КВАЛІФІКАЦІЙНА РОБОТА

Група ІПЗс-20
Белей М.-В. А.

2024

ЗВО УНІВЕРСИТЕТ КОРОЛЯ ДАНИЛА

Факультет суспільних та прикладних наук

Кафедра інформаційних технологій

на правах рукопису

Белей Мар'ян-Віталій Андрійович

УДК 004.4

Розробка сервісу з інтерактивною динамічною картою для велосипедистів

Спеціальність 121 – «Інженерія програмного забезпечення»

Кваліфікаційна робота на здобуття кваліфікації бакалавр

Нормоконтроль	Студент
_____Стисло О.В.	_____Белей М-В. А.
(підпис, дата, розшифрування підпису)	(підпис, дата, розшифрування підпису)
Допускається до захисту Завідувач кафедри	Керівник роботи
_____к.т.н., доц. Ващишак С.П.	_____ к.т.н., доц. Ващишак С.П.
(підпис, дата, розшифрування підпису)	(підпис, дата, розшифрування підпису)

Івано-Франківськ – 2024

ЗВО УНІВЕРСИТЕТ КОРОЛЯ ДАНИЛА

Факультет суспільних та прикладних наук

Кафедра інформаційних технологій

Освітній ступінь: «бакалавр»

Спеціальність: 121 «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____

«____» _____ 2024 року

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Белею Мар'яну-Віталію Андрійовичу

(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи:
Розробка сервісу з інтерактивною динамічною картою для велосипедистів

керівник роботи:
Ващишак Сергій Петрович, к.т.н., доц. каф. ІТ

затверджена наказом вищого навчального закладу від « 12 » березня 2024 року № 19/1

2. Термін подання студентом роботи 05.06.2024

3. Вихідні дані роботи: багатофункціональний сервіс для роботи з картою

4. Зміст кваліфікаційної роботи (перелік питань, які потрібно розробити)
   1. Аналіз існуючих рішень у сфері сервісів та додатків для велосипедистів
   2. Обґрунтування вибору технологій та платформи розробки
   3. Розробка семантичної моделі та архітектури сервісу
   4. Реалізація та тестування сервісу

5. Дата видачі завдання 14.03.2024

КОНСУЛЬТАНТИ РОЗДІЛІВ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Розділ	Консультант (прізвище, ініціали та посада)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1.	Аналіз існуючих рішень у сфері сервісів та додатків для велосипедистів	16.03.2024	Виконано
2.	Обґрунтування вибору технологій та платформи розробки	20.03.2024	Виконано
3.	Розробка семантичної моделі та архітектури сервісу	26.03.2024	Виконано
4.	Реалізація серверної частини	08.04.2024	Виконано
5.	Реалізація клієнтської частини	15.04.2024	Виконано
6.	Тестування та оптимізація сервісу	25.04.2024	Виконано
7.	Формування висновків та рекомендацій	05.05.2024	Виконано
8.	Оформлення пояснювальної записки	08.05.2024	Виконано
9.	Оформлення графічного матеріалу та підготовка до захисту роботи	10.05.2024	Виконано

Студент _____ Белей М.-В. А.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Ващишак С.П.
(підпис) (прізвище та ініціали)

Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

Сторінка	Опис графічного матеріалу	Сторінка	Опис графічного матеріалу
14	Вигляд додатку для велосипедистів Strava	25	Вигляд сервісу для велосипедистів Bikemap
17	Вигляд сервісу для велоподорожей Komoot	27	Вигляд сервісу для велосипедистів міста Брісбен Cycling Brisbane
19	Вигляд веб-сервісу для велосипедистів RidewithGPS	45	Діаграма класів архітектури бази даних
21	Вигляд картографічного сервісу із увімкненим велосипедним режимом Google Maps	100	Приклад вихідних даних при спробі хостингу сервісу за допомогою ngrok
23	Вигляд додатку OpenStreetMap		

АНОТАЦІЯ

У кваліфікаційній роботі представлено розробку онлайн сервісу з динамічною та інтерактивною картою для велосипедистів. Сервіс дозволяє користувачам планувати маршрути, відзначати на карті місця відпочинку, велосипедну інфраструктуру та ділитися інформацією з іншими учасниками. Робота включає аналіз існуючих рішень, обґрунтування вибору технологій, опис процесу розробки та практичної реалізації. Використано сучасний стек технологій: Ruby on Rails, React.js, MongoDB та інші.

Результатом є зручний інструмент для спільноти велосипедистів, який сприяє популяризації здорового способу життя та розвитку велоінфраструктури.

КЛЮЧОВІ СЛОВА: ВЕЛОСИПЕДНИЙ СЕРВІС, ІНТЕРАКТИВНА КАРТА, RUBY ON RAILS, REACT.JS, MONGODB, GRAPHQL, ГЕОПРОСТОРОВІ ДАНІ, ВЕЛОІНФРАСТРУКТУРА, ПЛАНУВАННЯ МАРШРУТІВ, СПІЛЬНОТА ВЕЛОСИПЕДИСТІВ, MATERIAL-UI, REDUX TOOLKIT.

SUMMARY

The thesis presents the development of an online service with a dynamic and interactive map for cyclists. The service allows users to plan routes, mark resting places, bicycle infrastructure on the map, and share information with other participants. The work includes an analysis of existing solutions, justification for the choice of technologies, a description of the development process, and practical implementation. A modern technology stack was utilized: Ruby on Rails, React.js, MongoDB, and others.

The result is a convenient tool for the cycling community that promotes a healthy lifestyle and the development of cycling infrastructure.

KEY WORDS: BICYCLE SERVICE, INTERACTIVE MAP, RUBY ON RAILS, REACT.JS, MONGODB, GRAPHQL, GEOSPATIAL DATA, BICYCLE INFRASTRUCTURE, ROUTE PLANNING, CYCLING COMMUNITY, MATERIAL-UI, REDUX TOOLKIT.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

MVC – (Model-View-Controller)

DRY – (Don't Repeat Yourself)

REST/RESTful API – (Representational State Transfer / RESTful Application Programming Interface)

ORM – (Object-Relational Mapping)

NoSQL – (Non-Relational Database)

DOM – (Document Object Model)

JSX – (JavaScript XML)

GPS – (Global Positioning System)

API – (Application Programming Interface)

UI – (User Interface)

UX – (User Experience)

HTML – (Hypertext Markup Language)

CSS – (Cascading Style Sheets)

ВСТУП

Актуальність теми. У сучасному світі інтерактивні карти та географічно-орієнтовані онлайн-сервіси стали невід'ємною частиною нашого повсякденного життя, допомагаючи орієнтуватися на місцевості, планувати маршрути, знаходити цікаві місця та об'єкти інфраструктури. Проте створення зручних і функціональних картографічних сервісів часто ускладнюється через складність збору, обробки та візуалізації географічних даних з різних джерел у єдиному інтерфейсі, необхідність забезпечення високої точності, актуальності та деталізації картографічної інформації, потребу у розробці інтуїтивного та зрозумілого для користувачів інтерфейсу взаємодії з картою.

Крім того, важливо забезпечити можливість динамічного оновлення даних на карті та вносу користувацького контенту, інтеграцію різноманітних шарів інформації (рельєф, транспортні маршрути, інфраструктурні об'єкти тощо) на одній карті, а також вимоги до високої продуктивності та масштабованості картографічних сервісів для забезпечення безперебійної роботи з великою кількістю користувачів.

Тому розробка сервісу з динамічною та інтерактивною картою, зокрема для велосипедистів, є актуальною задачею, оскільки дозволяє ефективно вирішити вищезазначені проблеми та забезпечити користувачів зручним інструментом для планування маршрутів, пошуку вело-інфраструктури та обміну досвідом.

Мета роботи. Метою кваліфікаційної роботи є розробка онлайн сервісу з динамічною картою, який забезпечує обмін інформацією між учасниками дорожнього руху (велосипедистами, користувачами електросамокатів, водіїв).

Об’єкт роботи. Об'єктом дослідження є процес створення онлайн сервісу для обміну інформацією між велосипедистами за допомогою сучасного стеку веб-технологій, включаючи Ruby on Rails, React.js, MongoDB та інших

інструментів, що забезпечують розробку динамічної та інтерактивної картографічної платформи.

Предмет роботи. Предметом дослідження є методи та засоби програмної інженерії, зокрема об'єктно-орієнтоване програмування, Agile-методології розробки програмного забезпечення, шаблони проєктування, технології веб-розробки, системи керування базами даних, а також інструменти для створення динамічної та інтерактивної картографічної платформи для обміну інформацією між велосипедистами.

Завдання роботи. Для досягнення мети створення сервісу з динамічною та інтерактивною картою для велосипедистів необхідно виконати наступні завдання. По-перше, слід провести аналіз існуючих онлайн-сервісів та додатків для велосипедистів, їх функціоналу та недоліків, щоб визначити прогалини та можливості для вдосконалення. Потім потрібно розробити концепцію інтерактивного сервісу з картою для велосипедистів, визначивши основні функції та можливості, які будуть реалізовані. Наступним кроком є вибір відповідних технологій та інструментів для розробки веб-сервісу та мобільного додатку, враховуючи вимоги до продуктивності, масштабованості, ефективності та зручності використання.

Далі необхідно спроектувати базу даних для зберігання інформації про маршрути, велосипедну інфраструктуру та відгуки користувачів, забезпечуючи ефективне зберігання та доступ до даних. Після цього слід розробити зручний та інтуїтивний інтерфейс користувача для веб-сервісу та мобільного додатку, враховуючи принципи юзабіліті та досвід користувача. Наступним завданням є реалізація функціоналу для додавання та редагування маршрутів, місць відпочинку, велосипедної інфраструктури на інтерактивній карті, що дозволить користувачам активно взаємодіяти з картою та поділитися інформацією.

Крім того, необхідно впровадити систему реєстрації та авторизації користувачів, а також надати можливість ділитися маршрутами та залишати відгуки, щоб сприяти створенню активної спільноти велосипедистів. Після завершення розробки важливо провести тестування сервісу на різних пристроях

та браузерах для виявлення та виправлення помилок, забезпечуючи стабільну роботу додатку. Нарешті, слід оцінити ефективність та зручність використання сервісу шляхом опитування цільової аудиторії, щоб отримати зворотний зв'язок та ідеї для подальшого вдосконалення.

Методи дослідження. У ході дослідження для вирішення поставлених завдань буде використано експериментальний метод, зокрема експериментальне тестування розробленого сервісу на різних пристроях та в різних умовах. Це дозволить виявити можливі проблеми та недоліки в роботі додатку, а також перевірити його сумісність і стабільність в реальних умовах експлуатації.

Крім того, для оптимізації алгоритмів пошуку маршрутів та відображення інформації на інтерактивній карті буде застосовано математичне моделювання та аналіз даних. Це передбачає створення математичних моделей, які описують процеси пошуку оптимальних маршрутів, візуалізації даних на карті та взаємодії користувачів з картографічною інформацією. Аналіз цих моделей та обробка відповідних даних дозволить удосконалити алгоритми та забезпечити ефективну роботу сервісу.

Практичне значення одержаних результатів. Розроблений сервіс з динамічною та інтерактивною картою для велосипедистів матиме широке практичне застосування. Основною сферою використання буде планування велосипедних маршрутів та навігація як для повсякденних поїздок містом, так і для туристичних або спортивних цілей. Сервіс дозволить користувачам легко знаходити оптимальні маршрути з урахуванням наявної велоінфраструктури, місць відпочинку та відгуків інших велосипедистів про ті чи інші ділянки. Це підвищить безпеку пересування на велосипеді та зробить такий вид транспорту більш зручним і привабливим.

Крім того, сервіс стане корисним інструментом для популяризації здорового способу життя та екологічно чистих видів транспорту. Завдяки можливості ділитися маршрутами та досвідом, буде створено онлайн-спільноту велосипедистів, що сприятиме обміну знаннями та підвищенню інтересу до велоруху.

Місцева влада також зможе використовувати сервіс для аналізу потреб велосипедистів та планування розвитку велоінфраструктури на основі реальних даних про популярні маршрути, проблемні ділянки та запити користувачів. Це дозволить ефективніше розподіляти ресурси для облаштування велодоріжок, парковок та інших об'єктів.

Власники закладів, орієнтованих на велосипедистів, зможуть відзначати свої локації на карті, рекламуючи свої послуги та залучаючи нових клієнтів.

Після успішного впровадження, розроблений сервіс буде постійно вдосконалюватися та розширюватися відповідно до запитів користувачів. Планується інтеграція з іншими додатками та розширення функціоналу, такого як можливість створення та приєднання до велоклубів, організація спільних поїздок тощо.

Апробація результатів дослідження. Матеріали кваліфікаційної роботи були представлені на XI Міжнародній науковій конференції «Студентські наукові дискусії поза форматом», яка відбулася 11 квітня 2024 року в Університеті Короля Данила.

Структура роботи. Розділи – 3. Обсяг основної частини – 90 сторінок. Список використаних джерел – 20.

РОЗДІЛ 1. АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ

Для аналізу існуючих рішень було розглянуто кілька популярних сервісів та додатків для велосипедистів.

1.1 Аналіз сервісу Strava

Strava – один з найпопулярніших додатків для відстеження активностей, включаючи їзду на велосипеді (рис. 1.1). Цей додаток дозволяє записувати маршрути, відстежувати статистику, ділитися досягненнями з друзями та брати участь у різноманітних викликах. Strava також має функцію відображення маршрутів на карті, проте вона не є інтерактивною та не дозволяє додавати інформацію про інфраструктуру чи місця відпочинку.

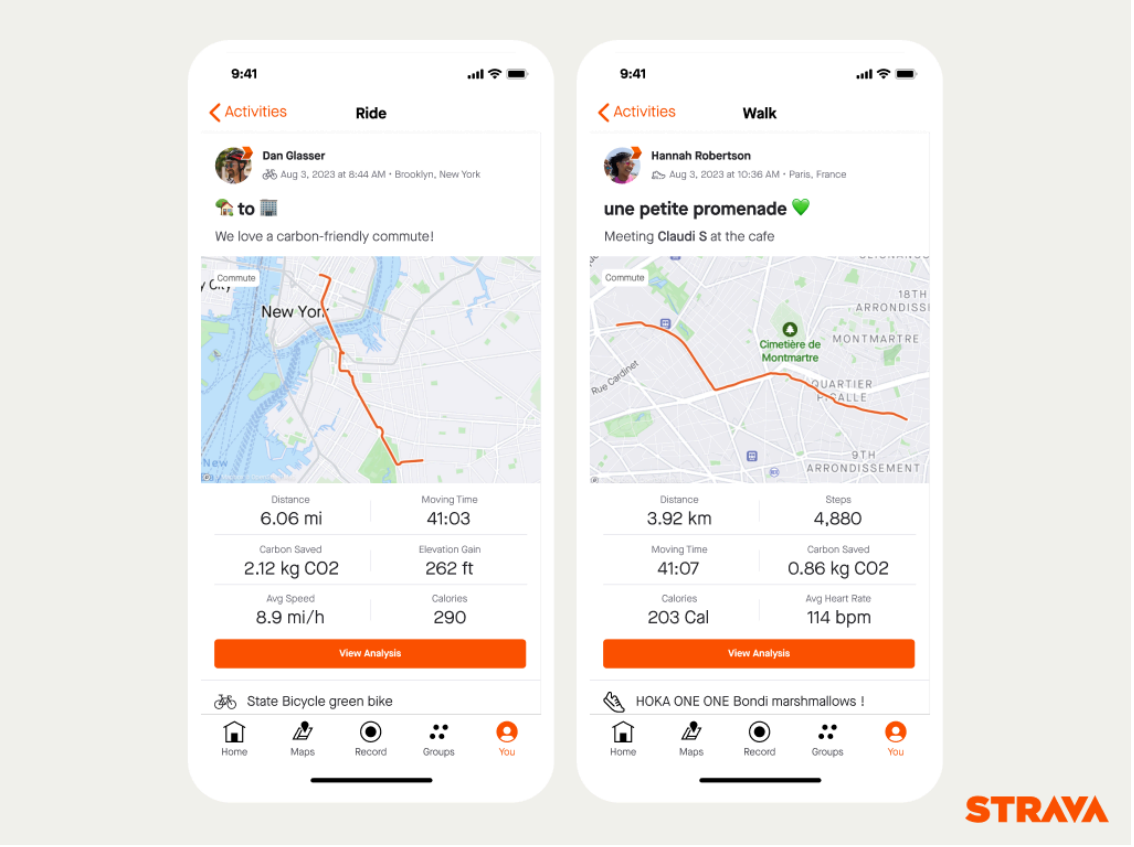


Рисунок 1.1 – Вигляд додатку Strava

Переваги Strava. Strava пропонує зручні можливості для відстеження велосипедних активностей.

Додаток дозволяє легко записувати та відстежувати поїздки, надаючи детальну інформацію про дистанцію, швидкість, час та висоту. Ці дані є корисними для моніторингу прогресу та встановлення персональних цілей, що мотивує велосипедистів до покращення своїх результатів.

Однією з ключових переваг Strava є її потужна соціальна складова. Додаток створює мотивуюче середовище для велосипедистів, дозволяючи їм ділитися своїми досягненнями, змагатися з друзями та брати участь у спільних викликах. Ця взаємодія з іншими користувачами додає елемент змагання та заохочує до активнішого використання велосипеда.

Strava також вирізняється своєю сумісністю з різними GPS-пристроями та фітнес-трекерами. Це дозволяє легко синхронізувати дані з інших джерел, забезпечуючи централізоване місце для зберігання та аналізу всієї інформації про велосипедні активності.

Цікавою особливістю Strava є автоматичне виявлення популярних сегментів маршрутів. Додаток дозволяє велосипедистам змагатися за найкращий час на цих ділянках, що створює додатковий елемент змагання та мотивації для покращення власних результатів.

Недоліки Strava. незважаючи на те, що Strava відображає маршрути на карті, функціональність карт у додатку є досить базовою та обмеженою. Користувачі не мають можливості інтерактивно додавати інформацію про велосипедну інфраструктуру, місця відпочинку чи інші важливі для велосипедистів деталі. Ця обмеженість інтерактивності карт може зменшувати корисність додатку для планування маршрутів та навігації.

Strava більше орієнтована на спортивну складову велоспорту, зосереджуючись на відстеженні показників та змаганнях. Через це додатку може бракувати функцій, які були б корисними для міських велосипедистів, які використовують велосипед для щоденних утилітарних поїздок містом. Для цієї категорії користувачів Strava може бути менш релевантною.

Деякі користувачі можуть мати застереження щодо конфіденційності, оскільки Strava передбачає передачу даних про поїздки та місцезнаходження. Хоча додаток має налаштування конфіденційності, вони можуть бути недостатньо гнучкими для користувачів, які прагнуть більшого контролю над своїми даними.

Частина розширених функцій Strava доступна лише з платною підпискою, що може обмежувати можливості користувачів, які не готові оплачувати додатковий функціонал. Це може стати перешкодою для повноцінного використання всіх можливостей додатку для деяких велосипедистів.

Загалом, Strava є потужним інструментом для спортивно орієнтованих велосипедистів, пропонуючи відстеження активності та соціальні функції. Проте, додаток може мати обмеження для велосипедистів, які шукають більш інтерактивні та інформативні карти або орієнтовані на утилітарне використання велосипеда в місті.

1.2 Аналіз сервісу Komoot

Komoot – це додаток для планування маршрутів та навігації для велосипедистів, пішоходів та мандрівників (рис. 1.2). Він пропонує детальні карти з різними шарами, такими як рельєф, типи доріг та покриття. Користувачі можуть планувати маршрути, враховуючи різні критерії, проте додаток не має можливості додавати інформацію про інфраструктуру чи місця відпочинку безпосередньо на карту. Незважаючи на ці обмеження, Komoot залишається популярним вибором серед любителів активного відпочинку завдяки зручному інтерфейсу та можливості ділитися своїми маршрутами з іншими користувачами. Додаток постійно вдосконалюється, і в майбутньому можна очікувати появи нових функцій, які зроблять планування подорожей ще більш комфортним та інформативним.

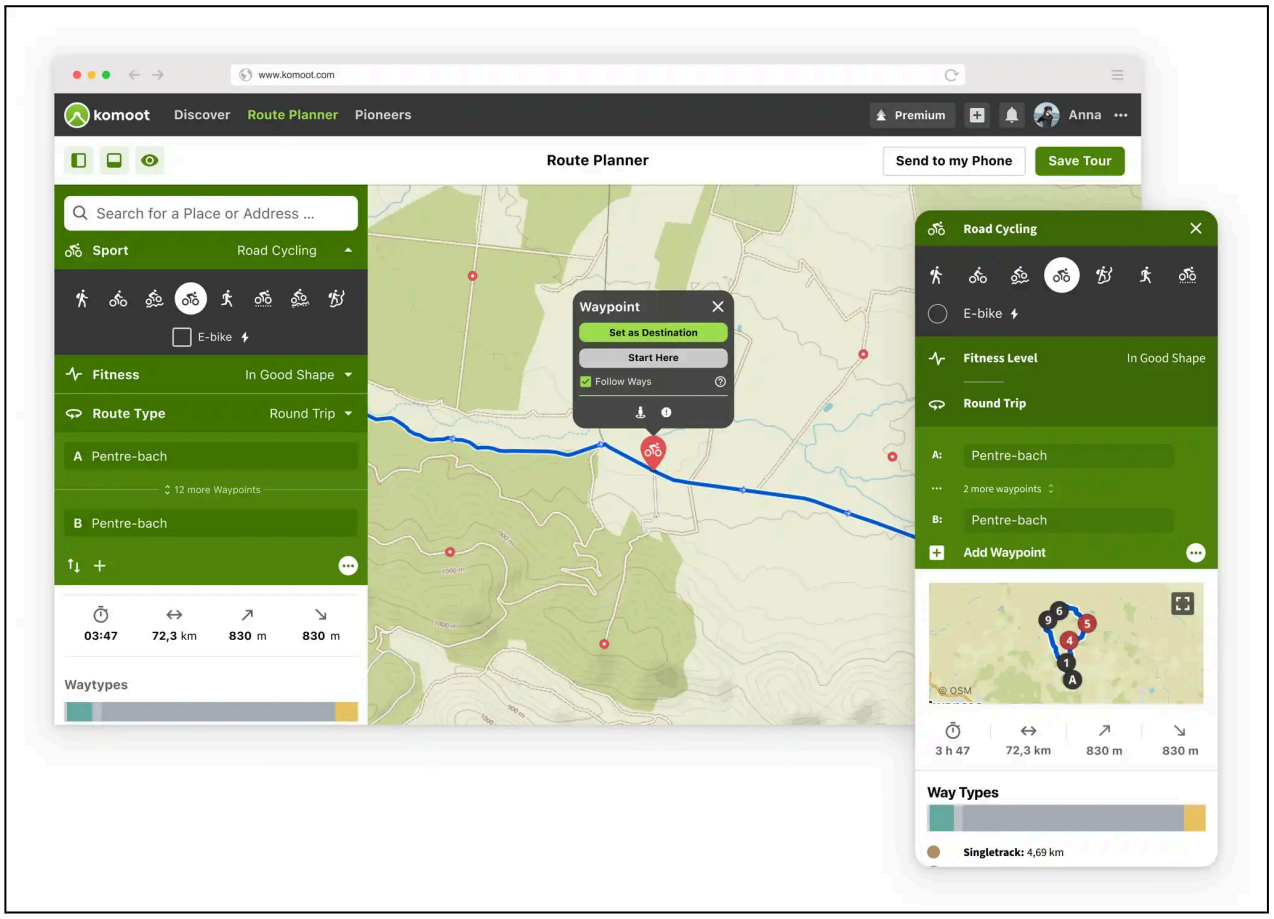


Рисунок 1.2 – Вигляд сервісу Komoot

Переваги Komoot. Komoot вирізняється своїми детальними картами, які містять різноманітні шари, такі як рельєф, типи доріг та покриття. Ця інформація є надзвичайно корисною для велосипедистів, оскільки дозволяє краще планувати маршрути та уникати несподіванок на дорозі. Завдяки цим деталям користувачі можуть обирати оптимальні шляхи, враховуючи свої преференції та рівень підготовки.

Однією з ключових переваг Komoot є можливість зручного планування маршрутів. Додаток дозволяє користувачам легко створювати персоналізовані маршрути, враховуючи різні критерії, такі як складність, тип місцевості та визначні пам'ятки. Це особливо корисно для велосипедистів, які люблять досліджувати нові місця та шукають цікаві й унікальні маршрути.

Komoot також пропонує зручну навігацію під час поїздки. Додаток надає покрокові інструкції, допомагаючи велосипедистам слідувати запланованому

маршруту та не збитися зі шляху. Ця функція є особливо корисною для тих, хто подорожує незнайомими місцевостями або складними маршрутами.

Ще однією перевагою Komoot є його активна спільнота користувачів. Велосипедисти можуть ділитися своїми маршрутами та досвідом з іншими, а також отримувати рекомендації щодо цікавих місць та маршрутів від інших членів спільноти. Це створює можливості для обміну знаннями та відкриття нових місць для велосипедних подорожей.

Недоліки Komoot. Незважаючи на те, що Komoot пропонує детальні карти, інтерактивність цих карт є обмеженою. Користувачі не мають можливості самостійно додавати інформацію про велосипедну інфраструктуру, місця відпочинку чи інші важливі для велосипедистів деталі безпосередньо на карту. Ця обмеженість може зменшувати корисність додатку для деяких користувачів, які хотіли б персоналізувати карти відповідно до своїх потреб.

