

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

бакалавр

(ступінь вищої освіти)

на тему РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ
МОНІТОРИНГУ КОМЕРЦІЙНИХ ПРОПОЗИЦІЙ НА ОСНОВІ
ГЕОЛОКАЦІЙНИХ ДАНИХ
DEVELOPMENT OF SOFTWARE FOR LOCATION-BASED
MONITORING OF COMMERCIAL OFFERS

Виконав(ла): студент(ка) 4 курсу, групи КНТ-113сп

Спеціальності 121 Інженерія програмного

(код і найменування спеціальності)

забезпечення

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

ПАНЧЕНКО Я.К.

(ПРІЗВИЩЕ та ініціали)

Керівник СКРУПСЬКИЙ С.Ю.

(ПРІЗВИЩЕ та ініціали)

Рецензент ШИТІКОВА О.В.

(ПРІЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
 (повне найменування закладу вищої освіти)

Факультет КНТ
 Кафедра програмних засобів
 Ступінь вищої освіти бакалавр
 Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
 Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
 “ ” 2026 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

ПАНЧЕНКА Ярослава Костянтиновича

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Розроблення програмного забезпечення для моніторингу комерційних пропозицій на основі геолокаційних даних. Development of Software for Location-Based Monitoring of Commercial Offers.

керівник проєкту (роботи) к.т.н., доцент, СКРУПСЬКИЙ Степан Юрійович,
(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від “7” квітня 2026 року № 139

2. Строк подання студентом проєкту (роботи) 10 червня 2026 року
 3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Розробка архітектури програми. 3. Реалізація програми. 4. Експлуатація та тестування програм.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів) Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	СКРУПСЬКИЙ С.Ю., доцент		
Нормоконтроль	КАЛІНІНА М.В., асистент		

7. Дата видачі завдання “ 20 ” квітня 2026 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	1 тиждень	Розділ 1
3	Вибір мови програмування та інших технологій розробки.	2 тиждень	Розділ 2
4	Розробка архітектури програми.	2 тиждень	Розділ 3
5	Розробка програми.	3-4 тижні	Розділ 3, 4
6	Тестування та експериментальне дослідження програмного забезпечення.	5 тиждень	Розділ 4
7	Оформлення пояснювальної записки та документів до неї.	6 тиждень	Додатки
8	Нормоконтроль та рецензування.	7 тиждень	
9	Захист роботи.	8 тиждень	

Студент(ка)

(підпис) Ярослав ПАНЧЕНКО
(Ім'я ПРІЗВИЩЕ)

Керівник проєкту (роботи)

(підпис) Степан СКРУПСЬКИЙ
(Ім'я ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра:
97 с., 2 табл., 36 рис., 3 дод., 21 джерело.

ГЕОСОЦІАЛЬНИЙ ЗАСТОСУНОК, ІНТЕРАКТИВНА КАРТА, ТОЧКИ ІНТЕРЕСУ (POI), ГЕОЛОКАЦІЯ, ПРОСТОРОВІ ДАНІ, ВЕБФРЕЙМВОРК, POSTGIS.

Об'єкт роботи – процес інженерії програмного забезпечення геосоціальної інформаційної системи для моніторингу локальних подій.

Предмет роботи – програмні засоби проектування і реалізації інтерактивного вебзастосунку для відображення, пошуку та обговорення локальних точок інтересу (POI) на базі просторових даних.

Мета роботи – підвищення ефективності інформування населення та взаємодії місцевих громад шляхом розроблення геосоціального вебзастосунку для моніторингу локальних подій, комерційних пропозицій та заходів у режимі реального часу.

Матеріали, методи та технічні засоби: мови програмування TypeScript та JavaScript, бібліотека React, вебфреймворк Next.js, система керування реляційними базами даних PostgreSQL із просторовим розширенням PostGIS, BaaS-платформа Supabase, картографічна бібліотека Leaflet.

Результати. Результатом виконання роботи є створений геосоціальний вебзастосунок, який забезпечує можливість створення, відображення та пошуку локальних подій на інтерактивній карті. Реалізовано функціонал геолокаційного пошуку, фільтрації за категоріями, а також систему взаємодії користувачів за допомогою коментарів та реакцій. Застосування просторової бази даних та оптимізованих запитів дозволило забезпечити високу швидкість обробки географічних даних та їх відображення у клієнтському інтерфейсі.

Висновки. Розроблене програмне забезпечення вирішує завдання

швидкого пошуку та обміну інформацією про локальні активності. Використання сучасного стека вебтехнологій та просторових індексів забезпечує високу продуктивність застосунку та швидкий відгук системи, а інтерактивний картографічний інтерфейс покращує користувацький досвід та рівень залученості до життя місцевої спільноти.

Галузь використання. Інформаційна підтримка місцевих громад, локальний маркетинг малого бізнесу, організація та координація громадських заходів, соціальні ініціативи.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 97 pages, 2 tables, 36 figures, 3 appendixes, 21 sources.

GEO-SOCIAL APP, INTERACTIVE MAP, POINTS OF INTEREST, GEOLOCATION, SPATIAL DATA, WEB FRAMEWORK, POSTGIS.

The object of this work is the software engineering process for a geosocial information system designed to monitor local events.

The subject of the work is the methods, models, and software tools for designing and implementing an interactive web application for displaying, searching, and discussing local points of interest (POI) based on spatial data.

The goal of the work is to increase the efficiency of public informing and local community interaction by developing a geo-social web application for real-time monitoring of local events, commercial offers, and activities.

Materials, methods and technical means: TypeScript and JavaScript programming languages, React library, Next.js web framework, PostgreSQL relational database management system with PostGIS spatial extension, Supabase BaaS platform, Leaflet mapping library.

The results. The outcome of the work is a created geo-social web application that provides the ability to create, display, and search for local events on an interactive map. Geolocation search functionality, category filtering, and a user interaction system via comments and reactions were implemented. The application of a spatial database and optimized queries ensured a high speed of processing geographic data and its rendering in the client interface.

Conclusions. The developed software solves the problem of quick search and information exchange regarding local activities. The use of a modern web technology stack and spatial indices ensures high application performance and fast system

response, while the interactive cartographic interface enhances user experience and the level of engagement in local community life.

Field of use. Information support for local communities, local marketing for small businesses, organization and coordination of public events, social initiatives.

ЗМІСТ

	С.
Перелік скорочень та умовних позначок	10
Вступ.....	11
1 Аналіз предметної області	13
1.1 Аналіз застосунків для організації офлайн-заходів та пошуку учасників	13
1.2 Порівняльний аналіз розробки та аналогів.....	18
1.3 Висновки за розділом 1.....	20
2 Розробка архітектури програми	22
2.1 Вибір засобів проєктування	22
2.2 Сценарії використання програми	25
2.3 Структура бази даних	28
2.4 Висновки за розділом 2.....	40
3 Реалізація програми.....	41
3.1 Визначення порядку роботи програми	41
3.2 Результати реалізації програмних одиниць застосунку	41
3.3 Висновки за розділом 3.....	52
4 Експлуатація та тестування програм	53
4.1 Призначення програми	53
4.2 Умови виконання програми	53
4.3 Виконання і тестування програми	54
4.4 Висновки за розділом 4.....	59
Висновки	60
Перелік джерел посилання	62

Додаток А Технічне завдання	64
Додаток Б Текст програми	70
Додаток В Слайди презентації	90

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API	– Application Programming Interface;
CSS	– Cascading Style Sheets;
HTML	– Hypertext Markup Language;
HTTP	– Hypertext Transfer Protocol;
JSON	– JavaScript Object Notation;
POI	– Point of Interest;
RLS	– Row-Level Security;
SQL	– Structured Query Language;
SSR	– Server-Side Rendering;
URL	– Uniform Resource Locator;
UUID	– Universally Unique Identifier;
WKB	– Well-Known Binary;
БД	– база даних;
ВФ	– вебфреймворк;
Гб	– гігабайт;
Гц	– гігагерц;
СУБД	– система управління базами даних.

ВСТУП

В сучасному світі локальні спільноти та бізнес є дуже важливою складовою міського середовища, оскільки вони забезпечують соціальну взаємодію та розвиток економіки. Це стосується проведення локальних ярмарків та концертів, відкриття нових закладів, організації тимчасових подій та швидкого поширення інформації про комерційні пропозиції. Підтримці та розвитку локальної економіки сьогодні приділяється багато уваги і саме це підтверджує актуальність та соціальну значущість розробки системи для поширення локальної інформації в режимі реального часу.

Проте інформаційна підтримка локальних ініціатив сьогодні здійснюється на різних рівнях за допомогою розрізнених інструментів:

- через локальні групи та канали у класичних соціальних мережах або месенджерах;
- через статичні картографічні сервіси, які орієнтовані переважно на постійні об'єкти (бізнеси, установи), а не на динамічні події;
- через традиційні засоби рекламного інформування на місцях (оголошення, флаєри).

Наразі недостатньо уваги приділяється створенню спеціалізованих платформ, які б поєднували можливості інтерактивної картографії зі зручністю соціальної взаємодії у режимі реального часу. Це стосується випадків, коли мешканці певного району прагнуть швидко дізнатися про поточні або короткочасні активності поруч або поділитися такою інформацією з іншими. Звичайно, певну частину подій можна знайти через алгоритмічні стрічки новин, проте відсутність точної географічної та часової прив'язки робить такий пошук не зручним та неефективним. Загалом же популярність сервісів, що базуються на місцезнаходженні, підкреслює потребу суспільства в інструментах для дослідження локальних подій та власного оточення. Тому у цій роботі основний акцент зроблено на створенні єдиної геосоціальної платформи для обміну часозалежною геопросторовою інформацією.

Метою роботи є підвищення ефективності інформування населення та взаємодії місцевих громад шляхом розроблення геосоціального вебзастосунку для моніторингу локальних подій, комерційних пропозицій та заходів у режимі реального часу.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз застосунків для організації офлайн-заходів та пошуку учасників

Популярність систем просторового позиціонування [1] (Location-Based Services) стрімко зростає на тлі глобальної цифровізації міського середовища та еволюції мобільних технологій. Такі геоінформаційні системи дозволяють користувачам взаємодіяти з просторовими даними у режимі реального часу, отримуючи інформацію, релевантну їхній поточній геолокації. Важливою складовою таких систем є концепція точок інтересу, яка традиційно використовується для позначення статичних об'єктів інфраструктури. Проте сучасні потреби міських громад вимагають переходу від статичної картографії до динамічного моніторингу тимчасових подій [2]. Під час аналізу предметної області було розглянуто ряд застосунків та платформ, які стосуються сфери обміну геоінформацією, при цьому жодна з них не має комплексного вирішення проблеми інтерактивного просторового моніторингу.

Серед існуючих рішень базовим інструментом для орієнтування у міському просторі є картографічні сервіси загального призначення. Як основний приклад такого програмного забезпечення доцільно виділити Google Maps [3]. Такі рішення чудово підходять для навігації та пошуку постійних установ, наприклад, ресторанів, музеїв чи магазинів, відображаючи години їх роботи та накопичені відгуки (рис. 1.1). Однак їхнім критичним недоліком є неможливість ефективно відстежувати динамічні та короткочасні події місцевих громад. На таких мапах складно виділити заходи, що тривають лише кілька годин, оскільки архітектура системи орієнтована на сталі географічні об'єкти з тривалим циклом модерації. Для вирішення такої проблеми у застосунку, який розробляється передбачається розробити систему, яка дозволить звичайним користувачам миттєво створювати географічні мітки та відображати події без складних процедур реєстрації бізнесу.

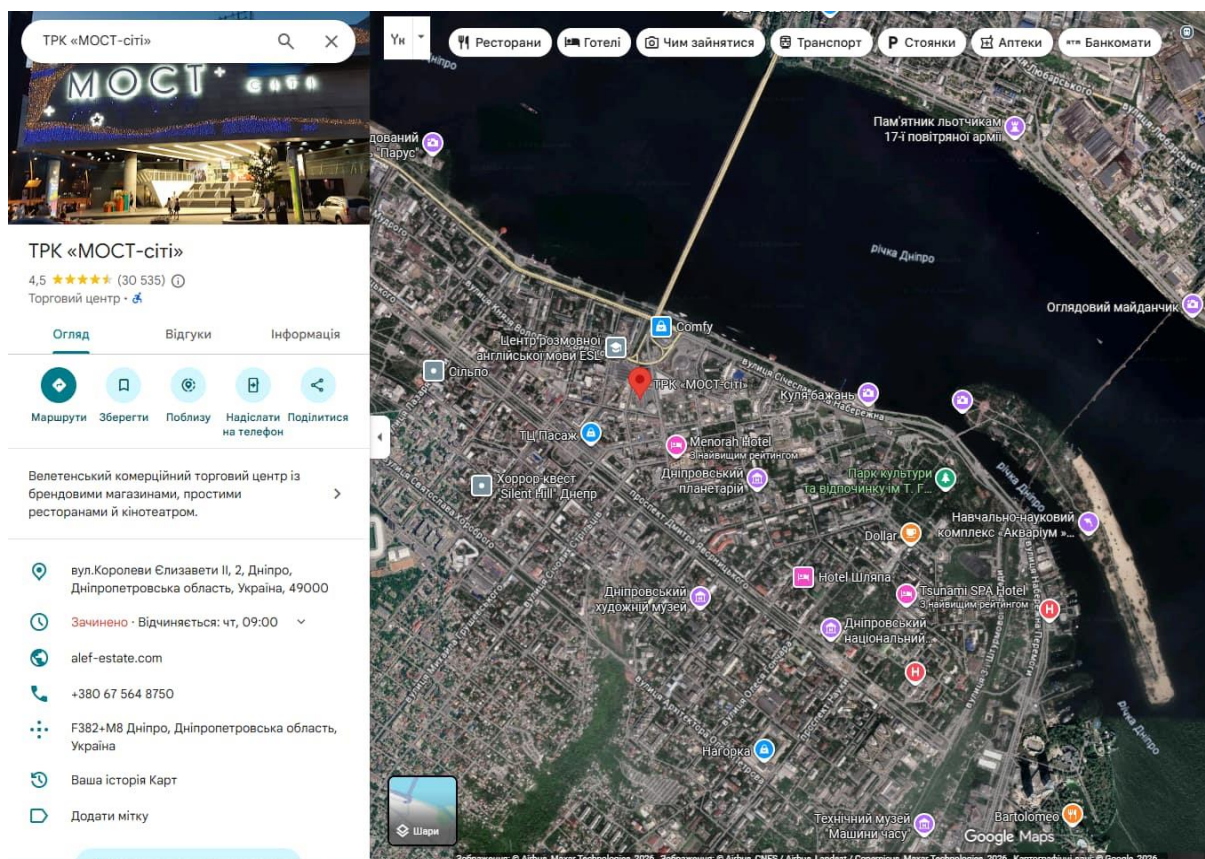


Рисунок 1.1 – Відображення статичної локації у застосунку Google Maps [3]

Альтернативний підхід до пошуку локацій реалізовано у рекомендаційних геосоціальних системах. Сервіс Foursquare (рис. 1.2) фокусується на соціальній взаємодії довкола існуючих закладів [4]. Він містить велику базу відгуків, фотографій та рекомендацій від інших користувачів. Проте об'єктами у цій системі виступають виключно постійно функціонуючі заклади та установи. Користувач не має зручної можливості швидко створити позначку про спонтанну подію з таймером зворотного відліку. Акцент зроблено на накопиченні історичних рейтингів, а не на відображенні актуального стану в режимі реального часу. Відповідно, у майбутній розробці планується впровадити систему динамічних статусів, що автоматично приховуватимуть неактуальні точки з карти після завершення події або втрати її актуальності.

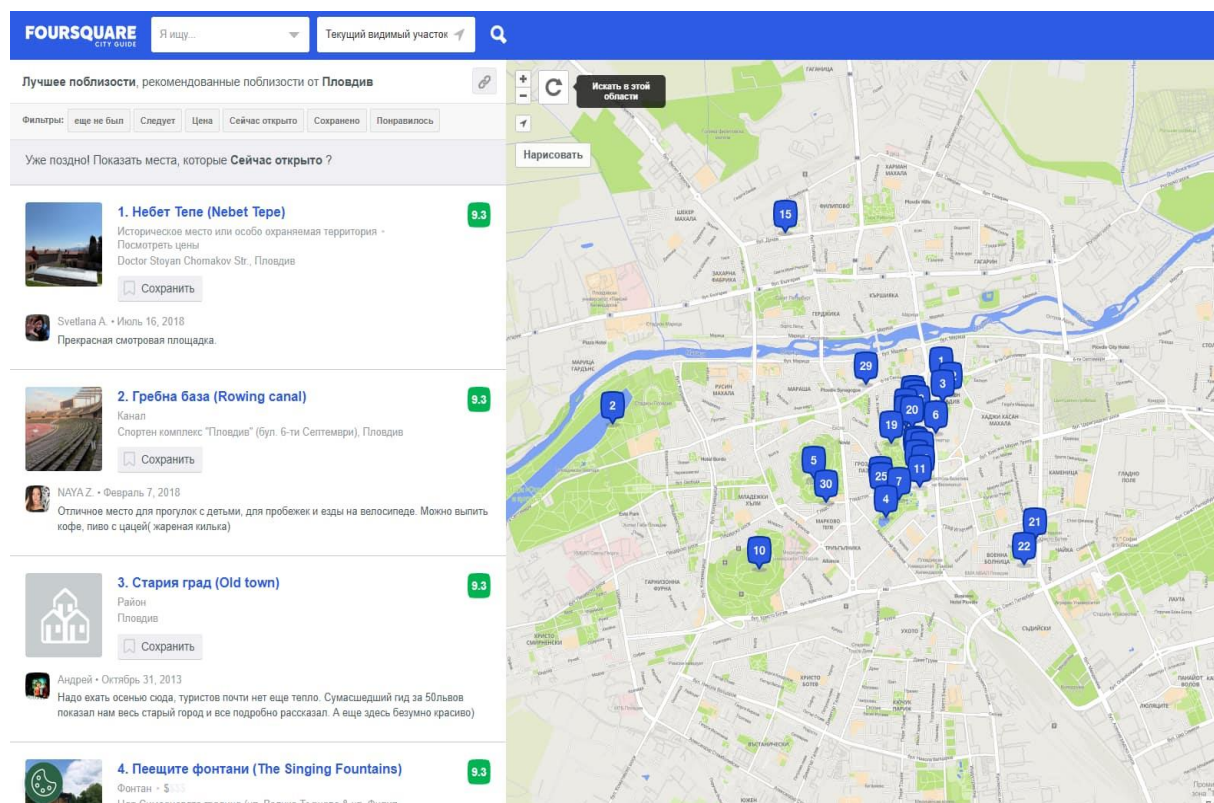


Рисунок 1.2 – Интерфейс пошуку та оцінювання закладів у вебзастосунку Foursquare [4]

Окремим напрямком інформаційного забезпечення міських громад є використання локальних модулів глобальних соціальних мереж. Платформа Facebook Local (рис. 1.3), надає потужні можливості для анонсування подій та обговорення життя району [5].

Проте основним архітектурним недоліком таких систем є алгоритмічний стрічковий підхід до відображення інформації. Інтерактивна мапа відіграє другорядну роль або взагалі відсутня під час перегляду основної стрічки новин. Інформація про географічне розташування часто подається виключно у вигляді тексту, що змушує користувача переходити на окремі сторінки та вручну співставляти адреси. Це ускладнює миттєвий пошук активностей поблизу. Задачею майбутнього проєкту є використання інтерактивного картографічного полотна як головного та єдиного інтерфейсу навігації.

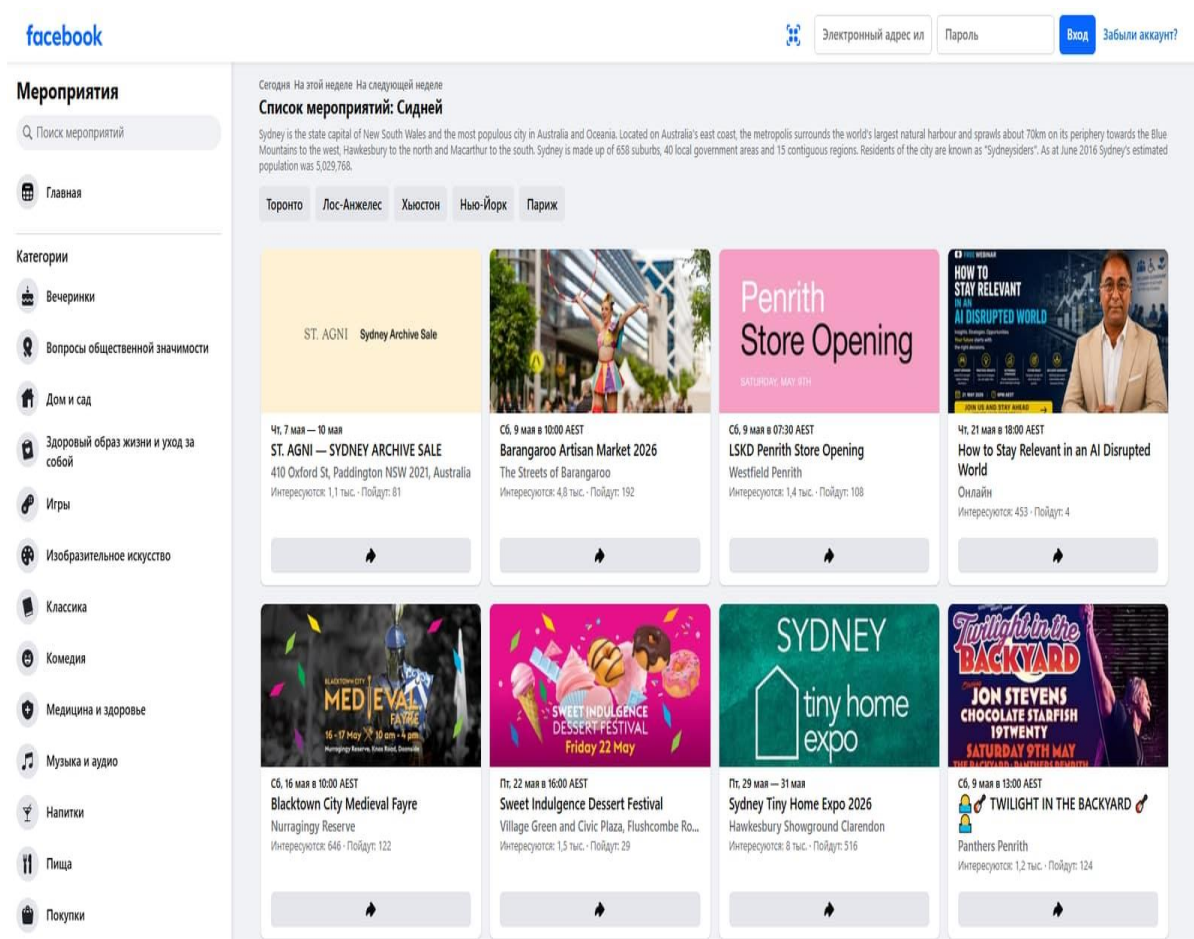


Рисунок 1.3 – Представлення локальних подій у Facebook Local [5]

Якщо розглядати платформи для безпосередньої організації зустрічей, користувачі найчастіше звертаються до сервісу Meetup (рис. 1.4).

