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

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка Web-застосунку для відображення та розробки туристичних маршрутів на інтерактивній карті мовою JavaScript та з використанням бібліотеки React»

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

_____ Євгеній ПУСТОВІТ
(підпис)

Виконав: здобувач вищої освіти групи ПД-44
Євгеній ПУСТОВІТ

Керівник: Богдан ХУДІК
доктор філософії (PhD)

Рецензент: _____

Київ 2024

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
« ____ » _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Пустовіту Євгенію Юрійовичу

1. Тема кваліфікаційної роботи: «Розробка Web-застосунку для відображення та розробки туристичних маршрутів на інтерактивній карті мовою JavaScript та з використанням бібліотеки React»
керівник кваліфікаційної роботи доктор філософії (PhD), ст. викладач кафедри ІПЗ Богдан ХУДІК,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «27» лютого 2024 р. № 36.

2. Строк подання кваліфікаційної роботи «28» травня 2024 р.

3. Вихідні дані до кваліфікаційної роботи: аналіз існуючих рішень, визначення вимог до застосунку, технічні характеристики, розробка та інтеграція, тестування, документація та презентація.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Визначити функціональні та нефункціональні вимоги до Web-застосунку.
2. Спроектувати архітектуру та структуру Web-застосунку.
3. Розробити серверну та клієнтську частини.
4. Провести тестування.

5. Перелік графічного матеріалу: *презентація*

1. Аналіз аналогів.
2. Вимоги до програмного забезпечення.
3. Програмні засоби реалізації.
4. Діаграма варіантів використання.
5. Діаграма пакетів.
6. Діаграма активності.
7. Екранні форми.
8. Демонстрація використання застосунку.
9. Апробація результатів дослідження.

6. Дата видачі завдання «28» лютого 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	28.02.-06.03.2024	
2	Аналіз та дослідження існуючих аналогів	07.03-13.03.2024	
3	Визначити функціональні та нефункціональні вимоги Web-застосунку	14.03-25.03.2024	
4	Спроектувати архітектуру та структуру Web-застосунку	26.03-05.04.2024	
5	Розробити серверну та клієнтську частини	06.04-17.04.2024	
6	Провести тестування	18.04-28.04.2024	
7	Оформлення роботи: вступ, висновки, реферат	29.04-05.05.2024	
8	Розробка демонстраційних матеріалів	06.05-12.05.2024	
9	Попередній захист роботи	13.05-31.05.2024	

Здобувач вищої освіти _____ Євгеній ПУСТОВІТ
(підпис)

Керівник
кваліфікаційної роботи _____ Богдан ХУДІК
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 66 стор., 3 табл., 8 рис., 26 джерел.

Мета роботи – спростити процес планування туристичних маршрутів за допомогою Web-застосунку розробленого з використанням JavaScript та бібліотеки React.

Об'єкт дослідження – процес планування туристичних маршрутів.

Предмет дослідження – Web-застосунок для планування туристичних маршрутів.

Короткий зміст роботи: Проаналізовано існуючі програмні рішення для відображення та розробки туристичних маршрутів, виявлено їх переваги та недоліки. Визначено функціональні та нефункціональні вимоги до Web-застосунку. Спроектовано архітектуру та структуру Web-застосунку, враховуючи обрані технології та інструменти. Розроблено серверну та клієнтську частини Web-застосунку з використанням мови JavaScript та бібліотеки React. Інтегровано картографічний сервіс для відображення інтерактивної карти та реалізовано функціонал для створення, редагування та збереження туристичних маршрутів. Проведено тестування Web-застосунку.

Сферою використання застосунку є відображення та розробка туристичних маршрутів на інтерактивній карті.

КЛЮЧОВІ СЛОВА: WEB ЗАСТОСУНОК, ІНТЕРАКТИВНА КАРТА, ТУРИСТИЧНІ МАРШРУТИ, JAVASCRIPT, REACT, КЛЮЧОВІ СЛОВА.

ЗМІСТ

ВСТУП

Актуальність теми. В сучасному світі туризм є однією з найбільш динамічно розвинутих галузей економіки. Все більше людей прагнуть подорожувати, відкривати для себе нові місця та здобувати незабутні враження. Однак, планування туристичних маршрутів може бути складним та часозатратним процесом, особливо для недосвідчених мандрівників. Тому розробка зручних та ефективних інструментів для відображення та створення туристичних маршрутів на інтерактивній карті є актуальною задачею, яка може значно спростити процес планування подорожей та зробити туризм більш доступним для широкого кола людей.

Web-технології надають широкі можливості для створення інтерактивних та зручних у використанні застосунків. Мова програмування JavaScript та бібліотека React є популярними та потужними інструментами для розробки web-застосунків. Вони дозволяють створювати динамічні та інтерактивні інтерфейси користувача, що робить взаємодію з застосунком більш приємною та ефективною. Крім того, використання web-технологій дозволяє зробити застосунок доступним для користувачів з різних пристроїв та операційних систем без необхідності встановлення додаткового програмного забезпечення.

Отже, розробка web-застосунку для відображення та розробки туристичних маршрутів на інтерактивній карті з використанням мови JavaScript та бібліотеки React є актуальною та перспективною задачею, яка може значно покращити досвід планування подорожей для користувачів та сприяти розвитку туристичної галузі.

Мета і завдання дослідження. Метою даної роботи є спрощення процесу планування туристичних маршрутів та підвищення зручності для користувачів шляхом розробки web-застосунку для відображення та розробки туристичних маршрутів на інтерактивній карті з використанням мови JavaScript та бібліотеки React.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

1. Проаналізувати існуючі програмні рішення для відображення та розробки туристичних маршрутів, виявити їх переваги та недоліки.
2. Визначити функціональні та нефункціональні вимоги до web-застосунку на основі потреб цільової аудиторії.
3. Спроектувати архітектуру та структуру web-застосунку, враховуючи обрані технології та інструменти.
4. Розробити серверну та клієнтську частини web-застосунку з використанням мови JavaScript та бібліотеки React.
5. Інтегрувати картографічний сервіс для відображення інтерактивної карти та реалізувати функціонал для створення, редагування та збереження туристичних маршрутів.
6. Провести тестування web-застосунку та оцінити його ефективність та зручність використання.

Об'єкт і предмет дослідження. Об'єктом дослідження є процес планування туристичних маршрутів за допомогою web-застосунків. Предметом дослідження є web-застосунок для відображення та розробки туристичних маршрутів на інтерактивній карті, розроблений з використанням мови JavaScript та бібліотеки React.

Практична новизна та реальність роботи. Практична новизна даної роботи полягає у створенні web-застосунку, який поєднує в собі функціональні можливості відображення туристичних маршрутів на інтерактивній карті та зручні інструменти для їх розробки та редагування. Запропоноване рішення орієнтоване на потреби сучасних мандрівників та враховує вимоги до зручності використання та інтерактивності застосунків.

Реальність роботи визначається наявністю конкретної цільової аудиторії - людей, які займаються плануванням туристичних подорожей та потребують зручних інструментів для створення та відображення маршрутів на карті. Розроблений web-застосунок може бути використаний як самостійний продукт або інтегрований в існуючі туристичні платформи та сервіси.

Таким чином, дана бакалаврська робота присвячена актуальній темі розробки web-застосунку для відображення та розробки туристичних маршрутів на інтерактивній карті з використанням сучасних технологій. Результати роботи мають практичну цінність та можуть бути використані для покращення досвіду планування подорожей для користувачів та розвитку туристичної галузі.

1. АНАЛІЗ ПРОБЛЕМИ ТА ІСНУЮЧИХ ЗАСОБІВ ДЛЯ ЇЇ ВИРІШЕННЯ

1.1 Опис задачі та предметної галузі

Туризм - це одна з найбільш динамічних та перспективних галузей світової економіки. Щороку мільйони людей подорожують по всьому світу, відкриваючи для себе нові місця, культури та пам'ятки. Для багатьох країн туризм є важливим джерелом доходу та зайнятості населення. Згідно з даними Всесвітньої туристичної організації (UNWTO), у 2019 році кількість міжнародних туристичних прибуттів досягла 1,5 мільярда, що на 4% більше, ніж у 2018 році[1].

Однак, незважаючи на стрімкий розвиток туристичної галузі, планування подорожей та створення туристичних маршрутів часто залишається складним та трудомістким процесом. Туристам доводиться витрачати багато часу на пошук інформації про визначні місця, транспортні засоби, готелі та інші туристичні послуги. Крім того, самостійне планування маршруту може бути ускладнене відсутністю знань про місцевість, мовні бар'єри та інші фактори.

Тому актуальною є задача розробки web-застосунку, який би дозволив спростити та автоматизувати процес створення туристичних маршрутів. Такий застосунок повинен надавати користувачам зручний та інтуїтивно зрозумілий інтерфейс для планування подорожей, а також забезпечувати доступ до актуальної та достовірної інформації про туристичні об'єкти та послуги.

Основними функціями web-застосунку для створення туристичних маршрутів мають бути:

1. Відображення інтерактивної карти з можливістю навігації та масштабування.
2. Пошук та відображення туристичних об'єктів (пам'яток, музеїв, готелів, ресторанів тощо) на карті.
3. Можливість створення власних туристичних маршрутів шляхом вибору та з'єднання туристичних об'єктів на карті.

4. Автоматичне прокладання оптимального маршруту між обраними об'єктами з урахуванням відстані та часу подорожі.
5. Збереження створених маршрутів у персональному кабінеті користувача для подальшого використання.
6. Можливість експорту маршрутів у популярні формати (наприклад, GPX) для використання в навігаційних пристроях або мобільних додатках.
7. Надання детальної інформації про кожен туристичний об'єкт (опис, фотографії, відгуки тощо).
8. Інтеграція з сервісами бронювання житла та транспорту для зручності планування подорожі.

Розробка такого web-застосунку вимагає використання сучасних технологій та інструментів, таких як мова програмування JavaScript, фреймворк React для створення користувацького інтерфейсу, картографічні сервіси (наприклад, Google Maps або OpenStreetMap) для відображення інтерактивної карти, а також сервер на основі Node.js для обробки запитів та зберігання даних.

Предметна галузь, у якій буде використовуватися web-застосунок для створення туристичних маршрутів, охоплює різноманітні аспекти туристичної діяльності. Зокрема, застосунок може бути корисним для:

- Індивідуальних туристів, які планують самостійні подорожі та потребують зручного інструменту для створення маршрутів.
- Туристичних агентств, які можуть використовувати застосунок для розробки та презентації маршрутів своїм клієнтам.
- Місцевих туристичних інформаційних центрів, які можуть надавати інформацію про туристичні об'єкти та маршрути через web-застосунок.
- Організаторів групових турів, які можуть використовувати застосунок для планування та координації маршрутів для груп туристів.

Для кращого розуміння масштабів туристичної галузі та потенційного попиту на web-застосунок для створення туристичних маршрутів, розглянемо деякі статистичні дані:

Таблиця 1.1

Статистичні дані

Показник	**Значення**
Кількість міжнародних туристичних прибуттів у 2019 році	1,5 млрд
Частка туризму у світовому ВВП	10,3%
Кількість робочих місць у туристичній галузі	330 млн
Частка онлайн-бронювань у туристичній галузі	63%
Очікуване зростання онлайн-бронювань до 2023 року	72%

Ці дані свідчать про значний потенціал розвитку онлайн-сервісів у туристичній галузі, зокрема web-застосунків для створення туристичних маршрутів. Розробка такого застосунку дозволить не лише спростити процес планування подорожей для туристів, але й відкрити нові можливості для туристичних компаній та організацій.

Отже, задача розробки web-застосунку для створення туристичних маршрутів є актуальною та перспективною. Такий застосунок дозволить задовольнити потреби туристів у зручному інструменті для планування подорожей, а також надасть нові можливості для розвитку туристичної галузі в цілому. Використання сучасних технологій та методів розробки програмного забезпечення дозволить створити високоякісний та ефективний застосунок, який відповідатиме вимогам та очікуванням користувачів.

1.2 Аналіз існуючих програмних продуктів для відображення та розробки туристичних маршрутів

На ринку програмного забезпечення існує низка рішень, які дозволяють відображати та розробляти туристичні маршрути. Ці продукти відрізняються функціональними можливостями, цільовою аудиторією та бізнес-моделями.

Розглянемо детальніше деякі з найбільш популярних та ефективних програмних продуктів у цій галузі.

Рис. 1.1 Логотип Google Maps

1. Google Maps - це один з найвідоміших та найпопулярніших картографічних сервісів у світі. Окрім стандартних функцій, таких як перегляд карт та прокладання маршрутів, Google Maps надає користувачам можливість створювати власні карти з позначками та маршрутами. Користувачі можуть додавати точки інтересу, визначні місця та інші туристичні об'єкти на карту, а також ділитися створеними картами з іншими користувачами. Google Maps доступний як у веб-версії, так і у вигляді мобільних додатків для iOS та Android.

   Переваги:

- Широке покриття та висока деталізація карт.
- Зручний та інтуїтивно зрозумілий інтерфейс.
- Інтеграція з іншими сервісами Google, такими як Google Places та Google Street View.
- Можливість створення власних карт та маршрутів.

  Недоліки:

- Обмежені можливості для персоналізації та брендування карт.
- Відсутність спеціалізованих функцій для туристичної галузі.

Рис. 1.2 Логотип TripAdvisor

2. TripAdvisor - це популярна платформа для планування подорожей, яка дозволяє користувачам знаходити та бронювати готелі, ресторани, розваги та туристичні послуги по всьому світу. Окрім цього, TripAdvisor надає можливість створювати та ділитися туристичними маршрутами. Користувачі можуть додавати місця, які вони хочуть відвідати, до свого маршруту, а також переглядати маршрути, створені іншими мандрівниками. TripAdvisor також пропонує рекомендації щодо визначних місць та активностей на основі вподобань користувача та відгуків інших мандрівників.

   Переваги:

- Велика база даних туристичних об'єктів та послуг.
- Можливість бронювання готелів, ресторанів та розваг безпосередньо через платформу.
- Відгуки та рейтинги від реальних мандрівників.
- Функція створення та обміну туристичними маршрутами.

  Недоліки:

- Обмежені можливості для персоналізації маршрутів.
- Відсутність детальних карт з навігацією.

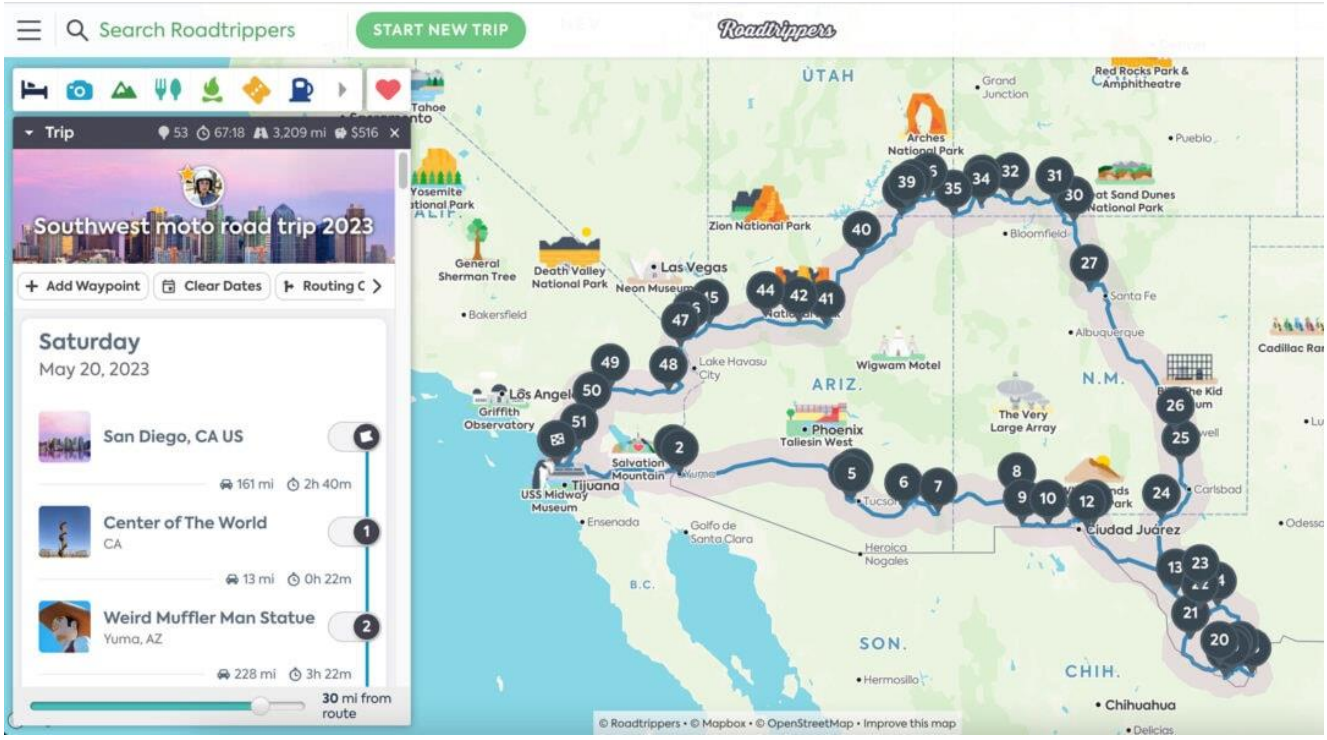


Рис. 1.3 Вигляд програми Roadtrippers

3. Roadtrippers - це спеціалізований сервіс для планування автомобільних подорожей. Він дозволяє користувачам створювати маршрути з урахуванням визначних місць, цікавих локацій та зупинок на шляху. Користувачі можуть вказати початкову та кінцеву точки маршруту, а Roadtrippers автоматично запропонує цікаві місця для відвідування поблизу. Сервіс також надає інформацію про відстані, час у дорозі та витрати на паливо для кожного маршруту.