Komoot більше орієнтований на планування туристичних та рекреаційних поїздок, а не на щоденне використання велосипеда в міському середовищі. Через це деякі функції, які були б корисними для міських велосипедистів, можуть бути відсутні або менш розвинені в додатку.

Для повноцінного використання функцій Komoot, таких як завантаження карт та навігація, потрібне стабільне інтернет-з'єднання. Це може створювати незручності у місцях з обмеженим покриттям мобільної мережі, особливо під час подорожей у віддалених районах.

Хоча базовий функціонал Komoot є безкоштовним, доступ до детальних карт для деяких регіонів може вимагати додаткової оплати. Це може обмежувати можливості користувачів, які не готові платити за додаткові карти, особливо якщо вони планують велосипедні подорожі в різних місцях.

Загалом, Komoot є корисним інструментом для велосипедистів, які люблять планувати маршрути та досліджувати нові місця. Додаток пропонує детальні карти та функції навігації, що робить його особливо привабливим для туристичних поїздок. Проте, Komoot може мати обмеження для велосипедистів,

які шукають більш інтерактивні карти або орієнтовані на щоденне використання велосипеда в місті.

1.3 Аналіз сервісу RidewithGPS

RidewithGPS – це веб-сервіс та мобільний додаток для планування велосипедних маршрутів (рис. 1.3). Він пропонує детальні карти з різними шарами, такими як рельєф, типи доріг та покриття. Користувачі можуть створювати, редагувати та ділитися маршрутами, проте функціональність додавання інформації про інфраструктуру чи місця відпочинку обмежена.

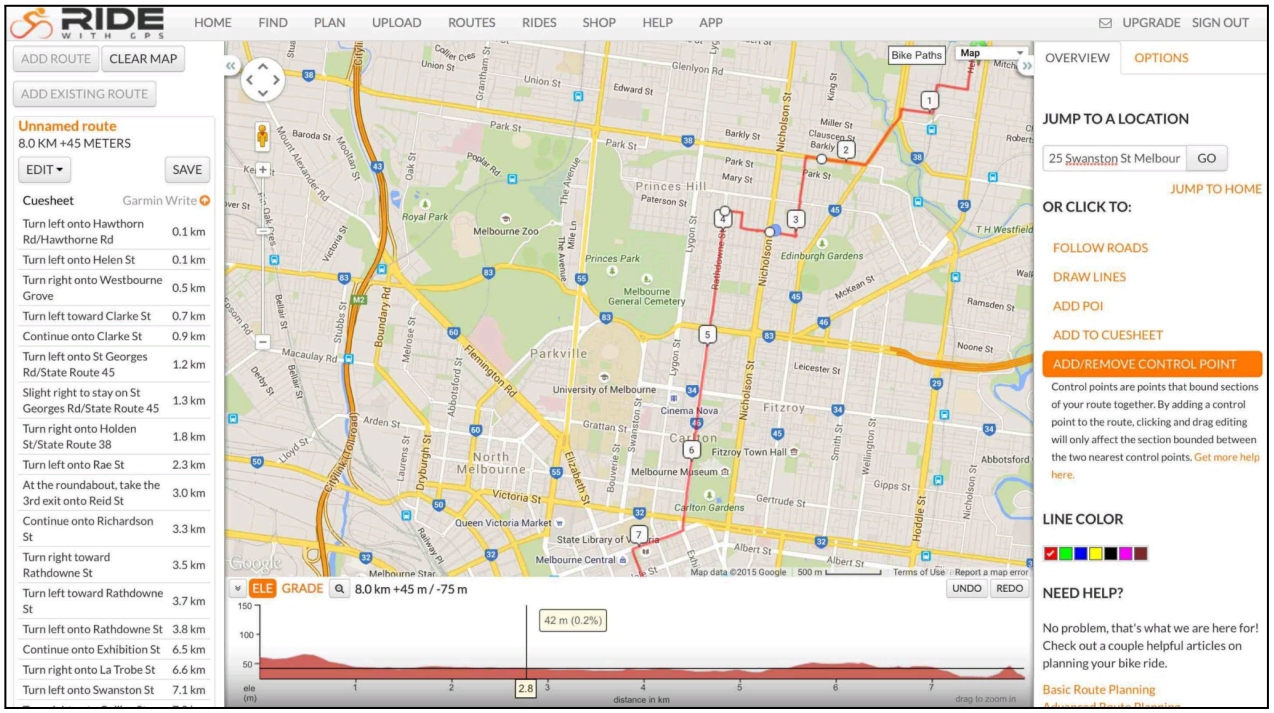


Рисунок 1.3 – Вигляд сервісу RidewithGPS

Сервіс RidewithGPS має низку переваг для велосипедистів. Однією з головних переваг є наявність детальних карт з різними шарами, такими як рельєф, типи доріг та покриття. Це дозволяє користувачам ретельно планувати свої маршрути, враховуючи особливості місцевості та уникаючи несподіванок під час поїздки.

Крім того, RidewithGPS надає можливість легко створювати та редагувати маршрути безпосередньо на карті, що дає велосипедистам гнучкість у плануванні поїздок відповідно до своїх преференцій та вимог. Сервіс також дозволяє ділитися створеними маршрутами з іншими користувачами, що сприяє обміну досвідом та пошуку нових цікавих маршрутів. Ще однією перевагою RidewithGPS є інтеграція з різними GPS-пристроями та велокомп'ютерами, що дозволяє легко синхронізувати маршрути та дані про поїздки.

Незважаючи на переваги, RidewithGPS має і деякі недоліки. Одним з них є обмежена функціональність щодо додавання інформації про інфраструктуру. Хоча сервіс дозволяє додавати деяку інформацію про об'єкти на маршруті, ця функціональність досить обмежена. Велосипедистам може бути складно знайти важливі об'єкти, такі як велопарковки, станції ремонту чи місця відпочинку.

Крім того, RidewithGPS більше орієнтований на спортивних велосипедистів та планування тренувальних маршрутів, а не на щоденне використання велосипеда в місті. Деякі функції, корисні для міських велосипедистів, можуть бути відсутні. Ще одним недоліком є необхідність інтернет-з'єднання для повноцінного використання функцій сервісу, таких як створення та редагування маршрутів. Це може бути незручно у місцях з обмеженим покриттям мобільної мережі. Також варто зазначити, що частина розширених функцій RidewithGPS, таких як детальні карти та деякі інструменти для планування маршрутів, доступні лише з платною підпискою, що може обмежувати можливості користувачів, які не готові або не мають змоги оплачувати додатковий функціонал.

Загалом, RidewithGPS є корисним інструментом для велосипедистів, які люблять планувати маршрути та ділитися ними з іншими. Сервіс пропонує детальні карти та функції для створення та редагування маршрутів, що робить його особливо привабливим для спортивних велосипедистів. Проте, RidewithGPS може мати обмеження для велосипедистів, які шукають більше інформації про міську інфраструктуру або орієнтовані на щоденне використання велосипеда в місті.

1.4 Аналіз сервісу Google Maps

Google Maps – це популярний картографічний сервіс, який також має функції планування маршрутів для велосипедистів (рис. 1.4). Проте він не спеціалізований на велосипедному русі та не має спеціальних функцій для додавання інформації про інфраструктуру чи місця відпочинку для велосипедистів на карті.

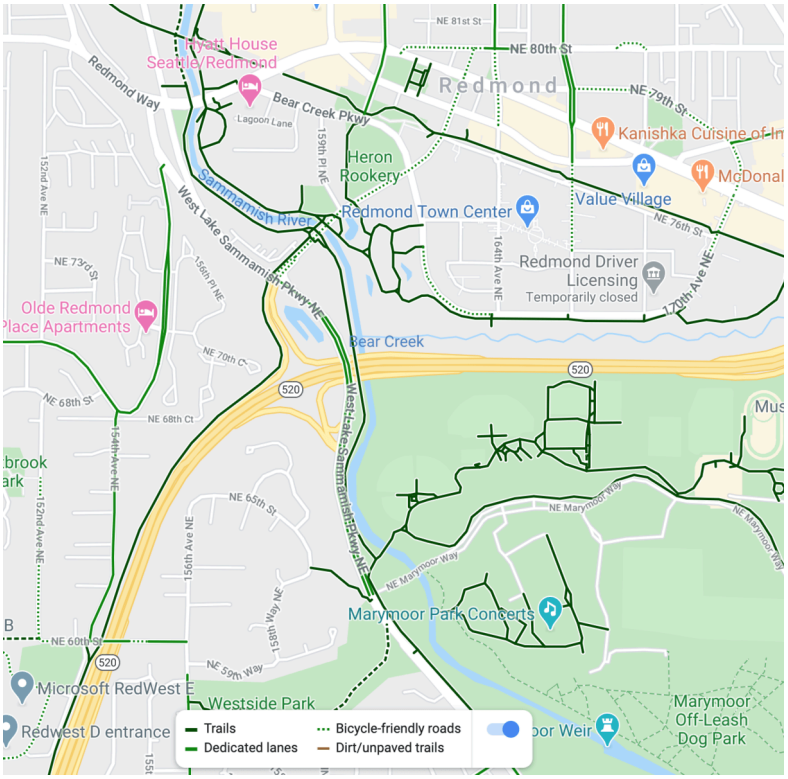


Рисунок 1.4 – Вигляд сервісу Google Maps

Серед переваг Google Maps можна відзначити широке покриття, адже сервіс пропонує карти для більшості регіонів світу, що дозволяє велосипедистам планувати маршрути майже будь-де.

Крім того, Google Maps є безкоштовним та легкодоступним на різних платформах, включаючи веб та мобільні пристрої, а багато користувачів вже знайомі з його інтерфейсом. Сервіс також пропонує покрокову навігацію для велосипедистів в реальному часі, враховуючи поточний трафік та умови на дорогах. Ще однією перевагою є інтеграція з іншими сервісами Google, що

дозволяє користувачам легко синхронізувати свої маршрути та дані з такими сервісами, як Google Calendar та Google Fit.

Незважаючи на переваги, Google Maps має і деякі недоліки для велосипедистів. Оскільки сервіс не є спеціалізованим для велосипедного руху, він може не мати деяких функцій, корисних для планування велосипедних маршрутів, таких як інформація про рельєф, типи покриття чи інфраструктуру.

Крім того, Google Maps не дозволяє користувачам додавати детальну інформацію про велосипедну інфраструктуру, таку як велопарковки, станції ремонту чи місця відпочинку, що може бути корисним для велосипедистів під час планування поїздок. Маршрути, запропоновані сервісом, можуть бути не завжди оптимальними для велосипедистів, оскільки алгоритм не завжди враховує специфічні преференції та вимоги велосипедистів, такі як уникнення жвавих доріг чи пошук мальовничих маршрутів. Ще одним недоліком є залежність від інтернет-з'єднання для повноцінного використання функцій Google Maps, таких як навігація в реальному часі, що може бути проблемою у місцях з обмеженим покриттям мобільної мережі.

Загалом, Google Maps є зручним та широкодоступним інструментом для планування велосипедних маршрутів, особливо завдяки широкому покриттю карт та зручності використання. Проте, сервіс може мати обмеження для велосипедистів, які шукають спеціалізовані функції, детальну інформацію про велосипедну інфраструктуру або оптимізовані маршрути. Для більш специфічних потреб велосипедистів можуть бути корисними спеціалізовані додатки та сервіси.

1.5 Аналіз сервісу OpenStreetMap

OpenStreetMap – це відкрита картографічна платформа, яка дозволяє користувачам редагувати карту та додавати різноманітні об'єкти, включаючи велосипедну інфраструктуру (рис. 1.5). Проте процес додавання та редагування

даних може бути складним для звичайних користувачів, а інтерфейс не завжди зручний для планування маршрутів.

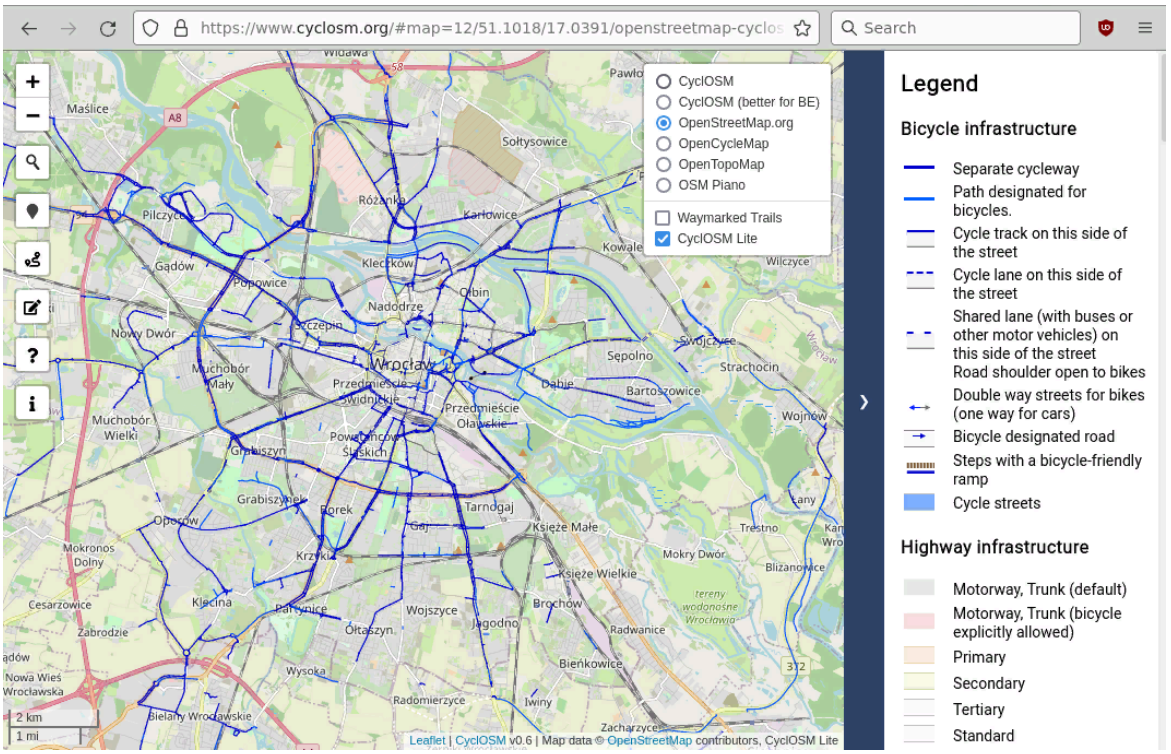


Рисунок 1.5 – Вигляд сервісу OpenStreetMap

Однією з головних переваг OpenStreetMap є її відкритість та колаборативність. Будь-хто може долучитися до створення та редагування карт, що дозволяє велосипедистам додавати інформацію про велосипедну інфраструктуру, яка може бути відсутня на інших картах. Завдяки великій спільноті редакторів, OSM часто має більш детальну інформацію про велодоріжки, велопарковки та інші об'єкти, важливі для велосипедистів, порівняно з комерційними картографічними сервісами.

Крім того, OSM є безкоштовним для використання, що робить його доступним для широкого кола велосипедистів. Ще однією перевагою є можливість завантаження даних OSM для використання в офлайн-режимі, що зручно для велопоїздок у місцях з обмеженим доступом до інтернету.

Незважаючи на переваги, OpenStreetMap має і деякі недоліки. Одним з них є складність процесу додавання та редагування даних для пересічних

користувачів через необхідність розуміння специфічних тегів та правил картографування. Хоча спільнота OSM активно працює над покращенням карт, у деяких регіонах інформація про велоінфраструктуру може бути неповною або застарілою. Базовий інтерфейс OSM не надає зручних інструментів для планування велосипедних маршрутів, таких як врахування типу покриття, ухилів та інтенсивності трафіку, тому для цього потрібно використовувати сторонні додатки, що базуються на даних OSM.

Крім того, оскільки дані в OSM додаються та редагуються волонтерами, існує ризик неточностей або помилок, особливо в менш популярних регіонах.

Підсумовуючи, OpenStreetMap є цінним ресурсом для велосипедистів завдяки відкритості, детальності та безкоштовності, але має певні недоліки, пов'язані зі складністю редагування, неповнотою даних та відсутністю зручного інтерфейсу для планування маршрутів.

1.6 Аналіз сервісу Bikemap

Bikemap – це спеціалізований додаток для велосипедистів, який пропонує детальні карти з різними шарами, включаючи рельєф, типи доріг та покриття (рис. 1.6). Користувачі можуть планувати маршрути, враховуючи різні критерії, такі як відстань, висота та складність. Додаток також дозволяє відзначати на карті певні місця, такі як пункти ремонту або цікаві локації. Проте функціонал додавання інформації про інфраструктуру для велосипедистів або місця відпочинку обмежений. Крім того, додаток не надає можливості ділитися маршрутами та відгуками з іншими користувачами, що могло б збагатити досвід велосипедистів. Це зменшує соціальну складову та обмежує можливості для взаємодії між любителями велоспорту. Також відсутня можливість отримувати актуальну інформацію про стан доріг, перекриття чи інші події, що могли б вплинути на безпеку та комфорт велосипедної поїздки. Загалом, Bikemap є корисним інструментом для планування велосипедних маршрутів, але має

обмежений функціонал порівняно з очікуваннями сучасних велосипедистів, які прагнуть більшої інтерактивності, соціальної взаємодії та актуальності даних.

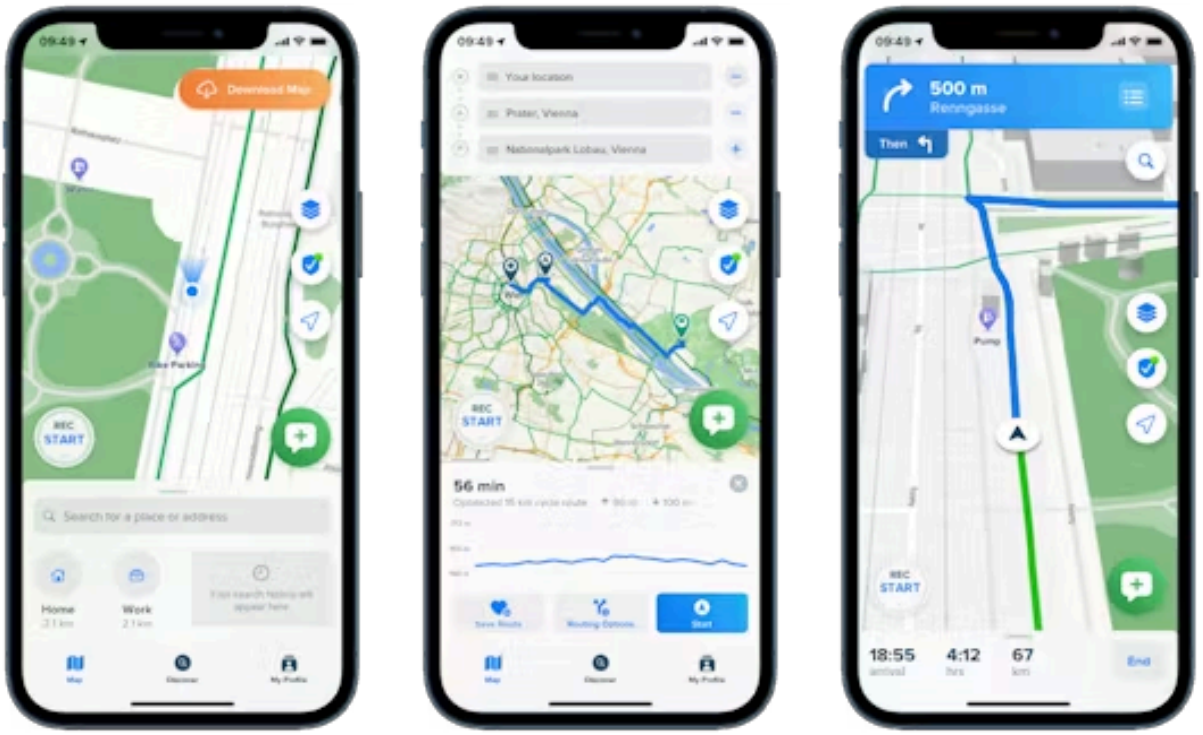


Рисунок 1.6 – Вигляд сервісу Bikemap

Сервіс має ряд переваг, які роблять його корисним для велосипедистів. По-перше, Bikemap спеціалізується саме на велосипедних маршрутах, що дозволяє краще враховувати потреби велосипедистів та надавати їм релевантну інформацію. Карти сервісу є досить деталізованими, відображаючи важливі для велосипедистів дані, такі як рельєф, типи доріг та покриття. Це допомагає користувачам краще планувати свої маршрути та уникати небажаних ділянок. Ще одна перевага – можливість планувати маршрути за різними критеріями, включаючи відстань, висоту, складність, що дозволяє підбирати оптимальні шляхи для різних рівнів підготовки та преференцій велосипедистів.

Додаток також дає змогу відзначати на карті важливі місця, як-от пункти ремонту або цікаві локації, що робить його більш інформативним та корисним. Важливо, що Bikemap доступний у вигляді мобільного додатку, що дозволяє велосипедистам користуватися ним безпосередньо під час поїздок.

Водночас, Bikemap має й певні недоліки. Зокрема, функціонал додавання детальної інформації про велосипедну інфраструктуру, таку як велопарковки, станції самообслуговування або місця відпочинку, є обмеженим порівняно з іншими сервісами, як-от OpenStreetMap. Якість та повнота даних у Bikemap також частково залежать від активності користувачів у додаванні та оновленні інформації, що може призводити до неповноти або неактуальності інформації в деяких регіонах.

Крім того, порівняно з іншими популярними сервісами для велосипедистів, спільнота користувачів Bikemap є меншою, що може вплинути на швидкість розвитку платформи.

Також існує ймовірність, що деякі функції Bikemap можуть бути доступні лише у платній версії, що може обмежувати можливості користувачів, які надають перевагу безкоштовним сервісам.

Отже, Bikemap є зручним та корисним додатком для велосипедистів, який пропонує деталізовані карти та можливості планування маршрутів, але має певні обмеження щодо додавання інформації про інфраструктуру та залежить від активності користувачів у наповненні контентом.

1.7 Аналіз сервісу Cycling Brisbane

Cycling Brisbane – це веб-сервіс та мобільний додаток, розроблений місцевою владою міста Брисбен, Австралія (рис. 1.7). Він пропонує карту з детальною інформацією про велосипедні доріжки, маршрути та інфраструктуру в місті. Користувачі можуть планувати маршрути, враховуючи наявні велосипедні доріжки та смуги.

Додаток також дозволяє користувачам повідомляти про проблеми на велосипедних маршрутах. Проте сервіс обмежений лише містом Брисбен і не дозволяє додавати інформацію про місця відпочинку чи ділитися власними маршрутами.

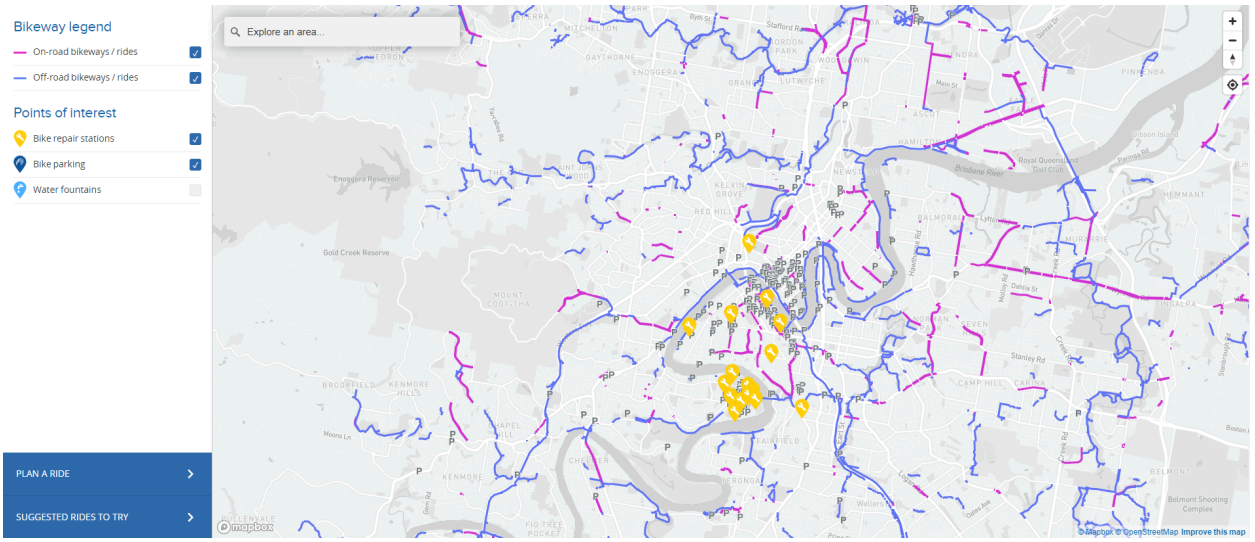


Рисунок 1.7 – Вигляд сервісу Cycling Brisbane

Однією з головних переваг Cycling Brisbane є його офіційна підтримка місцевою владою. Це забезпечує надійність, актуальність та якість інформації про велосипедну інфраструктуру міста. Сервіс надає вичерпні дані про велосипедні доріжки, маршрути та смуги, що допомагає велосипедистам зручно і безпечно пересуватися містом. Користувачі також можуть планувати свої маршрути з урахуванням наявних велодоріжок, що робить поїздки більш комфортними та ефективними.

Ще одна корисна функція – можливість повідомляти про проблеми на велосипедних маршрутах, сприяючи своєчасному покращенню та підтримці інфраструктури. Крім того, Cycling Brisbane доступний як веб-сервіс та мобільний додаток, що забезпечує зручність використання на різних пристроях.

