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

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка web-застосунку "Планувальник подорожей"»

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

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

_____ Ростислав БЕРЕЗОВСЬКИЙ
(підпис)

Виконав: здобувач вищої освіти групи ПД-43

Ростислав БЕРЕЗОВСЬКИЙ

Керівник: Віталій ЗАЛИВА
доктор філософії (PhD)

Рецензент: _____

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____Ірина ЗАМРІЙ
« ____» _____2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Березовському Ростиславу Ростиславовичу

1. Тема кваліфікаційної роботи: «Розробка web-застосунку "Планувальник подорожей"»
керівник кваліфікаційної роботи доктор філософії (PhD) Віталій ЗАЛИВА,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «24» лютого 2025 р. № 56.
2. Строк подання кваліфікаційної роботи «02» червня 2025 р.
3. Вихідні дані до кваліфікаційної роботи: Вихідними даними для розробки веб-додатку «Планувальник подорожей» є теоретичні відомості про принципи веб-розробки та UX/UI-дизайну, опис методів створення інтерактивних користувацьких інтерфейсів, аналіз функціональних можливостей подібних онлайн-платформ, а також технічна документація з використання сучасних технологій JavaScript та відповідних бібліотек для реалізації функціоналу додатку.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Охарактеризувати стан туристичних подорожей та проаналізувати функціональні можливості існуючих онлайн-платформ для планування подорожей
2. Розкрити основні методологічні підходи до розробки web-застосунків для планування подорожей та їх вплив на ефективність функціонування таких інструменті
3. Програмна реалізація та опис функціонування застосунку для планування подорожей
4. Тестування застосунку

5. Перелік графічного матеріалу: *презентація*
   1. Аналіз аналогів
   2. Вимоги до додатку
   3. Програмні засоби реалізації
   4. Структура бази даних
   5. Архітектура програми
   6. Мапа веб-застосунку
   7. Екранні форми
   8. Апробація результатів

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

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	25.02-04.03.2025	
2	Аналіз та дослідження існуючих аналогів	05.03-13.03.2025	
3	Розгляд теоретичних аспектів розробки веб-застосунків для планування подорожей	14.03-20.03.2025	
4	Опис технологій та інструментів, що використовуються для створення веб-застосунків	21.03-01.04.2025	
5	Програмна реалізація застосунку	02.04-20.04.2025	
6	Тестування застосунку	21.04-28.04.2025	
7	Оформлення роботи: вступ, висновки, реферат	29.04-11.05.2025	
8	Розробка демонстраційних матеріалів	12.05-18.05.2025	
9	Попередній захист роботи	19.05-30.05.2025	

Здобувач вищої освіти _____ Ростислав БЕРЕЗОВСЬКИЙ
(підпис)

Керівник
кваліфікаційної роботи _____ Віталій ЗАЛИВА
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавр: 75 сторінок, 12 рисунків, 32 джерела.

Мета роботи – розробка web-застосунку для планування подорожей.

Об'єкт дослідження – процес створення веб додатку організації подорожей.

Предмет дослідження – методи, технології та інструменти, що застосовуються для розробки web-застосунків, дизайну користувацького інтерфейсу та реалізації інтерактивних функцій з використанням JavaScript.

Короткий зміст роботи: У роботі досліджено методологічні підходи до створення web-застосунків, проаналізовано сучасні технології розробки, а також функціональні можливості існуючих туристичних сервісів. Розроблено алгоритм роботи web-застосунку та реалізовано основні функції: створення й редагування планів подорожей, пошук туристичних об'єктів з геолокацією, інтеграцію з Google Places API, збереження та синхронізацію даних через MongoDB, а також реєстрацію й авторизацію користувачів із шифруванням паролів.

У роботі застосовано сучасні web-технології та програмні засоби, зокрема: Node.js та Express.js для реалізації серверної частини, MongoDB у поєднанні з Mongoose для зберігання даних, Handlebars для динамічного формування інтерфейсу користувача, HTML5 та CSS3 для розмітки та стилізації, а також Google Places API для взаємодії з геолокаційними сервісами.

Сферою використання застосунку є індивідуальні та групові подорожі, корпоративні та ділові поїздки, освітні тури, діяльність туристичних агентств і компаній, що спеціалізуються на організації подорожей.

КЛЮЧОВІ СЛОВА: WEB-ЗАСТОСУНОК, ПЛАНУВАННЯ ПОДОРОЖЕЙ, JAVASCRIPT, UX/UI, МАРШРУТИ, ІНТЕРАКТИВНИЙ ІНТЕРФЕЙС, ТУРИСТИЧНІ СЕРВІСИ.

ЗМІСТ

ВСТУП

Актуальність роботи. Подорожі відіграють важливу роль у житті людей, дозволяючи відкривати нові країни, знайомитися з культурами та отримувати унікальні враження. Туристична індустрія стрімко розвивається: за даними Всесвітньої туристичної організації (UNWTO), у 2023 році кількість міжнародних подорожей перевищила 1,3 мільярда, а витрати мандрівників продовжують зростати. Однак разом із зростанням популярності подорожей зростає й потреба в ефективних інструментах для їхнього планування. Сучасні мандрівники стикаються з безліччю організаційних труднощів, що робить процес підготовки до поїздки складним і часозатратним.

Планування подорожі включає вибір маршруту, бронювання транспорту та житла, розрахунок бюджету та пошук цікавих місць. Наразі користувачі змушені користуватися різними сервісами для виконання кожного з цих завдань, що ускладнює систематизацію інформації. Наприклад, для бронювання житла використовують Booking.com чи Airbnb, для пошуку транспортних варіантів – Skyscanner або Rome2Rio, а для створення маршруту – Google Maps. Однак ці сервіси не інтегровані між собою, що змушує витрачати час на перемикання між додатками та збереження даних у різних місцях. Особливо складно організувати поїздку групою, оскільки необхідно координувати всіх учасників, узгоджувати маршрут, бюджет і бронювання.

Ще одна проблема – збереження та доступність важливої інформації під час подорожі. Часто туристи записують адреси, нотатки та плани в різних місцях: нотатниках, окремих додатках чи навіть на папері. Це може призвести до втрати важливих даних або плутанини в розкладі. Крім того, більшість туристичних сервісів не надають можливості спільного редагування маршрутів, що робить групове планування складним. У результаті процес організації поїздки стає стресовим, а мандрівники витрачають більше часу на підготовку, ніж на сам відпочинок.

Розробка web-застосунку "Планувальник подорожей" є актуальною, оскільки дозволяє централізувати всі аспекти підготовки до поїздки в одному місці. Він допоможе зекономити час, зробити планування зручнішим та забезпечить спільний доступ для групових подорожей. Такий інструмент усуне хаотичність у збереженні інформації, спростить процес вибору локацій і забезпечить більш комфортний досвід подорожей.

Мета роботи. Створити web-застосунок "Планувальник подорожей", який допоможе користувачам зручно складати маршрути і зберігати важливу інформацію про подорож.

В процесі дослідження вирішувалися наступні **завдання:**

1. Охарактеризувати статистичні дані щодо туристичних подорожей та переваги онлайн-інструментів для їх планування.
2. Розкрити сутність основних методологічних підходів до розробки таких застосунків.
3. З’ясувати роль і можливості JavaScript у розробці функціоналу планувальника подорожей.
4. Проаналізувати функціональні можливості існуючих платформ.
5. Описати ключові поняття та принципи веб-розробки.
6. Дослідити основні технології, що використовуються для створення web-застосунків.
7. Зіставити можливості різних бібліотек та інструментів JavaScript для їх використання в проєкті.
8. Обґрунтувати вибір методів та принципів проєктування інтерфейсу для зручності користувачів.
9. Вивчити особливості реалізації інтерактивних елементів для зручності користувачів.
10. Розробити прототип користувацького інтерфейсу.
11. Встановити основні вимоги до функціональних можливостей застосунку.
12. Удосконалити візуальне оформлення інтерфейсу відповідно до сучасних вимог UX/UI.

13. Розробити стильове оформлення та структуру веб-сторінок відповідно до дизайну.
14. Охарактеризувати основні функції, які будуть реалізовані за допомогою JavaScript.

Об'єктом дослідження є процес створення web-застосунків для планування подорожей.

Предметом дослідження є методи, технології та інструменти, що використовуються для розробки web-застосунку "Планувальник подорожей".

Методи дослідження: аналіз літературних джерел, порівняння, узагальнення, синтез, опис, класифікація.

Наукова новизна полягає у створенні інтегрованої платформи для планування подорожей, яка об'єднує функціонал різних сервісів (пошук житла, визначних місць, музеїв, парків) у єдиному середовищі.

Практична значущість. Полягає у створенні ефективного інструменту для туристів, що мінімізує час на планування поїздки та покращує взаємодію між учасниками подорожі. Web-застосунок може бути використаний туристичними агентствами, компаніями, що організовують корпоративні поїздки, а також індивідуальними мандрівниками для оптимізації процесу планування.

Апробація результатів та публікації.

Структура роботи. Робота складається зі вступу, 3 розділів, висновку, списку літератури (32 джерела).

1 ТЕОРЕТИЧНІ ОСНОВИ РОЗРОБКИ WEB-ЗАСТОСУНКУ

1.1. Актуальність та значення планування подорожей у сучасному світі

З розвитком цифрових технологій та збільшенням доступності інформації, процес планування подорожей зазнав значних змін, ставши значно зручнішим і доступнішим для широкого кола споживачів. Плани на подорожі більше не обмежуються лише вибором напрямку чи перевезення. Сьогодні вони включають в себе ретельний підбір маршрутів, бронювання житла та транспортних засобів, а також пошук місцевих атракцій і ресторанів. Це створює нові вимоги до туристичних послуг, а також до засобів планування, які мають допомогти ефективно і безпечно організувати подорож.

Одним з основних факторів, що підвищує актуальність планування подорожей, є постійне зростання міжнародного туризму. Згідно з даними Всесвітньої туристичної організації ООН (UNWTO), у 2019 році кількість міжнародних туристичних прибуттів сягнула близько 1,5 мільярда, що є свідченням стабільного інтересу до подорожей[1]. Попри численні економічні та соціальні виклики, пов'язані з пандемією COVID-19, туризм демонструє здатність до швидкого відновлення, і цей процес триває й сьогодні. Зокрема, в 2024 році міжнародні туристичні прибуття відновилися до 99% рівня 2019 року, що підкреслює тенденцію до відновлення до допандемічних обсягів[2].

Крім того, доходи від міжнародного туризму в 2024 році склали 1,9 трильйона доларів, що в середньому становить понад 1000 доларів на туриста. Це показує, що не тільки кількість подорожей зросла, а й економічний вплив туризму на глобальну економіку продовжує бути значним. Враховуючи цей рівень доходів, можна з упевненістю сказати, що туризм є важливим рушієм для економік багатьох країн[3].

Регіональні дані також демонструють цікаві тенденції. Європа прийняла 747 мільйонів туристів, що на 1% більше, ніж у 2019 році, і це свідчить про стійке

зростання популярності європейських напрямків. Водночас Азія та Тихоокеанський регіон, хоча й зросли до 87% від рівня 2019 року, все ж залишаються в певній мірі обмеженими через різноманітні локальні обмеження та адаптацію до нових умов. Зростання інтересу до Близького Сходу та Африки вражає: тут спостерігається 32%-ве зростання в порівнянні з допандемічними показниками, що вказує на зміщення географії туристичних потоків[3].

Цікавою є ситуація в Північній та Південній Америці, де кількість туристів зменшилася на 3% порівняно з 2019 роком. Це може бути зумовлено змінами в політичних та економічних умовах, а також можливими перебоями в транспортній інфраструктурі. У той же час такі країни, як Катар, показали вражаючі результати, зростаючи на 137% порівняно з попереднім роком, що свідчить про інвестиції в розвиток туристичної інфраструктури та стратегічну орієнтацію на залучення міжнародних відвідувачів[3].

Не можна оминути і місцеві успіхи в туристичному секторі, такі як рекордний туристичний збір Києва — 47,95 мільйона гривень за 11 місяців 2024 року. Це підтверджує важливість навіть в умовах воєнного стану підтримки туристичного потоку, що є важливим для відновлення економіки та іміджу країни на міжнародній арені[3].

Незважаючи на те, що в минулому планування подорожей вимагало значних зусиль та часових витрат, сучасні онлайн-інструменти значно спрощують цей процес. Сервіси для планування подорожей сьогодні використовують інтелектуальні алгоритми, які дозволяють персоналізувати кожну поїздку. Користувачі можуть отримувати рекомендації по маршрутах, враховуючи їхні уподобання та інтереси. Мапи, відгуки інших туристів, рейтинги й рекомендації допомагають знаходити найкращі варіанти для організації подорожі[4].

Загалом, планування подорожей є надзвичайно актуальним і необхідним елементом глобального туризму. Завдяки розвитку технологій, можливість здійснити подорож стала доступною для більшої кількості людей, а також стало простішим і швидшим планування. Онлайн-інструменти дозволяють значно зменшити час та зусилля, що потрібні для організації подорожі, підвищуючи рівень

зручності та безпеки. Таким чином, планування подорожей у сучасному світі є важливою складовою туризму, яка сприяє підвищенню якості туристичного досвіду.

1.2. Методологічні підходи до розробки web-застосунків для планування подорожей

Розвиток інформаційних технологій сприяє створенню нових рішень для організації подорожей. Вибір підходів до розробки web-застосунків відіграє ключову роль у забезпеченні їхньої надійності, функціональності та зручності використання.

Застосування відповідних методів дає змогу оптимізувати процес створення таких рішень, враховуючи вимоги користувачів та технічні можливості. Важливим аспектом є забезпечення гнучкості та масштабованості системи, що дозволяє адаптувати її до різних потреб. Розгляд основних підходів до розробки дає змогу обрати оптимальну стратегію реалізації, що сприятиме ефективній роботі web-застосунку.

Розробка web-застосунків для планування подорожей може відбуватися за різними методологіями. Одні з них передбачають гнучкий підхід із можливістю змін на будь-якому етапі, інші – чітку послідовність дій без відхилень від початкового плану. Вибір методології впливає на швидкість роботи, зручність для розробників і можливість враховувати побажання користувачів.

Гнучкі методології, такі як Agile, Scrum і Kanban, допомагають швидко адаптуватися до змін. Agile передбачає розробку у вигляді коротких етапів, де кожен новий функціонал тестується й покращується. Scrum організовує роботу в короткі відрізки часу (спринти), після яких команда оцінює виконану роботу та планує наступні завдання. Kanban зосереджується на рівномірному розподілі роботи між розробниками, щоб уникати перевантаження[5].

Для планувальника подорожей такі методи дуже зручні, адже вони дозволяють поступово додавати нові функції, покращувати інтерфейс і швидко

реагувати на потреби користувачів. Однак, гнучкі підходи вимагають активної комунікації в команді та можуть призводити до хаотичного процесу, якщо немає чіткого плану.

Інший варіант – каскадна модель Waterfall. Вона передбачає чітку послідовність етапів: спочатку аналізуються вимоги, потім створюється проєкт, пишеться код, тестується програма і лише після цього відбувається запуск. Цей підхід добре підходить для проєктів, де всі вимоги відомі заздалегідь і не змінюються під час розробки. Його перевага – зрозумілість і контрольованість процесу, але мінус у тому, що змінювати щось після початку роботи дуже складно[6].

Для планувальника подорожей Waterfall може бути корисним на етапі початкового планування, коли потрібно визначити основні функції. Проте для подальшої розробки краще використовувати гнучкі методи, щоб можна було легко додавати нові можливості.

Обидва підходи мають свої плюси і мінуси. Гнучкі методології дозволяють створювати зручні web-застосунки, які легко оновлювати, тоді як каскадна модель забезпечує порядок у процесі розробки. Найкращий варіант – комбінувати ці методи, використовуючи їх сильні сторони для досягнення якісного результату.

Монолітна та мікросервісна архітектури – це два основні підходи до розробки web-застосунків, кожен із яких має свої переваги та недоліки. Вибір між ними залежить від вимог проєкту, ресурсів команди та майбутніх планів щодо розвитку застосунку.

Монолітна архітектура передбачає, що всі частини застосунку – інтерфейс, серверна логіка та база даних – працюють як єдина система. Це спрощує початкову розробку, оскільки всі компоненти взаємодіють без додаткових налаштувань. Такий підхід добре підходить для невеликих проєктів, де важливо швидко створити робочий продукт із мінімальними витратами. Продуктивність монолітного застосунку зазвичай вища, оскільки всі процеси працюють у межах однієї системи без необхідності обміну даними між окремими сервісами[7].

Однак, зростання застосунку може створити проблеми. Чим більше функцій додається, тим складніше підтримувати код, вносити зміни та тестувати нові можливості. Якщо один компонент виходить з ладу, це може призвести до збоїв у всій системі. Оновлення застосунку вимагає перевірки та розгортання всієї програми, що може затримувати впровадження нових функцій.

Мікросервісна архітектура, навпаки, розбиває застосунок на окремі сервіси, кожен із яких виконує свою задачу. Наприклад, один сервіс може відповідати за бронювання готелів, інший – за прокладання маршрутів, а третій – за обробку платежів. Кожен сервіс працює незалежно та може оновлюватися окремо від інших. Це значно покращує масштабованість: якщо потрібно збільшити продуктивність, достатньо розширити лише той сервіс, який має найбільше навантаження[8].

Такий підхід також робить застосунок стійкішим до збоїв – якщо один сервіс перестає працювати, інші продовжують виконувати свої функції. Крім того, мікросервіси дозволяють розробникам працювати з різними технологіями в межах одного застосунку, що дає більшу гнучкість у виборі інструментів.

Втім, розробка мікросервісної системи складніша, ніж монолітної. Кожен сервіс потребує окремої інфраструктури, засобів для комунікації та системи управління розгортанням. Це збільшує витрати та вимагає від команди більшої компетентності. Крім того, через постійний обмін даними між сервісами продуктивність може знижуватися, особливо якщо не оптимізовано мережеві запити.

Для невеликих застосунків монолітна архітектура часто є найкращим вибором, оскільки дозволяє швидко створити продукт із мінімальними зусиллями. Однак, якщо проєкт має перспективи розвитку та розширення, мікросервісний підхід може значно спростити підтримку й масштабування. Вибір між цими двома варіантами залежить від цілей розробки, наявних ресурсів і довгострокових планів щодо розвитку застосунку.

UI/UX відіграє ключову роль у розробці планувальника подорожей, оскільки від зручності та зрозумілості інтерфейсу залежить, наскільки ефективно користувачі зможуть організовувати свої поїздки. Гарний дизайн не лише робить

застосунок привабливішим, а й допомагає людям швидко знаходити потрібні функції та легко взаємодіяти з системою.

Користувацький досвід (UX) визначає, наскільки інтуїтивно зрозумілою та комфортною є робота із застосунком. Планування подорожі – це процес, що включає вибір транспорту, бронювання житла, складання маршруту та багато інших дій. Якщо функціонал заплутаний або занадто складний, користувач може відмовитися від застосунку, віддавши перевагу більш простим альтернативам. Важливо, щоб усі етапи взаємодії були логічними та послідовними, а система давала зрозумілий зворотний зв’язок про виконані дії[9; 10].

Дизайн інтерфейсу (UI) має бути привабливим, але головне – він не повинен перевантажувати користувача. Принципи проєктування зручного інтерфейсу включають використання чітких і зрозумілих елементів, інтуїтивну навігацію та узгодженість стилю. Наприклад, кнопки для бронювання або перегляду маршруту мають бути легко помітними, а шрифти та кольори – комфортними для читання. Важливо дотримуватися балансу між функціональністю та простотою, щоб користувач міг швидко виконувати потрібні дії без зайвих зусиль[10].

Ще одним важливим аспектом є адаптивність застосунку. Користувачі можуть планувати подорожі з різних пристроїв – смартфонів, планшетів, ноутбуків. Інтерфейс має підлаштовуватися під розмір екрана, зберігаючи зручність використання. Наприклад, на мобільному пристрої важливо зробити великі інтерактивні елементи, щоб ними можна було легко користуватися без випадкових натискань.

UI/UX у планувальнику подорожей визначає не лише зовнішній вигляд застосунку, а й те, наскільки комфортним буде його використання. Якщо розробники приділяють увагу зручності, логіці взаємодії та адаптивності, це підвищує ймовірність того, що користувачі залишаться задоволеними і будуть використовувати застосунок у майбутньому.

Планувальник подорожей може зберігати інформацію про маршрути, бронювання, платіжні дані та інші конфіденційні відомості, тому захист цих даних є пріоритетом.