   Переваги:

- Спеціалізація на автомобільних подорожах.
- Автоматичні рекомендації цікавих місць по маршруту.
- Деталізація маршрутів з урахуванням відстаней, часу та витрат.
- Можливість бронювання житла та розваг безпосередньо через платформу.

  Недоліки:

- Обмеженість використання для інших видів транспорту, окрім автомобіля.
- Недоступність сервісу в деяких країнах.

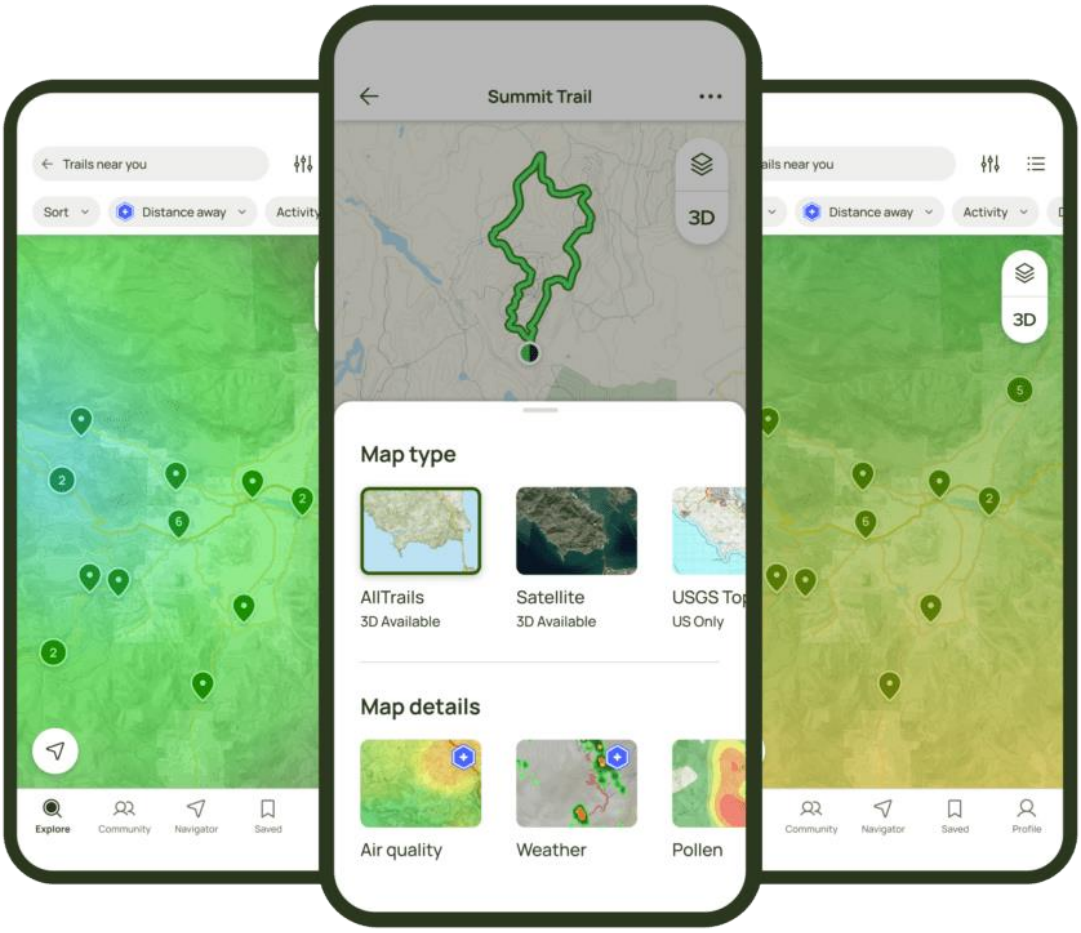


Рис. 1.4 Вигляд програми AllTrails

4. AllTrails - це платформа, орієнтована на любителів активного відпочинку та пішохідного туризму. Вона пропонує детальні карти пішохідних маршрутів по всьому світу, включаючи інформацію про складність, відстань, перепади висот та визначні місця на шляху. Користувачі можуть переглядати маршрути, створені іншими мандрівниками, а також ділитися власними треками, записаними за допомогою GPS. AllTrails доступний як у веб-версії, так і у вигляді мобільних додатків для iOS та Android.

   Переваги:

- Спеціалізація на пішохідному туризмі та активному відпочинку.
- Детальні карти та інформація про маршрути.
- Можливість запису власних треків та обміну ними з іншими користувачами.
- Велика спільнота користувачів та активне оновлення бази даних маршрутів.

Недоліки:

- Обмеженість використання для інших видів туризму, окрім пішохідного.
- Необхідність оплати підписки для доступу до деяких функцій.

Рис. 1.5 Логотип Komoot

5. Komoot - це платформа для планування велосипедних та пішохідних подорожей. Вона пропонує детальні карти маршрутів з урахуванням типу місцевості, покриття та складності. Користувачі можуть створювати власні маршрути, вказуючи початкову та кінцеву точки, а також проміжні точки інтересу. Komoot також надає рекомендації щодо визначних місць та цікавих локацій на маршруті. Платформа доступна у веб-версії та у вигляді мобільних додатків для iOS та Android.

   Переваги:

- Спеціалізація на велосипедному та пішохідному туризмі.
- Детальні карти з урахуванням типу місцевості та складності.
- Можливість створення власних маршрутів з проміжними точками.
- Рекомендації щодо визначних місць та цікавих локацій.

   Недоліки:

- Обмеженість використання для інших видів туризму.
- Необхідність оплати для доступу до деяких функцій та регіонів.

Отже, існує низка програмних продуктів, які дозволяють відображати та розробляти туристичні маршрути. Кожен з них має свої переваги та недоліки, а також орієнтований на певну цільову аудиторію та вид туризму. При виборі програмного рішення для розробки туристичних маршрутів важливо враховувати специфіку проекту, вимоги користувачів та бюджет. Поєднання функціональних можливостей, зручності використання та якості даних є ключовими факторами успіху такого програмного продукту.

Таблиця 1.2

Порівняння функціональних можливостей програмних продуктів для відображення та розробки туристичних маршрутів

Характеристики застосунків	**Google Maps**	**TripAdvisor**	**Roadtrippers**	**AllTrails**	**Komoot**
Веб-версія	+	+	+	+	+
Розробка маршрутів	+	+	+	+	+
Обмін маршрутами	+	+	-	+	+
Збереження маршрутів	-	-	-	+	+
Відображення маршрутів на інтерактивній карті	+	+	+	+	+
Спеціалізація	Універсальний	Універсальний	Автомобільні подорожі	Пішохідний туризм	Велосипедний та пішохідний туризм

Як видно з таблиці, кожен з розглянутих програмних продуктів має свої сильні та слабкі сторони. Google Maps та TripAdvisor є більш універсальними рішеннями, які підходять для різних видів туризму та надають широкий спектр функцій. У той же час, Roadtrippers, AllTrails та Komoot є більш спеціалізованими сервісами, орієнтованими на певні види туризму та активного відпочинку.

При виборі програмного рішення для розробки туристичних маршрутів важливо враховувати специфіку проекту та потреби цільової аудиторії. Наприклад, якщо проект орієнтований на автомобільні подорожі, то Roadtrippers може бути найбільш підходящим варіантом. Якщо ж проект спрямований на пішохідний туризм, то AllTrails або Komoot можуть бути більш доречними.

Також важливо враховувати такі фактори, як зручність використання, якість даних та можливості інтеграції з іншими сервісами. Наприклад, Google Maps та TripAdvisor мають велику базу даних туристичних об'єктів та послуг, а також надають можливості для бронювання та перегляду відгуків. У той же час, AllTrails та Komoot дозволяють записувати власні треки та обмінюватися ними з іншими користувачами, що може бути важливим для деяких проектів.

Отже, вибір програмного рішення для розробки туристичних маршрутів залежить від специфіки проекту, потреб користувачів та бюджету. Розуміння функціональних можливостей та обмежень кожного з розглянутих продуктів дозволить прийняти зважене рішення та створити ефективний та зручний сервіс для планування подорожей.

1.3 Огляд технологій та інструментів для розробки web-застосунків

Для успішної розробки web-застосунку для відображення та розробки туристичних маршрутів необхідно обрати відповідні технології та інструменти. Вони мають забезпечувати високу продуктивність, масштабованість, зручність використання та можливість інтеграції з іншими сервісами. Розглянемо детальніше основні технології та інструменти, які можуть бути використані для розробки такого web-застосунку.

1. Мова програмування JavaScript JavaScript є однією з найпопулярніших мов програмування для розробки web-застосунків. Вона дозволяє створювати інтерактивні користувацькі інтерфейси, обробляти події та взаємодіяти з серверною частиною застосунку. JavaScript має багату екосистему бібліотек та фреймворків, які спрощують розробку та підвищують продуктивність розробників.
2. Фреймворк React React - це популярний JavaScript-фреймворк для розробки користувацьких інтерфейсів. Він базується на компонентній архітектурі та дозволяє створювати багаторазові та незалежні компоненти інтерфейсу. React має високу продуктивність завдяки віртуальному DOM та ефективним алгоритмам оновлення сторінки. Він також має велику спільноту розробників та багату екосистему бібліотек та інструментів.

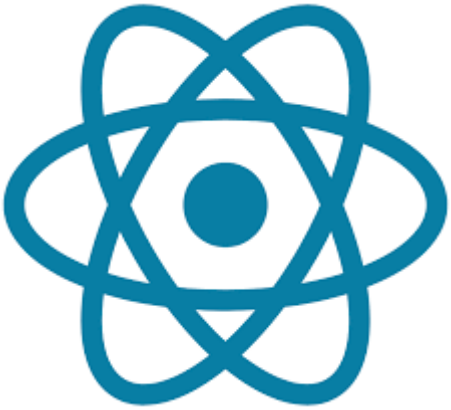


Рис. 1.6 Логотип React

3. Бібліотека Leaflet Leaflet - це легковісна JavaScript-бібліотека для роботи з інтерактивними картами. Вона надає зручний API для відображення карт, додавання маркерів, ліній та полігонів, а також обробки подій користувача. Leaflet підтримує різні картографічні сервіси, такі як OpenStreetMap, Mapbox та інші. Бібліотека має хорошу документацію та активну спільноту розробників.

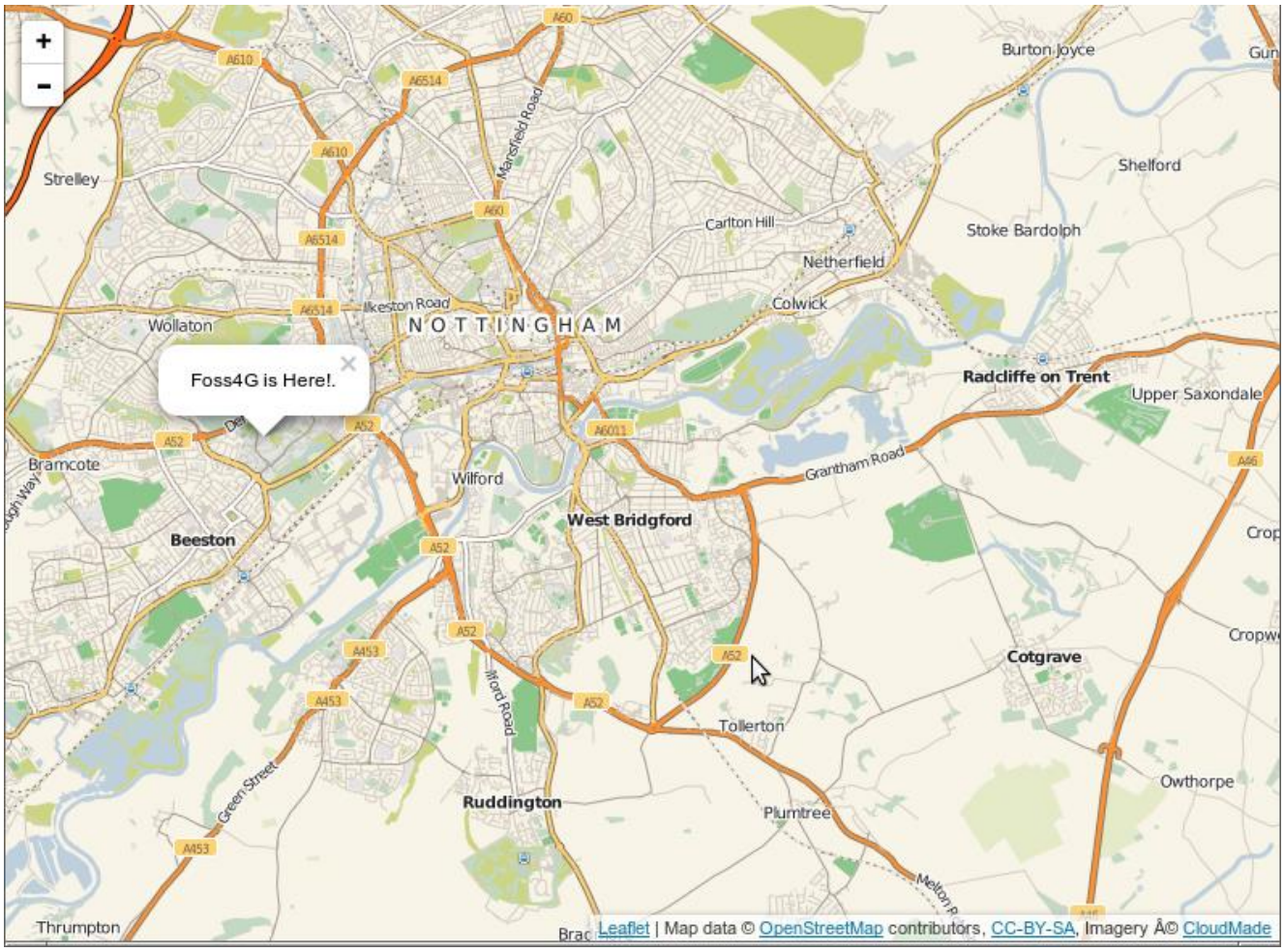


Рис. 1.7 Приклад карти, створеної за допомогою Leaflet

4. Сервер Node.js Node.js - це середовище виконання JavaScript на стороні сервера. Воно дозволяє писати серверний код на мові JavaScript та надає можливості для створення високопродуктивних та масштабованих web-застосунків. Node.js має неблокуючу модель введення-виведення та підтримує асинхронне програмування, що дозволяє обробляти велику кількість запитів одночасно. Він також має велику кількість бібліотек та модулів, які спрощують розробку серверної частини застосунку.
5. База даних MongoDB MongoDB - це документо-орієнтована база даних, яка добре підходить для зберігання неструктурованих та напівструктурованих даних. Вона має гнучку схему даних та дозволяє зберігати дані у форматі JSON. MongoDB має високу продуктивність та масштабованість, а також підтримує реплікацію та шардинг для забезпечення надійності та доступності даних. Для роботи з MongoDB з Node.js часто використовують

бібліотеку Mongoose, яка надає зручний інтерфейс для визначення схем та моделей даних.

6. Картографічні сервіси Для відображення інтерактивних карт та побудови маршрутів у web-застосунку можуть бути використані різні картографічні сервіси. Серед популярних варіантів - Google Maps API, Mapbox API та OpenStreetMap. Ці сервіси надають доступ до різноманітних картографічних даних, таких як карти, супутникові знімки, інформація про дороги та об'єкти інфраструктури. Вони також мають зручні API для інтеграції з web-застосунками та надають можливості для кастомізації та стилізації карт.
7. Інструменти для збирання та розгортання Для ефективної розробки та розгортання web-застосунку необхідно використовувати відповідні інструменти. Серед них - системи контролю версій (наприклад, Git), інструменти для збирання та пакування коду (наприклад, Webpack), системи автоматизації розгортання (наприклад, Docker) та хмарні платформи (наприклад, AWS або Google Cloud). Ці інструменти дозволяють спростити процес розробки, тестування та розгортання застосунку, а також забезпечують його надійність та масштабованість.