Водночас, сервіс має і певні недоліки. Він обмежений лише містом Брисбен, що робить його непридатним для велосипедистів з інших регіонів Австралії або світу. Cycling Brisbane також не дозволяє користувачам додавати дані про парки, кафе або цікаві локації, які можуть бути важливими для велосипедистів під час подорожей.

Крім того, користувачі не можуть ділитися своїми маршрутами з іншими, що обмежує можливості для взаємодії та обміну досвідом у спільноті.

Сервіс також не враховує специфічні потреби різних категорій велосипедистів. Нарешті, якість та повнота інформації в додатку залежить від даних, наданих адміністрацією Брисбена, що може впливати на точність та актуальність сервісу.

Загалом, Cycling Brisbane є корисним інструментом для велосипедистів у місті Брисбен, але має певні обмеження щодо функціональності та географічного охоплення. Для покращення сервісу можна розглянути можливість розширення його функцій та адаптації для використання в інших регіонах.

Аналіз існуючих рішень показав, що хоча є кілька додатків та сервісів для планування велосипедних маршрутів, жоден з них не пропонує комплексного рішення з інтерактивною картою, де користувачі можуть додавати інформацію про інфраструктуру, місця відпочинку та ділитися маршрутами в зручному та привабливому інтерфейсі.

Таким чином, розробка нового сервісу з динамічною та інтерактивною картою для велосипедистів може заповнити цю прогалину на ринку та задовольнити потреби велосипедистів.

Висновки до розділу 1

Аналіз існуючих рішень у сфері сервісів та додатків для велосипедистів виявив важливі аспекти, які необхідно врахувати при розробці сервісу з динамічною та інтерактивною картою. Цей процес підкреслив значення задоволення потреб велосипедистів, створення зручного та інтуїтивного інтерфейсу, а також забезпечення можливості обміну інформацією та досвідом.

Для успішної реалізації проекту було зосереджено увагу на наступних ключових елементах:

- забезпечення комплексного рішення з інтерактивною картою, на якій користувачі можуть додавати та редагувати інформацію про велосипедну інфраструктуру, місця відпочинку та ділитися власними маршрутами;

- створення зручного та інтуїтивного інтерфейсу для планування маршрутів, пошуку релевантної інформації та взаємодії із спільнотою велосипедистів. Інтеграція нових функцій персоналізації, щоб користувачі могли налаштувати відображення інформації на карті відповідно до своїх уподобань та потреб;
- забезпечення високої точності та актуальності даних на карті завдяки можливості спільного редагування інформації користувачами;
- впровадження функцій безпеки та приватності даних, щоб захистити інформацію про маршрути та особисті дані користувачів.

РОЗДІЛ 2. ОБҐРУНТУВАННЯ ВИБОРУ ТЕХНОЛОГІЙ ТА ПЛАТФОРМИ РОЗРОБКИ

Для розробки сервісу з динамічною та інтерактивною картою для велосипедистів було обрано сучасний стек технологій, який забезпечує ефективність, масштабованість та зручність у використанні як для розробників, так і для кінцевих користувачів. Детальне обґрунтування вибору кожної технології наведено нижче.

2.1 Вибір технологій для серверної частини (backend)

Для реалізації серверної частини (backend) було обрано Ruby on Rails – одну з найпопулярніших та потужних веб-фреймворків, заснованих на мові програмування Ruby. Цей вибір був зумовлений такими перевагами:

1. Продуктивність та швидкість розробки: Ruby on Rails використовує парадигму Model-View-Controller (MVC) та принципи Конвенція над Конфігурацією (Convention over Configuration) і Не Повторюй Себе (Don't Repeat Yourself, DRY), що значно прискорює процес розробки.
2. Велика екосистема та підтримка спільноти: Ruby on Rails має величезну та активну спільноту розробників, що забезпечує широкий вибір бібліотек, плагінів та ресурсів для швидкого вирішення різноманітних задач.
3. Вбудована підтримка RESTful API: Rails надає зручні інструменти для створення RESTful API, що є важливим для взаємодії з клієнтською частиною (frontend) нашого сервісу.
4. Вбудована обробка та валідація даних: Rails має потужну систему обробки та валідації даних, що забезпечує високу надійність та безпеку при роботі з даними користувачів та картографічною інформацією.
5. Інтеграція з базами даних: Ruby on Rails підтримує широкий спектр реляційних і нереляційних баз даних, таких як SQLite, PostgreSQL, MongoDB,

MySQL та інші, що дозволяє гнучко вибирати оптимальне рішення для зберігання даних сервісу.

Для роботи з нереляційною базою даних MongoDB, яка відмінно підходить для зберігання геопросторових даних, була використана ORM бібліотека Mongoid. Вона забезпечує зручний доступ до MongoDB із Ruby on Rails, підтримуючи основні принципи роботи з Active Record.

Переваги MongoDB:

1. Гнучка схема даних:

− MongoDB використовує документно-орієнтовану модель даних, на відміну від жорсткої схеми реляційних баз даних;

− це дозволяє легко адаптувати структуру даних до змін вимог додатку, не вимагаючи складних міграцій схеми;

− для геопросторових даних, які можуть мати складну ієрархічну структуру, гнучкість MongoDB є особливо корисною.

2. Ефективне зберігання геопросторових даних:

− MongoDB має вбудовану підтримку геопросторових індексів, таких як 2dsphere і 2d;

− ці індекси дозволяють ефективно виконувати просторові запити, такі як пошук об'єктів у радіусі, пошук найближчих об'єктів, визначення перетинів, обчислення відстаней тощо;

− це робить MongoDB ідеальним вибором для додатків, що працюють з картами, геолокацією, навігацією та іншими геопросторовими даними.

3. Горизонтальна масштабованість:

− MongoDB підтримує горизонтальне масштабування, дозволяючи розподіляти дані та навантаження на кілька серверів;

− це забезпечує високу продуктивність та надійність роботи додатку навіть при великих обсягах даних;

− завдяки автоматичному збалансуванню навантаження та репліці даних, MongoDB може легко масштабуватися відповідно до зростаючих потреб додатку.

4. Висока доступність:

− MongoDB пропонує вбудовані функції реплікації, що дозволяють створювати кластери реплік для забезпечення високої доступності даних;

− автоматичне відновлення після збоїв гарантує, що дані будуть збережені навіть у разі апаратних або мережевих проблем;

− це робить MongoDB надійним вибором для критично важливих додатків, де простої неприпустимі.

5. Зручна інтеграція з Ruby on Rails:

− завдяки ORM-бібліотеці Mongoid, робота з MongoDB у Ruby on Rails стає простою та інтуїтивно зрозумілою;

− Mongoid надає знайомий інтерфейс, схожий на Active Record, що дозволяє розробникам швидко освоїти роботу з MongoDB [5];

− це забезпечує плавний перехід від традиційних реляційних баз даних до NoSQL-рішення MongoDB в рамках Ruby on Rails-додатку.

Ці переваги MongoDB роблять її відмінним вибором для розробки додатків, що працюють з геопросторовими даними, таких як картографічні сервіси, системи управління нерухомістю, навігаційні програми тощо.

Крім того, для реалізації GraphQL API, що дозволяє ефективно працювати з даними з клієнтської сторони, було використано бібліотеку graphql-ruby. GraphQL забезпечує гнучкий доступ до даних та зменшує кількість запитів між клієнтом та сервером порівняно з традиційним REST API.

Переваги GraphQL:

1. Гнучкість доступу до даних:

− GraphQL дозволяє клієнтам точно визначати, які дані їм потрібні, замість отримання надлишкової інформації, як це часто буває з традиційним REST API;

− клієнти можуть запитувати лише необхідні їм поля та зв'язки, що підвищує ефективність передачі даних;

− особливо корисно для складних додатків з великою кількістю взаємопов'язаних даних, де REST API може ставати громіздким.

2. Зменшення кількості запитів:

− на відміну від REST API, де для отримання всієї необхідної інформації може знадобитися кілька запитів, GraphQL дозволяє отримати всі дані в одному запиті;

− це зменшує накладні витрати на мережевий трафік та підвищує продуктивність додатку, особливо для мобільних пристроїв з обмеженою пропускною здатністю;

− клієнти можуть отримувати лише ті дані, які їм необхідні, що покращує загальну ефективність застосунку.

3. Еволюція API без порушення існуючого функціоналу:

− при використанні REST API, додавання нових ендпойнтів або зміна існуючих може призвести до порушення роботи клієнтських додатків;

− GraphQL дозволяє розробникам сервера розширювати API, не впливаючи на існуючий функціонал клієнтів;

− клієнти можуть продовжувати використовувати існуючі запити, навіть якщо на серверній стороні відбулися зміни;

4. Ефективна кешованість:

− GraphQL спрощує кешування даних на клієнтській стороні, оскільки кожен запит чітко визначає, які дані необхідні;

− це дозволяє клієнтам ефективно кешувати результати запитів і повторно використовувати їх, зменшуючи навантаження на сервер.

Використання GraphQL разом з MongoDB дозволяє створити потужний, гнучкий та високопродуктивний додаток для роботи з геопросторовими даними.

Оскільки було використано фреймворк Ruby on Rails, основною мовою для серверної частини була мова програмування Ruby. Ось кілька переваг використання мови програмування Ruby:

1. Читабельність та продуктивність:

− Ruby відома своєю лаконічністю та виразністю синтаксису, що робить код легким для розуміння та підтримки;

− завдяки концепції "Конвенція над Конфігурацією", Ruby дозволяє швидко реалізовувати типові задачі, підвищуючи продуктивність розробників.

2. Динамічна типізація:

− Ruby – це динамічно типізована мова, що дозволяє писати гнучкіший та адаптивніший код без необхідності явного визначення типів;

− це особливо корисно при роботі з різноманітними даними, такими як геопросторові об'єкти, які можуть мати складну структуру.

3. Об'єктно-орієнтований підхід:

− Ruby повністю підтримує об'єктно-орієнтований парадигму, що дозволяє створювати модульний, масштабований та легко підтримуваний код;

− це добре узгоджується з архітектурою Model-View-Controller (MVC), яку використовує фреймворк Ruby on Rails.

4. Багата екосистема та бібліотеки:

− Ruby має велику і активну спільноту розробників, що забезпечує доступ до тисяч готових бібліотек та інструментів;

− це значно прискорює розробку, дозволяючи використовувати перевірені та надійні рішення для широкого спектру задач.

5. Простота та виразність:

− синтаксис Ruby вважається інтуїтивно зрозумілим і легким для вивчення, особливо для розробників, які вже мають досвід роботи з іншими мовами;

− це дозволяє швидко залучати нових членів до команди та скорочує час на навчання.

6. Крос-платформність. Ruby може працювати на різних операційних системах, включаючи Windows, macOS та Linux, що забезпечує гнучкість при розгортанні та супроводі додатку.

Ці переваги мови програмування Ruby, у поєднанні з ефективним фреймворком Ruby on Rails, роблять її відмінним вибором для розробки серверної частини картографічного додатку, який працює з геопросторовими та динамічними даними.

2.2 Вибір технологій для клієнтської частини (frontend)

Для розробки клієнтської частини (frontend) було обрано React.js – одну з найпопулярніших та ефективних бібліотек для створення користувацьких інтерфейсів. Цей вибір був зумовлений такими перевагами:

1. Компонентна архітектура: React.js базується на компонентній архітектурі, що дозволяє розбивати користувацький інтерфейс на окремі, ізольовані та багаторазово використовувані компоненти. Це полегшує розробку, супровід та тестування коду [1].

2. Віртуальний DOM: React використовує віртуальний DOM, який оптимізує оновлення інтерфейсу, виконуючи лише необхідні зміни на сторінці замість повного перезавантаження. Це забезпечує високу продуктивність та плавний досвід для користувачів.

3. Велика екосистема та підтримка спільноти: React.js має величезну та активну спільноту розробників, що забезпечує широкий вибір бібліотек, інструментів та ресурсів для швидкого вирішення різноманітних задач.

4. Можливість використання JSX: React дозволяє використовувати JSX – синтаксис, схожий на HTML, для опису структури користувацького інтерфейсу прямо в JavaScript-коді. Це робить код більш читабельним та зрозумілим для розробника.

5. Підтримка React Native: React Native дозволяє використовувати ті самі принципи розробки та знання React.js для створення мобільних додатків для iOS та Android, що може знадобитися у майбутньому для розширення функціоналу сервісу.

Для стилізації компонентів було використано Material-UI – одну з найпопулярніших бібліотек з готовими компонентами інтерфейсу, заснованими на фірмових принципах Material Design від Google. Це забезпечило швидку розробку та приємний для користувача дизайн.

Переваги Material-UI:

1. Готові компоненти та стилі:

− Material-UI надає великий набір високоякісних, готових до використання компонентів, таких як кнопки, меню, діалогові вікна, форми тощо;

− ці компоненти вже мають визначений стиль, заснований на принципах Material Design, що значно прискорює процес розробки інтерфейсу.

2. Уніфікований дизайн:

− використання Material-UI забезпечує послідовність та уніфікованість дизайну в межах всього додатку;

− це допомагає створити привабливий та інтуїтивно зрозумілий інтерфейс для кінцевих користувачів.

3. Адаптивність та відзивчивість:

− компоненти Material-UI розроблені з урахуванням принципів адаптивного дизайну, що забезпечує коректне відображення на різних пристроях та розмірах екранів;

− це важливо для забезпечення позитивного досвіду користувачів, незалежно від того, з якого пристрою вони працюють.

4. Гнучкість та кастомізація:

− Material-UI надає можливість гнучкої кастомізації стилів та поведінки компонентів відповідно до брендингу та дизайн-системи додатку;

− ви можете легко налаштовувати кольори, типографіку, розміри та інші візуальні аспекти, зберігаючи при цьому уніфікованість інтерфейсу.

5. Велика спільнота та екосистема:

− Material-UI має велику та активну спільноту розробників, що забезпечує доступ до широкого спектру додаткових компонентів, плагінів та ресурсів;

− це дозволяє розширювати функціональність додатку, використовуючи перевірені та підтримувані рішення.

6. Продуктивність розробки:

− використання готових компонентів Material-UI значно скорочує час на розробку інтерфейсу, дозволяючи зосередитися на реалізації основної бізнес-логіки;

− це підвищує загальну продуктивність команди розробників та скорочує терміни виходу продукту на ринок.

Застосування Material-UI для стилізації компонентів дозволяє створити сучасний, адаптивний та естетично привабливий інтерфейс користувача, що значно покращує загальний досвід взаємодії з картографічним сервісом.

Для управління станом додатку та організації потоків даних було використано Redux Toolkit – бібліотеку для управління станом додатку з використанням архітектурного патерну Redux [13]. Це дозволило ефективно керувати даними, отриманими з серверної частини, та забезпечило зручний доступ до них у різних компонентах.

Переваги Redux Toolkit:

1. Спрощення налаштування Redux:

− Redux Toolkit значно спрощує початкове налаштування Redux, мінімізуючи рутинну роботу з конфігурацією;

− він включає в себе необхідні залежності, налаштування Middleware та інші типові конфігурації, що дозволяє швидко розгорнути Redux в проекті.

2. Зменшення кількості коду:

− Redux Toolkit надає набір функцій-помічників, які дозволяють писати більш лаконічний та читабельний код для Redux;

− це включає в себе скорочення дублювання коду при створенні екшенів, редюсерів та селекторів.

3. Підтримка сучасних можливостей JavaScript. Redux Toolkit використовує новітні можливості JavaScript, такі як immer.js для оновлення стану, що робить код більш виразним та менш схильним до помилок.

4. Вбудована підтримка асинхронних операцій:

− Redux Toolkit інтегрує Redux Thunk або Redux Saga, що дозволяє ефективно управляти асинхронними операціями, такими як запити до API;

− це спрощує обробку асинхронних дій та забезпечує єдиний підхід до управління станом додатку.

5. Покращена типізація:

− Redux Toolkit надає вбудовану підтримку TypeScript, що робить код більш типобезпечним та зменшує кількість помилок під час розробки;

− це особливо корисно для великих та складних додатків, де типізація стає критично важливою.

6. Кращі практики та рекомендації:

− Redux Toolkit слідує найкращим практикам роботи з Redux, включаючи рекомендації від авторів бібліотеки;

− це допомагає уникнути типових помилок та забезпечує більш структурований та масштабований підхід до управління станом додатку.

7. Розширюваність:

− Redux Toolkit є модульним, що дозволяє вибірково підключати лише необхідні функціональності [18];

− це забезпечує гнучкість та можливість адаптації бібліотеки до специфічних потреб додатку.

Використання Redux Toolkit для управління станом дозволяє створити більш структурований, масштабований та ефективний підхід до роботи з даними, отриманими з серверної частини. Це значно покращує загальну якість та підтримуваність картографічного сервісу.

Для відображення інтерактивної карти було використано бібліотеку google-map-react, яка базується на Google Maps API та дозволяє легко інтегрувати карту в React-додаток.

Висновок до розділу 2

У цьому розділі було детально обґрунтовано вибір технологій та платформи для розробки сервісу з динамічною та інтерактивною картою для велосипедистів. Ретельний аналіз вимог проекту, продуктивності,

масштабованості та зручності для розробників і користувачів дозволив зробити виважений вибір.

Для реалізації серверної частини (backend) було обрано Ruby on Rails – потужний веб-фреймворк, який забезпечує високу продуктивність, швидкість розробки та підтримку великої спільноти розробників. Разом з бібліотеками Mongoid для роботи з MongoDB та graphql-ruby для реалізації GraphQL API, Rails надає ідеальне рішення для створення надійного та масштабованого серверного компонента.

Для клієнтської частини (frontend) було вибрано React.js – одну з найпопулярніших бібліотек для створення користувацьких інтерфейсів. Завдяки компонентній архітектурі, віртуальному DOM та великій екосистемі, React.js забезпечує високу продуктивність, зручність розробки та можливість створення привабливого та інтерактивного інтерфейсу для користувачів. Використання Material-UI для стилізації компонентів та Redux для управління станом додатку додатково підвищує якість та зручність front-end частини.

Вибір Vite як інструменту для створення веб-проектів забезпечує швидкий час запуску та оптимізацію продуктивності під час розробки [19]. Використання ESLint для статичного аналізу коду допомагає підтримувати високу якість та дотримуватися кращих практик кодування [20].

Обраний стек технологій та платформ забезпечує ідеальне поєднання продуктивності, масштабованості, зручності розробки та підтримки, що є ключовим для успішної реалізації сервісу з динамічною та інтерактивною картою для велосипедистів. Цей вибір дозволяє створити високоякісний, функціональний та зручний у використанні програмний продукт, який відповідає вимогам проекту та задовольняє потреби цільової аудиторії.

РОЗДІЛ 3. ПРАКТИЧНА РЕАЛІЗАЦІЯ

3.1 Створення дизайну сервісу

Наступним кроком після планування та вибору технологій, безумовно, є розробка дизайну та користувацького інтерфейсу сервісу. Зручний та інтуїтивний дизайн відіграє критичну роль у забезпеченні позитивного досвіду для користувачів та ефективної взаємодії з картографічними даними та функціоналом додатку.

Під час створення дизайну важливо врахувати кілька ключових аспектів:

- зручність навігації: Інтерфейс має бути логічно структурованим та забезпечувати легку навігацію між різними розділами та функціями сервісу. Це включає розміщення елементів керування в зручних та інтуїтивно зрозумілих місцях на сторінці;
- акцент на карті: Оскільки основною особливістю сервісу є інтерактивна карта, дизайн повинен надавати достатньо місця для її відображення та взаємодії з нею. Карта має бути чітко видимою та легкодоступною з будь-якої частини інтерфейсу;
- розумне групування функцій: Різні функції, такі як планування маршрутів, додавання інформації про інфраструктуру, обмін маршрутами, повинні бути логічно згруповані та розташовані у відповідних розділах інтерфейсу користувача. Це допоможе користувачам швидко знаходити потрібні їм інструменти;
- зрозумілі візуальні підказки: Використання чітких піктограми, кольорів та візуальних індикаторів для полегшення розуміння різних елементів інтерфейсу та їх функцій. Це особливо важливо для швидкої ідентифікації різних типів велосипедної інфраструктури на карті;
- адаптивний дизайн: Сервіс повинен коректно відображатися та функціонувати на різних пристроях, включаючи настільні комп'ютери,

планшети та смартфони. Адаптивний дизайн забезпечить зручність використання для всіх категорій користувачів;

− дотримання принципів юзабіліті: Дотримуйтесь загальновизнаних принципів юзабіліті, таких як послідовність, простота, зворотний зв'язок та запобігання помилкам. Це створить інтуїтивний та приємний досвід користування для велосипедистів різного рівня технічної грамотності;

− брендинг та візуальна привабливість: Дизайн повинен відповідати загальному брендингу сервісу, використовуючи кольорову гаму, шрифти та елементи стилю, які відображають його ідентичність. Привабливий візуальний дизайн сприяє підвищенню задоволеності користувачів та формуванню позитивного сприйняття сервісу.

Під час створення дизайну може бути корисним залучити досвідчених дизайнерів користувацького інтерфейсу та провести тестування з представниками цільової аудиторії (велосипедистами) для отримання зворотного зв'язку та вдосконалення дизайну.

Для створення дизайну сервісу з інтерактивною картою для велосипедистів було використано Figma – потужний інструмент для веб-дизайну та макетування інтерфейсів.

Figma – це онлайн-редактор з багатим функціоналом для створення дизайну веб-сайтів, мобільних додатків та інтерфейсів. Ця програма має низку переваг, які зробили її відмінним вибором для розробки дизайну сервісу:

− хмарна платформа: Figma є повністю хмарним інструментом, доступним через веб-браузер або як десктоп-додаток. Це дозволяє працювати з дизайном з будь-якого пристрою та легко співпрацювати з іншими дизайнерами та розробниками в режимі реального часу;

− векторна графіка: Figma базується на використанні векторної графіки, що забезпечує високу чіткість та масштабованість створених дизайнів на різних пристроях та роздільних здатностях;

− багатий інструментарій: Figma пропонує широкий спектр інструментів для створення макетів, вектор, растрової графіки, прототипування

та анімації. Це дозволяє створювати високоякісні дизайни з використанням різноманітних ефектів та візуальних елементів;

− бібліотеки та плагіни: Figma має велику екосистему бібліотек та плагінів, що дозволяє розширювати функціональність та підвищувати продуктивність роботи. Доступні бібліотеки UI-елементів, плагіни для інтеграції з іншими інструментами та багато іншого;

− співпраця в режимі реального часу: Одна з найбільших переваг Figma – це можливість співпраці в режимі реального часу. Кілька дизайнерів та розробників можуть одночасно працювати над одним проектом, бачачи зміни один одного та залишаючи коментарі.

Хоча Figma є потужним інструментом, вона також має деякі недоліки:

− обмежені можливості для роботи із растровою графікою: Оскільки Figma орієнтована на векторну графіку, її інструменти для роботи з растровими зображеннями можуть бути менш розвиненими порівняно зі спеціалізованими редакторами растрової графіки, такими як Adobe Photoshop;

− необхідність постійного підключення до інтернету: Через хмарну природу Figma, для роботи з нею потрібне постійне підключення до інтернету. Це може стати проблемою в умовах з поганим інтернет-покриттям.

− крива навчання: Незважаючи на інтуїтивний інтерфейс, Figma має досить широкий функціонал, що може вимагати певного часу на освоєння всіх інструментів та особливостей програми.

У рамках проекту, Figma дозволить ефективно створити високоякісний дизайн для сервісу з інтерактивною картою для велосипедистів, враховуючи різні аспекти користувацького досвіду, естетики та функціональності.

Для швидкої візуалізації ідей дизайну створюю wireframe сервісу.

Wireframe – це візуальний шаблон або макет інтерфейсу, який фокусується на функціональному розташуванні та організації елементів, не акцентуючи увагу на візуальних деталях, таких як кольори, шрифти та графіка. Його основна мета – визначити структуру та взаємодію компонентів інтерфейсу до того, як будуть розроблені остаточні дизайн та стилі.

Використання wireframes допомагає швидко візуалізувати ідеї та концепції дизайну, тестувати юзабіліті, отримувати зворотний зв'язок від зацікавлених сторін та ітеративно вдосконалювати структуру інтерфейсу до створення фінального дизайну. Це економить час та зусилля, оскільки змінювати wireframes набагато простіше, ніж переробляти повністю зроблений графічний дизайн.

Для ефективного планування структури та компонування інтерфейсу сервісу з інтерактивною картою для велосипедистів було створено wireframes ключових сторінок. Wireframes є візуальними макетами, що демонструють розташування елементів інтерфейсу та взаємодію між ними, не акцентуючи увагу на дизайнерських деталях, таких як кольори, шрифти та графіка. Вони допомагають швидко візуалізувати ідеї та концепції дизайну, тестувати юзабіліті та отримувати зворотний зв'язок від зацікавлених сторін.

Сервіс містить наступні основні сторінки:

1. Сторінка логіну (Додаток А): Ця сторінка дозволяє користувачам увійти до системи. Її wireframe містить форму реєстрації/логіну через Google-аккаунт.

2. Сторінка з картою та списком цікавих місць для незареєстрованих користувачів (Додаток Б): Ця сторінка демонструє основну функціональність сервісу незареєстрованим користувачам. Її wireframe включає: а) навігаційне меню; б) інтерактивну карту з фільтрами та кнопкою входу; в) список цікавих місць, доданих іншими користувачами; г) заклик до реєстрації; ґ) футер.