Одним із основних методів забезпечення безпеки є автентифікація користувачів. Найпоширенішим підходом є використання паролів, однак вони можуть бути вразливими до атак, якщо не дотримуватися певних правил. Використання двофакторної автентифікації значно підвищує рівень захисту, оскільки користувачеві потрібно підтвердити особу за допомогою додаткового коду або біометричних даних[11]. Також важливо впроваджувати системи виявлення підозрілих спроб входу, які можуть сигналізувати про злом облікового запису.

Шифрування даних допомагає захистити інформацію під час її передавання та зберігання[12]. Наприклад, використання протоколу HTTPS гарантує, що всі дані, які передаються між користувачем і сервером, залишаються захищеними від перехоплення. Крім того, важливі відомості, такі як паролі або платіжні дані, мають зберігатися у зашифрованому вигляді, щоб у разі витоку їх неможливо було використати[13].

Захист від основних веб-загроз є ще одним важливим напрямком. Однією з найпоширеніших атак є SQL-ін’єкції, які можуть дозволити зловмиснику отримати доступ до бази даних. Для запобігання таким атакам слід використовувати параметризовані запити та перевіряти введені дані. Інша загроза – міжсайтовий скриптинг (XSS), коли зловмисник впроваджує шкідливий код у веб-сторінку. Впровадження механізмів фільтрації вводу та Content Security Policy допомагає уникнути цієї небезпеки[14].

Інтеграція з зовнішніми сервісами відіграє важливу роль у розробці планувальника подорожей, оскільки дозволяє отримувати актуальну інформацію про транспорт, житло, погодні умови та інші аспекти подорожі. Без доступу до цих даних застосунок був би обмеженим, змушуючи користувачів самостійно шукати необхідну інформацію.

Одним із найефективніших способів отримання даних є підключення API[15]. Служби, такі як Google Maps API, надають можливість будувати маршрути, переглядати карти, визначати відстань між містами та знаходити важливі локації. Це значно спрощує користувачам планування поїздок, оскільки

вони можуть отримати точні географічні дані без необхідності перемикатися між різними сайтами.

Ще один важливий напрям – інтеграція із сервісами бронювання житла, такими як Booking API або Airbnb API. Завдяки цьому користувач може прямо в застосунку переглядати доступні варіанти проживання, порівнювати ціни та навіть здійснювати бронювання. Це не лише підвищує зручність, а й робить сервіс більш функціональним та привабливим.

Інформація про транспорт також є важливою частиною подорожі. API авіакомпаній, залізничних перевізників і сервісів таксі дозволяють отримувати розклади, ціни на квитки та можливість їхнього бронювання. Наприклад, інтеграція з Skyscanner API або Amadeus API дає змогу знайти найдешевші перельоти, а використання Uber API чи Bolt API – викликати транспорт прямо із застосунку.

Обмін даними між сервісами потребує не лише правильного підключення API, а й забезпечення безпеки. Використання токенів доступу, шифрування даних та дотримання стандартів безпеки дозволяє гарантувати, що особиста інформація користувачів не потрапить до третіх осіб. Крім того, слід враховувати обмеження API, такі як ліміти запитів або необхідність реєстрації для отримання доступу.

Інтеграція з API робить планувальник подорожей більш гнучким та корисним для користувачів. Автоматичне отримання інформації про маршрути, житло та транспорт значно спрощує процес організації поїздки, дозволяючи користувачам швидко отримувати необхідні дані без зайвих зусиль.

1.3. Використання JavaScript у створенні web-застосунків для планування подорожей

Розвиток інформаційних технологій відкриває нові можливості для автоматизації багатьох сфер життя, зокрема планування подорожей. Використання веб-застосунків дозволяє швидко організувати маршрут, знайти житло, перевірити розклад транспорту та отримати актуальну інформацію про місця призначення.

Однією з основних технологій, що забезпечують динамічність та інтерактивність таких застосунків, є JavaScript. Ця мова програмування дає змогу створювати зручні й гнучкі інтерфейси, взаємодіяти з зовнішніми сервісами, а також реалізовувати складні функції для покращення користувацького досвіду[16].

Важливість використання JavaScript у веб-розробці пояснюється його широкими можливостями та сумісністю з іншими технологіями. Завдяки цьому можна створювати застосунки, що працюють швидко, мають зручний інтерфейс і відповідають потребам користувачів у плануванні подорожей[16].

JavaScript є потужним інструментом для створення інтерактивних елементів в web-застосунках, особливо коли мова йде про планування подорожей. Завдяки своїм можливостям, JavaScript дозволяє значно покращити користувацький досвід, додаючи функції, які зроблять застосунок зручним і ефективним у використанні.

Одним з основних способів, за допомогою яких JavaScript забезпечує інтерактивність, є автозаповнення форм. Коли користувач починає вводити інформацію в поле пошуку, JavaScript взаємодіє з API або базами даних і автоматично підказує варіанти для заповнення, що значно прискорює процес планування подорожі. Це дозволяє заощаджувати час користувачів, які можуть швидше знайти потрібні локації, обрати транспорт або житло.

Ще однією важливою функцією є динамічне оновлення карт. Наприклад, при плануванні маршруту подорожі, користувач може безпосередньо на карті змінювати пункти призначення, додавати нові зупинки або переглядати альтернативні маршрути. JavaScript дозволяє здійснювати ці зміни без необхідності перезавантажувати всю сторінку, що значно покращує взаємодію користувача з веб-додатком.

JavaScript також може забезпечити відображення результатів пошуку в реальному часі. Це означає, що при введенні кожного нового параметра, наприклад, дати або місця призначення, користувач миттєво отримує оновлені результати без необхідності натискати кнопку "Пошук" або оновлювати сторінку. Завдяки взаємодії JavaScript з DOM (Document Object Model) застосунок автоматично

оновлює вміст сторінки, відображаючи нові дані, що забезпечує безперервну і динамічну взаємодію[17].

DOM є об'єктною моделлю документа, яка дозволяє JavaScript взаємодіяти з HTML і CSS. За допомогою DOM можна змінювати структуру сторінки, додавати або видаляти елементи, змінювати їхні властивості та стилі. Коли користувач взаємодіє з елементами на сторінці, JavaScript може миттєво відреагувати на ці зміни, без необхідності перезавантажувати весь документ, що робить веб-застосунки швидкими та зручними[17; 18].

Інтерактивність і динамічність, забезпечені JavaScript, роблять планування подорожей значно простішим і приємнішим для користувачів. Вони можуть працювати з застосунком без затримок, швидко отримувати необхідну інформацію та ефективно планувати свої поїздки, не витрачаючи часу на повторне завантаження сторінок або введення однакових даних.

Express.js — це популярний фреймворк для Node.js, який значно полегшує розробку серверної частини web-застосунків. Він дозволяє швидко налаштувати сервер, обробляти запити від користувачів, працювати з базами даних і інтегруватися з зовнішніми API. Це робить його ідеальним вибором для створення серверної частини застосунків, таких як планувальники подорожей, де потрібна висока ефективність і масштабованість[19].

Одна з основних переваг Express.js — це зручність обробки HTTP-запитів. Завдяки простій та зрозумілій структурі фреймворку, можна легко налаштувати маршрути для різних типів запитів (GET, POST, PUT, DELETE), що необхідно для отримання даних, оновлення інформації або видалення записів. Наприклад, в планувальнику подорожей користувач може запитувати інформацію про доступні варіанти транспорту або житло. Express.js на сервері обробляє ці запити та повертає необхідні дані[19].

Express.js також забезпечує простоту роботи з базами даних. За допомогою бібліотек, таких як mongoose (для MongoDB), або через інші ORM/ODM бібліотеки, можна швидко налаштувати зв'язок з базою даних, зберігати інформацію про користувачів, їхні маршрути, бронювання тощо. Запити до бази даних можуть бути

швидко виконані, і результат може бути повернутий користувачу в реальному часі, що є важливим для планувальника подорожей, де необхідно оперативно отримувати й оновлювати дані[19; 20].

Ще однією важливою функцією Express.js є можливість інтеграції з зовнішніми API для отримання актуальної інформації. У web-застосунках для планування подорожей часто необхідно підключати сервіси для отримання інформації про доступні варіанти транспорту, готелів, рейсів або маршрути. Express.js дозволяє зручно здійснювати HTTP-запити до сторонніх API і обробляти отримані дані, передаючи їх далі на фронт-енд або зберігаючи в базі даних. Наприклад, при плануванні маршруту Express.js може підключитися до Google Maps API або інших картографічних сервісів для виведення карт і прокладання маршрутів[20].

Крім того, Express.js забезпечує гнучкість та можливість налаштування для різних сценаріїв, що дозволяє адаптувати сервер під конкретні вимоги проєкту. Завдяки своїй простоті та масштабованості, він є відмінним вибором для розробки серверної частини web-застосунків, забезпечуючи надійність та ефективність при роботі з великими обсягами даних і запитами від користувачів.

Інтеграція з зовнішніми API є важливим елементом при розробці web-застосунків для планування подорожей. Вона дозволяє отримувати актуальну інформацію про різні аспекти подорожі, зокрема варіанти готелів, авіарейси, маршрути та багато іншого. Зовнішні API надають можливість підключатися до баз даних або сервісів, що постійно оновлюються, таким чином забезпечуючи користувачів найсвіжішими даними для планування своїх поїздок.

Однією з основних технологій для інтеграції API є Express.js, що забезпечує швидке налаштування серверної частини web-застосунку та можливість обробки запитів від користувачів. Але для безпосередньої роботи з зовнішніми сервісами можна використовувати й інші інструменти, такі як Axios або Fetch API. Ці технології дають змогу ефективно здійснювати HTTP-запити до сторонніх серверів і отримувати відповіді у форматі JSON, який є зручним для подальшої обробки.

Axios — це популярна бібліотека для здійснення HTTP-запитів у JavaScript, яка дозволяє легко взаємодіяти з API. Вона підтримує обробку помилок, автоматичне перетворення даних у формат JSON і має зручний синтаксис для роботи з запитами. Axios часто використовується в поєднанні з Express.js для інтеграції з різними зовнішніми сервісами, такими як системи пошуку рейсів чи готелів, дозволяючи отримувати необхідні дані для користувачів у реальному часі[21].

Fetch API є вбудованим інструментом у сучасних браузерах для виконання HTTP-запитів. Він є менш важким за Axios і також дуже зручний для інтеграції з API. Fetch API підтримує асинхронні запити та дозволяє працювати з промісами, що робить його відмінним вибором для обробки запитів в web-застосунках для подорожей. Проте, на відміну від Axios, Fetch не надає вбудованої обробки помилок, що може вимагати додаткових зусиль для правильного оброблення неуспішних запитів[21].

Використання цих технологій дозволяє web-застосункам для планування подорожей отримувати важливі дані з різних джерел — від інформації про наявні готелі до реального часу поїздок і варіантів транспорту. Наприклад, при плануванні маршруту користувач може звертатися до зовнішніх API для пошуку готелів у певному місті або перевірки наявності квитків на авіарейси. Інтеграція таких даних безпосередньо у застосунок робить процес планування подорожей набагато зручнішим і швидшим, дозволяючи користувачам отримувати актуальну інформацію за лічені секунди.

Завдяки можливості інтеграції з різними зовнішніми сервісами web-застосунки для планування подорожей здатні надати користувачам повний спектр послуг, які їм необхідні для організації поїздок, забезпечуючи ефективність та актуальність даних. Вибір технології для роботи з API, такої як Axios або Fetch, залежить від конкретних вимог проєкту, але в будь-якому випадку ці інструменти дозволяють реалізувати інтеграцію з зовнішніми сервісами на високому рівні.

1.4. Аналіз популярних онлайн-платформ для планування подорожей

Популярні онлайн-платформи для планування подорожей, такі як Airbnb, Booking.com і TripAdvisor, значно змінили спосіб, яким ми плануємо наші подорожі. Кожна з цих платформ надає широкий набір функцій, які роблять процес пошуку житла, організації маршрутів і бронювання квитків максимально зручним та ефективним для користувачів.

Airbnb, наприклад, спеціалізується на оренді житла від приватних осіб. Користувачі можуть знайти не тільки традиційні квартири або будинки, але й унікальні варіанти, такі як будиночки на дереві або замки. Платформа дозволяє гнучко налаштовувати пошук за такими критеріями, як ціна, розташування, зручності, тип житла. Однією з ключових переваг є система відгуків, де користувачі можуть залишати свої враження від проживання, що дозволяє новим мандрівникам зробити обґрунтований вибір[22].

Booking.com є однією з найбільших платформ для бронювання готелів, апартаментів та інших варіантів житла по всьому світу. Основною перевагою є зручність у пошуку житла за різними параметрами, такими як ціна, категорія готелю, наявність зручностей або безкоштовний сніданок. Booking.com також надає можливість бронювання турів, трансферів і автомобілів, що робить платформу універсальним інструментом для мандрівників. Платформа пропонує доступ до великої кількості пропозицій, що дозволяє швидко знайти найкраще житло в межах бюджету[23].

TripAdvisor – це платформа, яка не тільки дозволяє бронювати житло, але й дає можливість досліджувати відгуки про туристичні атракції, ресторани та інші місця. Одна з основних функцій TripAdvisor – це рекомендації від інших мандрівників. Користувачі можуть обирати найкращі варіанти за рейтингами та відгуками, що є важливим фактором при виборі місця для відпочинку. Платформа також має можливість прокласти маршрут, що дозволяє побудувати подорож по певному регіону або країні, враховуючи всі основні туристичні точки[24].

Ці три платформи – це лише невелика частина всього спектра інструментів, які доступні для планування подорожей. Вони пропонують не тільки базові функції пошуку житла та прокладання маршрутів, але й дозволяють мандрівникам брати участь у процесі обміну відгуками, бронюванні квитків і оренді автомобілів. Завдяки інтеграції з іншими сервісами, такими як програми для планування маршрутів, платформи можуть надавати користувачам всі необхідні інструменти для організації подорожі, що робить цей процес зручним, швидким та ефективним.

Порівняння зручності користування та інтерфейсів різних платформ для планування подорожей дозволяє оцінити, наскільки ефективно вони вирішують потреби користувачів у реальному часі. Платформи, такі як Airbnb, Booking.com і TripAdvisor, мають різні підходи до організації інтерфейсу, проте всі вони зосереджуються на тому, щоб забезпечити простоту використання та інтуїтивно зрозумілий досвід для мандрівників.

На рисунку 1.1 зображено Airbnb пропонує чистий і простий інтерфейс, де основна увага зосереджена на можливості швидко знайти житло за допомогою фільтрів.

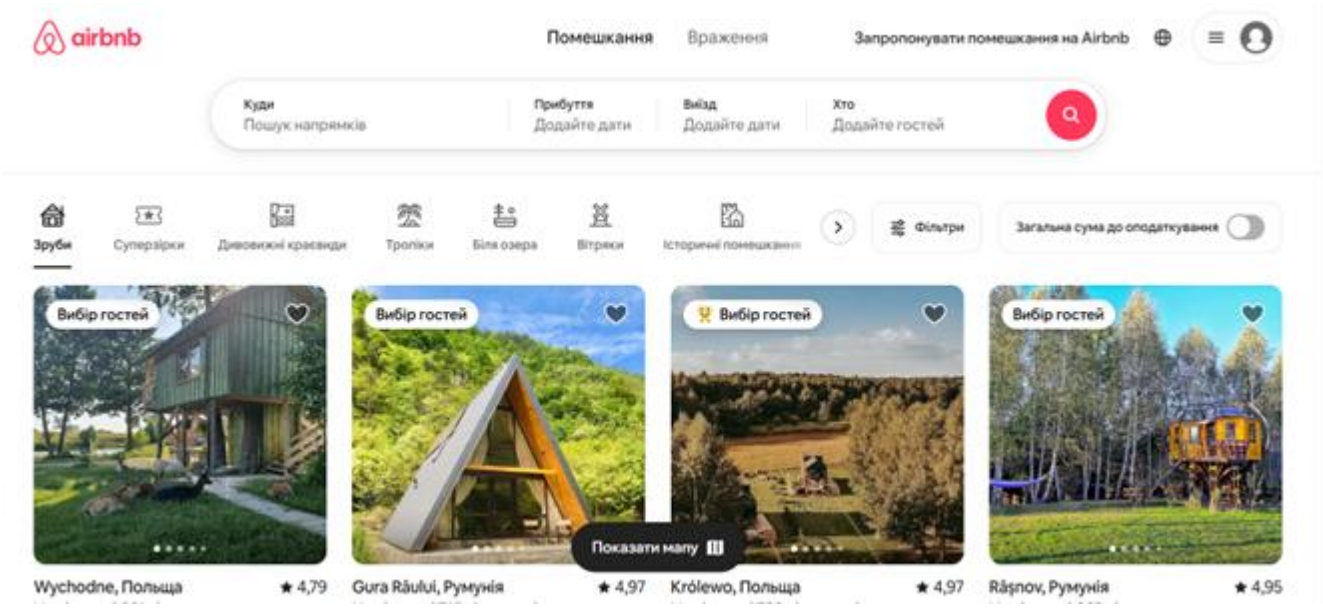


Рис. 1.1. Інтерфейс Airbnb

Вебсайт і мобільний додаток мають схожий дизайн, що робить платформу зручною на різних пристроях. Адаптивність дизайну є важливою перевагою,

оскільки користувачі можуть зручно переглядати пропозиції на телефонах, планшетах та комп’ютерах. Крім того, функціональність автоматичних рекомендацій дозволяє швидко знайти відповідні варіанти, базуючись на попередніх пошуках та уподобаннях користувача. Взаємодія з користувачем через інтерфейс чітка і зрозуміла, що дозволяє знижувати ймовірність помилок під час навігації.

Booking.com має інтерфейс, орієнтований на детальнішу організацію пошуку, зображено на рисунку 1.2.

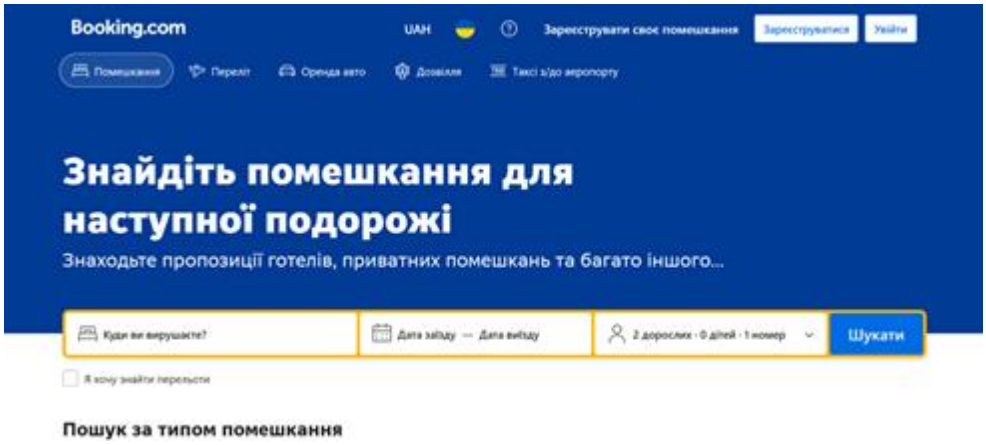


Рис. 1.2. Інтерфейс Booking.com

Більша кількість фільтрів і параметрів дає користувачеві можливість більш точно налаштувати свої вимоги до житла. Цей сайт також чудово адаптований до різних пристроїв. Мобільна версія зберігає основні функції настільної версії, що робить пошук на мобільних пристроях таким же зручним, як і на ПК. Інтерфейс має кілька розділів, таких як "Збережені об’єкти" та "Історія пошуків", що дозволяє користувачам швидко знаходити раніше переглянуті варіанти, і таким чином полегшує планування подорожі.

На рисунку 1.3. зооюражено TripAdvisor орієнтований більше на отримання рекомендацій від інших користувачів, а також на пошук атракцій, ресторанів і інших місць для відвідування.

