Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Веб-додаток для візуалізації змін кордонів з використанням клієнт-серверної архітектури»

ПОЯСНЮВАЛЬНА ЗАПИСКА

08-54.БКР.028.00.000 ПЗ

Виконав: студент 4 курсу, групи 2КІ-21б
спеціальності 123 – «Комп'ютерна інженерія»
освітня програма – «Комп'ютерна ін»енерія»
_____ Зубар М. О.
(прізвище та ініціали)

Керівник: к.т.н. доц. каф. ОТ
_____ Кожем'яко А. В.
(прізвище та ініціали)

Рецензент: к.т.н. доц. каф. ПЗ
_____ Майданюк В.П.
(прізвище та ініціали)

Допущено до захисту
Завідувач кафедри ОТ
д.т.н., проф. Азаров О. Д.
«___» червня 2025 р.

Вінниця ВНТУ – 2025 рік

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки
Освітньо-кваліфікаційний рівень бакалавр
Галузь знань – 12 «Інформаційні технології»
Спеціальність – 123 «Комп'ютерна інженерія»
Освітньо-професійна програма «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ
д.т.н., проф. Азаров О. Д.
«21» березня 2025 року

ЗАВДАННЯ

НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Зубару Максиму Олександровичу

1 Тема роботи: «Веб-додаток для візуалізації змін кордонів з використанням клієнт-серверної архітектури», керівник роботи Кожем'яко А. В. к.т.н., доц., доцент каф. ОТ затвердженні наказом вищого навчального закладу від «20» березня 2025 року №97.

2 Термін подання роботи студентом: 13.05.2025 р

3 Вихідні дані до роботи: програмне середовище vs code, бази даних — PostreSQL, PostGIS, QGIS.

4 Зміст розрахунково-пояснювальної записки: вступ, огляд та аналіз сучасних веб-додатків з географічними даними та історичних веб-додатків з географічними даними, вибір технологій та планування розробки веб-додатку, програмна реалізація веб-додатку, оцінка роботи веб-додатку висновки, список використаних джерел, додатки.

5 Перелік ілюстративного та графічного матеріалу: Карта OpenStreetMap, блок-схема алгоритму розрахунку кордонів наоснові координат GeoJSON, географічні дані для візуалізації кордонів у форматі GeoJSON, рисунки результатів роботи програми, вспливаючі вікна роботи веб-застосунку.

6 Консультанти розділів роботи наведені в таблиці 1.

Таблиця 1 — Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
Розділ 1-4	к.т.н., доц. каф. ОТ Кожем'яко А. В.	21.03.2025	13.05
Нормоконтроль	ас. каф. ОТ Швець С.І.	14.05.2025	30.05

7 Дата видачі завдання: 21.03.2025 р.

8 Календарний план виконання БДР наведено в таблиці 2.

Таблиця 2 — Календарний план

№	Назва та зміст етапу	Термін виконання		Примітка
		початок	закінчення	
1	Вибір, узгодження та затвердження теми БДР	21.03.2025	24.03.2025	Вик.
2	Освоєння необхідних мов програмування для розробки проєкту	25.03.2025	7.04.2025	Вик.
3	Розробка кінцевого змісту для записки	8.04.2025	11.04.2025	Вик.
4	Розробка тези та визначення з кінцевими технологіями використання	12.04.2025	20.04.2025	Вик.
5	Початок пошуку літературних джерел для детального розуміння можливих функцій програми	21.04.2025	26.04.2025	Вик.
6	Початок розробки перших 2 пунктів записки	26.04.2025	29.04.2025	Вик.
7	Програмна реалізація додатку	30.04.2025	12.05.2025	Вик.
8	Попередній захист БДР та доопрацювання всіх аспектів	13.05.2025	10.06.2025	Вик.
9	Нормоконтроль та рецензування БДР	10.06.2025	10.06.2025	Вик.
10	Захист БДР	13.06.2025	13.06.2025	Вик.

Студент Зубар М.О.

Керівник роботи к.т.н., доц. каф. ОТ Кожем'яко А. В.

АНОТАЦІЯ

УДК 004.5

Зубар М.О. Веб-додаток для візуалізації змін кордонів з використанням клієнт-серверної архітектури. Бакалаврська дипломна робота зі спеціальності 123 «Комп'ютерна інженерія», освітня програма «Комп'ютерна інженерія». Вінниця: ВНТУ, 2024. 88 с.