3. Сторінка з описом певного цікавого місця для незареєстрованих користувачів (Додаток В): Ця сторінка надає детальну інформацію про обране цікаве місце та коментарі інших користувачів. Її wireframe містить: а) навігаційне меню; б) інтерактивну карту з фільтрами; в) детальний опис цікавого місця; г) відгуки про обране місце; ґ) заклик до реєстрації; д) футер.

4. Сторінка з картою та списком цікавих місць для зареєстрованих користувачів (Додаток Г): Аналогічна сторінка до пункту 3, але з додатковими

функціями для зареєстрованих користувачів, такими як можливість додавати нові цікаві місця, залишати відгуки та редагувати власний контент.

5. Сторінка з описом певного цікавого місця для зареєстрованих користувачів (Додаток Г): Аналогічна сторінка до пункту 4, але з додатковими функціями для зареєстрованих користувачів, такими як можливість залишати відгуки та коментарі, а також редагувати власний контент.

6. Сторінка персонального кабінету користувача (Додаток Д): Ця сторінка дозволяє зареєстрованим користувачам переглядати та змінювати свої особисті дані, список друзів, а також список доданих ними цікавих місць та шляхів на карті.

Створення wireframes допомогло визначити оптимальний шлях взаємодії для користувачів та логічно організувати різні функції сервісу. Залучення велосипедистів та експертів з юзабіліті на етапі тестування wireframes дозволило отримати цінний зворотний зв'язок та внести необхідні ітерації для вдосконалення структури інтерфейсу. Остаточні wireframes стали основою для розробки візуального дизайну, забезпечуючи зручність використання та позитивний досвід для цільової аудиторії велосипедистів.

Після створення wireframes наступним кроком стало розроблення прототипу без використання реальних зображень. Це проміжна стадія між wireframes та остаточним візуальним дизайном, яка дозволяє краще уявити взаємодію користувача з інтерфейсом та провести більш реалістичне тестування.

У прототипі були використані тимчасові заповнювачі для зображень, такі як кольорові прямокутники або іконки, замість реальних зображень або графіки. Проте інші елементи інтерфейсу, такі як тексти, кнопки, навігаційні меню, були представлені більш реалістично, наближаючись до їх майбутнього вигляду в остаточному дизайні.

Створення прототипу без зображень дозволило зосередитися на структурі інтерфейсу, розташуванні елементів керування, ієрархії інформації та потоці взаємодії користувача. Це допомогло виявити та вирішити потенційні проблеми

юзабіліті на ранній стадії, до того, як були витрачені значні зусилля на детальний візуальний дизайн.

3.2 Створення архітектури серверної частини

Для зберігання даних сервісу з інтерактивною картою для велосипедистів буде використано нереляційну базу даних MongoDB. Для розуміння структури зберігання даних використовується діаграму класів для бази даних (рис. 3.2.1).

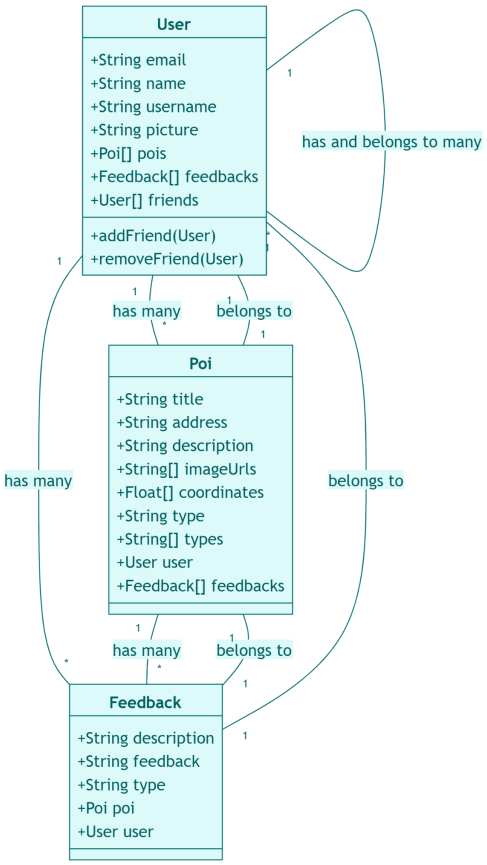


Рисунок 3.2.1 – Діаграма класів для бази даних MongoDB

Архітектура бази даних включає кілька ключових моделей:

1. User: ця модель представляє зареєстрованого користувача сервісу. Вона містить поля для зберігання логіну, електронної пошти, імені, імені користувача, посилання на зображення, а також списки змін Pois та Friends, додані цим користувачем.

2. Poi (скорочення від Point of Interest, точка зацікавлення): модель яка представляє будь-яку зміну для карти зроблену користувачами. Містить поля для зберігання типу, заголовку, адреси, опису зміни, перелік зображень, координатів, списку типів для місця, автора, списку відгуків.

3. Feedback: модель яка представляє відгук на зміни для карти зробленими користувачами. Містить поля для зберігання типу, опису, значення відгуку (вподобання, не вподобання), автора та зміни.

Ця архітектура бази даних забезпечує гнучке та ефективне зберігання даних сервісу, включаючи інформацію про користувачів, маршрути, цікаві місця та елементи інфраструктури. Використання MongoDB дозволяє легко працювати з геопросторовими даними та забезпечує масштабованість та високу продуктивність для додатків, пов'язаних з картографічними даними.

3.3 Розробка серверної частини

3.3.1 Ініціалізація проекту

Для ініціалізації серверної частини проекту було використано Ruby on Rails – потужний веб-фреймворк для мови програмування Ruby. Rails забезпечує зручний і швидкий спосіб для створення веб-додатків, а також надає багато вбудованих інструментів і бібліотек для прискорення розробки.

Процес ініціалізації проекту в Rails розпочався з використання команди rails new у терміналі. Ця команда створює нову директорію з усією необхідною структурою файлів та директорій для Rails-додатку [4]. Наприклад:

```
rails new rider-app
```

Ця команда згенерувала директорію rider-app із стандартною структурою Rails, включаючи такі основні директорії та файли:

1. app/ – Містить основну логіку додатку, розділену на підкаталоги для моделей, відображень, контролерів і допоміжних модулів.

2. config/ – Зберігає конфігураційні файли для налаштування Rails-додатку, бази даних, середовища розробки тощо.

3. db/ – Містить файли міграцій для керування схемою бази даних та seeds-файли для початкового наповнення даних.

4. Gemfile – Файл, який визначає залежності додатку від зовнішніх бібліотек (гемів).

Після ініціалізації проекту наступним кроком було налаштування середовища розробки. Це включало встановлення необхідних гемів, зазначених у Gemfile, за допомогою команди bundle install.

Одним з ключових гемів, встановлених для цього проекту, був mongoid – бібліотека для взаємодії з нереляційною базою даних MongoDB із Ruby on Rails [16]. MongoDB було обрано для зберігання геопросторових даних через її гнучку схему даних та вбудовану підтримку просторових індексів і запитів.

Після встановлення необхідних гемів, наступним кроком було налаштування з'єднання з базою даних MongoDB. Це було зроблено шляхом редагування файлу config/mongoid.yml, де були вказані параметри підключення до MongoDB, такі як хост, порт та ім'я бази даних.

З цього моменту Rails-додаток був готовий для подальшого розвитку та реалізації основної функціональності серверної частини сервісу з інтерактивною картою для велосипедистів.

3.3.2 Створення та підключення бази даних

Для зберігання даних сервісу з інтерактивною картою для велосипедистів було використано хмарну базу даних MongoDB Atlas. MongoDB Atlas – це хмарний сервіс, що надає повністю керовані кластери MongoDB, забезпечуючи високу доступність, безпеку та масштабованість.

Процес створення та підключення бази даних MongoDB Atlas включав наступні кроки:

1. Реєстрація в MongoDB Atlas та створення нового кластера [6].
2. Вибір провайдера хмарної платформи (наприклад, AWS, GCP або Azure) та регіону розміщення кластера.
3. Налаштування безпеки кластера, включаючи створення користувача з правами адміністратора.
4. Отримання рядка з'єднання для підключення до кластера.
5. Редагування файлу config/mongoid.yml в Rails-додатку та вказування отриманого рядка з'єднання.

Після успішного підключення до MongoDB Atlas, Rails-додаток міг взаємодіяти з базою даних за допомогою ORM-бібліотеки Mongoid, яка забезпечує зручний інтерфейс для роботи з MongoDB, аналогічний до Active Record для реляційних баз даних.

3.3.3 Створення точок доступу до бази даних

Базуючись на архітектурі бази даних, що була визначена раніше, було створено моделі Mongoid, які представляють точки доступу до даних у базі даних MongoDB.

Розглянемо модель користувача (User) як приклад:

```
class User
  include Mongoid::Document
  include Mongoid::Timestamps
  field :email, type: String
  field :name, type: String
  field :username, type: String
  field :picture, type: String
  has_many :pois
  has_many :feedbacks
```

```
    has_and_belongs_to_many :friends, class_name: 'User', inverse_of:
:friend_ids
  end
```

Модель користувача включає в себе підключення необхідних модулів Mongoid для роботи з документоорієнтованою базою даних та визначення полів, які зберігають інформацію про користувача: електронну пошту, ім'я, логін та посилання на зображення профілю.

Крім того, модель визначає зв'язки з іншими моделями:

− has_many :pois та has_many :feedbacks вказують, що один користувач може мати багато точок інтересу (POIs) та відгуків;

− has_and_belongs_to_many :friends реалізує зв'язок "багато-до-багатьох" для списку друзів, де кожен користувач може мати багато друзів, і кожен друг також є користувачем.

Для управління списком друзів у моделі User визначено два методи:

```
def add_friend(friend)
  unless self.friends.include?(friend) || self == friend
    self.friends << friend
    friend.friends << self
  end
end

def remove_friend(friend)
  if self.friends.include?(friend)
    self.friends.delete(friend)
    friend.friends.delete(self)
  end
end
```

Ці методи дозволяють додавати та видаляти друзів зі списку. Метод add_friend додає користувача до списку друзів, якщо він ще не є другом і не є самим користувачем. Метод remove_friend видаляє користувача зі списку друзів,

якщо він там присутній. Обидва методи працюють симетрично, тобто зміни відображаються в списках обох користувачів.

Після розгляду моделі користувача (User), перейдемо до іншої важливої моделі - моделі змін (Poi). Ця модель представляє точки інтересу або маршрути, які користувачі можуть створювати та ділитися з іншими. Ось її структура:

```
class Poi
  include Mongoid::Document
  include Mongoid::Timestamps
  field :title, type: String
  field :address, type: String
  field :description, type: String
  field :image_urls, type: Array
  field :coordinates, type: Array
  field :type, type: String
  field :types, type: Array
  belongs_to :user, required: true
  has_many :feedbacks, dependent: :destroy
end
```

Як і модель User, модель Poi використовує модулі Mongoid для роботи з MongoDB. Вона має кілька ключових полів:

- title і address для зберігання назви та адреси точки інтересу;
- description для детального опису;
- image_urls і coordinates - масиви для зберігання URL-адрес зображень та географічних координат;
- type і types для класифікації точки інтересу.

Модель також визначає зв'язки:

- belongs_to :user вказує, що кожна точка інтересу належить конкретному користувачеві;
- has_many :feedbacks дозволяє точці інтересу мати багато відгуків, які будуть видалені разом з нею (dependent: :destroy).

Важливою частиною моделі Poi є визначення типів точок інтересу:

```
TYPES = {
  place: {
    icon: 'src/path/to/icon.svg',
    description: 'Place of interest'
  },
  path: {
    icon: 'src/path/to/icon.svg',
    description: 'Path or route'
  }
}
validates :type, inclusion: { in: TYPES.keys.map(&:to_s) }
```

Тут використовується константа TYPES для визначення допустимих типів точок інтересу. На даний момент є два типи:

1. place - конкретне місце інтересу, наприклад, пам'ятка або ресторан.
2. path - маршрут або шлях, яким користувач хоче поділитись.

Для кожного типу цікавих місць визначено шлях до іконки (icon) та короткий опис (description).

Останній рядок є валідацією, яка гарантує, що значення поля type завжди буде одним із заздалегідь визначених ключів у хеші TYPES. Це запобігає помилкам введення даних і забезпечує цілісність даних.

Таким чином, модель Poi надає структуровану та гнучку основу для зберігання різноманітних точок інтересу. Вона дозволяє користувачам не тільки додавати базову інформацію, але й класифікувати свої дописи, що покращує користувацький досвід при навігації та пошуку.

Після розгляду моделей User та Poi, перейдемо до останньої ключової моделі - моделі відгуків (Feedback). Ця модель є критично важливою для створення інтерактивного та динамічного досвіду користувачів, дозволяючи їм ділитися своїми думками та реакціями на різні точки інтересу. Ось структура моделі Feedback:

```
class Feedback
  include Mongoid::Document
  include Mongoid::Timestamps
  field :description, type: String
  field :feedback, type: String
  field :type, type: String
  field :poi_id, type: BSON::ObjectId
  field :user_id, type: BSON::ObjectId
  belongs_to :poi, polymorphic: true
  belongs_to :place, optional: true
  belongs_to :path, optional: true
  belongs_to :user
end
```

Як і попередні моделі, Feedback використовує модулі Mongoid для інтеграції з MongoDB, забезпечуючи функціональність документоорієнтованої бази даних та автоматичне відстеження часових міток.

Модель має кілька текстових полів для зберігання різної інформації:

- description - для детального опису відгуку;
- feedback - для короткого відгуку або оцінки;
- type - для категоризації відгуку (наприклад, "позитивний", "негативний", "нейтральний").

Важливою особливістю є використання полів poi_id та user_id типу BSON::ObjectId. Це спеціальний тип даних MongoDB для ефективного зберігання та індексації унікальних ідентифікаторів. Ці поля зберігають посилання на конкретну точку інтересу та користувача, який залишив відгук.

Модель Feedback має складні відносини з іншими моделями:

1. belongs_to :poi, polymorphic: true - це поліморфний зв'язок, який дозволяє відгуку бути асоційованим з будь-яким об'єктом, що має інтерфейс poi. Це забезпечує гнучкість, якщо в майбутньому з'являться нові типи точок інтересу.

2. belongs_to :place, optional: true та belongs_to :path, optional: true - ці зв'язки дозволяють відгуку бути безпосередньо пов'язаним з конкретними типами точок інтересу.

Опція optional: true означає, що відгук може існувати без цих зв'язків, що є логічним, оскільки він буде пов'язаний лише з одним із цих типів.

3. belongs_to :user - кожен відгук належить конкретному користувачеві, який його залишив.

Ці моделі Mongoid забезпечують зручні точки доступу до даних у MongoDB для подальшої роботи з ними в Rails-додатку. Вони визначають поля, зв'язки між моделями та можуть містити додаткові валідації та методи для роботи з даними.

Окрім традиційного REST API, для надання гнучкого та ефективного доступу до даних з клієнтської сторони було використано GraphQL [17].

Бібліотека graphql-ruby була інтегрована в Rails-додаток для реалізації GraphQL API.

У директорії app/graphql, знаходяться всі необхідні файли для налаштування та роботи з GraphQL API. У файлі app/graphql/rider_app_schema.rb визначено основну схему GraphQL для додатку:

```
class RiderAppSchema < GraphQL::Schema
  mutation(Types::MutationType)
  query(Types::QueryType)
  use GraphQL::Dataloader
  def self.type_error(err, context)
    super
  end
  def self.resolve_type(abstract_type, obj, ctx)
    raise(GraphQL::RequiredImplementationMissingError)
  end
  validate_max_errors(100)
  def self.id_from_object(object, type_definition, query_ctx)
    object.to_gid_param
```

```
    end
    def self.object_from_id(global_id, query_ctx)
      GlobalID.find(global_id)
    end
  end
```

Ця схема посилається на типи Types::MutationType та Types::QueryType, які визначають доступні мутації (для зміни даних) та запити (для отримання даних) відповідно.

У директорії app/graphql/types знаходяться визначення різних типів даних, що використовуються в GraphQL API [7]. Наприклад, app/graphql/types/user_type.rb визначає тип UserType:

```
  module Types
    class UserType < Types::BaseObject
        field :id, ID, null: false, description: "MongoDB User id
string"
      field :email, String, null: true, description: "User's email"
      field :name, String, null: true, description: "User's name"
         field :username, String, null: true, description: "User's
username"
          field :picture, String, null: true, description: "User's
picture"
      field :pois, [Types::PoiType], null: true
      field :feedbacks, [Types::FeedbackType], null: true
      field :friends, [Types::UserType], null: true
    end
  end
```

Цей тип визначає поля, які можуть бути запитані для об'єктів типу User, такі як id, email, name, username, picture, а також зв'язки з іншими типами даних, як-от pois, feedbacks, friends.

У процесі створення GraphQL API важливу роль відіграє визначення типів, які точно відображають структуру даних. Розглянуто два ключові типи:

PoiType та CoordinateType. Спочатку було визначено допоміжний тип для переліку значень PoiTypeEnum:

```
module Types
  class PoiTypeEnum < Types::BaseEnum
    value "place", "A place of interest"
    value "path", "A path or route"
  end
end
```

У цьому коді визначено перелічуваний тип, який встановлює можливі значення для поля type в PoiType: "place" для конкретних місць інтересу та "path" для маршрутів або шляхів. Таке рішення допомагає чітко категоризувати точки інтересу, що є важливим для правильної інтерпретації даних клієнтами.

Далі було переведено увагу на основний тип PoiType:

```
module Types
  class PoiType < Types::BaseObject
      field :id, ID, null: false, description: "MongoDB Poi id
string"
    field :title, String, null: false
    field :address, String, null: true
    field :description, String, null: true
    field :image_urls, [String], null: true
    field :coordinates, [Types::CoordinateType], null: false
    field :type, Types::PoiTypeEnum, null: false
    # ...
  end
end
```

У цьому коді визначено набір полів, які точно відображають структуру моделі Poi у базі даних. Кожне поле має свій тип даних і визначено як обов'язкове або необов'язкове. Наприклад, title є рядком і обов'язковим полем,

тоді як address та description також є рядками, але можуть бути пустими. Особливу увагу було приділено полям coordinates та type. Для coordinates було використано масив типу CoordinateType, що дозволяє точно визначати географічне положення. А для type було застосовано раніше визначений PoiTypeEnum, забезпечуючи чітку та обмежену категоризацію точок інтересу. У наступній частині коду PoiType було додано зв'язки з іншими типами та обчислювані поля:

```
module Types
  class PoiType < Types::BaseObject
    # ...
    field :user, Types::UserType, null: false
    field :feedbacks, [Types::FeedbackType], null: true
    def likes
      object.feedbacks.count { |f| f.feedback == 'like' }
    end
    def dislikes
      object.feedbacks.count { |f| f.feedback == 'dislike' }
    end
    field :likes, Int, null: false
    field :dislikes, Int, null: false
  end
end
```

Тут було встановлено зв'язок з типом UserType через поле user, що вказує на власника точки інтересу. Також було додано масив типу FeedbackType через поле feedbacks для відображення всіх відгуків.

У цій же частині коду було створено обчислювані поля likes і dislikes. Ці поля не зберігаються безпосередньо в базі даних, а обчислюються в реальному часі. Вони підраховують кількість позитивних і негативних відгуків, аналізуючи значення поля feedback у кожному об'єкті відгуку.

Під кінець було розглянуто тип CoordinateType:

```
module Types
  class CoordinateType < Types::BaseObject
    field :lat, Float, null: false
    field :lng, Float, null: false
  end
end
```

У цьому коді, хоча й простому, визначено критично важливий тип. У ньому є два обов'язкових поля типу Float: lat для широти та lng для довготи. Такий підхід забезпечує точне та стандартизоване представлення географічних координат на карті.

Використання окремого типу для координат, як це зроблено в останньому коді, має кілька стратегічних переваг. По-перше, поля lat і lng мають чітке та універсальне значення. По-друге, цей тип може бути повторно використано в інших частинах схеми. По-третє, GraphQL може автоматично перевіряти, що надані значення є дійсними числами з плаваючою точкою.

Для завершення огляду ключових типів у GraphQL API, було розглянуто тип FeedbackType:

```
module Types
  class FeedbackType < Types::BaseObject
    field :id, ID, null: false
    field :description, String, null: true
    field :feedback, String, null: true
    field :type, String, null: true
    field :created_at , String , null: true
    field :poi, Types::PoiType, null: true
    field :user, Types::UserType, null: false
  end
end
```

Цей тип відображає структуру моделі Feedback, забезпечуючи доступ до основної інформації про відгуки: текстовий опис, тип та час створення.

Важливо відзначити зв'язки з іншими типами: poi пов'язує відгук з конкретною точкою інтересу, а user - з користувачем, який його залишив. Такі зв'язки дозволяють клієнтам отримувати вичерпну інформацію про відгуки в контексті всього додатку.

У файлах query_type.rb та mutation_type.rb визначаються доступні запити та мутації відповідно. Наприклад, у query_type.rb можна визначити запит для отримання списку користувачів та конкретного користувача:

```
module Types
class QueryType < Types::BaseObject
field :users, [Types::UserType], null: false, description: "Return users"
field :user, Types::UserType, null: true do
      argument :id, ID, required: true
end
def users
      User.all
end
def user(id:)
      User.find(id)
end
```

Визначення запиту для отримання змін (Poi). Визначення поля pois:

```
field :pois, [Types::PoiType], null: true, description: "Return Points of Interest" do
  argument :userId, String, required: false
  argument :type, String, required: false
end
```

Це поле повертає список об'єктів типу Types::PoiType. Воно приймає два необов'язкові аргументи: userId та type, які можуть бути використані для фільтрації результатів. Реалізація методу pois:

```
def pois(type: nil, userId: nil)
  pois = Poi.all
  if userId
    pois = pois.where(user_id: userId)
  end
  if type
    pois = pois.where(type: type)
  end
  pois
end
```

Метод pois отримує всі точки інтересу (Poi.all), а потім фільтрує їх за userId та type (якщо вони передані).

Визначення поля poi:

```
field :poi, Types::PoiType, null: true, description: "Return Point of Interest by ID" do
  argument :id, ID, required: true
end
```

Це поле повертає один об'єкт типу Types::PoiType. Воно приймає один обов'язковий аргумент: id, який використовується для отримання конкретної точки інтересу. Реалізація методу poi:

```
def poi(id:)
  Poi.find(id)
end
```

Метод poi отримує конкретну точку інтересу за її id.

В mutation_type.rb визначено мутацію для створення нової зміни (Poi). Визначення поля addPoi:

```
    field :addPoi, PoiType, null: false, description: "Add a new Point
of Interest" do
      argument :title, String, required: true
      argument :description, String, required: false
      argument :image_urls, [String], required: false
      argument :coordinates, [Inputs::CoordinateInput], required: true
      argument :type, String, required: true
      argument :types, [String], required: false
      argument :user_id, ID, required: true
      argument :address, String, required: true
    end
```

Це поле визначає мутацію addPoi, яка повертає об'єкт типу PoiType. Воно приймає кілька аргументів, необхідних для створення нової точки інтересу.

Реалізація методу addPoi:

```
    def addPoi(title:, description: nil, image_urls: [], coordinates:,
type:, types:, user_id:, address:)
      coordinatesData = coordinates.map { |c| {lat: c.lat, lng: c.lng}}
      poi = Poi.create!(
        title: title,
        description: description,
        image_urls: image_urls,
        coordinates: coordinatesData,
        type: type,
        types: types,
        user_id: user_id,
        address: address)
      poi
    end
```

Метод addPoi приймає аргументи, визначені в полі addPoi, та створює нову точку інтересу (Poi) з цими даними. Координати перетворюються з формату, наданого клієнтом, у формат, необхідний для збереження в базі даних.

Визначення мутації для створення нового відгуку (Feedback). Визначення поля addFeedback:

```
    field :addFeedback, Types::FeedbackType, null: true, description:
"Create a new feedback" do
      argument :description, String, required: true
      argument :feedback, String, required: true
      argument :poi_id, ID, required: true
      argument :user_id, ID, required: true
    end
```

Це поле визначає мутацію addFeedback, яка повертає об'єкт типу Types::FeedbackType. Воно приймає кілька аргументів, необхідних для створення нового відгуку. Реалізація методу addFeedback:

```
    def addFeedback(description:, feedback:, poi_id:, user_id:)
      poi = Poi.find(poi_id)
      user = User.find(user_id)
      Feedback.create!(
        description: description,
        feedback: feedback,
        poi: poi,
        user: user,
      )
    end
```

Метод addFeedback приймає аргументи, визначені в полі addFeedback, та створює новий відгук (Feedback) з цими даними. Він також знаходить відповідну точку інтересу (poi) та користувача (user) за їхніми ідентифікаторами.

Визначення мутації для зміни псевдо імені користувача. Визначення поля changeUserUsername:

```
    field :changeUserUsername, Types::UserType, null: false,
description: "Change user username" do
      argument :id, ID, required: true
      argument :username, String, required: true
    end
```

Це поле визначає мутацію changeUserUsername, яка повертає об'єкт типу Types::UserType. Вона приймає два аргументи: id (ідентифікатор користувача) та username (нове псевдо ім'я користувача). Реалізація методу changeUserUsername:

```
    def changeUserUsername(id:, username:)
      begin
        user = User.find(id)
        user.username = username
        user.save!
        user
      rescue => e
        puts "Error updating username: #{e.message}"
        raise GraphQL::ExecutionError, "Failed to update username"
      end
    end
```

Метод changeUserUsername спочатку знаходить користувача за його id, потім змінює його username на новий, зберігає зміни та повертає оновлений об'єкт користувача. Якщо виникає помилка під час оновлення, вона обробляється, а користувачу повертається відповідне повідомлення про помилку.