Однак цей сервіс спрямований переважно на планування групових зустрічей за інтересами, а не на спонтанне виявлення комерційних пропозицій чи тимчасових подій [6]. Місце проведення на сторінці події не відображається за допомогою вбудованої картографічної панелі, натомість подається виключно у вигляді текстової адреси. Просторовий контекст суттєво зменшується: користувач не має змоги вільно переміщувати мапу, не може продивитись детальніше що знаходиться навколо та не бачить інші активні події у вибраному районі на єдиному екрані. Це формує вимогу до розробки системи просторових запитів, яка буде динамічно підвантажувати маркери подій залежно від області видимості карти на екрані пристрою.

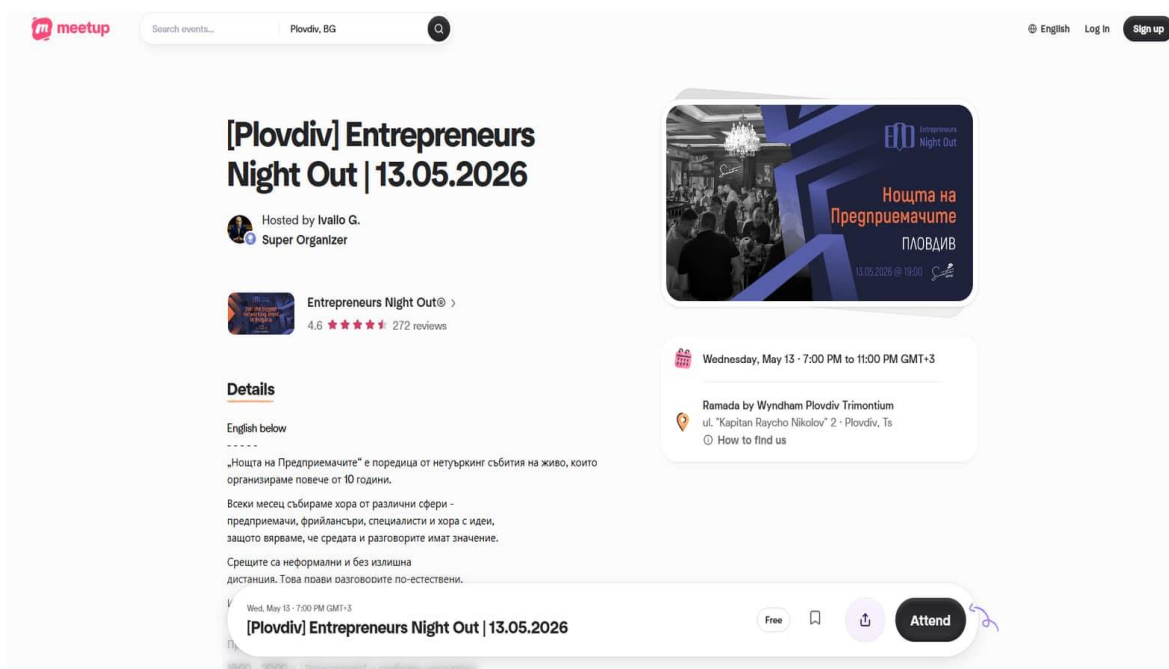


Рисунок 1.4 – Відображення події у застосунку Meetup [6]

Поряд із соціальними мережами, популярним вебзастосунком для пошуку заходів є платформа Eventbrite (рис. 1.5). Цей сервіс доступний безпосередньо у браузері та дозволяє шукати заплановані події з можливістю їх відображення на вбудованій карті [7]. Проте суттєвим обмеженням такого рішення є його жорстка орієнтація на формальну організацію та продаж квитків. Процес створення події є багатоступеневим і розрахованим на професійних організаторів. Звичайний мешканець громади позбавлений можливості спонтанно розмістити швидку мітку про поточну локальну активність, наприклад, про тимчасовий розпродаж або виступ вуличних музикантів. Крім того, платформа не підтримує механізмів швидкої соціальної оцінки подій у реальному часі. З огляду на це, у застосунку, що розробляється передбачається реалізація полегшеного алгоритму створення точок інтересу, який дозволяє будь-якому користувачу миттєво фіксувати події на мапі в один клік, а також розміщувати швидкі реакції та коментарії.

Підсумовуючи результати аналізу існуючих програмних продуктів, можна стверджувати, що ринок потребує єдиного спеціалізованого застосунку.

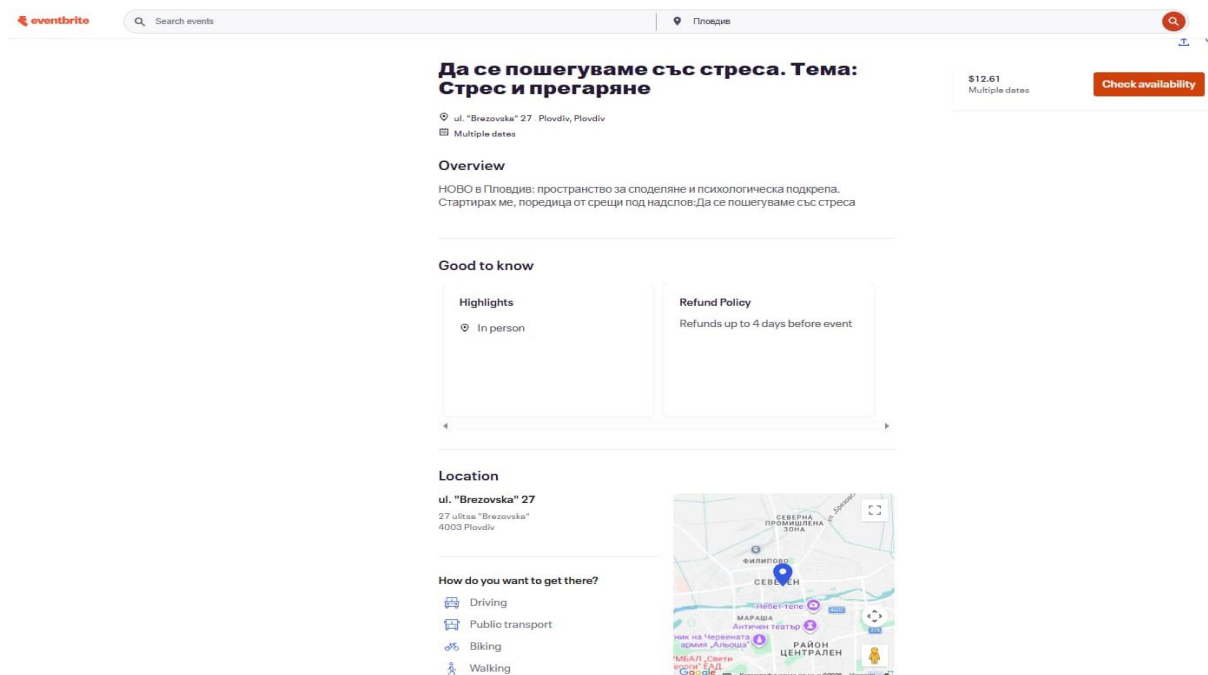


Рисунок 1.5 – Інтерфейс пошуку подій на платформі Eventbrite [7]

Жоден із розглянутих аналогів не пропонує гнучкого інструментарію, де головним об'єктом є тимчасова локальна подія з визначеним часовим проміжком існування та можливістю її швидкого створення самими користувачами. Тому подальша розробка буде спрямована на створення геосоціального вебзастосунку, який поєднуватиме якісний картографічний інтерфейс, акцент на тайм-менеджменті подій та зручні механізми взаємодії місцевих громад.

1.2 Порівняльний аналіз розробки та аналогів

Для остаточного обґрунтування актуальності та визначення функціональних переваг майбутньої розробки було проведено порівняльний аналіз відібраних програмних продуктів. До фінального порівняння увійшли застосунки Google Maps, Meetup та Eventbrite, функціональні можливості яких було зіставлено з вимогами до проєктованого геосоціального вебзастосунку. Результати порівняння за ключовими критеріями наведено у табл. 1.1.

Таблиця 1.1 – Порівняльний аналіз програм для моніторингу подій

Критерій оцінювання	Google Maps	Meetup	Eventbrite	Проектована розробка
Основний тип об'єктів	Статичні установи	Заплановані зустрічі груп за інтересами	Формальні заходи	Тимчасові події та локальні точки інтересу
Роль картографічного інтерфейсу	Головний інтерфейс відображення	Відсутній, лише текстова адреса	Додатковий інструмент пошуку	Головний інтерфейс з маркерами
Контроль часового циклу подій	Статичні години роботи закладів	Фіксована дата та час проведення заходу	Фіксована дата та час проведення заходу	Динамічні статуси з живим зворотним відліком
Створення міток користувачами	Тривала модерація або реєстрація бізнесу	Створення зустрічей адміністраторами груп	Багатоступеневе налаштування квитків	Миттєве розміщення в один клік по мапі
Соціальна взаємодія	Текстові відгуки та система рейтингу	Коментарі та підтвердження участі	Відсутня, акцент на транзакціях	Швидкі коментарі та система реакцій
Просторовий пошук за видимою областю	Присутній	Відсутній, пошук за назвою міста	Частково присутній	Присутній, оптимізований просторовими запитам

Порівняння, наведене у табл. 1.1, наочно демонструє відсутність на ринку комплексного рішення, яке б задовольняло потреби локальних громад у миттєвому обміні просторовою інформацією. Встановлено, що потужні картографічні платформи на кшталт Google Maps не пристосовані для роботи з динамічними даними через складність модерації та орієнтацію на бізнес. З іншого боку, соціальні платформи для організацій подій, такі як Meetup та Eventbrite, унеможливають спонтанне створення відміток звичайними користувачами та не надають повноцінного картографічного досвіду для візуального пошуку активностей поблизу. Окрім того, жоден із розглянутих аналогів не пропонує інтерактивного контролю часу із живим зворотним відліком, що є критично важливим для короткочасних подій, наприклад флеш-розпродажів чи вуличних виступів.

Геосоціальний застосунок, що проектується, повністю вирішує ці проблеми. Планується, що нова розробка поєднає швидкість створення міток, характерну для соціальних мереж, із точністю просторових запитів класичних картографічних систем. Застосування динамічних таймерів дозволить автоматично фільтрувати неактуальні події, а система швидких реакцій забезпечить високий рівень достовірності інформації. Таким чином, розробка запропонує унікальний інструментарій, якого наразі бракує у наявних аналогів.

1.3 Висновки за розділом 1

У першому розділі було проведено детальний аналіз предметної області та досліджено проблематику створення геоінформаційних систем для місцевих громад. Встановлено, що незважаючи на широке розповсюдження сервісів на базі визначення місцезнаходження, ніша моніторингу короткочасних та спонтанних подій у режимі реального часу залишається незаповненою.

Аналіз існуючих аналогів, серед яких глобальні картографічні платформи та соціальні сервіси для організації заходів, виявив низку суттєвих функціональних недоліків. Зокрема виділено жорстку орієнтацію на статичні довгострокові об'єкти, багатоступеневу складність створення міток звичайними користувачами, стрічковий алгоритмічний підхід до відображення інформації замість єдиного картографічного полотна, а також відсутність інструментів управління життєвим циклом подій.

На основі виявлених проблем було сформульовано такі загальні завдання для виконання дипломної роботи:

- проаналізувати наявне програмне забезпечення у сфері моніторингу локальних подій та визначити його архітектурні недоліки;
- обґрунтувати вибір сучасного технологічного стека та програмних засобів для проєктування геосоціального вебзастосунку;
- сформулювати функціональні вимоги до архітектури системи, розробити структуру бази просторових даних та клієнтського інтерфейсу;
- виконати програмну реалізацію інтерактивного вебзастосунку із забезпеченням механізмів миттєвого створення тимчасових точок інтересу, їхньої географічної фільтрації та швидкої соціальної взаємодії.

Для усунення недоліків існуючих рішень передбачається розробка спеціалізованого програмного забезпечення, яке поєднає швидкість роботи соціальних мереж із точністю просторових баз даних. Проєктування архітектури розроблюваної системи, обґрунтування вибору технологій та розробка логічної моделі бази даних будуть розглянуті у наступному розділі.

2 РОЗРОБКА АРХІТЕКТУРИ ПРОГРАМИ

2.1 Вибір засобів проєктування

Сучасний геосоціальний вебзастосунок доцільно створювати на основі надійного екосистемного рішення, таким чином отримуючи готовий архітектурний каркас та можливість зосередитися на безпосередній реалізації складної бізнес-логіки програми, а також забезпечити високий рівень її подальшої масштабованості. Враховуючи необхідність забезпечення високої інтерактивності в умовах постійного оновлення інформації, швидкого рендерингу картографічного інтерфейсу на різних типах пристроїв та складної роботи з масивами просторових даних, було вирішено побудувати систему виключно на базі сучасних вебстандартів і мов JavaScript [8] та TypeScript [9]. Цей стратегічний вибір дозволяє успішно використовувати єдину мову програмування як для клієнтської, так і для серверної частини застосунку, що суттєво знижує когнітивне навантаження на розробника, мінімізує кількість помилок при передачі даних та значно прискорює загальний процес розробки і підтримки програмного коду.

Як основні кандидати для реалізації розглядалися два популярні підходи у світі візуальної веброботи.

Першим варіантом є класичний стек на базі роздільної архітектури [10], що включає клієнтський застосунок на бібліотеці React [11], окремий сервер на базі Node.js [12] з фреймворком Express [13] та документоорієнтовану базу даних MongoDB [14].

Другим варіантом є сучасний гібридний підхід [15] на базі фреймворку Next.js у поєднанні з хмарною платформою Supabase [16] та реляційною базою даних PostgreSQL [17].

Критерії порівняння, наведені у табл. 2.1, стосуються загального архітектурного підходу, роботи з просторовими даними та принципів рендерингу інтерфейсу.

Таблиця 2.1 – Вибір засобів для розробки програми

Критерій порівняння	Класичний стек: React, Node.js, Express, MongoDB	Гібридний стек: Next.js, Supabase, PostgreSQL
Архітектурний підхід	Повністю роздільна архітектура, де клієнт спілкується із сервером виключно через ручне створення REST API	Об'єднана архітектура, де взаємодія з базою даних відбувається через безпечні серверні дії без окремих API-маршрутів
Робота з просторовими даними	Базові географічні запити у NoSQL базі даних, складність масштабування складних просторових розрахунків	Повноцінна інтеграція з реляційною базою даних та галузевим просторовим розширенням PostGIS безпосередньо з коробки
Рендеринг інтерфейсу	Виключно клієнтський рендеринг, що може викликати затримки при початковому завантаженні важких скриптів мапи	Гібридний рендеринг із серверною генерацією сторінок, що забезпечує швидке відображення каркаса застосунку
Інфраструктура та авторизація	Необхідність самотійного проєктування системи безпеки, налаштування серверів та зберігання медіафайлів	Вбудована система авторизації, управління політиками доступу на рівні рядків та готове хмарне сховище зображень

Враховуючи специфіку геосоціальної програми, критично важливою є роль ефективної обробки географічних координат, безпечного збереження прикріплених зображень та миттєвого оновлення маркерів подій на інтерактивній мапі. Якщо використовувати класичний роздільний стек з Node.js розробнику доводиться самотійно проєктувати та налаштовувати кожен рівень взаємодії між сервером і клієнтом [18]. Включаючи написання

контролерів, налаштування маршрутизації, управління політиками міждоменних запитів та ручну імплементацію проміжного програмного забезпечення для перевірки токенів авторизації. Крім того, нереляційні бази даних на кшталт MongoDB хоч і підтримують базові просторові запити, проте суттєво поступаються спеціалізованим інструментам при розрахунках перетинів та фільтрації об'єктів у межах обмежувальних рамок видимої області екрана [19].

Якщо ж використати гібридний фреймворк Next.js та хмарну платформу Supabase, це дозволить отримати потужну інфраструктуру одразу. Функціонал серверних дій фреймворку Next.js усуває потребу у створенні окремого мережевого програмного інтерфейсу. Дані безпечно передаються з клієнтських форм безпосередньо до серверних функцій зі збереженням суворої типізації. Зі свого боку платформа Supabase надає готову підсистему зберігання медіафайлів із вбудованою перевіркою прав доступу на рівні рядків таблиці, що нівелює необхідність налаштування сторонніх хмарних сховищ та мікросервісів завантаження.

Ключовою перевагою є використання реляційної бази даних PostgreSQL із розширенням PostGIS [20]. Застосування індексів узагальненого дерева пошуку та нативних операторів перетину дозволяє серверу миттєво відфільтровувати лише ті події, які фізично потрапляють у поточне вікно картографічної платформи Leaflet.

Усе це сукупно призводить до того, що час, витрачений на розгортання базової інфраструктури, стає значно меншим порівняно з альтернативними рішеннями. Уникнути потреби в таких оптимізованих інструментах неможливо, особливо враховуючи необхідність безпечного ізолювання клієнтської бібліотеки мапи від серверного рендерингу та потребу в миттєвому оновленні кешу при зміні часових статусів подій. Тому для програмної реалізації застосунку було остаточно обрано стек на базі Next.js у зв'язці з платформою Supabase та просторовою базою даних.

2.2 Сценарії використання програми

Для забезпечення безпечного доступу до картографічного функціоналу та коректного управління контентом у системі виділено три ключові ролі акторів: неавторизований відвідувач, авторизований користувач та адміністратор. Процес розроблення інтерактивного геосоціального вебзастосунок вимагає чіткого визначення поведінкових аспектів та можливостей для кожної з цих ролей. З метою формалізації вимог та наочної візуалізації меж проєктованої системи було застосовано методологію об'єктно-орієнтованого моделювання і побудовано діаграму прецедентів.

Розроблена модель системи базується на використанні відношення узагальнення, що є ефективним патерном для уникнення дублювання базових прецедентів. Завдяки впровадженому принципу ієрархічного наслідування прав доступу, авторизований користувач автоматично успадковує всі функціональні можливості звичайного відвідувача, розширюючи їх інструментами створення точок інтересу. Відповідно, адміністратор переймає всі повноваження користувача, доповнюючи їх розширеними функціями глобальної модерації та загального контролю платформи.

Основний потік сценаріїв системи формується навколо базових прецедентів, частина з яких має обов'язкові залежності типу включення. До таких сценаріїв належать:

- перегляд інтерактивної мапи та вільне переміщення картографічного полотна;
- фільтрація точок інтересу за визначеними категоріями;
- перегляд детальної панелі обраної події, що обов'язково включає прецедент відображення таймера зворотного відліку та прецедент перегляду медіагалереї;
- авторизація в системі за допомогою облікових даних;
- створення нової точки інтересу шляхом встановлення маркера безпосередньо на мапі, що включає прецедент просторового перетворення

координат та прецедент клієнтського стиснення завантажених фотографій перед відправленням на сервер;

- публікація коментаря під обраною локацією;
- оцінювання точок інтересу шляхом додавання реакції вподобання або невподобання;
- глобальна модерація контенту адміністратором системи шляхом примусового видалення нерелевантних маркерів або образливих коментарів незалежно від їхнього авторства.
- Окрім основного потоку, система передбачає низку альтернативних сценаріїв, які спрацьовують за певних умов і класифікуються як відношення розширення. До таких сценаріїв відносяться:
 - автоматичне центрування карти за збереженими координатами, що розширює сценарій перегляду мапи, коли система виявляє попередньо збережене просторове положення у локальному сховищі браузера;
 - скасування або зміна типу реакції, що розширює базовий сценарій оцінювання точки інтересу, коли користувач попередньо залишив позитивну реакцію, але згодом вирішив змінити її на негативну шляхом атомарного перемикання запису в базі даних;
 - видалення власної точки інтересу, що розширює прецедент керування власним профілем, коли захід було скасовано автором достроково і він більше не бажає відображати його для місцевої спільноти;
 - блокування створення події, що розширює прецедент встановлення часових меж, коли користувач намагається вказати дату початку події, яка є пізнішою за дату її завершення, що викликає спрацювання механізмів клієнтської валідації форм та зупиняє транзакцію.

Детальне графічне представлення всіх описаних прецедентів, відношень між ними та ієрархії акторів наведено на діаграмі прецедентів системи (рис. 2.1).