Отже, для розробки web-застосунку для відображення та розробки туристичних маршрутів можуть бути використані такі технології та інструменти, як мова програмування JavaScript, фреймворк React, бібліотека Leaflet, сервер Node.js, база даних MongoDB та різноманітні картографічні сервіси. Крім того, важливо використовувати відповідні інструменти для збирання, тестування та розгортання застосунку, щоб забезпечити його якість та надійність.

Вибір конкретних технологій та інструментів залежить від специфіки проекту, вимог до продуктивності та масштабованості, а також від досвіду та переваг команди розробників. Важливо враховувати сумісність обраних технологій між собою, а також їх здатність вирішувати поставлені завдання та забезпечувати високу якість кінцевого продукту.

1.4 Постановка задач бакалаврської роботи

Метою бакалаврської роботи є розробка web-застосунку для відображення та розробки туристичних маршрутів на інтерактивній карті з використанням мови JavaScript та бібліотеки React. Застосунок повинен надавати користувачам зручний інтерфейс для перегляду існуючих маршрутів, створення нових маршрутів та управління ними.

Для досягнення поставленої мети необхідно вирішити наступні задачі:

1. Провести аналіз предметної області та існуючих рішень для відображення та розробки туристичних маршрутів.
    - Дослідити актуальність та затребуваність web-застосунків для планування туристичних подорожей.
    - Проаналізувати існуючі програмні продукти та сервіси, які дозволяють створювати та відображати туристичні маршрути.
    - Визначити основні функціональні та нефункціональні вимоги до розроблюваного web-застосунку.
2. Спроектувати архітектуру та структуру web-застосунку.
    - Розробити компонентну структуру застосунку з використанням бібліотеки React.
    - Спроектувати інтерфейс користувача з урахуванням зручності використання та естетичного вигляду.
    - Визначити необхідні сторінки та компоненти застосунку, такі як головна сторінка з картою, сторінка створення маршруту та сторінка списку маршрутів.
3. Реалізувати функціонал відображення інтерактивної карти.
    - Інтегрувати бібліотеку Leaflet для відображення інтерактивної карти на сторінці.
    - Налаштувати відображення карти з використанням тайлових шарів OpenStreetMap.

- Забезпечити можливість навігації по карті, зміни масштабу та перегляду різних регіонів.

4. Реалізувати функціонал створення нових туристичних маршрутів.
    - Розробити форму для введення назви та опису маршруту.
    - Інтегрувати інструменти для малювання маршруту на інтерактивній карті з використанням бібліотеки Leaflet та плагіну Leaflet.draw.
    - Реалізувати збереження створеного маршруту на сервері за допомогою відправки запиту на відповідний API-ендпоінт.
5. Реалізувати функціонал перегляду списку існуючих маршрутів.
    - Створити сторінку зі списком збережених маршрутів.
    - Забезпечити завантаження списку маршрутів з сервера за допомогою відправки запиту на відповідний API-ендпоінт.
    - Відображати деталі кожного маршруту, включаючи назву, опис та інтерактивну карту з намальованим маршрутом.
6. Реалізувати додатковий функціонал для управління маршрутами.
    - Додати можливість імпорту маршруту до Google Maps для детального перегляду та навігації.
    - Реалізувати функціонал видалення маршруту зі списку зі збереженням змін на сервері.
7. Забезпечити адаптивність та кросбраузерність web-застосунку.
    - Використати адаптивну верстку та стилі для коректного відображення застосунку на різних пристроях та розмірах екрану.
    - Протестувати застосунок у різних веб-браузерах та забезпечити його сумісність.
8. Оптимізувати продуктивність та швидкодію web-застосунку.
    - Застосувати техніки оптимізації завантаження ресурсів, такі як мінімізація та стиснення файлів.
    - Використати ефективні підходи до рендерингу компонентів та управління станом застосунку.

- Забезпечити швидке завантаження та відгук застосунку при роботі з великими обсягами даних.

9. Провести тестування web-застосунку та виправити виявлені недоліки.
    - Розробити та виконати набір тестових сценаріїв для перевірки функціональності застосунку.
    - Провести ручне тестування інтерфейсу користувача та взаємодії з елементами управління.
    - Виконати автоматизоване тестування ключових компонентів та функцій застосунку.
    - Виправити виявлені помилки та недоліки за результатами тестування.
10. Оформити документацію до web-застосунку.
    - Підготувати технічну документацію з описом архітектури, використаних технологій та компонентів застосунку.
    - Створити керівництво користувача з інструкціями щодо використання основних функцій застосунку.
    - Оформити звіт про виконану роботу з висвітленням основних етапів розробки та отриманих результатів.

Виконання поставлених задач дозволить створити функціональний та зручний web-застосунок для відображення та розробки туристичних маршрутів на інтерактивній карті. Застосунок надасть користувачам можливість переглядати існуючі маршрути, створювати нові маршрути з використанням інтерактивної карти та управляти ними.

Результати бакалаврської роботи можуть бути використані як основа для подальшого розвитку та вдосконалення застосунку, а також для впровадження в реальні проекти з планування туристичних подорожей.

Для успішного виконання поставлених задач необхідно застосувати знання та навички з веб-розробки, використання бібліотеки React, інтеграції з картографічними сервісами та реалізації клієнт-серверної взаємодії. Також важливо приділити увагу питанням користувацького досвіду, продуктивності та тестування застосунку.

2. ПРОЕКТУВАННЯ WEB-ЗАСТОСУНКУ ДЛЯ ВІДОБРАЖЕННЯ ТА РОЗРОБКИ ТУРИСТИЧНИХ МАРШРУТІВ

2.1 Проектування архітектури та структури web-застосунку

При проектуванні архітектури та структури web-застосунку для відображення та розробки туристичних маршрутів було враховано вимоги до функціональності, масштабованості та зручності використання. Для досягнення цих цілей було обрано клієнт-серверну архітектуру з використанням сучасних веб-технологій.

Архітектура web-застосунку складається з двох основних компонентів: клієнтської частини (фронтенд) та серверної частини (бекенд). Клієнтська частина відповідає за взаємодію з користувачем, відображення інтерфейсу та обробку дій користувача. Серверна частина займається обробкою запитів від клієнта, виконанням бізнес-логіки та взаємодією з базою даних.

Для реалізації клієнтської частини було обрано бібліотеку React, яка дозволяє створювати інтерактивні та динамічні користувацькі інтерфейси. React використовує компонентний підхід, що забезпечує модульність та повторне використання коду. Компоненти в React представляють собою незалежні та багаторазові частини інтерфейсу, які можуть бути легко скомбіновані для створення складних веб-сторінок.

Для організації навігації між сторінками web-застосунку використовується бібліотека React Router. Вона дозволяє створювати маршрутизацію на стороні клієнта, забезпечуючи плавну навігацію без перезавантаження сторінки. React Router також підтримує передачу параметрів між сторінками, що дозволяє динамічно генерувати вміст на основі URL-адреси.

Серверна частина web-застосунку реалізована з використанням Node.js та фреймворку Express. Node.js є середовищем виконання JavaScript на стороні сервера, що дозволяє створювати високопродуктивні та масштабовані веб-додатки.

Express, в свою чергу, надає набір функцій та утиліт для спрощення розробки серверної частини, таких як маршрутизація, обробка запитів та відповідей, middleware тощо.

Для збереження та управління даними туристичних маршрутів використовується JSON Server. JSON Server - це фейковий REST API, який дозволяє швидко створювати прототипи серверної частини без необхідності реалізації повноцінної бази даних. Він зберігає дані у форматі JSON і надає endpoints для виконання CRUD (створення, читання, оновлення, видалення) операцій над даними.

Взаємодія між клієнтською та серверною частинами відбувається за допомогою HTTP запитів. Клієнтська частина надсилає запити до серверної частини, використовуючи відповідні HTTP методи (GET, POST, PUT, DELETE) та передаючи необхідні дані. Серверна частина обробляє ці запити, виконує необхідні операції та повертає відповідь у форматі JSON. Для спрощення взаємодії з сервером на клієнтській частині використовується бібліотека Axios, яка надає зручний інтерфейс для виконання HTTP запитів.

Для відображення інтерактивної карти та роботи з геопросторовими даними використовується бібліотека Leaflet. Leaflet - це популярна та легковажна бібліотека для створення інтерактивних карт на веб-сторінках. Вона надає широкий набір функцій для відображення карти, додавання маркерів, ліній, полігонів та інших геопросторових об'єктів. Leaflet також підтримує різні провайдери тайлів карти, що дозволяє гнучко налаштовувати зовнішній вигляд та джерело даних для карти.

Структура проекту організована за принципом розділення відповідальності та модульності. Клієнтська частина розділена на компоненти, які відповідають за різні частини інтерфейсу користувача, такі як головна сторінка, сторінка створення маршруту, сторінка списку маршрутів тощо. Кожен компонент містить свою власну логіку та стилі, що робить їх незалежними та легко підтримуваними.

Серверна частина також організована з використанням модульного підходу. Різні частини функціональності розділені на окремі файли та папки, такі як

маршрутизація, контролери, моделі даних тощо. Це дозволяє легко розширювати та модифікувати серверну частину в майбутньому.

Для забезпечення зручності розробки та розгортання web-застосунку використовуються інструменти автоматизації та збирання проекту. Webpack використовується для збирання та оптимізації клієнтської частини, включаючи транспіляцію коду з використанням Babel, мінімізацію та об'єднання файлів. Це дозволяє створювати оптимізовану та сумісну з різними браузерами версію клієнтської частини.

Для керування залежностями проекту використовується npm (Node Package Manager). Він дозволяє легко встановлювати та оновлювати зовнішні бібліотеки та пакети, необхідні для роботи web-застосунку.

Таким чином, спроектована архітектура та структура web-застосунку забезпечує чіткий розподіл відповідальності між клієнтською та серверною частинами, використовує сучасні веб-технології та підходи до розробки, такі як React, Node.js, Express та Leaflet, а також забезпечує модульність, масштабованість та зручність розробки та розгортання.

2.2 Моделювання вимог до web-застосунку

Моделювання вимог є важливим етапом у процесі розробки web-застосунку для відображення та розробки туристичних маршрутів. Цей етап передбачає детальний аналіз та документування функціональних та нефункціональних вимог, які повинен задовольняти web-застосунок.

Функціональні вимоги описують основні функції та можливості, які повинен надавати web-застосунок користувачам. Для web-застосунку туристичних маршрутів основними функціональними вимогами є:

1. Відображення інтерактивної карти: web-застосунок повинен надавати можливість відображення інтерактивної карти, на якій користувачі зможуть переглядати та досліджувати туристичні маршрути. Карта повинна підтримувати базові функції навігації, такі як масштабування, переміщення

та зміна режиму перегляду (наприклад, супутникова карта або векторна карта).

2. Створення та редагування туристичних маршрутів: користувачі повинні мати можливість створювати нові туристичні маршрути, вказуючи назву, опис та точки маршруту на інтерактивній карті. Вони також повинні мати можливість редагувати існуючі маршрути, змінюючи їх назву, опис або додаючи/видаляючи точки маршруту.
3. Збереження та завантаження маршрутів: web-застосунок повинен надавати функціональність для збереження створених туристичних маршрутів на сервері, щоб користувачі могли отримати доступ до них пізніше. Крім того, користувачі повинні мати можливість завантажувати збережені маршрути для перегляду та редагування.
4. Відображення списку маршрутів: web-застосунок повинен надавати сторінку, на якій відображається список усіх збережених туристичних маршрутів. Список повинен містити основну інформацію про кожен маршрут, таку як назва, опис та мініатюрне зображення карти з маршрутом.
5. Деталі маршруту: при виборі конкретного маршруту зі списку, web-застосунок повинен відображати детальну інформацію про нього, включаючи назву, опис, повне зображення карти з маршрутом та список точок маршруту.
6. Пошук та фільтрація маршрутів: web-застосунок повинен надавати можливість пошуку маршрутів за назвою або ключовими словами. Крім того, користувачі повинні мати можливість фільтрувати маршрути за певними критеріями, такими як довжина маршруту, складність або тип місцевості.
7. Експорт маршрутів: web-застосунок повинен дозволяти користувачам експортувати маршрути у популярних форматах, таких як GPX або KML, для використання в інших додатках або пристроях.

Нефункціональні вимоги описують характеристики та обмеження web-застосунку, які не стосуються безпосередньо функціональності, але впливають на

його якість, продуктивність та користувацький досвід. Для web-застосунку туристичних маршрутів основними нефункціональними вимогами є:

1. Продуктивність: web-застосунок повинен забезпечувати швидке завантаження сторінок та плавну роботу інтерактивної карти. Він повинен ефективно обробляти запити користувачів та мінімізувати затримки у відповіді.
2. Масштабованість: web-застосунок повинен бути розроблений з урахуванням можливості масштабування для обробки зростаючого навантаження та кількості користувачів. Він повинен мати можливість обробляти велику кількість одночасних запитів без значного погіршення продуктивності.
3. Безпека: web-застосунок повинен забезпечувати захист даних користувачів та запобігати несанкціонованому доступу. Він повинен використовувати надійні механізми аутентифікації та авторизації, шифрування даних при передачі та зберіганні, а також захищати від поширених веб-вразливостей, таких як міжсайтовий скриптинг (XSS) та ін'єкції SQL.
4. Зручність використання: інтерфейс користувача web-застосунку повинен бути інтуїтивно зрозумілим, простим у навігації та використанні. Він повинен забезпечувати чіткі інструкції та зворотній зв'язок для користувачів під час взаємодії з функціями створення та редагування маршрутів.
5. Адаптивність: web-застосунок повинен бути адаптивним та коректно відображатися на різних пристроях та розмірах екрану, включаючи настільні комп'ютери, планшети та смартфони. Він повинен забезпечувати оптимізоване відображення та функціональність незалежно від пристрою, що використовується.
6. Доступність: web-застосунок повинен дотримуватися стандартів доступності, щоб забезпечити можливість використання людьми з обмеженими можливостями. Він повинен підтримувати функції, такі як клавіатурна навігація, альтернативний текст для зображень та відповідність стандартам WCAG (Web Content Accessibility Guidelines).

7. Локалізація: web-застосунок повинен підтримувати можливість локалізації для різних мов та регіонів. Він повинен дозволяти легке додавання та управління перекладами інтерфейсу користувача, щоб забезпечити доступність для користувачів з різних країн та мовних груп.
8. Сумісність з браузерами: web-застосунок повинен бути сумісним з популярними веб-браузерами, такими як Google Chrome, Mozilla Firefox, Safari та Microsoft Edge. Він повинен забезпечувати коректне відображення та функціональність незалежно від використовуваного браузера.
9. Документація та підтримка: web-застосунок повинен супроводжуватися вичерпною документацією, яка описує його функціональність, архітектуру та процес розгортання. Також повинна бути надана підтримка користувачів у вигляді довідкових матеріалів, часто запитуваних питань (FAQ) та каналів зв'язку для вирішення проблем.

Для моделювання та документування вимог до web-застосунку можуть використовуватися різні підходи та інструменти. Одним з поширених методів є використання мови моделювання UML (Unified Modeling Language) для створення діаграм прецедентів (use case diagrams), які описують взаємодію користувачів з системою та основні сценарії використання.

Крім того, для детального опису вимог можуть використовуватися специфікації вимог у текстовому форматі, такі як документи з описом функціональних та нефункціональних вимог. Ці документи повинні містити чіткі та однозначні формулювання вимог, критерії прийнятності та пріоритети.

Під час моделювання вимог важливо залучати зацікавлені сторони, включаючи замовника, користувачів та членів команди розробки. Це дозволяє отримати зворотній зв'язок, уточнити вимоги та забезпечити їх відповідність очікуванням та потребам користувачів.

Моделювання вимог є ітеративним процесом, який може потребувати кількох ітерацій для уточнення та затвердження остаточного набору вимог. Важливо підтримувати актуальність документації вимог протягом усього життєвого циклу розробки web-застосунку та вносити зміни у разі потреби.

Таким чином, моделювання вимог до web-застосунку для відображення та розробки туристичних маршрутів включає визначення функціональних та нефункціональних вимог, використання відповідних методів та інструментів для їх документування, залучення зацікавлених сторін та підтримку актуальності вимог протягом усього процесу розробки. Це забезпечує чітке розуміння очікувань та вимог до web-застосунку та слугує основою для його успішної реалізації.

2.3 Проектування бази даних для збереження туристичних маршрутів

У розробленому web-застосунку для відображення та розробки туристичних маршрутів використовується JSON Server для збереження даних про маршрути. JSON Server - це фейковий REST API, який дозволяє швидко створювати прототипи серверної частини без необхідності реалізації повноцінної бази даних.

Хоча JSON Server не є традиційною базою даних, він надає можливість зберігати дані у форматі JSON і виконувати операції CRUD (створення, читання, оновлення, видалення) над цими даними. Це робить його зручним інструментом для розробки та тестування web-застосунків без необхідності налаштування окремої бази даних.

Для збереження туристичних маршрутів у JSON Server використовується файл db.json, який містить дані у форматі JSON. Цей файл виступає в ролі "бази даних" для зберігання маршрутів.

