використовується Google Places API.

API дозволяє:

- шукати локації;

- отримувати координати місць;
- відображати рейтинги;
- отримувати фотографії та адреси;
- будувати маршрути.

Схема взаємодії із Google Places API зображена на рис. 2.7.

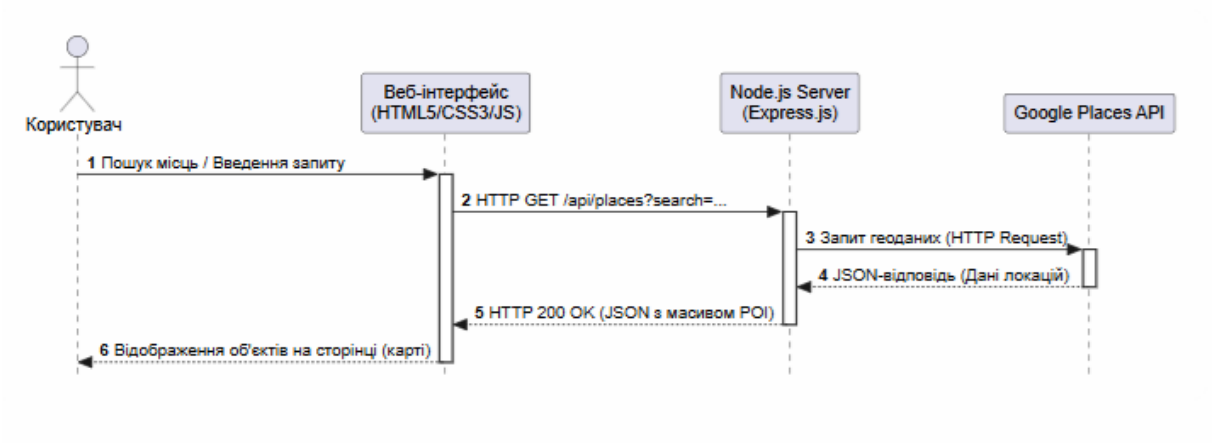

Рисунок 2.7. Схема взаємодії із Google Places API

Приклад взаємодії з API

```
const axios = require('axios');

const response = await axios.get(
  'https://maps.googleapis.com/maps/api/place/textsearch/json',
  {
    params: {
      query: 'restaurants in Kyiv',
      key: process.env.GOOGLE_API_KEY
    }
  }
);
```

2.11. UML-діаграми веб-застосунку

Діаграма варіантів використання представлена на рис. 2.8.

Передбачено три варіанти:

- створення маршруту;
- пошук місць;
- перегляд подорожей.

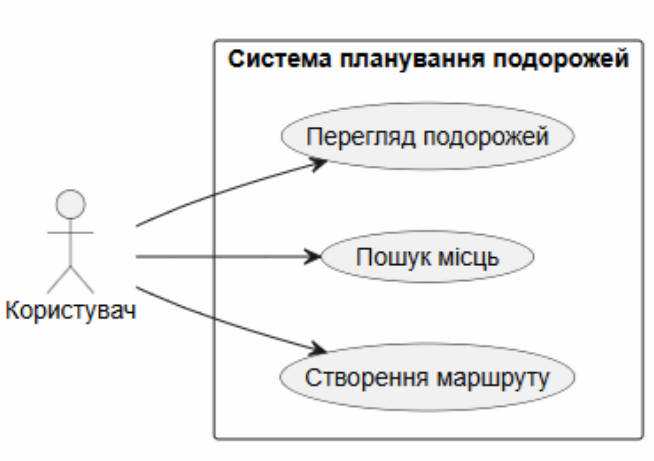

Рисунок 2.8. Діаграма варіантів використання

Діаграма компонентів представлена на рис. 2.9.

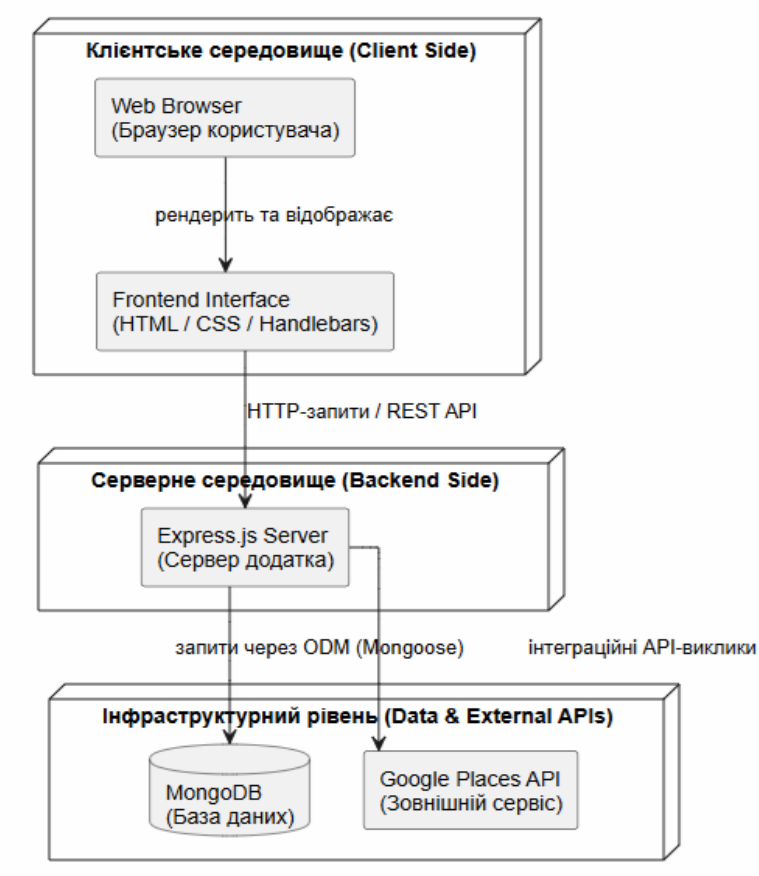

Рисунок 2.9. Діаграма компонентів

Діаграма послідовності представлена на рис. 2.10.

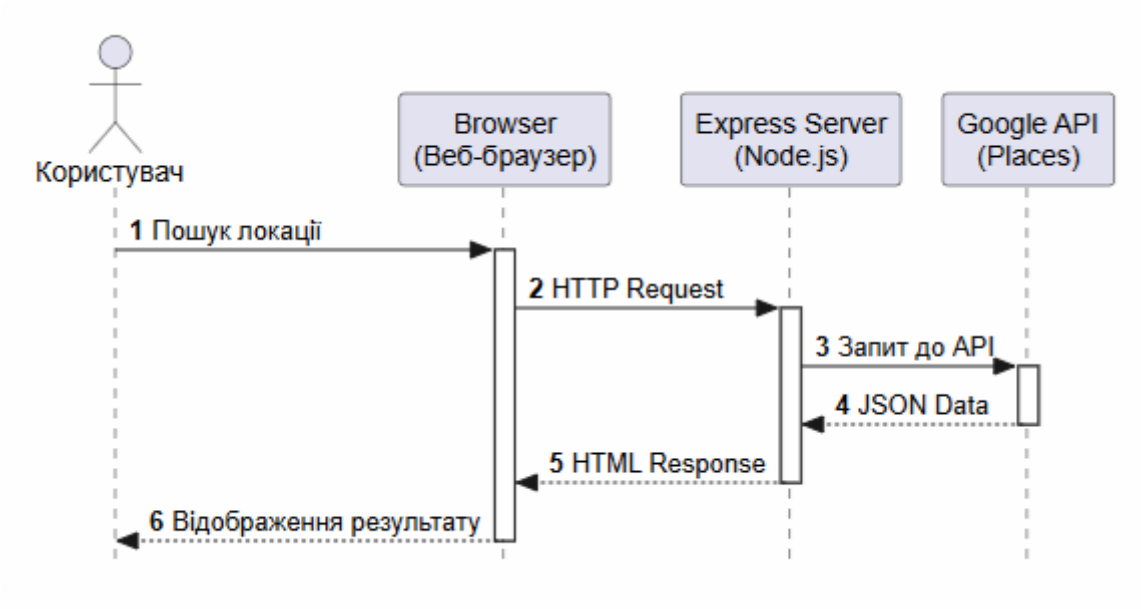

Рисунок 2.10. Діаграма послідовності

2.12. Проєктування безпеки системи

Для забезпечення безпеки веб-застосунку передбачено:

- авторизацію користувачів;
- хешування паролів;
- захист HTTP-сесій;
- валідацію введених даних;
- захист від XSS та CSRF-атак.