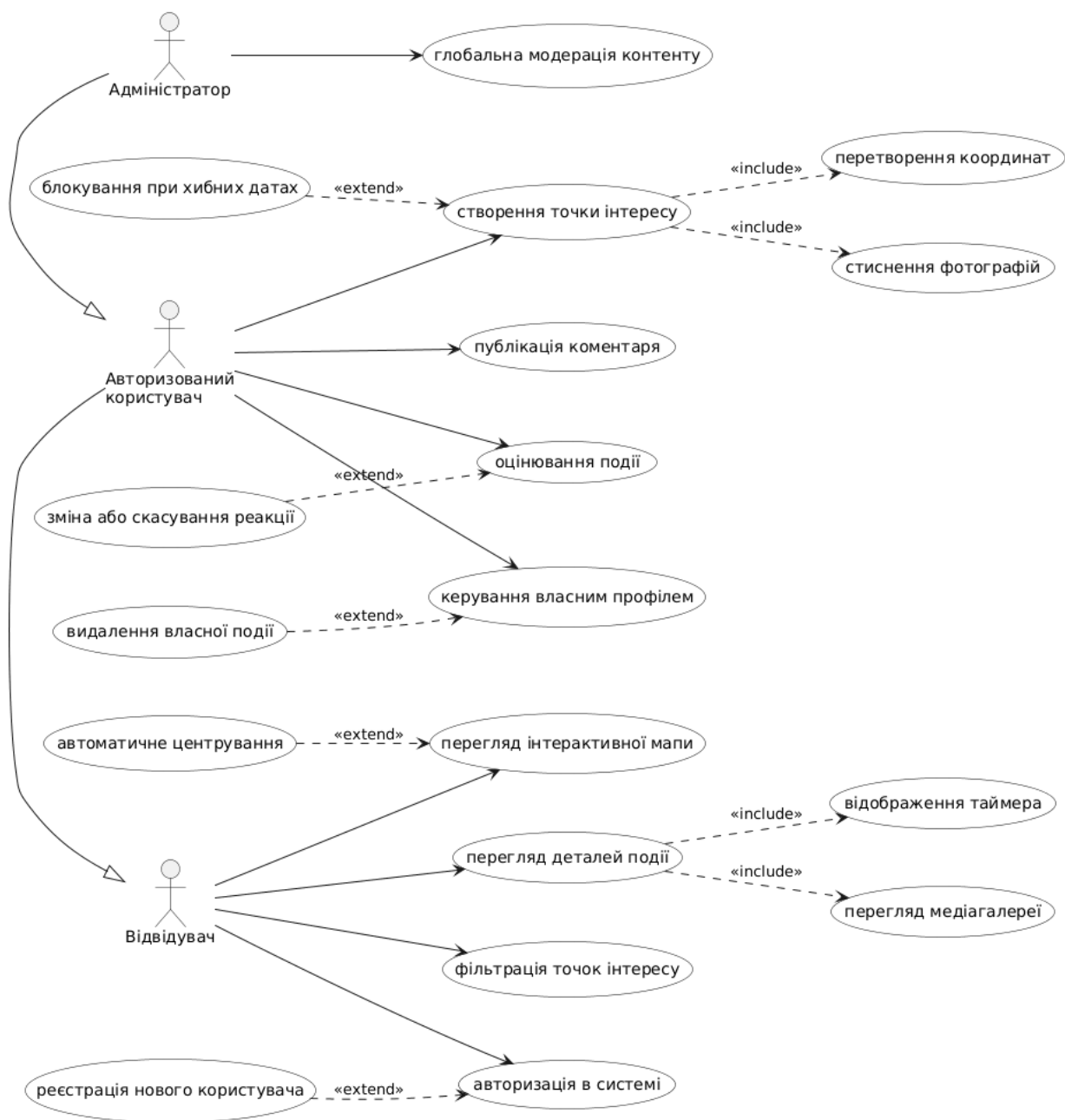


Рисунок 2.1 – Діаграма прецедентів програми

Окрім визначення ролей та прецедентів, проєктування системи вимагає чіткої організації навігаційного простору. Функціональна схема програми базується на просторовій навігації, де користувач починає роботу з ініціалізації інтерактивного картографічного полотна. Подальший алгоритм передбачає вибір відповідного функціонального напрямку: використання панелі фільтрації, взаємодію з картою для перегляду або створення подій, а також перехід до панелі керування профілем. Після виконання обраної дії система забезпечує автоматичне повернення користувача до головного екрана

для продовження циклу роботи. Візуальне представлення цієї структури наведено на рис. 2.2.

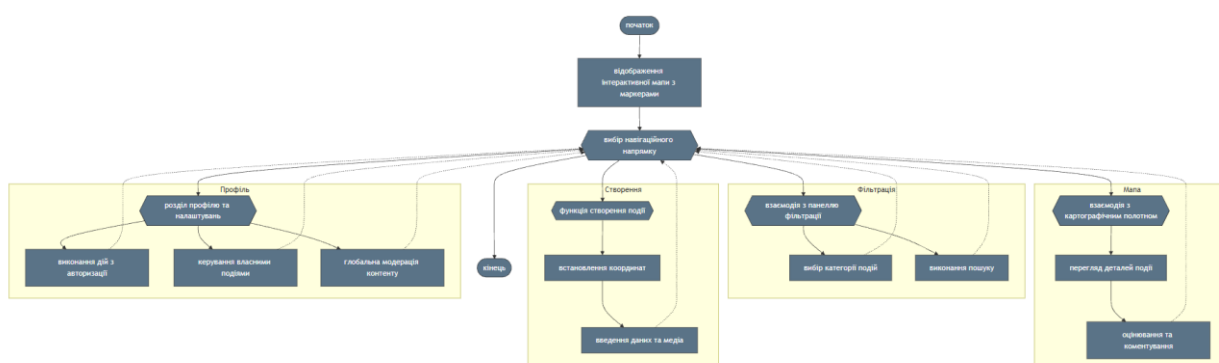


Рисунок 2.2 – Функціональна схема навігаційної структури застосунку

2.3 Структура бази даних

Перед детальним описом структури збереження інформації необхідно визначити архітектурні компоненти та алгоритми роботи ключових функцій системи. Для наочної демонстрації взаємозв'язків між клієнтською частиною, серверним середовищем та базою даних було побудовано діаграму компонентів застосунку. Ця модель ілюструє гібридну архітектуру, де фронтенд та бекенд тісно інтегровані в межах єдиного фреймворку.

Клієнтський рівень відповідає за рендеринг інтерфейсу користувача та взаємодію з картографічним рушієм Leaflet [21]. Серверний рівень, побудований на базі серверних дій Next.js, виконує роль проміжного шару безпеки та обробки бізнес-логіки. Рівень даних забезпечується хмарною інфраструктурою Supabase, яка включає реляційну базу даних із просторовим розширенням, систему автентифікації та об'єктне сховище для медіафайлів.

Структурну організацію та напрямки потоків даних між цими базовими компонентами наочно представлено на рис. 2.3.

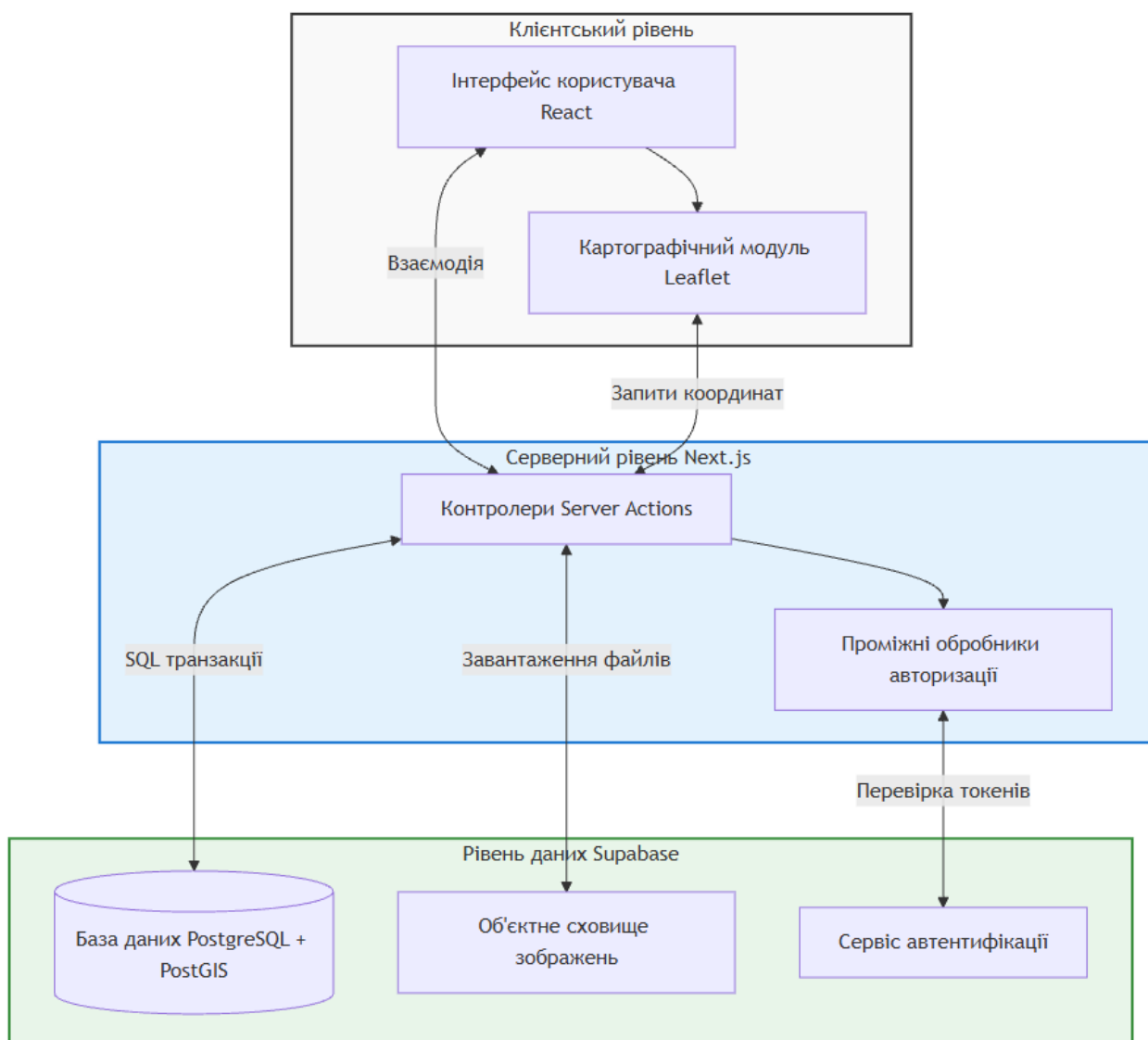


Рисунок 2.3 – Діаграма компонентів геосоціального вебзастосунку

Деталізація бізнес-логіки вимагає окремого моделювання поведінки користувача під час створення нової локальної події. Для цього було розроблено діаграму діяльності, яка демонструє покроковий процес взаємодії від моменту вибору координат на картографічному полотні до остаточного збереження даних на сервері.

Процес ініціюється користувачем шляхом встановлення маркера на мапі, після чого система відкриває модальне вікно форми. На цьому етапі користувач вводить текстову інформацію, обирає категорію та обов'язково встановлює часові рамки існування події. Ключовим етапом є клієнтська валідація введених дат: система перевіряє, чи не передує час завершення часу

початку. У разі виявлення логічної помилки процес повертається на етап редагування. У разі успішної перевірки відбувається завантаження прикріплених зображень, їхнє клієнтське стиснення для оптимізації мережевого трафіку та передача сформованого пакета даних до серверного контролера. Завершується процес транзакцією у просторову базу даних та ініціацією миттєвого оновлення маркерів на екрані.

Графічне представлення описаної послідовності дій алгоритму наведено на рис. 2.4.

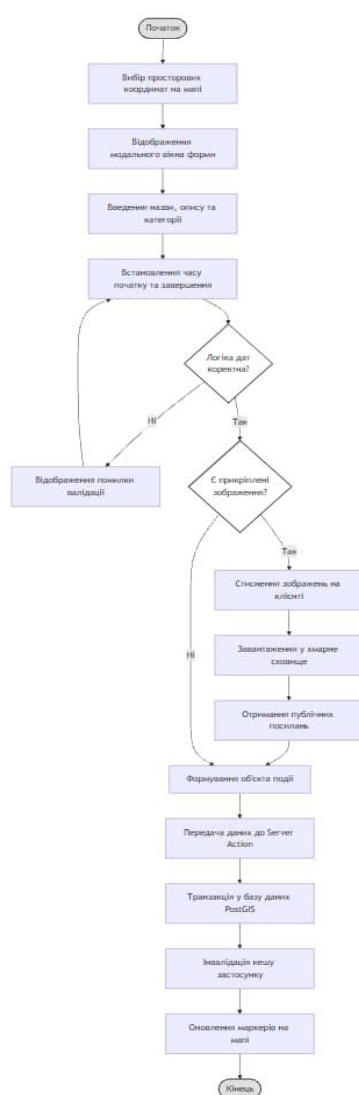


Рисунок 2.4 – Діаграма компонентів геосоціального вебзастосунку

Важливою складовою процесу створення точки інтересу, окрім логічної валідації на стороні клієнта, є архітектура збереження прикріплених

медіафайлів та їхня прив'язка до просторового об'єкта. Для деталізації цього етапу розроблено діаграму послідовності взаємодії клієнта з хмарним сховищем та базою даних. Алгоритм передбачає генерацію унікальних імен для оптимізованих зображень, їхнє завантаження у відповідний бакет та отримання публічних адрес. Лише після успішного збереження медіафайлів система перетворює координати у бінарний формат, формує фінальний запит до бази даних та інвалідує кеш маршрутів. Повну послідовність цих викликів відображено на рис. 2.5.

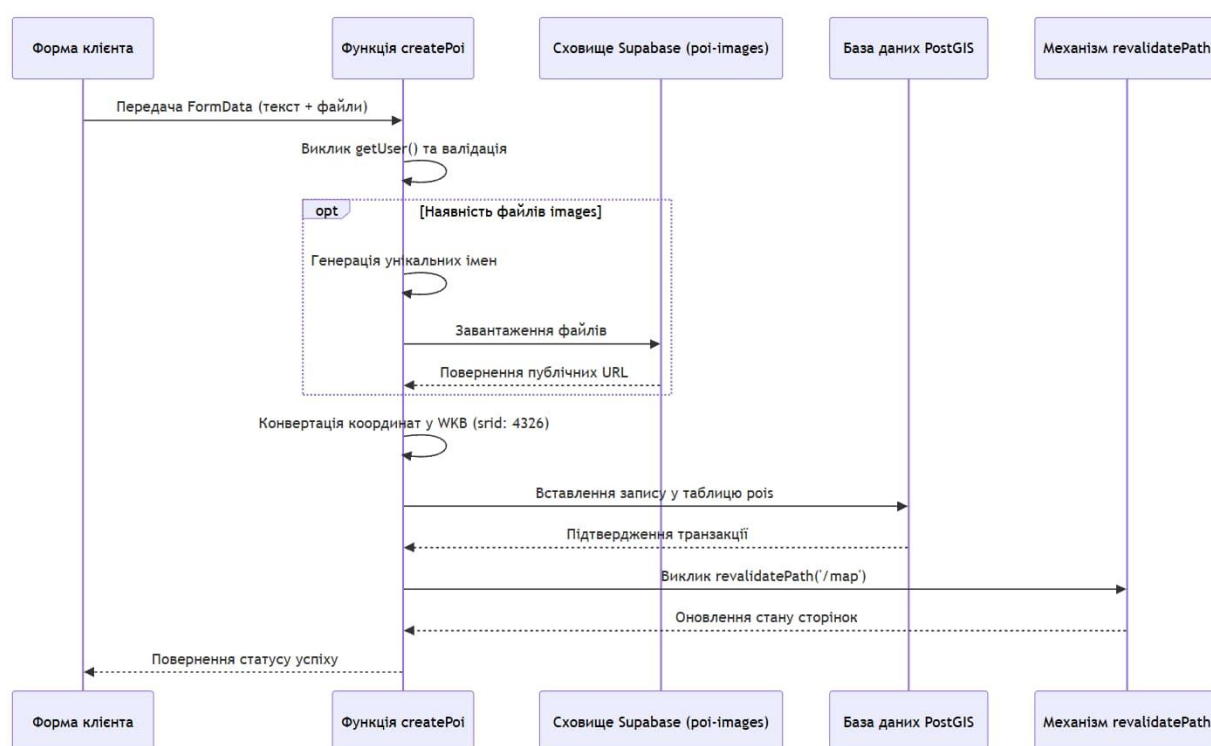


Рисунок 2.5 – Діаграма послідовності створення просторового об'єкта

Окрім процесу створення контенту, критично важливим аспектом функціонування геосоціального застосунку є динамічне завантаження даних під час переміщення користувачем картографічного полотна. Для візуалізації цього процесу та опису взаємодії між клієнтськими компонентами і серверним середовищем було побудовано діаграму послідовності.

В основі цього сценарію лежить подія зміни меж видимої області карти, яка ініціює збір географічних координат кутів екрана. Ці координати

передаються до серверної дії, яка формує просторовий запит до бази даних PostgreSQL. Використання операторів PostGIS дозволяє на рівні ядра бази даних відсікати об'єкти, що не потрапляють у поточний огляд, значно скорочуючи обсяг переданих даних та навантаження на клієнтський пристрій. Отриманий масив об'єктів повертається у форматі JSON, після чого картографічний модуль Leaflet виконує рендеринг маркерів та їхню кластеризацію.

Повна послідовність обміну повідомленнями та викликів функцій між учасниками процесу відображена на рис. 2.6.

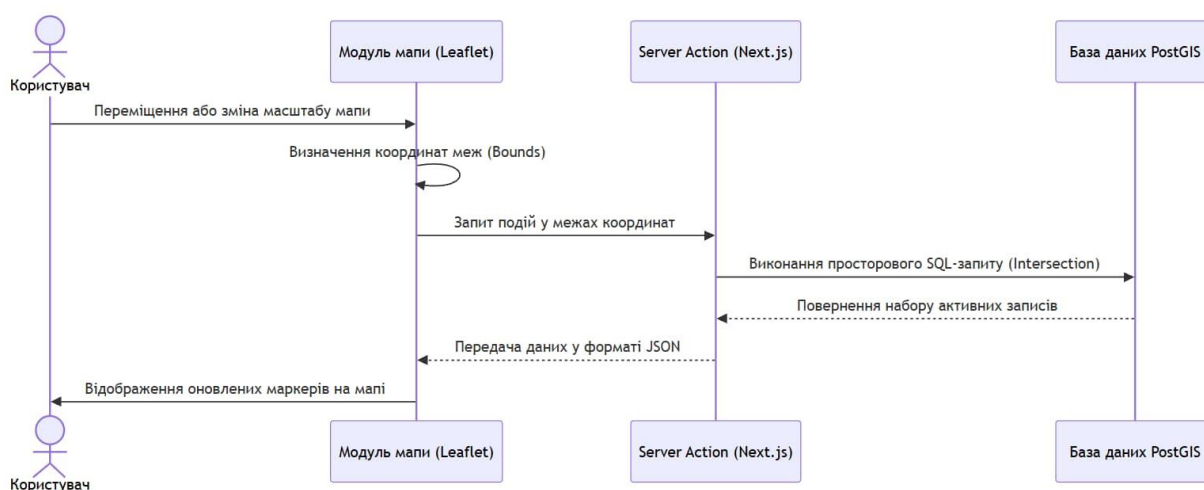


Рисунок 2.6 – Діаграма послідовності завантаження геопросторових даних

Специфікою проєктованого застосунку є робота з тимчасовими точками інтересу, що мають обмежений термін існування. Для детального аналізу поведінки системи залежно від часових параметрів було розроблено діаграму станів. Вона описує переходи об'єкта від моменту публікації до його остаточного переходу в архів або видалення.

Кожна точка інтересу проходить через кілька логічних статусів. Після створення подія може перебувати у стані очікування, якщо час її початку заплановано на майбутнє. При настанні визначеного терміну об'єкт автоматично переходить у статус активного відображення на мапі. Особливу

увагу приділено критичному стану завершення події: при досягненні кінцевої дати система припиняє видачу цього об'єкта у просторових запитах, переводячи його у неактивний стан. Це забезпечує актуальність інформаційного поля та запобігає захащенню мапи недійсними даними.

Логіку зміни статусів та умови переходів між ними наочно представлено на рис. 2.7.

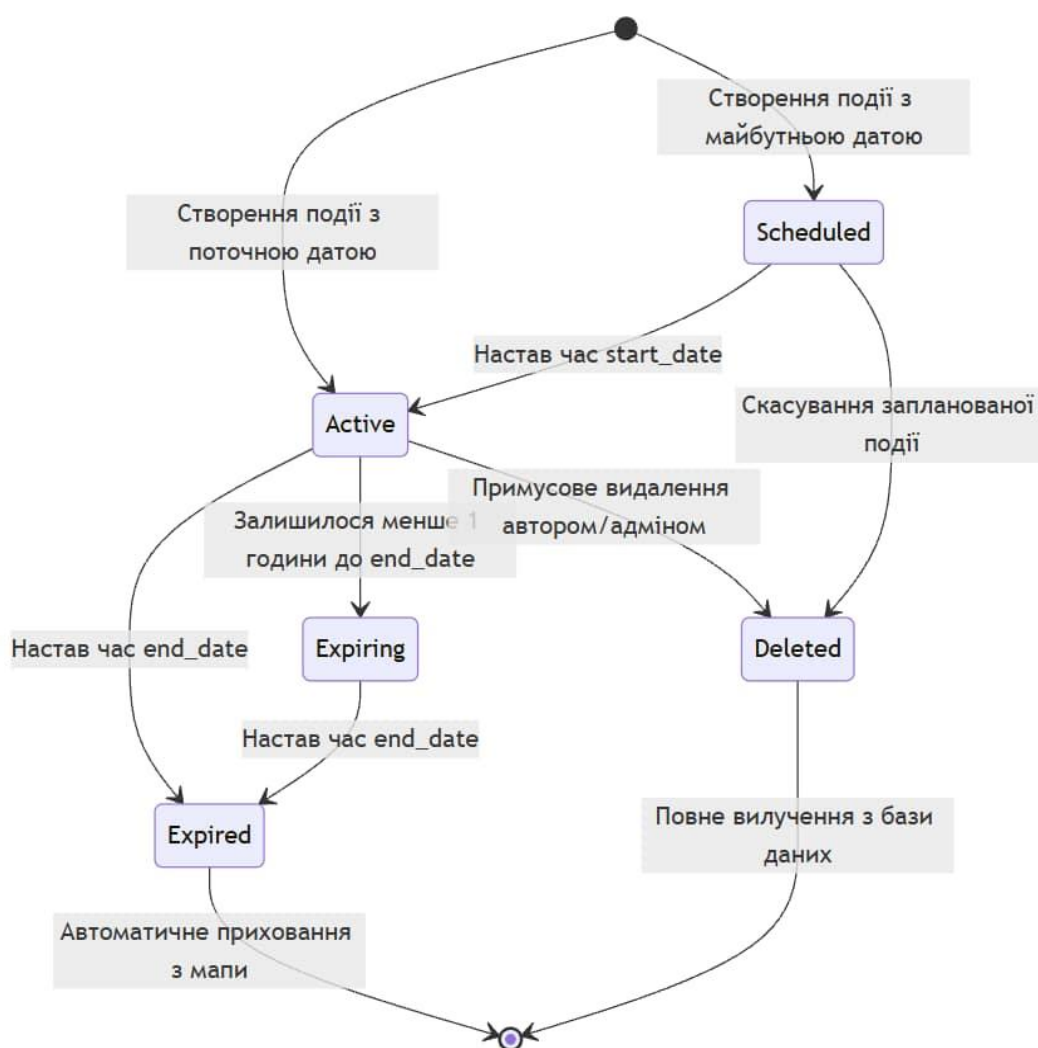


Рисунок 2.7 – Діаграма станів життєвого циклу точки інтересу

Основою збереження інформації геосоціального вебзастосунку є система управління реляційними базами даних PostgreSQL із просторовим розширенням PostGIS. Така архітектура забезпечує високий рівень надійності

та безпрецедентну швидкість виконання географічних обчислень. Логічна структура бази даних базується на кількох взаємопов'язаних таблицях.