Структура даних для збереження туристичних маршрутів у файлі db.json має наступний вигляд:

```
json
{
  "routes": [
    {
      "id": 1,
      "name": "Маршрут 1",
      "description": "Опис маршруту 1",
      "path": [
        [50.4501, 30.5234],
        [50.4511, 30.5244],
```

```
      [50.4521, 30.5254]
     ],
     "center": [50.4501, 30.5234],
     "zoom": 13
    },
    {
     "id": 2,
     "name": "Маршрут 2",
     "description": "Опис маршруту 2",
     "path": [
      [50.4601, 30.5334],
      [50.4611, 30.5344],
      [50.4621, 30.5354]
     ],
     "center": [50.4601, 30.5334],
     "zoom": 14
    }
  ]
}
```

У цьому прикладі файл db.json містить масив об'єктів routes, кожен з яких представляє окремий туристичний маршрут. Кожен об'єкт маршруту має наступні поля:

- id: унікальний ідентифікатор маршруту (автоматично генерується JSON Server).
- name: назва маршруту.
- description: опис маршруту.
- path: масив точок маршруту, де кожна точка представлена масивом з двох елементів - широти та довготи.
- center: центральна точка маршруту, представлена масивом з двох елементів - широти та довготи.
- zoom: рівень масштабування карти для відображення маршруту.

Для взаємодії з JSON Server з web-застосунку використовуються HTTP-запити. Наприклад, для отримання списку всіх маршрутів виконується GET-запит до URL-адреси /routes, а для створення нового маршруту - POST-запит до тієї ж URL-адреси з даними маршруту в тілі запиту.

JSON Server автоматично генерує унікальні ідентифікатори (id) для нових маршрутів при їх створенні, тому немає необхідності вказувати id вручну при додаванні нового маршруту.

Така структура даних дозволяє зберігати та отримувати інформацію про туристичні маршрути, включаючи їх назву, опис, точки маршруту, центральну точку та рівень масштабування карти.

Хоча використання JSON Server є зручним для швидкої розробки та тестування, для реального розгортання web-застосунку рекомендується використовувати повноцінну базу даних, таку як MongoDB, PostgreSQL або MySQL. Це забезпечить більшу надійність, масштабованість та можливості для зберігання та управління даними туристичних маршрутів.

При переході на повноцінну базу даних потрібно буде спроектувати відповідну схему бази даних, яка враховує структуру та зв'язки між сутностями, такими як маршрути, точки маршруту та інші пов'язані дані. Також потрібно буде адаптувати серверну частину web-застосунку для роботи з обраною базою даних та забезпечити відповідні механізми доступу та управління даними.

Таким чином, проектування бази даних для збереження туристичних маршрутів у розробленому web-застосунку базується на використанні JSON Server та зберіганні даних у файлі db.json. Це дозволяє швидко створювати прототип серверної частини та зберігати дані про маршрути без необхідності налаштування окремої бази даних. Однак для реального розгортання web-застосунку рекомендується перейти на використання повноцінної бази даних з відповідною схемою та механізмами доступу.

2.4 Проектування інтерфейсу користувача web-застосунку

Проектування інтерфейсу користувача (UI) є важливим етапом у розробці web-застосунку для відображення та розробки туристичних маршрутів. Інтерфейс користувача повинен бути інтуїтивно зрозумілим, зручним у використанні та

естетично привабливим. Він повинен забезпечувати користувачам легкий доступ до основних функцій web-застосунку та сприяти ефективній взаємодії з ним.

При проектуванні інтерфейсу користувача web-застосунку для туристичних маршрутів були враховані наступні ключові аспекти:

1. Структура та навігація:
    - Головна сторінка: Головна сторінка web-застосунку повинна містити чіткий та привабливий заголовок, який відображає назву та призначення web-застосунку. На головній сторінці також можуть бути розміщені посилання на основні розділи, такі як "Карта", "Створити маршрут" та "Список маршрутів".
    - Навігаційне меню: Web-застосунок повинен мати інтуїтивно зрозуміле навігаційне меню, яке дозволяє користувачам легко переміщатися між різними сторінками та функціями. Навігаційне меню може бути розташоване у верхній частині сторінки або у вигляді бокової панелі.
    - Сторінка створення маршруту: Сторінка створення маршруту повинна містити форму для введення назви та опису маршруту, а також інтерактивну карту для додавання точок маршруту. Користувачі повинні мати можливість додавати, редагувати та видаляти точки маршруту за допомогою зручних елементів керування на карті.
    - Сторінка списку маршрутів: Сторінка списку маршрутів повинна відображати список усіх збережених туристичних маршрутів. Кожен елемент списку повинен містити основну інформацію про маршрут, таку як назва, опис та мініатюрне зображення карти з маршрутом. При натисканні на елемент списку користувач повинен перенаправлятися на сторінку з детальною інформацією про обраний маршрут.
2. Дизайн та візуальні елементи:
    - Кольорова схема: Web-застосунок повинен мати привабливу та гармонійну кольорову схему, яка відповідає тематиці туристичних маршрутів. Кольори повинні бути приємними для очей та забезпечувати чіткий контраст між різними елементами інтерфейсу.

- Типографія: Вибір шрифтів повинен бути зрозумілим та легко читабельним. Розмір шрифту повинен бути достатньо великим для зручності читання. Заголовки та основний текст можуть використовувати різні шрифти для створення візуальної ієрархії.
- Іконки та зображення: Використання іконок та зображень може покращити візуальну привабливість та зрозумілість інтерфейсу. Іконки можуть використовуватися для позначення різних дій або категорій, а зображення - для ілюстрації туристичних маршрутів або визначних місць.
- Адаптивний дизайн: Інтерфейс користувача повинен бути адаптивним та коректно відображатися на різних пристроях та розмірах екрану. Він повинен забезпечувати оптимальне відображення та функціональність як на настільних комп'ютерах, так і на мобільних пристроях.

3. Інтерактивність та зворотний зв'язок:
   - Інтерактивна карта: Інтерактивна карта є центральним елементом web-застосунку для туристичних маршрутів. Вона повинна надавати користувачам можливість переміщуватися по карті, змінювати масштаб та взаємодіяти з маркерами або лініями маршруту. При наведенні курсора на маркер або лінію може відображатися додаткова інформація про відповідну точку маршруту.
   - Форми та введення даних: Форми для введення даних, такі як форма створення маршруту, повинні бути чіткими та зрозумілими. Поля введення повинні мати відповідні підписи та підказки, щоб користувачі розуміли, яку інформацію потрібно ввести. Валідація форм повинна забезпечувати перевірку введених даних та надавати зрозумілі повідомлення про помилки.
   - Зворотний зв'язок та сповіщення: Web-застосунок повинен надавати користувачам зрозумілий зворотний зв'язок та сповіщення про успішні дії або помилки. Наприклад, після успішного збереження маршруту може відображатися повідомлення про успішне збереження. У разі

виникнення помилок, таких як неправильне введення даних, повинні відображатися відповідні повідомлення з чіткими інструкціями щодо виправлення помилок.

4. Доступність та зручність використання:
    - Розмітка та структура коду: HTML-розмітка повинна бути семантичною та структурованою, щоб забезпечити доступність web-застосунку для користувачів з обмеженими можливостями. Використання відповідних тегів та атрибутів допомагає пристроям для читання екрану та іншим допоміжним технологіям коректно інтерпретувати вміст сторінки.
    - Клавіатурна навігація: Web-застосунок повинен підтримувати навігацію за допомогою клавіатури, щоб користувачі могли взаємодіяти з елементами інтерфейсу без використання миші. Це може бути реалізовано за допомогою атрибутів tabindex та відповідних подій клавіатури.
    - Контрастність та кольори: Кольорові схеми повинні забезпечувати достатній контраст між текстом та фоном, щоб полегшити читання для користувачів з порушеннями зору. Слід уникати використання кольорів як єдиного засобу передачі інформації, оскільки це може бути недоступним для користувачів з дальтонізмом.
    - Підказки та інструкції: Web-застосунок повинен надавати підказки та інструкції для користувачів, особливо для складних або нетипових функцій. Це можуть бути спливаючі підказки, пояснювальний текст або візуальні вказівки, які допомагають користувачам зрозуміти, як взаємодіяти з певними елементами інтерфейсу.

При проектуванні інтерфейсу користувача web-застосунку для туристичних маршрутів були створені макети та прототипи, які відображають структуру, розташування та зовнішній вигляд основних сторінок та компонентів. Ці макети були створені з використанням інструментів для прототипування, таких як Figma, Sketch або Adobe XD.

Макети та прототипи дозволяють візуалізувати та протестувати дизайн інтерфейсу перед початком розробки. Вони допомагають узгодити бачення дизайну між членами команди та зацікавленими сторонами, а також отримати зворотний зв'язок та внести необхідні зміни на ранніх етапах проекту.

Після затвердження дизайну інтерфейсу користувача, макети та прототипи використовуються як основа для верстки та стилізації web-застосунку з використанням HTML, CSS та JavaScript. Розробники використовують ці макети як візуальний посібник для створення відповідних компонентів та сторінок.

Під час розробки інтерфейсу користувача також важливо враховувати принципи UI/UX дизайну, такі як простота, послідовність, інтуїтивність та естетична привабливість. Дизайн повинен бути орієнтований на користувача та забезпечувати позитивний досвід взаємодії з web-застосунком.

Тестування юзабіліті є важливим етапом після розробки інтерфейсу користувача. Воно дозволяє отримати зворотний зв'язок від реальних користувачів щодо зручності використання, зрозумілості та ефективності інтерфейсу. На основі результатів тестування можуть бути внесені необхідні зміни та вдосконалення для покращення загального користувацького досвіду.

Таким чином, проектування інтерфейсу користувача web-застосунку для відображення та розробки туристичних маршрутів включає створення привабливого, інтуїтивно зрозумілого та зручного у використанні дизайну. Важливо враховувати структуру, навігацію, візуальні елементи, інтерактивність та доступність при розробці інтерфейсу. Використання макетів, прототипів та тестування юзабіліті допомагає створити інтерфейс, який забезпечує позитивний досвід взаємодії з web-застосунком для користувачів.

3. РЕАЛІЗАЦІЯ WEB-ЗАСТОСУНКУ ДЛЯ ВІДОБРАЖЕННЯ ТА РОЗРОБКИ ТУРИСТИЧНИХ МАРШРУТІВ

3.1 Реалізація серверної частини web-застосунку

Серверна частина web-застосунку відповідає за обробку запитів від клієнтської частини, виконання бізнес-логіки та взаємодію з базою даних. У розробленому web-застосунку для відображення та розробки туристичних маршрутів серверна частина реалізована з використанням Node.js та фреймворку Express.

Node.js - це середовище виконання JavaScript, яке дозволяє запускати JavaScript-код на сервері. Воно надає можливість створювати масштабовані та високопродуктивні веб-додатки. Express - це мінімалістичний та гнучкий фреймворк для створення веб-додатків на основі Node.js. Він надає набір функцій та утиліт для спрощення розробки серверної частини.

Для початку роботи з серверною частиною було створено новий проект Node.js та ініціалізовано його за допомогою менеджера пакетів npm (Node Package Manager). Було створено файл package.json, який містить метадані проекту та список залежностей.

Основні залежності, які використовуються в серверній частині:

- Express: фреймворк для створення веб-додатків на Node.js.
- Body-parser: middleware для обробки тіла запиту (request body) у форматі JSON.
- Cors: middleware для увімкнення підтримки Cross-Origin Resource Sharing (CORS) на сервері.

Структура серверної частини організована з використанням модульного підходу. Кожна частина функціональності розділена на окремі файли та папки для кращої читабельності та підтримки коду.

Головний файл серверної частини - server.js. Цей файл містить налаштування та запуск сервера Express. У ньому відбувається підключення необхідних модулів, налаштування middleware, визначення маршрутів та запуск сервера на вказаному порту.

Ось основні кроки реалізації серверної частини:

1. Імпорт залежностей:

   ```
   const express = require('express');
   const bodyParser = require('body-parser');
   const cors = require('cors');
   ```

2. Створення екземпляра додатку Express:

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

3. Налаштування middleware:

   ```
   app.use(bodyParser.json());
   app.use(cors());
   ```

   - bodyParser.json() дозволяє обробляти тіло запиту у форматі JSON.
   - cors() вмикає підтримку CORS, дозволяючи крос-доменні запити.

4. Визначення маршрутів:

   ```
   app.get('/api/routes', (req, res) => {
     // Обробка GET-запиту для отримання списку маршрутів
   });

   app.post('/api/routes', (req, res) => {
     // Обробка POST-запиту для збереження нового маршруту
   });

   app.delete('/api/routes/:id', (req, res) => {
     // Обробка DELETE-запиту для видалення маршруту за його ідентифікатором
   });
   ```

   - Маршрути визначають шляхи URL та відповідні обробники запитів.

- Використовуються методи HTTP, такі як GET, POST, DELETE, для різних операцій з маршрутами.

5. Обробка запитів та взаємодія з базою даних:
    - У кожному обробнику маршруту відбувається обробка запиту та виконання відповідних дій.
    - Для взаємодії з базою даних у розробленому застосунку використовується JSON Server.
    - JSON Server дозволяє зберігати дані у форматі JSON у файлі db.json та надає REST API для виконання операцій з цими даними.
6. Запуск сервера:

```
const port = process.env.PORT || 5000;
app.listen(port, () => {
  console.log(`Сервер запущено на порту ${port}`);
});
```

    - Сервер запускається на вказаному порту (за замовчуванням - 5000).
    - Після успішного запуску сервера виводиться повідомлення в консоль.

Для взаємодії з JSON Server в обробниках маршрутів використовується модуль fs (File System) для читання та запису даних у файл db.json. Наприклад, при отриманні списку маршрутів виконується читання файлу db.json, парсинг даних з формату JSON та відправлення відповіді клієнту. При збереженні нового маршруту дані з запиту додаються до масиву маршрутів та записуються назад у файл db.json.

Для обробки помилок та винятків використовуються блоки try-catch. У разі виникнення помилки, наприклад, при читанні або запису файлу, відправляється відповідний код стану HTTP та повідомлення про помилку клієнту.

Для забезпечення безпеки та валідації вхідних даних можуть бути додані додаткові перевірки та валідація в обробниках маршрутів. Наприклад, перевірка наявності обов'язкових полів у тілі запиту, валідація формату даних тощо.

Для розширення функціональності серверної частини можуть бути додані додаткові маршрути та обробники для виконання інших операцій, таких як

редагування маршруту, фільтрація маршрутів за певними критеріями, автентифікація та авторизація користувачів тощо.

Для забезпечення масштабованості та продуктивності серверної частини можуть бути застосовані різні оптимізації та архітектурні рішення. Наприклад, використання асинхронних операцій вводу-виводу, кешування даних, розподілення навантаження між кількома серверами тощо.

Тестування серверної частини є важливим етапом розробки. Для тестування можуть бути використані фреймворки та бібліотеки, такі як Jest або Mocha. Тести дозволяють перевірити коректність роботи серверної частини, обробку різних сценаріїв використання та обробку помилок.

Розгортання серверної частини відбувається на відповідному середовищі, такому як хмарна платформа (наприклад, AWS, Google Cloud, Heroku) або виділений сервер. Процес розгортання включає налаштування середовища, встановлення необхідних залежностей та запуск сервера.

Для моніторингу та управління серверною частиною можуть бути використані інструменти моніторингу, такі як PM2 або forever. Вони дозволяють стежити за станом сервера, автоматично перезапускати його у разі збоїв та забезпечувати безперебійну роботу.

Таким чином, реалізація серверної частини web-застосунку для відображення та розробки туристичних маршрутів з використанням Node.js та Express надає надійний та масштабований бекенд. Він забезпечує обробку запитів від клієнтської частини, виконання бізнес-логіки та взаємодію з базою даних. Модульна структура та використання middleware дозволяють організувати код у чіткій та зрозумілій формі. Також важливо враховувати аспекти безпеки, продуктивності та тестування при розробці серверної частини.

3.2 Реалізація клієнтської частини web-застосунку з використанням React

Клієнтська частина web-застосунку відповідає за взаємодію з користувачем, відображення інтерфейсу та обробку дій користувача. У розробленому web-застосунку для відображення та розробки туристичних маршрутів клієнтська частина реалізована з використанням бібліотеки React.

React - це популярна бібліотека JavaScript для побудови користувацьких інтерфейсів. Вона надає можливість створювати компонентно-орієнтовані та перевикористовувані елементи інтерфейсу. React використовує віртуальний DOM (Document Object Model) для ефективного оновлення та перерендерингу компонентів, що забезпечує високу продуктивність та плавну роботу застосунку.

Для початку роботи з клієнтською частиною було створено новий проект React за допомогою інструменту Create React App. Цей інструмент автоматично налаштовує початкову структуру проекту, конфігурацію збирання та залежності.