На укр. мові. Бібліогр.: назв 21,: рис.:40,табл.:1.

У бакалаврській дипломній роботі запропоновано підхід до розробки веб-додатка для візуалізації змін державних кордонів у різні історичні періоди. Додаток базується на клієнт-серверній архітектурі та використовує сучасні веб-технології для інтерактивного відображення історичних даних. Основну увагу приділено методам обробки геопросторової інформації, інтеграції бази даних та алгоритмам анімації змін територій.

Ключові слова: візуалізація, веб-додаток, зміна кордонів, клієнт-серверна архітектура, геопросторові дані, REST API, GeoJSON.

ABSTRACT

Zubar M.O. Web application for visualization of boundary changes using client-server architecture. Bachelor's thesis in specialty 123 “Computer Engineering”, educational program “Computer Engineering”. Vinnytsia: VNTU, 2024. 88 p.

In Ukrainian. Bibliogr.: names: 21, pic.40,tab:1

The bachelor's thesis proposes an approach to the development of a web application for visualizing changes in state borders in different historical periods. The application is based on client-server architecture and uses modern web technologies for interactive display of historical data. The main attention is paid to the methods of geospatial information processing, database integration, and algorithms for animating changes in territories.

Keywords: visualization, web application, boundary change, client-server architecture, geospatial data, REST API, GeoJSON.

ЗМІСТ

ВСТУП

Історія розвитку держав та змін їхніх кордонів є однією з найважливіших тем у галузі історії, геополітики та суспільствознавства. У сучасному світі цифрових технологій зростає потреба у зручних інструментах для візуалізації просторових змін у динаміці. Традиційні джерела — друковані карти або статичні зображення — не дозволяють гнучко працювати з історичними даними, не надають можливості інтерактивного перегляду змін і, як правило, не масштабуються на різні періоди часу. У зв'язку з цим актуальною є розробка веб-додатку, який дозволяє інтерактивно візуалізувати зміну державних кордонів у різні історичні епохи.

Метою роботи є створення клієнт-серверного веб-додатку для візуалізації змін державних кордонів у часі з використанням відкритих географічних даних та інтерактивної мапи на основі OpenStreetMap.

Завдання бакалаврської дипломної роботи:

— провести огляд та аналіз сучасних технологій візуалізації геоданих у вебсередовищі;

— обґрунтувати вибір архітектури додатку;

— реалізувати клієнтську частину з використанням бібліотеки Leaflet та реалізувати серверну частину з використанням FastAPI та бази даних PostgreSQL з розширенням PostGIS;

— організувати зберігання геопросторової інформації з часовою прив'язкою;

— зробити простий інтерфейс для користувача.

Об'єктом дослідження є процеси цифрової візуалізації просторових історичних даних з використанням сучасних технологій

Предметом дослідження є програмна реалізація веб-додатку, який дозволяє користувачеві вивчати зміну державних кордонів у різні роки через інтерфейс інтерактивної карти

У роботі використовуються наступні **методи дослідження**:

— аналіз наукової літератури на на тему застосування геоданих в сучасних додатках;

— аналіз наукової літератури на тему технологій які для цього використовуються;

— пошук необхідних технологій та проектування на їх основі веб - додатку;

— пошук необхідних координат для бази даних;

— тестування веб – додатку.

Новизна роботи полягає в тому що дана робота використовуватиме для цього велику кількість координат, приблизно кілька тисяч на одний шар кордонів держави. Це дозволить у свою чергу візуалізувати історичні кордони з високим рівнем деталізації, що часто не притаманне для такого роду веб - додатків.

Результати бакалаврської кваліфікаційної роботи були опубліковані та апробовані на Всеукраїнській науково-технічній конференції факультету інформаційних технологій та комп'ютерної інженерії (2025):

М.О Зубар А.В. Кожем’яко. Веб-додаток для візуалізації змін кордонів з використанням клієнт-серверної архітектури. (2025) ВНТКП ВНТУ. Факультет інформаційних технологій та комп'ютерної інженерії. Отримано з https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki 2025/paper/view/23974/19790 [1]

1 ОГЛЯД СУЧАСНИХ ТЕХНОЛОГІЙ ТА АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ

1.1 Визначення основних понять та проблематики візуалізації змін кордонів

Візуалізація змін кордонів — це процес відображення кордонів на цифрових платформах з плином часу. На відміну від звичайних картографічних зображень, де дані організовані в статичній формі та досить важкій для сприйняття пересічною людиною, використання візуалізації допомагає полегшити сприйняття інформації нею. Це досягається насамперед, збільшенням інтерактивності за допомогою цифрових технологій. Однак, збільшення функціоналу того чи іншого продукту супроводжується супутніми проблемами, які виникають по мірі виконання поставленої цілі. Візуалізація карт має ряд проблем з якими необхідно зіштовхнутися. Для того щоб краще їх зрозуміти потрібно розібрати ключові терміни що зробить розуміння проблем чіткішими:

— візуалізація змін кордонів — це процес відображення трансформацій територіальних меж протягом певного історичного періоду, який може включати як просторову (географічну), так і часову складову, наприклад, у вигляді інтерактивної карти, що дозволяє переглядати динаміку змін;

— геоінформаційна система (ГІС) — це апаратно-програмний комплекс для збору, зберігання, обробки, аналізу та візуалізації просторових (географічних) даних, ГІС використовується в багатьох галузях, включаючи картографію, історичні дослідження, міське планування;

— історична ГІС (ІГІС) — це різновид геоінформаційних систем, який застосовується для збереження та аналізу даних про територіальні зміни у минулому та дозволяє інтегрувати історичні карти, статистику та текстові джерела з географічними координатами [2].

З поданої вище інформації можна зробити висновок, що даний проєкт базується насамперед на роботі з георафічними даними, через що розробка веб-додатку для візуалізації змін кордонів стикнеться з низкою таких проблем:

— обмеженість історичних даних. карти минулих століть, особливо до нового часу не деталізовані та не точні через слабкий технологічний розвиток, релігійні та політичні погляди того часу;

— різнорідність джерел. історичні карти, архівні документи, демографічні таблиці — це різні формати даних, які потребують попередньої обробки, конвертації та узгодження;

— складність геопросторової обробки. необхідно забезпечити правильне географічне позиціонування об'єктів у часовому вимірі. це потребує використання відповідних гіс-інструментів і знань;

— політична чутливість. відображення деяких територіальних змін може бути спірним внаслідок історичного розвитку держави, народу, етносу. важливо дотримуватися нейтрального підходу та посилатися на офіційні джерела;

— стандартизація та зберігання. З метою зручності інтеграції в базу даних та подальшої обробки дані мають зберігатися у стандартизованому форматі, наприклад, GeoJSON або у просторовій СУБД на базі PostgreSQL з розширенням PostGIS[3].

1.2 Огляд сучасних веб-додатків для роботи з географічними даними

З розвитком технологій, оцифровані географічні дані стали важливим аспектом багатьох сфер діяльності людини, які пов’язані з простором, географією та економікою. Насамперед, йдеться про логістику та урбаністику. Потреба у відповідних даних зростає щороку та у свою чергу створює запит на відповідне програмне забезпечення яке ми зараз детально розглянемо.

Google Maps — це одна з найпотужніших веб - програм відповідної спеціалізації на сьогодні. Даний веб-картографічний сервіс представлений від компанії Google, яким користується понад 1 млрд. користувачів щороку. Таку широку популярність додаток здобув завдяки величезній кількості інформації інтегрованій в ній та широкому наборі користувальницьких функцій. Google Maps надає можливість прокладати маршрути, переглядати супутникові знімки місцевості, отримувати інформацію про трафік у реальному часі. Цікавою

особливістю додатку є режим Street View, що дозволяє вивчати панорамні види тої чи іншої місцевості з точки зору людини, яка там знаходиться. Крім того, платформа активно інтегрується з іншими сервісами Google, підтримує бізнес-рейтингування та дозволяє залишати відгуки про заклади, сприяючи точному орієнтуванню та покращенню якості обслуговування. Постійний розвиток технологій, зокрема штучного інтелекту сприяють вдосконаленні системи, тим самим зробивши даний веб-сервіс важливим інструментом як для звичайного користувача так і для професійного, наприклад, логіста. На рисунку 1.1 нижче зображено інтерфейс даної програми:

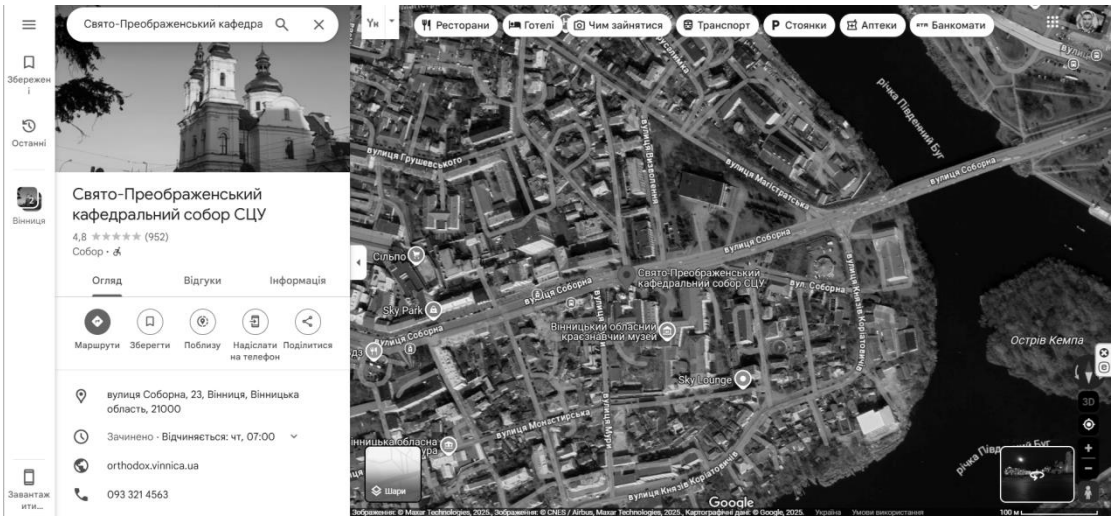

Рисунок 1.1 — Інтерфейс веб-програми Goggle Maps

OpenStreetMap (OSM) — це також один із найпотужніших веб-картографічних сервісів на сьогодні. Його головною особливістю є відкритість та управління даними всесвітньою спільнотою. Проєкт є глобальною ініціативою, якою користуються мільйони людей та організацій у всьому світі. Кожен може вносити зміни до карти світу, редагуючи «ноди», «шляхи» і «відношення» з відповідними тегами для опису об’єктів інфраструктури та природних елементів. Популярність платформи обумовлена її відкритими даними, що дозволяють використовувати карту для різноманітних задач — від навігації до геоаналітики. OpenStreetMap надає можливість створювати та редагувати картографічні дані, отримувати детальні топографічні карти, а також аналізувати геопросторову

інформацію. Унікальність сервісу полягає у тому, що карта постійно оновлюється безпосередньо користувачами, що забезпечує її високу точність та актуальність. Крім того, платформа активно інтегрується із зовнішніми сервісами та додатками, підтримує використання у професійних галузях, таких як логістика, урбаністика та гуманітарні місії. Завдяки постійному розвитку технологій, включаючи машинне навчання, OpenStreetMap стає важливим інструментом для аналізу, навігації та дослідження територій. На рисунку 1.2 зображено інтерфейс даної програми:

Рисунок 1.2 — Інтерфейс веб-програми OpenStreetMap

Mapbox — це одна з найпотужніших платформ для візуалізації та аналізу геопросторових даних. Сервіс надає можливість створювати персоналізовані карти, адаптувати стиль та масштабування відповідно до потреб користувача. Mapbox використовується у різних сферах — від урбаністики та логістики до мобільних додатків і навігаційних систем. Підтримує детальні топографічні карти, прокладання маршрутів, аналіз транспортного потоку та пошук географічних об'єктів. Ключовою особливістю сервісу є можливість інтеграції власних даних, налаштування стилів через Mapbox Studio та використання широкого набору API для роботи з геокодуванням, навігацією та просторовими запитами.

Цікавою функцією платформи є її здатність до кастомізації: розробники можуть створювати спеціалізовані візуалізації даних, інтегрувати тривимірні моделі та застосовувати алгоритми машинного навчання для аналізу геопросторової інформації. Завдяки адаптивності та можливості інтеграції з іншими технологіями, Mapbox став важливим інструментом для професійних розробників, транспортних аналітиків і фахівців з міського планування.

Mapbox формує інтерактивний простір для аналізу, моделювання та взаємодії з географічним середовищем, а не лише надає відповідну картографічну інформацію. Функціональність даного середовища більш ширша, аніж наприклад, у Google Maps чи OSM, однак, дане середовище платне що дозволило двом останнім програмам нав'язувати більшу конкуренцію[4]. Інтерфейс даної програми зображено на рисунку 1.3.