Таблиця бази даних `users` визначає користувача програми за стовпцями:

- `id` – ідентифікатор користувача, який успадковується від сервісу автентифікації;
- `username` – унікальне ім'я, за яким відомий цей користувач серед інших учасників;
- `avatar_url` – текстове посилання на збережене зображення профілю користувача;
- `created_at` – дата та час реєстрації облікового запису;
- `is_admin` – логічне значення, що визначає наявність у користувача прав адміністратора для глобальної модерації контенту.

Точки інтересу та локальні події визначені таблицею `pois` за стовпцями:

- `id` – унікальний ідентифікатор локальної події;
- `creator_id` – користувач, який створив цю відмітку на мапі;
- `title` – загальна коротка назва події;
- `description` – деталізований опис заходу;
- `category` – категорія, до якої належить подія для забезпечення фільтрації;
- `location` – просторові координати розташування події у спеціальному географічному форматі;
- `start_date` – час початку запланованої активності;
- `end_date` – час завершення активності, після якого об'єкт приховується з мапи;
- `upvotes` – поточна кількість позитивних реакцій від спільноти;
- `downvotes` – поточна кількість негативних реакцій від спільноти;
- `created_at` – час публікації події у системі;
- `image_urls` – масив текстових посилань на прикріплені до події зображення.

Поле `creator_id` є зовнішнім ключем до таблиці бази даних `users`, таким чином закріплюючи те, що кожна подія може бути створена тільки одним користувачем, при цьому один користувач може створити від нуля і більше подій, утворюючи зв'язок типу один-до-багатьох.

Коментарі до точок інтересу визначені таблицею `comments` за стовпцями:

- `id` – ідентифікатор залишеного коментаря;
- `poi_id` – локальна подія, до якої було написано коментар;
- `user_id` – користувач, який залишив цей коментар;
- `content` – текстовий вміст повідомлення;
- `created_at` – час публікації повідомлення;
- `image_urls` – масив текстових посилань на зображення, прикріплені безпосередньо до коментаря.

Поле `poi_id` є зовнішнім ключем до таблиці бази даних `pois`, таким чином закріплюючи те, що кожен коментар стосується виключно однієї події, при цьому одна подія може мати від нуля і більше коментарів, формуючи зв'язок типу один-до-багатьох.

Поле `user_id` є зовнішнім ключем до таблиці бази даних `users`, таким чином закріплюючи те, що кожен коментар створюється тільки одним користувачем, при цьому один користувач може залишити від нуля і більше коментарів, утворюючи зв'язок типу один-до-багатьох.

Реакції користувачів для формування рейтингу подій визначені таблицею `reactions` за стовпцями:

- `poi_id` – ідентифікатор події, яка отримала реакцію;
- `user_id` – ідентифікатор користувача, який встановив реакцію;
- `type` – тип реакції, що набуває виключно двох значень: позитивного або негативного.

Комбінація полів `poi_id` та `user_id` слугує складеним первинним ключем, що унеможливорює залишення одним користувачем кількох реакцій

під однією подією. Поле `poi_id` є зовнішнім ключем до таблиці `pois`, а поле `user_id` є зовнішнім ключем до таблиці `users`.

Для забезпечення цілісності даних при роботі з таблицею реакцій спроектовано транзакційний алгоритм обробки соціальної взаємодії на рівні бази даних. Цей алгоритм керує зміною лічильників оцінок і запобігає дублюванню голосів. Він перевіряє наявність попередньої реакції користувача під конкретною подією: якщо її немає – створюється новий запис; якщо нова реакція збігається з існуючою – голос скасовується; у разі зміни типу реакції відбувається атомарне оновлення відповідних лічильників позитивних та негативних оцінок одночасно. Логіку роботи спроектованого алгоритму наведено на рис. 2.8.

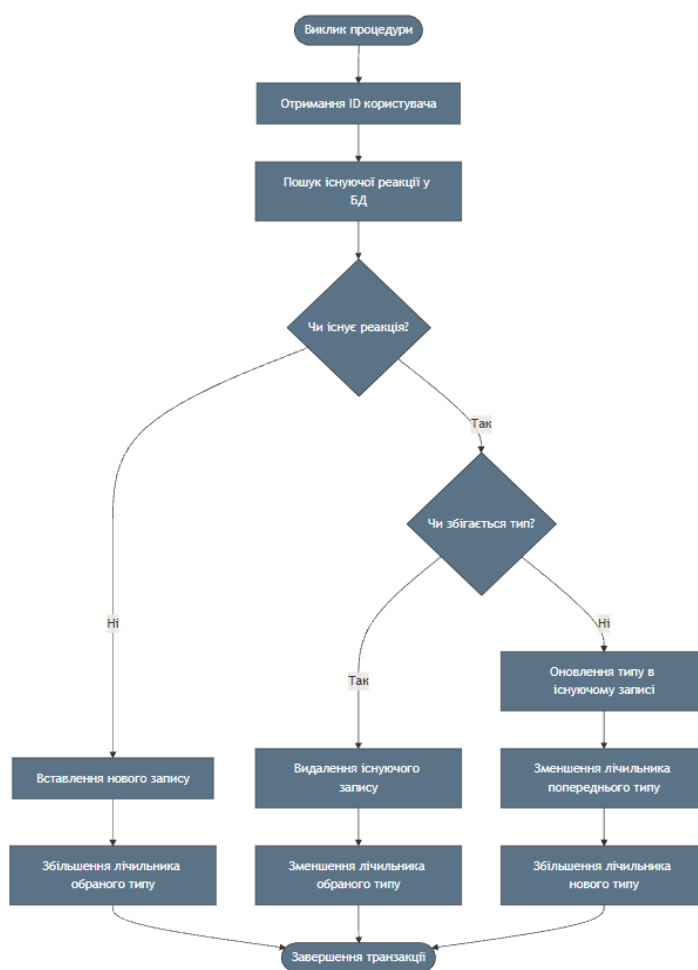


Рисунок 2.8 – Блок-схема алгоритму обробки соціальних реакцій у базі даних

Візуальна структура зв'язків між описаними сутностями бази даних представлена на рис. 2.9.

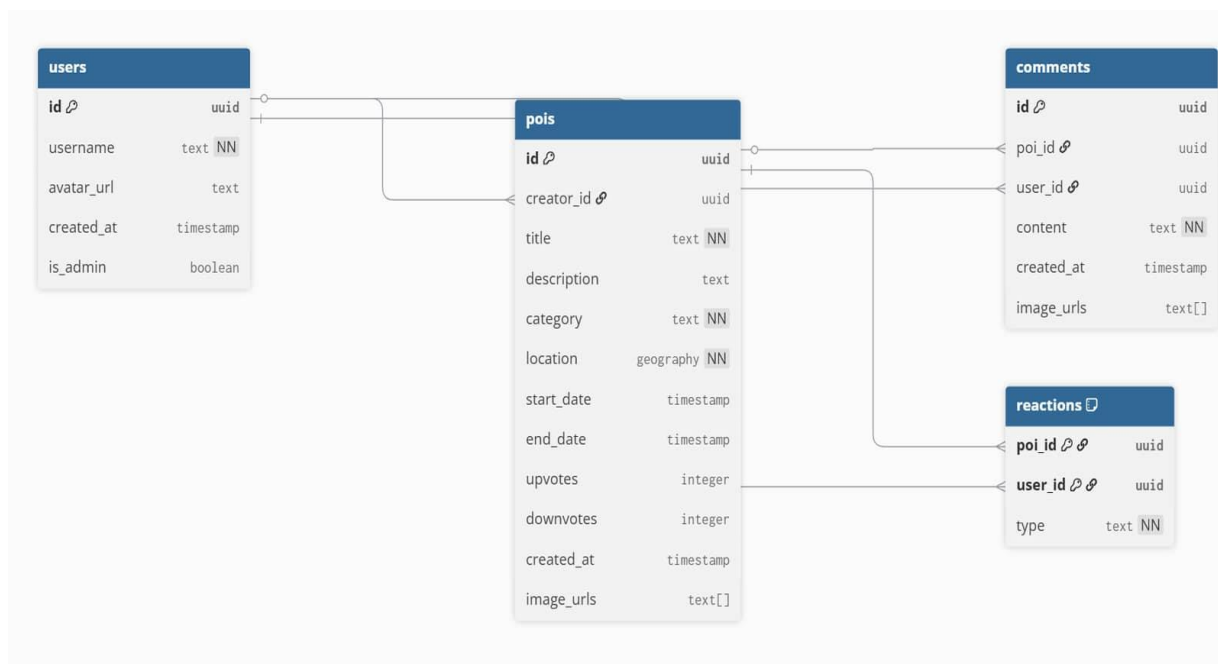


Рисунок 2.9 – Схема реляційної бази даних геосоціального застосунку

Для автоматизованого розгортання розробленої реляційної структури у вибраній системі управління базами даних було підготовлено комплексний SQL-скрипт. Виконання цього скрипта забезпечує послідовну ініціалізацію створення всіх необхідних таблиць із визначенням відповідних типів даних, встановлення строгих обмежень цілісності на рівні атрибутів та налаштування зв'язків між сутностями за допомогою зовнішніх ключів. Детальний код, що ілюструє процес ініціалізації бази даних, наведено нижче (лістинг 2.1 – 2.4).

Лістинг 2.1 - Лістинг таблиці comments

```

CREATE TABLE public.comments (
    id uuid NOT NULL DEFAULT gen_random_uuid(),
    poi_id uuid NOT NULL,
    user_id uuid NOT NULL,
    content text NOT NULL,
    created_at timestamp with time zone DEFAULT now(),
    image_urls ARRAY DEFAULT '{}'::text[],
    CONSTRAINT comments_pkey PRIMARY KEY (id),

```

```

        CONSTRAINT comments_poi_id_fkey FOREIGN KEY (poi_id)
REFERENCES public.pois(id),
        CONSTRAINT comments_user_id_fkey FOREIGN KEY (user_id)
REFERENCES public.users(id)

```

Лістинг 2.2 - Лістинг таблиці pois

```

CREATE TABLE public.pois (
    id uuid NOT NULL DEFAULT gen_random_uuid(),
    creator_id uuid NOT NULL,
    title text NOT NULL,
    description text,
    category text NOT NULL,
    location GEOGRAPHY(POINT, 4326) NOT NULL,
    upvotes integer DEFAULT 0,
    downvotes integer DEFAULT 0,
    created_at timestamp with time zone DEFAULT now(),
    image_urls ARRAY DEFAULT '{}':text[]
);

```

Лістинг 2.3 - Лістинг таблиці users

```

CREATE TABLE public.users (
    id uuid NOT NULL,
    username text NOT NULL UNIQUE,
    avatar_url text,
    created_at timestamp with time zone DEFAULT now(),
    is_admin boolean NOT NULL DEFAULT false,
    CONSTRAINT users_pkey PRIMARY KEY (id)
);

```

Лістинг 2.4 - Лістинг таблиці reactions

```

CREATE TABLE public.reactions (
    poi_id uuid NOT NULL,
    user_id uuid NOT NULL,
    type text NOT NULL CHECK (type = ANY (ARRAY['upvote':text,
'downvote':text])),
    CONSTRAINT reactions_pkey PRIMARY KEY (poi_id, user_id),
    CONSTRAINT reactions_user_id_fkey FOREIGN KEY (user_id)
REFERENCES public.users(id),
    CONSTRAINT reactions_poi_id_fkey FOREIGN KEY (poi_id)
REFERENCES public.pois(id)
);

```

Завершальним етапом проєктування архітектури стало розроблення низькодеталізованих прототипів інтерфейсу для візуалізації базового функціоналу та основних навігаційних переходів. Такий підхід дозволяє

зосередитися на структурі компонентів та логіці роботи застосунку без передчасної прив'язки до фінального графічного дизайну (рис. 2.10).

Розроблена схема ілюструє основний навігаційний потік клієнтського застосунку. Перший екран відображає базовий стан інтерфейсу з повноекранною картою, панеллю головної навігації, інструментами просторової фільтрації та формою створення нової точки інтересу. Навігаційний перехід ініціюється шляхом взаємодії користувача з маркером існуючої події на картографічному полотні. Ця дія перериває поточний сценарій перегляду та викликає відкриття деталізованого інформаційного вікна, яке накладено поверх карти.

Другий стан інтерфейсу демонструє модальне вікно з розширеною інформацією про обрану подію. Прототип цього екрана включає блок перегляду прикріплених медіафайлів, текстовий опис із визначеними часовими рамками існування події, а також модулі соціальної взаємодії для залишення реакцій та публікації коментарів. Запропоноване просторове компонування елементів та використання модальних вікон забезпечує швидкий доступ до інформації та підтримує безперервність роботи користувача із застосунком без необхідності повного перезавантаження сторінки.

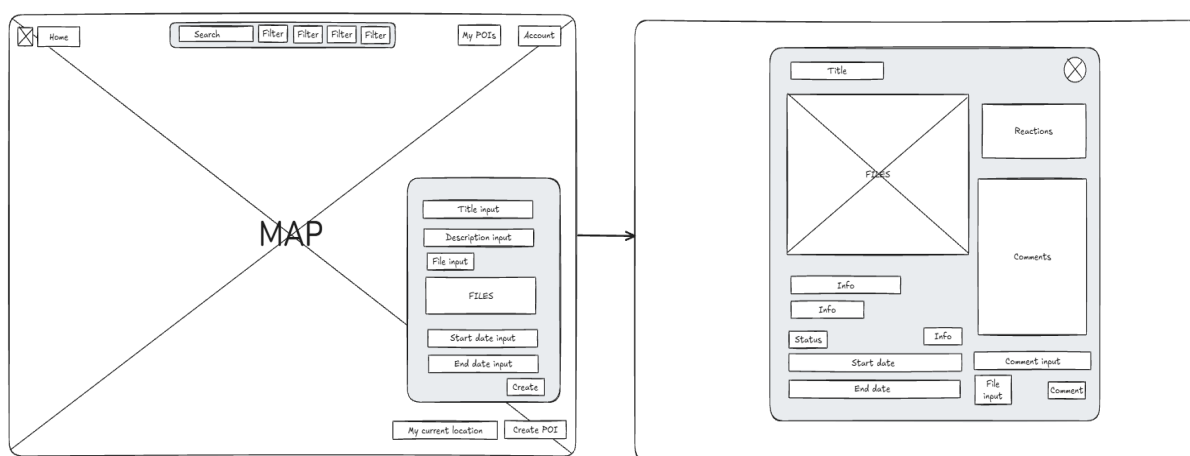


Рисунок 2.10 – Прототип інтерфейсу основного навігаційного потіку

2.4 Висновки за розділом 2

Обґрунтовано вибір сучасного гібридного технологічного стека на базі вебфреймворку Next.js та хмарної платформи Supabase, що забезпечує оптимальну продуктивність для обробки просторових даних та управління станом мапи порівняно з класичними роздільними архітектурами.

Виконано моделювання функціональних вимог до системи за допомогою діаграми прецедентів, де визначено межі геосоціального вебзастосунок та ієрархію прав доступу для трьох ключових ролей: неавторизованого відвідувача, авторизованого користувача та адміністратора.

Деталізацію системних взаємодій та бізнес-логіки програми виконано шляхом побудови діаграм компонентів, діяльності, станів та послідовності, а також розробки функціональної структури навігації і блок-схеми транзакційного алгоритму соціальної взаємодії. Зазначені моделі наочно ілюструють процеси створення тимчасових подій, обробки медіафайлів, динамічного завантаження географічних даних та управління рейтингами.

Окрім цього спроектовано логічну структуру реляційної бази даних із застосуванням просторового розширення PostGIS та наведено відповідні лістинги мовою структурованих запитів для ініціалізації таблиць. На завершення розроблено низькодеталізовані прототипи користувацького інтерфейсу головного екрана для візуалізації базового функціоналу та обґрунтування навігаційних переходів.

3 РЕАЛІЗАЦІЯ ПРОГРАМИ

3.1 Визначення порядку роботи програми

Функціональна схема геосоціального вебзастосунку базується на інтерактивному картографічному полотні. Це полотно виступає головним екраном та пропонується користувачу для роботи одразу після ініціалізації системи, починаючи з відображення активних локальних подій поблизу. Подальший порядок роботи визначається вибором відповідного напрямку дій: базові картографічні операції, управління точками інтересу або робота з профілем.

Кожен із цих логічних напрямків дозволяє переходити до вкладених функцій, таких як створення нових маркерів, публікація коментарів, фільтрація контенту або авторизація. Характерною рисою побудованої архітектури є циклічність: після виконання будь-якої цільової дії система забезпечує повернення користувача до головного картографічного інтерфейсу для продовження роботи або вибору іншого розділу.

3.2 Результати реалізації програмних одиниць застосунку

На рівні реляційної бази даних PostgreSQL із просторовим розширенням PostGIS визначено ключові збережені процедури, від яких безпосередньо залежить швидкодія клієнтської частини та безпека транзакцій.

Формування системи соціальної взаємодії реалізовано збереженою процедурою `toggle_reaction`. Вона приймає ідентифікатор точки інтересу та строковий параметр типу реакції, що набуває виключно двох значень для позначення позитивної або негативної оцінки. Алгоритм роботи процедури починається з отримання ідентифікатора поточного користувача через виклик вбудованої системної функції автентифікації. Далі виконується перевірка існування попередньої реакції у відповідній таблиці бази даних. Залежно від

результату перевірки алгоритм направляється за одним із трьох можливих сценаріїв.

Перший сценарій передбачає відсутність попередньої реакції. У такому разі вставляється новий запис, а лічильник загальних реакцій відповідного типу у головній таблиці подій збільшується на одиницю. Другий сценарій спрацьовує при збігу існуючої реакції з переданим типом, що ініціює видалення запису та зменшення лічильника, імітуючи механізм скасування голосу. Третій сценарій обробляє зміну типу реакції з одночасною корекцією обох лічильників позитивних та негативних оцінок. Задля забезпечення безпеки та цілісності даних під час атомарного оновлення процедура виконується з підвищеними привілеями розробника, що дозволяє обійти базові політики безпеки на рівні рядків.

Отримання геопросторових даних оптимізовано за допомогою функції `get_pois_in_bounds`, яка призначена для просторової вибірки. Функція приймає чотири дробові параметри, що описують координати кутів видимої області екрана застосунку. Тіло функції містить запит до основної таблиці подій з умовою геометричного перетину стовпця локації із прямокутним полігоном, побудованим на основі переданих меж. Завдяки використанню індексу узагальненого дерева пошуку, побудованого по стовпцю просторових координат, забезпечується висока швидкість вибірки об'єктів навіть за великого обсягу збережених даних. Додатково застосовується фільтрація за часом завершення події, що гарантує повернення виключно актуальних записів, для яких термін дії ще не минув або має невизначене значення.

Розширення базової моделі безпеки реалізовано через відповідний файл міграції бази даних. Завдяки цій міграції до таблиці користувачів додається логічний атрибут прав адміністратора зі значенням за замовчуванням, яке забороняє розширений доступ. Зміни також зачіпають політики безпеки на рівні рядків для таблиць подій та коментарів. Оновлені правила дозволяють виконання операції видалення запису як його безпосередньому автору, так і будь-якому користувачу з активним статусом

адміністратора, що забезпечує гнучкий механізм глобальної модерації контенту.

Основна логіка створення, отримання і видалення точок інтересу зосереджена у файлі `src/app/actions/poi.ts`, який оголошено з директивою серверного виконання `use server`. Такий підхід гарантує, що чутливі операції та прямі запити до бази даних не потраплять до клієнтського коду.

Локальна допоміжна функція `parseLocation` відповідає за перетворення просторових даних із внутрішнього формату PostGIS у придатний для фронтенду вигляд (лістинг 3.1). База даних зберігає географічні координати як бінарні об'єкти, а через інструменти Supabase вони надходять у вигляді шістнадцяткових рядків. Алгоритм функції `parseLocation` приймає такий рядок, створює буфер даних і передає його до бібліотеки просторового аналізу, результат якого конвертується у стандартну географічну структуру GeoJSON з масивом координат. Якщо процес перетворення завершується невдало, функція повертає вхідне значення без змін, що дозволяє уникнути аварійного завершення роботи застосунку.

Лістинг 3.1 - Функція `parseLocation` для конвертації WKB-даних у формат GeoJSON

```
function parseLocation(loc: any) {
  try {
    const geometry = wkx.Geometry.parse(Buffer.from(loc,
'hex'))
    return geometry.toGeoJSON()
  } catch (e) {
    console.error('Error parsing WKB location:', e)
    return loc
  }
  return loc
}
```

Отримання списку точок інтересу в межах видимості екрана реалізовано функцією `getPoisInBounds`, яка приймає координати поточного вікна перегляду мапи у вигляді чотирьох числових аргументів і повертає масив типізованих об'єктів `Poi`. Спочатку формується виклик збереженої процедури

`get_pois_in_bounds` через метод віддаленого виклику. У випадку помилки функція переходить до резервної стратегії. Ця стратегія полягає у вибірці всіх записів з таблиці подій, значення `end_date` яких перевищує поточну дату, після чого їхні координати перетворюються у числовий формат і фільтруються за заданими межами. Такий подвійний підхід забезпечує безперебійну працездатність застосунку на етапі розгортання.

Комплексний процес створення нової точки інтересу обробляється функцією `createPoi` (лістинг 3.2), яка приймає об'єкт `FormData`. На початку виконання алгоритму перевіряється статус автентифікації користувача через запит `getUser`. Далі з форми витягуються текстові поля `title`, `description`, `category` та координати `lat` і `lng`, які одразу перетворюються у числа з рухомою комою. Окремому аналізу піддається масив файлів із ключем `images`. Для кожного файлу генерується унікальне ім'я, що складається з мітки часу у мілісекундах, випадкового буквено-цифрового суфіксу та очищеного оригінального імені файлу.

Процес збереження нової події розпочинається з асинхронного завантаження прикріпленого медіафайлу до хмарного бакета `poi-images`, після чого система отримує його публічну адресу. Далі географічні координати маркера форматуються у стандарт WKB шляхом створення просторового об'єкта з чітко встановленим ідентифікатором системи координат SRID 4326 та його подальшим перетворенням у шістнадцятковий рядок. Підготовлений таким чином об'єкт транзакційно вставляється у таблицю `pois` із збереженням ідентифікатора `creator_id`, валідованих текстових полів, перетворених у строгий ISO-формат дат `startDate` і `endDate` та масиву отриманих адрес зображень.

Після успішної вставки ініціюється скидання серверного кешу викликом функції `revalidatePath` для маршрутів перегляду мапи, що гарантує миттєве відображення нової точки для всіх користувачів.