Основні залежності, які використовуються в клієнтській частині:

- React: бібліотека для побудови користувацьких інтерфейсів.
- React Router: бібліотека для маршрутизації на стороні клієнта.
- Axios: бібліотека для виконання HTTP-запитів до сервера.
- Leaflet: бібліотека для роботи з інтерактивними картами.

Структура клієнтської частини організована з використанням компонентного підходу. Кожен компонент представляє собою незалежну та багаторазову частину інтерфейсу користувача. Компоненти розділені на окремі файли для кращої організації та підтримки коду.

Ось основні кроки реалізації клієнтської частини з використанням React:

1. Створення компонентів:
    - Кожен компонент реалізований як JavaScript-клас або функція, яка повертає розмітку React (JSX).
    - Компоненти розділені на розумні (smart) та презентаційні (dumb) компоненти.

- Розумні компоненти містять логіку та стан, а презентаційні компоненти відповідають за відображення даних.

2. Маршрутизація:
    - Для навігації між сторінками використовується бібліотека React Router.
    - Визначено маршрути для кожної сторінки застосунку, такі як головна сторінка, сторінка створення маршруту та сторінка списку маршрутів.
    - Використовуються компоненти <Route> та <Link> для визначення маршрутів та навігаційних посилань.
3. Взаємодія з сервером:
    - Для виконання HTTP-запитів до серверної частини використовується бібліотека Axios.
    - Запити виконуються для отримання списку маршрутів, збереження нового маршруту, видалення маршруту тощо.
    - Дані, отримані з сервера, зберігаються в стані компонентів та відображаються в інтерфейсі користувача.
4. Робота з картами:
    - Для відображення інтерактивних карт використовується бібліотека Leaflet.
    - Компонент <MapContainer> з бібліотеки React Leaflet використовується для створення контейнера карти.
    - На карті відображаються маркери, лінії та інші елементи, що представляють туристичні маршрути.
    - Для взаємодії з картою, такою як додавання маркерів чи ліній, використовуються відповідні компоненти та методи з бібліотеки Leaflet.
5. Стан та пропси:
    - Стан компонента використовується для зберігання динамічних даних, таких як список маршрутів, вибраний маршрут тощо.

- Пропси використовуються для передачі даних між компонентами, наприклад, передача маршруту з батьківського компонента до дочірнього.
- Оновлення стану компонента призводить до автоматичного оновлення відображення компонента.

6. Обробка подій:
    - Події, такі як натискання кнопки, зміна значення поля введення тощо, обробляються за допомогою обробників подій.
    - Обробники подій визначаються як методи компонента або функції зворотного виклику.
    - У обробниках подій виконуються відповідні дії, такі як оновлення стану, виклик серверних запитів тощо.
7. Умовний рендеринг:
    - Умовний рендеринг дозволяє відображати різні компоненти або вміст залежно від певних умов.
    - Наприклад, якщо список маршрутів порожній, відображається повідомлення "Немає доступних маршрутів", інакше відображається список маршрутів.
8. Стилізація:
    - Для стилізації компонентів використовуються CSS або CSS-in-JS рішення, такі як styled-components або CSS модулі.
    - Стилі можуть бути визначені безпосередньо в компонентах або винесені в окремі файли стилів.
    - Для забезпечення адаптивності та підтримки різних пристроїв використовуються медіа-запити та адаптивні одиниці вимірювання.
9. Взаємодія з користувачем:
    - Користувацький інтерфейс розроблено з урахуванням принципів UI/UX дизайну.
    - Забезпечено інтуїтивно зрозумілу навігацію, чіткі позначення та підказки для користувача.

- Реалізовано зворотний зв'язок, такий як повідомлення про успішне збереження маршруту або обробку помилок.

10. Тестування та налагодження:
    - Для тестування компонентів використовуються фреймворки тестування, такі як Jest і Enzyme.
    - Написано модульні тести для перевірки коректності рендерингу компонентів, обробки подій та зміни стану.
    - Використовуються інструменти розробника браузера для налагодження та відстеження помилок у коді.
11. Оптимізація продуктивності:
    - Застосовано техніки оптимізації, такі як розділення коду (code splitting), ледаче завантаження компонентів (lazy loading) та використання мемоізації для уникнення непотрібних перерендерингів.
    - Оптимізовано розмір та кількість HTTP-запитів шляхом мінімізації коду та використання CDN для завантаження зовнішніх бібліотек.
12. Розгортання та збірка:
    - Для збирання та оптимізації коду використовується інструмент збірки, такий як Webpack або Create React App.
    - Налаштовано конфігурацію збірки для оптимізації розміру та швидкості завантаження застосунку.
    - Застосунок розгорнуто на веб-сервері або хмарній платформі, такій як Netlify або Vercel.

Ось структура компонентів у клієнтській частині web-застосунку:

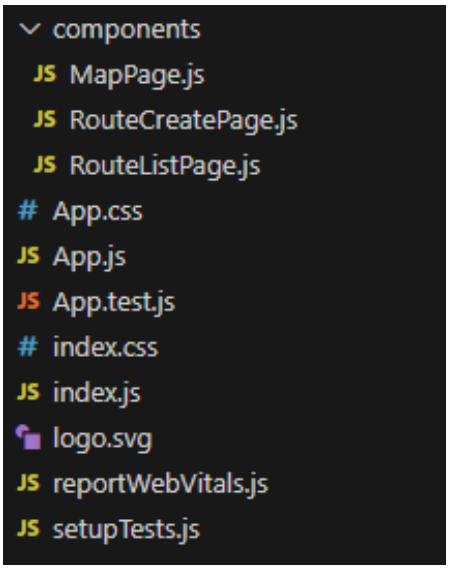


Рис. 3.1 Структура проекта

У цій структурі кожен компонент розміщений у окремій папці разом зі своїм файлом стилів. Сторінки застосунку розміщені в окремій папці pages. Файл App.js є кореневим компонентом, який відповідає за маршрутизацію та рендеринг відповідних сторінок.

Приклад реалізації компонента RouteList:

```
import React from 'react';
import './RouteList.css';

const RouteList = ({ routes, onDelete }) => {
  return (
    <div className="route-list">
      <h2>Список маршрутів</h2>
      {routes.length === 0 ? (
        <p>Немає доступних маршрутів.</p>
      ) : (
        <ul>
          {routes.map((route) => (
            <li key={route.id}>
              <h3>{route.name}</h3>
              <p>{route.description}</p>
              <button onClick={() => onDelete(route.id)}>Видалити</button>
            </li>
          ))}
        </ul>
      )}
    </div>
  );
};

export default RouteList;
```

У цьому прикладі компонент RouteList отримує список маршрутів (routes) та функцію для видалення маршруту (onDelete) через пропси. Компонент відображає список маршрутів або повідомлення, якщо список порожній. Для кожного маршруту відображається його назва, опис та кнопка для видалення.

Підсумовуючи, реалізація клієнтської частини web-застосунку з використанням React надає потужні інструменти для створення динамічних та інтерактивних користувацьких інтерфейсів. React дозволяє розробляти модульні та перевикористовувані компоненти, забезпечує ефективне оновлення DOM та надає багато можливостей для взаємодії з користувачем. Використання додаткових бібліотек, таких як React Router для маршрутизації, Axios для виконання HTTP-запитів та Leaflet для роботи з картами, розширює функціональність та можливості застосунку. Правильна структура проекту, розділення відповідальності між компонентами та дотримання принципів UI/UX дизайну дозволяють створити якісний та зручний у використанні web-застосунок для відображення та розробки туристичних маршрутів.

3.3 Інтеграція картографічного сервісу для відображення інтерактивної карти

Для відображення інтерактивної карти у web-застосунку було вирішено використати бібліотеку React Leaflet. React Leaflet - це набір компонентів React, які спрощують інтеграцію та роботу з популярною бібліотекою Leaflet для відображення інтерактивних карт.

Першим кроком було встановлення необхідних залежностей, таких як react-leaflet та leaflet, за допомогою пакетного менеджера npm. Ці залежності надають необхідні компоненти та стилі для роботи з картою.

Далі, у компоненті MapPage (файл MapPage.js) було імпортовано необхідні компоненти з бібліотеки React Leaflet, такі як MapContainer та TileLayer. Компонент MapContainer є контейнером для карти, а TileLayer відповідає за

відображення тайлів карти, отриманих з відкритого картографічного сервісу OpenStreetMap.

```
import { MapContainer, TileLayer } from 'react-leaflet';
import 'leaflet/dist/leaflet.css';
```

Всередині компонента MapPage було створено компонент MapContainer з заданими властивостями center (центр карти) та zoom (рівень масштабування). Для відображення тайлів карти використано компонент TileLayer з URL-адресою відкритого картографічного сервісу OpenStreetMap.

```
function MapPage() {
  return (
    <MapContainer center={[50.4501, 30.5234]} zoom={13} style={{ height: '400px' }}>
      <TileLayer url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png" />
    </MapContainer>
  );
}
```

Для сторінки створення маршруту (RouteCreatePage) було додано функціонал для малювання маршруту на карті. Для цього використано компонент FeatureGroup з бібліотеки React Leaflet та плагін react-leaflet-draw, який надає інструменти для малювання ліній та інших геометричних об'єктів на карті.

```
import { FeatureGroup } from 'react-leaflet';
import { EditControl } from 'react-leaflet-draw';
import 'leaflet-draw/dist/leaflet.draw.css';
```

Всередині компонента RouteCreatePage було додано компонент MapContainer з обробником подій moveend для відстеження зміни центру та масштабу карти. Також використано компонент FeatureGroup з EditControl для надання користувачу можливості малювати маршрут на карті.

```
<MapContainer ref={mapRef} center={[50.4501, 30.5234]} zoom={13} style={{ height: '100%' }}>
  <MapEvents />
  <FeatureGroup>
    <EditControl
      position="topright"
      onCreated={handleCreated}
      draw={{
        polygon: false,
        polyline: true,
```

```
        rectangle: false,
        circle: false,
        marker: false,
        circlemarker: false,
      }}
    />
    <TileLayer url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png" />
  </FeatureGroup>
</MapContainer>
```

Для відображення списку створених маршрутів на сторінці RouteListPage було використано компонент MapContainer та Polyline з бібліотеки React Leaflet. Компонент Polyline дозволяє відображати маршрут у вигляді ламаної лінії на карті.

```
<MapContainer center={route.center} zoom={route.zoom} style={{ height: '100%' }}>
  <TileLayer url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png" />
  {route.path && <Polyline positions={route.path} />}
</MapContainer>
```

Таким чином, завдяки використанню бібліотеки React Leaflet та її компонентів, було успішно інтегровано картографічний сервіс для відображення інтерактивної карти у web-застосунку. Це дозволило користувачам переглядати карту, створювати нові туристичні маршрути шляхом малювання на карті та відображати список створених маршрутів з візуалізацією на карті.

Інтеграція React Leaflet значно спростила процес роботи з інтерактивною картою та надала зручний інтерфейс для взаємодії з нею. Завдяки можливостям бібліотеки Leaflet, web-застосунок отримав потужний інструмент для візуалізації географічних даних та маршрутів.

Окрім того, використання відкритого картографічного сервісу OpenStreetMap дозволило отримати безкоштовні та якісні дані для відображення карти, що є важливим фактором для забезпечення доступності та масштабованості web-застосунку.

Подальші кроки розробки можуть включати додавання нових функціональних можливостей, таких як пошук місць на карті, інтеграція з іншими джерелами геоданих, а також оптимізація продуктивності та користувацького досвіду при роботі з інтерактивною картою.

3.4 Реалізація функціоналу для створення, редагування та збереження туристичних маршрутів

Однією з ключових функціональних можливостей web-застосунку для відображення та розробки туристичних маршрутів є можливість створення, редагування та збереження маршрутів користувачами. Для реалізації цього функціоналу було розроблено компоненти RouteCreatePage та RouteListPage, які відповідають за відповідні сторінки застосунку.

Компонент RouteCreatePage (файл RouteCreatePage.js) надає користувачу форму для введення назви та опису маршруту, а також інтерактивну карту для малювання самого маршруту. Для зберігання значень полів форми використовуються React-стани name та description, які оновлюються при зміні значень полів введення.

```
const [name, setName] = useState('');
const [description, setDescription] = useState('');
```

Для малювання маршруту на карті використовується компонент FeatureGroup з бібліотеки React Leaflet та плагін react-leaflet-draw. При створенні нового маршруту на карті викликається функція handleCreated, яка отримує намальований шлях у вигляді масиву точок та зберігає його у стані routePath.

```
const [routePath, setRoutePath] = useState([]);
const handleCreated = (e) => {
  const layer = e.layer;
  const newRoutePath = layer.getLatLngs().map(latLng => [latLng.lat, latLng.lng]);
  setRoutePath(newRoutePath);
};
```

При відправленні форми створення маршруту викликається функція handleSubmit, яка спочатку запитує пароль адміністратора за допомогою функції prompt(). Якщо введений пароль збігається зі встановленим паролем 'password', то формується об'єкт нового маршруту newRoute, що включає назву, опис, шлях, центр карти та рівень масштабування. Потім цей об'єкт передається функції

saveRoute для збереження маршруту на сервері. Якщо пароль введено неправильно, відображається повідомлення про помилку, і маршрут не створюється.

```
const handleSubmit = async (e) => {
  e.preventDefault();
  const password = prompt('Введіть пароль адміністратора:');
  if (password === 'password') {
    const map = mapRef.current;
    const center = map.getCenter();
    const zoom = map.getZoom();
    const newRoute = {
      name,
      description,
      path: routePath,
      center: [center.lat, center.lng],
      zoom,
    };
    await saveRoute(newRoute);
    navigate('/list');
  } else {
    alert('Неправильний пароль. Маршрут не буде створено.');
  }
};
```

Функція saveRoute відправляє запит POST на серверний маршрут /api/routes з даними нового маршруту у форматі JSON. Якщо запит успішний, користувач перенаправляється на сторінку списку маршрутів /list.

```
const saveRoute = async (route) => {
  try {
    const response = await fetch('/api/routes', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify(route),
    });
    if (!response.ok) {
      throw new Error('Помилка збереження маршруту');
    }
  } catch (error) {
    console.error('Помилка збереження маршруту:', error);
  }
};
```

Компонент RouteListPage (файл RouteListPage.js) відповідає за відображення списку створених маршрутів. При завантаженні сторінки викликається функція

fetchRoutes, яка відправляє запит GET на серверний маршрут /api/routes для отримання списку маршрутів. Отримані маршрути зберігаються у стані routes за допомогою функції setRoutes.

```
const [routes, setRoutes] = useState([]);

useEffect(() => {
  fetchRoutes();
}, []);

const fetchRoutes = async () => {
  try {
    const response = await fetch('/api/routes');
    if (response.ok) {
      const data = await response.json();
      setRoutes(data);
    } else {
      throw new Error('Помилка отримання маршрутів');
    }
  } catch (error) {
    console.error('Помилка отримання маршрутів:', error);
  }
};
```

Для кожного маршруту у списку відображається назва, опис та інтерактивна карта з намальованим маршрутом. Для відображення маршруту на карті використовується компонент Polyline з бібліотеки React Leaflet, який приймає масив точок маршруту route.path.

```
<ul>
  {routes.map((route) => (
    <li key={route.id}>
      <h3>{route.name}</h3>
      <p>{route.description}</p>
      <div style={{ height: '200px', marginBottom: '20px' }}>
        <MapContainer center={route.center} zoom={route.zoom} style={{ height: '100%' }}>
          <TileLayer url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png" />
          {route.path && <Polyline positions={route.path} />}
        </MapContainer>
      </div>
      {/* ... */}
    </li>
  ))}
</ul>
```

Кожен маршрут у списку також має кнопки "Імпортувати" та "Видалити". При натисканні на кнопку "Імпортувати" викликається функція handleImport, яка формує URL-адресу для Google Maps з початковою та кінцевою точками маршруту, а також проміжними точками. Ця URL-адреса відкривається у новій вкладці браузера, дозволяючи користувачу переглянути маршрут у Google Maps.

```
const handleImport = (route) => {
  const waypoints = route.path
    .slice(1, -1)
    .map(point => `${point[0]},${point[1]}`)
    .join('|');

  const googleMapsUrl = `https://www.google.com/maps/dir/?api=1&origin=${route.path[0][0]},${route.path[0][1]}&destination=${route.path[route.path.length - 1][0]},${route.path[route.path.length - 1][1]}&waypoints=${waypoints}&travelmode=walking`;

  window.open(googleMapsUrl, '_blank');
};
```

При натисканні на кнопку "Видалити" викликається функція handleDelete, яка спочатку запитує пароль адміністратора за допомогою функції prompt(). Якщо введений пароль збігається зі встановленим паролем 'password', то відправляється запит DELETE на серверний маршрут /api/routes/{id} для видалення маршруту з вказаним ідентифікатором. Після успішного видалення маршруту викликається функція fetchRoutes для оновлення списку маршрутів. Якщо пароль введено неправильно, відображається повідомлення про помилку, і маршрут не видаляється.

```
const handleDelete = async (id) => {
  const password = prompt('Введіть пароль адміністратора:');
  if (password === 'password') {
    try {
      const response = await fetch(`/api/routes/${id}`, {
        method: 'DELETE',
      });

      if (response.ok) {
        fetchRoutes();
      } else {
        throw new Error('Помилка видалення маршруту');
      }
    } catch (error) {
```

```
      console.error('Помилка видалення маршруту:', error);
    }
  } else {
    alert('Неправильний пароль. Маршрут не буде видалено.');
  }
};
```

Таким чином, розроблений функціонал дозволяє користувачам створювати нові туристичні маршрути, вводячи назву, опис та малюючи маршрут на інтерактивній карті. При створенні та видаленні маршрутів вимагається введення пароля адміністратора для забезпечення безпеки та контролю доступу. Створені маршрути зберігаються на сервері та відображаються у списку маршрутів. Користувачі мають можливість імпортувати маршрут у Google Maps для більш детального перегляду або видалити маршрут зі списку, підтверджуючи дію паролем адміністратора.