Визначення мутації для зміни імені користувача. Визначення поля changeUserName:

```
    field :changeUserName, Types::UserType, null: false, description:
"Change user name" do
      argument :id, ID, required: true
      argument :name, String, required: true
```

```
end
```

Це поле визначає мутацію changeUserName, яка повертає об'єкт типу Types::UserType. Вона приймає два аргументи: id (ідентифікатор користувача) та name (нове ім'я користувача). Реалізація методу changeUserName:

```
def changeUserName(id:, name:)
  begin
    user = User.find(id)
    user.name = name
    user.save!
    user
  rescue => e
    puts "Error updating name: #{e.message}"
    raise GraphQL::ExecutionError, "Failed to update name"
  end
end
```

Метод changeUserName спочатку знаходить користувача за його id, потім змінює його name на нове, збережує зміни та повертає оновлений об'єкт користувача. Якщо виникає помилка під час оновлення, вона обробляється, а користувачу повертається відповідне повідомлення про помилку.

Визначення мутації для додавання користувача в друзі. Визначення поля addUserFriend:

```
field :addUserFriend, Types::UserType, null: false, description:
"Add user friend" do
  argument :id, ID, required: true
  argument :friend_id, ID, required: true
end
```

Це поле визначає мутацію addUserFriend, яка повертає об'єкт типу Types::UserType. Вона приймає два аргументи: id (ідентифікатор користувача) та

friend_id (ідентифікатор друга, якого потрібно додати). Реалізація методу addUserFriend:

```
def addUserFriend(id:, friend_id:)
  begin
    user = User.find(id)
    friend = User.find(friend_id)
    user.add_friend(friend)
    user.save!
    user
  rescue => e
    puts "Error adding friend: #{e.message}"
  end
end
```

Метод addUserFriend спочатку знаходить користувача та його друга за їхніми ідентифікаторами. Потім він використовує метод add_friend моделі User, щоб додати друга до списку друзів користувача. Після цього зміни зберігаються, і повертається оновлений об'єкт користувача. Якщо виникає помилка під час додавання друга, вона обробляється, але результат не повертається.

Визначення мутації для видалення користувача з списку друзів. Визначення поля removeUserFriend:

```
field :removeUserFriend, Types::UserType, null: false, description:
"Remove user friend" do
  argument :id, ID, required: true
  argument :friend_id, ID, required: true
end
```

Це поле визначає мутацію removeUserFriend, яка повертає об'єкт типу Types::UserType. Вона приймає два аргументи: id (ідентифікатор користувача) та friend_id (ідентифікатор друга, якого потрібно видалити). Реалізація методу removeUserFriend:

```
def removeUserFriend(id:, friend_id:)
  begin
    user = User.find(id)
    friend = User.find(friend_id)
    user.remove_friend(friend)
    user.save!
    user
  rescue => e
    puts "Error adding friend: #{e.message}"
  end
end
```

Метод removeUserFriend спочатку знаходить користувача та його друга за їхніми ідентифікаторами. Потім він використовує метод remove_friend моделі User, щоб видалити друга зі списку друзів користувача. Після цього зміни зберігаються, і повертається оновлений об'єкт користувача. Якщо виникає помилка під час видалення друга, вона обробляється.

Ці визначення дозволяють клієнтам виконувати запити та мутації за допомогою GraphQL API для отримання, створення, оновлення та видалення даних у додатку. Після налаштування GraphQL API його можна використовувати у веб-додатку.

Клієнтська частина додатку може відправляти GraphQL-запити на цей контролер та взаємодіяти з даними сервера через GraphQL API.

Використання GraphQL API у поєднанні з традиційним REST API забезпечує гнучкість та ефективність при роботі з даними в Rails-додатку. Клієнти можуть вибирати підхід, який найкраще відповідає їхнім потребам та вимогам до даних.

3.3.4 Хостинг серверної частини

Після розробки серверної частини на базі Ruby on Rails, постало питання розгортання та хостингу додатку.

Для вирішення цього завдання було обрано комбінацію кількох інструментів.

Для хостингу самого Ruby on Rails додатку було використано ngrok – сервіс, який дозволяє створювати безпечні тунелі до локально запущених веб-додатків, надаючи їм публічні URL-адреси [15].

Перевагою використання ngrok стала можливість швидко розгорнути додаток для тестування та демонстрації, без необхідності налаштування власного публічного хостингу на цьому етапі розробки. Процес розгортання на ngrok включав запуск локального Ruby on Rails додатку та створення тунелю за допомогою ngrok CLI, що надавало публічний URL-адрес для доступу до даного додатку.

Для зберігання даних сервісу було обрано рішення MongoDB Atlas – керований сервіс хмарної бази даних MongoDB. MongoDB Atlas забезпечує просте розгортання, масштабування та адміністрування MongoDB кластерів на хмарних платформах. Це дозволило отримати переваги, такі як автоматичне резервне копіювання, моніторинг продуктивності, можливість горизонтального масштабування та інтеграція з іншими хмарними сервісами. Під час розгортання було налаштовано підключення між Ruby on Rails додатком, розгорнутим за допомогою ngrok, та MongoDB Atlas кластером.

Для реалізації сервісного GraphQL API також було використано інструмент ngrok. Це дозволило швидко розгорнути GraphQL API для тестування та демонстрації, не вимагаючи публічного хостингу на цьому етапі розробки сервісу.

Таким чином, поєднання ngrok для хостингу Ruby on Rails додатку та MongoDB Atlas для хмарного зберігання даних забезпечило надійну, масштабовану та гнучку інфраструктуру для серверної частини сервісу з інтерактивною картою для велосипедистів. Ця архітектура хостингу дозволяє легко керувати, розширювати та підтримувати працездатність серверної частини додатку, одночасно забезпечуючи високу доступність, безпеку та ефективність роботи з даними.

3.4 Розробка клієнтської частини

3.4.1 Ініціалізація проекту

Для ініціалізації клієнтської частини веб-застосунку на основі React.js необхідно виконати наступні кроки:

1. Встановлення Node.js та npm.

Node.js – це середовище виконання JavaScript, яке дозволяє запускати JavaScript поза браузером. NPM (Node Package Manager) – це менеджер пакетів, який використовується для встановлення та керування залежностями проекту. Встановлення Node.js та npm є обов'язковим початковим кроком для будь-якого проекту на основі React.

2. Створення React-застосунку.

Після встановлення Node.js та npm створюється початкова структура React-застосунку за допомогою утиліти create-react-app. Ця утиліта забезпечує швидке створення початкового шаблону React-застосунку із необхідною конфігурацією та залежностями.

3. Налаштування середовища розробки.

На цьому етапі встановлюються та налаштовуються необхідні інструменти та бібліотеки для розробки клієнтської частини, такі як:

- React Router – для маршрутизації у застосунку
- Apollo Client – для взаємодії з GraphQL API
- Material-UI – бібліотека компонентів інтерфейсу користувача на основі матеріального дизайну
- Redux – бібліотека для управління глобальним станом застосунку

Налаштування цих інструментів та бібліотек вимагає відповідних конфігураційних файлів, імпортів та налаштувань у коді React-застосунку.

4. Встановлення системи контролю версій.

Для забезпечення ефективного відстеження змін у коді та співпраці над проектом встановлюється система контролю версій Git та створюється репозиторій для клієнтської частини проекту на GitHub або подібному хостингу.

5. Налаштування середовища розгортання.

На завершальному етапі ініціалізації клієнтської частини проекту налаштовується середовище для її розгортання. Це включає налаштування збірки та розгортання статичних файлів React-застосунку.

Ретельна ініціалізація клієнтської частини проекту є важливою для забезпечення належного середовища розробки, структури проекту та інструментів, необхідних для ефективної розробки інтерфейсу користувача та взаємодії з серверною частиною через GraphQL API.

3.4.2 Налаштування React Router

React Router – це популярна бібліотека, що забезпечує навігацію та маршрутизацію в React-застосунках. Вона дозволяє створювати одностраничні застосунки (SPA) з можливістю переходу між різними компонентами без повного перезавантаження сторінки. Налаштування React Router є важливим кроком у розробці сучасних веб-застосунків на React.

У рамках проекту кваліфікаційної роботи налаштування React Router включало наступні етапи:

1. Встановлення бібліотеки

Перш за все, було встановлено необхідні пакети для роботи з React Router за допомогою менеджера пакетів npm. Цей пакет містить основні компоненти та функціональність для роботи з маршрутизацією в React.

2. Імпортування компонентів.

Після встановлення імпортовано необхідні компоненти з бібліотеки react-router-dom у відповідних файлах проекту. Найчастіше це були компоненти:

− BrowserRouter – корневий компонент, що оточує все застосування та забезпечує функціональність маршрутизації;

− Route – компонент, який визначає шлях і відповідний йому компонент для відображення;

− Switch – гарантує, що лише один маршрут буде відображатися одночасно;

− Link – компонент для створення посилань між різними маршрутами.

3. Налаштування маршрутів.

Наступним кроком було визначення маршрутів у застосунку. Було створено окремий файл Routes.js і визначено усі необхідні маршрути за допомогою компонентів Route і Switch. Кожен маршрут був пов'язаний з певним компонентом, який мав відображатися при переході на відповідний шлях.

4. Інтеграція в застосунок.

Після налаштування маршрутів було інтегровано React Router у основний компонент застосунку (App.js). Це включало оточення всього застосунку компонентом BrowserRouter і рендеринг визначених маршрутів за допомогою компонентів Switch і Route.

5. Навігація між маршрутами.

Для забезпечення навігації між різними маршрутами використовувався компонент Link з react-router-dom. Цей компонент дозволяє створювати посилання, які при натисканні не призводять до повного перезавантаження сторінки, а лише змінюють URL і відображають відповідний компонент.

6. Додаткові можливості.

React Router надає низку додаткових можливостей, таких як вкладені маршрути, маршрутизація на основі шаблонів, програмна навігація та захищені маршрути з перевіркою даних.

Належне налаштування React Router дозволило створити зручну навігацію в моєму веб-застосунку, забезпечити плавний перехід між різними розділами та компонентами без повного перезавантаження сторінки, що значно покращило користувацький досвід.

3.4.3 Підключення бібліотек

Для розширення функціональності клієнтської частини веб-застосунку було підключено та налаштовано кілька популярних бібліотек та фреймворків. Ці бібліотеки забезпечили додаткові можливості, покращили продуктивність, полегшили розробку та надали зручні інструменти для побудови інтерфейсу.

React Router React Router – це бібліотека для маршрутизації в React-застосунках. Вона дозволяє створювати одностранічні додатки (SPA) з можливістю переходу між різними компонентами без повного перезавантаження сторінки. Налаштування React Router включало встановлення бібліотеки за допомогою npm, імпортування необхідних компонентів, визначення маршрутів, інтеграцію маршрутизації в основний компонент застосунку (App.js) та налаштування навігації між маршрутами за допомогою компонента Link. Для підключення бібліотеки використовується команда:

```
npm install react-router-dom
```

Apollo Client Apollo Client – це бібліотека для взаємодії з GraphQL API у React-застосунках. Вона забезпечує зручний інтерфейс для виконання GraphQL-запитів, отримання та кешування даних, а також управління станом даних у додатку. Підключення Apollo Client включало встановлення бібліотеки через npm, налаштування клієнта в окремому файлі, імпортування необхідних хуків і компонентів у відповідні частини застосунку та використання їх для виконання GraphQL-запитів і відображення даних. Для підключення бібліотеки до проекту використовується команда:

```
npm install @apollo/client graphql
```

Material-UI Material-UI – це бібліотека компонентів інтерфейсу користувача, створена для React і заснована на принципах матеріального

дизайну від Google. Ця бібліотека надає готові стильові компоненти, такі як кнопки, поля введення, діалогові вікна, навігаційні панелі та багато іншого, що значно полегшує розробку інтерфейсу користувача.

Підключення Material-UI відбувалося через встановлення бібліотеки за допомогою npm, імпортування необхідних компонентів у відповідні файли застосунку та налаштування стилів і тем для забезпечення узгодженого візуального оформлення. Для підключення бібліотеки до проекту використовується команда [2]:

```
npm install @mui/material @emotion/react @emotion/styled @mui/icons-material
```

Redux Redux – це бібліотека для управління глобальним станом застосунку. Вона забезпечує передбачуваний потік даних, що полегшує відстеження змін стану та налагодження проблем. Підключення Redux включало встановлення бібліотеки та додаткових пакетів, таких як react-redux та redux-thunk, створення файлів для визначення дій, редюсерів і початкового стану, а також інтеграцію Redux у основний компонент застосунку за допомогою провайдера Redux. Для підключення бібліотеки до проекту використовується команда:

```
npm install react-redux
```

Використання цих бібліотек та фреймворків значно розширило можливості клієнтської частини веб-застосунку, забезпечило зручну маршрутизацію, ефективну взаємодію з GraphQL API, стильний та узгоджений інтерфейс користувача, а також надійне управління станом додатку. Це дозволило створити повноцінний, функціональний та зручний для користувачів веб-застосунок.

3.4.4 Інтеграція з Google OAuth

Для забезпечення безпечної аутентифікації користувачів у веб-застосунку було інтегровано механізм аутентифікації Google OAuth. Це дозволяє користувачам входити в систему за допомогою своїх облікових записів Google, що забезпечує зручність та безпеку процесу аутентифікації.

Процес інтеграції з Google OAuth включав наступні кроки:

1. Реєстрація проекту на Google Cloud Platform: Першим кроком було створення нового проекту на сторінці https://console.cloud.google.com. Після входу в консоль Google Cloud Platform, було створено новий проект, надавши йому унікальну назву “Rider google login” та налаштувавши необхідні параметри вказуючи необхідні дані для проекту.

2. Налаштування автентифікації в проекті Google Cloud: У розділі "APIs & Services" консолі Google Cloud Platform було активовано API "Google+ API" для використання OAuth-автентифікації. Також було налаштовано "OAuth consent screen" для проекту, вказавши необхідну інформацію, таку як ім'я додатку, електронну адресу підтримки та умови конфіденційності.

3. Створення облікових даних OAuth: У розділі "Credentials" консолі Google Cloud Platform було створено нові облікові дані OAuth для веб-додатку. Під час цього процесу було вказано URL-адресу, на якій запущено веб-застосунок, а також URL-адресу для перенаправлення (redirect) після успішної аутентифікації.

4. Отримання ключа клієнта OAuth: Після створення облікових даних OAuth, було отримано ключ клієнта OAuth, який містить ідентифікатор клієнта (client ID) та секретний ключ клієнта (client secret). Ці дані необхідні для інтеграції з механізмом автентифікації Google OAuth у веб-застосунку.

5. Підключення ключа OAuth у проекті: У вихідному коді клієнтської частини веб-застосунку було імпортовано бібліотеку для роботи з Google OAuth (react-google-login)[14]. Потім було налаштовано компонент для автентифікації, передавши йому ідентифікатор клієнта OAuth. Після успішної автентифікації

користувача, його облікові дані та токен доступу можна було використовувати для подальшої взаємодії з серверною частиною застосунку.

Для збереження чутливих даних проект містить директорію config в якій створено нову директорію oAuth2 з файлом google.js всередині де описано ідентифікатор клієнта та секретний ключ клієнта у наступному вигляді:

```
export const GOOGLE = {
    CLIENT_ID: <CLIENT_ID>,
    CLIENT_SECRET: <CLIENT_SECRET>
}
```

Інтеграція з Google OAuth забезпечила безпечний та зручний спосіб автентифікації користувачів у веб-застосунку, використовуючи їхні існуючі облікові записи Google. Це дозволило уникнути необхідності створення та підтримки окремої системи аутентифікації, покладаючись на надійні та перевірені механізми Google OAuth.

3.4.5 Обробка авторизації та аутентифікації користувачів

Після успішної інтеграції з Google OAuth, в клієнтській частині веб-застосунку було реалізовано обробку авторизації та аутентифікації користувачів. Цей процес включав наступні кроки:

1. Опрацювання відгуку Google OAuth: Після того, як користувач успішно увійшов через Google OAuth, було отримано відгук від Google, який містив облікові дані користувача та токен доступу. Цей токен доступу було збережено у локальному сховищі (localStorage) для подальшого використання.
2. Відправлення запиту на сервер для аутентифікації: Після отримання токена доступу від Google, було відправлено запит на серверну частину застосунку для аутентифікації користувача. У цьому запиті було передано токен доступу у заголовку Authorization.

Серверна частина перевіряє дійсність токена доступу та, у разі успішної аутентифікації, генерувала власний токен автентифікації для користувача. Цей токен було повернуто у відповіді на запит.

3. Збереження токена аутентификации: Отриманий від сервера токен автентифікації було збережено у локальному сховищі (localStorage) для подальшого використання у застосунку.

4. Обробка стану аутентифікації: Для відстеження стану аутентифікації користувача було використано Redux. У файлі loginSlice визначено початковий стан ідентифікації та дії для оновлення цього стану (наприклад, setLoginData).

5. Перевірка стану аутентифікації: При завантаженні застосунку та на відповідних маршрутах було перевірено стан аутентифікації користувача. Якщо користувач був аутентифікований (токен автентифікації присутній у локальному сховищі), відбулося перенаправлення на відповідну сторінку головну сторінку.

6. Використання токена аутентификации: Для захищених маршрутів та операцій, що вимагають аутентифікацію, в заголовках запитів до серверної частини було додано токен автентифікації з локального сховища. Серверна частина перевіряла дійсність цього токена та дозволяла або забороняє доступ до ресурсів залежно від статусу аутентифікації.

7. Інтеграція з Apollo Client: Для взаємодії з GraphQL API було використано бібліотеку Apollo Client. Під час налаштування Apollo Client було створено спеціальний ApolloLink для автоматичного додавання токена доступу у заголовок Authorization для всіх GraphQL-запитів.

Ретельна обробка авторизації та аутентифікації користувачів забезпечила безпечний доступ до функціональності та даних веб-застосунку. Використання токенів доступу та аутентифікації, а також їх збереження у локальному сховищі дозволило захистити чутливі ресурси від несанкціонованого доступу та надає користувачам безпечний і зручний спосіб входу в систему за допомогою їхніх облікових записів Google.

3.4.6 Управління станом авторизації (Redux)

У веб-застосунку було використано Redux для ефективного управління станом авторизації користувачів. Redux є широко використовуваною бібліотекою для управління глобальним станом додатку, що забезпечує передбачуваний потік даних та полегшує відстеження змін стану.

Реалізація управління станом авторизації за допомогою Redux включала наступні кроки:

1. Створення Redux-редюсера для авторизації (loginSlice):

– визначено початковий стан авторизації (initialState) як null;

– створено редюсер setLoginData для оновлення стану авторизації з даними про користувача після успішної авторизації.

2. Налаштування Redux-сховища (store):

– імпортовано створений loginReducer;

– додано loginReducer до об'єкту редюсерів у функції configureStore.

3. Обробка дій авторизації:

– у відповідних місцях застосунку (наприклад, після успішної авторизації через Google OAuth) було відправлено дію setLoginData з даними про користувача за допомогою диспетчера Redux (dispatch).

4. Доступ до даних авторизації – створено селектори для отримання даних про стан авторизації з Redux-сховища, такі як:

– selectLoginData: повертає весь об'єкт даних авторизації;

– selectLoginName: повертає ім'я авторизованого користувача;

– selectLoginPicture: повертає URL-адресу зображення профілю користувача;

– isLoggedIn: перевіряє, чи користувач авторизований;

– selectUserId: повертає ідентифікатор користувача.

5. Використання даних авторизації в компонентах:

– у відповідних компонентах застосунку було імпортовано необхідні селектори та використано функцію useSelector з бібліотеки react-redux для отримання даних про стан авторизації з Redux-сховища;

– ці дані було використано для відображення інформації про користувача, перевірки статусу авторизації та управління доступом до різних функцій застосунку.

Управління станом авторизації за допомогою Redux забезпечило централізоване та передбачуване управління даними про авторизацію користувачів у веб-застосунку. Це полегшило відстеження та оновлення стану авторизації, а також забезпечило зручний доступ до даних про авторизацію в різних частинах застосунку. Використання Redux також сприяло кращій модульності та можливості повторного використання коду, оскільки дані про авторизацію були відокремлені від компонентів інтерфейсу користувача.

3.4.7 Розробка компонентів інтерфейсу

Для створення сучасного та інтерактивного інтерфейсу користувача використано бібліотеку React. React забезпечує ефективну та гнучку розробку інтерфейсів користувача, дозволяючи створювати компоненти, які можуть бути повторно використані та вдосконалені. Це, в свою чергу, сприяє кращій підтримці та легшому внесенню змін.

Під час розробки були створені власні компоненти, які візуалізують дані та взаємодію. До них відносяться:

1. Navigation – Компонент для навігації та організації переміщення між сторінками. Він дозволяє створювати інтерактивні навігаційні панелі з кнопками, навігаційними смугами прогресу тощо.
2. Map – Компонент для відображення google карти на сторінці.
3. POI – Компонент для відображення елементів, які знаходяться на карті, їх інформацію та автора.
4. Logo – Компонент для демонстрації логотипу веб-сайту.

5. Place – Компонент для відображення інформації цікавих місць з динамічної карти.

6. Profile – Компонент для відображення персонального кабінету користувача з детальною інформацією про нього.

7. User – Компонент для відображення даних користувача та взаємодії з ними.

3.4.8 Стилізація компонентів

У процесі розробки інтерфейсу одним з ключових аспектів було забезпечення належного візуального представлення компонентів та узгодженого стилю у всьому додатку. Для досягнення цієї мети було використано потужність CSS у поєднанні з бібліотекою Sass, що надає зручний синтаксис для написання складних стилів.

Для кожного компонента було створено окремий файл стилів, де визначався візуальний вигляд компонента, його розміри, кольори, шрифти та інші аспекти оформлення. Ці файли стилів були спеціально створені для певного компонента, забезпечуючи модульність та відокремлення логіки компонента від його представлення. Наприклад, для компонента Navigation було створено окремий файл стилів navigation.scss, де було детально описано візуальний вигляд та поведінку цього компонента.

Окрім задання стилів безпосередньо для компонентів, в проекті було створено спеціальний каталог assets/styles, який містив загальні стилі та налаштування для всього додатку. У цьому каталозі знаходився файл base.scss, де були визначені базові стилі для елементів інтерфейсу, такі як кольори, шрифти та розміри. Також у цьому каталозі зберігаються файли з описами кольорів, які могли бути імпортовані та використані в різних частинах додатка, забезпечуючи узгодженість оформлення.

Для імпортування необхідних стилів у компонентах було використано синтаксис, подібний до наступного:

```
import "./navigation.scss";
import "./assets/styles/colors.scss";
```

Це дозволяє компонентам отримувати доступ до необхідних стилів та забезпечувало модульність та впорядкованість коду.

Завдяки використанню Sass та його можливостей із імпортування і модульності, розробка інтерфейсу компонентів стала більш структурованою та зручною. Це забезпечило легке підтримування коду, повторне використання стилів та узгоджене оформлення в усьому додатку.

3.4.9 Реалізація маршрутизації

В даному веб-застосунку для реалізації маршрутизації використовується бібліотека React Router. Ця бібліотека забезпечує гнучку і зрозумілу систему маршрутизації, яка дозволяє визначати різні маршрути для відображення відповідних компонентів.

Маршрутизація налаштовується в головному компоненті App.jsx, де оголошується BrowserRouter з бібліотеки react-router-dom. Всередині BrowserRouter визначаються Routes та окремі маршрути за допомогою компонента Route. Наприклад, в App.jsx маршрути визначені наступним чином:

```
<BrowserRouter>
  <Navigation />
  <Routes>
    <Route path="/*" element={<HomePage />} />
    <Route path="/login" element={<LoginPage />} />
  </Routes>
  <Footer />
</BrowserRouter>
```

У цьому випадку, маршрут "/" та всі інші маршрути, що не вказані явно, будуть відображати компонент HomePage. Маршрут "/login" буде відображати компонент LoginPage.

Усередині компонента HomePage також визначені додаткові маршрути для відображення різних компонентів:

```
<Routes>
  <Route path="/profile/*" element={<Profile />} />
</Routes>
<Map />
<Routes>
  <Route path="/" element={<POIList />} />
      <Route path="/profile" element={<POIList userId={userId}
link={"/profile"} />} />
  <Route path="/:id" element={<POIDetails />} />
   <Route path="/profile/:id" element={<POIDetails form={false}/>}
/>
  <Route path="/edit" element={<POIForm />} />
</Routes>
```

Тут визначено маршрути для відображення компонентів Profile, Map, POIList, POIDetails та POIForm. Деякі маршрути використовують динамічні параметри, наприклад, /:id, що дозволяє передавати ідентифікатор як частину URL-адреси.

Для навігації між маршрутами можна використовувати компонент Link з бібліотеки react-router-dom або хук useNavigate для програмної навігації.

Реалізація маршрутизації за допомогою React Router забезпечує чітке розділення компонентів відповідно до URL-адрес, що полегшує розробку і підтримку веб-застосунку. Це також забезпечує зручну навігацію для користувачів і можливість створювати односторінкові додатки (SPA) з плавною зміною вмісту без перезавантаження сторінки.

3.4.10 Налаштування Apollo Client

У веб-застосунку використовується Apollo Client для взаємодії з GraphQL API. Apollo Client є популярною бібліотекою, яка забезпечує зручний спосіб виконання GraphQL-запитів і керування даними в React-застосунках [12].