Лістинг 3.2 – Частина функції createPoi для обробки та збереження просторового об'єкта

```

const title = formData.get('title') as string
const description = formData.get('description') as string
const category = formData.get('category') as string
const lat = parseFloat(formData.get('lat') as string)
const lng = parseFloat(formData.get('lng') as string)
const startDate = formData.get('startDate') as string
const endDate = formData.get('endDate') as string
const imageFiles = formData.getAll('images') as File[]
const imageUrls: string[] = []

for (const file of imageFiles) {
  if (file.size === 0) continue
  const fileName = `${Date.now()}-${
    file.name.replace(/^[^a-zA-Z0-9.-]/g, '_')}`

  const { data: uploadData, error: uploadError } = await
supabase.storage
    .from('poi-images')
    .upload(fileName, file)
  const { data: publicUrlData } = storage.from('poi-
images').getPublicUrl(fileName)
  imageUrls.push(publicUrlData.publicUrl)
}

const point = new wkx.Point(lng, lat)
point.srid = 4326
const locationWkb = point.toWkb().toString('hex')
const { error } = await supabase.from('pois').insert({
  creator_id: user.id,
  title,
  description,
  category,
  location: locationWkb,
  start_date: startDate ? new
Date(startDate).toISOString() : null,
  end_date: endDate ? new Date(endDate).toISOString() :
null,
  image_urls: imageUrls
})
if (error) {
  console.error('Failed to create POI:', error)
  throw new Error('Failed to create POI')
}
revalidatePath('/map')
revalidatePath('/my-pois')
return { success: true }
}

```

Видалення точки інтересу виконує функція `deletePoi`, що приймає строковий ідентифікатор запису `id`. Перед виконанням цієї деструктивної операції з таблиці `users` витягується логічний прапорець `is_admin` для поточного авторизованого користувача. Запит на видалення будується поетапно: спочатку створюється базовий ланцюжок звернення до таблиці `pois`, а потім, якщо користувач не має розширених прав, до запиту додається додатковий фільтр перевірки поля `creator_id`. Завдяки такому підходу адміністратор системи має можливість видалити будь-який запис, тоді як звичайний користувач може вилучити виключно власні публікації.

Логіку роботи з коментарями, соціальними реакціями та отримання детальної інформації про просторові об'єкти винесено в окремий файл серверних дій `src/app/actions/interaction.ts`.

Функція `addComment` приймає рядковий ідентифікатор події `poiId` та об'єкт `FormData`, з якого витягується обов'язковий текст `content` та масив файлів зображень. Процес обробки медіафайлів ідентичний алгоритму створення точки інтересу: генерується унікальне ім'я, файл завантажується у бакет `poi-images`, після чого формується масив публічних адрес. Новий запис вставляється у таблицю `comments` із зазначенням ідентифікатора автора `user_id`. Завершується операція інвалідацією кешу сторінки події за допомогою методу `revalidatePath`.

Вибірка коментарів до заданої точки інтересу виконується функцією `getComments`. Запит звертається до таблиці `comments` із явним об'єднанням із таблицею `users` для отримання полів `username` та `avatar_url` автора. Результати фільтруються за полем `poi_id` і впорядковуються за часом створення `created_at` у зростаючому порядку, що забезпечує коректне хронологічне відображення нитки обговорення.

Видалення коментаря реалізовано у функції `deleteComment` за шаблоном, аналогічним до видалення події. Спочатку перевіряється статус `is_admin`, після чого формується запит на видалення з умовним фільтром: операція виконується або виключно за ідентифікатором коментаря для

адміністратора, або з додатковою перевіркою авторства для звичайного користувача.

Функція `toggleReaction` обробляє запити голосування. Вона приймає ідентифікатор події та тип реакції, після чого ініціює виклик однойменної збереженої процедури `toggle_reaction` через метод `grpc` клієнта бази даних. Уся транзакційна логіка перемикавання голосів виконується на стороні бази даних атомарно. Отримання повного набору даних конкретної точки реалізовано у функції `getPoiDetails`, яка виконує запит до таблиці `pois` із об'єднанням через зовнішній ключ для вибірки імені автора. Результат обробляється функцією `single`, що гарантує повернення єдиного об'єкта замість масиву.

Процеси автентифікації та реєстрації користувачів зосереджені у файлі `src/app/auth/actions.ts`. Функція `login` отримує електронну пошту `email` та пароль `password`, передаючи їх у метод `signInWithPassword`. У разі невдалої спроби виконується перенаправлення на сторінку входу з текстовим повідомленням про помилку, закодованим через глобальну функцію `encodeURIComponent` у параметрі `message` адресного рядка. При успішній автентифікації глобальний кеш застосунку скидається викликом `revalidatePath`, і користувач автоматично перенаправляється на інтерактивну мапу.

Функція `signup` додатково приймає поле `username` (лістинг 3.3). Перед зверненням до інструментів автентифікації виконується перевірка унікальності обраного імені. Для цього формується запит до таблиці `users` із параметром конфігурації, який повертає лише лічильник існуючих записів без завантаження самих даних. Якщо ім'я вже існує, процес переривається перенаправленням із відповідним сповіщенням. Після успішного створення запису в системі безпеки функція автоматично генерує публічний профіль у таблиці `users`. Вихід із системи реалізовано у функції `logout`, яка викликає метод `signOut` та очищує сеанс користувача на рівні файлів `cookie`.

Лістинг 3.3 – Програмна реалізація алгоритму реєстрації нового користувача у файлі auth/actions.ts

```

export async function signup(formData: FormData) {
  const supabase = await createClient()
  const email = formData.get('email') as string
  const password = formData.get('password') as string
  const username = formData.get('username') as string

  const { count } = await supabase
    .from('users')
    .select('id', { count: 'exact', head: true })
    .eq('username', username)

  if (count && count > 0) {
    redirect('/register?message='
+ encodeURIComponent('Username is already taken'))
  }

  const { error, data } = await supabase.auth.signUp({
    email,
    password,
  })

  if (error) {
    redirect('/register?message='
+ encodeURIComponent(error.message))
  }

  if (data.user) {
    const { error: profileError } = await
supabase.from('users').insert({
      id: data.user.id,
      username: username,
    })

    if (profileError) {
      console.error('Profile creation failed:',
profileError)
    }
  }

  if (!data.session) {
    redirect('/login?message='+
encodeURIComponent('Registration successful! Please check your
email to confirm your account (or disable email confirmation in
Supabase).'))
  }

  revalidatePath('/', 'layout')
  redirect('/map')
}

```

Управління налаштуваннями профілю обробляється у файлі `src/app/account/actions.ts`. Функція `updateUsername` отримує нове ім'я, видаляє зайві пробіли та оновлює запис у таблиці `users` (лістинг 3.4). При перехопленні помилки унікальності інформує користувача про те, що обране ім'я вже зайняте. Зміна облікових даних реалізована функціями `updatePassword` та `updateUsername`, які передають нові значення у метод `updateUser` сервісу автентифікації. Після виконання будь-якої з цих дій відбувається перенаправлення на сторінку налаштувань із передачею статусу операції через параметри запиту для його подальшого відображення інтерфейсним компонентом сповіщень.

Лістинг 3.4 – Функція `updateUsername`

```
export async function updateUsername(formData: FormData) {
  const username = (formData.get('username') as
string)?.trim()
  if (!username) redirect('/account?message=Username is
required')

  const supabase = await createClient()
  const { data: { user } } = await supabase.auth.getUser()
  if (!user) redirect('/login')

  const { error } = await supabase
    .from('users')
    .update({ username })
    .eq('id', user.id)

  if (error) {
    const msg = error.code === '23505'
      ? 'Username is already taken'
      : 'Failed to update username: ' + error.message
    redirect('/account?message=' + encodeURIComponent(msg))
  }

  redirect('/account?message=Username updated
successfully')
}
```

Візуалізація даних та безпосередня взаємодія з користувачем забезпечуються клієнтськими компонентами фреймворку `React`. Головна сторінка застосунку `src/app/page.tsx` використовує серверний рендеринг для

перевірки стану автентифікації та відображає інтерактивну фонову мапу. Вона реалізована через компонент-обгортку `HeroMapWrapper` із відключеним серверним рендерингом для запобігання конфліктам бібліотеки картографії із серверним середовищем `Node.js`.

Сторінка мапи `src/app/map/page.tsx` також ініціалізується на сервері, отримуючи профіль користувача, після чого передає ці дані до головного клієнтського компонента `InteractiveMap`. Цей компонент інкапсулює всю логіку роботи з картографічним рушієм `Leaflet`. Стан компонента зберігає поточні межі видимості, координати центру та масштаб, які автоматично синхронізуються з локальним сховищем браузера `localStorage`. При зміні меж екрана викликається функція завантаження точок інтересу `fetchPois`, яка також здійснює локальну фільтрацію результатів за категоріями та пошуковими запитами. Формування маркерів на мапі здійснюється шляхом генерації HTML-елементів за допомогою методу `divIcon` з відповідними кольоровими ідентифікаторами категорій. Модальне вікно створення події інтегрує бібліотеку `browser-image-compression`, що дозволяє оптимізувати розмір фотографій безпосередньо на пристрої клієнта перед відправкою на сервер.

Система пошуку реалізована у компоненті `MapFilterBar`. Він забезпечує подвійний механізм обробки запитів: локальну фільтрацію за назвами вже завантажених подій та глобальне геокодування через зовнішній інтерфейс програмування застосунків `Nominatim`. Усі активні фільтри та пошукові запити синхронізуються з параметрами адресного рядка, що гарантує збереження поточного стану інтерфейсу при оновленні сторінки та дозволяє ділитися прямими посиланнями на конкретні результати пошуку.

Детальна інформація про просторовий об'єкт відображається компонентом `PoiDetailPanel`, який паралельно завантажує дані точки інтересу та масив коментарів до неї через конструкцію `Promise.all`. Компонент динамічно розраховує поточний статус події на основі часових рамок, присвоюючи індикатори очікування, активності або завершення. Крім того, він підтримує механізм оптимістичного оновлення інтерфейсу: при додаванні

чи видаленні коментарів зміни миттєво відображаються на екрані, не чекаючи підтвердження від сервера, що значно покращує показники чуйності інтерфейсу.

Захист маршрутів застосунку та управління сеансами реалізовано на рівні проміжного програмного забезпечення у файлі `src/middleware.ts` (лістинг 3.5). Цей модуль перехоплює всі вхідні HTTP-запити, оновлює токени доступу у файлах `cookie` за допомогою клієнта Supabase та виконує примусове перенаправлення неавторизованих користувачів на сторінку входу при спробі доступу до захищених розділів керування профілем.

Лістинг 3.5 – Частина реалізації проміжного програмного забезпечення для контролю доступу та оновлення сесій

```
const supabase = createServerClient(
  process.env.NEXT_PUBLIC_SUPABASE_URL!,
  process.env.NEXT_PUBLIC_SUPABASE_ANON_KEY!,
  {
    cookies: {
      getAll()
      {
        return request.cookies.getAll()
      },
      setAll(cookiesToSet) {
        cookiesToSet.forEach(({ name, value }) =>
request.cookies.set(name, value))
        supabaseResponse = NextResponse.next(
          {
            request,
          })
        cookiesToSet.forEach(({ name, value, options })
=>
supabaseResponse.cookies.set(name, value, options)
        )
      },
    },
  )

const {
  data: { user },
} = await supabase.auth.getUser()

const url = request.nextUrl.clone()
url.pathname = '/login'
return NextResponse.redirect(url)
)
```

3.3 Висновки за розділом 3

Розроблено функціональну схему геосоціального вебзастосунку, яка базується на просторовій навігації та ітеративній взаємодії користувача з інтерактивною мапою. Детально описано програмну реалізацію архітектурних рішень на всіх рівнях системи.

На рівні бази даних реалізовано збережені процедури просторової вибірки за допомогою функцій PostGIS та розроблено механізми транзакційної обробки соціальних реакцій з обходом обмежень безпеки на рівні рядків. На серверному рівні створено захищені контролери на базі серверних дій Next.js для керування точками інтересу, обробки та збереження медіафайлів у хмарному об'єктному сховищі, а також безпечної автентифікації користувачів. На клієнтському рівні реалізовано інтерактивний картографічний інтерфейс із динамічним оновленням меж, інтеграцією зовнішнього сервісу відкритого геокодування та синхронізацією стану пошукових фільтрів з параметрами адресного рядка.

Таким чином, у ході розробки було успішно реалізовано всі заявлені функції програми, налаштовано безпечну взаємодію між клієнтським і серверним середовищами та вичерпно описано всі створені програмні одиниці застосунку.

4 ЕКСПЛУАТАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМ

4.1 Призначення програми

Розроблений геосоціальний вебзастосунок призначений для створення інтерактивного простору пошуку однодумців та організації локальних тимчасових заходів. Програма забезпечує користувачам можливість створювати власні точки інтересу з прив'язкою до географічних координат, відстежувати активність інших учасників на інтерактивному картографічному полотні, а також взаємодіяти зі спільнотою шляхом публікації текстових коментарів та залишення соціальних реакцій. Крім того, система призначена для управління власним профілем, динамічної фільтрації подій за категоріями та часовими рамками, а також для здійснення глобальної модерації контенту користувачами з відповідними правами доступу.

4.2 Умови виконання програми

Обов'язковою умовою експлуатації геосоціального вебзастосунку є наявність активного підключення до мережі Інтернет та попереднє розгортання серверної частини. До обов'язків адміністратора системи входить первинне налаштування реляційної бази даних із просторовим розширенням, конфігурація політик безпеки хмарного об'єктного сховища медіафайлів та подальша глобальна модерація створеного користувачами контенту.

Для коректної роботи серверної частини та системи управління базами даних технічне забезпечення сервера повинно відповідати таким мінімальним вимогам:

- центральний процесор з мінімум двома обчислювальними ядрами та базовою тактовою частотою від 2,4 ГГц;
- оперативний запам'ятовуючий пристрій обсягом не менше 4 Гб для забезпечення швидкого виконання просторових обчислень рушієм бази даних;

- твердотільний накопичувач інформації обсягом від 50 гігабайт для збереження медіафайлів та операційної системи;
- широкосмугове підключення до мережі Інтернет зі швидкістю передачі даних від 100 мегабіт за секунду.

Умови експлуатації клієнтської частини застосунку вимагають від користувача наявності сучасного браузера з увімкненою підтримкою виконання клієнтських сценаріїв. Крім того, для повноцінного використання функцій автоматичного центрування мапи пристрій користувача повинен підтримувати апаратні або програмні інтерфейси визначення геолокації.

4.3 Виконання і тестування програми

Реалізацію тестування забезпечено в межах перевірки набору функціональних можливостей, які повинні були бути забезпеченими для роботи користувача. Тестування продемонструвало успішні результати виконання всіх запланованих системних алгоритмів, коректну обробку просторових даних та стабільну роботу клієнт-серверної взаємодії. Далі на основі результатів тестування описано приклади і логіку роботи з програмою.

Стартова сторінка застосунку виконує важливу роль концептуальної візитної картки системи, формуючи перше враження користувача під час його початкового візиту. Її інтерфейс логічно структурований і містить головний інформаційний блок із назвою проєкту, лаконічним описом його ключових функціональних переваг та інтерактивними елементами заклику до дії. Естетичним фоном для цієї сторінки слугує стилізована декоративна неінтерактивна мапа, яка одразу занурює користувача у відповідну тематичну атмосферу геосоціального застосунку. Завдяки продуманій системі навігації, з цієї початкової сторінки неавторизовані відвідувачі можуть швидко перейти до процесу створення нового облікового запису, тоді як уже зареєстровані користувачі отримують змогу здійснити перехід безпосередньо до головного робочого простору (рис. 4.1).

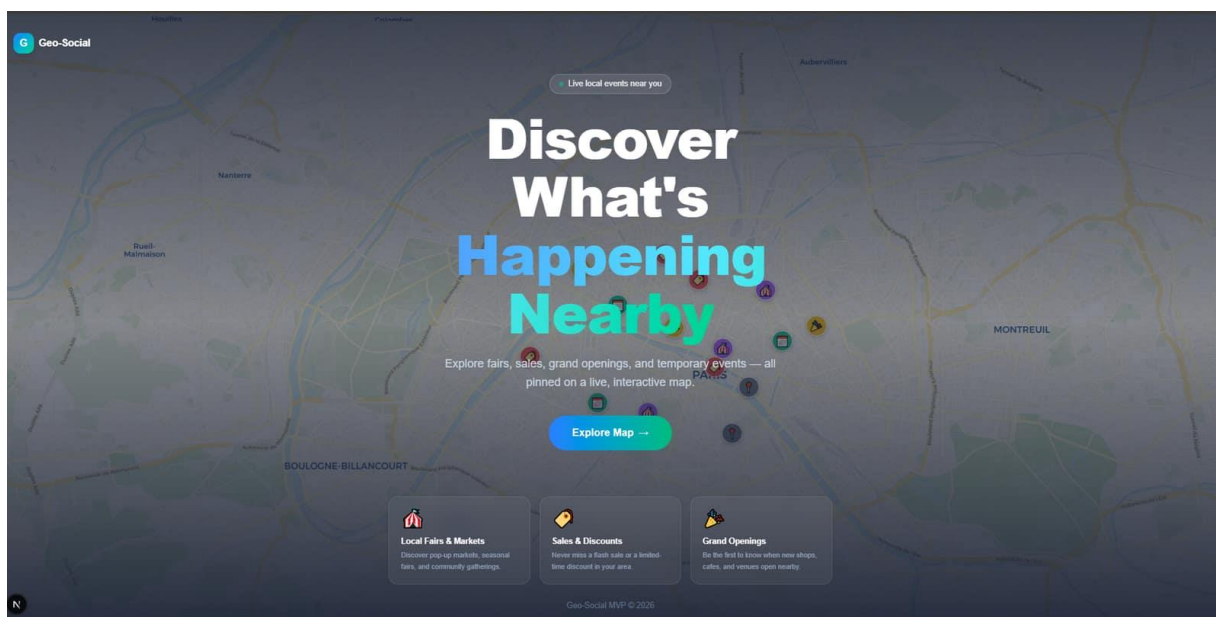


Рисунок 4.1 – Стартова сторінка геосоціального застосунку

Для отримання доступу до повного функціоналу системи відвідувач має пройти процес реєстрації або авторизації (рис. 4.2). Відповідна сторінка містить форми для введення облікових даних та реалізує механізми перевірки унікальності імені користувача. У разі виникнення помилок введення або спроби використання вже зареєстрованої електронної пошти система генерує відповідні попереджувальні повідомлення для користувача.

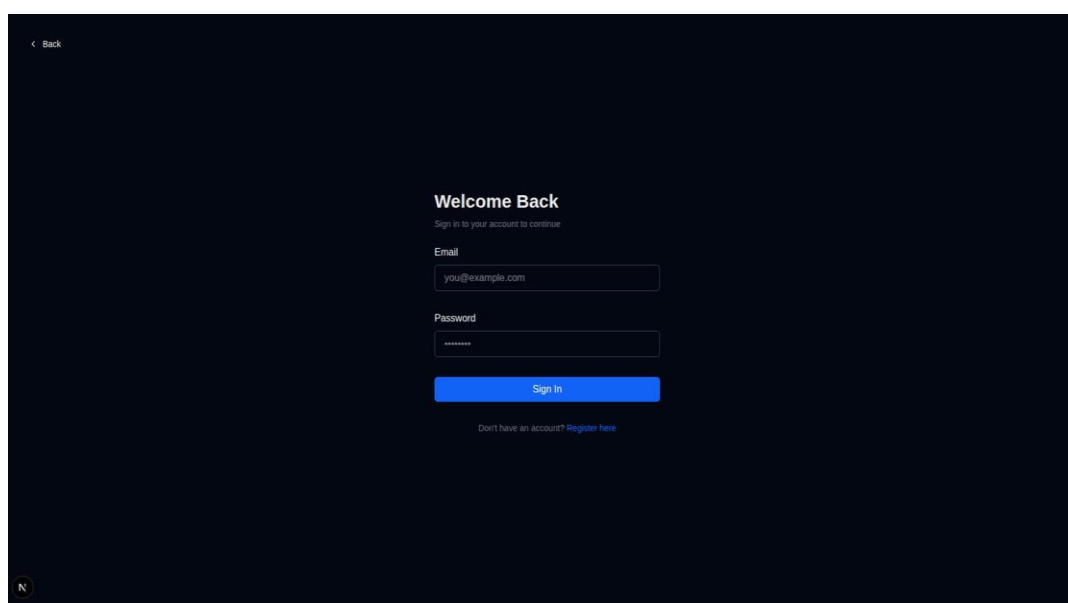


Рисунок 4.2 – Сторінка авторизації та реєстрації облікового запису

Після успішної автентифікації або переходу з головного екрана користувач потрапляє на сторінку інтерактивної мапи (рис. 4.3), яка є основним робочим середовищем системи. На ній відображаються маркери актуальних локальних подій, які завантажуються динамічно залежно від поточних координат меж екрана. Користувач отримує повний набір інструментів для взаємодії: можливість змінювати масштаб, переміщувати огляд, а також використовувати панель у верхній частині інтерфейсу для просторової фільтрації або пошуку. Взаємодія з будь-яким маркером ініціює відкриття детальної панелі з розширеною інформацією про обрану точку інтересу.

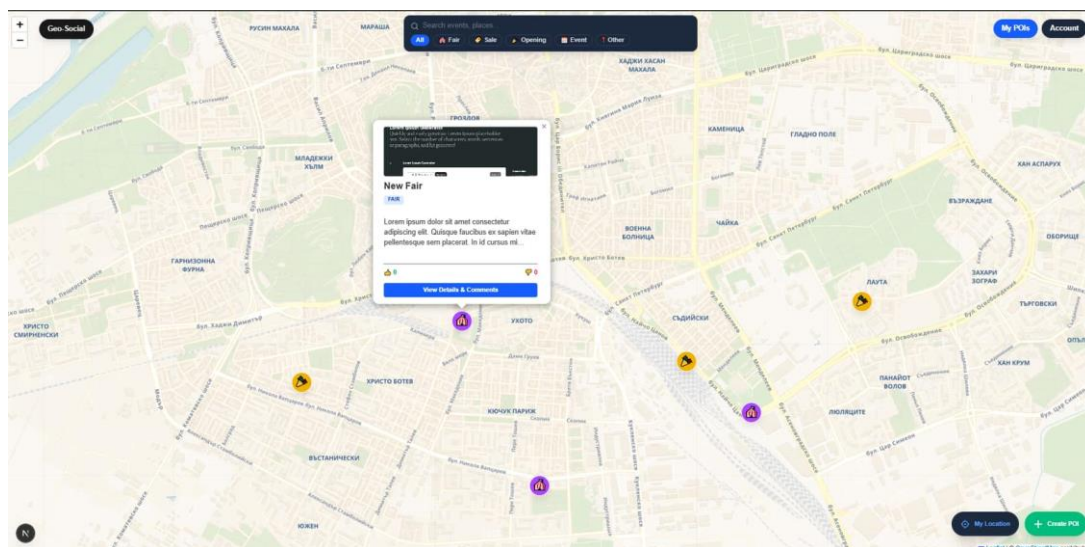


Рисунок 4.3 – Робоча сторінка інтерактивної мапи та фільтрації даних

Вибір конкретного маркера на мапі відкриває модальне вікно з детальною інформацією про обрану точку інтересу (рис. 4.4). Інтерфейс цієї панелі надає вичерпні дані про локацію, включаючи текстовий опис, галерею прикріплених зображень та індикатор статусу події. Крім того, саме тут реалізовано блок соціальної взаємодії: користувачі можуть переглядати коментарі, публікувати власні повідомлення та впливати на загальний рейтинг події шляхом додавання реакцій вподобання або невподобання.