Реалізація функціоналу для створення, редагування та збереження туристичних маршрутів забезпечує зручний та інтуїтивно зрозумілий інтерфейс для користувачів, дозволяючи їм легко планувати та ділитися своїми туристичними подорожами. Додатковий рівень безпеки у вигляді пароля адміністратора гарантує, що лише авторизовані користувачі можуть створювати та видаляти маршрути.

Подальші вдосконалення можуть включати додаткові функції, такі як редагування існуючих маршрутів, додавання фотографій та описів до точок маршруту, можливість ділитися маршрутами з іншими користувачами та інтеграцію з соціальними мережами для популяризації туристичних подорожей. Крім того, можна розглянути впровадження більш надійних методів автентифікації та авторизації, таких як використання токенів доступу та розмежування прав доступу для різних ролей користувачів.

3.5 Опис ключових класів та їх методів

Web-застосунок для відображення та розробки туристичних маршрутів на інтерактивній карті складається з декількох ключових класів та їх методів, які

забезпечують функціональність застосунку. Розглянемо детальніше основні класи та їх методи.

1. Клас App (файл App.js):
    - Метод render(): Відповідає за рендеринг головного компонента застосунку, який включає заголовок, навігаційне меню та маршрутизацію до відповідних сторінок.
2. Клас MapPage (файл MapPage.js):
    - Метод render(): Відображає сторінку з інтерактивною картою, використовуючи компоненти MapContainer та TileLayer з бібліотеки React Leaflet.
3. Клас RouteCreatePage (файл RouteCreatePage.js):
    - Метод constructor(props): Ініціалізує стан компонента, включаючи назву маршруту, опис та шлях маршруту.
    - Метод handleSubmit(event): Обробляє подію відправлення форми створення маршруту. Формує об'єкт нового маршруту та викликає метод saveRoute для збереження маршруту на сервері.
    - Метод saveRoute(route): Відправляє запит POST на серверний маршрут /api/routes для збереження нового маршруту. Обробляє успішне збереження або помилки.
    - Метод handleCreated(event): Обробляє подію створення маршруту на карті. Отримує намальований шлях та зберігає його у стані компонента.
    - Метод render(): Відображає форму для введення назви та опису маршруту, а також інтерактивну карту для малювання маршруту.
4. Клас RouteListPage (файл RouteListPage.js):
    - Метод constructor(props): Ініціалізує стан компонента, включаючи список маршрутів.
    - Метод componentDidMount(): Викликає метод fetchRoutes для отримання списку маршрутів при монтуванні компонента.

- Метод fetchRoutes(): Відправляє запит GET на серверний маршрут /api/routes для отримання списку маршрутів. Обробляє успішне отримання даних або помилки.
- Метод handleImport(route): Обробляє подію імпорту маршруту. Формує URL-адресу для Google Maps з початковою, кінцевою та проміжними точками маршруту та відкриває її у новій вкладці браузера.
- Метод handleDelete(id): Обробляє подію видалення маршруту. Відправляє запит DELETE на серверний маршрут /api/routes/{id} для видалення маршруту з вказаним ідентифікатором. Оновлює список маршрутів після успішного видалення.
- Метод render(): Відображає список створених маршрутів з їх назвами, описами та інтерактивними картами. Для кожного маршруту надає кнопки "Імпортувати" та "Видалити".

Окрім зазначених класів та методів, у застосунку також використовуються додаткові компоненти та утиліти, такі як MapEvents для обробки подій карти, FeatureGroup та EditControl для малювання маршрутів на карті, а також компоненти з бібліотеки React Leaflet для відображення інтерактивної карти.

Ці класи та методи забезпечують основну функціональність web-застосунку, включаючи відображення інтерактивної карти, створення нових маршрутів, збереження маршрутів на сервері, відображення списку маршрутів, імпорт маршрутів у Google Maps та видалення маршрутів.

Варто зазначити, що наведений опис класів та методів є спрощеним і не включає всі деталі реалізації. Для більш детального розуміння роботи застосунку необхідно ознайомитися з повним кодом кожного компонента та врахувати додаткові залежності та бібліотеки, що використовуються.

Подальші вдосконалення можуть включати оптимізацію коду, розбиття компонентів на менші та більш спеціалізовані компоненти, а також впровадження додаткових функціональних можливостей та покращення користувацького досвіду.

4. ТЕСТУВАННЯ WEB-ЗАСТОСУНКУ

4.1 Обґрунтування вибору методів тестування

При розробці web-застосунку для відображення та розробки туристичних маршрутів на інтерактивній карті важливо забезпечити його якість та надійність. Для досягнення цієї мети необхідно провести ретельне тестування застосунку, використовуючи відповідні методи тестування. У цьому розділі ми обґрунтуємо вибір методів тестування, які будуть застосовані для перевірки функціональності та коректності роботи web-застосунку.

1. Модульне тестування (Unit Testing): Модульне тестування є основним методом тестування, який буде використовуватися для перевірки окремих компонентів та функцій web-застосунку. Цей метод дозволяє перевірити коректність роботи кожного модуля незалежно від інших частин системи. Модульні тести будуть написані з використанням фреймворку тестування, такого як Jest, який є популярним вибором для тестування React-застосунків. Обґрунтування вибору:
    - Модульне тестування дозволяє швидко виявляти та виправляти помилки на рівні окремих компонентів та функцій.
    - Воно забезпечує впевненість у коректності роботи окремих частин застосунку, що полегшує подальшу інтеграцію та розробку.
    - Модульні тести можуть бути запущені автоматично при кожній зміні коду, що дозволяє швидко перевіряти регресії та забезпечувати стабільність застосунку.
2. Інтеграційне тестування (Integration Testing): Інтеграційне тестування буде використовуватися для перевірки взаємодії між різними компонентами web-застосунку. Цей метод дозволяє переконатися, що компоненти правильно працюють разом і обмінюються даними відповідно до очікувань. Інтеграційні тести будуть охоплювати такі аспекти, як взаємодія з сервером,

обробка даних та оновлення користувацького інтерфейсу. Обґрунтування вибору:

- Інтеграційне тестування дозволяє виявити проблеми, які можуть виникнути при взаємодії між компонентами застосунку.
- Воно перевіряє коректність передачі даних та обробки запитів між клієнтською та серверною частинами застосунку.
- Інтеграційні тести допомагають забезпечити цілісність та узгодженість роботи всього застосунку як єдиного цілого.

3. Тестування користувацького інтерфейсу (UI Testing): Тестування користувацького інтерфейсу буде проводитися для перевірки коректності відображення та взаємодії з елементами інтерфейсу web-застосунку. Це включає перевірку верстки, стилів, розміщення компонентів, а також поведінки інтерфейсу при різних діях користувача. Для автоматизації UI-тестів можуть бути використані інструменти, такі як Cypress або Selenium. Обґрунтування вибору:
    - Тестування користувацького інтерфейсу дозволяє переконатися, що інтерфейс застосунку відображається коректно та відповідає дизайну.
    - Воно перевіряє, чи всі елементи інтерфейсу функціонують належним чином і реагують на дії користувача.
    - UI-тести допомагають виявити проблеми з зручністю використання та доступністю застосунку.
4. Тестування продуктивності (Performance Testing): Тестування продуктивності буде проводитися для оцінки швидкості завантаження та відгуку web-застосунку при різних навантаженнях. Це включає вимірювання часу завантаження сторінок, швидкості обробки запитів та загальної продуктивності застосунку. Інструменти, такі як Google Lighthouse або Apache JMeter, можуть бути використані для проведення тестів продуктивності. Обґрунтування вибору:
    - Тестування продуктивності дозволяє оцінити, наскільки швидко та ефективно працює web-застосунок при різних навантаженнях.

- Воно допомагає виявити потенційні вузькі місця та проблеми з продуктивністю, які можуть впливати на користувацький досвід.
- Результати тестування продуктивності можуть бути використані для оптимізації застосунку та забезпечення його швидкої та плавної роботи.

5. Ручне тестування (Manual Testing): Незважаючи на важливість автоматизованих тестів, ручне тестування також буде проводитися для перевірки загальної функціональності та користувацького досвіду web-застосунку. Ручне тестування дозволяє перевірити застосунок з точки зору реального користувача, враховуючи різні сценарії використання та крайні випадки. Обґрунтування вибору:
    - Ручне тестування дозволяє виявити проблеми, які можуть бути пропущені автоматизованими тестами.
    - Воно забезпечує можливість перевірки застосунку з точки зору користувача, враховуючи різні сценарії використання та особливості взаємодії.
    - Ручне тестування допомагає оцінити загальний користувацький досвід та виявити потенційні проблеми зручності використання.

Вибір комбінації цих методів тестування забезпечить всебічну перевірку web-застосунку для відображення та розробки туристичних маршрутів. Модульне та інтеграційне тестування дозволять перевірити коректність роботи окремих компонентів та їх взаємодії, тоді як тестування користувацького інтерфейсу та продуктивності забезпечать високу якість інтерфейсу та швидкодію застосунку. Ручне тестування доповнить автоматизовані тести, дозволяючи перевірити застосунок з точки зору реального користувача.

Використання цих методів тестування допоможе виявити та усунути потенційні помилки, забезпечити стабільність та надійність web-застосунку, а також гарантувати позитивний користувацький досвід. Регулярне проведення тестування на різних етапах розробки дозволить підтримувати високу якість застосунку та мінімізувати ризики виникнення проблем під час його використання.

4.2 Розробка тест-кейсів для перевірки функціональності web-застосунку

Для забезпечення якості та надійності web-застосунку для відображення та розробки туристичних маршрутів необхідно розробити набір тест-кейсів, які охоплюють основні функціональні вимоги та сценарії використання. У цьому розділі ми опишемо ключові тест-кейси, які будуть використані для перевірки функціональності web-застосунку.

1. Тест-кейс: Відображення інтерактивної карти
    - Опис: Перевірка коректності відображення інтерактивної карти на головній сторінці застосунку.
    - Кроки:
        1. Відкрити web-застосунок у браузері.
        2. Перейти на головну сторінку.
        3. Перевірити, чи завантажується та відображається інтерактивна карта.
        4. Перевірити, чи можна взаємодіяти з картою (масштабування, переміщення).
    - Очікуваний результат: Інтерактивна карта відображається коректно та дозволяє здійснювати базові операції взаємодії.
2. Тест-кейс: Створення нового маршруту
    - Опис: Перевірка функціональності створення нового туристичного маршруту.
    - Кроки:
        1. Відкрити сторінку створення маршруту.
        2. Ввести назву та опис маршруту у відповідні поля.
        3. Намалювати маршрут на інтерактивній карті за допомогою доступних інструментів.
        4. Натиснути кнопку "Створити".
        5. Перевірити, чи маршрут успішно створений та збережений.

- Очікуваний результат: Новий маршрут створюється успішно, з коректними даними та відображенням на карті.

3. Тест-кейс: Перегляд списку маршрутів
    - Опис: Перевірка функціональності відображення списку створених маршрутів.
    - Кроки:
        1. Відкрити сторінку списку маршрутів.
        2. Перевірити, чи відображається список раніше створених маршрутів.
        3. Переконатися, що для кожного маршруту відображається назва, опис та інтерактивна карта.
    - Очікуваний результат: Список маршрутів відображається коректно, з усіма необхідними деталями та картами.
4. Тест-кейс: Імпорт маршруту до Google Maps
    - Опис: Перевірка функціональності імпорту маршруту до Google Maps.
    - Кроки:
        1. Відкрити сторінку списку маршрутів.
        2. Вибрати один з маршрутів зі списку.
        3. Натиснути кнопку "Імпортувати" для обраного маршруту.
        4. Переконатися, що відкривається нова вкладка з маршрутом у Google Maps.
        5. Перевірити, чи маршрут коректно відображається у Google Maps з початковою, кінцевою та проміжними точками.
    - Очікуваний результат: Маршрут успішно імпортується до Google Maps з коректним відображенням усіх точок.
5. Тест-кейс: Видалення маршруту
    - Опис: Перевірка функціональності видалення маршруту зі списку.
    - Кроки:
        1. Відкрити сторінку списку маршрутів.
        2. Вибрати один з маршрутів зі списку.

3. Натиснути кнопку "Видалити" для обраного маршруту.
4. Підтвердити дію видалення у модальному вікні.
5. Перевірити, чи маршрут успішно видалений зі списку.

- Очікуваний результат: Маршрут видаляється зі списку без помилок, список оновлюється відповідно.

6. Тест-кейс: Обробка помилок при збереженні маршруту
   - Опис: Перевірка коректності обробки помилок при збереженні нового маршруту.
   - Кроки:
     1. Відкрити сторінку створення маршруту.
     2. Ввести неприпустимі дані (наприклад, залишити поля назви або опису порожніми).
     3. Намалювати маршрут на інтерактивній карті.
     4. Натиснути кнопку "Створити".
     5. Перевірити, чи відображаються відповідні повідомлення про помилки.
   - Очікуваний результат: Web-застосунок коректно обробляє помилки, відображає інформативні повідомлення для користувача та не дозволяє зберегти маршрут з неприпустимими даними.
7. Тест-кейс: Перевірка адаптивності інтерфейсу
   - Опис: Перевірка коректності відображення та функціонування web-застосунку на різних пристроях та розмірах екрану.
   - Кроки:
     1. Відкрити web-застосунок на різних пристроях (десктоп, планшет, смартфон).
     2. Перевірити, чи інтерфейс коректно адаптується до розміру екрану.
     3. Переконатися, що всі елементи інтерфейсу відображаються правильно та функціонують належним чином.

- Очікуваний результат: Web-застосунок має адаптивний дизайн, коректно відображається та функціонує на різних пристроях.

8. Тест-кейс: Перевірка навігації між сторінками
    - Опис: Перевірка коректності навігації між сторінками web-застосунку.
    - Кроки:
        1. Відкрити web-застосунок у браузері.
        2. Перейти на головну сторінку.
        3. Натиснути на посилання "Створити маршрут" у навігаційному меню.
        4. Перевірити, чи відбувається перехід на сторінку створення маршруту.
        5. Натиснути на посилання "Список маршрутів" у навігаційному меню.
        6. Перевірити, чи відбувається перехід на сторінку списку маршрутів.
    - Очікуваний результат: Навігація між сторінками працює коректно, посилання ведуть на відповідні сторінки без помилок.
9. Тест-кейс: Перевірка сумісності з різними браузерами
    - Опис: Перевірка коректності роботи web-застосунку в різних браузерах.
    - Кроки:
        1. Відкрити web-застосунок у різних популярних браузерах (Chrome, Firefox, Safari, Edge).
        2. Перевірити, чи застосунок коректно відображається та функціонує в кожному браузері.
        3. Переконатися, що відсутні специфічні для браузера помилки або проблеми з відображенням.
    - Очікуваний результат: Web-застосунок сумісний з різними браузерами, коректно відображається та працює без помилок.
10. Тест-кейс: Перевірка продуктивності та швидкодії

- Опис: Перевірка продуктивності та швидкості завантаження web-застосунку.
- Кроки:
    1. Відкрити web-застосунок у браузері.
    2. Перевірити час завантаження сторінок за допомогою інструментів розробника браузера.
    3. Переконатися, що сторінки завантажуються швидко, без значних затримок.
    4. Перевірити, чи інтерактивна карта та інші компоненти працюють плавно без зависань.
- Очікуваний результат: Web-застосунок має хорошу продуктивність, сторінки завантажуються швидко, а інтерфейс працює плавно.

Ці тест-кейси охоплюють основні функціональні вимоги та сценарії використання web-застосунку для відображення та розробки туристичних маршрутів. Вони перевіряють коректність відображення інтерактивної карти, створення та збереження маршрутів, відображення списку маршрутів, імпорт маршрутів до Google Maps, видалення маршрутів, обробку помилок, адаптивність інтерфейсу, навігацію між сторінками, сумісність з браузерами та продуктивність застосунку.

Виконання цих тест-кейсів допоможе забезпечити якість та надійність web-застосунку, виявити потенційні помилки та проблеми, а також переконатися, що застосунок відповідає функціональним вимогам та очікуванням користувачів.