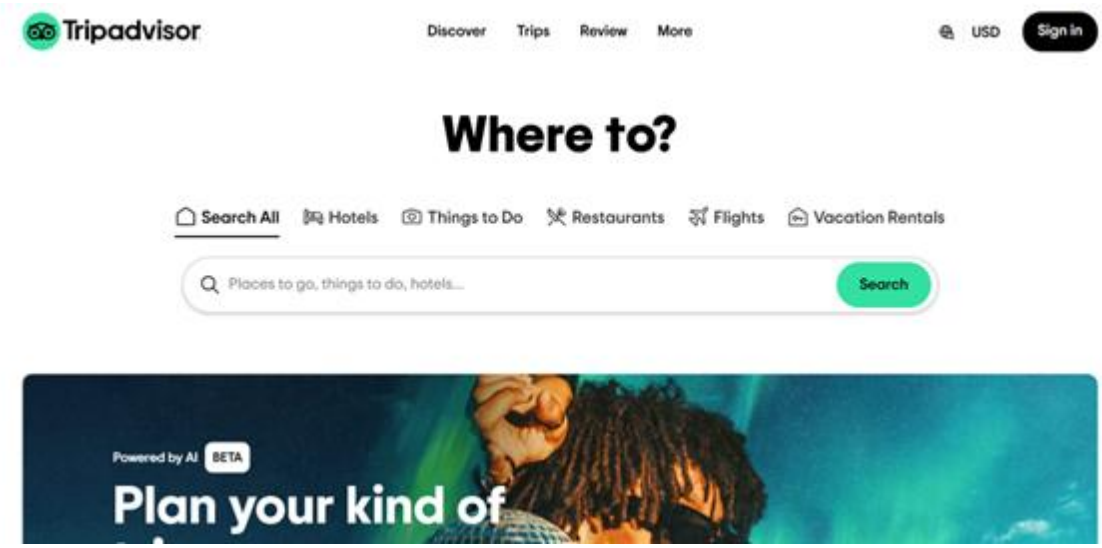


Рис. 1.3. Інтерфейс TripAdvisor

Інтерфейс платформи зручний у використанні завдяки простому та мінімалістичному дизайну.Адаптивність дизайну дозволяє ефективно використовувати платформу на різних пристроях, однак, у порівнянні з Booking.com і Airbnb, вона не надає такої ж кількості фільтрів для пошуку. Замість цього вона зосереджена на рекомендаціях та відгуках, що дозволяє користувачам швидше отримати загальну картину про місце чи послугу.

Усі три платформи ефективно використовують адаптивний дизайн, що дозволяє безперешкодно переміщатися між різними пристроями, будь то комп'ютер чи смартфон. Кожна з них організовує взаємодію з користувачем по-своєму: Airbnb зосереджується на простоті пошуку житла, Booking.com дає більше можливостей для детальної настройки пошуку, а TripAdvisor покладається на відгуки та рекомендації. Кожен з цих підходів має свої переваги та недоліки, залежно від потреб користувача, але всі вони забезпечують зручність і ефективність у процесі планування подорожі.

Кожна з платформ має свої особливості та переваги. Наприклад, Booking.com фокусується на бронюванні житла, TripAdvisor надає корисні відгуки та рекомендації, а Airbnb пропонує унікальні варіанти проживання. Незважаючи на відмінності, всі ці сервіси намагаються зробити процес планування подорожей

більш ефективним і приємним, забезпечуючи зручний доступ до різноманітних функцій та актуальних даних.

Загалом, аналіз цих платформ дозволяє зробити висновок, що зручність користування, інтеграція з зовнішніми сервісами та адаптивний дизайн є ключовими чинниками їхньої популярності.

2 АНАЛІЗ ТЕХНОЛОГІЙ ТА ІНСТРУМЕНТІВ ДЛЯ РОЗРОБКИ

2.1. Основи веб-розробки

Веб-розробка є невід'ємною частиною технологічного прогресу, що дозволяє створювати інтерактивні та функціональні веб-застосунки, що використовуються для різноманітних цілей, від персональних блогів до масштабних бізнес-платформ. Знання основ веб-розробки є ключовим для створення ефективних і зручних веб-ресурсів. Це включає в себе вивчення основних технологій, таких як HTML, CSS, JavaScript, а також розуміння принципів проєктування, використання фреймворків і бібліотек, що допомагають прискорити процес розробки.

Основи веб-розробки охоплюють як фронтенд, так і бекенд-розробку. Фронтенд відповідає за вигляд і взаємодію користувача з веб-сторінкою, тоді як бекенд забезпечує обробку даних і підтримку функціональності. Розуміння цих двох аспектів дозволяє створювати інтегровані системи, які ефективно працюють разом для забезпечення користувацького досвіду.

HTML (HyperText Markup Language) є основною мовою розмітки для створення веб-сторінок. Його роль у веб-розробці важлива, оскільки HTML дозволяє визначати структуру та зміст веб-сторінки, а також організовувати взаємодію між різними елементами контенту. За допомогою HTML можна створювати базову структуру сторінки, визначати, де знаходяться заголовки, абзаци, списки, посилання, зображення та інші важливі компоненти[25].

Основні елементи HTML складаються з тегів, які огортають текст чи інші елементи. Теги є інструкціями для браузера, як відобразити контент на екрані користувача. Наприклад, тег <h1> використовується для створення заголовків, що допомагає організувати ієрархію інформації на сторінці. Теги заголовків, від <h1> до <h6>, визначають рівень важливості тексту. Теги <p> використовуються для абзаців, що дозволяє створювати блоки тексту, легкі для читання[25].

Списки в HTML можуть бути нумерованими (<ol>) або маркованими (<ul>), що допомагає створювати структуровані переліки елементів. Тег <li> позначає окремі пункти списку. Щоб додавати посилання на інші сторінки або ресурси, використовуються теги <a>, які містять атрибут href, що вказує на адресу цільової сторінки. В HTML також є можливість вставляти зображення за допомогою тега <img>, де атрибути src і alt визначають джерело зображення та текстову альтернативу для недоступних зображень[25].

Атрибути в HTML дозволяють додавати додаткові параметри до елементів, такі як стилі, класи, ідентифікатори та багато іншого. Наприклад, атрибут class дозволяє групувати елементи для подальшого стилізування за допомогою CSS, а атрибут id надає унікальне ім'я для елемента, що дозволяє звертатися до нього через JavaScript.

HTML є основою для створення будь-якої веб-сторінки, і без нього неможливо уявити сучасний інтернет. Його роль не обмежується лише візуальним відображенням контенту, а й допомагає визначити, як веб-сторінка буде взаємодіяти з іншими елементами та користувачами.

CSS (Cascading Style Sheets) є мовою стилізації, що використовується для оформлення та структурування веб-сторінок. Він дозволяє змінювати вигляд HTML-елементів, додаючи кольори, шрифти, відступи, розміри та інші властивості. CSS працює окремо від HTML і визначає, як контент має виглядати, а не саму структуру сторінки. Завдяки CSS можна значно покращити зовнішній вигляд веб-ресурсів, зробивши їх більш привабливими та зручними для користувачів[26].

Синтаксис CSS є досить простим і складається з селекторів та властивостей. Селектори використовуються для вибору HTML-елементів, до яких будуть застосовуватися стилі. Наприклад, селектор p вибирає всі елементи абзаців на сторінці. Після вибору елемента вказуються властивості, наприклад, color для кольору тексту чи font-size для розміру шрифту. Кожна властивість має своє значення, яке вказується після двокрапки[26]. Ось так може виглядати базовий CSS-стиль:

```
    p {
  color: blue;
  font-size: 16px;
}
```

CSS підтримує широкий набір властивостей, серед яких можна виділити відступи (margin, padding), кольори (color, background-color), шрифти (font-family, font-weight), а також компонування (display, position). Крім того, є можливість використовувати складніші властивості для анімацій, трансформацій або зміни вигляду елементів при взаємодії з ними, таких як при наведенні курсора[26].

Адаптивний дизайн, який є важливим аспектом веб-розробки, вимагає використання CSS для того, щоб сторінка коректно відображалася на різних пристроях з різними розмірами екранів. Для цього використовуються медіа-запити (media queries), що дозволяють змінювати стилі в залежності від характеристик пристрою, таких як ширина екрану або орієнтація[26]. Наприклад, можна зробити так, щоб на великих екранах текст мав більший розмір, а на мобільних пристроях – компактний та зручний для перегляду:

```
@media (max-width: 600px) {
  p {
    font-size: 14px;
  }
}
```

Таким чином, CSS відіграє ключову роль у формуванні зовнішнього вигляду веб-сторінок і дозволяє створювати гнучкі та адаптивні інтерфейси. Завдяки йому веб-ресурси можуть бути зручними і доступними для користувачів на різних пристроях, що є важливим аспектом у розробці ефективних і комфортних сайтів.

JavaScript є однією з основних мов програмування для веб-розробки, яка дозволяє додавати інтерактивність і динамічність на веб-сторінки. Завдяки JavaScript, веб-сторінки можуть реагувати на дії користувача в реальному часі, що значно підвищує їх функціональність і покращує досвід користувача[16].

Однією з головних функцій JavaScript є обробка подій. Це дозволяє реагувати на взаємодії користувача з елементами сторінки, наприклад, на кліки, наведення миші або введення тексту в форму. Коли користувач взаємодіє з елементами, JavaScript може виконати певні дії, наприклад, змінити стиль елемента, вивести повідомлення або оновити частину сторінки без її перезавантаження. Це створює відчуття швидкої та зручної взаємодії з сайтом[16].

Ще однією важливою функцією JavaScript є валідація форм. Завдяки JavaScript можна перевіряти правильність введених користувачем даних ще до їх надсилання на сервер. Наприклад, можна перевірити, чи заповнені всі обов'язкові поля, чи правильно введено email або номер телефону. Це дозволяє зменшити кількість помилок і підвищити ефективність взаємодії користувача з сайтом[16].

Анімації та ефекти — це ще одна важлива частина, де JavaScript грає ключову роль. Завдяки цій мові програмування можна створювати різноманітні анімації, які надають сторінкам більшої динамічності. Наприклад, можна анімувати появу елементів на сторінці, рухомі кнопки, плавне прокручування або зміни кольорів. Такі анімації роблять сторінки більш привабливими і зручними для користувачів, сприяючи кращому сприйняттю інформації.

JavaScript також дає змогу здійснювати взаємодію з іншими технологіями через асинхронні запити. За допомогою AJAX (Asynchronous JavaScript and XML) або Fetch API можна без перезавантаження сторінки отримувати дані з сервера, наприклад, для оновлення новин, пошуку результатів або додавання коментарів. Це дозволяє значно покращити швидкість і зручність веб-застосунків[17].

Таким чином, JavaScript є важливим інструментом для створення інтерактивних і динамічних елементів на веб-сторінках. Від обробки подій до складних анімацій і асинхронних запитів — ця мова дозволяє створювати сучасні та ефективні веб-додатки, які відповідають вимогам користувачів і забезпечують зручну взаємодію з веб-ресурсами.

Серверна частина є невід'ємною складовою веб-розробки, оскільки вона забезпечує функціональність і взаємодію з користувачами через обробку запитів, зберігання даних та виконання логіки програми. Сервер — це комп'ютер, який

обробляє запити, що надходять від користувачів через веб-браузери, і надсилає їм відповідні ресурси. Веб-застосунки зазвичай складаються з двох основних частин: клієнтської та серверної. Клієнтська частина відповідає за взаємодію з користувачем, а серверна частина займається обробкою даних, забезпеченням безпеки і надає дані користувачу через API або інші інтерфейси.

Однією з основних ролей серверної частини є обробка запитів. Коли користувач вводить адресу веб-сайту в браузері або взаємодіє з інтерфейсом, браузер відправляє HTTP-запит до сервера. Сервер, у свою чергу, обробляє цей запит, взаємодіє з базою даних або іншими системами, а потім надсилає результат назад на клієнтську частину у вигляді HTML, JSON або інших форматів. Запити можуть бути різними: на отримання даних, на відправлення нової інформації, на оновлення або видалення.

Основні серверні технології, які використовуються для розробки веб-застосунків, включають Node.js, PHP і Python. Node.js — це середовище для виконання JavaScript на сервері, яке дозволяє створювати масштабовані веб-застосунки з високою продуктивністю. Завдяки асинхронній обробці запитів Node.js може ефективно працювати з великою кількістю одночасних з'єднань, що робить його ідеальним для реальних веб-додатків. PHP — це один з найпопулярніших інтерпретованих мов програмування, який широко використовується для створення динамічних веб-сторінок та взаємодії з базами даних. Завдяки своїй простоті у використанні та широкій підтримці PHP є ідеальним для розробки сайтів різної складності. Python, зокрема через популярні фреймворки як Django чи Flask, є потужним інструментом для розробки серверної частини завдяки своїй гнучкості, масштабованості та простоті[27].

Серверна частина також відповідає за зберігання та обробку даних. Для цього використовуються бази даних, які можуть бути реляційними (наприклад, MySQL або PostgreSQL) або нереляційними (наприклад, MongoDB). Сервер забезпечує взаємодію з цими базами даних для збереження, оновлення та отримання даних. Сервери можуть виконувати складну логіку обробки даних, таку як фільтрація,

агрегація або перевірка вхідної інформації, перш ніж передати її назад на клієнтську частину[28].

Реляційні бази даних, такі як MySQL та PostgreSQL, використовують таблиці для зберігання даних, де кожен запис організований у вигляді рядків і стовпців. Ці бази даних забезпечують сувору схему, що означає, що дані повинні відповідати заздалегідь визначеним типам і структурам. MySQL є популярною базою даних завдяки своїй простоті та ефективності в управлінні великими обсягами даних. PostgreSQL, в свою чергу, підтримує складніші типи даних і запити, що робить її хорошим вибором для великих і складних веб-застосунків. Однією з основних переваг реляційних баз є можливість використання SQL для складних запитів та операцій з даними, таких як об’єднання таблиць, агрегація та фільтрація[28].

Нереляційні бази даних, такі як MongoDB, використовують гнучкіші структури для зберігання даних. Замість таблиць і рядків, MongoDB зберігає дані у вигляді документів, що дозволяє зберігати дані в форматі JSON. Цей підхід дає змогу зберігати дані без жорсткої схеми, що робить базу даних ідеальною для застосунків, де структура даних може змінюватися з часом або коли необхідна висока гнучкість у зберіганні різних типів інформації. Наприклад, у випадку з веб-застосунками для планування подорожей, MongoDB може бути використана для зберігання інформації про користувачів, їхні плани, маршрути, а також для інтеграції з іншими сервісами для отримання актуальної інформації, такої як погода чи транспортні розклади[28].

Однією з головних переваг нереляційних баз є їхня здатність до горизонтального масштабування. Вони добре підходять для великих даних або даних з високою варіативністю, що змінюються із часом. Наприклад, додавання нових полів у документи MongoDB не потребує значних змін у всій базі, що робить її ідеальним вибором для застосунків, що активно розвиваються або змінюються. Однак, з іншого боку, для складних зв'язків між даними реляційні бази, такі як PostgreSQL, можуть бути кращими через їх здатність обробляти складні SQL-запити.

Веб-розробка вимагає ефективної роботи з базами даних, щоб забезпечити зберігання та швидкий доступ до даних користувачів, контенту, а також підтримку масштабованості та гнучкості. Як реляційні, так і нереляційні бази даних відіграють важливу роль у забезпеченні цих можливостей, і правильний вибір бази даних є критичним для успішного розвитку веб-застосунків.

2.2. Огляд технологій і систем для розробки web-застосунків

Розробка веб-застосунків потребує ефективного використання різноманітних технологій, які забезпечують зручний, швидкий та масштабований процес створення як фронтенду, так і бекенду. Вибір правильних інструментів залежить від потреб проєкту, його складності, цілей та вимог до функціональності.

Для розробки фронтенду найбільш популярними є три фреймворки: React, Angular та Vue.js. React — це бібліотека для створення інтерфейсів користувача, розроблена Facebook. Вона базується на компонентному підході, що дозволяє легко створювати багатофункціональні та масштабовані додатки. Однією з основних переваг React є її велика спільнота та екосистема, що забезпечує доступ до численних бібліотек та інструментів, спрощуючи процес розробки. Angular, розроблений Google, є більш комплексним фреймворком для створення веб-застосунків. Він пропонує не тільки компонентну архітектуру, а й багатий набір інструментів для тестування, маршрутизації, обробки форм та інтеграції з серверною частиною. Angular добре підходить для великих та складних проєктів, де необхідно мати все в одному рішенні. Vue.js, з іншого боку, є менш складним, але дуже гнучким фреймворком, який поєднує кращі риси React і Angular. Він пропонує простоту в освоєнні, а також потужний інструментарій для створення динамічних інтерфейсів, завдяки чому є популярним серед розробників невеликих та середніх проєктів[29].

Що стосується бекенду, існує багато популярних фреймворків, серед яких Express.js, Django та Flask. Express.js є мінімалістичним і гнучким фреймворком для Node.js, що дозволяє швидко налаштувати серверну частину веб-застосунку. Його

основною перевагою є швидкість розробки та великий вибір додаткових модулів, які дозволяють налаштувати будь-які аспекти сервера. Express.js чудово підходить для невеликих та середніх проєктів, де важлива швидкість виконання та простота. Django, побудований на Python, є потужним фреймворком для розробки складних веб-застосунків. Його особливістю є наявність великої кількості вбудованих інструментів для обробки форм, автентифікації, адміністративних панелей і так далі. Django підходить для великих проєктів, де важливо мати структуру «з коробки» для швидкого початку роботи. Flask, також створений на Python, є мікрофреймворком, що пропонує більше гнучкості та меншої кількості вбудованих інструментів, ніж Django. Він дозволяє більш точно налаштовувати проект, що робить його ідеальним вибором для проєктів, де потрібно більше контролю та менше стандартних рішень[30].

Вибір між цими технологіями залежить від типу проєкту. Якщо потрібно швидко створити веб-застосунок з мінімумом додаткових налаштувань, тоді React для фронтенду та Express.js для бекенду стануть чудовим вибором. Для більш складних та масштабованих проєктів, що потребують багатих функцій і структури «з коробки», Angular або Django будуть більш доречними. Вибір залежить від того, чи важлива гнучкість або швидкість, а також від досвіду команди розробників.

API (інтерфейс програмування застосунків) є важливим інструментом для інтеграції різних систем і забезпечення взаємодії між ними. Веб-застосунки часто потребують доступу до зовнішніх даних або сервісів для надання користувачам більш повної та корисної інформації. Завдяки API можна отримувати актуальні дані з різних джерел, таких як погодні сервіси, транспортні системи чи платіжні сервіси, що значно підвищує функціональність застосунків і робить їх більш зручними та ефективними для користувачів[15].

Одним з основних способів інтеграції з зовнішніми сервісами є використання HTTP-запитів. За допомогою методів GET, POST, PUT та DELETE можна отримувати та відправляти дані через API. Наприклад, інтеграція з погодними сервісами, такими як OpenWeather або WeatherAPI, дозволяє отримувати актуальні дані про погоду в реальному часі для користувачів, що робить застосунки

корисними для тих, хто планує подорожі або працює на відкритому повітрі. Ці сервіси надають дані у форматі JSON або XML, що робить їх зручними для подальшої обробки і відображення на сайті чи в мобільному застосунку[15].

Інтеграція з транспортними системами, такими як Google Maps API або транспортні сервіси для бронювання квитків, дає можливість надавати користувачам точні маршрути, розклади та навіть функцію реального відстеження транспорту. Це полегшує планування подорожей і робить користувацький досвід значно приємнішим. Наприклад, за допомогою API Google Maps можна інтегрувати карти на сайті та забезпечити функції побудови маршрутів, пошуку найближчих зупинок або визначення часу прибуття.

Ще однією важливою інтеграцією є сервери для оплати, такі як Stripe або PayPal, через які можна реалізувати безпечні платіжні системи. Це дозволяє веб-застосункам обробляти платіжні транзакції, не зберігаючи конфіденційну інформацію на своїх серверах. Вони забезпечують високий рівень безпеки та дозволяють проводити оплату без необхідності створювати складні платіжні системи з нуля.

Такі інтеграції з API є критично важливими для забезпечення функціональності веб-застосунків, оскільки вони дозволяють динамічно отримувати та обробляти дані з різних джерел у реальному часі, що підвищує зручність та ефективність для кінцевих користувачів. Без доступу до таких сервісів багато функцій, які ми звикли бачити у веб-додатках (як, наприклад, погодні прогнози, інформація про маршрути або можливість платіжних операцій), не були б можливі.

2.3. Аналіз існуючих інструментів і бібліотек JavaScript для планувальника подорожей

У розробці веб-застосунків для планування подорожей важливу роль відіграють інструменти і бібліотеки, що дозволяють швидко та ефективно реалізувати різноманітні функціональні можливості. JavaScript є основною мовою

програмування для створення інтерактивних веб-інтерфейсів, а також для забезпечення динамічності та взаємодії з користувачем. Вибір правильних інструментів та бібліотек для таких проєктів дозволяє значно полегшити процес розробки, підвищити ефективність та якість роботи застосунку. Існує багато бібліотек, що спеціалізуються на різних аспектах функціональності для планувальників подорожей, таких як інтеграція карт, пошук житла, відображення інформації про транспорт чи маршрути, а також взаємодія з користувачем. Цей огляд інструментів і бібліотек JavaScript дає змогу оцінити їхні можливості, переваги та обмеження у контексті створення ефективних і зручних веб-застосунків для планування подорожей.