Для хешування паролів використовується бібліотека bcrypt.

Схема авторизації користувача представлена на рис. 2.11.

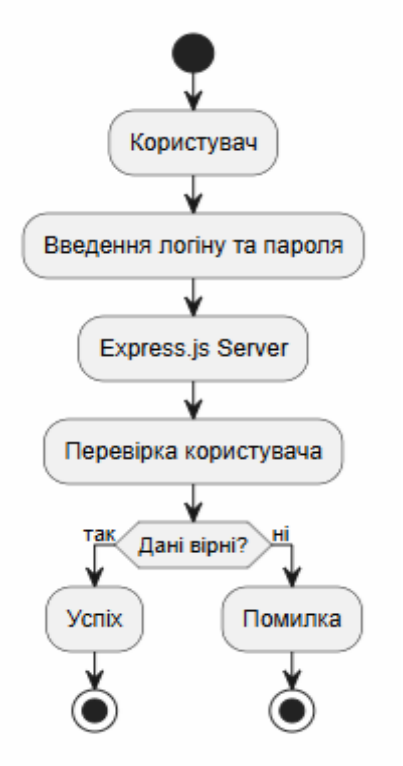

Рисунок 2.11. Схема авторизації користувача

Висновки до другого розділу

У процесі проєктування веб-застосунку було визначено архітектуру системи, структуру бази даних, модель взаємодії компонентів та принципи інтеграції із зовнішніми сервісами.

Було обґрунтовано вибір стеку технологій Node.js, Express.js, MongoDB, Mongoose, Handlebars, HTML5, CSS3 та Google Places API.

Запропонована архітектура забезпечує:

- модульність системи;
- масштабованість;
- ефективну взаємодію компонентів;
- зручність супроводу програмного забезпечення;
- підтримку сучасних веб-технологій.

3. ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБ-ЗАСТОСУНКУ

3.1. Загальна структура програмної реалізації веб-застосунку

Розроблений веб-застосунок для планування подорожей реалізовано за клієнт-серверною архітектурою із використанням середовища Node.js та фреймворку Express.js.

Основною метою програмної реалізації є забезпечення:

- зручного створення маршрутів подорожей;
- взаємодії із геолокаційними сервісами;
- збереження даних користувачів;
- відображення інтерактивного інтерфейсу;
- підтримки багатокористувацького режиму роботи.

Програмна система складається з таких основних модулів:

- серверний модуль;
- модуль маршрутизації;
- модуль роботи з базою даних;
- модуль інтеграції з Google Places API;
- модуль авторизації;
- модуль генерації HTML-сторінок;
- модуль клієнтської взаємодії.

Структура програмного проєкту представлена на рис. 3.1.

Основним файлом запуску системи є app.js, який виконує:

- ініціалізацію Express.js;
- налаштування middleware;
- підключення маршрутизації;
- запуск HTTP-сервера.

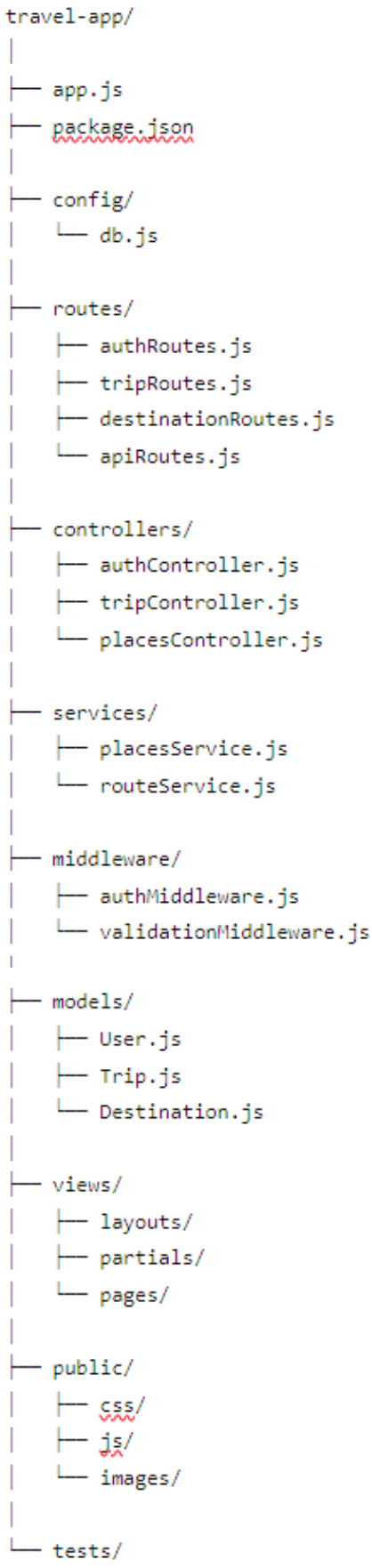

Рисунок 3.1. Структура програмного проєкту

3.2. Налаштування середовища розробки

Перед початком програмної реалізації необхідно підготувати середовище розробки і ініціалізувати проєкт. Для роботи з кодом використовується Visual Studio Code – редактор, що підтримує широкий спектр розширень для JavaScript і Node.js.

Node.js встановлюється з офіційного сайту nodejs.org. Після встановлення у системі доступні утиліти node і npm. Версія Node.js перевіряється командою node -v, npm – npm -v. Новий проєкт ініціалізується командою npm init -y, що автоматично створює файл package.json. Основні пакети встановлюються командою: npm install express mongoose express-session connect-mongodb-session dotenv bcryptjs express-handlebars. Пакет nodemon встановлюється як залежність розробки для автоматичного перезапуску сервера.

Структура файлів проєкту організована відповідно до принципу розділення відповідальностей: каталог routes містить файли маршрутизаторів, models – схеми і моделі Mongoose, middleware – функції проміжного програмного забезпечення, public – статичні файли (CSS, JavaScript, зображення), views – шаблони Handlebars. Головний файл index.js розміщується в корені проєкту і відповідає за ініціалізацію сервера і реєстрацію маршрутизаторів.

Файл .env розміщується в корені проєкту і містить конфіденційні налаштування: рядок підключення до MongoDB, ключ Google Places API, секрет для підписання сесій. Цей файл додається до .gitignore, щоб виключити потрапляння чутливих даних до публічного репозиторію.

3.3. Реалізація серверної частини застосунку та підключення бази даних

Серверна частина системи реалізована на платформі Node.js із використанням Express.js.

Основними задачами серверної частини є:

- обробка HTTP-запитів;
- маршрутизація;
- обробка бізнес-логіки;
- взаємодія з MongoDB;
- інтеграція з Google Places API;
- генерація HTML-відповідей.

Ініціалізація Express.js

```
const express = require('express');
const app = express();

app.use(express.json());
app.use(express.urlencoded({ extended: true }));

app.listen(3000, () => {
    console.log('Server started');
});
```

Підключення маршрутизації

```
const tripRoutes = require('./routes/tripRoutes');

app.use('/trips', tripRoutes);
```

Express.js забезпечує:

- підтримку REST-архітектури;
- модульність маршрутизації;
- підтримку middleware;
- обробку HTTP-запитів.

Серверна частина застосунку побудована на основі Express.js і організована за триступеневою архітектурою: рівень маршрутизації (routes), рівень бізнес-логіки і рівень доступу до даних (моделі Mongoose). Взаємодія між клієнтом і сервером здійснюється через HTTP-протокол.

Ініціалізація сервера здійснюється у файлі index.js. Спочатку імпортуються необхідні модулі: express, mongoose, express-session, connect-

mongodb-session і dotenv. Потім налаштовується сервер: підключається парсер тіла запитів, Handlebars як рушій шаблонів, налаштовується обробник сесій із MongoDB-сховищем. Підключення до MongoDB виконується асинхронно перед запуском HTTP-сервера: якщо підключення не вдалося, сервер не запускається.

Схема користувача визначена у файлі models/user.js. Вона містить поля: name (ім'я користувача), email (унікальний ідентифікатор облікового запису), password (хеш пароля) і plans (масив об'єктів із полями name, address і image). Модель також містить методи addToPlan і removePlan, що реалізують бізнес-логіку управління планами.

Маршрутизатор авторизації обробляє п'ять ендпоінтів: відображення і обробку форм реєстрації і входу, а також вихід із системи. При реєстрації перевіряється унікальність email; якщо користувач вже існує, відображається повідомлення. Пароль хешується за допомогою bcrypt.hash() із коефіцієнтом складності 10 перед збереженням. При вході хеш зберігається в базі і порівнюється із введеним паролем через bcrypt.compare().