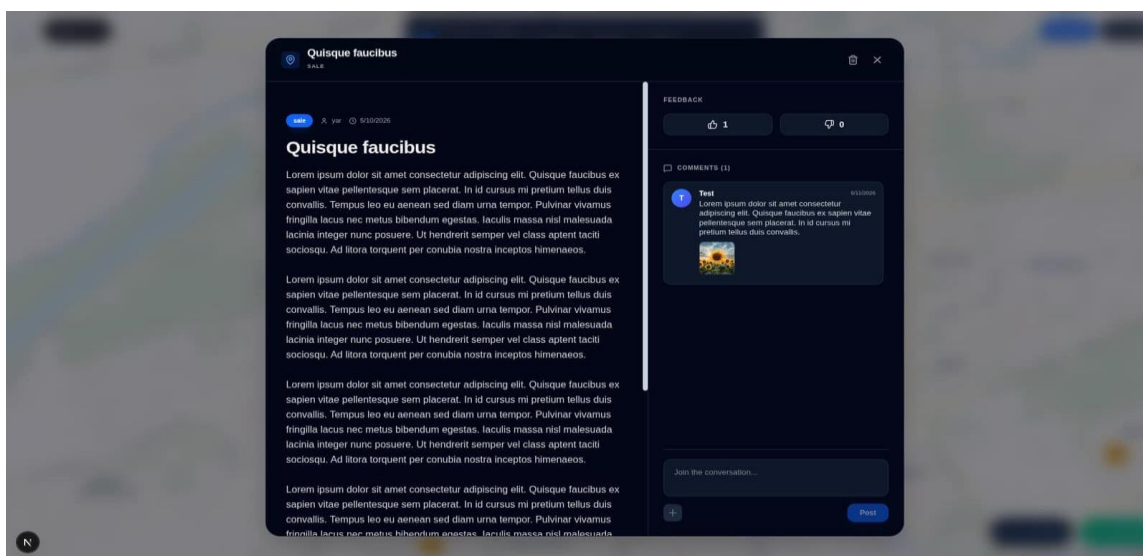


Рисунок 4.4 – Панель детальної інформації про подію та соціальної взаємодії

Для створення нової події користувач ініціює відповідний режим та фіксує координати безпосередньо на картографічній підоснові, після чого система викликає модальне вікно форми створення (рис. 4.5). У цьому вікні необхідно заповнити текстові поля назви та опису, обрати категорію з наданого переліку, встановити обов'язкові часові межі існування події та, за потреби, прикріпити ілюстративні фотографії. Система автоматично стискає завантажені зображення та перевіряє логічну коректність введених дат перед відправкою пакета даних на сервер.

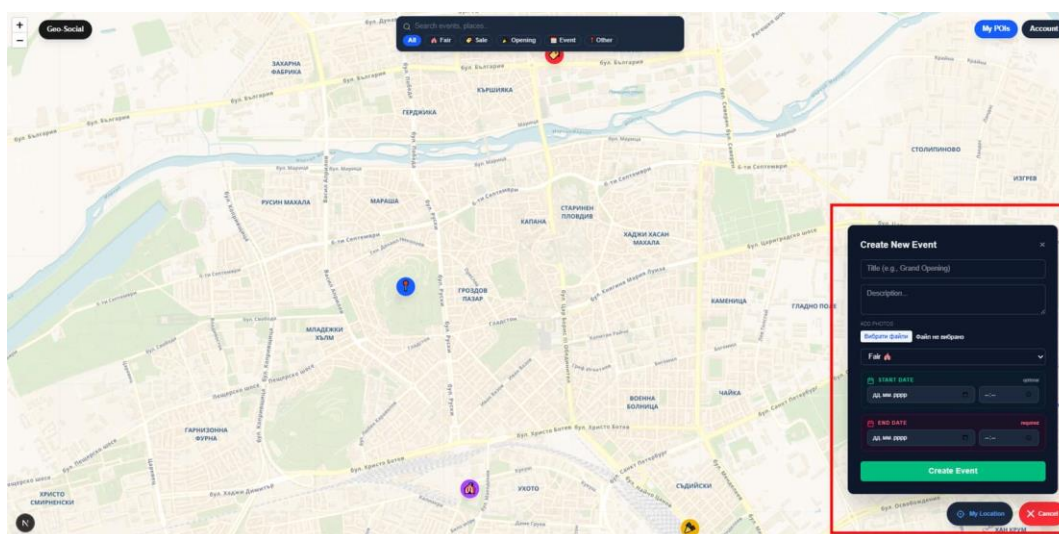


Рисунок 4.5 – Форма створення нової точки інтересу на мапі

Розділ керування власними публікаціями дозволяє авторизованому користувачу переглядати історію створених ним тимчасових подій (рис. 4.6). Інтерфейс цього розділу реалізовано у вигляді сітки карток, кожна з яких містить базову текстову інформацію, індикатори часових рамок, статистику отриманих соціальних реакцій від спільноти, а також навігаційні елементи. За допомогою цих елементів користувач може здійснити швидке центрування мапи на обраній події або виконати її безповоротне видалення з бази даних.

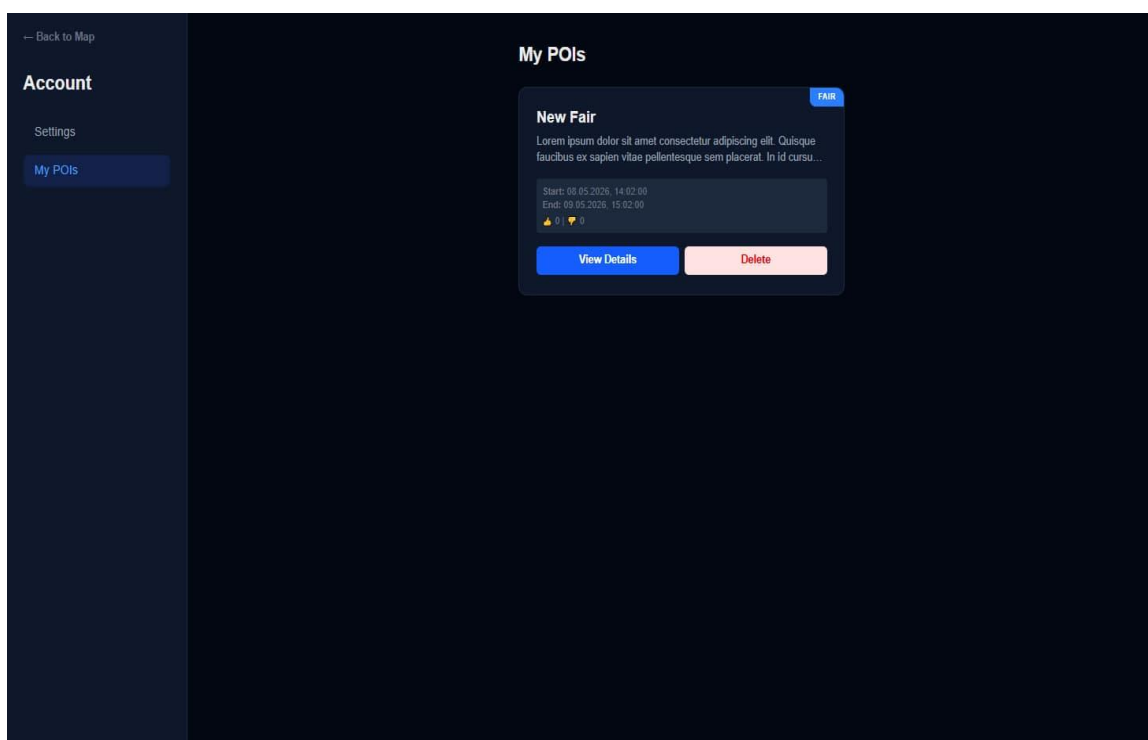


Рисунок 4.6 – Сторінка перегляду власних точок інтересу користувача

Контроль за персональними даними та налаштування профілю здійснюються на окремій сторінці облікового запису (рис. 4.7). Тут користувач отримує можливість змінити своє публічне ім'я або встановити новий пароль для доступу до системи. Кожна форма обладнана механізмом перевірки на стороні сервера, який сповіщає користувача про успішне збереження змін або наявність помилок. У нижній частині інтерфейсу розташовано елемент для безпечного завершення сеансу та виходу з облікового запису.

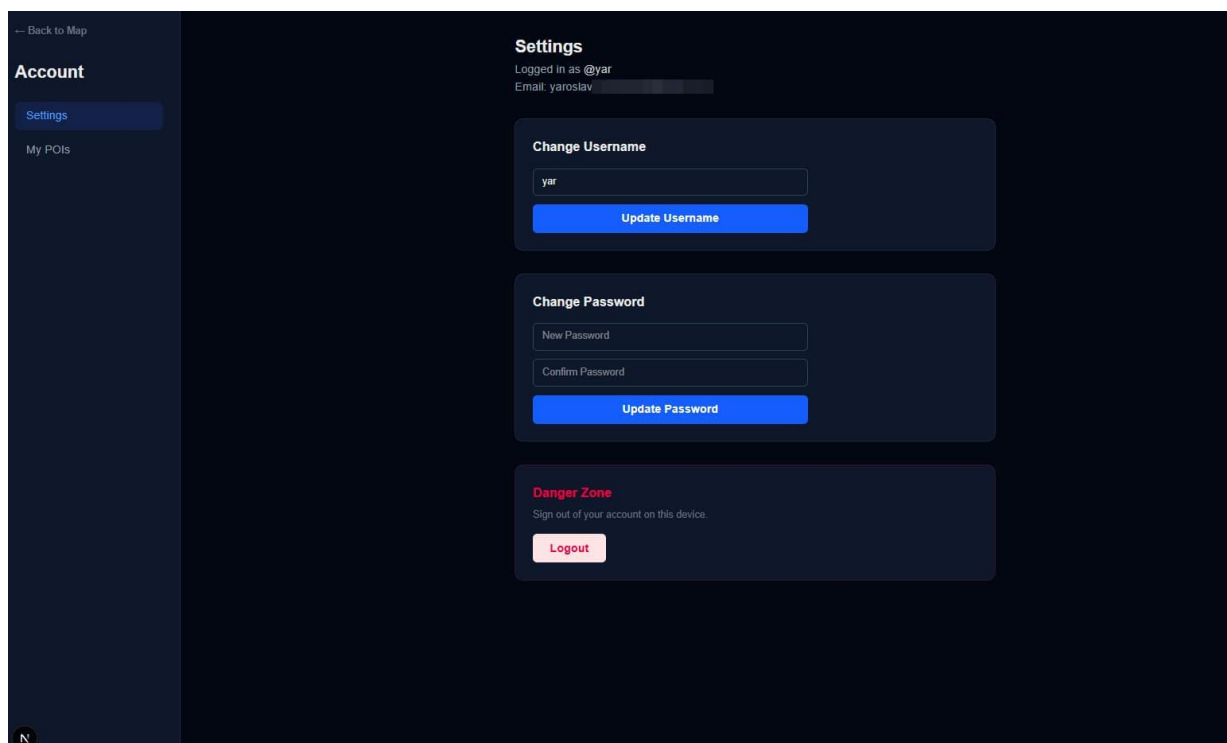


Рисунок 4.7 – Сторінка керування обліковим записом та безпекою

4.4 Висновки за розділом 4

Геосоціальний вебзастосунок успішно реалізовано із забезпеченням усіх функціональних можливостей, що були визначені на етапі проєктування. Проведено експлуатаційне тестування розробленої системи, яке підтвердило коректність динамічного відображення просторових даних, стабільність роботи алгоритмів соціальної взаємодії та безперебійне функціонування картографічного інтерфейсу з відповідними навігаційними панелями для зручного керування послідовністю дій користувача.

ВИСНОВКИ

На основі отриманих результатів у роботі успішно досягнуто поставлену мету. Розроблено геосоціальний вебзастосунок для моніторингу локальних подій, комерційних пропозицій та заходів у режимі реального часу. Це забезпечило підвищення ефективності інформування населення та створило зручну екосистему для взаємодії місцевих громад і підтримки локальної економіки.

Виконаний аналіз інформаційного середовища підтвердив недостатню ефективність розрізнених інструментів, таких як статичні картографічні сервіси, паперові оголошення чи алгоритмічні стрічки новин без точної просторової прив'язки. На ліквідацію цих недоліків спрямовано дану розробку, яка органічно поєднала можливості інтерактивної просторової навігації зі зручністю соціальної взаємодії. Для вирішення проблеми пошуку короточасних активностей визначено ключові сценарії поведінки користувача та спроектовано відповідну архітектуру платформи.

Для реалізації визначених завдань обрано та застосовано набір сучасних технологій, до яких увійшли мова програмування TypeScript, бібліотека React, вебфреймворк Next.js та система управління реляційними базами даних PostgreSQL із розширенням PostGIS. Загальним підсумком стало створення застосунку, здатного ефективно обробляти часозалежну геопросторову інформацію. Розробка успішно розв'язала проблему фрагментованості даних, об'єднавши створення тимчасових маркерів, інтерактивну фільтрацію просторових об'єктів та обмін повідомленнями в одному інструменті.

У підсумку впроваджено програмне рішення, яке надало жителям міст сучасну альтернативу традиційним засобам локального інформування. Програма забезпечила повноцінні інструменти для миттєвого дослідження власного оточення, швидкої публікації подій, обміну коментарями та формування рейтингів локальних ініціатив. Такий підхід не лише спрощує

пошук актуальних заходів, але й стимулює розвиток місцевого малого бізнесу через швидкий доступ до цільової аудиторії. Завдяки цьому суттєво вдосконалено механізми соціально-географічної орієнтації користувачів у межах обраної місцевості та створено надійне цифрове підґрунтя для подальшої інтеграції місцевих спільнот.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Болстад П. GIS Fundamentals: A First Text on Geographic Information Systems. Нью-Йорк : XanEdu Publishing, 2022. 770 p.
2. Піндл Ф. Getting to Know Web GIS. Редлендс : Esri Press, 2022. 480 p.
3. Google Maps. *Google*. URL: <https://www.google.com/maps> (date of access: 08.05.2026).
4. Foursquare City Guide. *Foursquare*. URL: <https://foursquare.com> (date of access: 08.05.2026).
5. Facebook Events. *Meta*. URL: <https://www.facebook.com/events/> (date of access: 08.05.2026).
6. Meetup. *Meetup*. URL: <https://www.meetup.com> (date of access: 08.05.2026).
7. Eventbrite. *Eventbrite*. URL: <https://www.eventbrite.com> (date of access: 08.05.2026).
8. JavaScript. *MDN Web Docs*. URL: <https://developer.mozilla.org/en-US/docs/Web/JavaScript> (date of access: 08.05.2026).
9. Голдберг Дж. Learning TypeScript: Enhance Your Web Development Skills Using Type-Safe JavaScript. Нью-Йорк : O'Reilly Media, 2022. 270 p.
10. Субраманіан В. Pro MERN Stack: Full Stack Web App Development with Mongo, Express, React, and Node. Нью-Йорк : Apress, 2023. 600 p.
11. React Documentation. *React*. URL: <https://react.dev> (date of access: 08.05.2026).
12. Node.js Documentation. *Node.js*. URL: <https://nodejs.org/en/docs/> (date of access: 08.05.2026).
13. Express.js Documentation. *Express.js*. URL: <https://expressjs.com/en/4x/api.html> (date of access: 08.05.2026).
14. MongoDB Documentation. *MongoDB*. URL: <https://www.mongodb.com/docs/> (date of access: 08.05.2026).

15. Piva M. Real-World Next.js: Build scalable, high-performance, and modern web applications using Next.js. Бірмінгем : Packt Publishing, 2022. 402 p.
16. Supabase Official Documentation. *Supabase*. URL: <https://supabase.com/docs> (date of access: 08.05.2026).
17. PostgreSQL 16 Documentation. *PostgreSQL*. URL: <https://www.postgresql.org/docs/16/> (date of access: 08.05.2026).
18. Маміно Л., Касіапо М. Node.js Design Patterns: Design and implement production-grade Node.js applications using proven patterns and techniques. Бірмінгем : Packt Publishing, 2020. 664 p.
19. Коронел К., Морріс С. Database Systems: Design, Implementation, & Management. Бостон : Cengage Learning, 2023. 848 p.
20. Об Р., Сюй Л. PostGIS in Action. Нью-Йорк : Manning Publications, 2021. 600 p.
21. Leaflet Documentation. *Leaflet*. URL: <https://leafletjs.com/reference.html> (date of access: 08.05.2026).

ДОДАТОК А
Технічне завдання

Вступ

Сучасне міське середовище та локальні спільноти відзначаються високим рівнем соціальної активності, яка постійно генерує нові інформаційні приводи. Це стосується організації різноманітних тимчасових подій, благодійних ярмарків, локальних розпродажів, вуличних виступів та громадських зібрань. На прикладі того, що розвитку міських просторів та підтримці місцевих ініціатив приділяється багато уваги, можна говорити про значний потенціал взаємодії людей на локальному рівні. Проте інформаційна підтримка таких ініціатив здійснюється через інструменти, які не завжди відповідають специфіці гіперлокальних подій: через глобальні соціальні мережі, через створення окремих груп у месенджерах або через традиційні статичні картографічні сервіси.

При цьому не приділяється достатня увага реалізації можливості зручного та швидкого просторового пошуку тимчасових подій поблизу користувача. Це стосується випадків, коли люди готові долучитися до локальної ініціативи або відвідати захід, але просто не знають про його проведення у сусідньому кварталі. Звичайно, що глобальні платформи дозволяють поширювати інформацію, проте в їхніх стрічках новин повідомлення про невеликі локальні зібрання швидко губляться, а традиційні карти не розраховані на відображення об'єктів, які існують лише кілька годин.

Встановлення зв'язків між жителями одного району або туристами через прив'язку до географічних координат може значно сприяти розвитку місцевих спільнот та підвищенню рівня просторової обізнаності. Люди можуть бути готовими взаємодіяти, допомагати один одному на локальних розпродажах чи обмінюватися враженнями, якщо матимуть для цього зручний інструмент. Загалом же популярність сервісів на основі геолокації сама по собі підкреслює потенційну потребу суспільства у зручному інструменті пошуку ситуативних подій поруч. Саме на забезпеченні можливості миттєвого

геосоціального обміну інформацією про тимчасові точки інтересу має бути зроблено акцент у цій розробці.

A.1 Підстави для розробки

Дипломна кваліфікаційна робота виконувалась у відповідності та на підставі Наказу № 139 від 07 квітня 2026 р. за Національним університетом «Запорізька політехніка», яким визначено тему роботи – «Розроблення програмного забезпечення для моніторингу комерційних пропозицій на основі геолокаційних даних».

A.2 Призначення розробки

Програма призначена для оперативного пошуку та створення тимчасових точок інтересу на інтерактивній картографічній підоснові. Функціонал розробки забезпечує можливість візуалізації локальних подій у реальному часі, інтерактивну взаємодію між користувачами через систему соціальних реакцій та коментарів, просторову фільтрацію об'єктів за географічними координатами та категоріями, а також надання інструментарію для персонального керування власними публікаціями та налаштуваннями профілю учасника.

A.3 Основні вимоги до програми

A.3.1 Вимоги до функціональних характеристик

Функціональні характеристики геосоціального вебзастосунку повинні включати:

- реєстрація нових учасників як користувачів у системі, безпечна авторизація та завершення сеансу;

- редагування облікових даних профілю, включаючи зміну публічного імені, адреси електронної пошти та пароля доступу;
- відображення інтерактивного картографічного полотна з підтримкою вільного масштабування та просторової навігації;
- динамічне завантаження просторових об'єктів та тимчасових подій виключно в межах поточної видимої області екрана;
- створення нової локальної події із зазначенням географічних координат, прив'язаних до мапи;
- встановлення текстової назви, розширеного опису, тематичної категорії та обов'язкових часових меж існування для кожної нової події; – завантаження, клієнтська оптимізація розміру та прикріплення графічних медіафайлів до публікацій;
- перегляд деталізованої інформації про обрану точку інтересу, включаючи автоматично розрахований статус її активності;
- виконання текстового пошуку локальних подій за збігом у назві або описі;
- просторова фільтрація відображуваних маркерів подій за обраними користувачем тематичними категоріями;
- виконання запитів до зовнішнього сервісу геокодування для пошуку географічних об'єктів та автоматичного центрування мапи на них;
- публікація текстових коментарів із можливістю прикріплення фотографій до існуючих об'єктів на мапі;
- перегляд хронологічної стрічки обговорення та коментарів для кожної локальної події;
- встановлення соціальної реакції до обраної події для формування її рейтингу шляхом позитивної або негативної оцінки;
- зміна типу встановленої соціальної реакції або її повне скасування;
- перегляд переліку власних створених публікацій у відокремленому розділі керування профілем;

- швидкий перехід до перегляду власної події на мапі безпосередньо з панелі керування профілем;
- вилучення власних точок інтересу та раніше залишених коментарів з бази даних;
- здійснення глобальної модерації контенту, що передбачає можливість вилучення будь-яких подій або коментарів користувачами з підтвердженими правами адміністратора системи.

A.3.2 Умови експлуатації

Обов'язковою умовою експлуатації геосоціального вебзастосунку є наявність адміністратора системи або фахівця, здатного виконувати відповідні технічні та управлінські функції. До обов'язків адміністратора належить проведення глобальної модерації створеного користувачами контенту, контроль за дотриманням встановлених правил платформи та безпосереднє технічне розгортання програмного забезпечення на серверних потужностях.

A.3.3 Вимоги до складу та параметрів технічних засобів

Технічне забезпечення серверної частини повинно відповідати наступним мінімальним вимогам:

- багатоядерний центральний процесор із тактовою частотою не менше 2,4 ГГц;
- накопичувач обсягом від 50 Гб, при цьому активне завантаження медіафайлів користувачами може вимагати динамічного розширення дискового простору або впровадження програмних обмежень на максимальний розмір завантажуваних зображень;
- оперативна пам'ять обсягом від 4 Гб для забезпечення стабільної роботи бази даних та серверних процесів.