Для створення картографічних функцій у веб-застосунках часто використовуються бібліотеки, які дозволяють інтегрувати карти та працювати з геолокацією. Три з найбільш популярних і ефективних інструментів для цього — Leaflet, Google Maps API та Mapbox.

Leaflet — це легка та гнучка бібліотека з відкритим кодом, яка дозволяє створювати інтерактивні карти. Вона підходить для невеликих проєктів та застосунків, де важлива швидкість завантаження та простота інтеграції. Leaflet підтримує інтерактивність карт, прокладання маршрутів, відображення об'єктів, таких як маркери, полигони та лінії. З Leaflet легко працювати з іншими картографічними сервісами, наприклад, OpenStreetMap, що робить її дешевшим варіантом для інтеграції карт. Бібліотека має безліч плагінів, що дозволяють додавати додаткові функції, такі як пошук місць або відображення погодних умов[31].

Google Maps API — один з найпопулярніших інструментів для інтеграції карт, який пропонує надзвичайно потужні функціональні можливості. API Google Maps дозволяє не тільки відображати карти, але й використовувати складнішу геолокаційну обробку. Наприклад, Google Maps надає можливості для пошуку місць, визначення маршрутів, розрахунку часу подорожі, а також інтеграції з іншими сервісами Google, такими як Street View або Google Places API. Це робить

Google Maps ідеальним вибором для великих проєктів з вимогами до точності та додаткових сервісів[31].

Mapbox — потужна платформа для створення інтерактивних карт, яка особливо популярна серед розробників завдяки своїй гнучкості та настроюваності. Mapbox дозволяє створювати кастомізовані карти з різними стилями, інтегрувати дані та працювати з геолокацією за допомогою власного API. Вона підтримує не лише базові функції, такі як відображення маркерів і прокладання маршрутів, але й надає розширені можливості для роботи з даними геопросторових даних, аналізу та інтерактивних карт. Mapbox також добре взаємодіє з іншими інструментами, такими як аналіз даних про трафік або погодні умови[31].

Кожна з цих бібліотек має свої особливості та переваги. Leaflet — чудовий вибір для легких і простих проєктів з картами. Google Maps API буде корисний для великих проєктів, які потребують інтеграції з іншими сервісами Google. Mapbox відрізняється високою гнучкістю та підтримкою складних картографічних функцій, що робить її чудовим варіантом для кастомізованих рішень. Вибір інструменту залежить від конкретних потреб проекту, бюджету та складності задач.

Для інтеграції з API сторонніх сервісів у веб-застосунках, таких як планувальники подорожей, часто використовуються бібліотеки, які дозволяють здійснювати HTTP-запити та обробляти дані. Два найбільш популярних інструменти для роботи з API в JavaScript — Axios та Fetch API. Обидва дозволяють ефективно взаємодіяти з зовнішніми сервісами, наприклад, для отримання актуальної інформації про погодні умови, транспортні послуги або житло.

Axios — це популярна бібліотека для здійснення HTTP-запитів, яка має зручний синтаксис та багато вбудованих можливостей. Вона дозволяє працювати з обіцянками (Promises) та підтримує асинхронні запити, що робить її чудовим інструментом для інтеграції з API. Axios відзначається своєю здатністю автоматично перетворювати відповіді серверів у формат JSON, що спрощує обробку даних. Також бібліотека надає можливість легко налаштовувати параметри запитів, обробляти помилки, а також працювати з заголовками HTTP. У

контексті планувальника подорожей Axios є корисним для роботи з великими обсягами даних, такими як запити до API для отримання інформації про погоду, розклад транспорту або доступність житла в реальному часі[21].

Fetch API, з іншого боку, є вбудованим інтерфейсом браузера для здійснення HTTP-запитів. Він підтримує проміси і є частиною стандарту JavaScript, що робить його доступним без необхідності підключати сторонні бібліотеки. Fetch API підтримує асинхронні запити та дозволяє отримувати дані у форматі JSON. Хоча Fetch API є потужним і зручним інструментом, він вимагає більше зусиль для обробки помилок і налаштування параметрів запитів порівняно з Axios. Проте для планувальника подорожей Fetch API також є відмінним вибором, коли потрібно інтегрувати інформацію з різних зовнішніх сервісів, зокрема для запитів до API, що надають актуальну інформацію про наявність номерів у готелях, ціни на авіаквитки або погодні умови в реальному часі[32].

Обидва інструменти мають свої переваги і обмеження. Axios часто обирають за її зручний синтаксис та додаткові можливості, такі як автоматична обробка помилок та налаштування запитів. Вона особливо підходить для великих проектів, де важлива стабільність і масштабованість. Fetch API є простим у використанні та не вимагає зовнішніх залежностей, що робить його хорошим вибором для невеликих проектів або для тих, хто хоче уникнути підключення додаткових бібліотек. Оскільки обидва інструменти дозволяють інтегрувати зовнішні сервіси, їх ефективність у контексті планувальника подорожей залежить від конкретних вимог проекту, таких як складність інтеграції, вимоги до швидкості та обсягів даних.

Для розробки планувальників подорожей, які дозволяють користувачам швидко шукати і бронювати житло, авіаквитки або транспортні послуги, важливо використовувати інструменти та API, які надають актуальну інформацію з різних джерел. Ось кілька популярних бібліотек та API, які значно полегшують інтеграцію таких функцій: Airbnb API, Booking.com API та Skyscanner API.

Airbnb API дозволяє розробникам інтегрувати функціонал пошуку житла з найбільшою платформою для оренди квартир і будинків. З допомогою цього API

можна отримувати дані про доступні варіанти житла на основі конкретних запитів користувача, таких як місце розташування, дати перебування, кількість гостей та інші фільтри. API дозволяє розробникам не лише здійснювати пошук, але й бронювати варіанти житла через інтеграцію з Airbnb. Це особливо зручно для планувальників подорожей, оскільки вони можуть безпосередньо надавати користувачам доступ до великих баз даних із різними варіантами житла, без необхідності переходу на сторонні платформи.

Booking.com API — це ще один популярний інструмент для інтеграції можливості пошуку та бронювання житла. Це API пропонує доступ до великої кількості готелів, апартаментів та інших варіантів розміщення. Платформа підтримує фільтрацію за ціною, зручностями, типами номерів та іншими критеріями. За допомогою API можна створювати інтерактивні інтерфейси, де користувачі можуть вибирати житло, переглядати рейтинги і відгуки, а також бронювати номери безпосередньо через ваш застосунок. Однією з важливих переваг API є його надійність та швидкість отримання даних, що значно покращує користувацький досвід.

Skyscanner API пропонує інтерфейс для інтеграції з пошуком авіаквитків та транспортних послуг. Це API дозволяє отримувати актуальні дані про перельоти, порівнювати ціни на авіаквитки та переглядати доступні рейси за заданими критеріями. Завдяки цьому інструменту можна надавати користувачам можливість швидко знаходити найбільш вигідні варіанти подорожей, без необхідності звертатися до кожного авіаперевізника окремо. API також підтримує інтеграцію з іншими типами транспорту, такими як автобуси та поїзди, що розширює можливості застосунків для планування подорожей.

Кожен з цих інструментів дає можливість швидко отримувати дані з перевірених джерел і надає гнучкі інтерфейси для інтеграції. За допомогою Airbnb API та Booking.com API можна отримати інформацію про житло, відфільтровувати варіанти за різними критеріями і бронювати їх безпосередньо через застосунок. Skyscanner API дозволяє інтегрувати пошук авіаквитків і порівняння цін, надаючи користувачам зручний доступ до інформації про рейси. Інтеграція з цими сервісами

забезпечує не лише зручність для користувачів, але й підвищує ефективність веб-застосунків, дозволяючи користувачам здійснювати бронювання в кілька кліків.

2.4. Методи та принципи проєктування користувацького інтерфейсу

Проєктування користувацького інтерфейсу (UI) є важливим етапом у створенні веб-застосунків та мобільних додатків. Від того, наскільки зручним, інтуїтивно зрозумілим і естетично привабливим буде інтерфейс, залежить успіх продукту серед користувачів. Існують кілька методів і принципів, що допомагають досягти цих цілей, і вони включають дослідження користувачів, структурування інформації, використання принципів дизайну та тестування прототипів.

Першим кроком у проєктуванні інтерфейсу є проведення дослідження користувачів. Це дозволяє зрозуміти їхні потреби, звички та проблеми, з якими вони стикаються при взаємодії з продуктом. Використання різних методів збору інформації, таких як інтерв'ю, опитування, аналіз конкурентів або навіть спостереження за користувачами в реальному часі, допомагає створити інтерфейс, який максимально відповідає потребам кінцевих користувачів.

Далі, при створенні інтерфейсу важливо правильно структурувати інформацію. Це означає організацію контенту та елементів таким чином, щоб користувач міг легко знайти необхідну інформацію чи функцію. Основними інструментами для цього є карти сайту, прототипи та wireframes, які дозволяють візуалізувати структуру сторінок і взаємодію елементів між собою. Інтерфейс повинен мати логічну й послідовну структуру, де кожен елемент має своє місце й функцію, що дозволяє уникнути перевантаження інформацією.

Принципи дизайну також є важливими для створення зручного інтерфейсу. Один з основних принципів — це принцип "зрозумілої доступності". Інтерфейс має бути таким, щоб користувачі могли інтуїтивно зрозуміти, як ним користуватися. Це досягається завдяки використанню зрозумілих та стандартних елементів інтерфейсу, таких як кнопки, меню, поля введення та інші компоненти. Інший важливий принцип — мінімалізм, який сприяє зменшенню кількості елементів і

можливих варіантів вибору, щоб не перевантажувати користувача надмірною інформацією. Кожен елемент інтерфейсу має чітку мету й бути необхідним для виконання завдання.

Після створення початкової версії інтерфейсу важливо проводити тестування. Один з найбільш поширених методів тестування інтерфейсу — це юзабіліті-тестування, коли реальні користувачі взаємодіють із прототипом, і розробники спостерігають за їхніми діями. Це допомагає виявити проблеми в навігації, недоліки в дизайні або функціональності, а також зрозуміти, чи зручний інтерфейс для виконання основних завдань.

Важливим аспектом є адаптивний дизайн, який дозволяє інтерфейсу адаптуватися до різних розмірів екранів і пристроїв. Користувачі можуть взаємодіяти з продуктом як на мобільних телефонах, так і на великих екранах комп'ютерів, тому інтерфейс повинен бути зручним і читабельним на всіх пристроях. Використання медіа-запитів і гнучких макетів дозволяє створювати адаптивні інтерфейси, які змінюють свій вигляд залежно від розмірів екрану.

Таким чином, методи та принципи проєктування користувацького інтерфейсу орієнтовані на забезпечення зручності та ефективності використання продукту. Дослідження користувачів, правильна структура інформації, застосування принципів дизайну та регулярне тестування прототипів допомагають створювати інтерфейси, які легко сприймаються і дозволяють користувачам ефективно виконувати завдання.

2.5. Інтерактивні функції в web-додатках для подорожей

Інтерактивні функції в web-додатках для подорожей є важливим елементом, який визначає досвід користувачів і їх взаємодію з платформами. Вони дозволяють створювати більш зручний, персоналізований та ефективний процес планування подорожей, забезпечуючи користувачів необхідною інформацією та інструментами для прийняття рішень.

Однією з найбільш поширених інтерактивних функцій є пошук і фільтрація контенту. Наприклад, в додатках для бронювання готелів або авіаперельотів користувач може вказати параметри пошуку, такі як дати, місце перебування, тип транспорту, бюджет або зручності, які необхідні. Платформи, такі як Airbnb чи Booking.com, використовують динамічні фільтри та відгуки для підвищення зручності пошуку і забезпечення точності результатів. Завдяки інтерактивним елементам, як-от слайдери та випадаючі списки, користувач може швидко знаходити варіанти, які відповідають його вимогам.

Іншою важливою функцією є інтеграція карт та геолокації. Використання інтерактивних карт дозволяє користувачам візуалізувати локації на карті, прокладати маршрути між точками, знаходити цікаві місця або вибирати найближчі готелі чи ресторани. Такі платформи, як Google Maps або Mapbox, надають зручні інтерфейси для побудови маршрутів, перегляду фотографій та інформації про об'єкти. Це допомагає користувачам ефективно планувати свої подорожі, обирати оптимальні маршрути та локації для відпочинку або ділових поїздок.

Ще однією ключовою інтерактивною функцією є система відгуків та рейтингу. Це дозволяє користувачам ділитися своїм досвідом, обираючи готелі, ресторани або екскурсії. Відгуки не тільки покращують взаємодію з платформою, а й створюють спільноту, де користувачі можуть отримати корисну інформацію від тих, хто вже відвідав певні місця. Важливою складовою є й інтеграція соціальних мереж, що дозволяє ділитися поїздками, маршрутам та порадами з іншими користувачами.

Один із найбільш зручних інструментів для підвищення інтерактивності — це чат-боти та віртуальні асистенти. Вони дозволяють автоматизувати допомогу користувачам, відповідати на запитання, підбирати варіанти для подорожей, а також виконувати інші функції, такі як повідомлення про зміни в рейсах чи погоді. Віртуальні помічники допомагають зберігати час користувачів, надаючи їм швидкі відповіді на запити без необхідності взаємодіяти з операторами.

Крім того, для покращення досвіду планування подорожей активно використовуються інтерактивні календарі. Вони дозволяють користувачам зручно переглядати доступні дати для бронювання житла або авіаквитків, а також організовувати свою поїздку на основі доступних варіантів. Календарі, інтегровані з іншими сервісами, дозволяють користувачам бачити актуальні ціни та зміни в реальному часі, що робить процес планування ще зручнішим.

Інтерактивність у web-додатках для подорожей робить процес планування більш інтуїтивно зрозумілим, швидким та приємним для користувачів. Вона допомагає не тільки швидко знаходити необхідну інформацію, а й значно полегшує прийняття рішень, забезпечує персоналізацію та підвищує задоволеність від користування платформами.

3 РОЗРОБКА І РЕАЛІЗАЦІЯ WEB-ЗАСТОСУНКУ

3.1. Прототипування користувацького інтерфейсу планувальника подорожей

Важливою частиною прототипування користувацького інтерфейсу для планувальників подорожей є створення зручного та інтуїтивно зрозумілого інтерфейсу, який би сприяв ефективному взаємодії користувача з системою.

Платформи для планування подорожей повинні надавати можливість швидкого пошуку житла, визначних місць, а також планування маршрутів, що вимагає продуманого та оптимізованого дизайну інтерфейсу. Врахування потреб користувачів, а також забезпечення легкості використання та швидкого доступу до всіх функцій. Успішно спроектований планувальник повинен забезпечувати легкий доступ до найнеобхідніших функцій і при цьому не перевантажувати користувача зайвою інформацією.

Один з основних блоків планувальника — це пошук житла. Користувачі часто починають свій процес планування саме з цього етапу. Важливо, щоб пошук був простим і зручним. Блок пошуку житла має включати відображення фотографії, деталі про адресу, рейтинг, кожного варіанту а також збереження в список обраних.

Крім житла, платформа повинна дозволяти здійснювати пошук інших важливих об'єктів, таких як музеї, пам’ятники, торгові центри та нічні клуби. Користувач повинен мати можливість ввести назву міста в поле пошуку, після чого система повертає список відповідних місць, кожне з яких відображається у вигляді картки з назвою, вулицею, рейтингом, зображенням та кнопкою «Додати в плани», як зображено на рисунку 3.1.

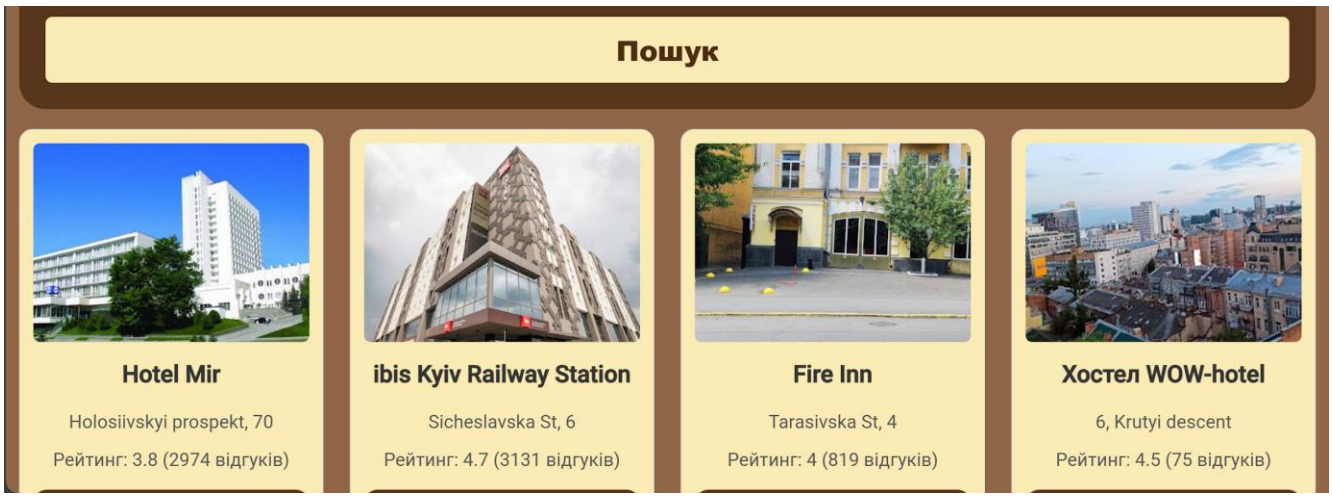


Рис. 3.1. Макет карток житла

Важливим елементом є функція створення власного списку планів. Користувач може додавати знайдені місця до особистого списку, який зберігається в базі даних і доступний після входу в обліковий запис. В особистому розділі «Мої плани» відображаються всі додані місця, включаючи готелі, музеї та інші локації. Для зручності передбачена можливість асинхронного видалення місць зі списку, а також прокладання маршруту до вибраного об'єкта.

Для збереження персоналізованого досвіду користувач повинен мати можливість зареєструватися на платформі та увійти до свого акаунта, зображено на рисунку 3.2. Це забезпечує доступ до власного списку планів з будь-якого пристрою.

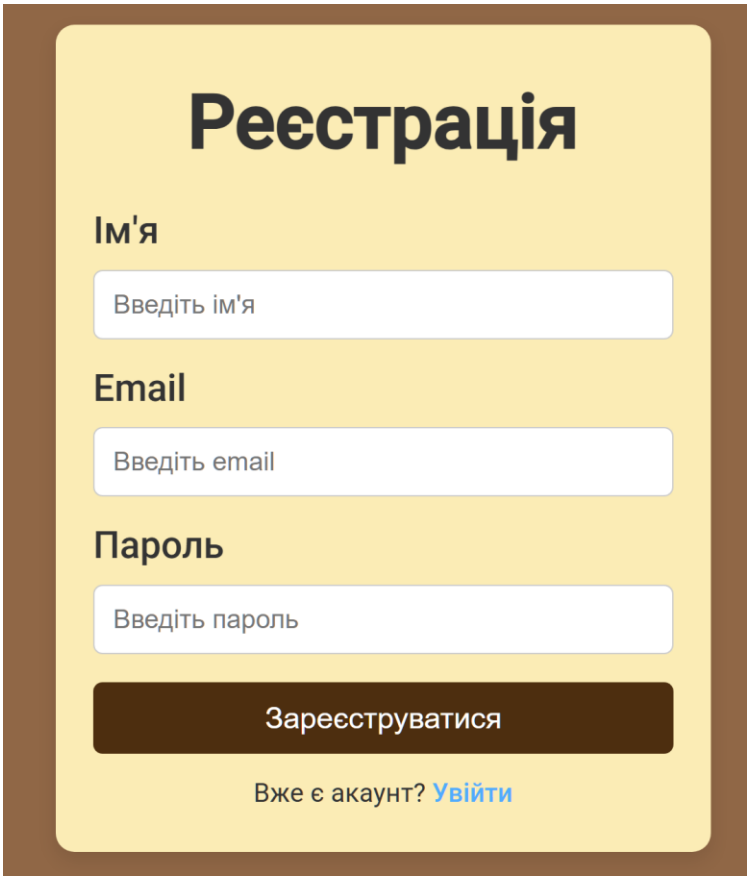


Рис. 3.2. Макет форми для реєстрації

Всі ці блоки повинні бути логічно організовані та зручно розміщені в інтерфейсі. Користувач повинен мати змогу швидко і легко переміщатися між функціями без зайвих затримок, а навігація має бути інтуїтивно зрозумілою. Важливо забезпечити, щоб ці функції не тільки відповідали потребам користувача, але й дозволяли ефективно організувати процес планування подорожі, економлячи час і зусилля.