Маршрутизатор головної сторінки обробляє запити до основного функціоналу: відображення головної сторінки, пошук місць через Google Places API, додавання місця до плану, видалення місця і відображення сторінки «Мої плани». Запит до Google Places Text Search API формується шляхом об'єднання назви міста і категорії; відповідь API нормалізується перед відправленням клієнту.

Middleware автентифікації перевіряє наявність і дійсність сесії для захищених маршрутів. Якщо сесія відсутня, запит перенаправляється на сторінку входу. Такий механізм забезпечує, що персональні дані доступні лише авторизованим користувачам. Структура бази даних включає три основні сутності: користувачі (User), що зберігаються в MongoDB і містять поля особистих даних і масив планів; плани подорожей (Plans), вбудовані у

документ користувача у вигляді піддокументів; зовнішні дані Google Places API, що не зберігаються в базі, а отримуються динамічно при кожному пошуковому запиті.

3.4. Реалізація протоколів взаємодії

Для взаємодії між клієнтською та серверною частинами використовується протокол HTTP/HTTPS.

Система підтримує такі типи HTTP-запитів:

- GET – отримання даних;
- POST – створення записів;
- PUT – оновлення інформації;
- DELETE – видалення даних.

Схема взаємодії клієнта та сервера представлена на рис. 3.2.

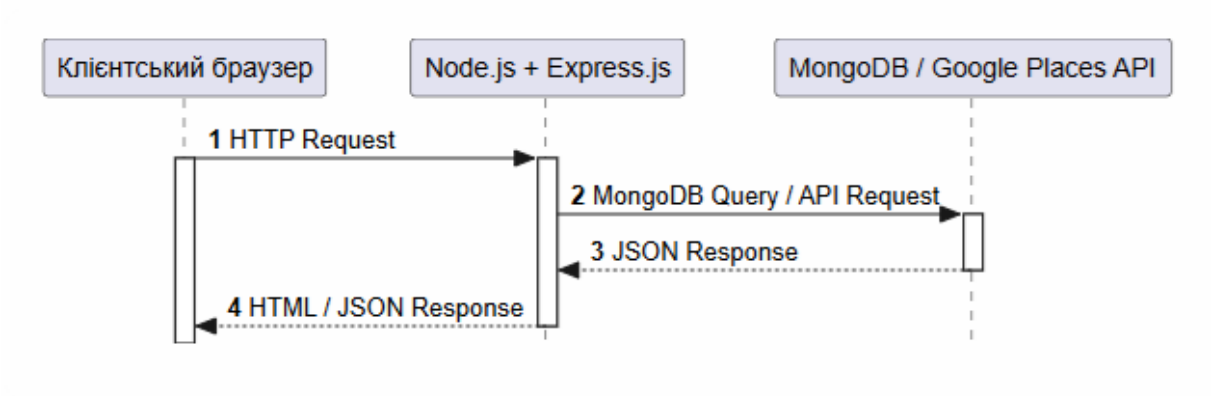

Рисунок 3.2. Схема взаємодії клієнта та сервера

Приклад REST API-запиту представлено на рис. 3.3.

```
router.post('/create', async (req, res) => {
    const trip = new Trip(req.body);

    await trip.save();

    res.redirect('/trips');
});
```

Рисунок 3.3. Приклад REST API-запиту

Формат JSON-відповіді представлено на рис. 3.4.

```
{
  "success": true,
  "message": "Trip created successfully"
}
```

Рисунок 3.4. Схема взаємодії клієнта та сервера

Для забезпечення безпечної передачі даних передбачено використання HTTPS.

3.5. Реалізація взаємодії з MongoDB

Для збереження інформації використовується MongoDB разом із бібліотекою Mongoose.

Mongoose забезпечує:

- створення схем даних;
- валідацію;
- ORM-подібну взаємодію;
- підтримку асинхронних операцій.

Підключення до бази даних зображено на рис. 3.5.

```
const mongoose = require('mongoose');

mongoose.connect(process.env.MONGO_URI)
    .then(() => console.log('MongoDB connected'));
```

Рисунок 3.5. Підключення до бази даних

Реалізація моделі користувача представлена на рис. 3.6.

```
const mongoose = require('mongoose');

const UserSchema = new mongoose.Schema({
    username: String,
    email: String,
    password: String
});
```

Рисунок 3.6. Реалізація моделі користувача

Реалізація моделі подорожі представлена на рис. 3.7.

```
const TripSchema = new mongoose.Schema({
    title: String,
    destination: String,
    startDate: Date,
    endDate: Date,
    budget: Number
});
```

Рисунок 3.7. Реалізація моделі подорожі

3.6. Реалізація клієнтської частини застосунку

Клієнтська частина застосунку реалізована засобами:

- HTML5;
- CSS3;
- JavaScript;
- Handlebars.

Handlebars використовується для генерації динамічних HTML-сторінок.

Структура шаблонів Handlebars представлена на рис. 3.8.

```
views/
│
├── layouts/
│   └── main.hbs
│
├── partials/
│   ├── header.hbs
│   └── footer.hbs
│
└── pages/
    ├── home.hbs
    ├── trips.hbs
    └── login.hbs
```

Рисунок 3.8. Структура шаблонів Handlebars

Приклад шаблону Handlebars

```
<h1>{{title}}</h1>

<ul>
  {{#each trips}}
    <li>{{this.destination}}</li>
  {{/each}}
</ul>
```

Реалізація стилізації CSS3

```
.container {
    max-width: 1200px;
    margin: 0 auto;
}

.trip-card {
    border: 1px solid #ccc;
    padding: 20px;
}
```

Інтерфейс застосунку формується на основі шаблонного рушія Handlebars. Базовий макет визначає загальну структуру HTML-документа і підключає спільні частини: head (мета-теги, підключення шрифтів і стилів),

header (навігаційне меню) і footer. Змінний контент кожної сторінки вставляється через механізм {{{body}}}.

Кожна сторінка має власний шаблон Handlebars. Шаблони можуть отримувати дані від сервера через об'єкт контексту, що передається функції res.render(). Для відображення списку планів використовується вбудований хелпер #each, що ітерує масив об'єктів і генерує картку для кожного елемента.

CSS-стилізація організована в єдиному файлі. Глобальні правила визначають нульові відступи для всіх елементів (box-sizing: border-box), шрифт Roboto як основний, темно-сірий колір тексту (#333) і міжрядковий інтервал 1.6. Фон сторінки реалізований через background-image з властивістю background-attachment: fixed, що створює ефект «паралакса». Заголовок є фіксованим (position: fixed) і завжди залишається видимим при прокручуванні.

Сітка карток реалізована засобами Flexbox: контейнер .cards має display: flex, flex-wrap: wrap і gap між картками. Кожна картка .card має фіксовану ширину 200px, білий фон і тінь box-shadow. Зображення обрізається до фіксованої висоти 150px через object-fit: cover. Для зображень, що не вдалося завантажити, через атрибут onerror встановлюється placeholder-картинка.

Адаптивність для мобільних пристроїв реалізована через медіа-запит @media (max-width: 768px). На малих екранах сітка карток переходить до вертикального розподілу (flex-direction: column), картки розтягуються на всю ширину (width: 100%), а панель вибору категорій стає горизонтально прокручуваним рядком (overflow-x: auto). Це забезпечує зручне використання на смартфонах.

Клієнтський JavaScript відповідає за динамічну поведінку інтерфейсу. При відправленні форми пошуку стандартна поведінка перезавантаження пригнічується через e.preventDefault(). Натомість викликається функція

searchPlaces(), яка формує URL запиту до серверного ендпоінту і надсилає GET-запит через Fetch API. Отримана відповідь JSON обробляється: для кожного результату динамічно створюється DOM-вузол картки і додається до відповідного контейнера. Додавання місця до плану і видалення реалізовані аналогічно – через Fetch API без перезавантаження сторінки.

Функція автодоповнення при введенні назви міста реалізована на основі відфільтрованого списку міст. При кожному введенні символу відповідні варіанти відображаються у випадаючому переліку під полем введення. При виборі варіанту поле заповнюється і список приховується. Перемикання між вкладками категорій реалізовано через обробник події click, що додає і видаляє клас active на кнопках і відповідних панелях вмісту.

3.7. Реалізація алгоритмів планування подорожей

Одним із ключових компонентів системи є алгоритм формування маршруту подорожі.

Основні етапи алгоритму:

1. Отримання даних користувача;
2. Пошук локацій через Google Places API;
3. Формування списку місць;
4. Розрахунок маршруту;
5. Збереження результатів.