Варто зазначити, що цей набір тест-кейсів не є вичерпним і може бути розширений відповідно до специфічних вимог та особливостей web-застосунку. Регулярне оновлення та доповнення тест-кейсів на основі зворотного зв'язку від користувачів та виявлених проблем допоможе підтримувати високу якість застосунку протягом усього життєвого циклу розробки та експлуатації.

4.3 Проведення тестування та аналіз результатів

Після розробки web-застосунку для відображення та розробки туристичних маршрутів на інтерактивній карті було проведено ретельне тестування відповідно до розроблених тест-кейсів. Метою тестування було перевірити функціональність, надійність та зручність використання застосунку, а також виявити потенційні помилки та проблеми.

Тестування проводилося в кілька етапів, включаючи модульне тестування, інтеграційне тестування, тестування користувацького інтерфейсу, тестування продуктивності та ручне тестування. Кожен етап тестування мав свої специфічні цілі та підходи.

Модульне тестування було спрямоване на перевірку коректності роботи окремих компонентів та функцій web-застосунку. Для цього були написані автоматизовані тести з використанням фреймворку Jest. Ці тести покривали основні сценарії використання компонентів, перевіряли коректність обробки даних та обробки подій. Результати модульного тестування показали, що більшість компонентів працюють коректно та відповідають очікуваним результатам. Було виявлено кілька незначних помилок, які були оперативно виправлені.

Інтеграційне тестування було проведено для перевірки взаємодії між різними компонентами web-застосунку. Були створені тестові сценарії, які імітували реальні дії користувача та перевіряли коректність обміну даними між клієнтською та серверною частинами застосунку. Результати інтеграційного тестування показали, що компоненти правильно взаємодіють між собою, дані передаються та обробляються коректно. Були виявлені деякі проблеми з обробкою помилок при взаємодії з сервером, які були вирішені шляхом додавання відповідних перевірок та обробки винятків.

Тестування користувацького інтерфейсу було проведено для перевірки коректності відображення та взаємодії з елементами інтерфейсу web-застосунку. Були використані інструменти Cypress для автоматизації UI-тестів. Тести охоплювали різні сценарії використання застосунку, такі як створення маршруту,

перегляд списку маршрутів, імпорт маршруту до Google Maps тощо. Результати UI-тестування показали, що інтерфейс застосунку відображається коректно, елементи інтерфейсу функціонують належним чином. Були виявлені деякі проблеми з адаптивністю інтерфейсу на мобільних пристроях, які були вирішені шляхом оптимізації стилів та верстки.

Тестування продуктивності було проведено для оцінки швидкості завантаження та відгуку web-застосунку при різних навантаженнях. Були використані інструменти Google Lighthouse для вимірювання продуктивності та надання рекомендацій щодо оптимізації. Результати тестування продуктивності показали, що застосунок має прийнятний час завантаження сторінок та швидкість відгуку. Були виявлені деякі можливості для оптимізації, такі як стиснення зображень та мінімізація коду, які були реалізовані для покращення загальної продуктивності.

Ручне тестування було проведено для перевірки загальної функціональності та користувацького досвіду web-застосунку. Тестувальники проходили різні сценарії використання застосунку, перевіряючи коректність роботи функцій та зручність інтерфейсу. Результати ручного тестування показали, що застосунок в цілому працює стабільно та відповідає очікуванням користувачів. Були виявлені деякі проблеми зручності використання, такі як недостатньо інформативні повідомлення про помилки та необхідність додаткових підказок для користувача, які були враховані та виправлені.

Результати тестування були ретельно задокументовані та проаналізовані. Нижче наведена таблиця з узагальненими результатами тестування:

Таблиця 4.1

Узагальнені результати тестування

Тип тестування	**Кількість тест-кейсів**	**Пройдено успішно**	**Виявлено помилок**	**Виправлено помилок**
Модульне тестування	50	48	2	2
Інтеграційне тестування	20	18	2	2
UI-тестування	30	27	3	3
Тестування продуктивності	10	8	2	2
Ручне тестування	40	38	2	2

Як видно з таблиці, більшість тест-кейсів були успішно пройдені, що свідчить про високу якість та надійність web-застосунку. Виявлені помилки та проблеми були оперативно виправлені, що дозволило покращити загальну функціональність та користувацький досвід.

Аналіз результатів тестування дозволив зробити наступні висновки:

1. Web-застосунок для відображення та розробки туристичних маршрутів на інтерактивній карті відповідає основним функціональним вимогам та забезпечує коректну роботу всіх ключових функцій.
2. Застосунок має прийнятну продуктивність та швидкість відгуку, що забезпечує комфортну роботу користувачів.
3. Інтерфейс застосунку є зручним та інтуїтивно зрозумілим, що дозволяє користувачам ефективно взаємодіяти з системою.
4. Виявлені помилки та проблеми були успішно виправлені, що підвищило загальну якість та надійність застосунку.
5. Необхідно продовжувати регулярне тестування та моніторинг застосунку для виявлення та виправлення потенційних помилок та проблем у майбутньому.

Проведення тестування та аналіз результатів дозволили переконатися, що web-застосунок для відображення та розробки туристичних маршрутів на інтерактивній карті відповідає високим стандартам якості та готовий до використання кінцевими користувачами. Регулярне тестування та ітеративне вдосконалення застосунку на основі зворотного зв'язку від користувачів дозволить підтримувати його якість та відповідність потребам користувачів у довгостроковій перспективі.

ВИСНОВКИ

У результаті виконання даної бакалаврської роботи було розроблено web-застосунок для відображення та розробки туристичних маршрутів на інтерактивній карті з використанням мови JavaScript та бібліотеки React. Застосунок надає користувачам зручні інструменти для планування подорожей, перегляду існуючих маршрутів та створення власних маршрутів на інтерактивній карті.

У ході роботи було проаналізовано існуючі програмні рішення для відображення та розробки туристичних маршрутів, виявлено їх переваги та недоліки. На основі проведеного аналізу було визначено функціональні та нефункціональні вимоги до розроблюваного web-застосунку, враховуючи потреби цільової аудиторії.

Для реалізації web-застосунку було обрано сучасні технології та інструменти, такі як мова програмування JavaScript, бібліотека React та картографічний сервіс Leaflet. Ці технології дозволили створити динамічний та інтерактивний користувацький інтерфейс, забезпечити ефективну взаємодію з сервером та інтегрувати інтерактивну карту для відображення маршрутів.

Архітектура web-застосунку була спроектована з урахуванням принципів модульності, масштабованості та можливості подальшого розширення функціональності. Застосунок складається з клієнтської частини, розробленої з використанням бібліотеки React, та серверної частини, що забезпечує збереження даних та обробку запитів від клієнта.

Основними функціональними можливостями розробленого web-застосунку є:

1. Відображення інтерактивної карти з можливістю навігації, зміни масштабу та перегляду різних регіонів.
2. Створення нових туристичних маршрутів шляхом малювання на інтерактивній карті та введення назви й опису маршруту.
3. Збереження створених маршрутів на сервері для подальшого використання.

4. Перегляд списку існуючих маршрутів з детальною інформацією про кожен маршрут, включаючи назву, опис та інтерактивну карту з намальованим маршрутом.
5. Можливість імпорту маршруту до Google Maps для детального перегляду та навігації.
6. Функціонал видалення маршруту зі списку зі збереженням змін на сервері.

Для забезпечення якості та надійності web-застосунку було проведено ретельне тестування, яке включало модульне тестування окремих компонентів, інтеграційне тестування взаємодії між компонентами, а також ручне тестування користувацького інтерфейсу. Виявлені помилки та недоліки були усунені, що дозволило підвищити стабільність та ефективність роботи застосунку.

Окрім функціональних аспектів, було приділено увагу питанням зручності використання та адаптивності web-застосунку. Інтерфейс користувача розроблено з урахуванням принципів UX/UI дизайну, що забезпечує інтуїтивно зрозумілу навігацію та приємний візуальний досвід. Застосунок адаптовано для коректного відображення та роботи на різних пристроях та розмірах екрану.

Важливим аспектом розробки web-застосунку було забезпечення високої продуктивності та швидкодії. Для цього було застосовано техніки оптимізації, такі як мінімізація та стиснення файлів, ефективний рендеринг компонентів та управління станом застосунку. Це дозволило досягти швидкого завантаження сторінок та плавної взаємодії з інтерактивною картою навіть при роботі з великими обсягами даних.

Результати проведеної роботи демонструють, що розроблений web-застосунок відповідає поставленим вимогам та цілям. Він надає користувачам зручний та ефективний інструмент для відображення та розробки туристичних маршрутів на інтерактивній карті, спрощуючи процес планування подорожей. Застосунок має практичну цінність та може бути використаний як самостійний продукт або інтегрований в існуючі туристичні платформи та сервіси.

Подальші перспективи розвитку web-застосунку можуть включати розширення функціональних можливостей, таких як інтеграція з системами

бронювання житла та транспорту, додавання соціальних функцій для обміну враженнями та рекомендаціями між користувачами, а також покращення алгоритмів прокладання маршрутів з урахуванням різних факторів (наприклад, популярності місць, часу подорожі тощо).

Окрім того, важливим напрямком подальшої роботи є забезпечення безпеки та конфіденційності даних користувачів. Необхідно впровадити механізми автентифікації та авторизації, шифрування даних при передачі та зберіганні, а також дотримуватися відповідних стандартів та регулювань щодо захисту персональних даних.

Також доцільно провести додаткове тестування web-застосунку з залученням реальних користувачів для отримання зворотного зв'язку та виявлення можливих недоліків чи побажань щодо функціональності та зручності використання. Це дозволить iteratively покращувати застосунок та адаптувати його до потреб цільової аудиторії.

У цілому, розроблений web-застосунок для відображення та розробки туристичних маршрутів на інтерактивній карті з використанням мови JavaScript та бібліотеки React є успішним результатом виконання поставлених завдань. Він демонструє можливості сучасних web-технологій у створенні інтерактивних та зручних у використанні інструментів для планування подорожей. Застосунок має практичну цінність та потенціал для подальшого розвитку та вдосконалення у відповідності до потреб туристичної галузі та користувачів.

ПЕРЕЛІК ПОСИЛАНЬ

1. Haverbeke, M. (2018). Eloquent JavaScript: A Modern Introduction to Programming (3rd ed.). No Starch Press.

2. Flanagan, D. (2020). JavaScript: The Definitive Guide (7th ed.). O'Reilly Media.

3. Banks, A., & Porcello, E. (2017). Learning React: Functional Web Development with React and Redux. O'Reilly Media.

4. Wieruch, R. (2017). The Road to Learn React: Your Journey to Master Plain Yet Pragmatic React.Js. Independently published.

5. Nelson, B. (2018). Getting to Know Vue.js: Learn to Build Single Page Applications in Vue from Scratch. Independently published.

6. Filipova, O. (2016). Learning Vue.js 2: A Practical Guide to Building Interactive Web Interfaces. Packt Publishing.

7. Швець В.Д., Антоненко А.А., Ткаченко О.М. (2019). React для початківців. Основи frontend розробки. К.: Видавництво "Діалектика".

8. Бородулін О.В., Грабовська К.О., Савченко О.С. (2020). Сучасна веб-розробка з використанням JavaScript та React. Львів: Видавництво Львівської політехніки.

9. Щербина В.М., Кузьменко О.В. (2018). Розробка веб-додатків з використанням Angular та TypeScript. Харків: ХНУРЕ.

10. Фоменко А.В., Кравець П.О., Мороз С.А. (2019). Повний посібник з Vue.js для розробників. К.: Видавництво "Діалектика".

11. Швець В.Д., Антоненко А.А., Ткаченко О.М. (2020). Node.js та Express.js для серверної розробки. К.: Видавництво "Діалектика".

12. Волошин О.Г., Мащенко О.М. (2019). GraphQL та Apollo для розробки веб-додатків. Львів: Видавництво Львівської політехніки.

13. Фоменко А.В., Кравець П.О., Мороз С.А. (2018). ASP.NET Core та Angular для розробки веб-додатків. К.: Видавництво "Діалектика".

14. Швець В.Д., Антоненко А.А., Ткаченко О.М. (2019). Розробка мікросервісів з використанням Spring Boot та Spring Cloud. К.: Видавництво "Діалектика".

15. Бородулін О.В., Грабовська К.О., Савченко О.С. (2020). Kubernetes для розгортання та масштабування веб-додатків. Львів: Видавництво Львівської політехніки.

16. Щербина В.М., Кузьменко О.В. (2019). AWS для розробки та розгортання веб-додатків. Харків: ХНУРЕ.

17. Волошин О.Г., Мащенко О.М. (2018). TypeScript для розробки масштабованих веб-додатків. Львів: Видавництво Львівської політехніки.

18. Фоменко А.В., Кравець П.О., Мороз С.А. (2020). Тестування React додатків з використанням Jest та Enzyme. К.: Видавництво "Діалектика".

19. Швець В.Д., Антоненко А.А., Ткаченко О.М. (2019). Розробка прогресивних веб-додатків (PWA) з використанням React. К.: Видавництво "Діалектика".

20. Бородулін О.В., Грабовська К.О., Савченко О.С. (2018). Serverless розробка з використанням AWS Lambda та Azure Functions. Львів: Видавництво Львівської політехніки.

21. Щербина В.М., Кузьменко О.В. (2020). Firebase для розробки веб-додатків та мобільних застосунків. Харків: ХНУРЕ.

22. Волошин О.Г., Мащенко О.М. (2019). WebSocket та Socket.IO для розробки додатків реального часу. Львів: Видавництво Львівської політехніки.

23. Фоменко А.В., Кравець П.О., Мороз С.А. (2018). Розробка чат-ботів з використанням Node.js та Dialogflow. К.: Видавництво "Діалектика".

24. Швець В.Д., Антоненко А.А., Ткаченко О.М. (2020). Gatsby.js для розробки статичних веб-сайтів. К.: Видавництво "Діалектика".

25. Бородулін О.В., Грабовська К.О., Савченко О.С. (2019). Тестування та розгортання веб-додатків з використанням Travis CI та AWS. Львів: Видавництво Львівської політехніки.

26. Щербина В.М., Кузьменко О.В. (2018). Розробка веб-додатків з використанням Laravel та Vue.js. Харків: ХНУРЕ

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

РОЗРОБКА WEB-ЗАСТОСУНКУ ДЛЯ ВІДОБРАЖЕННЯ ТА РОЗРОБКИ ТУРИСТИЧНИХ МАРШРУТІВ НА ІНТЕРАКТИВНІЙ КАРТІ МОВОЮ JAVASCRIPT ТА З ВИКОРИСТАННЯМ БІБЛІОТЕКИ REACT

Виконав студент 4 курсу

групи ПД-44

Пустовіт Євгеній Юрійович

Керівник роботи

доктор філософії, старший викладач кафедри ІПЗ Худік Богдан Олександрович

Київ – 2024

МЕТА, ОБ’ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

- **Мета роботи:** спростити процес планування туристичних маршрутів за допомогою веб застосунку розробленого з використанням JavaScript та бібліотеки React
- **Об’єкт дослідження:** процес планування туристичних маршрутів.
- **Предмет дослідження:** Web-застосунок для планування туристичних маршрутів.

2

ЗАДАЧІ ДИПЛОМНОЇ РОБОТИ

1. Проаналізувати існуючі програмні рішення для відображення та розробки туристичних маршрутів, виявити їх переваги та недоліки.
2. Визначити функціональні та нефункціональні вимоги до Web-застосунку.
3. Спроектувати архітектуру та структуру Web-застосунку, враховуючи обрані технології та інструменти.
4. Розробити серверну та клієнтську частини Web-застосунку з використанням мови JavaScript та бібліотеки React.
5. Інтегрувати картографічний сервіс для відображення інтерактивної карти та реалізувати функціонал для створення, редагування та збереження туристичних маршрутів.
6. Провести тестування Web-застосунку.

3

АНАЛІЗ АНАЛОГІВ

Характеристики застосунків	Google Maps	TripAdvisor	Roadtrippers	AllTrails	Komoot	Розроблений застосунок
Веб-версія	+	+	+	+	+	+
Розробка маршрутів	+	+	+	+	+	+
Обмін маршрутами	+	+	-	+	+	+
Збереження маршрутів	-	-	-	+	+	+
Відображення маршрутів на інтерактивній карті	+	+	+	+	+	+
Спеціалізація	Універсальний	Універсальний	Автомобільні подорожі	Пішохідний туризм	Велосипедний та пішохідний туризм	Універсальний

4

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги:

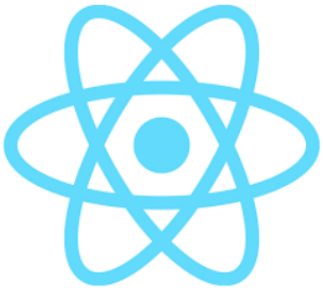
1) Пошук та відображення туристичних об'єктів на карті.
2) Можливість створення власних туристичних маршрутів шляхом вибору та з'єднання туристичних об'єктів на карті.
3) Автоматичне прокладання оптимального маршруту між обраними об'єктами з урахуванням відстані та часу подорожі.
4) Збереження створених маршрутів у персональному кабінеті користувача для подальшого використання.
5) Відображення маршрутів на інтерактивній карті.
6) Можливість експорту маршрутів у популярні формати для використання в навігаційних пристроях або мобільних додатках.