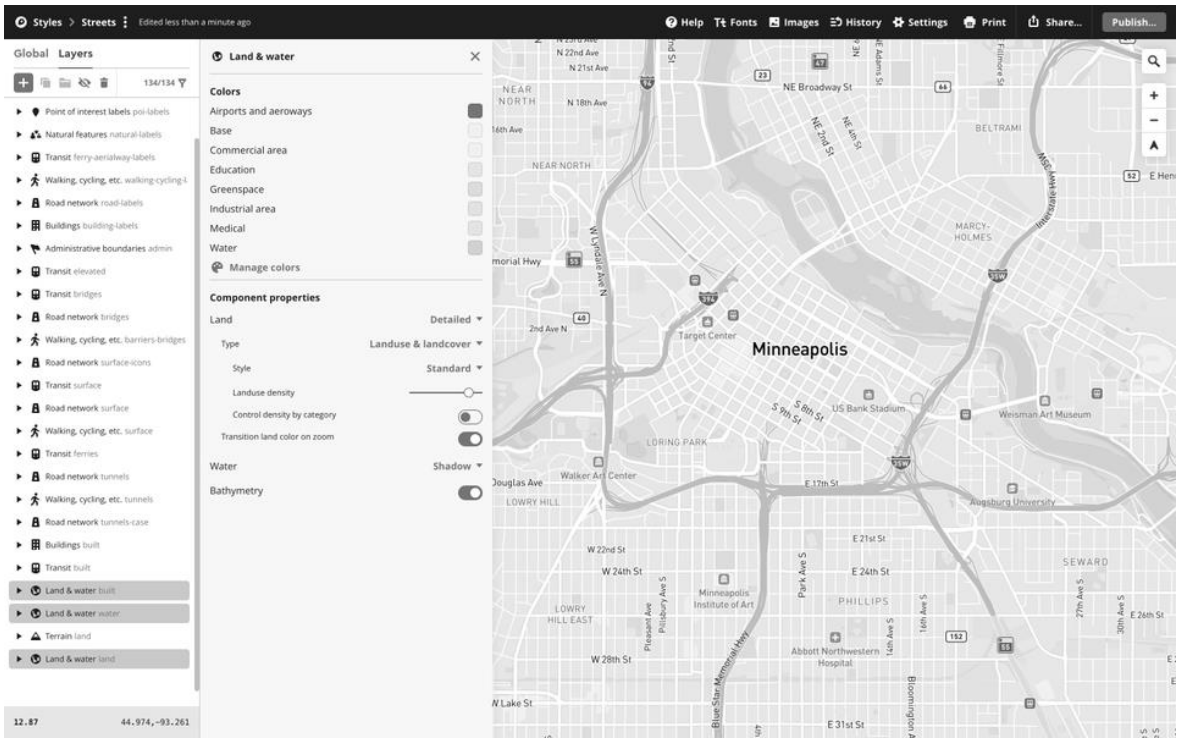

Рисунок 1.3 — Приклад одного з інтерфейсів платформи Mapbox

Mango Map — це одна з найзручніших веб-платформ для створення та управління інтерактивними картами. Даний геоінформаційний сервіс надає користувачам можливість швидко візуалізувати просторові дані без необхідності програмування чи складних технічних налаштувань. Платформа дозволяє легко

додавати шари даних, інтегрувати різні типи геоінформації та налаштовувати візуальні елементи для забезпечення максимальної деталізації. Mango Map підтримує створення приватних і публічних карт, що дає змогу контролювати доступ до інформації та керувати рівнями редагування. Завдяки інтуїтивному інтерфейсу користувачі можуть швидко імпортувати дані, аналізувати їх та створювати наочні інтерактивні візуалізації. Цікавою особливістю сервісу є його орієнтованість на доступність: веб-додаток працює без необхідності встановлення додаткового програмного забезпечення, а всі дані зберігаються у хмарі, що дозволяє легко ділитися картами з колегами або клієнтами. Mango Map також підтримує інтеграцію з GIS-системами та надає можливість імпорту файлів у популярних форматах, таких як GeoJSON та shapefiles. [4] Інтерфейс платформи зображено на рисунку 1.4 нижче.