A.4 Порядок контролю та приймання

Починаючи з дати видачі завдання – 7 квітня, протягом наступних 5 тижнів має бути забезпечено виконання роботи за календарним планом, що містить в завданні на роботу. Календарний план визначає послідовність етапів виконання роботи та за кожним таким етапом – строк виконання і основний очікуваний результат. Після забезпечення цих очікуваних результатів формується пояснювальна записка до дипломної кваліфікаційної роботи. Остаточним кроком приймання роботи є її публічний захист.

ДОДАТОК Б
Текст програми

Б.1 Текст файлу InteractiveMap.tsx

```
'use client'

import { useEffect, useRef } from 'react'
import {
  MapContainer,
  TileLayer,
  Marker,
  useMap
} from 'react-leaflet'
import 'leaflet/dist/leaflet.css'
import L from 'leaflet'

const DEMO_POIS = [
  { lat: 48.8566, lng: 2.3522, cat: 'fair'},
  { lat: 48.8606, lng: 2.3376, cat: 'opening'},
  { lat: 48.8530, lng: 2.3499, cat: 'sale'},
  { lat: 48.8650, lng: 2.3210, cat: 'event'},
  { lat: 48.8490, lng: 2.3600, cat: 'other'},
  { lat: 48.8700, lng: 2.3450, cat: 'sale'},
  { lat: 48.8440, lng: 2.3300, cat: 'fair'},
  { lat: 48.8580, lng: 2.3700, cat: 'event'},
  { lat: 48.8720, lng: 2.3100, cat: 'opening'},
  { lat: 48.8400, lng: 2.3550, cat: 'other'},
  { lat: 48.8550, lng: 2.2950, cat: 'sale'},
  { lat: 48.8680, lng: 2.3650, cat: 'fair'},
  { lat: 48.8460, lng: 2.3150, cat: 'event'},
  { lat: 48.8610, lng: 2.3800, cat: 'opening'},
  { lat: 48.8750, lng: 2.3350, cat: 'sale'},
]

const catColors: Record<string, string> = {
  fair: '#a855f7',
  sale: '#ef4444',
  opening: '#eab308',
  event: '#10b981',
  other: '#6b7280',
}

function getDemoIcon(cat: string, emoji: string) {
  return L.divIcon({
    className: 'custom-div-icon',
    html: `
```

```

function SlowPan() {
  const map = useMap()
  const dirRef = useRef({ lat: 0.0001, lng: 0.00025 })

  useEffect(() => {
    const interval = setInterval(() => {
      const center = map.getCenter()
      map.panTo(
        [center.lat + dirRef.current.lat, center.lng +
dirRef.current.lng],
        { animate: true, duration: 2, easeLinearity: 1 }
      )
    }, 3000)

    return () => clearInterval(interval)
  }, [map])

  return null
}

export default function HeroMap() {
  return (
    <div className="absolute inset-0 z-0">
      <MapContainer
        center={[48.8566, 2.3400]}
        zoom={13}
        scrollWheelZoom={false}
        dragging={false}
        doubleClickZoom={false}
        zoomControl={false}
        attributionControl={false}
        className="h-full w-full"
        style={{ cursor: 'default' }}
      >
        <TileLayer
          url="https://{s}.basemaps.cartocdn.com/rastertiles/voyager/{z}/{
x}/{y}/{r}.png"
          />
        <SlowPan />
        {DEMO_POIS.map((poi, i) => (
          <Marker
            key={i}
            position={[poi.lat, poi.lng]}
            icon={getDemoIcon(poi.cat, poi.emoji)}
            interactive={false}
          />
        ))}
      </MapContainer>
    </div>
  )
}

```

Б.2 Текст файлу interaction.ts

```
'use server'

import { createClient } from '@lib/supabase/server'
import { revalidatePath } from 'next/cache'

export async function addComment(
  poiId: string,
  formData: FormData
) {
  const supabase = await createClient()
  const { data:
    {
      user
    }
  } = await supabase.auth.getUser()

  if (!user) {
    throw new Error('Not authenticated')
  }

  const content = formData.get('content') as string
  if (!content) throw new Error('Comment content is
required')

  const imageFiles = formData.getAll('images') as File[]
  const imageUrls: string[] = []

  for (const file of imageFiles) {
    if (file.size === 0) continue
    const fileName =
`${Date.now()}-${Math.random().toString(36).substring(7)}
-
${file.name.replace(/^[^a-zA-Z0-9.-]/g, '_')}`

    const { data: uploadData, error: uploadError } = await
supabase.storage
      .from('poi-images')
      .upload(fileName, file)

    if (uploadError) {
      console.error('Upload error:', uploadError)
    } else if (uploadData) {
      const { data: publicUrlData } =
supabase.storage.from('poi-images').getPublicUrl(fileName)
      imageUrls.push(publicUrlData.publicUrl)
    }
  }

  const { error } = await supabase.from('comments').insert({
```

```

        poi_id: poiId,
        user_id: user.id,
        content,
        image_urls: imageUrls
    })

    if (error) {
        console.error('Failed to add comment', error)
        throw new Error('Failed to add comment')
    }

    revalidatePath(`/poi/${poiId}`)
}

export async function toggleReaction(poiId: string, type:
'upvote' | 'downvote') {
    const supabase = await createClient()
    const { data: { user } } = await supabase.auth.getUser()

    if (!user) {
        throw new Error('Not authenticated')
    }

    const { error } = await supabase.rpc('toggle_reaction', {
        p_poi_id: poiId,
        p_type: type
    })

    if (error) {
        console.error('Failed to toggle reaction', error)

        throw new Error('Failed to toggle reaction')
    }

    revalidatePath(`/poi/${poiId}`)
    revalidatePath('/map')
}

export async function getPoiDetails(poiId: string) {
    const supabase = await createClient()
    const { data, error } = await supabase
        .from('pois')
        .select('*', users!pois_creator_id_fkey(username))
        .eq('id', poiId)
        .single()

    if (error) {
        console.error('getPoiDetails error:', error)
        throw new Error(`Failed to load POI details:
${error.message}`)
    }
    return data
}

```

Б.3 Текст файла PoiDetailPanel.tsx

```
'use client'

import { useState, useEffect } from 'react'
import { getPoiDetails, getComments, toggleReaction,
addComment, deleteComment } from '@app/actions/interaction'
import { deletePoi } from '@app/actions/poi'
import ImageCarousel from '@components/ui/ImageCarousel'
import imageCompression from 'browser-image-compression'
import toast from 'react-hot-toast'
import Link from 'next/link'
import { X, MessageSquare, ThumbsUp, ThumbsDown, User,
Clock, MapPin, Plus, Trash2 } from 'lucide-react'

interface PoiDetailPanelProps {
  poiId: string
  userId?: string
  isAdmin?: boolean
  onClose: () => void
  onPoiLoaded?: (lat: number, lng: number) => void
  onPoiDeleted?: () => void
}

export default function PoiDetailPanel({ poiId, userId,
isAdmin,      onClose,      onPoiLoaded,      onPoiDeleted      }:
PoiDetailPanelProps) {
  const [poi, setPoi] = useState<any>(null)
  const [comments, setComments] = useState<any[]>([])
  const [loading, setLoading] = useState(true)
  const [commentText, setCommentText] = useState('')
  const [photos, setPhotos] = useState<File[]>([])
  const [submitting, setSubmitting] = useState(false)

  const loadData = async () => {
    try {
      const [poiData, commentsData] = await Promise.all([
        getPoiDetails(poiId),
        getComments(poiId)
      ])
      setPoi(poiData)
      setComments(commentsData)

      if (onPoiLoaded && poiData.location) {
        try {
          const loc = typeof poiData.location === 'string' ?
JSON.parse(poiData.location) : poiData.location
          if (loc && loc.coordinates) {
            onPoiLoaded(loc.coordinates[1],
loc.coordinates[0])
          }
        }
      }
    }
  }
}
```

```

        } catch (e) { }
      }
    } catch (e) {
      console.error(e)
      toast.error('Failed to load details')
    } finally {
      setLoading(false)
    }
  }
}

useEffect(() => { loadData() }, [poiId])

const handleReaction = async (type: 'upvote' | 'downvote')
=> {
  if (!userId) return toast.error('Sign in to vote')
  try {
    await toggleReaction(poiId, type)
    loadData()
  } catch { toast.error('Failed to vote') }
}

const handleDeleteComment = async (commentId: string) => {
  setComments(prev => prev.filter(c => c.id !== commentId))
  try {
    await deleteComment(commentId)
  } catch {
    toast.error('Failed to delete comment')
    const refreshed = await getComments(poiId)
    setComments(refreshed || [])
  }
}

const handleDeletePoi = async () => {
  if (!window.confirm('Delete this POI? This cannot be
undone.')) return
  try {
    await deletePoi(poiId)
    toast.success('POI deleted')
    onPoiDeleted?.()
  } catch {
    toast.error('Failed to delete POI')
  }
}

const handleComment = async (e: React.FormEvent) => {
  e.preventDefault()
  if (!commentText.trim()) return
  setSubmitting(true)
  try {
    const formData = new FormData()
    formData.set('content', commentText)
  }
}

```

```

        for (const file of photos) {
            try {
                const options = { maxSizeMB: 0.8, maxWidthOrHeight:
1024, useWebWorker: true }
                const compressed = await imageCompression(file,
options)
                const newFile = new File(
[compressed], file.name,
{ type: file.type })
                formData.append('images', newFile)
            } catch (err) {
                console.error('Compression failed, using
original', err)
                formData.append('images', file)
            }
        }

        await addComment(poiId, formData)
        setCommentText('')
        setPhotos([])
        toast.success('Comment posted!')
        loadData()
    } catch (err) {
        console.error('Comment error:', err)
        toast.error('Failed to post comment')
    } finally { setSubmitting(false) }
}

const handlePhotoAdd = (e:
React.ChangeEvent<HTMLInputElement>) => {
    if (e.target.files) {
        const newFiles = Array.from(e.target.files)
        setPhotos(prev => [...prev, ...newFiles])
    }
    e.target.value = ''
}

const removePhoto = (idx: number) => {
    setPhotos(prev => prev.filter((_, i) => i !== idx))
}

return (
    <div className="fixed inset-0 z-[9999] flex items-center
justify-center pointer-events-auto p-4 sm:p-6 lg:p-8"
onClick={onClose}>
        <div className="absolute inset-0 bg-slate-950/50
backdrop-blur-md" />

        <div
            className="relative z-10 w-full max-w-5xl h-[90vh]
overflow-hidden bg-white dark:bg-slate-950 rounded-3xl shadow-2xl
flex flex-col border border-white/10"
            onClick={e => e.stopPropagation()}

```

```

>
  { /* Header */ }
  <div className="flex items-center justify-between px-
6 py-4 border-b border-gray-100 dark:border-slate-800 bg-white/90
dark:bg-slate-950/90 backdrop-blur-sm shrink-0">
    <div className="flex items-center gap-3">
        <div className="p-1.5 bg-blue-100 dark:bg-blue-
900/30 text-blue-600 dark:text-blue-400 rounded-lg">
            <MapPin size={18} />
        </div>
        <div>
            <h2 className="font-bold text-lg tracking-
tight leading-none">{poi?.title || 'Loading...'}</h2>
            {poi && <span className="text-[9px] font-bold
uppercase tracking-widest text-gray-400">{poi.category}</span>}
        </div>
    </div>
    <div className="flex items-center gap-1">
        {poi && (isAdmin || userId === poi.creator_id) &&
(
            <button
                onClick={handleDeletePoi}
                className="p-2                                hover:bg-red-50
dark: hover:bg-red-950/30 rounded-full transition-all text-gray-
400 hover:text-red-500"
                title="Delete POI"
            >
                <Trash2 size={18} />
            </button>
        )}
        <button
            onClick={onClose}
            className="p-2                                hover:bg-gray-100
dark: hover:bg-slate-800 rounded-full transition-all text-gray-400
hover:text-gray-900 dark: hover:text-white"
        >
            <X size={20} />
        </button>
    </div>
</div>

{loading ? (
    <div className="flex-1 flex flex-col items-center
justify-center gap-4 py-20">
        <div className="animate-spin rounded-full h-10 w-
10 border-b-2 border-blue-600" />
        <p className="text-gray-500 animate-pulse text-
xs font-medium">Fetching details...</p>
    </div>
) : poi ? (
    <div className="flex-1 flex flex-col lg:flex-row
overflow-hidden">

```

```

        { /* Left Column: Media & Core Info */ }
        <div className="lg:w-[60%] border-r border-gray-
100 dark:border-slate-800 flex flex-col overflow-y-auto custom-
scrollbar">
            <div className="p-6 sm:p-8">
                <ImageCarousel images={poi.image_urls} />

                <div className="mt-6">
                    <div className="flex flex-wrap items-
center gap-3 mb-4">
                        <span className="px-3 py-1 rounded-full
bg-blue-600 text-white text-[10px] font-bold shadow-lg shadow-
blue-500/10">
                            {poi.category}
                        </span>
                        <div className="flex items-center gap-
1.5 text-[11px] text-gray-500 dark:text-gray-400 font-medium">
                            <User size={12} />
                            {poi.users?.username || 'GeoExplorer'}
                        </div>
                        <div className="flex items-center gap-
1.5 text-[11px] text-gray-500 dark:text-gray-400 font-medium">
                            <Clock size={12} />
                            {new
Date(poi.created_at).toLocaleDateString()}
                        </div>
                    </div>

                    <h1 className="text-3xl font-extrabold mb-
4 tracking-tight leading-tight">{poi.title}</h1>
                    <p className="text-base text-gray-600
dark:text-gray-300 leading-relaxed mb-6 whitespace-pre-wrap font-
light">
                        {poi.description}
                    </p>

                    { /* Timeline Date Section */ }
                    <div className="mt-2">
                        { (() => {
                            const now = new Date()
                            const start = poi.start_date ? new
Date(poi.start_date) : null
                            const end = poi.end_date ? new
Date(poi.end_date) : null

                            const hasStarted = !start || start <=
now
                            const msLeft = end ? end.getTime() -
now.getTime() : null
                            const isOver = msLeft !== null &&
msLeft <= 0

```

```

        let statusColor = 'bg-emerald-500'
        let statusLabel = 'Live'
        let statusPulse = true
        if (!hasStarted) { statusColor = 'bg-
blue-500'; statusLabel = 'Upcoming'; statusPulse = false }
        if (isOver) { statusColor = 'bg-gray-
400'; statusLabel = 'Ended'; statusPulse = false }

        return (

            { /* Right Column: Reactions & Comments */}
            <div className="lg:w-[40%] flex flex-col bg-gray-
50/30 dark:bg-slate-900/10">

                { /* Feedback Area */}
                <div className="p-6 border-b border-gray-100
dark:border-slate-800 shrink-0">
                    <h3      className="text-[10px]      font-bold
uppercase tracking-widest text-gray-400 mb-4 flex items-center
gap-2">
                        Feedback
                    </h3>
                    <div className="flex gap-3">
                        <button      onClick={ ()          =>
handleReaction('upvote')} disabled={!userId} className="flex-1
flex items-center justify-center gap-2 py-2 px-3 rounded-xl bg-
white dark:bg-slate-900 border border-gray-100 dark:border-slate-
800      hover:border-emerald-500      hover:text-emerald-600
dark:hover:text-emerald-400      transition-all      shadow-sm
disabled:opacity-50">
                            <ThumbsUp size={16} />
                            <span      className="text-sm      font-
bold">{poi.upvotes}</span>
                        </button>
                        <button      onClick={ ()          =>
handleReaction('downvote')} disabled={!userId} className="flex-1
flex items-center justify-center gap-2 py-2 px-3 rounded-xl bg-
white dark:bg-slate-900 border border-gray-100 dark:border-slate-
800      hover:border-rose-500      hover:text-rose-600      dark:hover:text-
rose-400      transition-all      shadow-sm      disabled:opacity-50">
                            <ThumbsDown size={16} />
                            <span      className="text-sm      font-
bold">{poi.downvotes}</span>
                        </button>
                    </div>
                </div>

                { /* Comments Section */}
                <div className="flex-1 flex flex-col overflow-
hidden p-6">
                    <h3      className="text-[10px]      font-bold
uppercase tracking-widest text-gray-400 mb-4 flex items-center
gap-2 shrink-0">

```

```

        <MessageSquare size={14} />
        Comments ({comments.length})
    </h3>

    <div className="flex-1 overflow-y-auto
space-y-4 pr-2 custom-scrollbar pb-4">
        {comments.map(comment => (
            <div key={comment.id} className="flex
gap-3 bg-white dark:bg-slate-900 p-3 rounded-xl shadow-sm border
border-gray-100 dark:border-slate-800">
                <div className="w-8 h-8 rounded-full
bg-gradient-to-br from-blue-500 to-indigo-600 flex items-center
justify-center font-bold text-white uppercase shrink-0 text-xs
shadow-md">
                    {comment.users?.username?.[0] ||
'U'}
                </div>
                <div className="flex-1 min-w-0">
                    <div className="flex items-center
justify-between gap-2 mb-0.5">
                        <span className="font-bold text-xs
text-gray-900 dark:text-white truncate">{comment.users?.username
|| 'Unknown'}</span>
                        <div className="flex items-center
gap-1 shrink-0">
                            <span
                                className="text-[9px]
                                whitespace-nowrap">{new
Date(comment.created_at).toLocaleDateString()}</span>
                            {(isAdmin || userId ===
comment.user_id) && (
                                <button
                                    onClick={ () =>
handleDeleteComment(comment.id)}
                                    className="p-0.5 text-gray-
300 hover:text-red-500 dark:text-gray-600 dark:hover:text-red-400
transition-colors"
                                    title="Delete comment"
                                >
                                    <Trash2 className="w-3 h-3"
                                />
                                </button>
                            )}
                        </div>
                    </div>
                    <p className="text-[13px] text-gray-
600 dark:text-gray-300 leading-tight mb-2">{comment.content}</p>
                    {comment.image_urls
                        &&
comment.image_urls.length > 0 && (
                        <div className="flex gap-1.5
overflow-x-auto pb-1 custom-scrollbar">
                            {comment.image_urls.map((url:
string, i: number) => (

```



```

        multiple
        onChange={handlePhotoAdd}
        className="hidden"
      />
      <Plus size={18} />
    </label>
    {photos.length > 0 && <span
className="text-[9px] font-bold text-blue-500 bg-blue-50 dark:bg-
blue-900/30 px-1.5 py-0.5 rounded-md">+{photos.length}</span>}
    </div>
    <button
                                type="submit"
disabled={submitting || !commentText.trim()} className="bg-blue-
600 hover:bg-blue-700 text-white font-bold py-2 px-5 rounded-xl
text-xs      transition-all      shadow-md      shadow-blue-500/10
disabled:opacity-50">
      {submitting ? 'Post...' : 'Post'}
    </button>
  </div>

  {/* Photo Previews */}
  {photos.length > 0 && (
    <div className="flex gap-2 overflow-
x-auto pb-2 pt-1 custom-scrollbar overflow-visible">
      {photos.map((file, idx) => (
        <div
                                key={idx}
className="relative shrink-0 w-14 h-14">
          <img

src={URL.createObjectURL(file)}
                                alt=""
                                className="w-full      h-full
object-cover rounded-xl border border-gray-200 dark:border-slate-
800"
          />
          <button
            type="button"
            onClick={ (e) => {
e.preventDefault(); e.stopPropagation(); removePhoto(idx); }}
            className="absolute -top-1.5
-right-1.5 bg-red-500 hover:bg-red-600 text-white rounded-full w-
5 h-5 flex items-center justify-center shadow-lg border border-
white dark:border-slate-800 z-10 transition-colors pointer-
events-auto"
          >
            <X size={12} strokeWidth={3}
          />
        </button>
      </div>
    ) ) }
  </div>
) }
</div>
</form>

```

```

        ) : (
            <div className="mt-4 p-4 bg-gray-100
dark:bg-slate-800 rounded-xl text-center shrink-0">
                <Link href="/login" className="text-
[11px] text-blue-600 hover:underline font-bold">Sign in to
Comment</Link>
            </div>
        )}
    </div>
</div>
</div>
) : (
    <div className="flex-1 flex flex-col items-center
justify-center gap-4 py-20 text-center">
        <MapPin size={40} className="text-gray-300" />
        <p className="text-sm text-gray-500">POI not
found.</p>
        <button onClick={onClose} className="text-xs
text-blue-500 font-bold hover:underline">Back to map</button>
    </div>
    )}
</div>
</div>
)
}

```

Б.4 Текст файла MapFilterBar.tsx

```

'use client'

import { useState, useEffect, useRef } from 'react'
import { useRouter, useSearchParams } from 'next/navigation'
import { Poi } from '@app/actions/poi'
import { Search, X, MapPin } from 'lucide-react'

interface NominatimResult {
  place_id: number
  display_name: string
  lat: string
  lon: string
}

const clearAllCats = () => {
  const params = new
URLSearchParams(searchParamsRef.current.toString())
  params.delete('cat')
  router.replace(`/map?${params.toString()}`, { scroll:
false })
}

const handlePoiSelect = (poi: Poi) => {

```

```

    try {
      const loc = typeof poi.location === 'string' ?
JSON.parse(poi.location) : poi.location
      if (loc?.coordinates) {
        const [lng, lat] = loc.coordinates as [number,
number]
        onLocationSelect(lat, lng)
      }
    } catch { /* ignore parse errors */ }
    const params = new
URLSearchParams(searchParamsRef.current.toString())
    params.set('poi', poi.id)
    router.push(`/map?${params.toString()}`, { scroll:
false })
    setShowDropdown(false)
  }

  const handleLocationSelect = (result: NominatimResult) =>
{
  onLocationSelect(parseFloat(result.lat),
parseFloat(result.lon))
  setShowDropdown(false)
}

  const poiSuggestions = inputValue.length >= 2
    ? pois.filter(p =>
p.title.toLowerCase().includes(inputValue.toLowerCase()))
    .slice(0, 4)
    : []

  const hasDropdownContent = poiSuggestions.length > 0 ||
locationResults.length > 0 || isGeoLoading
  const isFiltered = filteredCount !== totalCount

  return (
    <div ref={containerRef} className="flex flex-col">
      <div className="bg-white dark:bg-slate-800 rounded-xl
shadow-lg border border-gray-200 dark:border-slate-700">
        /* Search input row */
        <div className="flex items-center px-3 pt-2.5 pb-1.5
gap-2">
          <Search className="w-4 h-4 text-gray-400 shrink-
0" />
          <input
            type="text"
            value={inputValue}
            onChange={e =>
handleInputChange(e.target.value)}
            onFocus={() => inputValue.length >= 2 &&
setShowDropdown(true)}
            onKeyDown={e => e.key === 'Escape' &&
setShowDropdown(false)}
            placeholder="Search events, places..."