Алгоритм формування маршруту представлено на рис. 3.9.

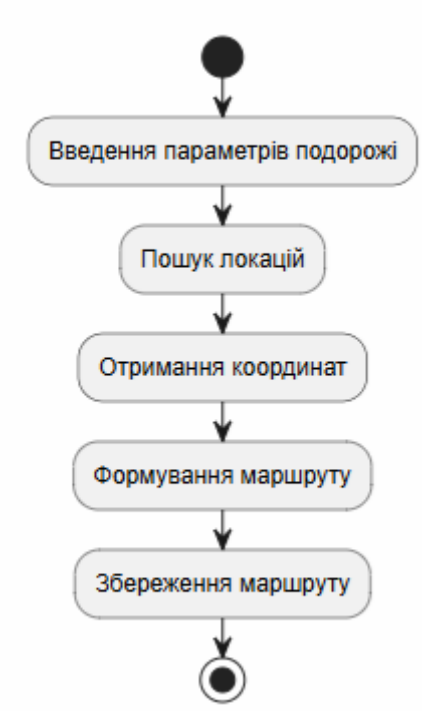

Рисунок 3.9. Алгоритм формування маршруту

Приклад пошуку локацій:

```
const axios = require('axios');

async function searchPlaces(query) {
    const response = await axios.get(
        'https://maps.googleapis.com/maps/api/place/textsearch/json',
        {
            params: {
                query,
                key: process.env.GOOGLE_API_KEY
            }
        }
    );

    return response.data.results;
}
```

3.8. Реалізація авторизації користувачів

Для забезпечення контролю доступу реалізовано механізм авторизації користувачів.

Функціонал включає:

- реєстрацію;
- вхід у систему;
- хешування паролів;
- підтримку сесій.

Реалізація хешування пароля передбачена наступним чином:

```
const bcrypt = require('bcrypt');

const hashedPassword = await bcrypt.hash(password, 10);
```

Перевірка користувача виконується наступним чином:

```
const isValid = await bcrypt.compare(
    password,
    user.password
);
```

Схема авторизації зображена на рис. 3.10.

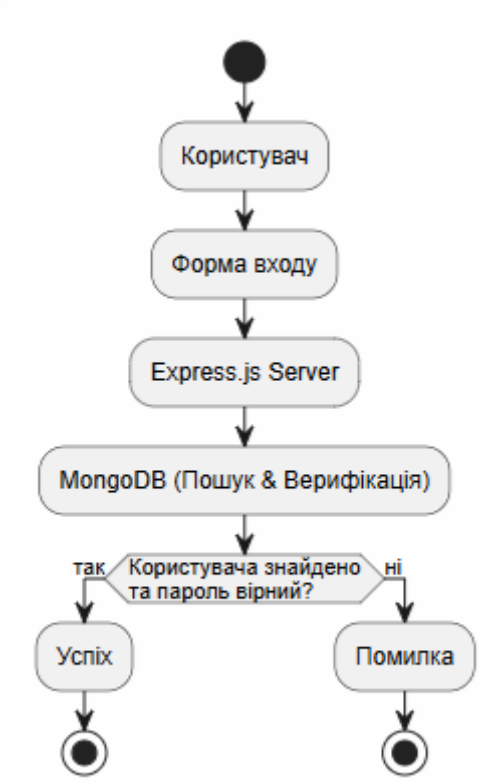

Рисунок 3.10. Схема авторизації користувача

3.9. Реалізація інтеграції з Google Places API

Google Places API використовується для:

- пошуку туристичних місць;
- отримання координат;
- відображення рейтингів;
- отримання описів локацій.

Схема інтеграції з Google Places API представлена на рис. 3.11.

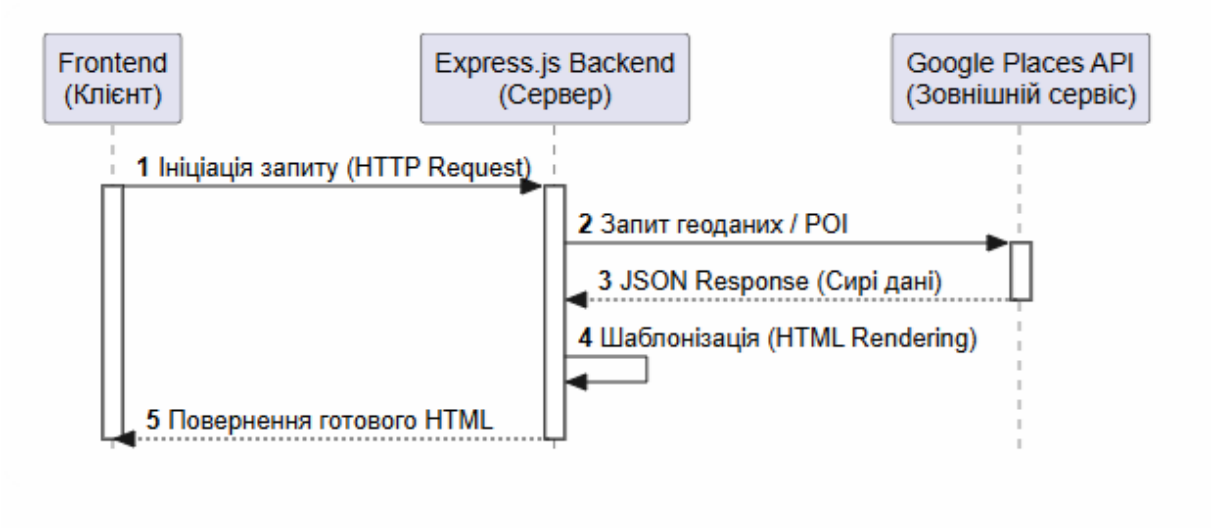

Рисунок 3.11. Схема інтеграції з Google Places API

Приклад API-запиту

```
const response = await axios.get(
  'https://maps.googleapis.com/maps/api/place/nearbysearch/json',
  {
      params: {
          location: '50.4501,30.5234',
          radius: 5000,
          key: process.env.GOOGLE_API_KEY
      }
  }
);
```

3.10. Прототипування інтерфейсу планувальника подорожей

Прототипування є невід'ємним етапом розробки, що передує написанню програмного коду. Воно дозволяє виявити і усунути проблеми у логіці взаємодії на значно ранньому і менш витратному етапі. Для туристичного планувальника прототипування охопило чотири ключові екрани: головну сторінку з пошуком, сторінку «Мої плани», форму реєстрації і форму входу.

Центральним елементом головної сторінки є блок пошуку туристичних об'єктів. Він містить текстове поле для введення назви міста з

механізмом автодоповнення, що відображає список пропозицій у вигляді випадаючого переліку під час набору тексту. Нижче розташовані кнопки-вкладки для вибору категорії: «Готелі», «Музеї», «Пам'ятники», «Торгові центри» і «Клуби». Активна вкладка візуально виділяється кольором. Результати пошуку відображаються у вигляді сітки карток нижче панелі вибору категорій.

Картка місця є базовим елементом UI і містить: фотографію об'єкта (або placeholder-зображення), назву (жирним шрифтом), скорочену адресу (два перших компоненти), рейтинг із кількістю відгуків і кнопку «+ в план» для додавання до персонального списку. Дизайн картки є умисно лаконічним: зайва інформація виключена, щоб користувач міг швидко переглянути декілька варіантів і прийняти рішення.

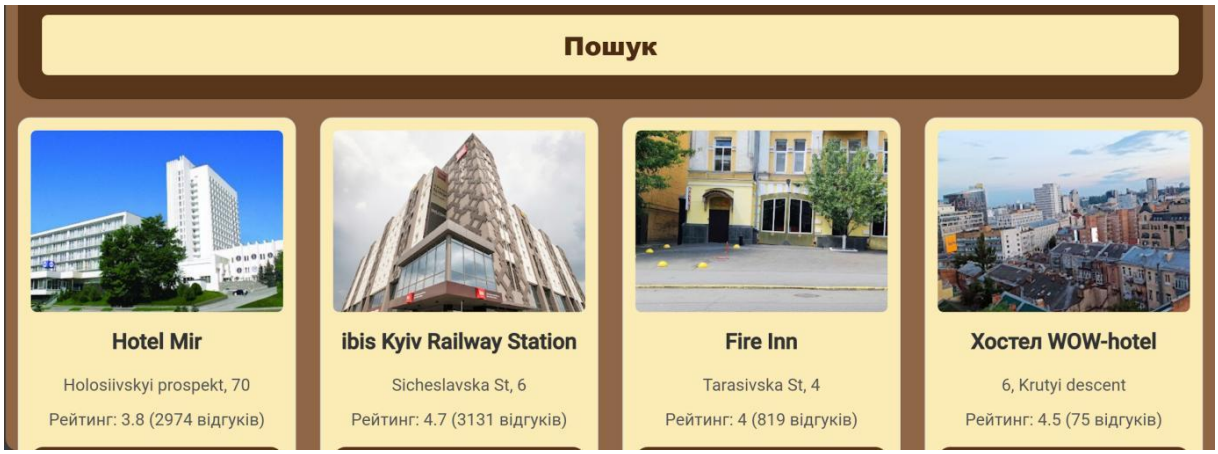

Рисунок 3.12. Картка житла

Сторінка «Мої плани» відображає персональний список збережених місць конкретного користувача. Кожен елемент представлений карткою з фото, назвою, адресою і двома кнопками: «Видалити» (для асинхронного видалення без перезавантаження) і «Побудувати маршрут» (для відкриття Google Maps). Якщо список порожній, відображається відповідне повідомлення із закликом почати пошук.