Кнопки є наступним важливим інструментом для взаємодії. Вони повинні бути чітко видимими та мати зрозумілі підписи, щоб користувач міг одразу зрозуміти, до яких функцій вони ведуть, представлено на рисунку 3.3. Важливою є і їхня розташованість: вони повинні бути в доступних місцях і завжди в полі зору.

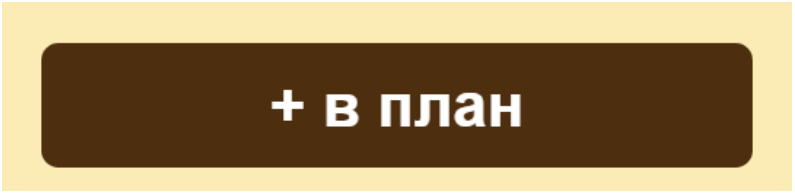


Рис. 3.3. Макет кнопки

Адаптивність інтерфейсу дозволяє забезпечити комфортне використання платформи на різних пристроях. Користувачі можуть звертатися до планувальників подорожей із смартфонів, планшетів або комп'ютерів, і для кожного з цих пристроїв інтерфейс має бути оптимізований. Це важливо, адже зручність користування безпосередньо впливає на досвід взаємодії з сервісом і на ефективність виконання завдань.

Перш за все, адаптивний дизайн дозволяє змінювати розмітку та елементи інтерфейсу в залежності від розміру екрану. Наприклад на маленьких екранах смартфонів або планшетів потрібно оптимізувати інтерфейс, зменшуючи кількість одночасно відображуваних елементів, щоб користувач мав змогу легко переглядати та взаємодіяти з контентом, не перевантажуючи екран.

Адаптивний інтерфейс включає зміну розташування кнопок, меню та інших елементів так, щоб вони були доступні та зручні для користувача незалежно від пристрою. Наприклад, на великих екранах меню може бути вертикальним або горизонтальним, в той час як на мобільних пристроях воно може бути згорнутим в іконку, що розкривається, для економії місця. Важливо також звертати увагу на розмір шрифтів та інтервалів, щоб текст залишався читабельним на будь-якому пристрої.

Іншою важливою складовою адаптивності є забезпечення функціональності всіх ключових можливостей, як-от пошук житла, на всіх пристроях. Це вимагає ретельного тестування інтерфейсу на різних платформах, щоб переконатися, що кожна функція працює належним чином.

Одним з основних підходів для досягнення адаптивності є використання медіа-запитів CSS, які дозволяють змінювати стильову інформацію в залежності від розміру екрана. Це дозволяє забезпечити гнучкість і масштабованість інтерфейсу, який виглядатиме оптимально на будь-якому пристрої.

Адаптивність інтерфейсу не лише покращує користувацький досвід, але й дозволяє зробити застосунок більш доступним для ширшої аудиторії. Користувачі можуть без проблем здійснювати необхідні операції незалежно від того, чи

використовують вони комп'ютер вдома, планшет під час подорожі або смартфон на ходу. Врахування цих аспектів під час розробки інтерфейсу дозволяє забезпечити високий рівень зручності та задоволення від використання планувальника подорожей.

3.2. Аналіз вимог до функціоналу планувальника

Кожен етап планування подорожі потребує точного налаштування інтерфейсу та можливостей, щоб користувачі могли ефективно взаємодіяти з платформою та отримувати необхідні дані в режимі реального часу. Розгляд функціональних вимог дозволяє створити продукт, що відповідає потребам користувачів і гарантує зручність у використанні.

Платформа для організації подорожей повинна надавати користувачам можливість не лише швидко знаходити цікаві місця, а й зручно планувати маршрути відповідно до їхніх уподобань. Важливо, щоб усі ці можливості були об'єднані в єдину систему, яка допоможе спростити процес пошуку готелів, музеїв, пам’ятників, торгових центрів, клубів та інших важливих локацій. Завдяки цьому користувачі зможуть ефективно організовувати свої подорожі без необхідності перемикатися між різними сервісами чи зберігати інформацію вручну.

Щоб пошук працював швидко та точно, система повинна використовувати потужні джерела даних. У цьому випадку інтеграція з Google Places API стане важливим елементом функціональності, адже вона дозволить отримувати актуальну інформацію про різні заклади та локації безпосередньо з бази Google. Це забезпечить користувачам доступ до найсвіжіших даних про місця, що їх цікавлять, включаючи їхні адреси, рейтинги, категорії, зображення та інші корисні деталі. Такий підхід допоможе не лише покращити точність результатів, а й гарантувати, що користувач отримає найактуальнішу інформацію про кожен об’єкт.

Важливим аспектом роботи платформи має бути зручність взаємодії з пошуком. Користувачеві буде достатньо ввести назву міста або конкретного місця в поле пошуку, після чого система автоматично підбере найвідповідніші

результати. Інформація має відображатися у вигляді карток, що міститимуть усі основні характеристики локації: назву, вулицю, рейтинг, фотографії та можливість додати її у список планів. Завдяки цьому кожен користувач зможе швидко ознайомитися з пропозиціями та вибрати найкращі варіанти відповідно до своїх потреб.

Окрім цього, інтеграція з Google API допоможе зробити процес пошуку більш динамічним. Дані про місця повинні оновлюватися в реальному часі, щоб користувачі завжди отримували тільки найактуальнішу інформацію. Це стане особливо корисним у випадках, коли необхідно дізнатися про зміни у розкладі роботи закладу, наявність вільних місць у готелях або актуальні відгуки відвідувачів.

Таким чином, продумана система пошуку повинна забезпечити не лише швидкість і точність отримання інформації, а й зробити процес планування подорожі максимально зручним і приємним. Завдяки інтеграції з Google API користувачі матимуть можливість легко знаходити потрібні локації та отримувати про них усю необхідну інформацію в одному місці, що значно спростить організацію мандрівок та заощадить час.

Також, система повинна передбачати зручне керування списком планів. У вкладці «Мої плани» користувач зможе переглядати всі збережені місця, включаючи готелі, музеї, пам’ятники та інші локації. Кожне місце повинно бути представлено у вигляді картки з назвою, адресою, зображенням та двома кнопками: «Видалити» та «Прокласти маршрут». Функція видалення працюватиме асинхронно, що забезпечить швидку взаємодію без перезавантаження сторінки.

Крім цього, платформа повинна передбачати можливість реєстрації та авторизації. Це дозволить користувачам зберігати власний список планів у базі даних та мати доступ до нього з будь-якого пристрою.

Загалом, ефективність платформи для пошуку та планування подорожей залежить від швидкості доступу до інформації, інтуїтивної організації функцій та простоти взаємодії з системою. Гармонійне поєднання цих аспектів дозволяє користувачам легко знаходити цікаві місця та створювати власні списки планів.

3.3. Розробка дизайну за допомогою Figma

3.3.1. Вибір кольорової схеми та шрифтів

Вибір кольорової схеми та шрифтів для планувальника подорожей має безпосередній вплив на зручність користування додатком, тому цей аспект розробки є надзвичайно важливим. Колірна палітра та типографіка повинні бути не лише естетично привабливими, а й сприяти швидкому орієнтуванню та комфортному сприйняттю інформації.

У процесі вибору візуального стилю враховувалась не лише візуальна привабливість, а й функціональність кожного елементу: кольорів, шрифтів та фону.

У дизайні буде використано теплу, природну кольорову палітру з відтінками жовтого, бежевого та коричневого (#fbe4ad, #fbecb5, #4c2c12), що викликають асоціації з натуральністю, затишком і домашньою атмосферою. Для ключових елементів інтерфейсу — таких як навігація, кнопки, форми та картки товарів — застосовуватимуться м’які пастельні тони з акцентами темного шоколаду, що створюють контраст і фокус без агресивності.

Кнопки, зокрема, матимуть насичене тепле забарвлення з чіткими hover-ефектами (наприклад, перехід із жовтого до глибокого червоного #f04e4e або глибокого синього #004f60), що додає інтерфейсу інтерактивності без візуального перевантаження. Такі кольори добре вписуються у загальний настрій ресурсу — затишок, довіра, ручна робота.

Для основного тексту використано Roboto — сучасний, нейтральний шрифт із чудовою читабельністю. Він підходить як для коротких підписів, так і для довгих описів товарів або інструкцій. Для заголовків застосовано Poppins — геометричний і елегантний, він чудово контрастує з основним шрифтом і створює візуальні акценти на заголовках секцій, діях або ключових повідомленнях.

Колір тексту — темний сірий #333, що забезпечує відмінний контраст зі світлим фоном, не створюючи втоми для очей. Міжрядковий інтервал (line-height) встановлено на рівні 1.6 — це покращує зчитуваність, особливо на мобільних пристроях або при швидкому перегляді інформації.

3.3.2. Створення макету інтерфейсу

Створення макету інтерфейсу в Figma є важливим етапом у розробці веб-сайтів та веб-додатків. Оскільки дизайн впливає на взаємодію користувачів з платформою, необхідно ретельно продумати кожен елемент інтерфейсу, щоб він був не тільки естетичним, але й зручним для користувачів. У цьому випадку, макет буде включати такі ключові розділи, як пошук подорожі, результати пошуку, а також форми для реєстрації та входу.

Для розділу пошуку подорожі в Figma буде створено чистий та функціональний інтерфейс, який подано на рисунку 3.5. Він дозволить користувачам легко вводити місце призначення за допомогою текстового поля з підказкою "Введіть місто подорожі...". Під час введення назви міста буде з’являтись випадаючий список із підказками (autocomplete-дропдаун), що допоможе швидко знайти потрібне місто, навіть якщо користувач не пам’ятає точну назву або допускає помилки в написанні.

Це не лише зекономить час, але й зменшить імовірність помилок, покращуючи загальний досвід користування інтерфейсом. Кожна підказка в дропдауні буде візуально чітко відокремлена та матиме просту структуру (назва міста + країна), для полегшення орієнтації.

Кнопка "Пошук" буде виділена яскравим кольором, щоб привертати увагу й забезпечити наочність і доступність дії. Інтерфейс також відображатиме результати пошуку та міститиме категорії, такі як "Готелі", "Музеї", "Пам'ятники" тощо, щоб користувач міг швидко вибрати бажаний тип локації.

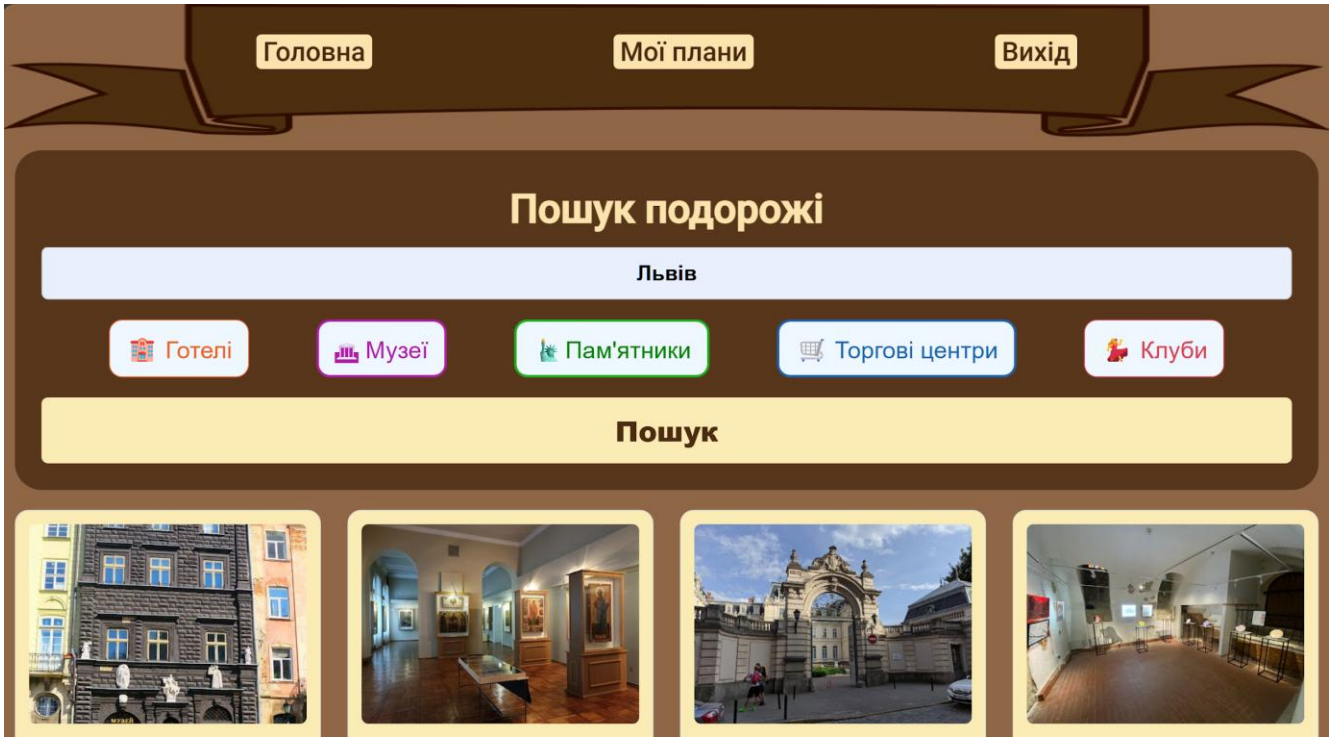


Рис. 3.5. Макет головної сторінки “планувальника подорожей”

Панелі для кожної категорії будуть використовувати ті ж самі кольори, що й для вкладок, щоб зберегти єдність дизайну. Карти з результатами будуть організовані в "картках", що зробить перегляд інформації зручнішим і швидким,(див. рисунок 3.7).

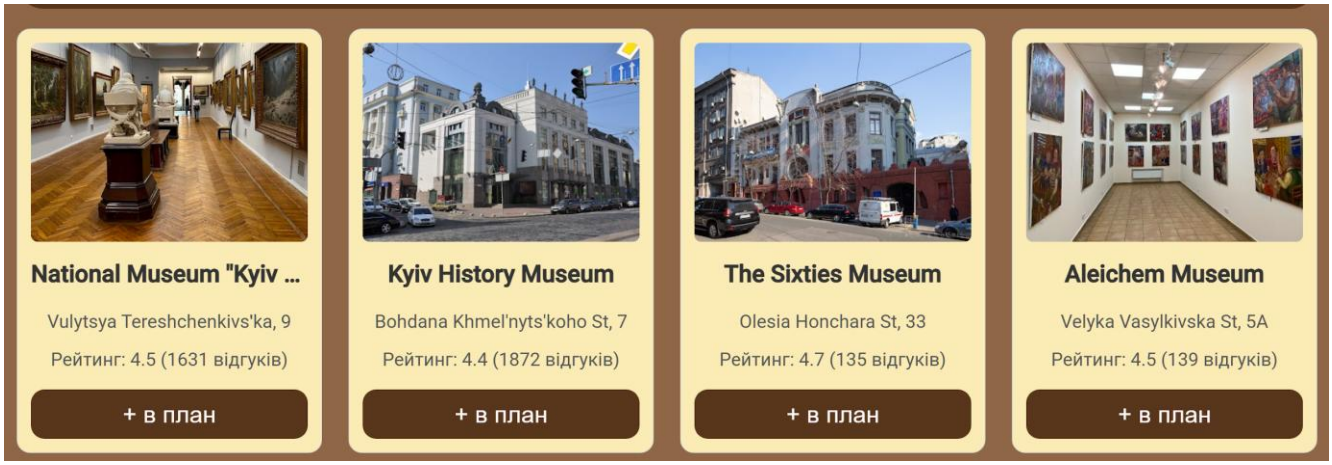


Рис. 3.7. Макет карток з результатами

Форма входу та реєстрації на сайті, яка буде розташована в окремих контейнерах, буде організована в простому, зрозумілому вигляді з чіткими

підказками та полями для введення електронної пошти та пароля, як на рисунку 3.8. Для користувачів, які ще не мають акаунту, буде надано посилання на реєстрацію, яке також буде помітно виділено.

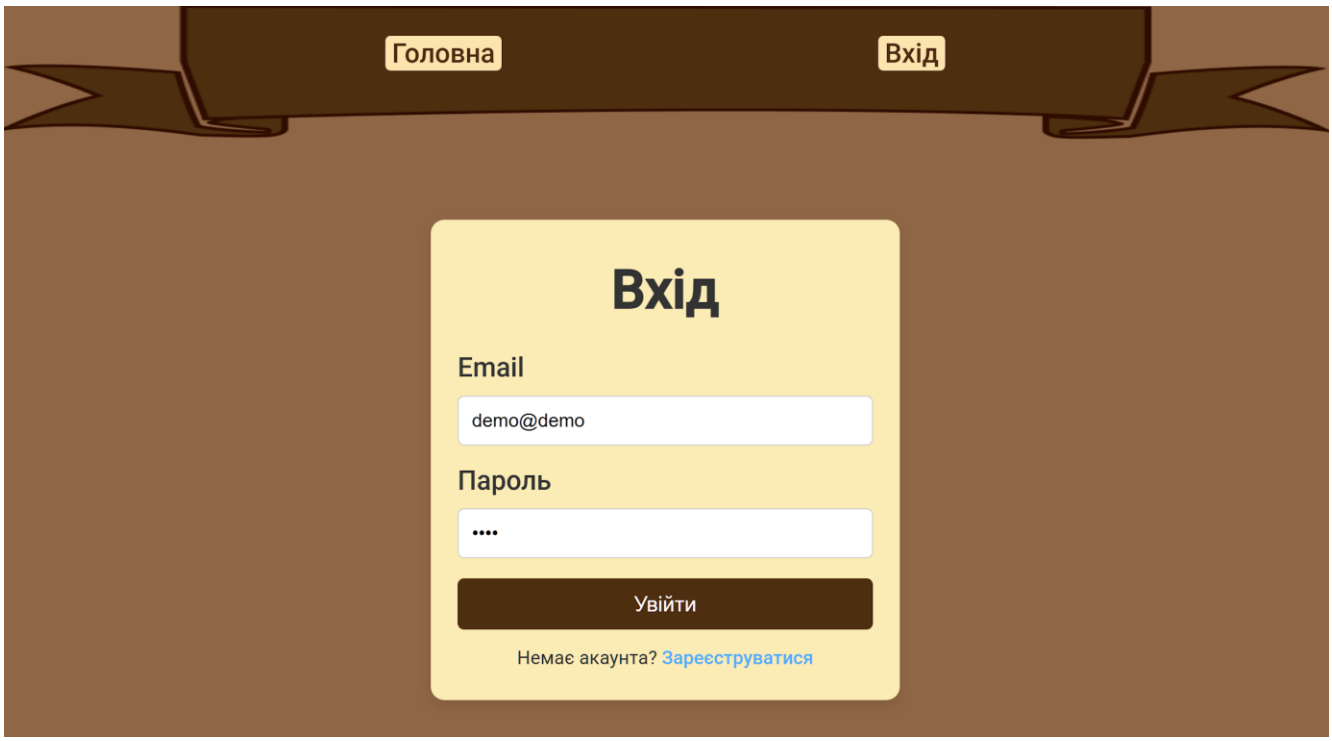


Рис. 3.8. Макет форми входу та реєстрації на сайті

Сторінка "Мої плани", де користувач зможе переглядати свої плани, буде включати "картки планів", які міститимуть зображення, назву та адресу кожного плану який подано на рисунку 3.9. Кожна картка буде мати кнопки для видалення плану або побудови маршруту. Цей макет буде включати в себе елементи взаємодії з користувачем, щоб він міг без проблем керувати своїми подорожами.