Налаштування Apollo Client відбувається в головному файлі застосунку, де створюється екземпляр ApolloClient з необхідними налаштуваннями. Створення HTTP-з'єднання:

```
const httpLink = new HttpLink({
  uri: `${URL.SERVER.HOST + URL.SERVER.ENDPOINT.GRAPHQL}`,
  credentials: "include",
});
```

У цій частині коду створюється екземпляр HttpLink, який встановлює з'єднання з GraphQL API за вказаною URI-адресою. Параметр credentials: "include" дозволяє включати cookie-файли у запити, що може знадобитися для автентифікації користувача. Створення AuthLink:

```
const authLink = new ApolloLink((operation, forward) => {
  const accessToken = localStorage.getItem("accessToken");
  if (accessToken) {
    operation.setContext({
      headers: {
        Authorization: `Bearer ${accessToken}`,
      },
    });
  return forward(operation);
});
```

Ця частина коду створює ApolloLink з назвою authLink. Цей посередник перевіряє, чи є в локальному сховищі (localStorage) токен доступу (accessToken). Якщо він присутній, то додає його до заголовка Authorization у контексті запиту.

Створення екземпляра ApolloClient:

```
const client = new ApolloClient({
  link: from([authLink, httpLink]),
  cache: new InMemoryCache(),
});
```

Тут створюється екземпляр ApolloClient з необхідними налаштуваннями. Параметр link містить масив посередників, які будуть застосовуватися до всіх запитів. У цьому випадку це authLink та httpLink. Також налаштовується кеш InMemoryCache для зберігання даних, отриманих від GraphQL API.

Після створення екземпляру ApolloClient, його необхідно надати у провайдері ApolloProvider, щоб компоненти могли отримати доступ до функціональності Apollo Client. Це робиться у головному компоненті застосунку (main.jsx):

```
<ApolloProvider client={client}>
  <React.StrictMode>
    <App />
  </React.StrictMode>
</ApolloProvider>
```

Таким чином, усі компоненти, вкладені в ApolloProvider, матимуть доступ до функціональності Apollo Client і зможуть виконувати GraphQL-запити та взаємодіяти з даними, отриманими з GraphQL API.

3.4.11 Визначення GraphQL запитів та мутацій

У веб-застосунку GraphQL-запити та мутації визначаються у окремих файлах для зручності їх підтримки та повторного використання. Наприклад, згідно з наданою інформацією, запити визначені у файлі config/graphql/query.js, а мутації – у файлі config/graphql/mutation.js.

У цих файлах використовується синтаксис tagged template literals для визначення GraphQL-запитів та мутацій. Імпорт модуля gql з @apollo/client:

```
import { gql } from "@apollo/client";
```

У цій частині було імпортовано функцію gql з бібліотеки @apollo/client. Ця функція використовується для визначення GraphQL-запитів у вигляді tagged template literals.

Визначення запиту ALL_POIS:

```
export const QUERY = {
  ALL_POIS: gql`
    query allPois($userId: String, $type: String) {
      pois(userId: $userId, type: $type) {
          id, type, types, title, address, description, imageUrls, createdAt, coordinates {lat, lng}, user {id, name, username}, likes, dislikes, feedbacks { feedback}
      }
    }`
};
```

У цій частині було визначено GraphQL-запит `ALL_POIS`, який дозволяє отримати список точок інтересу (POIs), відфільтрованих за ідентифікатором користувача та типом. Запит повертає інформацію про кожну точку, включаючи її ідентифікатор, тип, назву, адресу, опис, зображення, дату створення, координати, інформацію про користувача, кількість лайків і дизлайків, а також список відгуків.

Визначення запиту POI:

```
QUERY: {
  POI: gql`
    query poi($id: ID!) {
      poi(id: $id) {
```

```
                id, type, types, title, address, description, imageUrls,
createdAt, coordinates {lat, lng}, user {id name username}, likes,
dislikes, feedbacks { id, description, feedback, createdAt, user {name}}
            }
          }`,
      };
```

У цій частині було визначено GraphQL-запит `POI`, який дозволяє отримати детальну інформацію про одну конкретну точку інтересу за її ідентифікатором. Запит повертає інформацію про точку, включаючи її ідентифікатор, тип, назву, адресу, опис, зображення, дату створення, координати, інформацію про користувача, кількість лайків і дизлайків, а також список відгуків із додатковою інформацією про кожен відгук.

Визначення запиту USER:

```
      QUERY: {
        USER: gql`
          query user($id: ID!) {
               user(id: $id) { id, email, name, username, picture,
friends { id, name, username, picture}}
          }`,
      };
```

У цій частині було визначено GraphQL-запит `USER`, який дозволяє отримати детальну інформацію про користувача за його ідентифікатором. Запит повертає інформацію про користувача, включаючи його ідентифікатор, електронну пошту, ім'я, ім'я користувача, зображення профілю, а також список його друзів із їхньою інформацією.

Ці запити можуть бути використані у компонентах застосунку за допомогою хука useQuery з Apollo Client.

Аналогічно, у файлі mutation.js визначено мутації. Визначення мутації NEW_POI:

```
export const MUTATION = {
  NEW_POI: gql`
    mutation addPoi(
            $type: String!, $title: String!, $address: String!,
$description: String, $coordinates: [CoordinateInput!]!, $imageUrls:
[String!], $userId: ID!, $types: [String!]) {
          addPoi(type: $type, types: $types, title: $title, address:
$address, description: $description, imageUrls: $imageUrls, coordinates:
$coordinates, userId: $userId) {id}
    }`
};
```

У цій частині було визначено GraphQL-мутацію NEW_POI, яка дозволяє додавати нову точку інтересу (POI) на карту. Мутація приймає такі параметри: тип, назву, адресу, опис, координати, URL-и зображень, ідентифікатор користувача та список типів. Повертає ідентифікатор доданої точки.

Визначення мутації NEW_FEEDBACK:

```
export const MUTATION = {
  NEW_FEEDBACK: gql`
     mutation addFeedback($description: String!, $feedback: String!,
$poiId: ID!, $userId: ID!) {addFeedback(description: $description,
feedback: $feedback, poiId: $poiId, userId: $userId) {id, description,
feedback, createdAt, user { name }}}`
};
```

У цій частині було визначено GraphQL-мутацію NEW_FEEDBACK, яка дозволяє додавати новий відгук на існуючу точку інтересу. Мутація приймає такі параметри: опис, текст відгуку, ідентифікатор точки та ідентифікатор користувача. Повертає інформацію про доданий відгук, включаючи його ідентифікатор, опис, текст, дату створення та ім'я користувача.

Визначення мутації CHANGE_USERNAME:

```
export const MUTATION = {
  CHANGE_USERNAME: gql`
    mutation changeUsername($id: ID!, $username: String! ) {
      changeUserUsername(
          id: $id, username: $username) {id, email, name, username,
picture, friends { id, name, username, picture }}}`
};
```

У цій частині було визначено GraphQL-мутацію CHANGE_USERNAME, яка дозволяє змінити псевдонім користувача. Мутація приймає ідентифікатор користувача та новий псевдонім. Повертає оновлену інформацію про користувача, включаючи ідентифікатор, електронну пошту, ім'я, псевдонім, зображення профілю та список друзів.

Мутації можуть бути використані у компонентах за допомогою хука useMutation з Apollo Client.

Визначення запитів та мутацій у окремих файлах забезпечує модульність та полегшує їх повторне використання в різних частинах застосунку. Це також сприяє кращій підтримці коду, оскільки всі GraphQL-запити та мутації зібрані в одному місці.

3.4.12 Обробка даних, отриманих від серверної частини

Обробка даних, отриманих від серверної частини, є важливою складовою розробки клієнтської частини веб-застосунку. Основна мета цього процесу полягає в тому, щоб забезпечити коректне відображення та використання інформації, яка надходить від GraphQL API. У цьому розділі розглядаються ключові аспекти цієї обробки, зокрема робота з хукамі useQuery та useMutation з бібліотеки Apollo Client, а також управління станом даних та обробка помилок.

При використанні Apollo Client у React-застосунку, основними інструментами для отримання та обробки даних є хуки useQuery та useMutation. Хук useQuery дозволяє виконувати запити до GraphQL API та отримувати

необхідні дані. Наприклад, для отримання списку точок інтересу (POIs) використовується наступний код:

```
import { useQuery } from "@apollo/client";
import { QUERY } from "./config/graphql/query";

const { loading, error, data } = useQuery(QUERY.ALL_POIS, {
  variables: { userId, type },
});
```

У цьому прикладі хук useQuery виконує запит ALL_POIS, передаючи необхідні змінні для фільтрації результатів. Параметри loading, error та data використовуються для управління станом запиту та обробки результатів. Якщо запит знаходиться у стані завантаження, параметр loading матиме значення true, що дозволяє відображати індикатор завантаження для покращення користувацького досвіду. У разі виникнення помилки параметр error міститиме інформацію про цю помилку, що дозволяє обробляти її належним чином та надавати користувачеві відповідне повідомлення.

Дані, отримані від GraphQL API, зберігаються у параметрі data. Наприклад, якщо запит ALL_POIS успішно виконався, параметр data міститиме список точок інтересу. Ці дані можна використовувати для відображення у відповідних компонентах застосунку. Важливою частиною обробки даних є нормалізація та трансформація інформації для зручності її використання у компоненті. Це може включати форматування дат, об'єднання або фільтрацію масивів даних, а також інші операції, необхідні для забезпечення коректного відображення інформації.

Аналогічно, хук useMutation використовується для виконання мутацій до GraphQL API, що дозволяє змінювати дані на сервері. Наприклад, для додавання нової точки інтересу можна використовувати наступний код:

```
import { useMutation } from "@apollo/client";
```

```
import { MUTATION } from "./config/graphql/mutation";
const [addPoi, { loading, error, data }] = useMutation(MUTATION.NEW_POI);
const handleAddPoi = async (poiData) => {
  try {
    const { data } = await addPoi({ variables: { ...poiData } });
  } catch (error) {}
};
```

У цьому прикладі хук useMutation створює функцію addPoi, яка використовується для виконання мутації NEW_POI. Параметри loading, error та data аналогічно використовуються для управління станом мутації та обробки результатів. Функція handleAddPoi викликає мутацію з переданими змінними, обробляючи успішний результат або помилку відповідним чином.

Окрім базових аспектів використання хукамі useQuery та useMutation, обробка даних включає управління станом застосунку. Використовуючи кеш Apollo Client, можна забезпечити зберігання та повторне використання отриманих даних, що зменшує кількість запитів до серверної частини та покращує продуктивність застосунку. Для цього використовується InMemoryCache, який автоматично оновлюється при виконанні запитів та мутацій.

Важливим аспектом обробки даних є обробка помилок, що виникають під час запитів або мутацій. Для цього можна використовувати onError з Apollo Link, що дозволяє централізовано обробляти всі помилки, які виникають під час взаємодії з GraphQL API. Це забезпечує покращення користувацького досвіду шляхом надання зрозумілих повідомлень про помилки та можливість повторного виконання запитів або мутацій.

Таким чином, обробка даних, отриманих від серверної частини, є комплексним процесом, що включає використання хукамі useQuery та useMutation, управління станом даних за допомогою кешу Apollo Client, а також обробку помилок для забезпечення надійності та продуктивності

веб-застосунку. Ці аспекти сприяють створенню зручного та ефективного інтерфейсу для користувачів, забезпечуючи коректне відображення та використання інформації, що надходить від GraphQL API.

3.4.13 Інтеграція бібліотеки для відображення карти

Використання Google Maps дозволяє відображати інтерактивні карти, що значно підвищує функціональність і зручність використання застосунку.

Для відображення карти використовується бібліотека google-map-react, яка забезпечує просту інтеграцію Google Maps API з React. Основний компонент карти (Map.jsx) містить необхідні налаштування для ініціалізації карти та управління її поведінкою.

Компонент Map містить основну логіку для взаємодії з картою. Він імпортує всі необхідні залежності, включаючи бібліотеку google-map-react, стилі, компоненти для відображення місць на карті, а також Redux для управління станом [10].

Ключовим аспектом є налаштування початкових параметрів карти, таких як центр і рівень масштабу. Для цього у компоненті визначено об'єкт defaultProps, який містить координати центру та початковий рівень зуму [9]:

```
const defaultProps = {
  center: { lat: 48.9354547453022, lng: 24.700544141400652 },
  zoom: 16,
};
```

Функція handleApiLoaded викликається при успішному завантаженні API Google Maps і дозволяє виконувати додаткові налаштування карти, такі як додавання панелей управління та налаштування ліній на карті. У цьому випадку до карти додаються ліві та праві панелі, а також створюється полілінія, яка може бути редагована користувачем у режимі редагування:

```
const handleApiLoaded = async (map, maps) => {
  const leftPanel = document.querySelector(".left-panel");
  const rightPanel = document.querySelector(".right-panel");
  map.controls[maps.ControlPosition.LEFT].push(leftPanel);
  map.controls[maps.ControlPosition.RIGHT].push(rightPanel);

  polyLineRef.current = new maps.Polyline();
  mapRef.current = map;
  mapsRef.current = maps;
   map.addListener("click",   (event)   =>   mapClickHandler(event,
editMode));
};
```

Додавання нової точки до полілінії:

```
const path = polyLineRef.current.getPath();
path.push(new mapsRef.current.LatLng(latLng.lat(), latLng.lng()));
```

Після отримання координат кліку, ця частина коду додає нову точку до поточної полілінії, використовуючи метод push() об'єкту path.

Створення нового маркера на карті:

```
new mapsRef.current.Marker({
  position: latLng,
  title: "#" + path.getLength(),
  map: mapsRef.current,
});
```

Ця частина коду створює новий маркер на карті, використовуючи координати кліку, як позицію маркера. Також встановлюється заголовок маркера, який відображає порядковий номер точки в полілінії.

Оновлення стану полілінії:

```
const newPathArray = path.getArray().map((latLng) => ({lat: latLng.lat(), lng: latLng.lng()}));
dispatch(setNewPath(newPathArray));
```

Нарешті, ця частина коду перетворює масив точок полілінії у масив об'єктів {lat, lng} і відправляє його до Redux-стору за допомогою дії setNewPath.

Для відображення конкретних місць на карті використовується компонент Place, який відображає маркер на відповідних координатах. Дані про місця завантажуються з Redux і відображаються на карті за допомогою компонента GoogleMap. Отримання даних про місця з Redux:

```
const places = useSelector(getPOIPlaces);
```

У цій частині коду отримуються дані про місця (POIs) з Redux-стору за допомогою селектора getPOIPlaces.

Рендеринг компонента GoogleMap:

```
<GoogleMap
  options={mapOption}
  hoverDistance={20}
  bootstrapURLKeys={{ key: map.key }}
  defaultCenter={defaultProps.center}
  defaultZoom={defaultProps.zoom}
  yesIWantToUseGoogleMapApiInternals
  onGoogleApiLoaded={({ map, maps }) => handleApiLoaded(map, maps)}
  onClick={(event) => mapClickHandler(event, editMode)}
  onChildClick={childClickHandler}
  onChange={changeHandler}
>
  {/* Рендеринг компонентів Place */}
</GoogleMap>
```

Ця частина відповідає за рендеринг компонента GoogleMap, який відображає карту. Він отримує різні параметри, такі як опції карти, ключ API Google Maps, початкові координати та масштаб. Також визначені обробники подій, такі як клік по карті, клік по дочірньому елементу (наприклад, маркеру) та зміна карти.

Рендеринг компонентів Place:

```
{places?.map((place) => (
  <Place
    key={place.id}
    lat={place.coordinates[0].lat}
    lng={place.coordinates[0].lng}
    text={place.title}
    place={place}
  />
))}
```

Ця частина відповідає за рендеринг компонентів Place для кожного місця, отриманого з Redux-стору. Кожен Place компонент відображає маркер на карті з відповідними координатами, назвою та іншими даними про місце.

Інтеграція бібліотеки Google Maps надає багаті можливості для відображення та взаємодії з картографічними даними. Використання компонентів React та Redux дозволяє забезпечити зручну та ефективну роботу з картою, включаючи динамічне завантаження та відображення місць, управління полілініями та обробку подій користувача. Цей підхід дозволяє створити інтуїтивно зрозумілий та функціональний інтерфейс для користувачів веб-застосунку.

3.4.14 Відображення маркерів на карті

У цьому розділі розглянуто процес відображення маркерів на карті за допомогою бібліотеки Google Maps у клієнтській частині веб-застосунку.

Маркери використовуються для позначення точок інтересу (POI) на карті, забезпечуючи користувачів інформацією про різні типи місць, такі як майстерні, ресторани, місця для відпочинку, паркінги та місця з інтенсивним рухом.

Основним компонентом для відображення маркерів є компонент Place. Цей компонент відповідає за відображення іконки, що позначає тип місця, та забезпечує взаємодію користувача з маркером. У компоненті використовуються іконки з бібліотеки Material-UI для візуального представлення різних типів місць:

```
import BuildIcon from "@mui/icons-material/Build";
import RestaurantIcon from "@mui/icons-material/Restaurant";
import BedIcon from "@mui/icons-material/Bed";
import LocalParkingIcon from "@mui/icons-material/LocalParking";
import TrafficIcon from "@mui/icons-material/Traffic";
import { Link, useNavigate } from "react-router-dom";
import "./place.scss";
import { useSelector } from "react-redux";
import { getZoomLevel } from "../../features/map/mapSlice";
import React, { useEffect, useState } from "react";
import ShortInfo from "./ShortInfo";
```

Іконки для різних типів місць зберігаються у об'єкті placeTypes, що дозволяє легко отримувати потрібну іконку за типом місця:

```
const placeTypes = {
  workshop: <BuildIcon className="icon" />,
  food: <RestaurantIcon className="icon" />,
  rest: <BedIcon className="icon" />,
  park: <LocalParkingIcon className="icon" />,
  traffic: <TrafficIcon className="icon" />,
};
```

Компонент Place отримує властивості place, text та $hover. Властивість $hover використовується для визначення, чи знаходиться курсор миші над маркером, що дозволяє відображати додаткову інформацію про місце. Використання хука useState дозволяє керувати станом відображення цієї інформації:

```
export default function Place(props) {
  const { $hover, place, text } = props;
  const [showInfo, setShowInfo] = useState(false);
  const navigate = useNavigate();
  const zoom = useSelector(getZoomLevel);
   const iconStyles = {width: `${zoom * 1.2}px`, height: `${zoom * 1.2}px`,};
  const handleMouseEnter = (e) => {setShowInfo(true)};
  const handleMouseLeave = (e) => {setShowInfo(false)};
  if (!place) return null;
```

Для забезпечення взаємодії користувача з маркером, використовується компонент Link з бібліотеки React Router. Це дозволяє користувачам переходити до детальної інформації про місце при натисканні на маркер. Використання компонента Link з React Router [11]:

```
 <Link
  to={`/${place.id}`}
  className="place-container"
  onMouseEnter={handleMouseEnter}
  onMouseLeave={handleMouseLeave}
>
  {/* Вміст компонента */}
</Link>
```

Цей код використовує компонент Link з бібліотеки React Router для створення посилання на сторінку з детальною інформацією про місце. Атрибут

to визначає шлях до цієї сторінки, який складається з ідентифікатора місця. Також визначені обробники подій onMouseEnter та onMouseLeave, які використовуються для відображення додаткової інформації про місце.

Відображення списку типів місця:

```
<ul className="list-types">
  {place.types.map((placeType) => (
    <li className="type-el" key={placeType}>
              {React.cloneElement(placeTypes[placeType], { style: iconStyles })}
    </li>
  ))}
</ul>
```

Ця частина коду відображає список типів місця у вигляді списку. Для кожного типу місця створюється відповідний елемент списку, який містить іконку, яка відображає тип місця. Іконки отримуються з об'єкта placeTypes, а їхній стиль задається за допомогою iconStyles.

Відображення додаткової інформації:

```
{showInfo && <ShortInfo place={place} />}
```

Ця частина коду відображає додаткову інформацію про місце, якщо змінна showInfo має значення true. Додаткова інформація відображається за допомогою компонента ShortInfo, який отримує дані про місце як значення.

Компонент Place також використовує хук useSelector для отримання поточного рівня зуму з Redux, що дозволяє динамічно змінювати розмір іконок в залежності від рівня зуму карти. Це забезпечує кращу візуалізацію маркерів при зміні масштабу карти:

```
const zoom = useSelector(getZoomLevel);
const iconStyles = {
```

```
    width: `${zoom * 1.2}px`,
    height: `${zoom * 1.2}px`,
  };
```

Таким чином, компонент Place забезпечує відображення маркерів на карті, використовуючи іконки для різних типів місць, динамічно змінює розмір іконок в залежності від рівня зуму та дозволяє користувачам взаємодіяти з маркерами для отримання детальної інформації про місце. Це робить карту більш інтерактивною та інформативною, покращуючи загальний користувацький досвід (UX).

3.4.15 Обробка взаємодії користувача з картою

Взаємодія користувача з картою є важливим аспектом розробки клієнтської частини веб-застосунку, що використовує Google Maps. Зручність і інтуїтивність такої взаємодії значно впливають на користувацький досвід, забезпечуючи ефективну роботу з картографічними даними. У цьому розділі розглянуто основні методи та підходи до обробки взаємодії користувача з динамічною картою.

Основний компонент Map відповідає за відображення карти та обробку різних подій, що виникають під час взаємодії користувача з нею. Однією з ключових функцій є обробка кліків по карті. Це дозволяє користувачам додавати нові точки на карті у режимі редагування. Функція mapClickHandler обробляє ці події, додаючи нові координати до поточного шляху та відображаючи їх на карті у вигляді маркерів:

```
const mapClickHandler = useCallback(
  (event, editMode) => {
    if (!editMode) return;
    let latLng;
    if (event.latLng) {
      latLng = {lat: event.latLng.lat(),lng: event.latLng.lng()};
```

```
    } else {
      latLng = {lat: event.lat, lng: event.lng};
    }
  },[polyLineRef, mapsRef, dispatch]
);
```

Цей обробник події кліку на карті викликається, коли користувач натискає на карту в режимі редагування. Він отримує координати кліку та зберігає їх у змінній latLng. Інший важливий аспект взаємодії користувача з картою — це обробка кліків по лініях, які відображають маршрути на карті. Функція handlePolylineClick перенаправляє користувача до сторінки з детальною інформацією про обраний маршрут:

```
const handlePolylineClick = (polyline, path) => {
  navigate(`/${path.id}`);
};
```

Для покращення візуального зворотного зв’язку при взаємодії з полілініями використовуються функції handlePolylineHover та handlePolylineMouseOut. Ці функції змінюють стиль полілінії при наведенні курсору миші та повертають його до початкового стану при відведенні курсору:

```
const handlePolylineHover = (polyline) => {
  polyline.setOptions({
    strokeColor: COLOR.PRIMARY,
    strokeWeight: 8,
  });
};
const handlePolylineMouseOut = (polyline) => {
  polyline.setOptions({
    strokeColor: COLOR.BICYCLE_PATH,
    strokeWeight: 4,
  });
};
```

Додатково, важливою функціональністю є обробка зміни масштабу карти. Функція changeHandler оновлює рівень зуму в стані застосунку при зміні масштабу карти:

```
const changeHandler = (event) => {
  dispatch(setZoomLevel(event.zoom));
};
```

Також, для забезпечення інтерактивності при відображенні точок інтересу на карті використовується компонент Place, який відповідає за відображення інформації про місця при наведенні на них курсору. Використовується функціональність onMouseEnter та onMouseLeave, яка контролює відображення додаткової інформації про місце:

```
const handleMouseEnter = (e) => {
  setShowInfo(true);
};
const handleMouseLeave = (e) => {
  setShowInfo(false);
};
```

Таким чином, обробка взаємодії користувача з картою включає різні аспекти, такі як обробка кліків по карті та полілініях, зміна масштабу карти та відображення інформації про місця. Використання цих методів забезпечує інтуїтивний та зручний користувацький досвід, дозволяючи користувачам ефективно взаємодіяти з картографічними даними у веб-застосунку.

3.4.16 Відображення повідомлень та сповіщень

Відображення повідомлень та сповіщень є важливою складовою будь-якого сучасного веб-застосунку. Вони надають користувачам зворотний

зв'язок про виконання різних дій, такі як успішна авторизація, помилки під час виконання запитів або інші важливі події. У цьому розділі розглянуто, як реалізовано відображення повідомлень та сповіщень у клієнтській частині веб-застосунку за допомогою бібліотеки react-toastify.

Для реалізації сповіщень використовується компонент ToastContainer з бібліотеки react-toastify, який дозволяє легко налаштовувати сповіщення та відображати їх у різних частинах застосунку. Основні налаштування компонента ToastContainer включають позицію сповіщення, тривалість його відображення, можливість закриття сповіщення кліком та інші параметри [3]:

```
import { ToastContainer, toast } from "react-toastify";
import "react-toastify/dist/ReactToastify.css";
<ToastContainer
  position="top-right"
  autoClose={5000}
  draggable
  pauseOnHover
/>
```

Основний компонент застосунку App включає ToastContainer для відображення сповіщень у верхньому правому куті екрану. Всі зміни та необхідні повідомлення для користувача відображаються за допомогою об’єкту toast та методом для виклику сповіщення. Наприклад після успішної авторизації відбувається збереження токена доступу в локальному сховищі та надсилання запиту до серверної частини для отримання додаткової інформації про користувача. Якщо авторизація проходить успішно, користувачу відображається сповіщення про успішний вхід:

```
toast.success("Logged in successfully");
```

У разі виникнення помилки під час авторизації або запиту до серверної частини відображається сповіщення про невдалу спробу входу:

```
toast.error("Failed to log in");
```

Ці сповіщення забезпечують користувачів зворотним зв'язком про результати їхніх дій, що значно покращує взаємодію з застосунком. Використання react-toastify дозволяє легко та швидко інтегрувати сповіщення в будь-яку частину застосунку, забезпечуючи зручний та інтуїтивно зрозумілий інтерфейс для користувачів.