Форми реєстрації і входу мають мінімалістичний вигляд: заголовок, мічки полів, поля введення і кнопка підтвердження. Форма реєстрації включає поля «Ім'я», «Email» і «Пароль», форма входу – «Email» і «Пароль». Під кнопкою розміщено посилання для переходу між формами.

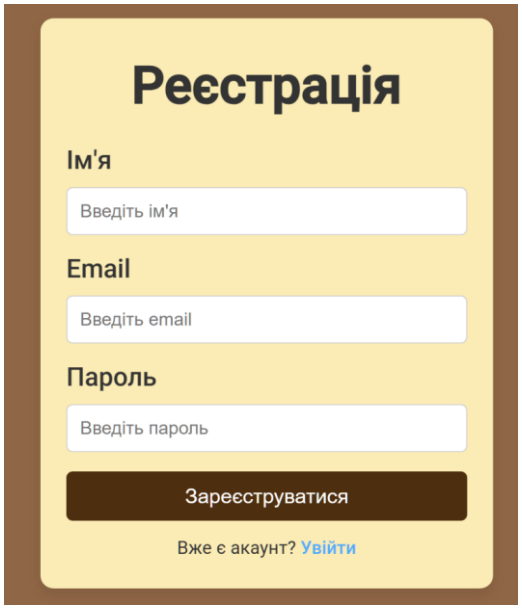

Рисунок 3.13. Форма реєстрації

Навігаційне меню присутнє на всіх сторінках і містить посилання «Головна» та «Вхід» (для неавторизованих) або «Головна», «Мої плани» і «Вихід» (для авторизованих). Меню реалізовано у вигляді фіксованого горизонтального заголовку, що залишається видимим під час прокручування сторінки.

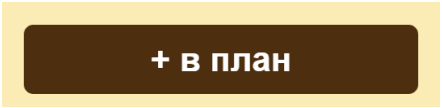

Рисунок 3.14. Макет кнопки

3.11. Розробка візуального дизайну засобами Figma

Візуальний дизайн застосунку розроблено з дотриманням принципу смислової єдності: кольорова палітра, типографіка та характер елементів

інтерфейсу формують цілісний образ, що асоціюється з природністю, затишком і надійністю.

Кольорова схема базується на теплих відтінках. Основний фон карток і форм – м'який жовтувато-бежевий (#fbe4ad, #fbecb5) – створює відчуття затишку. Фон навігаційного меню і акцентних елементів – насичений темно-коричневий (#4c2c12) – забезпечує чіткий контраст. Інтерактивні кнопки у стані hover переходять до темно-червоного (#f04e4e), що сигналізує про активну дію. Текст на світлому фоні – темно-сірий (#333333) – забезпечує комфортний рівень контрастності.

Типографіка застосунку включає два шрифти. Roboto (вага 400 і 500) використовується для основного тексту: описів, адрес, рейтингів, підписів до полів форм. Цей шрифт вирізняється нейтральністю і відмінною читабельністю. Poppins (вага 600) застосовується для заголовків і виділених елементів – він має геометричну структуру і створює виразний візуальний акцент. Міжрядковий інтервал 1.6 і кегль основного тексту 14–16px забезпечують зручне читання на всіх екранах.

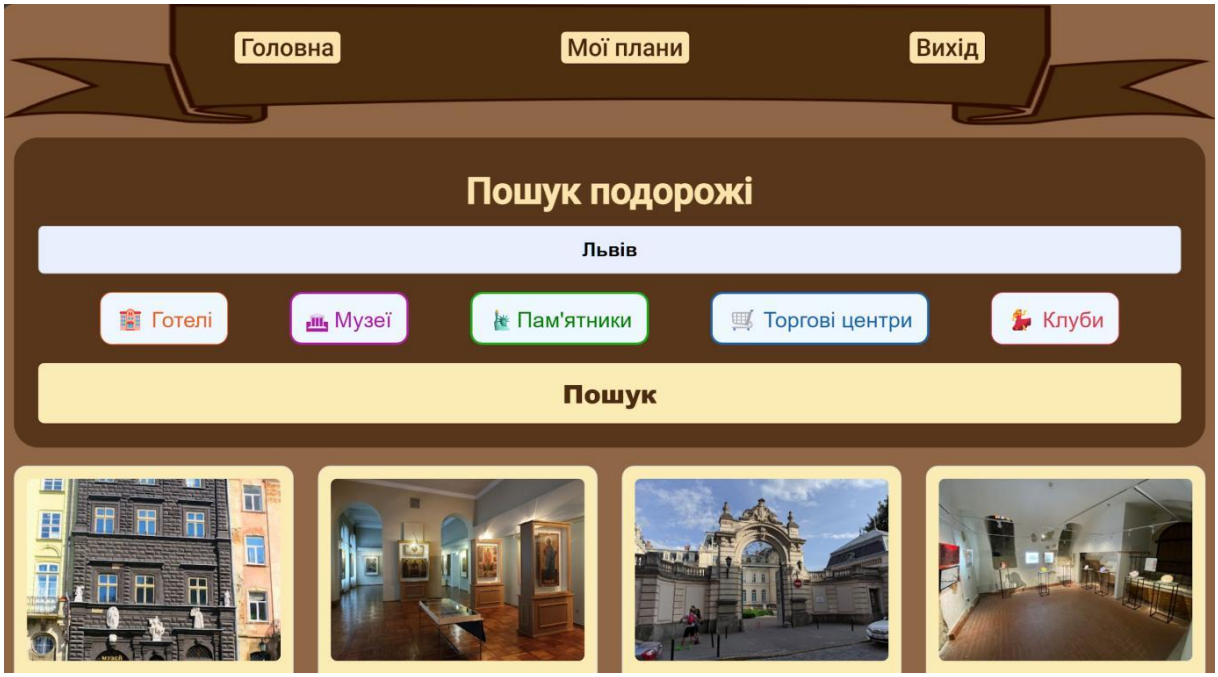

Рисунок 3.15. Макет головної сторінки

Макет головної сторінки організовано за принципом вертикальної прокрутки: фіксований заголовок з навігацією, блок пошуку по центру, сітка карток нижче. Блок пошуку виділяється напівпрозорим фоновим прямокутником з заокругленими кутами. Картки результатів мають однакову ширину і вирівняні за сіткою.

Кнопки розділені на два типи: первинні (btn-primary) – заповнені кольором, для головних дій (пошук, реєстрація, вхід) – і вторинні (btn-secondary) – з кольоровою рамкою і прозорим фоном, для допоміжних дій (вибір категорії, «+ в план»). Такий поділ допомагає користувачеві миттєво визначити ключову дію на кожному екрані.