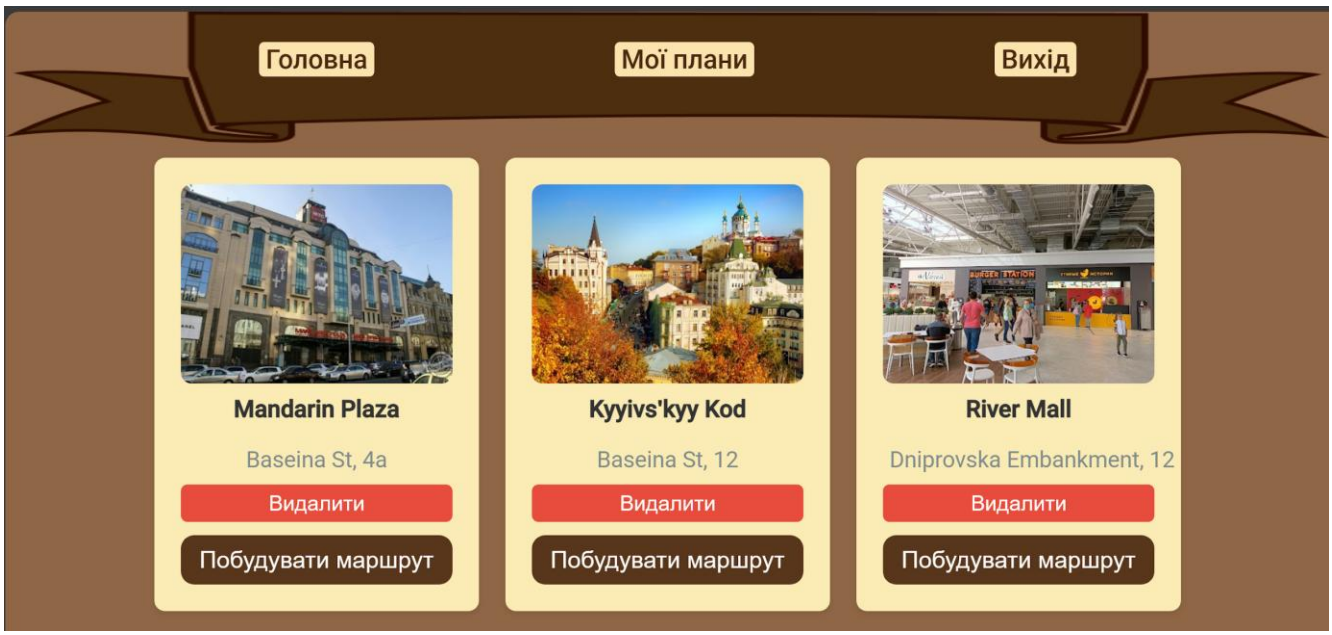


Рис. 3.9. Макет сторінки - “мої плани”

Фігма дозволяє на всіх етапах дизайну тестувати різні елементи і виводити їх на екран в реальному часі, що дає можливість точно налаштувати компоненти під потреби користувача та під вимоги проекту. Макет буде враховувати всі ці деталі, щоб створити інтерфейс, який поєднує в собі зручність, стиль і функціональність, залишаючи простір для подальших адаптацій та змін.

3.4. Налаштування робочого середовища та встановлення необхідних технологій

Створення веб-застосунку починається з налаштування робочого середовища та встановлення необхідних технологій. Використання правильних інструментів значно спрощує процес розробки та дозволяє працювати ефективніше.

Перше, що потрібно зробити, — встановити Visual Studio Code (VS Code), який стане основним редактором коду. Його зручний інтерфейс, велика кількість розширень і підтримка різних мов програмування роблять його чудовим вибором для веб-розробки.

Наступним кроком буде встановлення Node.js – середовища виконання JavaScript, що дозволяє запускати код на сервері. Разом із ним автоматично

встановлюється npm (Node Package Manager), який використовується для управління пакетами та залежностями проєкту. Переконатися в успішному встановленні можна, виконавши в терміналі команди node -v та npm -v, які повинні вивести відповідні версії.

Далі необхідно створити папку для проєкту та ініціалізувати його за допомогою npm init -y. Це створить файл package.json, у якому зберігатимуться всі залежності та налаштування проєкту. У ньому можна одразу вказати скрипти для запуску сервера та інших процесів, що спрощує роботу з проєктом.

Серед необхідних пакетів, які потрібно встановити, варто виділити Express — мініфреймворк для створення серверної частини, dotenv для роботи зі змінними середовища. Встановлення цих пакетів виконується командою npm install express dotenv.

Після цього можна налаштувати базову структуру проєкту: створити папки routes, models, middleware, public і views, а також головний файл сервера index.js.

Ці кроки створять основу для подальшої роботи над веб-застосунком, дозволяючи швидко перейти до розробки серверної частини, підключення бази даних та реалізації функціоналу.

3.5. Розробка серверної частини та підключення бази даних

Процес розробки серверної частини сучасного веб-застосунку є важливим етапом у створенні функціонального та надійного програмного продукту. У межах цього етапу реалізуються ключові механізми взаємодії з базою даних, обробки HTTP-запитів, керування сесіями користувачів та забезпечення безпеки, зокрема через шифрування паролів. Побудова логічної структури та взаємозв’язків між компонентами додатку дозволяє досягти високої продуктивності, масштабованості та підтримуваності системи.

Структура компонентів веб-застосунку «Планувальник подорожей» побудована на основі підходу поділу обов’язків між клієнтською та серверною частинами, як зображено на рисунку 3.10.

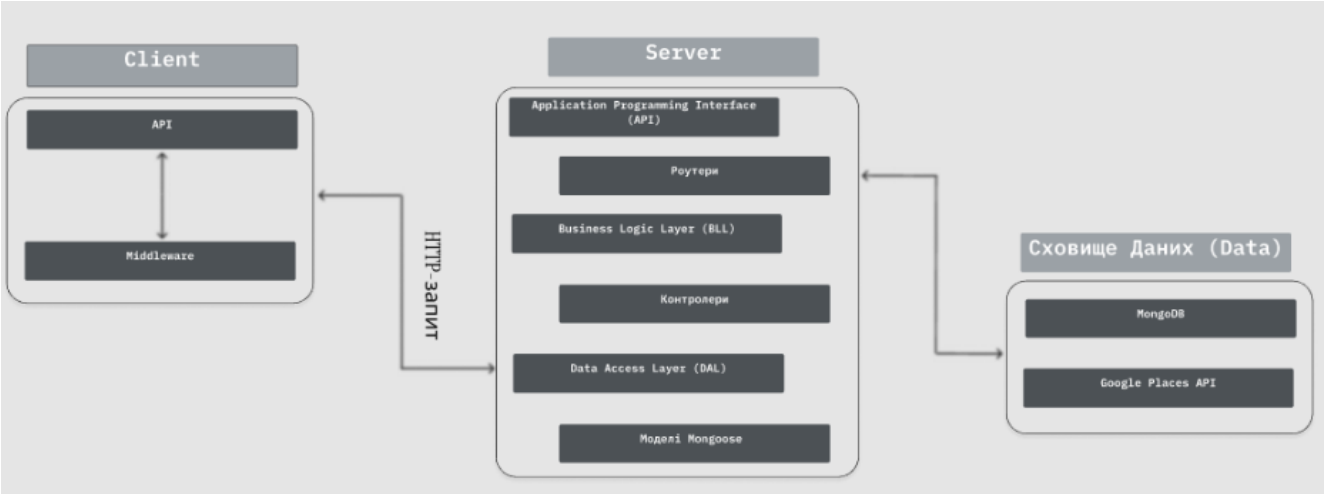


Рис. 3.10 Структура компонентів

Невід’ємною частиною архітектури веб-застосунку є структура бази даних, яка визначає спосіб зберігання, організації та доступу до інформації. У даному проєкті реалізовано документно орієнтовану модель на основі MongoDB, що дозволяє зберігати дані у вигляді гнучких об’єктів. Основними сутностями бази даних є користувачі, плани подорожей та зовнішні дані з API, між якими встановлено логічні зв’язки. Структура бази даних розроблена таким чином, щоб забезпечити ефективну роботу з обліковими записами, створенням та збереженням маршрутів, а також динамічне завантаження інформації про місця на основі даних із зовнішнього API. Залом база даних нашого веб-додатку має таку структуру, як зображено на рисунку 3.11.

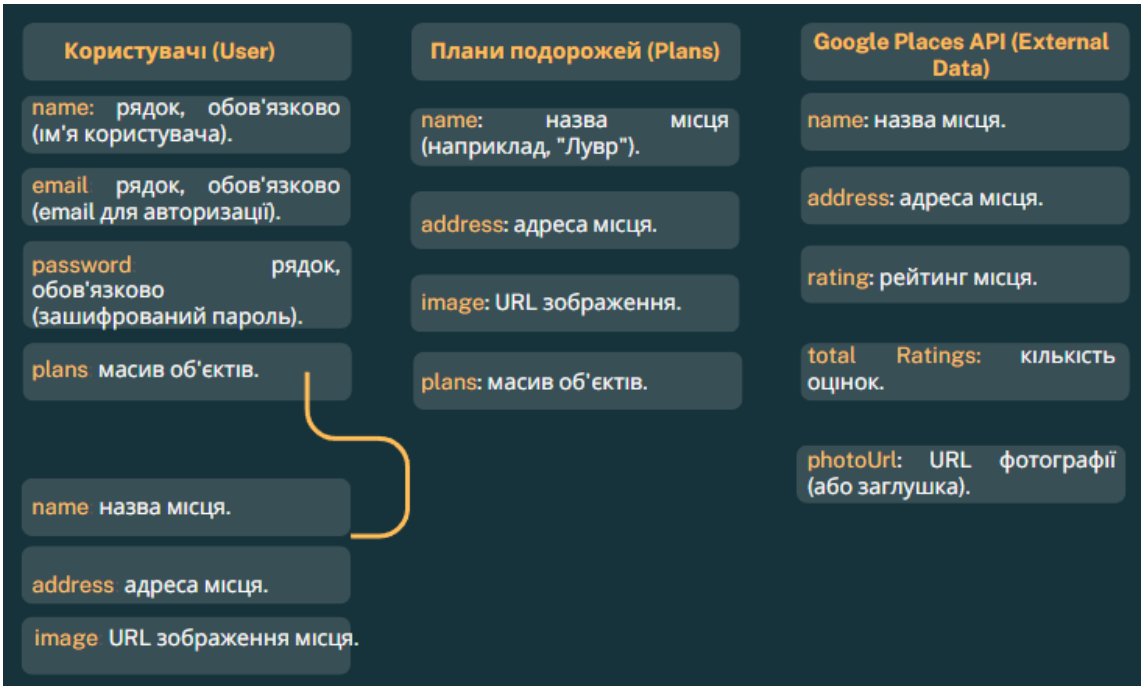


Рис. 3.11 Структура бази даних

Таким чином, розробка серверної частини веб-застосунку передбачає не лише технічну реалізацію функціональності, а й побудову логічної, узгодженої структури даних і компонентів. Наступним етапом є безпосереднє налаштування Express, підключення до бази даних MongoDB через Mongoose, реалізація механізмів аутентифікації, сесій та шифрування паролів, що забезпечують цілісність та безпечну взаємодію з користувачем. Спочатку імпортуються необхідні модулі:

```
const express = require('express')
const mongoose = require('mongoose')
const session = require('express-session')
const mongoSession = require('connect-mongodb-session')(session)
const dotenv = require('dotenv').config()
```

Express використовується для створення сервера, Mongoose – для роботи з базою даних, а express-session разом із connect-mongodb-session зберігають сесії користувачів у MongoDB. Файл .env містить змінні середовища, такі як URI бази даних.

Підключення до MongoDB відбувається перед запуском сервера:

```
async function start() {
    try {
```

```
        await mongoose.connect(uri)
        app.listen(3000, () => console.log('Server started'))
    } catch (error) {
        console.error(error)
    }
}
start()
```

Це гарантує, що сервер не запуститься, якщо база даних недоступна.

Створюється обробник сесій:

```
const store = mongoSession({
    collection: 'sessions',
    uri: uri
})
app.use(session({
    secret: 'secret',
    resave: false,
    saveUninitialized: false,
    store
}))
```

Це дозволяє зберігати сесії користувачів у базі даних.

Нарешті, додаються маршрути:

```
const routeHome = require('./routes/home')
const routeAuth = require('./routes/auth')

app.use('/', routeHome)
app.use('/', routeAuth)
```

Ці маршрути обробляють запити до головної сторінки та авторизації користувачів.

У даному випадку для обробки запитів до головної сторінки веб-застосунку використовуємо Router з Express .

```
const { Router } = require("express");
const router = Router();
```

Роутер використовується для організації маршрутів у додатку, що дозволяє створювати логічні групи маршрутів, зручні для розділення функціональностей.

Далі, обробляється запит GET на головну сторінку:

```
router.get("/", (req, res) => {
    res.render("home", {
      title: "Планувальник подорожей",
```

```
    });
  });
```

У цьому фрагменті код відповідає за рендеринг шаблону "home.hbs" за допомогою функції res.render(). Шаблон Handlebars отримує об'єкт з даними, в якому передається заголовок сторінки, що потім відображається у відповідному HTML-коді. Це дозволяє динамічно генерувати контент сторінки в залежності від переданих параметрів.

Таким чином, роутер головної сторінки забезпечує відображення шаблону, організує передавання необхідних даних для рендерингу та підключається до сервера для обробки запитів.

Підключення до бази даних та збереження персональних даних користувача є важливими етапами при розробці серверної частини. Використовуючи Mongoose, можна легко створювати схеми та моделі для зберігання даних у MongoDB.

схема користувача:

```
const UserSchema = new Schema({
    name: String,
    email: String,
    password: String,
    plans: [
        {
            name: String,    // Назва місця
            address: String, // Адреса місця
            image: String    // URL зображення
        }
    ]
});
```

Схема UserSchema містить кілька полів: name, email та password, крім того, поле plans зберігає масив об'єктів, кожен з яких описує план подорожі. Це дозволяє користувачам додавати плани та зберігати їх у базі даних.

Далі розглянемо частину коду для реєстрації нового користувача та збереження його даних:

```
router.post('/registration', async (req, res) => {
    const { name, email } = req.body
    const candidate = await User.findOne({ email }).lean()

    if (candidate) {
        res.send('Користувач з таким email вже існує')
```

```
    } else {
        const password = await bcrypt.hash(req.body.password, 10)
        const newUser = new User({ name, email, password, plans: [] })
        await newUser.save()
        res.redirect('/login')
    }
});
```

У цьому фрагменті коду, коли користувач намагається зареєструватися, спершу перевіряється, чи існує користувач з таким самим email у базі даних. Якщо так, система виводить повідомлення про те, що такий користувач вже зареєстрований. Якщо ж користувача немає, пароль шифрується за допомогою bcrypt, створюється новий користувач з вказаними даними, а потім зберігається в базу даних. Після успішної реєстрації користувач перенаправляється на сторінку входу.

Завдяки цьому підходу можна безпечно зберігати користувацькі дані в базі даних та забезпечити їх шифрування для запобігання несанкціонованому доступу до конфіденційної інформації.

Наступним етапом є інтеграція зовнішніх API для отримання даних, в обробник маршруту для пошуку місць. У цьому випадку використаємо Google Places API для пошуку місць за вказаними параметрами.

Код починається з отримання параметрів пошуку з запиту. Якщо параметр location не переданий, за замовчуванням виконується пошук у Києві. Те ж саме стосується параметра category, де за замовчуванням шукаються готелі:

```
const location = req.query.location || "Київ";
const category = req.query.category || "lodging";
```

Далі формуємо запит для Google Places API, де ми комбінуємо місце та категорію у вигляді рядка:

```
const placeName = `${location} ${categoriesMap[category] || "hotel"}`;
```

Формуємо пошуковий запит

```
const url = `https://maps.googleapis.com/maps/api/place/textsearch/json?query=${encodeURIComponent(placeName)}&key=${apiKey}`;
```

Цей запит виконується за допомогою fetch, який отримує результат від API:

```
const response = await fetch(url);
```

```
const data = await response.json();
```

Якщо дані знайдені, вони обробляються та форматуються для зручного використання на клієнті. Для кожного місця створюється об'єкт, який містить назву, адресу, рейтинг, кількість відгуків, а також URL фото місця, якщо воно доступне:

```
const places = data.results.map(place => ({
    name: place.name, // Назва місця
    address: place.formatted_address, // Адреса
    rating: place.rating || "Немає рейтингу", // Рейтинг
    photoUrl: place.photos
      ?
`https://maps.googleapis.com/maps/api/place/photo?maxwidth=400&photoreference=${place
.photos[0].photo_reference}&key=${apiKey}`
      : "/images/placeholder.jpg", // Отримуємо фото або показуємо заглушку
    arr: place
  }));
```

Якщо результатів не знайдено, повертається повідомлення про помилку:

```
if (!data.results || data.results.length === 0) {
    return res.json({ error: "Місця не знайдено" });
  }
```

Крім того, якщо виникає помилка під час запиту до API, сервер повертає повідомлення про помилку:

```
catch (error) {
    res.status(500).json({ error: "Помилка сервера" });
  }
```

Цей підхід дозволяє ефективно обробляти пошукові запити користувачів, інтегруючи зовнішні сервіси для отримання інформації про місця, що можуть бути використані в додатку.

Щодо додавання та видалення місць з плану подорожей користувача використовується POST-запит на маршрут /add-to-plan. В обробнику цього запиту ми отримуємо дані з тіла запиту, зокрема фото, назву та адресу місця. Ці дані передаються в метод addToPlan, який додає місце до плану користувача:

```
router.post("/add-to-plan", async (req, res) => {
    try {
      const { photoUrl, name, address } = req.body;
      await req.user.addToPlan(photoUrl, name, address);
      res.json({ message: "Місце додано в план!" });
    } catch (error) {
```

```
    res.status(500).json({ error: "Помилка сервера" });
  }
});
```

Метод addToPlan — це частина моделі користувача, яка відповідає за додавання нового місця до масиву plans, що зберігається в базі даних. У випадку успіху клієнт отримує відповідь з повідомленням про те, що місце додано, а у разі помилки сервер відправляє код статусу 500 з відповідним повідомленням про помилку.

Також передбачено видалення місця з плану. Для цього використовується DELETE-запит на маршрут /remove-plan/:id, де :id — це ідентифікатор місця, яке потрібно видалити з плану. Метод removePlan викликається для видалення місця, і після цього сервер відправляє оновлений список планів:

```
router.delete("/remove-plan/:id", async (req, res) => {
    res.json(await req.user.removePlan(req.params.id));
});
```

Цей код дозволяє користувачам зручно додавати нові місця до своїх подорожей та видаляти їх при необхідності. Операції виконуються асинхронно, що забезпечує ефективність та відсутність блокування виконання програми під час обробки запитів до бази даних.

Підсумовуючи, розробка серверної частини веб-застосунку включає налаштування серверу за допомогою Express, інтеграцію з MongoDB через Mongoose, збереження сесій користувачів у базі даних та забезпечення безпеки даних. Використовуючи Express, можна легко налаштувати маршрути та обробку запитів, що дозволяє створювати гнучку архітектуру для веб-застосунку.

MongoDB разом із Mongoose забезпечують ефективне зберігання та обробку даних, таких як персональні дані користувачів та їхні плани. Моделі та схеми дозволяють зручно працювати з даними, а також підтримувати різноманітні функції, такі як додавання, видалення та оновлення інформації.

Застосування сесій за допомогою express-session та connect-mongodb-session дозволяє зберігати сесії користувачів у MongoDB, що є важливим аспектом для забезпечення автентифікації та збереження стану користувача між запитами.

Шифрування паролів за допомогою бібліотеки bcrypt додає ще один рівень безпеки, запобігаючи збереженню паролів у відкритому вигляді. Це дозволяє безпечно зберігати конфіденційну інформацію, захищаючи її від несанкціонованого доступу.

Інтеграція з зовнішніми API, такими як Google Places, забезпечує можливість отримання актуальних даних про місця та використання їх у додатку для зручного планування подорожей. Пошук місць за вказаними параметрами допомагає користувачам зберігати тільки релевантну інформацію, яка підходить для їхніх планів.

Використання асинхронних запитів до бази даних та зовнішніх API забезпечує ефективність роботи серверу без блокувань, що позитивно впливає на загальну продуктивність та швидкість відгуку веб-застосунку. Ці функціональні можливості формують надійну та масштабовану серверну частину для будь-яких веб-застосунків, орієнтованих на обробку та збереження даних користувачів.