Таким чином, відображення повідомлень та сповіщень у веб-застосунку за допомогою бібліотеки react-toastify є ефективним способом забезпечення зворотного зв'язку для користувачів. Це допомагає створити більш інформативний та інтерактивний інтерфейс, що сприяє покращенню загального користувацького досвіду.

3.4.17 Хостинг клієнтської частини

Після розробки клієнтської частини веб-застосунку виникло питання її розгортання та хостингу. Для швидкого та зручного розгортання було обрано сервіс ngrok, який дозволяє створювати безпечні тунелі до локально запущених веб-додатків, надаючи їм публічні URL-адреси. Використання ngrok має кілька переваг, зокрема можливість миттєвого розгортання для тестування та демонстрації без необхідності налаштування власного публічного хостингу на цьому етапі розробки.

Процес хостингу клієнтської частини за допомогою ngrok включає декілька кроків:

1. Встановлення ngrok: Перш за все, необхідно встановити ngrok. Це можна зробити, завантаживши ngrok з офіційного веб-сайту ngrok.com або за допомогою пакетного менеджера (наприклад, Homebrew для macOS).

2. Запуск локального серверу: Клієнтська частина зазвичай запускається на локальному сервері, наприклад, за допомогою команди npm

start для React-застосунків. Це запускає локальний сервер, зазвичай доступний за адресою http://localhost:5173.

```
npm start
```

3. Запуск ngrok: Після запуску локального серверу необхідно створити тунель до цього серверу за допомогою ngrok. Це робиться за допомогою команди:

```
ngrok http 5173
```

Ця команда створює тунель до локального сервера на порту 5173 і надає публічний URL-адрес, за якою можна отримати доступ до клієнтської частини з Інтернету. Приклад вихідних даних після запуску команди (рис. 3.4.1).

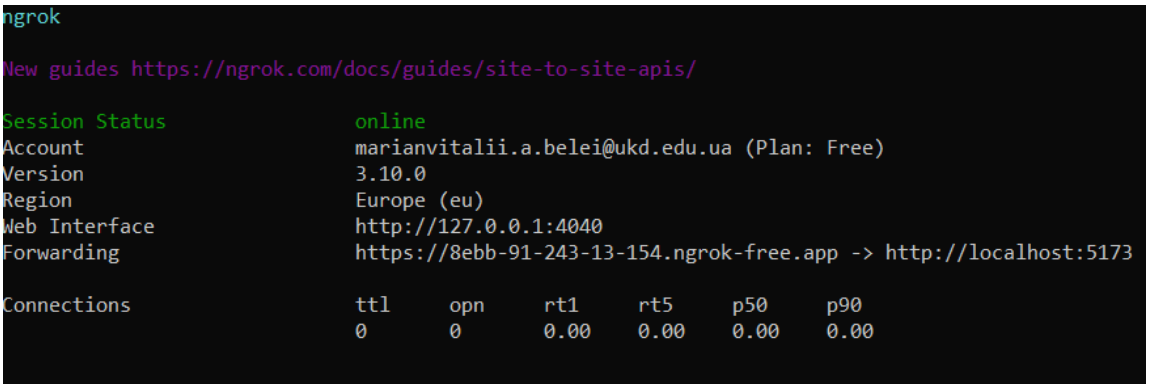


Рисунок 3.4.1 – Приклад вихідних даних після запуску хостингу ngrok

4. Отримання публічної URL-адреси: Після запуску ngrok, можна використовувати наданий публічний URL (наприклад, https://256d-91-243-13-154.ngrok-free.app) для доступу до клієнтської частини застосунку. Цей URL можна використовувати для тестування або демонстрації застосунку іншим користувачам.

Використання ngrok для хостингу клієнтської частини дозволяє миттєво розгорнути застосунок для тестування та демонстрації. Це особливо корисно на

ранніх етапах розробки, коли налаштування власного хостингу може бути зайвим і трудомістким процесом. Ngrok надає простий та ефективний спосіб надати доступ до локально запущених веб-застосунків з Інтернету, що значно полегшує процес тестування та зворотного зв'язку.

Таким чином, використання ngrok для хостингу клієнтської частини забезпечує зручне та швидке рішення для розгортання та тестування веб-застосунку, дозволяючи зосередитися на розробці та вдосконаленні функціональності без зайвих витрат часу та ресурсів на налаштування інфраструктури хостингу.

Висновок до розділу 3

У третьому розділі було детально розглянуто практичну реалізацію сервісу з інтерактивною картою для велосипедистів. Розпочато з розробки дизайну та користувацького інтерфейсу, де було приділено увагу ключовим аспектам, таким як зручність навігації, акцент на карті, логічне групування функцій, використання зрозумілих візуальних підказок та адаптивність. Для створення дизайну було обрано Figma – потужний інструмент для веб-дизайну, що дозволив ефективно спроектувати високоякісний інтерфейс з урахуванням потреб цільової аудиторії.

Загалом, третій розділ детально описує практичну реалізацію сервісу з інтерактивною картою для велосипедистів. Він охоплює ключові аспекти, такі як розробка дизайну та користувацького інтерфейсу, архітектура бази даних, створення серверної частини на основі Ruby on Rails та GraphQL, а також розробку клієнтської частини з використанням React.js, популярних бібліотек та інтеграцією Google Maps. Особливу увагу приділено забезпеченню безпеки, зручності та ефективності взаємодії користувачів з сервісом. Це закладає міцний фундамент для подальшого розвитку та вдосконалення функціональності веб-застосунку.

ВИСНОВОК

У ході виконання даної кваліфікаційної роботи було розроблено сервіс з інтерактивною картою для велосипедистів. Основною метою проекту було створення зручного та функціонального веб-застосунку, який дозволяє користувачам ефективно взаємодіяти з картографічними даними, планувати маршрути та отримувати інформацію про велосипедну інфраструктуру.

Для досягнення цієї мети було проведено ретельний аналіз існуючих рішень, виявлено їхні переваги та недоліки. На основі цього аналізу було сформульовано функціональні вимоги до майбутнього сервісу та визначено ключові аспекти, які потребували подальшого вивчення та розробки.

У процесі розробки використовувалися сучасні технології та підходи, такі як Ruby on Rails, MongoDB, React.js, Google Maps API, GraphQL та інші. Це дозволило створити продукт з інтуїтивно зрозумілим інтерфейсом, високою продуктивністю та надійністю. Особлива увага приділялася забезпеченню безпеки, зручності та інтерактивності сервісу, що дозволяє користувачам ефективно взаємодіяти з картографічними даними та планувати свої маршрути.

Розроблений сервіс має ряд переваг. Він є зручним у використанні та може бути легко розширений новими функціями. Це робить його привабливим для широкого кола велосипедистів, які можуть отримувати актуальну інформацію про велоінфраструктуру та планувати свої поїздки незалежно від їхнього місцезнаходження. Сервіс також забезпечує високу продуктивність, стабільність та безпеку, що є важливими характеристиками для подібних веб-застосунків.

Практичне значення отриманих результатів полягає у можливості використання розробленого сервісу як основи для створення інших інноваційних рішень, спрямованих на підвищення безпеки та зручності велосипедного руху. Крім того, напрацювання, отримані в ході виконання кваліфікаційної роботи, можуть бути застосовані для вдосконалення існуючих

картографічних сервісів та розробки нових інтерактивних додатків для велосипедистів та інших учасників дорожнього руху.

Загалом, виконана кваліфікаційна робота демонструє ефективне поєднання сучасних підходів до розробки веб-застосунків, глибоке розуміння предметної області та прагнення створити якісний продукт, що відповідає потребам цільової аудиторії. Отримані результати можуть бути корисними як для подальшого розвитку даного сервісу, так і для розробки інших веб-застосунків, орієнтованих на покращення велосипедної інфраструктури та підвищення безпеки велосипедного руху.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. React.js Documentation. URL: https://reactjs.org/docs/getting-started.html (дата звернення: 12.10.2023).

2. Material-UI Documentation. URL: https://mui.com/material-ui/getting-started/installation/ (дата звернення: 15.10.2023).

3. react-toastify Documentation. URL:-https://fkhadra.github.io/react-toastify/introduction (дата звернення: 02.11.2023).

4. Ruby on Rails Guides. URL: https://guides.rubyonrails.org/ (дата звернення: 25.11.2023).

5. MongoDB Documentation. URL: https://www.mongodb.com/docs/ (дата звернення: 25.11.2023).

6. MongoDB Atlas Documentation. URL: https://www.mongodb.com/atlas/database (дата звернення: 26.11.2023).

7. GraphQL Documentation. URL: https://graphql.org/learn/ (дата звернення: 02.01.2024).

8. Figma Documentation. URL: https://www.figma.com/best-practices/ (дата звернення: 11.10.2023).

9. Google Maps Platform Documentation. URL: https://developers.google.com/maps/documentation (дата звернення: 25.01.2024).

10. google-map-react Documentation. URL: https://github.com/google-map-react/google-map-react (дата звернення: 25.01.2024).

11. React Router Link Documentation. URL: https://reactrouter.com/en/6.23.1/components/link (дата звернення: 24.01.2024).

12. Apollo Client Documentation. URL: https://www.apollographql.com/docs/react/ (дата звернення: 25.02.2024).

13. Redux Documentation. URL: https://redux.js.org/introduction/getting-started (дата звернення: 27.11.2023).

14. react-google-login Documentation. URL: https://www.npmjs.com/package/react-google-login (дата звернення: 01.03.2024).

15. Ngrok Documentation. URL: https://ngrok.com/docs (дата звернення: 20.05.2024).

16. Mongoid Documentation. URL: https://docs.mongodb.com/mongoid/ (дата звернення: 28.11.2023).

17. graphql-ruby Documentation. URL: https://graphql-ruby.org/getting_started.html (дата звернення: 02.01.2024).

18. Redux Toolkit Documentation. URL: https://redux-toolkit.js.org/introduction/getting-started (дата звернення: 28.11.2023).

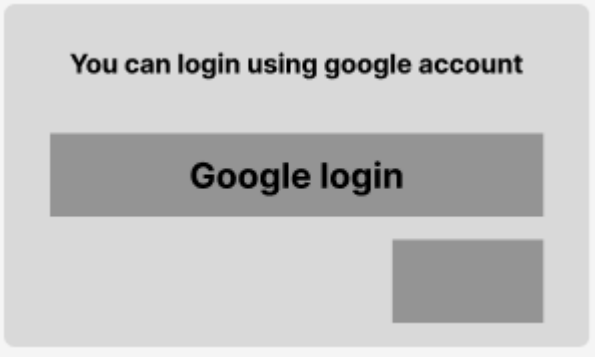
19. Vite Documentation. URL: https://vitejs.dev/guide/ (дата звернення: 12.11.2023).

20. ESLint Documentation. URL: https://eslint.org/docs/user-guide/getting-started (дата звернення: 12.11.2023).

ДОДАТКИ

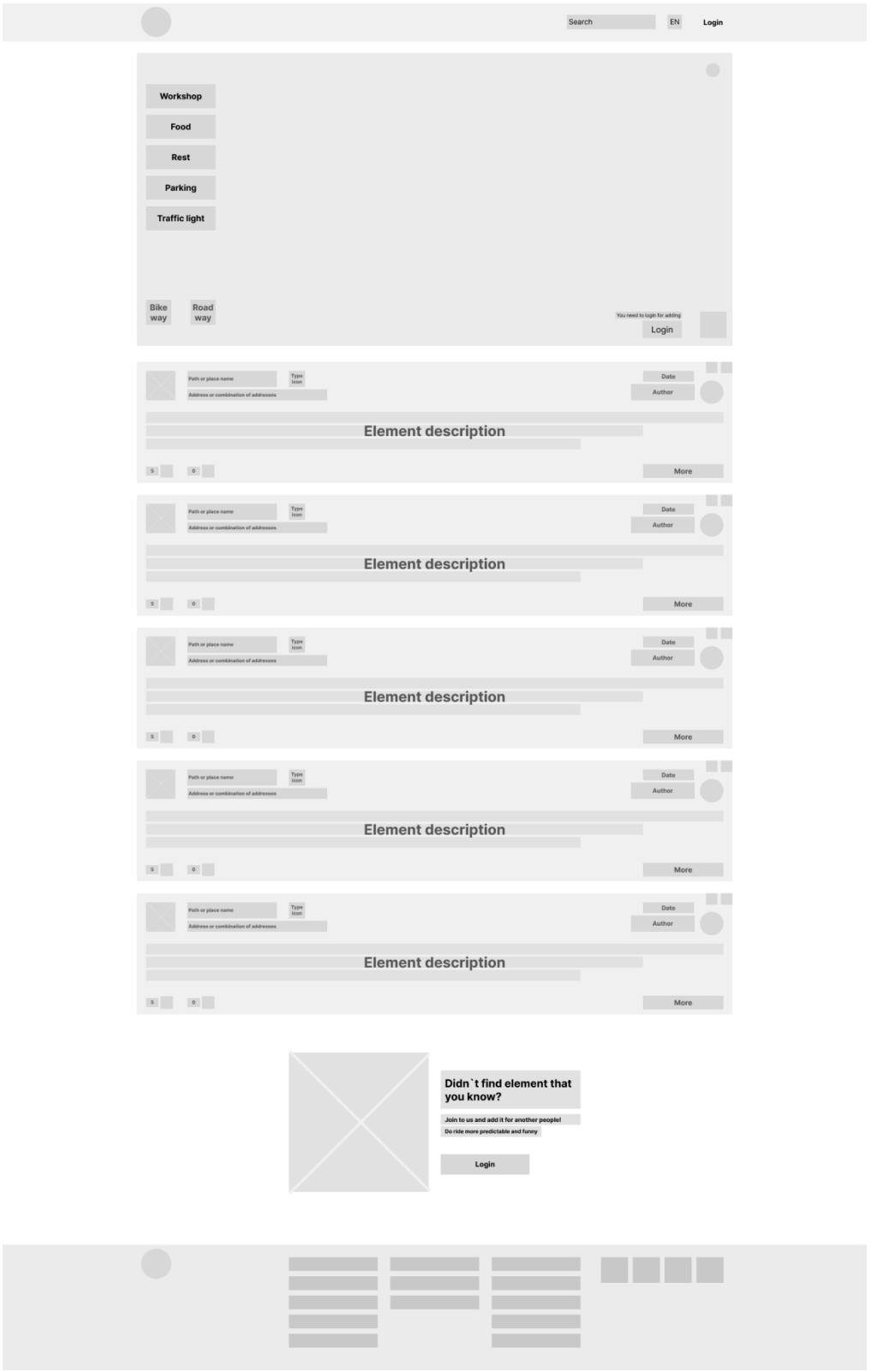
Додаток А

Wireframe схема компоненту логіну



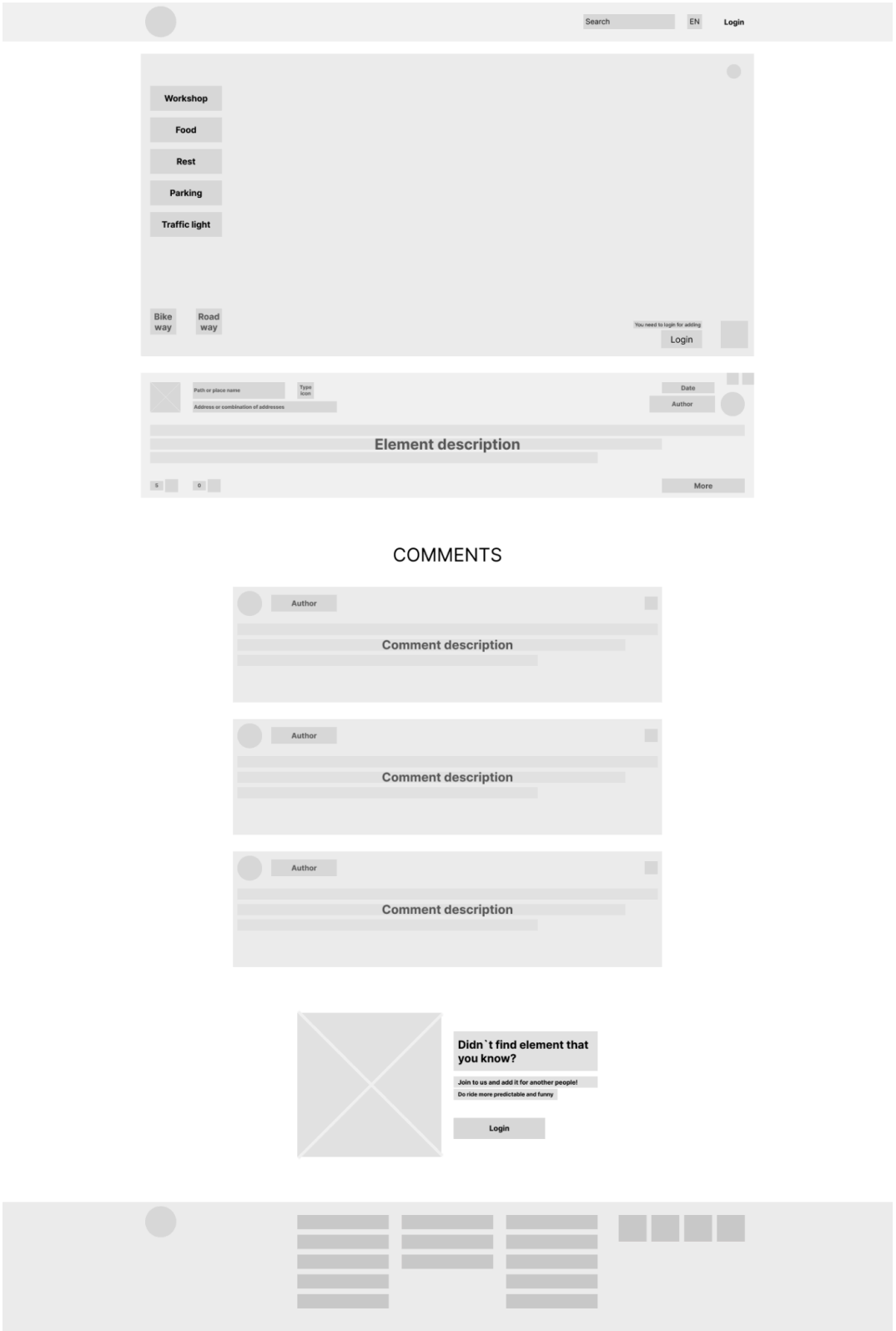
Додаток Б

Wireframe схема сторінки з картою та списком цікавих місць для незареєстрованих користувачів



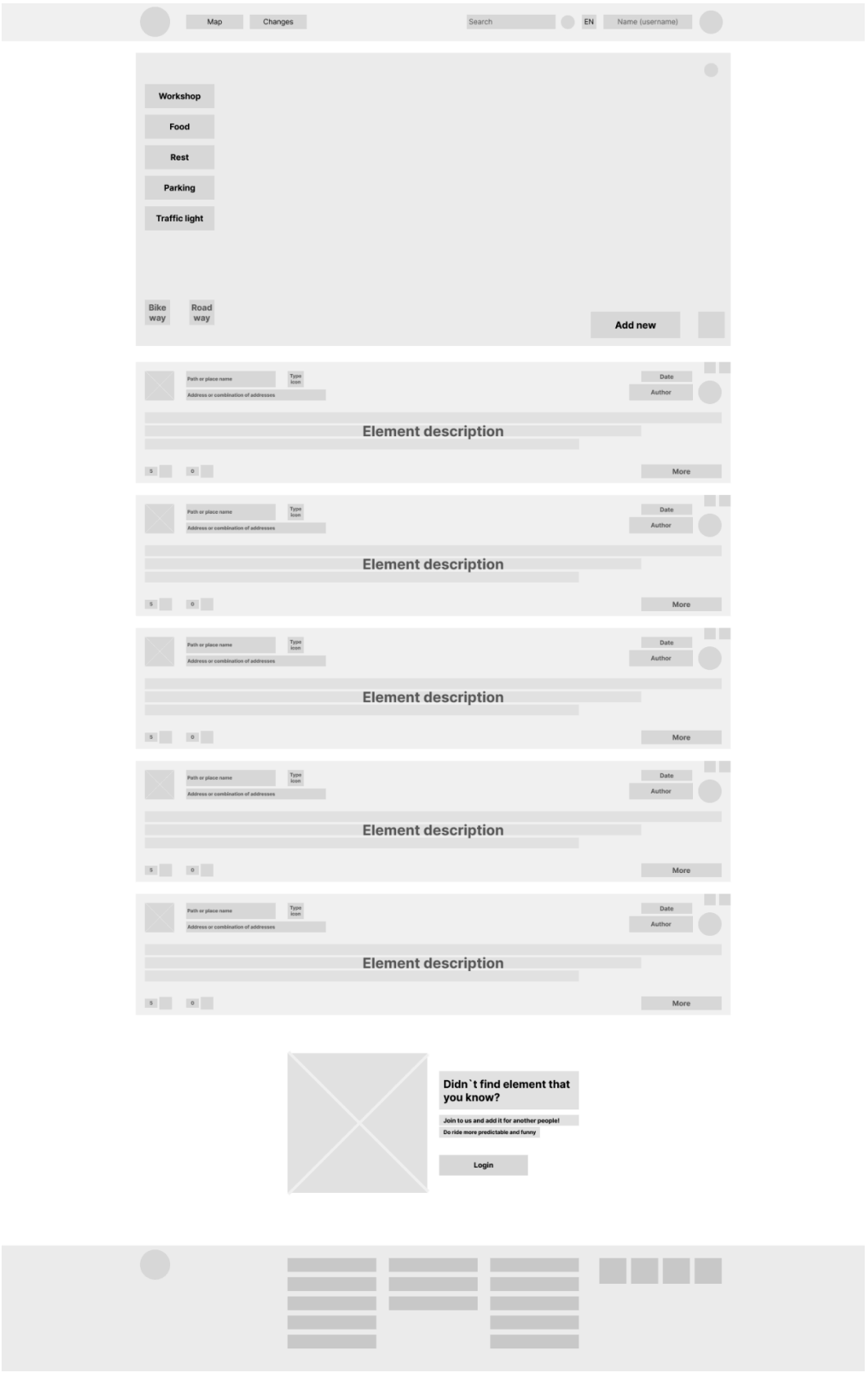
Додаток В

Wireframe схема сторінки з описом певного цікавого місця для незареєстрованих користувачів



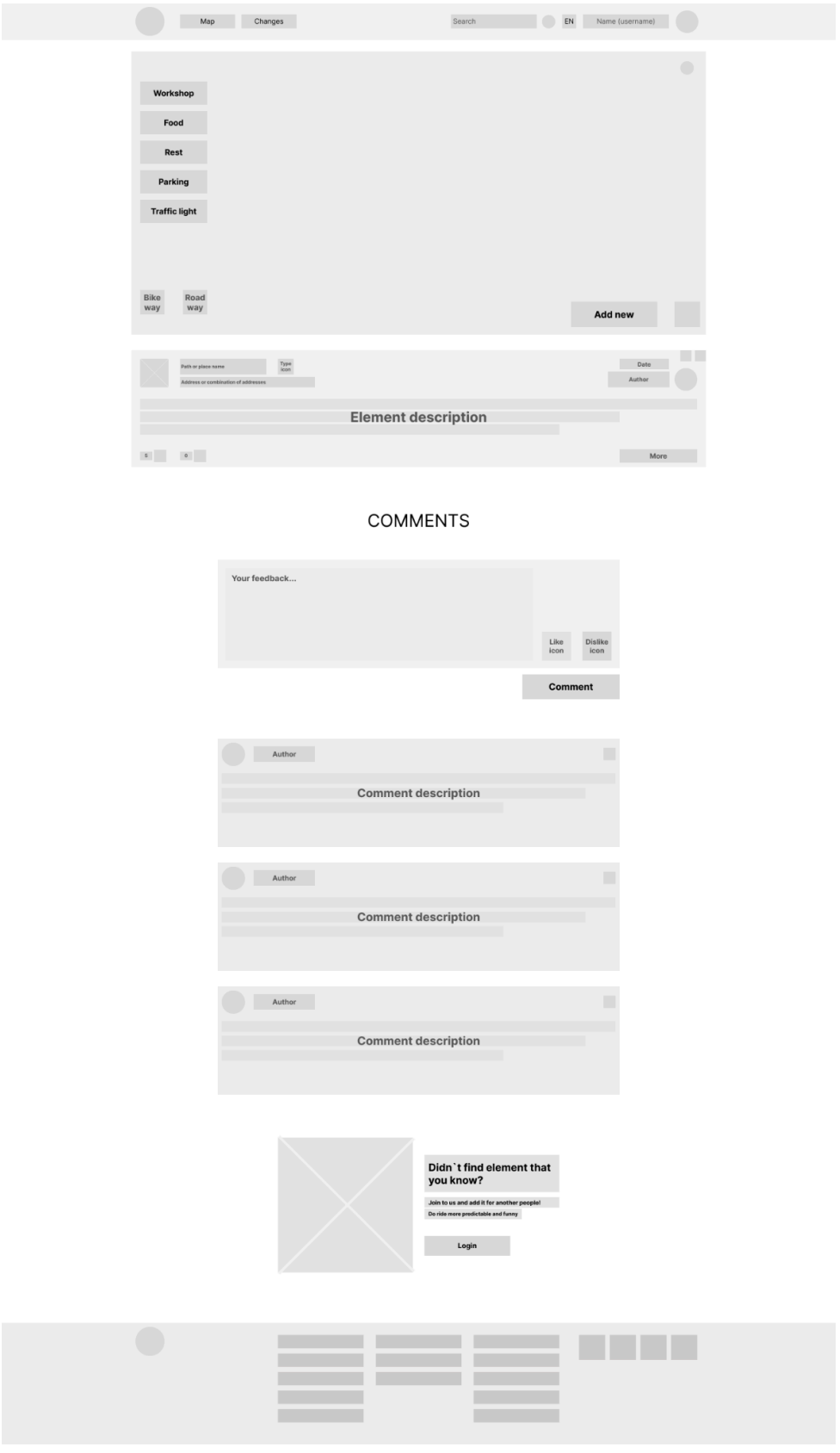
Додаток Д

Wireframe схема сторінки з картою та списком цікавих місць для зареєстрованих користувачів