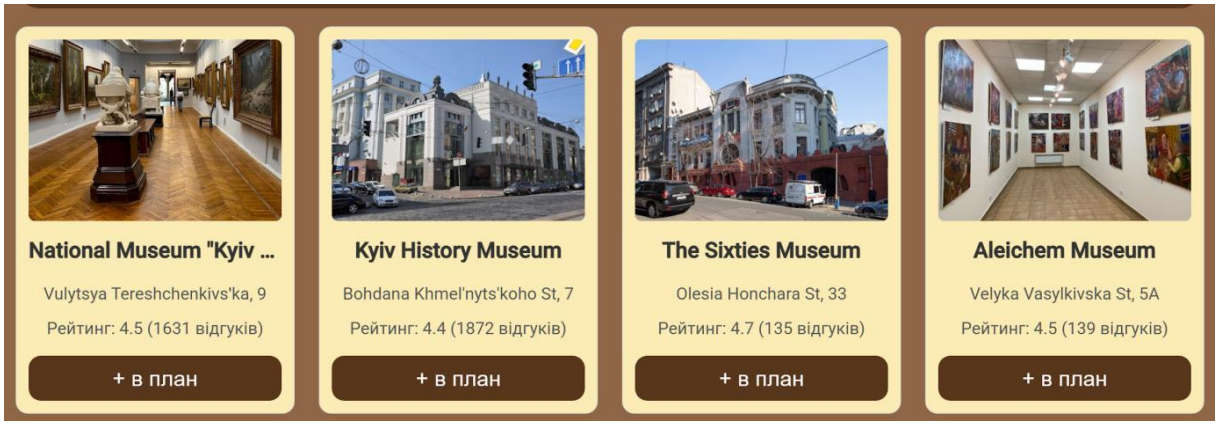

Рисунок 3.16. Макет карток з результатами пошуку

Сторінка «Мої плани» побудована за принципом карткового списку: кожен збережений об'єкт відображається карткою з фото, назвою, адресою і двома кнопками управління. При порожньому списку відображається спонукальне повідомлення. Форми входу і реєстрації розміщено по центру екрана в окремих контейнерах із чіткою ієрархією елементів.

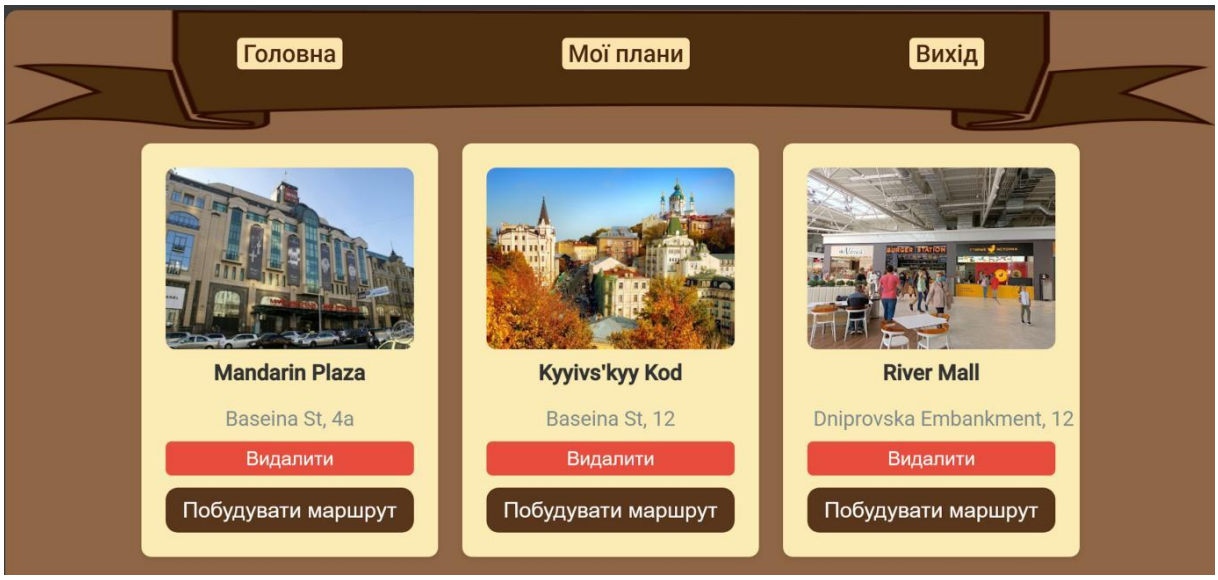

Рисунок 3.17. Макет сторінки «Мої плани»

Рисунок 3.18. Мапа веб-застосунку

Висновки до третього розділу

У процесі розроблення веб-застосунку було реалізовано серверну та клієнтську частини системи, механізми взаємодії між компонентами, інтеграцію із зовнішніми API та систему збереження даних.

Було розроблено:

- REST-архітектуру серверної частини;
- структуру MongoDB-моделей;
- шаблони Handlebars;

- алгоритми планування маршрутів;
- механізми авторизації;
- інтеграцію з Google Places API.

РОЗДІЛ 4. ТЕСТУВАННЯ ТА ВПРОВАДЖЕННЯ ВЕБ-ЗАСТОСУНКУ

4.1. Стратегія та методика тестування програмного забезпечення

Тестування розробленого веб-застосунку для планування подорожей є критично важливим етапом життєвого циклу розробки, спрямованим на виявлення дефектів, перевірку відповідності реалізованого функціоналу сформованим технічним вимогам та оцінку загальної надійності системи. Для забезпечення високої якості програмного продукту було обрано комплексну стратегію тестування, яка поєднує різні рівні та методи контролю якості.

Тестування системи проводилось для перевірки:

- коректності роботи серверної частини;
- правильності маршрутизації;
- взаємодії з MongoDB;
- роботи API;
- коректності інтерфейсу користувача.

Процес верифікації веб-застосунку базувався на концепції піраміди тестування та охоплював такі рівні:

- Модульне тестування (Unit Testing) – ізольована перевірка коректності роботи окремих функцій, утиліт та методів бізнес-логіки без залучення зовнішніх залежностей (бази даних чи сторонніх API);
- Інтеграційне тестування (Integration Testing) – перевірка правильної взаємодії між різними компонентами системи, зокрема коректності виконання запитів серверною частиною (Express.js) до бази даних (MongoDB) та обробки відповідей від Google Places API;
- Функціональне та системне тестування (End-to-End / System Testing) – валідація поведінки веб-застосунку як єдиного цілого з позиції кінцевого користувача, що покриває основні користувацькі сценарії;

- Тестування інтерфейсу та зручності використання (Usability Testing) – оцінка адаптивності інтерфейсу, ергономіки навігації та відповідності розроблених сторінок Handlebars створеним у Figma дизайн-макетам.

Основним підходом до тестування модулів і логіки було обрано тестування «чорної скриньки» (Black-box testing) для перевірки функціональних специфікацій та тестування «білої скриньки» (White-box testing) під час написання автоматизованих юніт-тестів для перевірки внутрішньої структури коду.

4.2. Модульне та інтеграційне тестування компонентів

Для автоматизації процесу модульного та інтеграційного тестування серверної частини Node.js було використано фреймворк Jest у зв'язці з бібліотекою Supertest, яка дозволяє імітувати HTTP-запити до ендпоінтів Express.js без запуску реального мережевого сервера.

4.2.1. Модульне тестування бізнес-логіки та моделей

Об'єктом модульного тестування стали функції валідації вхідних даних користувачів, модулі обчислення параметрів маршруту (наприклад, сортування локацій за днями подорожі) та методи моделей Mongoose.

Нижче наведено фрагмент автоматизованого тест-кейсу для перевірки функції створення нового об'єкта подорожі у базі даних за допомогою Jest та Mongoose (рис. 4.1).

```
const mongoose = require('mongoose');
const { Trip } = require('../models/Trip');

describe('Модульне тестування моделі Trip', () => {
  it('повинно успішно створити об\'єкт подорожі з коректними полями', () => {
    const tripData = {
      title: 'Відпустка у Карпатах',
      destination: 'Яремче',
      startDate: new Date('2026-07-01'),
      endDate: new Date('2026-07-07'),
      locations: []
    };

    const trip = new Trip(tripData);
    expect(trip.title).toBe(tripData.title);
    expect(trip.destination).toBe(tripData.destination);
    expect(trip.locations).toEqual([]);
  });

  it('повинно повернути помилку валідації, якщо обов\'язкове поле title відсутнє', () => {
    const trip = new Trip({ destination: 'Львів' });
    const err = trip.validateSync();
    expect(err.errors.title).toBeDefined();
  });
});
```

Рисунок 4.1. Автоматизований тест-кейс

Модульне тестування проводилось наступним чином:

```
test('Create trip object', () => {
    const trip = {
        title: 'Vacation',
        budget: 1000
    };

    expect(trip.title).toBe('Vacation');
});
```

4.2.2. Інтеграційне тестування REST API та взаємодії з MongoDB

Інтеграційні тести були спрямовані на верифікацію ланцюжка «Клієнтський запит $\rightarrow$ Контролер Express.js $\rightarrow$ СУБД MongoDB». Для уникнення засмічення продуктової бази даних під час тестів використовувалася локальна база даних пам'яті (In-memory DB).

Перевірці підлягали такі ключові компоненти:

1. *Маршрути авторизації* **(**/auth/register**,** /auth/login**):** тестування коректності хешування паролів за допомогою bcrypt, створення сесій та перенаправлення користувача на головну сторінку.