3.6. Реалізація дизайну інтерфейсу за допомогою HTML та CSS

3.6.1 Динамічне створення HTML-сторінок через Handlebars

Для реалізації дизайну інтерфейсу веб-застосунку, використовується технологія шаблонів Handlebars, яка дозволяє динамічно створювати HTML-сторінки. Завдяки цій технології, можна ефективно розділяти структуру сайту на частини (partials) та використовувати макети (layouts), що робить код чистим і зручним для підтримки. Кожна частина веб-сторінки, така як head, header, footer, може бути окремо визначена, а потім підключена до основного макету сторінки.

Основний макет сторінки виглядає наступним чином:

```
<!DOCTYPE html>
<html lang="uk">
  {{> head}} <!-- Підключаємо head -->
  <body>
    {{> header}} <!-- Підключаємо header -->
    {{{body}}} <!-- Підключаємо основний вміст -->
    {{> footer}} <!-- Підключаємо footer -->
  </body>
</html>
```

Цей макет використовує частини, визначені окремо, що дозволяє гнучко керувати вмістом кожної сторінки. Частини, як-от head, визначають основні метадані сторінки, такі як заголовок та підключення стилів, а також загальні елементи інтерфейсу, як-от шрифти та стилі:

```
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>{{title}}</title>
  <link
href="https://fonts.googleapis.com/css2?family=Roboto:wght@400;500&family=Poppins:wgh
t@600&display=swap" rel="stylesheet">
  <link rel="stylesheet" href="/css/styles.css">
</head>
```

footer надає просту структуру для відображення копірайту або іншої загальної інформації на сторінці:

```
<footer>
  <p>&copy; 2025 Планувальник подорожей</p>
</footer>
```

Використання Handlebars для створення сторінок зручне завдяки можливості створювати динамічні елементи, такі як списки результатів пошуку або інформація про плани подорожей. Наприклад, для головної сторінки (home) надається шаблон, який включає форму для пошуку місць, а також розділ з результатами пошуку, що включає кнопки для переключення між категоріями (готелі, музеї, пам'ятники тощо):

```
<main>
  <section class="search">
    <form id="searchForm">
      <h1>Пошук подорожі</h1>

      <div class="autocomplete-wrapper">
        <input type="text" id="location" placeholder="Введіть місто подорожі.."
required>
        <div id="autocomplete-dropdown" class="autocomplete-dropdown"></div>
      </div>

      <div class="tabs">
        <button class="tab-button btn-secondary active" data-tab="lodging" data-
category="lodging" style="background-color:aliceblue; border: 1px solid; border-color:
rgb(233 97 43); color:rgb(233 97 43)">🏨 Готелі</button>
```

```
        <button     class="tab-button     btn-secondary"     data-tab="museum"     data-
category="museum" style="background-color:aliceblue; border: 2px solid; border-color:
#af1fa8; color:#af1fa8">🏛️ Музеї</button>
        <button     class="tab-button     btn-secondary"     data-tab="landmark"     data-
category="landmark" style="background-color:aliceblue; border: 2px solid; border-color:
#1faf1f; color:green">🗿Пам'ятники</button>
        <button   class="tab-button   btn-secondary"   data-tab="shopping_mall"   data-
category="shopping_mall" style="background-color:aliceblue; border: 2px solid; border-
color: #1f62af; color:#1f62af">🛒 Торгові центри</button>
        <button    class="tab-button    btn-secondary"    data-tab="night_club"    data-
category="night_club"  style="background-color:aliceblue;  border:  1px  solid;  border-
color: rgb(217 64 85); color:rgb(217 64 85)">🕺 Клуби</button>
      </div>
      <button type="submit" class="btn-primary">Пошук</button>
    </form>


    <div class="results">
      <div class="tab-content">
        <div id="lodging" class="tab-panel active">
          <div class="loading" style="display: none;">Завантаження...</div>
          <div id="lodging-list" class="cards"></div>
        </div>
        <div id="museum" class="tab-panel">
          <div class="loading" style="display: none;">Завантаження...</div>
          <div id="museum-list" class="cards"></div>
        </div>
        <div id="landmark" class="tab-panel">
          <div class="loading" style="display: none;">Завантаження...</div>
          <div id="landmark-list" class="cards"></div>
        </div>
        <div id="shopping_mall" class="tab-panel">
          <div class="loading" style="display: none;">Завантаження...</div>
          <div id="shopping_mall-list" class="cards"></div>
        </div>
        <div id="night_club" class="tab-panel">
          <div class="loading" style="display: none;">Завантаження...</div>
          <div id="night_club-list" class="cards"></div>
        </div>
      </div>
    </div>
  </section>
</main>
```

При реєстрації або вході користувачеві пропонується форма для введення особистих даних. Відповідно до шаблону для сторінки реєстрації та логіну, передбачено заповнення полів, таких як ім'я, email та пароль:

```
<div class="container">
  <div class="form-box">
    <h1>Реєстрація</h1>
    <form action="/registration" method="post">
      <label for="username">Ім'я</label>
      <input type="text" id="username" name="name" placeholder="Введіть ім'я">
      <label for="email-reg">Email</label>
      <input type="email" id="email-reg" name="email" placeholder="Введіть email">
      <label for="password-reg">Пароль</label>
      <input  type="password"  id="password-reg"  name="password"  placeholder="Введіть
пароль">
      <button type="submit" class="btn-primary">Зареєструватися</button>
      <p>Вже є акаунт? <a href="/login">Увійти</a></p>
    </form>
  </div>
</div>
```

Шаблон для сторінки планів подорожей дозволяє відображати список планів користувача. Для кожного плану відображаються його деталі, включаючи назву, адресу та можливість видалення або створення маршруту:

```
<main>
  <section class="plans">
    <h1>{{h1}}</h1>
    <div class="plan-list" id="planList">
      {{#each plans}}
        <div class="plan-card">
          <img                    src="{{image}}"                    alt="{{name}}"
onerror="this.src='/images/placeholder.png'">
          <h3>{{name}}</h3>
          <p>{{address}}</p>
          <button class="btn-remove" id="{{id}}">Видалити</button>
          <button class="btn-route btn-secondary">Побудувати маршрут</button>
        </div>
      {{/each}}
    </div>
  </section>
</main>
```

Таким чином, реалізація дизайну інтерфейсу через Handlebars дозволяє зручно і ефективно керувати контентом, структуруючи сторінки, підключаючи спільні частини та забезпечуючи динамічний вміст за допомогою шаблонів і JavaScript.

3.6.2 Робота зі стилями та адаптивністю через CSS

Створення адаптивного і стильного інтерфейсу за допомогою CSS дозволяє створювати привабливі та зручні для користувача веб-сторінки, які працюють на різних пристроях та екранах. Ось як можна реалізувати ефективну адаптивність і стиль через CSS, використовуючи прості, але потужні техніки.

Основи стилізації веб-сторінки включають в себе використання основних властивостей CSS для задавання стилю елементів. Наприклад, на самому початку задається базовий стиль для всіх елементів через універсальний селектор *, що визначає відсутність відступів, та налаштовує коректне відображення коробки для всіх елементів за допомогою box-sizing: border-box. Це дозволяє забезпечити передбачуване відображення елементів на всіх розмірах екранів.

Для загального вигляду сторінки також важливо встановити фон, шрифти і кольори, що створюють загальну атмосферу сайту. Наприклад, фон сторінки може бути заданий через background-image, що дозволяє створити стильний фон за допомогою зображення. Встановлення шрифтів через font-family з використанням популярних веб-шрифтів Google Fonts додає естетичної привабливості. Текст на сторінці можна зробити чітким та читабельним за допомогою правильного налаштування кольору та міжрядкового інтервалу за допомогою color і line-height.

Застосування стилів для основних елементів інтерфейсу, таких як заголовки, меню навігації, форми пошуку та результати пошуку, також є важливою частиною проектування веб-сторінки. Для навігаційного меню можна використовувати списки без маркерів, які стилізуються через nav ul, де кожен елемент списку li розташовується горизонтально через display: inline-block. Крім того, кнопки для навігації мають бути легкими в інтеракції, і можна додавати ефекти при наведенні через властивість :hover, щоб покращити досвід користувача.

Адаптивність інтерфейсу забезпечується за допомогою медіа-запитів CSS. Наприклад, для мобільних пристроїв, коли розмір екрана менший за 768px, елементи можуть змінювати свою орієнтацію чи розміри, щоб виглядати зручніше. Для цього можна використовувати медіа-запити, наприклад, за допомогою @media (max-width: 768px), щоб змінити дисплей карток на вертикальний або відображати

вкладки як горизонтальний скрол. Це дозволяє адаптувати вигляд веб-сторінки під різні типи пристроїв, будь то мобільний телефон або планшет.

Динамічні елементи, такі як вкладки для категорій пошуку, можуть бути реалізовані за допомогою стилів, що керують видимістю різних панелей вкладок за допомогою властивості display. Використання таких властивостей дозволяє користувачеві вибирати між різними категоріями результатів, не перезавантажуючи сторінку, а за допомогою плавних переходів, таких як transition, можна створити гармонійний та інтерактивний досвід користування.

Окрім цього, варто зазначити важливість використання CSS для стилізації форм та кнопок. Для форми пошуку важливо забезпечити правильне вирівнювання елементів, доступність і зручність для користувача, використовуючи відступи та радіуси кута. Кнопки, наприклад, можуть змінювати колір при наведенні, щоб сигналізувати про можливість їх натискання.

Один із найбільш важливих аспектів реалізації дизайну через CSS — це забезпечення інтерактивності та зручності користувача при взаємодії з елементами інтерфейсу. Для цього використовуються ефекти наведення, анімації при зміні стану елементів, а також стилі для забезпечення кращого відображення на малих екранах через адаптивний дизайн. Такий підхід дозволяє створювати не лише гарні, але й функціональні інтерфейси, що відповідають сучасним вимогам.

Таким чином, через правильне використання CSS можна створити не лише стильний, але й адаптивний інтерфейс для будь-якого веб-додатку, що дозволить користувачам насолоджуватися комфортним досвідом взаємодії з сайтом на різних пристроях та екранах.

Отже, CSS-стилі, що визначають глобальні налаштування для всього веб-додатку, починаються з блоку, який усуває відступи та межі за замовчуванням:

```
* {
  margin: 0;
  padding: 0;
  box-sizing: border-box;
}
```

Цей стиль гарантує, що всі елементи мають нульові відступи та внутрішні відстані, а також використовують стандартну модель коробки для розмірів. Це

дозволяє забезпечити більш точний контроль над розмірами елементів та їх взаємодією.

Основний фон сторінки визначено через зображення, яке фіксується при прокручуванні, що дозволяє створити ефект фону на весь екран:

```
  html {
    font-family: 'Roboto', sans-serif;
    color: #333;
    line-height: 1.6;
    padding: 0 0px;
    background-image: url('/images/background.png');
    background-size: cover;
    background-position: center;
    background-attachment: fixed;
    background-repeat: no-repeat;
  }
```

Зображення фону покриває весь екран, а його позиція залишається незмінною при прокручуванні сторінки, що створює враження "статичності" фону на тлі зміни контенту.

Стилізація хедера включає використання фонового зображення, що задає візуальний характер верхній частині сторінки. У стилях header використовується:

```
  header {
    height: 100px;
    width: 100%;
    position: fixed;
    top: 0;
    display: flex;
    background-image: url(/images/header-background.png);
    background-size: 100% 100%;
    background-position: center;
    background-repeat: no-repeat;
    justify-content: center;
}
```

Фон реалізовано як растрове зображення, що розтягується на всю ширину та висоту хедера завдяки background-size: 100% 100%. Це забезпечує статичний, візуально привабливий заголовок, який не повторюється (no-repeat) і завжди центрований. Властивість position: fixed закріплює хедер у верхній частині сторінки, що робить його завжди доступним при прокрутці.

Навігаційне меню реалізоване у вигляді горизонтального списку:

```
  nav ul {
    list-style: none;
    display: flex;
    padding: 20px;
    justify-content: space-evenly;
  }

  nav ul li {
    display: inline-block;
    margin: 0 10px;
}

  nav ul li a {
    color: #492910;
    text-decoration: none;
    font-weight: 500;
    font-size: 20px;
    background-color: #fbe4ad;
    border-radius: 4px;
    padding: 1px 5px;
}
```

Тут використано display: flex і justify-content: space-evenly, що рівномірно розподіляє пункти меню по ширині контейнера, забезпечуючи симетричний і гармонійний вигляд. Відсутність маркерів (list-style: none) та внутрішні відступи (padding: 20px) покращують загальну читабельність.

Використання inline-block для <li> дозволяє елементам бути в одному рядку з контролем за зовнішніми відступами. Посилання (<a>) оформлені у контрастному стилі: темний текст (#492910) на світло-жовтому фоні (#fbe4ad), із заокругленими кутами й додатковим паддінгом. Це покращує зорове сприйняття та клікабельність меню, роблячи його зручним і привабливим.

Блок пошуку оформлено так, щоб він займав центральну частину екрана та виглядав візуально піднятими над фоном.

```
  .search {
    min-height: calc(100vh - 100px);
    display: flex;
    flex-direction: column;
    justify-content: center;
    text-align: center;
    margin-top: 100px;
    border-radius: 8px;
}
```

Блок .search має мінімальну висоту, що дорівнює висоті вікна мінус хедер (100px) — завдяки calc(100vh - 100px) — що дозволяє розмістити його по центру вільного простору. Завдяки display: flex, flex-direction: column і justify-content: center вміст вирівнюється по вертикалі, а text-align: center відповідає за горизонтальне вирівнювання. border-radius: 8px додає округлення, що робить форму візуально м'якою та сучасною.

Кнопки пошуку мають стиль, що дозволяє змінювати їх колір при наведенні:

```
.search .btn-primary {
  background-color: #fbecb5;
  color: #4c2c12;
  padding: 10px 20px;
  font-size: 21px;
  font-weight: 800;
  border: none;
  border-radius: 5px;
  cursor: pointer;
  transition: background-color 0.3s ease;
}

.search .btn-primary:hover {
  background-color: #f04e4e;
}
```

Картки для результатів пошуку оформлені так, щоб виглядати акуратно і зручно для користувача:

```
.card {
  background-color: #ffffff;
  padding: 20px;
  border-radius: 8px;
  box-shadow: 0 4px 10px rgba(0, 0, 0, 0.1);
  width: 200px;
  text-align: center;
}
```

Зображення на картках адаптовано до розміру, і якщо зображення не доступне, використовується зображення за замовчуванням:

```
.card img {
  width: 100%;
  height: 150px;
  object-fit: cover;
  border-radius: 5px;
}
```

Адаптивність сайту забезпечена через використання медіа-запитів. Наприклад, на екранах меншого розміру, картки та вкладки автоматично змінюють свою орієнтацію, забезпечуючи зручність використання на мобільних пристроях:

```
@media(max-width: 768px){
  .tabs {
    display: flex;
    gap: 10px;
    overflow-x: auto;
    white-space: nowrap;
    padding-bottom: 10px;
    -webkit-overflow-scrolling: touch;
  }

  .cards {
    flex-direction: column;
  }

  .card {
    width: 100%;
  }
}
```

Цей стиль дозволяє елементам адаптуватися до різних розмірів екранів, покращуючи користувацький досвід на мобільних пристроях.

У підсумку, такі CSS-стилі, створюють естетично привабливий та функціональний інтерфейс, де кожен елемент має свою роль. Від хедера до карток місць, всі частини дизайну гармонійно поєднуються для забезпечення зручного і красивого користувацького досвіду. Адаптивність також є важливою частиною цього процесу, дозволяючи сайту коректно відображатися на різних пристроях.

3.7. Використання JavaScript для інтерактивних функцій

JavaScript відіграє ключову роль у створенні інтерактивних функцій, він дозволяє користувачам взаємодіяти з веб-сторінками без необхідності оновлення контенту. Однією з таких функцій є пошук місць на основі введеного користувачем запиту.

Перша важлива частина функціоналу полягає в обробці події відправки форми для пошуку місць. Стандартна поведінка форм у веб-браузерах передбачає перезавантаження сторінки при їх відправці. Для того, щоб уникнути цього, ми

використовуємо метод preventDefault(), який перешкоджає оновленню сторінки. Замість цього викликається функція searchPlaces(), яка відповідає за виконання пошукового запиту на сервер.

```
document.querySelector("form").addEventListener("submit", function (e) {
    e.preventDefault();
    searchPlaces();
});
```

Ще одна важлива інтерактивна функція — це управління вкладками. Користувач може перемикатися між різними категоріями місць (наприклад, готелі, ресторани, атракціони). Для цього необхідно додати обробники подій на всі кнопки вкладок. Після натискання на кнопку відповідна вкладка стає активною, а попередня — деактивованою. Цей процес реалізується за допомогою класів active, які додаються або видаляються з елементів кнопок та панелей вкладок.

```
document.querySelectorAll(".tab-button").forEach(button => {
    button.addEventListener("click", function() {
        // Видаляємо клас "active" у всіх кнопок і панелей вкладок
        document.querySelectorAll(".tab-button").forEach(btn =>
btn.classList.remove("active"));
        document.querySelectorAll(".tab-panel").forEach(panel =>
panel.classList.remove("active"));

        // Додаємо клас "active" лише для натиснутої кнопки та відповідної панелі
        this.classList.add("active");
        document.getElementById(this.dataset.tab).classList.add("active");

        // Запускаємо пошук місць для обраної категорії
        searchPlaces();
    });
});
```

Після того, як користувач вибрав категорію і ввів локацію, необхідно виконати запит до сервера для отримання відповідних місць. Для цього використовується функція searchPlaces(), яка формує URL запиту на сервер і обробляє відповідь у форматі JSON. Після цього результати відображаються на сторінці, а також додаються кнопки для додавання місць у план подорожей.

```
function searchPlaces() {
    let location = document.querySelector("#location").value.trim(); // Отримуємо
значення локації
    let activeTabButton = document.querySelector(".tab-button.active"); // Активна
вкладка
```

```
    let category = activeTabButton ? activeTabButton.getAttribute("data-category") :
"lodging"; // Категорія (за замовчуванням "lodging")

    // Виконуємо запит на сервер
    fetch(`/search-
places?location=${encodeURIComponent(location)}&category=${category}`)
        .then(response => response.json()) // Обробка відповіді
        .then(places => {
            const container =
document.querySelector(`#${activeTabButton.dataset.tab}-list`); // Контейнер для
місць
            container.innerHTML = ""; // Очищаємо контейнер перед оновленням

            if (places.error) {
                container.innerHTML = `<p>${places.error}</p>`; // Повідомлення про
помилку
                return;
            }

            places.forEach(place => {
                const formattedAddress = formatAddress(place.address); // Форматуємо
адресу
                const card = document.createElement("div"); // Створюємо картку для
місця
                card.classList.add("card");

                const btnAddToPlan = document.createElement("button"); // Кнопка
"Додати в план"
                btnAddToPlan.classList.add("btn-secondary");
                btnAddToPlan.textContent = "+ в план";

                btnAddToPlan.addEventListener("click", async () => {
                    try {
                        const response = await fetch("/add-to-plan", {
                            method: "POST",
                            headers: {
                                "Content-Type": "application/json",
                            },
                            body: JSON.stringify({ photoUrl: place.photoUrl, name:
place.name, address: formattedAddress }),
                        });

                        const result = await response.json();
                        if (response.ok) {
                            alert("Додано в план!");
                        } else {
                            alert('Увійдіть в систему');
                        }
                    } catch (error) {
                        console.error("Помилка запиту:", error);
```

```
                        alert("Сталася помилка, спробуйте ще раз.");
                    }
                });

                card.innerHTML = `
                    <img src="${place.photoUrl}" alt="${place.name}"
onerror="this.src='/images/placeholder.png'">
                    <h3>${place.name}</h3>
                    <p>${formattedAddress}</p>
                    <p>Рейтинг: ${place.rating || "Немає рейтингу"}
(${place.totalRatings || 0} відгуків)</p>
                `;

                card.appendChild(btnAddToPlan); // Додаємо кнопку в картку
                container.appendChild(card); // Додаємо картку в контейнер
            });
        })
        .catch(error => console.error("Помилка завантаження:", error)); // Обробка
помилок
}
```

Щоб на сторінці не відображалася надлишкова інформація, наприклад, індекс або країна, використовується функція форматування адреси. Вона розбиває адресу на частини і виводить тільки перші два елементи.

```
function formatAddress(fullAddress) {
    const parts = fullAddress.split(", ");
    return parts.length >= 3 ? `${parts[0]}, ${parts[1]}` : fullAddress; // Повертаємо
перші дві частини адреси
}
```

JavaScript дозволяє створити зручний і динамічний інтерфейс, що дає змогу користувачам ефективно взаємодіяти з веб-додатками. У нашому прикладі ми реалізували пошук місць, управління вкладками та можливість додавання місць до плану подорожей. Залучення асинхронних запитів та обробки подій дозволяє створити безперервний і швидкий досвід для користувачів без необхідності перезавантаження сторінки.