Додаток Е

Wireframe схема сторінки з описом певного цікавого місця для зареєстрованих користувачів

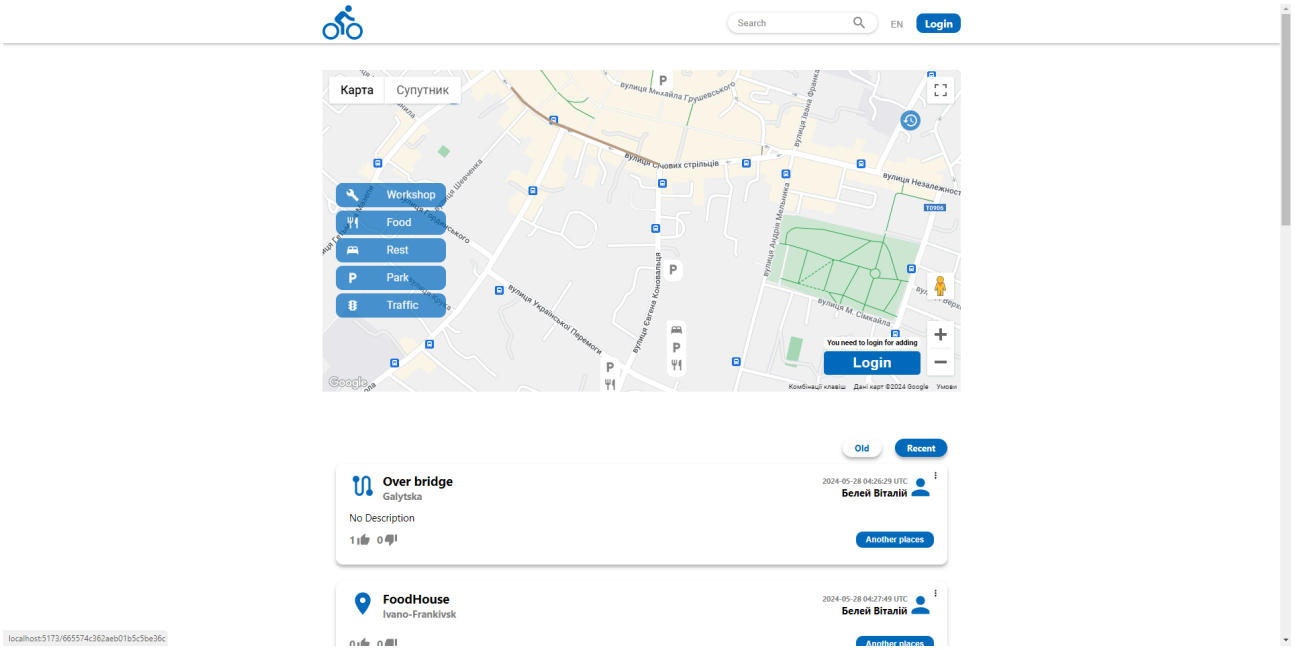


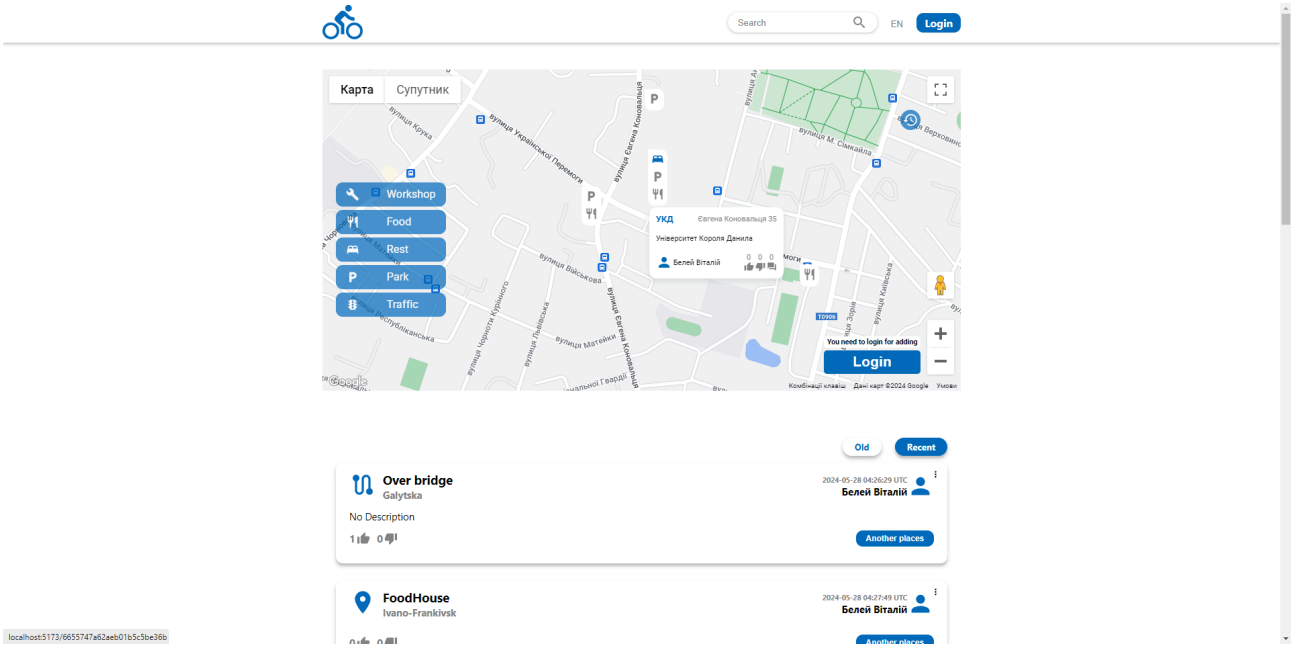
Додаток Ж

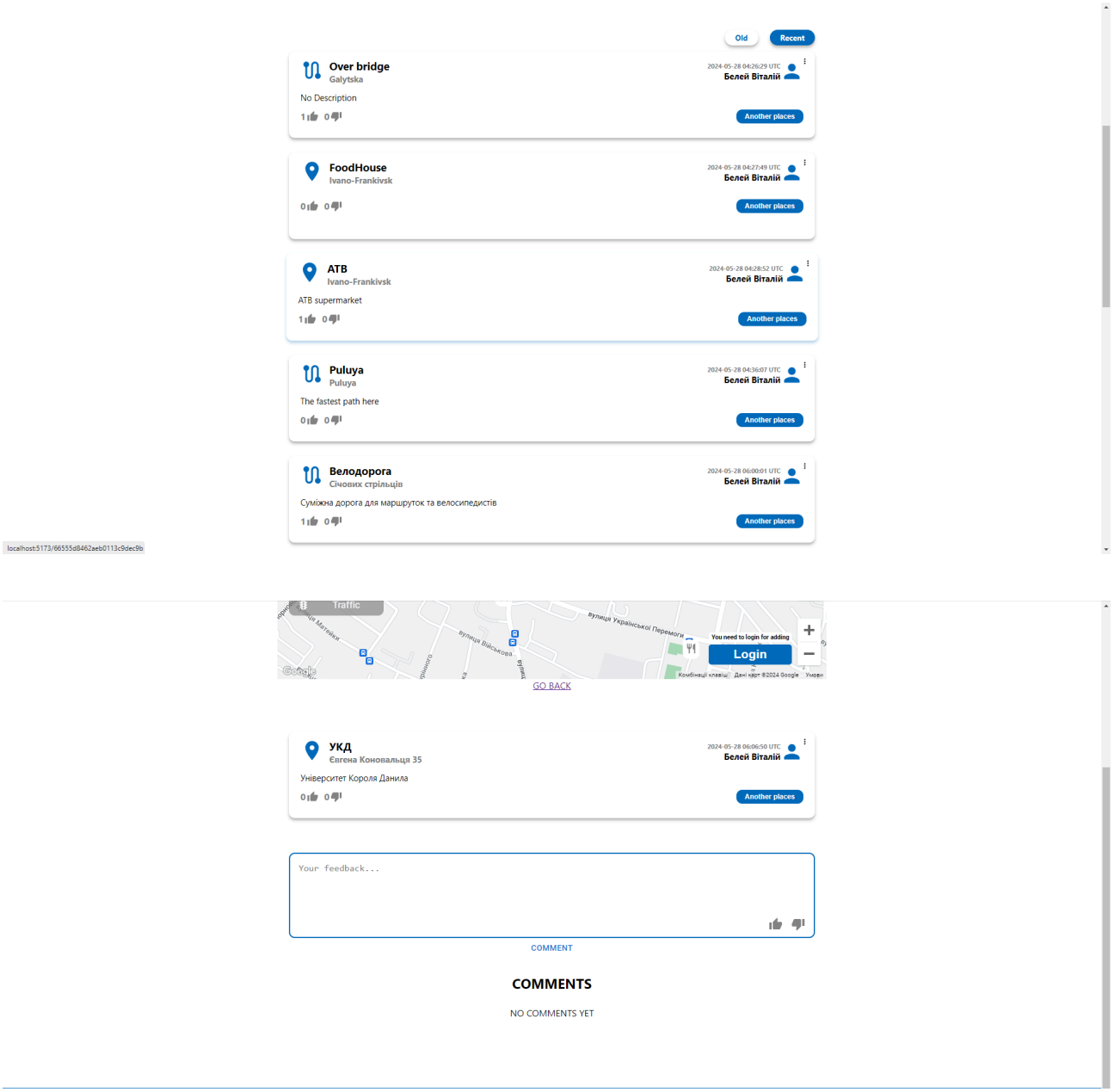
Wireframe схема сторінки персонального кабінету користувача

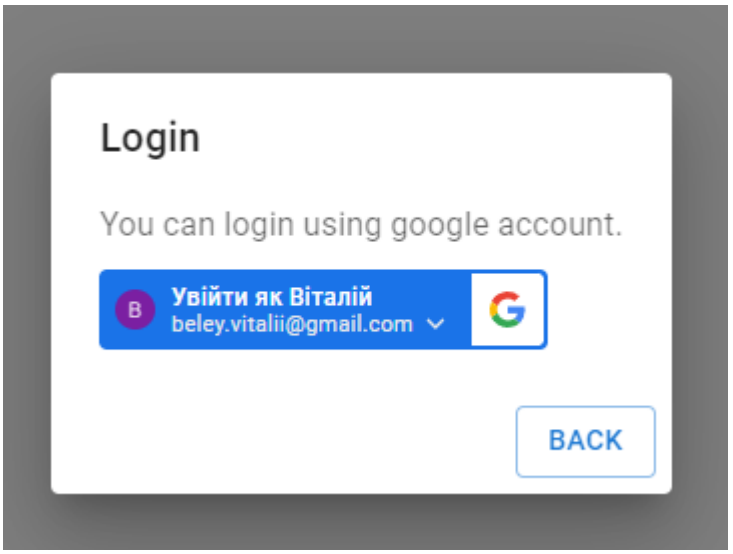
Додаток З

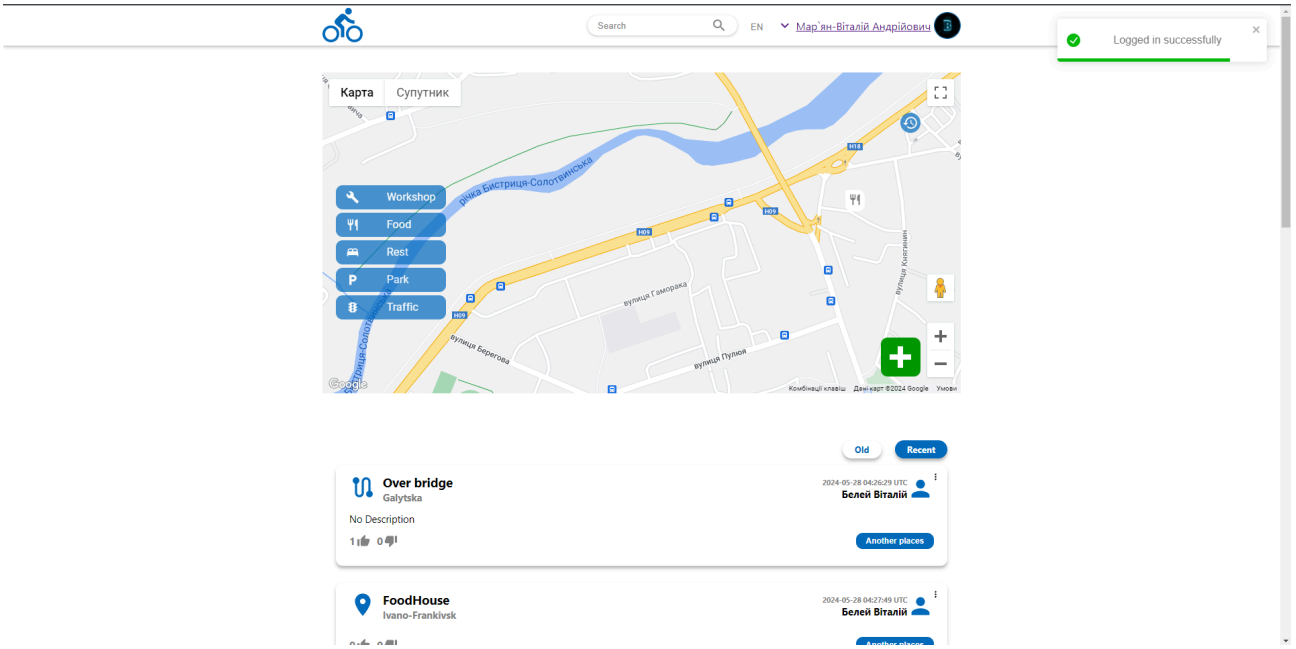
Демонстрація роботи сервісу

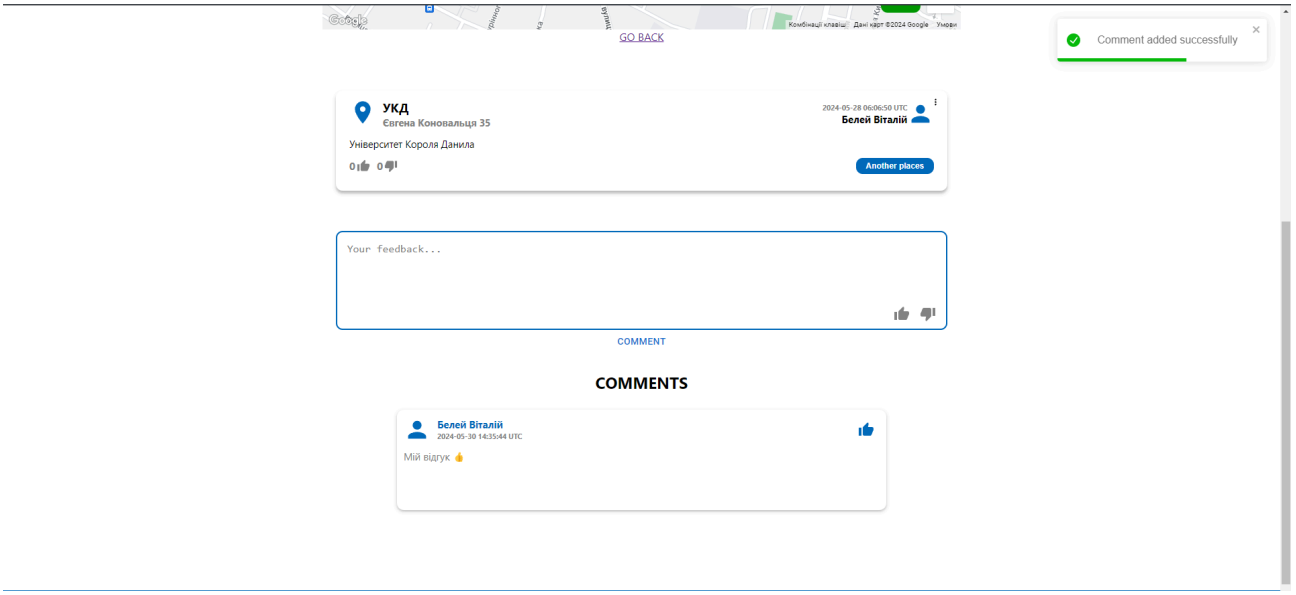





Old
Recent
Over bridge
Galytska
2024-05-28 04:26:29 UTC
Белей Віталій
No Description
1 0
Another places
FoodHouse
Ivano-Frankivsk
2024-05-28 04:27:49 UTC
Белей Віталій
0 0
Another places
ATB
Ivano-Frankivsk
2024-05-28 04:28:52 UTC
Белей Віталій
ATB supermarket
1 0
Another places
Puluya
Puluya
2024-05-28 04:36:07 UTC
Белей Віталій
The fastest path here
0 0
Another places
Велодорога
Січових стрільців
2024-05-28 06:00:01 UTC
Белей Віталій
Суміжна дорога для маршруток та велосипедистів
1 0
Another places
localhost:5173/66555d8462aeb0113c9dec9b
Traffic
вулиця Української Перемоги
вулиця Військова
You need to login for adding
Login
GO BACK
УКД
Євгена Коновальця 35
2024-05-28 06:06:50 UTC
Белей Віталій
Університет Короля Данила
0 0
Another places
Your feedback...
COMMENT
COMMENTS
NO COMMENTS YET


Login
You can login using google account.
Увійти як Віталій
beley.vitalii@gmail.com
BACK


Search
EN
Мар'ян-Віталій Андрійович
Logged in successfully
Карта
Супутник
Workshop
Food
Rest
Park
Traffic
річка Бистриця-Солотвинська
вулиця Гамората
вулиця Берегова
вулиця Пулюя
Old
Recent
Over bridge
Galytska
2024-05-28 04:26:29 UTC
Белей Віталій
No Description
1 0
Another places
FoodHouse
Ivano-Frankivsk
2024-05-28 04:27:49 UTC
Белей Віталій


GO BACK
Comment added successfully
УКД
Євгена Коновальця 35
2024-05-28 06:06:50 UTC
Белей Віталій
Університет Короля Данила
0 0
Another places
Your feedback...
COMMENT
COMMENTS
Белей Віталій
2024-05-30 14:35:44 UTC
Мій відгук

Title
Address

Describe this point of interest...

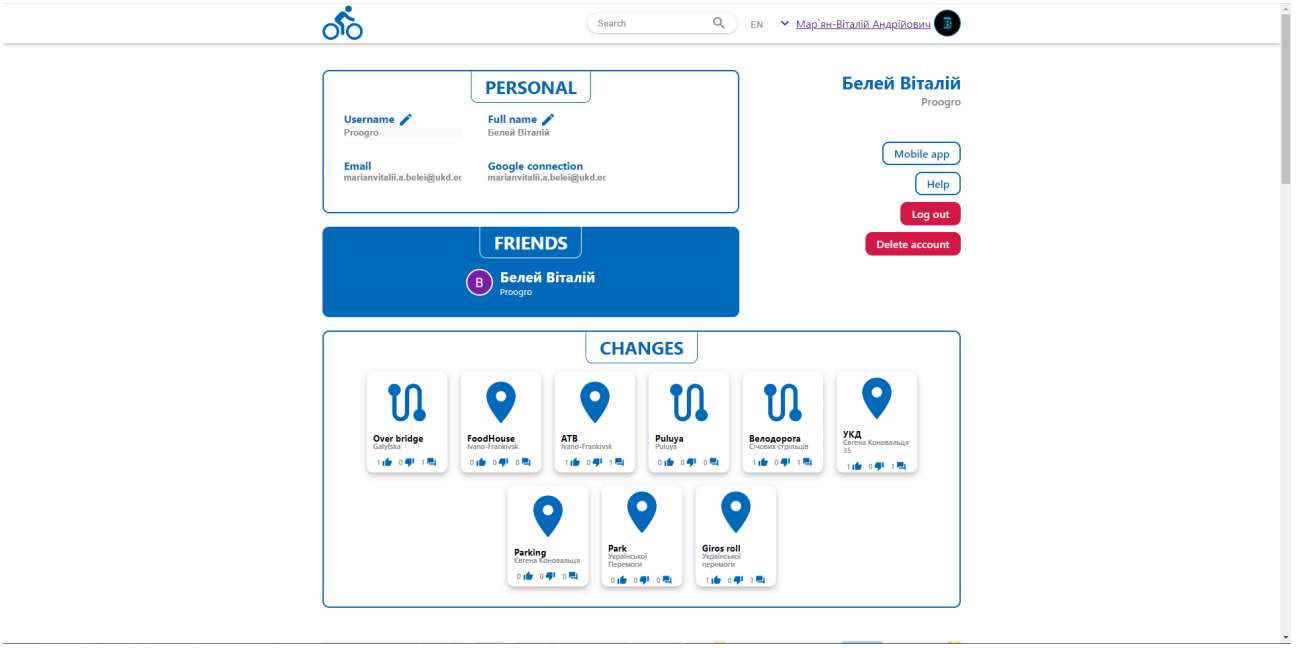
CREATE

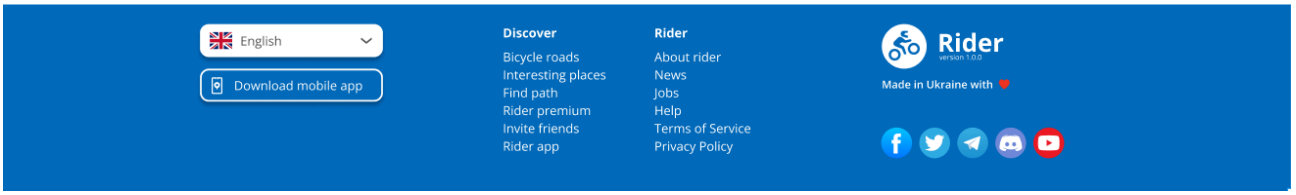
Title
Address

Describe this point of interest...

workshop
food
rest
park
traffic

CREATE


Search
EN
Мар'ян-Віталій Андрійович
PERSONAL
Username
Proogro
Full name
Белей Віталій
Email
marianvitalii.a.belei@ukd.ec
Google connection
marianvitalii.a.belei@ukd.ec
FRIENDS
Белей Віталій
Proogro
Белей Віталій
Proogro
Mobile app
Help
Log out
Delete account
CHANGES
Over bridge
Galytska
FoodHouse
Ivano-Frankivsk
ATB
Ivano-Frankivsk
Puluya
Puluyi
Велодорога
Очеван стрільця
УКД
Євгена Коновальця
35
Parking
Євгена Коновальця
Park
Української Перемоги
Giros roll
Української перемоги


English
Download mobile app
Discover
Bicycle roads
Interesting places
Find path
Rider premium
Invite friends
Rider app
Rider
About rider
News
Jobs
Help
Terms of Service
Privacy Policy
Rider
version 1.0.0
Made in Ukraine with

метадані

Заголовок

Сервіс з динамічною та інтерактивною картою для велосипедистів

Автор
Белей М-В.

Науковий керівник / Експерт
кандидат технічних наук Сергій Ващишак

підрозділ
King Danylo University

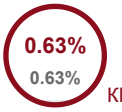
Тривога

У цьому розділі ви знайдете інформацію щодо текстових спотворень. Ці спотворення в тексті можуть говорити про МОЖЛИВІ маніпуляції в тексті. Спотворення в тексті можуть мати навмисний характер, але частіше характер технічних помилок при конвертації документа та його збереженні, тому ми рекомендуємо вам підходити до аналізу цього модуля відповідально. У разі виникнення запитань, просимо звертатися до нашої служби підтримки.

Заміна букв	ß	1
Інтервали	A→	0
Мікропробіли	·	0
Білі знаки	ß	0
Парафрази (SmartMarks)	a	6

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.



КП 1

КП 2

25
Довжина фрази для коефіцієнта подібності 2

9232
Кількість слів

72258
Кількість символів

Подібності за списком джерел

Нижче наведений список джерел. В цьому списку є джерела із різних баз даних. Колір тексту означає в якому джерелі він був знайдений. Ці джерела і значення Коефіцієнту Подібності не відображають прямого плагіату. Необхідно відкрити кожне джерело і проаналізувати зміст і правильність оформлення джерела.

10 найдовших фраз — Колір тексту

ПОРЯДКОВИЙ НОМЕР	НАЗВА ТА АДРЕСА ДЖЕРЕЛА URL (НАЗВА БАЗИ)	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)	
1	https://cad.kpi.ua/attachments/093_2016d_Azzuz.pdf	13	0.14 %
2	Вільзовик О.О диплом основна частина.pdf [redacted] 6/5/2023 State University of Telecommunications (ННІІТ)	13	0.14 %
3	Дизайн «Абетки» Наталки Поклад [redacted] 6/16/2023 Ukrainian Academy of Printing (Кафедра ГДМК)	11	0.12 %
4	https://graphql-ruby.org/api-doc/2.2.0/GraphQL/Schema	9	0.10 %