2. *Маршрути управління маршрутами подорожей* **(**/trips**):** перевірка операцій CRUD (створення, читання, оновлення, видалення записів подорожей у MongoDB).

За результатами запуску тестового покриття за допомогою вбудованого інструменту Jest було досягнуто 78% покриття ліній коду (Code Coverage), що свідчить про високий ступінь надійності серверної бізнес-логіки.

Під час інтеграційного тестування перевірялися:

- взаємодія Express.js із MongoDB;
- робота Google Places API;
- маршрутизація HTTP-запитів.

Результати інтеграційного тестування представлені в табл. 4.1.

Таблиця 4.1. Результати інтеграційного тестування

№	**Функція**	**Очікуваний результат**	**Результат**
1	Реєстрація користувача	Створення акаунта	Успішно
2	Авторизація	Вхід у систему	Успішно
3	Створення маршруту	Збереження подорожі	Успішно
4	Пошук локацій	Отримання результатів API	Успішно
5	Відображення сторінки	Коректний рендеринг	Успішно

4.3. Функціональне тестування та сценарії валідації

Функціональне тестування проводилося шляхом ручного та напівавтоматизованого виконання тест-кейсів, розроблених на основі прецедентів використання системи (Use Cases).

Метою було підтвердження того, що користувач може безперешкодно пройти всі етапи планування поїздки.

Для проведення випробувань було сформовано матрицю тест-кейсів, результати виконання яких наведено в табл. 4.2.

Усі критичні дефекти, виявлені під час функціонального тестування (наприклад, помилка парсингу відповідей від Google API при відсутності фотографії об'єкта), були успішно усунені на етапі налагодження коду.

4.4. Тестування користувацького інтерфейсу та адаптивності

Оскільки сучасні туристи найчастіше використовують мобільні пристрої безпосередньо під час подорожей, особлива увага при проєктуванні та розробці приділялася кросбраузерності та адаптивності інтерфейсу.

Тестування UI/UX включало два етапи:

1. *Валідація верстки та адаптивності*. Перевірка коректності відображення шаблонів Handlebars, форм, навігаційних меню та карток туристичних об'єктів на різних роздільних здатностях екрана за допомогою інструментів розробника Google Chrome DevTools (Responsive Design Mode). Тестування проводилося для таких роздільних здатностей:

- Настільні ПК (Desktop): 1920×1080, 1440×900;
- Планшети (Tablet): 768×1024;
- Мобільні пристрої (Mobile): 375×812 (iPhone), 412×915 (Android).

2. *Кросбраузерне тестування.* Перевірка працездатності JavaScript-сценаріїв та стилів CSS3 у популярних сучасних браузерах: Google Chrome, Mozilla Firefox, Apple Safari, Microsoft Edge.

Таблиця 4.2. Матриця функціональних тест-кейсів веб-застосунку

ID	Назва тест-кейсу	Опис кроків	Очікуваний результат	Статус
TC-01	Реєстрація нового користувача	Введення валідного email та пароля, надсилання форми	Створення запису в MongoDB, редирект на /login	Успішно
TC-02	Авторизація користувача	Введення зареєстрованих даних у форму входу	Створення сесії користувача, доступ до приватної панелі	Успішно
TC-03	Створення картки подорожі	Натискання «Створити подорож», введення назви, дат та міста	Поява нової подорожі у списку, збереження в БД	Успішно
TC-04	Інтеграція з Google Places API	Введення назви об'єкта (готель, музей) у пошуковий рядок	Динамічне виведення підказок, отримання координат, адреси й рейтингу з API	Успішно
TC-05	Додавання локації до маршруту	Натискання кнопки «Додати до плану» на знайденій локації	Оновлення масиву locations у моделі подорожі, відображення на сторінці	Успішно
TC-06	Обробка помилок авторизації	Введення некоректного пароля при вході	Відхилення запиту, виведення повідомлення про помилку, редирект	Успішно

Результати тестування підтвердили, що завдяки використанню адаптивної Flexbox/Grid-верстки елементи інтерфейсу масштабуються коректно, текстовий контент залишається читабельним, а інтерактивні елементи (кнопки додавання локацій, поля вводу Google Places) є зручними для сенсорного керування на мобільних телефонах.

4.5. Розгортання та впровадження системи

Фінальним етапом роботи стало впровадження розробленого веб-застосунку та його розгортання у хмарному середовищі для забезпечення постійної доступності через мережу Інтернет.

Процес розгортання складався з таких кроків:

1. *Налаштування хмарної бази даних.* Замість локального екземпляра MongoDB було розгорнуто хмарний кластер у сервісі MongoDB Atlas. Налаштовано правила мережевого доступу (IP Whitelisting) та створено безпечні облікові записи користувачів бази даних з обмеженими правами доступу.

2. *Конфігурація середовища* (Environment Variables). Для захисту конфіденційних даних (ключів доступу до сторонніх систем) усі налаштування було винесено у змінні середовища файлу .env:

- MONGO_URI – рядок підключення до хмарного кластера MongoDB Atlas;
- GOOGLE_PLACES_API_KEY – приватний токен доступу до сервісів Google Maps Platform;
- SESSION_SECRET – ключ для шифрування сесій користувачів.

3. *Публікація серверної частини*. Застосунок розгорнуто на платформі Render (або Heroku/Railway), яка підтримує безперервну інтеграцію (CI/CD) з репозиторієм проєкту на GitHub. При кожному push-запиті в головну гілку репозиторію система автоматично запускає процес

складання, інсталює залежності через npm install та перезапускає Node.js-сервер.

Впровадження хмарної інфраструктури дозволило отримати повністю готовий до практичного використання веб-застосунок, доступний користувачам за постійною URL-адресою з будь-якої точки світу.

4.6. Забезпечення продуктивності та безпеки системи

Для забезпечення стабільної роботи системи реалізовано:

- валідацію даних;
- middleware для перевірки запитів;
- обробку помилок;
- оптимізацію MongoDB-запитів;
- кешування окремих результатів.

Для забезпечення безпеки використовуються:

- bcrypt;
- express-session;
- Helmet.js;
- захист від XSS та CSRF-атак.

Висновки до четвертого розділу

1. Розроблено та успішно реалізовано комплексну стратегію тестування веб-застосунку для планування подорожей, яка охопила модульний, інтеграційний, функціональний рівні та перевірку користувацького інтерфейсу.

2. За допомогою фреймворку Jest проведено модульне та інтеграційне тестування REST API й компонентів взаємодії з MongoDB. Покриття коду тестами склало 78%, що підтверджує стабільність роботи бізнес-логіки системи.

3. Сформована матриця функціональних тест-кейсів дозволила верифікувати ключові користувацькі сценарії: від реєстрації та входу до інтеграції з Google Places API й формування інтегрованих маршрутів подорожей.

4. Кросбраузерне та інтерфейсне тестування підтвердило повну адаптивність системи та зручність її використання як на настільних персональних комп'ютерах, так і на мобільних пристроях під керуванням iOS та Android.

5. Виконано повний цикл розгортання системи із використанням хмарної бази даних MongoDB Atlas та хостингової платформи Render з налаштуванням захищених змінних оточення, що забезпечило готовність веб-застосунку до реальної експлуатації.

Проведене тестування підтвердило працездатність програмного забезпечення, коректність взаємодії модулів та стабільність функціонування веб-застосунку.

ВИСНОВКИ

У процесі виконання кваліфікаційної роботи досліджено актуальні тенденції розвитку туристичної галузі та обґрунтовано потребу в цифрових інструментах комплексного планування подорожей. Статистичні дані UNWTO підтверджують стабільне відновлення і зростання міжнародного туризму, що свідчить про значущість і перспективність розробленого рішення.

Проведено детальний аналіз методологічних підходів до розробки веб-застосунків: порівняно гнучкі і каскадні моделі, монолітну і мікросервісну архітектури. Для проєкту обрано монолітну архітектуру з модульною внутрішньою організацією, що оптимально відповідає початковому масштабу застосунку і дозволяє в майбутньому здійснити міграцію на мікросервіси.

Досліджено роль JavaScript як ключової технології для реалізації інтерактивного інтерфейсу і серверної логіки. Використання Node.js і Express.js на серверній стороні забезпечує однорідний технологічний стек і спрощує розробку. Порівняльний аналіз картографічних API обґрунтував вибір Google Places API як основного джерела даних про туристичні об'єкти.