ВИСНОВКИ

У результаті дослідження та розробки web-застосунку "Планувальник подорожей" було охарактеризовано сучасні тенденції у сфері туристичних подорожей, а також визначено ключові переваги онлайн-інструментів для їхнього планування. Аналіз статистичних даних показав, що цифрові технології відіграють дедалі важливішу роль у виборі маршрутів, пошуку житла та організації відпочинку загалом. Це підтверджує актуальність створення зручного та функціонального застосунку, який дозволить користувачам ефективно планувати подорожі.

У процесі дослідження було розглянуто основні методологічні підходи до розробки веб-застосунків, що дало змогу обрати найбільш ефективні технології та принципи для реалізації функціоналу. Зокрема, застосування JavaScript та його бібліотек стало ключовим аспектом у створенні інтерактивного інтерфейсу та забезпеченні зручності користувачів. Було вивчено й зіставлено можливості різних бібліотек і фреймворків, що дозволило обрати оптимальні рішення для роботи з пошуком місць та збереженням планів подорожей.

Детальний аналіз існуючих платформ продемонстрував, що ефективний планувальник подорожей повинен мати інтуїтивно зрозумілий інтерфейс, гнучкі інструменти і можливість інтеграції з зовнішніми сервісами, такими як Google places API. У розробці інтерфейсу було використано підхід, орієнтований на користувача, що дозволило забезпечити високу зручність та доступність застосунку.

Розроблений прототип web-застосунку охоплює основні функціональні можливості, необхідні для планування подорожей: пошук місць за категоріями, додавання обраних локацій до персонального маршруту, інтеграцію з картографічними сервісами та динамічну взаємодію з користувачем. Особливу увагу приділено інтерактивним елементам, що сприяють зручності навігації та покращенню UX/UI-дизайну.

Таким чином, у процесі роботи було розкрито сутність веб-розробки, досліджено технології, необхідні для створення сучасних web-застосунків, та реалізовано концепцію інструменту, що може бути корисним для широкого кола користувачів. Одержані результати можуть слугувати основою для подальшого вдосконалення застосунку та його адаптації до потреб сучасних мандрівників.

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

1. Березовський Р.Р., Залива В.В. Оптимізація CSS у розробці web-застосунку «Планувальник подорожі» // Матеріали VI Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». 24.04.2025, Київ, Державний університет комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2025, С. 211-213.
2. Березовський Р.Р., Залива В.В. Безпека веб-застосунку планувальника подорожей на платформі EXPRESS.JS // Матеріали VI Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». 24.04.2025, Київ, Державний університет комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2025, С. 447-450.

ПЕРЕЛІК ПОСИЛАНЬ

1. Торік зафіксовано 1,5 мільярда туристичних подорожей – UNWTO. URL : Торік зафіксовано 1,5 мільярда туристичних подорожей – UNWTO
2. Світовий туризм повернувся до показників до пандемії – ООН, 2025. URL : Туризм – світовий туризм оговтався після пандемії, у 2024 році міжнародні поїздки здійснили близько 1,4 млрд туристів » Слово і Діло
3. Мельник К. Туризм відновлює допандемічні показники. Понад 1,4 мільярда людей подорожували світом у 2024 році, 2025. URL : Туризм відновлює допандемічні показники. Понад 1,4 мільярда людей подорожували світом у 2024 році — The Village Україна
4. Корисні туристичні додатки для ваших подорожей Європою, 2024. URL : Visit World - Корисні туристичні додатки для ваших подорожей Європою
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 (IJITEE), 2020. №9(4). P. 2075-2084.
6. Susanto Anna Dara Andriana R. Perbandingan model waterfall dan prototyping untuk pengembangan sistem informasi. Majalah Ilmiah UNIKOM. 2016. URL : https://repository.unikom.ac.id/30459/1/4.miu-14-no-1-rani.pdf
7. Приходченко С. Д., Роина К. С., Поштак Р. В. Обгрунтування вибору мікросервісної архітектури в порівнянні з монолитною, 2018. URL : https://ir.nmu.org.ua/bitstream/handle/123456789/153837/70-73.pdf?sequence=1
8. Киселевич В. Мікросервісна архітектура: переваги та недоліки її практичного застосування. Information Technology: Computer Science, Software Engineering and Cyber Security, 2024. №2. С. 50-59.
9. Hamidli N. Introduction to UI/UX design: key concepts and principles. Preuzeto, 2023. № 28. P. 2024.
10. Чеботарьова І. Б., Черкашина Г. І. Основні тренди UI/UX дизайну 2024 року. Поліграфічні, мультимедійні та web-технології : матеріали Молодіжної

школи-семінару IX Міжнар. наук.-техн. конф., 14-28 травня 2024 р. Харків : ТОВ «Друкарня Мадрид», 2024. Т. 2. С. 40-47.

11. Баришев Ю. В., Каплун В. А. Метод автентифікації віддалених користувачів для мережевих сервісів. 2014. URL : http://ir.lib.vntu.edu.ua/bitstream/handle/123456789/2242/Itki_2014_2_4.pdf?sequence=1&isAllowed=y
12. Лубко Д. В., Мірошниченко М. Ю. (2024). Механізми безпеки інформації та їх виклики. Науковий вісник Таврійського державного агротехнологічного університету, 2024. №14(2). URL : https://oj.tsatu.edu.ua/index.php/visnik/article/download/835/789
13. Яковишен П. О., Тужанський С. Є.. Протоколи передавання даних в реальному часі в телемедичних системах (Doctoral dissertation, ВНТУ). 2024. URL : http://ir.lib.vntu.edu.ua/bitstream/handle/123456789/41525/20398.pdf?sequence=3&isAllowed=y
14. Циганенко, Д. П. Методи та технології захисту цифрових активів веб-додатків в умовах DDoS-атак. 2024. URL : https://essuir.sumdu.edu.ua/bitstream/123456789/96828/1/Tsyganenko_bak_rob.pdf
15. Gu X., Zhang H., Zhang D., Kim S. Deep API learning. In Proceedings of the 2016 24th ACM SIGSOFT international symposium on foundations of software engineering, 2016. URL :https://arxiv.org/pdf/1605.08535
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/vikoristannya-asinhronnogo-javascript-ta-ajax-zapitiv-u-veb-rozrobtsi
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. Rich Internet Applications w/HTML and Javascript, 2017. 15.URL : https://uol.de/f/2/informatik/ag/svs/download/reader/reader-seminar-ws2016.pdf#page=18
20. Mardan, A. (2014). Pro express. js: Master express. js: The node. js framework for your web development. Apress, 2014. URL : https://link.springer.com/book/10.1007/978-1-4842-0037-7
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?aid=2311236;label=uk-ua-booking-desktop-9iB*LpNwNENJAwTZ8*6VogS653685400986:pl:ta:p1:p2:ac:ap:neg:fi:tikwd-334108349:lp1030492:li:dec:dm;ws=&gad_source=1&gclid=CjwKCAjw7pO_BhAlEiwA4pMQvHzyjh9kuRmi7oSP4EpXMcSAfdFmRzgE7tey2G4IiqhTotkQSLU4XhoCw28QAvD_BwE
24. Офіційний сайт. TripAdvisor. URL : https://www.tripadvisor.com/
25. Lawson B., Sharp R. . Introducing html5. New Riders.2011. URL : https://ptgmedia.pearsoncmg.com/images/9780321687296/samplepages/0321687299.pdf
26. Mazinanian D., Tsantalis N., Mesbah A. Discovering refactoring opportunities in cascading style sheets. In Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2014. URL : https://people.ece.ubc.ca/amesbah/resources/papers/fse14.pdf
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. (2016). Дослідження особливостей застосування реляційних і нереляційних баз даних на прикладі SQL Server та

MongoDb. Наукові праці Національного університету харчових технологій, 2016. № 5. С. 15-24.

29. Bielak K., Borek B., Plechawska-Wójcik M. (2022). 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. URL : https://www.theseus.fi/bitstream/handle/10024/858208/Kukkonen_Sanna.pdf?sequence=2
31. Скряга Є. Є. Розробка сучасних методів побудови інтерактивного ресурсу картографічної моделі. URL : http://biblio.umsf.dp.ua/jspui/bitstream/123456789/7452/1/2025_Skriaha_121_Mag.pdf
32. Kaindl J. Implementation of the Fetch API for Graal. js (Doctoral dissertation, Johannes Kepler University Linz), 2023. URL : https://ssw.jku.at/Teaching/BachelorTheses/2023/Kaindl_Julian.pdf

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра інженерії програмного забезпечення

Тема бакалаврської роботи: РОЗРОБКА WEB-ЗАСТОСУНКУ "ПЛАНУВАЛЬНИК ПОДОРОЖЕЙ"

Виконав студент 4 курсу
групи ПД-43
Березовський Ростислав Ростиславович
доктор філософії (PhD), старший викладач кафедри ІПЗ Залива Віталій Вікторович

Київ - 2025

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

- **Мета**

Розробка web-застосунку «Планувальник подорожей», який забезпечує користувачам можливість створення, збереження та керування маршрутами подорожей з урахуванням місць призначення, дат, бюджету та особистих уподобань, із використанням сучасних засобів веб-розробки.

- **Об'єк та предмет**

Об'єктом дослідження є процес створення web-застосунків для планування подорожей.

- **Предмет**

Архітектурні рішення, методи та технології реалізації веб-застосунку для планування подорожей, включаючи створення маршруту, збереження даних користувача, інтерактивну взаємодію з інтерфейсом і забезпечення функцій реєстрації, авторизації та персоналізованого доступу.

ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

•Проаналізувати статистику подорожей та переваги онлайн-інструментів планування.

•Визначити методології розробки web-застосунків у сфері туризму.

•Дослідити можливості JavaScript для реалізації функціоналу планувальника.

•Провести огляд існуючих платформ для планування подорожей.

•Розробити прототип користувацького інтерфейсу.

•Сформувати функціональні вимоги до застосунку.

•Удосконалити дизайн інтерфейсу з урахуванням принципів UX/UI.

•Створити структуру та стилі вебсторінок відповідно до дизайну.

•Реалізувати основні функції застосунку за допомогою JavaScript.

АНАЛІЗ АНАЛОГІВ

	Airbnb	Booking.com	TripAdvisor	Планувальник подорожей
Наявність мобільного додатку	+	+	+	+
Бронювання житла	+	+	+	-
Пошук транспорту (авіа, поїзди)	-	+	+	+
Кастомізація маршруту	+	+	-	+
Персоналізовані рекомендації	+	+	+	+
Платні функції	-	+	-	-
Легкість доступу до підтримки	+	+	+	+

ВИМОГИ ДО ДОДАТКУ

ФУНКЦІОНАЛЬНІ

- ✓ Можливість додавати та редагувати інформацію про плани подорожей.
- ✓ Пошук та додавання туристичних місць (готелів, пам’ятників, ресторанів, музеїв тощо) з прив'язкою до конкретних локацій.
- ✓ Інтеграція Google API для пошуку туристичних місць та їх адрес
- ✓ Можливість збереження та синхронізації даних про подорожі між різними пристроями через базу даних.
- ✓ Реєстрація та авторизація.
- ✓ Зберігання паролів в зашифрованому вигляді.

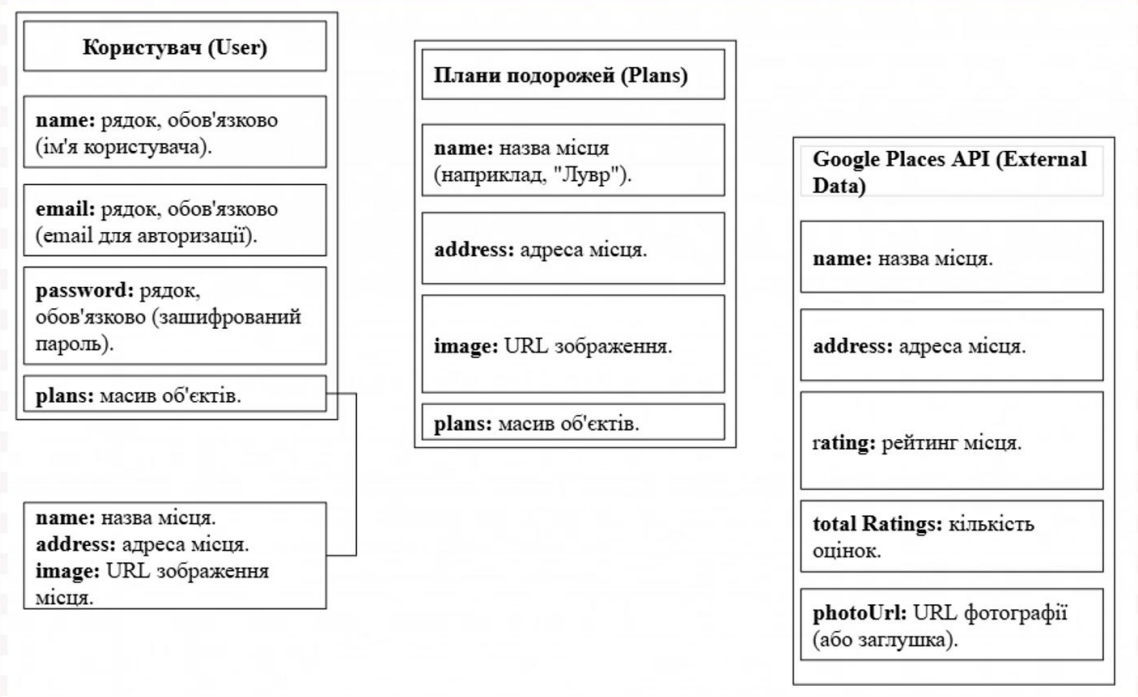
НЕФУНКЦІОНАЛЬНІ

- ✗ Збереження паролів у зашифрованому вигляді (безпека).
- ✗ Підтримка синхронізації між різними пристроями (доступність та масштабованість).

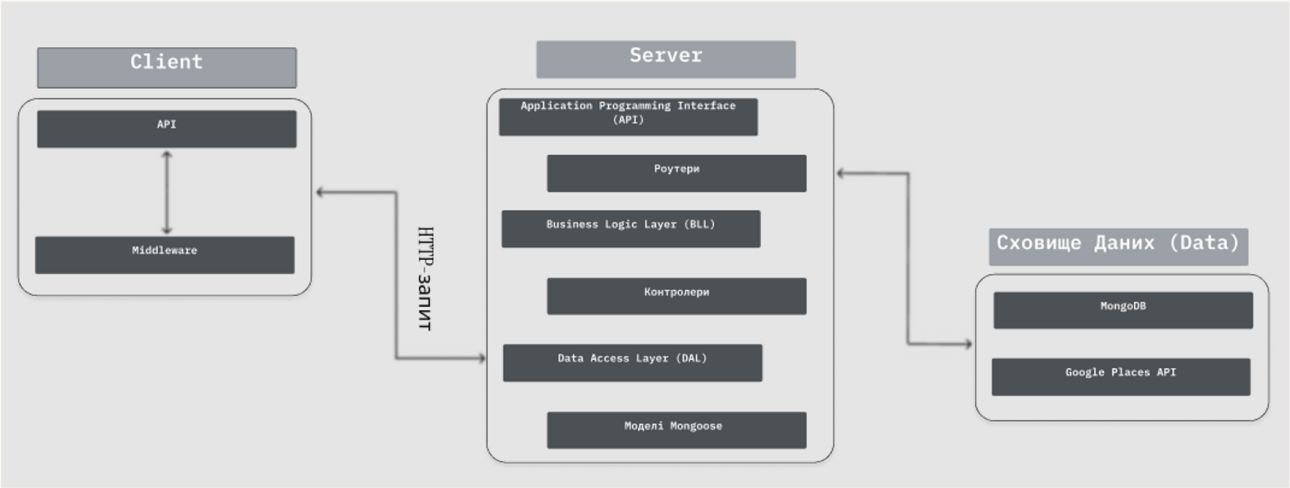
ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

Node.js	Express.js	MongoDB	Mongoose
Handlebars	Google Places API	HTML5	CSS3

СТРУКТУРА БАЗИ ДАНИХ

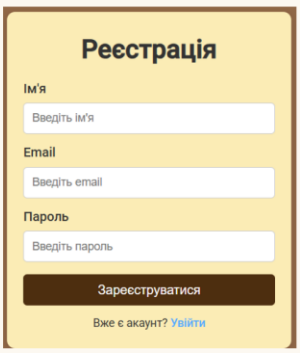


АРХІТЕКТУРА ПРОГРАМИ

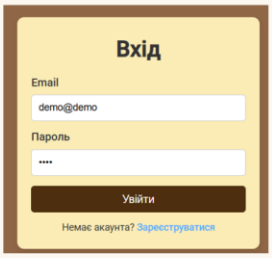


МАПА ВЕБ-ЗАСТОСУНКУ

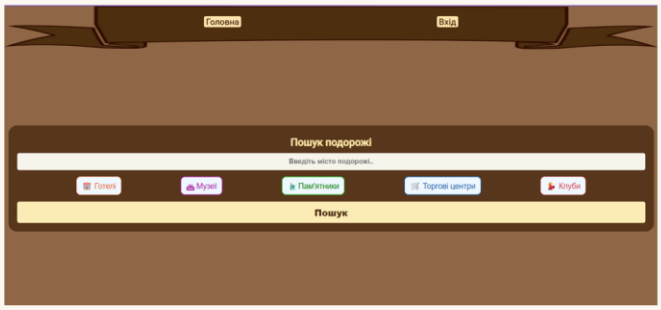
ЕКРАННІ ФОРМИ



Форма реєстрації

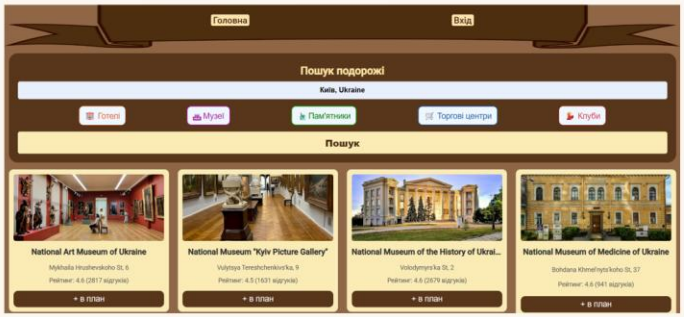


Форма входу



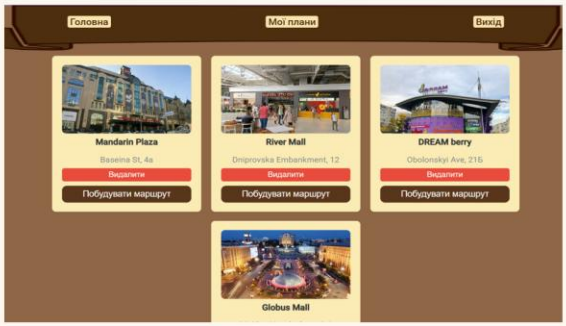
Головна сторінка

ЕКРАННІ ФОРМИ



Результати пошуку

Картки результатів пошуку



Сторінка “мої плани”

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Березовський Р.Р., Залива В.В. Оптимізація CSS у розробці web-застосунку «Планувальник подорожі» // Матеріали VI Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». 24.04.2025, Київ, Державний університет комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2025, С. 211-213.

2. Березовський Р.Р., Залива В.В. Безпека веб-застосунку планувальника подорожей на платформі EXPRESS.JS // Матеріали VI Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». 24.04.2025, Київ, Державний університет комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2025, С. 447-450.

ВИСНОВКИ

У ході розробки web-застосунку «Планувальник подорожей» було проаналізовано сучасні тенденції в туризмі та переваги цифрових інструментів для планування мандрівок. Статистичні дані підтвердили зростання ролі технологій у виборі маршрутів, пошуку житла та організації подорожей, що підкреслює актуальність такого застосунку.

Було вивчено методологічні підходи до веб-розробки, проаналізовано бібліотеки JavaScript, обрано оптимальні інструменти для створення зручного, інтерактивного інтерфейсу. Акцент зроблено на інтеграції з Google Places API, інтуїтивному дизайні та зручності користувача.

Прототип реалізує основні функції: пошук місць, додавання до маршруту, взаємодію з картами та збереження планів. Результати дослідження можуть слугувати базою для подальшого розвитку та адаптації застосунку до потреб мандрівників.

ДОДАТОК Б. 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
```