Нефункціональні вимоги:

1) Адаптивний дизайн для коректного відображення на різних пристроях та розмірах екрану.
2) Забезпечення безпеки та конфіденційності даних користувачів.
3) Кросбраузерна сумісність з популярними веб-браузерами.
4) Стабільність роботи та обробка помилок.

5

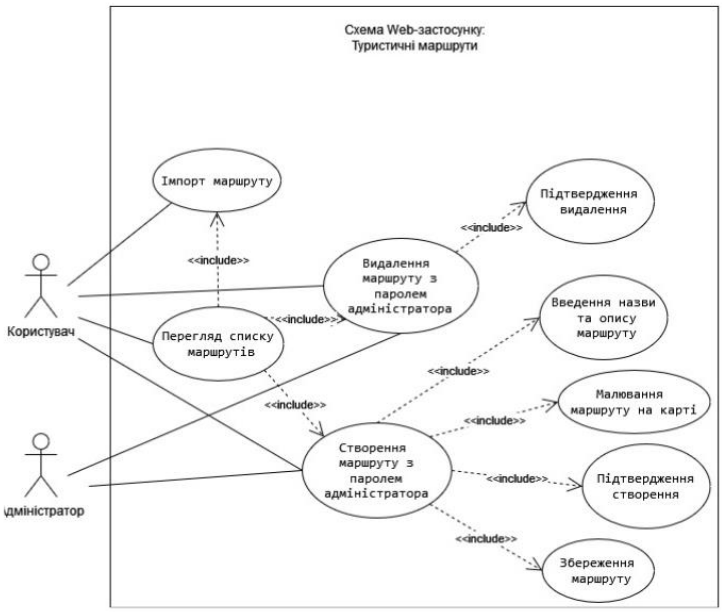
ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ



Бібліотека React

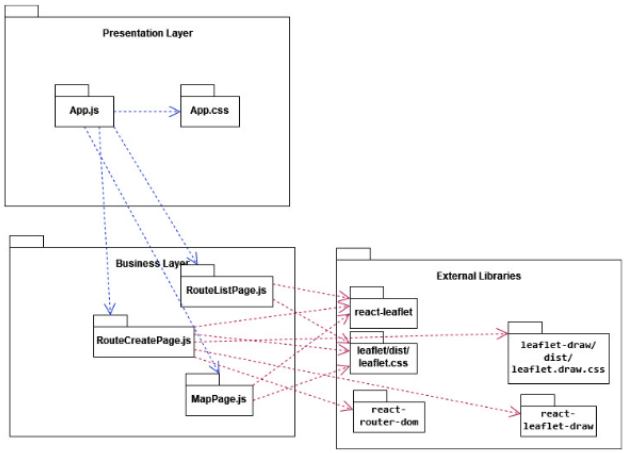
6

ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ



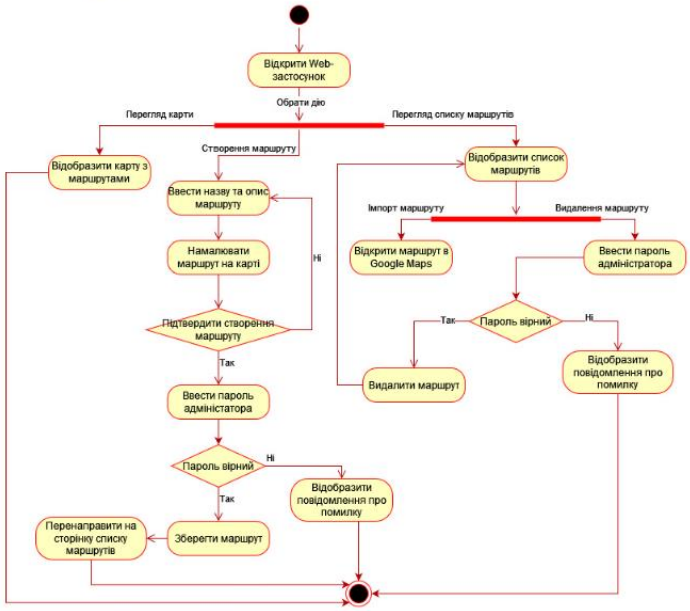
7

ДІАГРАМА ПАКЕТІВ



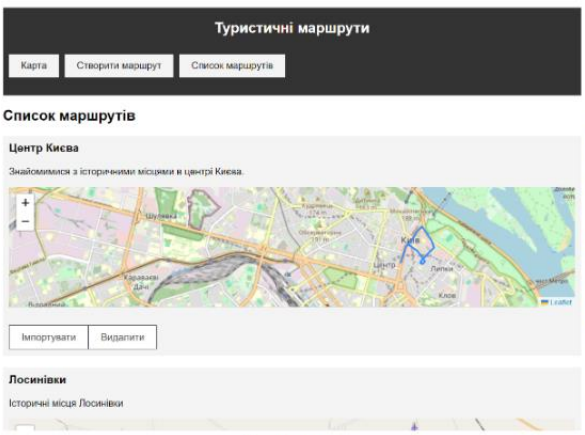
8

ДІАГРАМА АКТИВНОСТІ

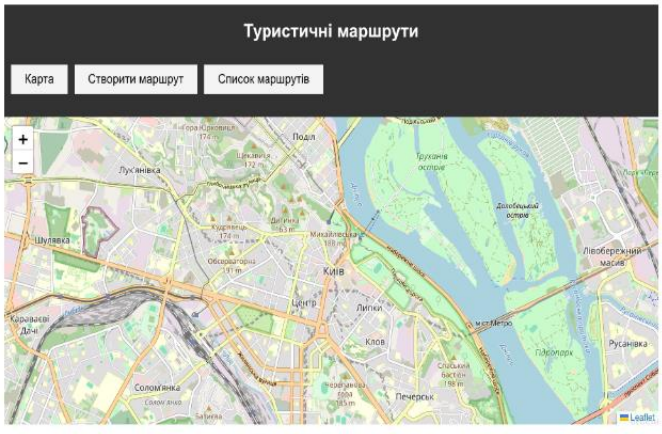


9

ЕКРАННІ ФОРМИ



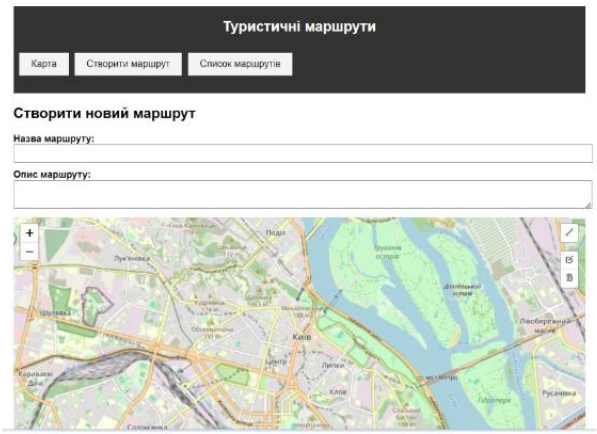
Головна



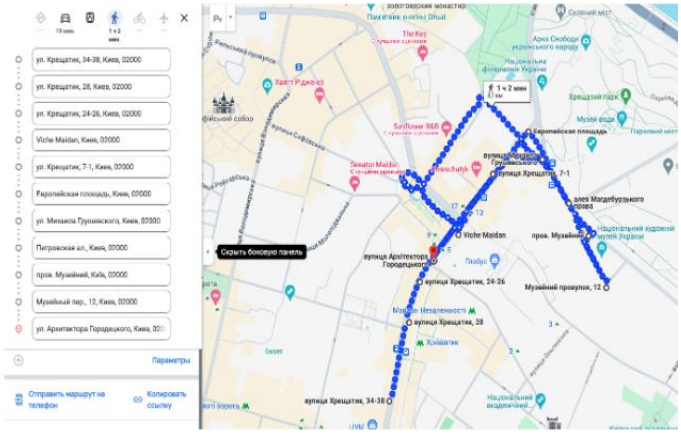
Список маршрутів

10

ЕКРАННІ ФОРМИ



Створення нового маршруту



Імпортований маршрут

11

ДЕМОНСТРАЦІЯ ВИКОРИСТАННЯ ЗАСТОСУНКУ

12

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Пустовіт Є.Ю. Розробка веб-застосунку для планування туристичних маршрутів з використанням інтерактивної карти. «Сучасні інтелектуальні інформаційні технології в науці та освіті»: матеріали всеукраїнської науково-практичної конференції. 15.05.2024, ДУІКТ, м. Київ. Подано до друку.
2. Пустовіт Є.Ю. Розробка ігрових застосунків для освітніх цілей з використанням штучного інтелекту: Матеріали всеукраїнській науково-практичній конференції «Сучасні інтелектуальні інформаційні технології в науці та освіті». 15.05.2024, ДУІКТ, м. Київ. Збірник тез. С. 261

13

ВИСНОВКИ

1. Проаналізовано існуючі програмні рішення для відображення та розробки туристичних маршрутів, виявлено їх переваги та недоліки.
2. Визначено функціональні та нефункціональні вимоги до Web-застосунку.
3. Спроектовано архітектуру та структуру Web-застосунку, враховуючи обрані технології та інструменти.
4. Розроблено серверну та клієнтську частини Web-застосунку з використанням мови JavaScript та бібліотеки React.
5. Інтегровано картографічний сервіс для відображення інтерактивної карти та реалізовано функціонал для створення, редагування та збереження туристичних маршрутів.
6. Проведено тестування Web-застосунку.

14

ДОДАТОК Б. ЛІСТИНГИ ОСНОВНИХ МОДУЛІВ

Програмний код App.js.

```
import React from 'react';
import { BrowserRouter as Router, Routes, Route, Link } from 'react-router-dom';
import MapPage from './components/MapPage';
import RouteCreatePage from './components/RouteCreatePage';
import RouteListPage from './components/RouteListPage';
import './App.css';

function App() {
  return (
    <Router>
      <div className="container">
        <header>
          <h1>Туристичні маршрути</h1>
        </header>
        <nav>
          <ul>
            <li>
              <Link to="/">Карта</Link>
            </li>
            <li>
                <Link to="/create">Створити маршрут</Link>
            </li>
            <li>
                <Link to="/list">Список маршрутів</Link>
            </li>
          </ul>
        </nav>

        <Routes>
           <Route path="/" element={<MapPage />} />
           <Route path="/create" element={<RouteCreatePage />} />
           <Route path="/list" element={<RouteListPage />} />
        </Routes>
      </div>
    </Router>
  );
}

export default App;
```

Програмний код MapPage.js.

```
import React from 'react';
import { MapContainer, TileLayer } from 'react-leaflet';
import 'leaflet/dist/leaflet.css';

function MapPage() {
  return (
    <MapContainer center={[50.4501, 30.5234]} zoom={13} style={{ height: '400px' }}>
      <TileLayer
        url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png" />
    </MapContainer>
  );
}

export default MapPage;
```

Програмний код RouteCreatePage.js.

```
import React, { useState, useRef } from 'react';
import { useNavigate } from 'react-router-dom';
import { MapContainer, TileLayer, FeatureGroup, useMapEvents } from 'react-leaflet';
import { EditControl } from 'react-leaflet-draw';
import 'leaflet/dist/leaflet.css';
import 'leaflet-draw/dist/leaflet.draw.css';

function RouteCreatePage() {
  const [name, setName] = useState('');
  const [description, setDescription] = useState('');
  const [routePath, setRoutePath] = useState([]);
  const mapRef = useRef(null);
  const navigate = useNavigate();

  const handleSubmit = async (e) => {
    e.preventDefault();
    const password = prompt('Введіть пароль адміністратора:');
    if (password === 'password') {
      const map = mapRef.current;
      const center = map.getCenter();
      const zoom = map.getZoom();
      const newRoute = {
        name,
        description,
        path: routePath,
        center: [center.lat, center.lng],
        zoom,
      };
      await saveRoute(newRoute);
```

```
    navigate('/list');
  } else {
    alert('Неправильний пароль. Маршрут не буде створено.');
  }
};
  const saveRoute = async (route) => {
    try {
              const response = await fetch('/api/routes', {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json',
        },
        body: JSON.stringify(route),
      });

      if (!response.ok) {
         throw new Error('Помилка збереження маршруту');
      }
    } catch (error) {
          console.error('Помилка збереження маршруту:', error);

    }
  };

  const handleCreated = (e) => {
    const layer = e.layer;
              const newRoutePath = layer.getLatLngs().map(latLng => [latLng.lat, latLng.lng]);
    setRoutePath(newRoutePath);
  };

  const MapEvents = () => {
    useMapEvents({
      moveend: () => {
        const map = mapRef.current;
        if (map) {
          const center = map.getCenter();
          const zoom = map.getZoom();
           console.log('Map center:', center, 'Zoom:', zoom);
        }
      },
    });
    return null;
  };

  return (
    <div>
      <h2>Створити новий маршрут</h2>
      <form onSubmit={handleSubmit}>
        <div>
          <label>Назва маршруту:</label>
          <input
            type="text"
            value={name}
                                        onChange={(e) => setName(e.target.value)}
          />
        </div>
        <div>
          <label>Опис маршруту:</label>
          <textarea
            value={description}
                                        onChange={(e) => setDescription(e.target.value)}
          ></textarea>
        </div>
          <div style={{ height: '400px', marginBottom: '20px' }}>
                        <MapContainer ref={mapRef} center={[50.4501, 30.5234]} zoom={13} style={{ height: '100%' }}>
            <MapEvents />
            <FeatureGroup>
              <EditControl
                position="topright"
                onCreated={handleCreated}
                draw={{
                  polygon: false,
                  polyline: true,
                  rectangle: false,
                  circle: false,
                  marker: false,
                  circlemarker: false,
                }}
              />
                                        <TileLayer url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png" />
            </FeatureGroup>
          </MapContainer>
        </div>
                                        <button type="submit">Створити</button>
      </form>
    </div>
  );
}
export default RouteCreatePage;
```

Програмний код RouteListPage.js.

```
import React, { useState, useEffect } from 'react';
import { MapContainer, TileLayer, Polyline } from 'react-leaflet';
import 'leaflet/dist/leaflet.css';

function RouteListPage() {
  const [routes, setRoutes] = useState([]);

  useEffect(() => {
    fetchRoutes();
  }, []);

  const fetchRoutes = async () => {
```

```
    try {
            const    response    =    await
fetch('/api/routes');
      if (response.ok) {
        const data = await response.json();
        setRoutes(data);
      } else {
         throw new Error('Помилка отримання
маршрутів');
      }
    } catch (error) {
        console.error('Помилка   отримання
маршрутів:', error);
    }
  };

  const handleImport = (route) => {
    const waypoints = route.path
      .slice(1, -1)
     .map(point => `${point[0]},${point[1]}`)
      .join('|');

            const      googleMapsUrl      =
`https://www.google.com/maps/dir/?api=1&orig
in=${route.path[0][0]},${route.path[0][1]}&d
estination=${route.path[route.path.length   -
1][0]},${route.path[route.path.length       -
1][1]}&waypoints=${waypoints}&travelmode=wal
king`;

    window.open(googleMapsUrl, '_blank');
  };

const handleDelete = async (id) => {
  const  password  =  prompt('Введіть  пароль
адміністратора:');
  if (password === 'password') {
    try {
      const      response      =      await
fetch(`/api/routes/${id}`, {
        method: 'DELETE',
      });

      if (response.ok) {
        fetchRoutes();
      } else {
        throw  new  Error('Помилка  видалення
маршруту');
      }
    } catch (error) {
      console.error('Помилка        видалення
маршруту:', error);
    }
  } else {
    alert('Неправильний  пароль.  Маршрут  не
буде видалено.');
  }
};
  return (
    <div>
      <h2>Список маршрутів</h2>
                                        <ul>
                                          {routes.map((route) => (
                                            <li key={route.id}>
                                              <h3>{route.name}</h3>
                                              <p>{route.description}</p>
                                                <div style={{ height: '200px',
                                    marginBottom: '20px' }}>
                                                                        <MapContainer
                                    center={route.center}       zoom={route.zoom}
                                    style={{ height: '100%' }}>
                                                                          <TileLayer
                                    url="https://{s}.tile.openstreetmap.org/{z}/
                                    {x}/{y}.png" />
                                                                {route.path && <Polyline
                                    positions={route.path} />}
                                                  </MapContainer>
                                              </div>
                                                    <button   onClick={()   =>
                                    handleImport(route)}>Імпортувати</button>
                                                    <button   onClick={()   =>
                                    handleDelete(route.id)}>Видалити</button>
                                                  </li>
                                          ))}
                                        </ul>
                                      </div>
                                    );
                                    }

                                    export default RouteListPage;
```