Проведено аналіз ринку туристичних платформ (Airbnb, Booking.com, TripAdvisor) і визначено основний конкурентний недолік існуючих рішень – відсутність єдиної платформи для комплексного планування, що охоплює пошук різнопланових об'єктів і персоналізоване управління маршрутами.

Розроблено прототип і дизайн інтерфейсу засобами Figma з урахуванням принципів UX/UI: теплу кольорову палітру, зрозумілу навігацію, лаконічні картки місць і зручні форми взаємодії. Адаптивний дизайн забезпечує коректне відображення на мобільних, планшетних і настільних пристроях.

Реалізовано серверну частину застосунку на базі Node.js і Express.js, підключено базу даних MongoDB через Mongoose. Впроваджено механізми

реєстрації і авторизації з хешуванням паролів (bcrypt) і збереженням сесій у MongoDB. Інтеграція з Google Places API забезпечує отримання актуальних даних про туристичні об'єкти.

Побудовано інтерфейс засобами HTML5, CSS3 і Handlebars. Клієнтський JavaScript реалізує динамічне відображення результатів пошуку, перемикання між категоріями, асинхронне управління планами і функцію автодоповнення. Адаптивна CSS-верстка забезпечує зручність використання на мобільних пристроях.

Розроблений веб-застосунок для планування подорожей є функціональним прототипом, готовим до подальшого розвитку. Веб-застосунок успішно протестований і розгорнутий у хмарі.

Перспективи вдосконалення: додавання спільного редагування маршрутів, інтеграція з сервісами бронювання транспорту, впровадження push-сповіщень і розробка нативних мобільних застосунків на основі існуючого API.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Торік зафіксовано 1,5 мільярда туристичних подорожей – UNWTO. URL: https://www.unwto.org/tourism-data/unwto-tourism-dashboard

2. Світовий туризм повернувся до показників до пандемії – ООН, 2025. URL: https://www.un.org/sustainabledevelopment/tourism

3. Мельник К. Туризм відновлює допандемічні показники. Понад 1,4 мільярда людей подорожували світом у 2024 році. URL: https://www.thevillage.com.ua

4. Корисні туристичні додатки для ваших подорожей Європою, 2024. URL: https://visitworld.today/ua/blog/tourism-apps-europe

5. Bhavsar K., Shah V., Gopalan S. Scrumbanfall: an agile integration of scrum and kanban with waterfall in software engineering. International Journal of Innovative Technology and Exploring Engineering. 2020. №9(4). P. 2075–2084.

6. Susanto A., Andriana R. Comparison of Waterfall and Prototype Models for Information System Development. Majalah Ilmiah UNIKOM. 2016.

7. Приходченко С. Д., Роина К. С., Поштак Р. В. Обґрунтування вибору мікросервісної архітектури в порівнянні з монолітною. 2018.

8. Киселевич В. Мікросервісна архітектура: переваги та недоліки її практичного застосування. Information Technology. 2024. №2. С. 50–59.

9. Hamidli N. Introduction to UI/UX design: key concepts and principles. 2023.

10. Чеботарьова І. Б., Черкашина Г. І. Основні тренди UI/UX дизайну 2024 року. IX Міжнар. наук.-техн. конф. Харків: ТОВ «Друкарня Мадрид», 2024. С. 40–47.

11. Баришев Ю. В., Каплун В. А. Метод автентифікації віддалених користувачів для мережевих сервісів. 2014.

12. Лубко Д. В., Мірошниченко М. Ю. Механізми безпеки інформації та їх виклики. Науковий вісник ТДАТУ. 2024. №14(2).

13. Яковишен П. О., Тужанський С. Є. Протоколи передавання даних в реальному часі в телемедичних системах. ВНТУ. 2024.

14. Циганенко Д. П. Методи та технології захисту цифрових активів веб-додатків в умовах DDoS-атак. 2024.

15. Gu X., Zhang H., Zhang D., Kim S. Deep API learning. Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering. 2016.

16. Маслов М. Функції в JavaScript: чи все ви про них знаєте. 2024. URL: https://dou.ua/forums/topic/47630/

17. Використання асинхронного JavaScript та AJAX-запитів у веб-розробці. 2024. URL: https://it-rating.ua

18. Functions JavaScript. 2025. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Functions

19. Peters C. Building rich internet applications with node.js and express.js. 2017.

20. Mardan A. Pro express.js: Master express.js: The node.js framework for your web development. Apress. 2014.

21. Rodrigues-Pinto E., Baron T. H. Evaluation of the AXIOS stent for the treatment of pancreatic fluid collections. Expert review of medical devices. 2016. №13(9). P. 793–805.

22. Офіційний сайт Airbnb. URL: https://www.airbnb.com.ua/

23. Офіційний сайт Booking.com. URL: https://www.booking.com/index.uk.html

24. Офіційний сайт TripAdvisor. URL: https://www.tripadvisor.com/

25. Lawson B., Sharp R. Introducing html5. New Riders. 2011.

26. Mazinanian D., Tsantalis N., Mesbah A. Discovering refactoring opportunities in cascading style sheets. Proceedings of the 22nd ACM SIGSOFT International Symposium. 2014.

27. Rompis A. C., Aji R. F. Perbandingan Performa Kinerja Node.js, PHP, dan Python dalam Aplikasi REST. CogITo Smart Journal. 2018. №4(1). P. 171–187.

28. Брацький В. О., М'якшило О. М. Дослідження особливостей застосування реляційних і нереляційних баз даних. Наукові праці НУХТ. 2016. №5. С. 15–24.

29. Bielak K., Borek B., Plechawska-Wójcik M. Web application performance analysis using Angular, React and Vue.js frameworks. Journal of Computer Sciences Institute. 2022. №23. P. 77–83.

30. Kukkonen S. Express.js vs. Django: Kahden palvelinpuolen vertailu. 2024.

31. Скряга Є. Є. Розробка сучасних методів побудови інтерактивного ресурсу картографічної моделі. 2025.

32. Kaindl J. Implementation of the Fetch API for Graal.js. Johannes Kepler University Linz. 2023.

ДОДАТОК А

Лістинг програмного модуля

```
const { Router } = require('express'); // Імпортуємо Router з Express
const router = Router(); // Створюємо екземпляр роутера
const User = require('../models/user');
// Імпортуємо модель користувача
const bcrypt = require('bcryptjs');
// Імпортуємо bcrypt для шифрування паролів
// GET-запит на сторінку реєстрації
router.get('/registration', (req, res) => {
       res.render('registration', {
            title: 'Реєстрація',
         }); // Рендеримо сторінку реєстрації
    });
// POST-запит для реєстрації користувача
router.post('/registration', async (req, res) => {
       const { name, email } = req.body; // Отримуємо дані з форми
       const candidate = await User.findOne({ email }).lean();
       // Шукаємо користувача в базі за email, та повертаємо чисті
       // данні, без методів mongoose
        if (candidate) {
           res.send('Користувач з таким email вже існує');
            // Якщо користувач вже є, виводимо повідомлення
            } else {
            const password=await bcrypt.hash(req.body.password, 10);
            // Шифруємо пароль
            const newUser=new User({ name, email, password, plans: []
          }); // Створюємо нового користувача
        await newUser.save(); // Зберігаємо користувача в базу
        res.redirect('/login');
       // Після реєстрації перенаправляємо на сторінку входу
     }
  });
// GET-запит на сторінку логіна
router.get('/login', (req, res) => {
    res.render('login', {
```

```
    title: 'Вхід',
   }); // Рендеримо сторінку входу
  });
// POST-запит для авторизації користувача
router.post('/login', async (req, res) => {
   const { email, password } = req.body; // Отримуємо дані з форми
   const candidate = await User.findOne({ email }).lean();
   // Шукаємо користувача за email
     if (!candidate) {
        res.redirect('/login');
        // Якщо користувач не знайдений, повертаємо на сторінку входу
        } else {
           const  verifyPassword  =  await  bcrypt.compare(password,
candidate.password);
        // Порівнюємо хешований пароль
       if (!verifyPassword) {
          res.redirect('/login');
        // Якщо пароль не співпав, повертаємо на логін
        } else {
           req.session.user = candidate;
           // Зберігаємо користувача в сесії
           req.session.isAuth = true;
           // Позначаємо, що користувач авторизований
           req.session.save(() => {
               res.redirect('/');
         // Після збереження сесії перенаправляємо на головну сторінку
       });
      }
    }
  });
// GET-запит для виходу з облікового запису
router.get('/logout', (req, res) => {
   req.session.destroy(() => {
   res.redirect('/login');
    // Видаляємо сесію і перенаправляємо на сторінку входу
   });
  });
module.exports = router; // Експортуємо роутер
```