```

```

        className="flex-1      bg-transparent      text-sm
outline-none placeholder-gray-400 dark:placeholder-gray-500 text-
gray-800 dark:text-gray-200 min-w-0"
      />
      {isGeoLoading && (
        <svg      className="w-3.5      h-3.5      text-gray-400
shrink-0      animate-spin"      xmlns="http://www.w3.org/2000/svg"
fill="none" viewBox="0 0 24 24">
          <circle      className="opacity-25"      cx="12"
cy="12" r="10" stroke="currentColor" strokeWidth="4" />
          <path      className="opacity-75"
fill="currentColor" d="M4 12a8 8 0 018-8V0C5.373 0 0 5.373 0 12h4z"
/>
        </svg>
      )}
      {inputValue && (
        <button onClick={handleClear} className="text-
gray-400 hover:text-gray-600 dark:hover:text-gray-200 shrink-0">
          <X className="w-4 h-4" />
        </button>
      )}
    </div>

    {/* Category chips row */}
    <div className="flex items-center gap-1.5 px-3 pb-
2.5 flex-wrap">
      <button
        onClick={clearAllCats}
        className={`text-xs font-semibold px-2.5 py-0.5
rounded-full border transition-colors ${
          activeCats.length === 0
            ? 'bg-blue-600 text-white border-blue-600'
            : INACTIVE_CHIP
        }`}
      >
        All
      </button>
      {CATEGORIES.map(cat => (
        <button
          key={cat.value}
          onClick={() => toggleCategory(cat.value)}
          className={`text-xs font-semibold px-2.5 py-
0.5 rounded-full border transition-colors ${
            activeCats.includes(cat.value)
              ?
cat.activeCls : INACTIVE_CHIP
          }`}
        >
          {cat.emoji} {cat.label}
        </button>
      ))}
      {isFiltered && (
        <span className="ml-auto text-[10px] text-gray-
400 dark:text-gray-500 whitespace-nowrap font-medium">

```

```

        {filteredCount}/{totalCount} shown
      </span>
    )}
  </div>
</div>

  {/* Dropdown suggestions */}
  {showDropdown && hasDropdownContent && (
    <div className="mt-1 bg-white dark:bg-slate-800
rounded-xl shadow-xl border border-gray-200 dark:border-slate-700
max-h-64 overflow-y-auto">
      {poiSuggestions.length > 0 && (
        <div>
          <div className="px-3 py-1.5 text-[10px]
uppercase font-bold text-gray-400 dark:text-gray-500 tracking-
widest border-b border-gray-100 dark:border-slate-700">
            POIs
          </div>
          {poiSuggestions.map(poi => (
            <button
              key={poi.id}
              onClick={() => handlePoiSelect(poi)}
              className="w-full text-left px-3 py-2
hover:bg-gray-50 dark:hover:bg-slate-700 flex items-center gap-2
text-sm text-gray-800 dark:text-gray-200 transition-colors"
            >
              <span className="shrink-0 text-base">
                {CATEGORIES.find(c => c.value ===
poi.category)?.emoji ?? '📍'}
              </span>
              <span
className="truncate">{poi.title}</span>
            </button>
          )})}
        </div>
      )}

      {(locationResults.length > 0 || isGeoLoading) &&
(
        <div>
          <div className={`px-3 py-1.5 text-[10px]
uppercase font-bold text-gray-400 dark:text-gray-500 tracking-
widest border-b border-gray-100 dark:border-slate-700
${poiSuggestions.length > 0 ? 'border-t' : ''}`}>
            Locations
          </div>
          {isGeoLoading && locationResults.length === 0
? (
            <div className="px-3 py-2.5 text-sm text-
gray-400 flex items-center gap-2">
              <svg className="w-3.5 h-3.5 animate-spin"
xmlns="http://www.w3.org/2000/svg" fill="none" viewBox="0 0 24
24">

```

```

        <circle className="opacity-25" cx="12"
cy="12" r="10" stroke="currentColor" strokeWidth="4" />
        <path                      className="opacity-75"
fill="currentColor" d="M4 12a8 8 0 018-8V0C5.373 0 0 5.373 0 12h4z"
/>
    </svg>
    Searching...
</div>
) : (
  locationResults.map(result => (
    <button
      key={result.place_id}
      onClick={() =>
handleLocationSelect(result)}
      className="w-full text-left px-3 py-2
hover:bg-gray-50 dark:hover:bg-slate-700 flex items-center gap-2
text-sm text-gray-800 dark:text-gray-200 transition-colors"
    >
    <MapPin className="w-3.5 h-3.5 text-
gray-400 shrink-0" />
    <span
className="truncate">{result.display_name}</span>
    </button>
  ))
  )}
</div>
)}
</div>
)}
</div>
)
}

```

Б.5 Текст файлу layout.tsx

```

import type { Metadata } from "next";
import { Geist, Geist_Mono } from "next/font/google";
import "../globals.css";
import { Toaster } from 'react-hot-toast';

export const metadata: Metadata = {
  title: "Geo-Social",
  description: "Explore local events and commercial offers
in real-time.",
};

export default function RootLayout({
  children,
}: Readonly<{
  children: React.ReactNode;
}>) {

```

```

    return (
      <html
        lang="uk"
        className={` ${geistSans.variable}
  ${geistMono.variable} h-full antialiased`}
        suppressHydrationWarning
      >
        <body      className="min-h-full      flex      flex-col"
suppressHydrationWarning>
          <Toaster position="top-center" />
          {children}
        </body>
      </html>
    );
  }

```

ДОДАТОК В
Слайди презентації

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЗАПОРІЗЬКА ПОЛІТЕХНІКА»
КАФЕДРА ПРОГРАМНИХ ЗАСОБІВ

ТЕМА ДИПЛОМНОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ БАКАЛАВРА:
**Розроблення програмного забезпечення для моніторингу комерційних
пропозицій на основі геолокаційних даних.**
Development of Software for Location-Based Monitoring of Commercial Offers

Виконавець: студент КНТ-113сп Ярослав ПАНЧЕНКО
Керівник: к.т.н., доцент Степан СКРУПСЬКИЙ

2026

Рисунок В.1 – Слайд 1

Об'єкт, мета та основні завдання дослідження

Об'єкт роботи – процес програмної інженерії при розробленні, впровадженні та супроводі геосоціальних інформаційних систем для моніторингу локальних подій.

Мета роботи – підвищення ефективності інформування населення та взаємодії місцевих громад шляхом розроблення геосоціального вебзастосунку для моніторингу локальних подій, комерційних пропозицій та заходів у режимі реального часу.

Завдання дослідження:

1. Проаналізувати недоліки існуючих картографічних та соціальних платформ.
2. Спроектувати архітектуру гібридної системи з урахуванням обробки просторових даних.
3. Розробити алгоритми динамічного завантаження об'єктів та транзакційної обробки соціальних реакцій.
4. Реалізувати програмний продукт з використанням технологій Next.js та PostGIS.

Рисунок В.2 – Слайд 2

Порівняння існуючих рішень

Критерії оцінювання	Google Maps	Meetup	Eventbrite	Проектована розробка
Основний тип об'єктів	Статичні установи та інфраструкту-ра	Заплановані зустрічі груп за інтересами	Формальні заходи та продаж квитків	Тимчасові події та локальні точки інтересу
Роль картографічно-го інтерфейсу	Головний інтерфейс відображення	Відсутній, доступна лише текстова адреса	Допоміжний інструмент пошуку	Головний інтерфейс з кастомними маркерами
Контроль часового циклу події	Статичні години роботи закладів	Фіксована дата та час проведення заходу	Фіксована дата та час проведення заходу	Динамічні статуси з живим зворотним відліком
Створення міток користувачами	Тривала модерація або реєстрація бізнесу	Створення зустрічей адміністраторами груп	Багатоступе-неве налаштування квитків	Миттєве розміщення в один клік по мапі
Соціальна взаємодія	Текстові відгуки та система зіркового рейтингу	Коментарі та підтвердження участі	Відсутня, акцент на транзакціях	Швидкі коментарі та система реакцій
Просторовий пошук за видимою областю	Присутній	Відсутній, пошук за назвою міста	Частково присутній	Присутній, оптимізований просторовими запитами

3

Рисунок В.3 – Слайд 3

Обґрунтування технологічного стека

Критерій порівняння	Класичний стек: React, Node.js, Express, MongoDB	Гібридний стек: Next.js, Supabase, PostgreSQL
Архітектурний підхід	Повністю роздільна архітектура, де клієнт спілкується із сервером виключно через ручне створення REST API	Об'єднана архітектура, де взаємодія з базою даних відбувається через безпечні серверні дії без окремих API-маршрутів
Робота з просторовими даними	Базові географічні запити у NoSQL базі даних, складність масштабування складних просторових розрахунків	Повноцінна інтеграція з реляційною базою даних та галузевим просторовим розширенням PostGIS безпосередньо з коробки
Рендеринг інтерфейсу	Виключно клієнтський рендеринг, що може викликати затримки при початковому завантаженні важких скриптів мапи	Гібридний рендеринг із серверною генерацією сторінок, що забезпечує швидке відображення каркаса застосунку
Інфраструктура та авторизація	Необхідність самостійного проектування системи безпеки, налаштування серверів та зберігання медіафайлів	Вбудована система авторизації, управління політиками доступу на рівні рядків та готове хмарне сховище зображень

4

Рисунок В.4 – Слайд 4

Архітектура ролей та сценарії використання

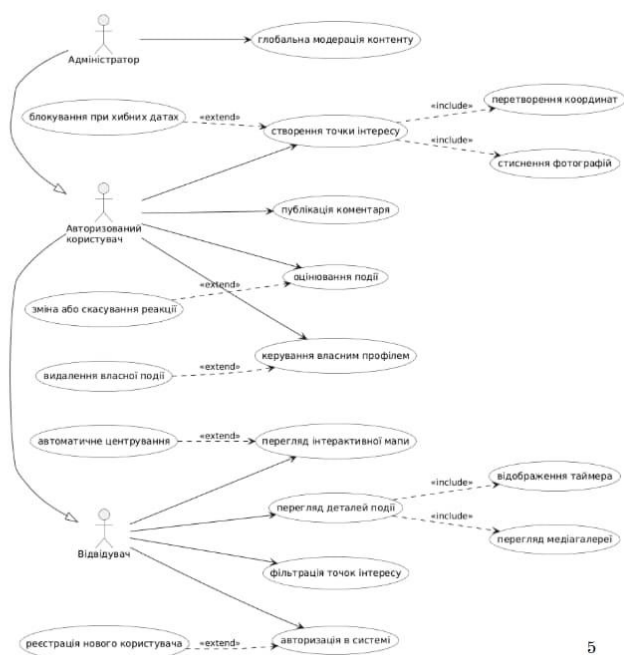


Рисунок В.5 – Слайд 5

Компонентна архітектура системи

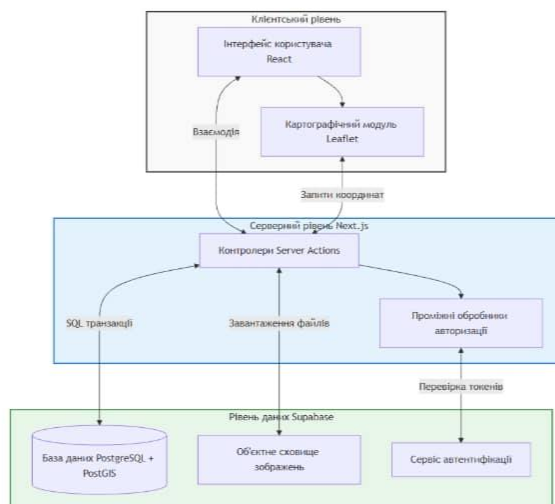
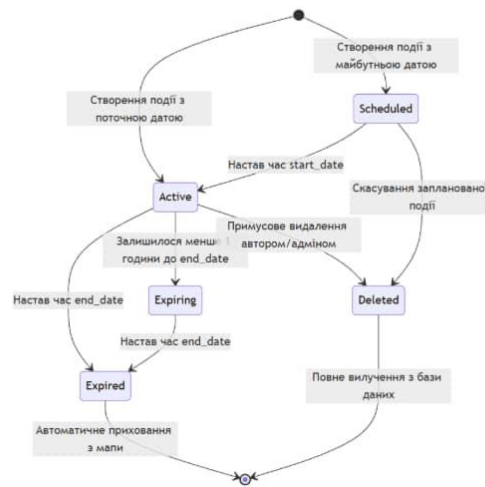


Рисунок В.6 – Слайд 6

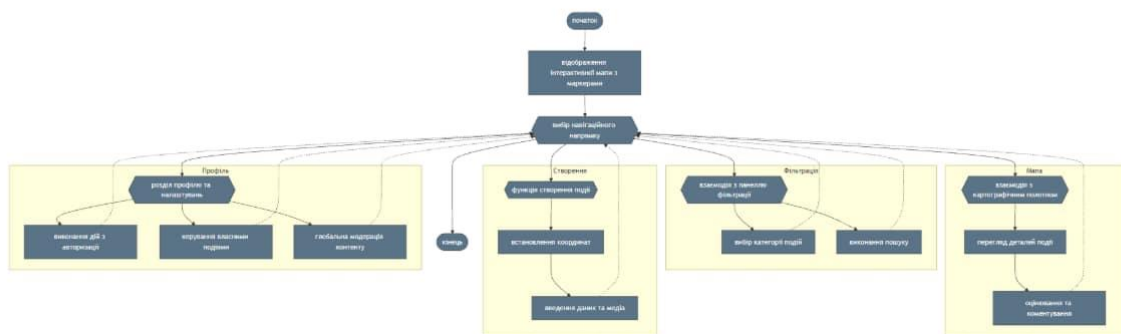
Життєвий цикл тимчасових подій



7

Рисунок В.7 – Слайд 7

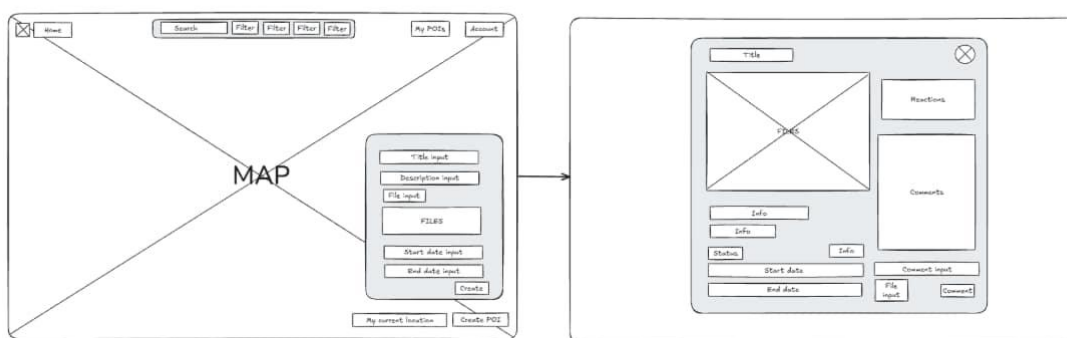
Функціональна структура навігації



8

Рисунок В.8 – Слайд 8

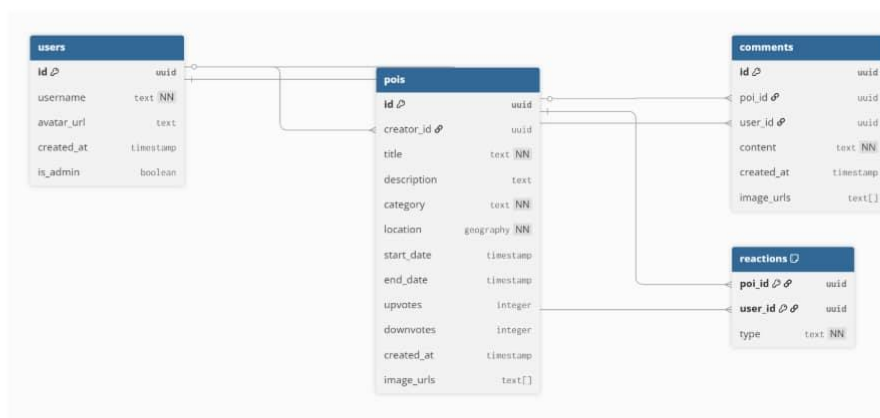
Прототипування користувацького інтерфейсу



9

Рисунок В.9 – Слайд 9

Схема реляційної бази даних застосунку

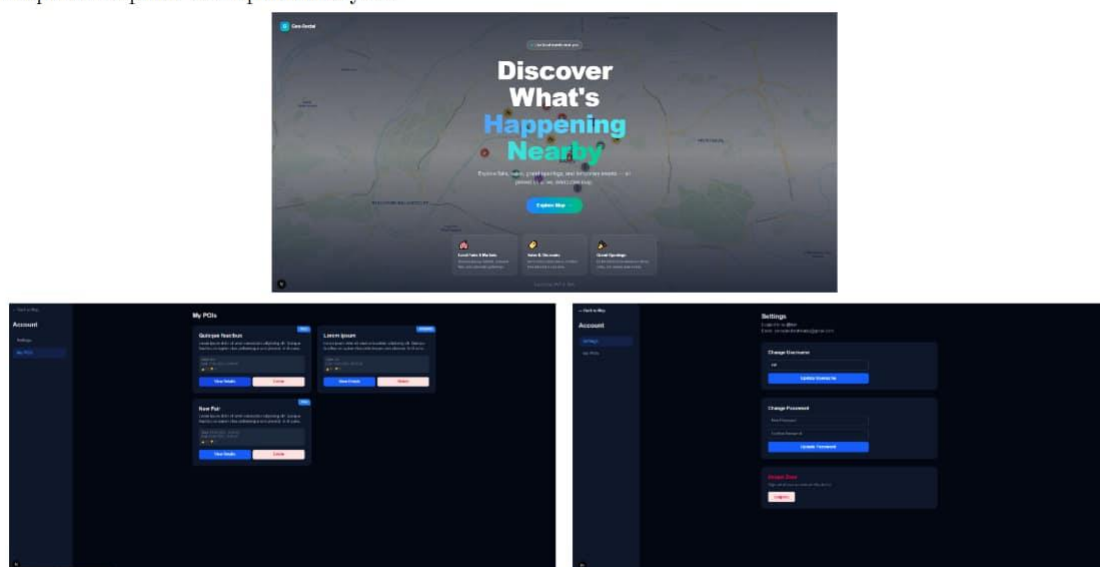


10

Рисунок В.10 – Слайд 10

Інтерфейс розробленого застосунку

Стартова сторінка та сторінка аккаунта

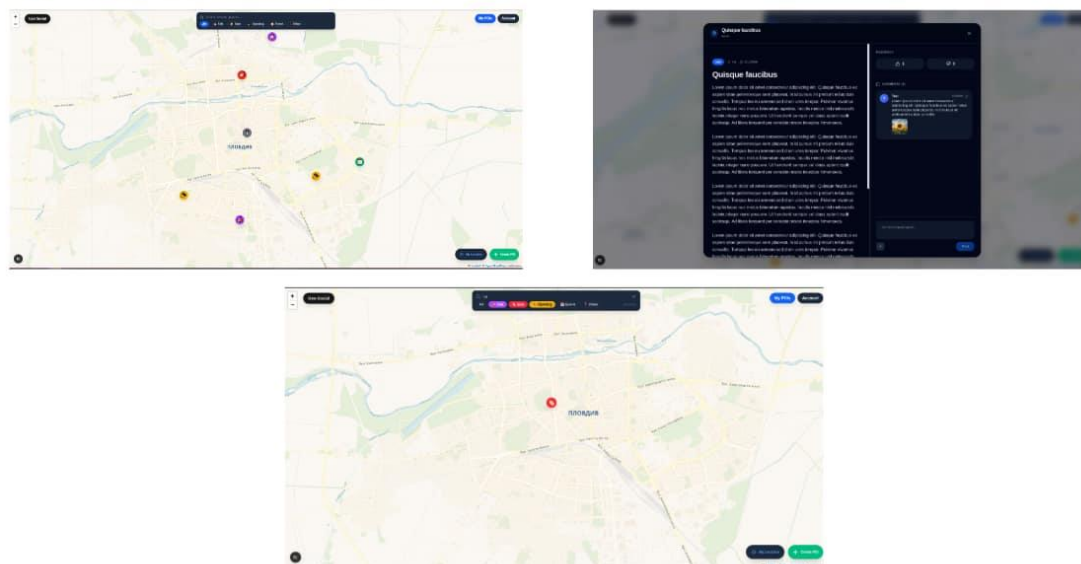


11

Рисунок В.11 – Слайд 11

Інтерфейс розробленого застосунку

Сторінка карти та детальний огляд ROI

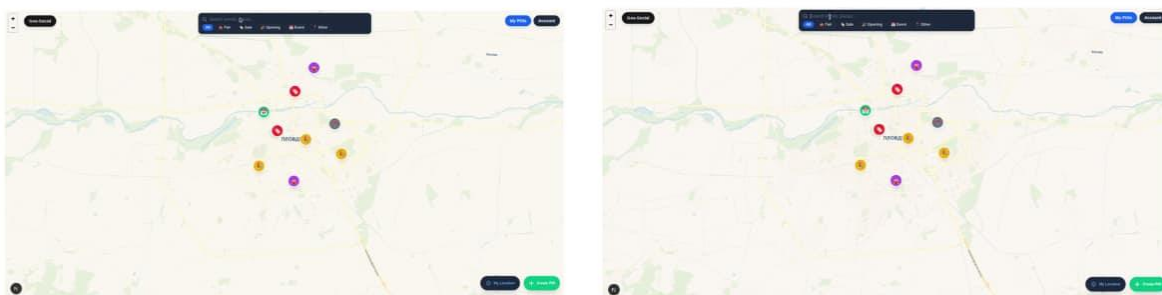


12

Рисунок В.12 – Слайд 12

Інтерфейс розробленого застосунку

Фільтри та пошук



13

Рисунок В.13 – Слайд 13

Висновки

- Розроблено геосоціальний застосунок для моніторингу тимчасових подій у реальному часі.
- Впроваджено просторове індексування PostGIS, що оптимізувало завантаження геоданих.
- Реалізовано атомарну систему соціальної взаємодії для формування локальних рейтингів.
- Доведено ефективність гібридного стека Next.js для побудови продуктивних ГІС-систем.

14

Рисунок В.14 – Слайд 14