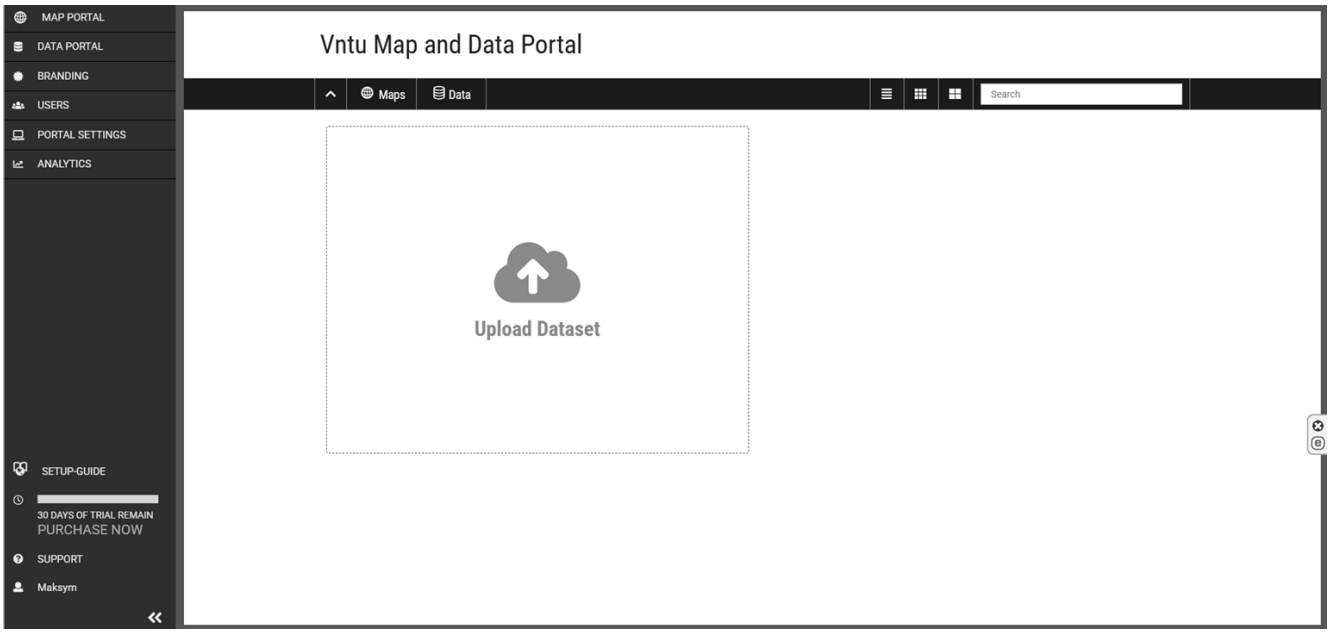

Рисунок 1.4 — Інтерфейс платформи Mango Map, для створення цифрової карти

1.3. Аналіз аналогів та їх функціональних можливостей

Розробка веб-додатку для візуалізації змін кордонів потребує ґрунтовного порівняння існуючих інструментів, що використовують геоінформаційні системи та інтерактивні картографічні сервіси. Аналіз дозволяє виділити рішення з

необхідним ступенем кастомізації, продуктивності та підтримкою просторово-часових даних, а також врахувати особливості клієнт–серверної архітектури.
Для цього ми проведемо аналіз аналогів та їх функціональних можливостей на основі описаних вище програм, за такими важливими критеріями.

Функціональність — важливий аспект інтерактивних картографічних сервісів. Даний критерій включає можливість редагування, аналізу та версіювання даних та дозволяє гнучко адаптувати карти до дослідницьких завдань. Наявність багатошарових структур, просторової аналітики та варіативності візуалізації є важливою особливістю для даного критерію.

Гнучкість інтеграції — визначає можливість розширення функціоналу. Інструменти повинні підтримувати стандартизовані формати — GeoJSON, EsriJSON, TopoJSON, а також API для доступу до даних. Сумісність із сторонніми сервісами та можливість кастомізації визначає їхню ступінь застосування у розробці складних веб-програм.

Підтримка часових змін — візуалізація історичних меж вимагає коректного відображення змін у часовій перспективі. У свою чергу передбачає інтеграцію таймлайнів, метаданих start/end date та архівних оверлеїв, що дозволяє аналізувати динаміку розвитку територій та геополітичні трансформації.

Ліцензування та вартість — інструмент може розповсюджуватися за моделлю Open-source, Freemium або SaaS, що визначає його доступність для дослідників та розробників. Важливим є співвідношення функціональності та вартості, особливо якщо враховувати витрати на хостинг, обробку геоданих та обмеження API.

Технічний стек — визначає гнучкість впровадження технології. Клієнтська частина може базуватися на JavaScript, React або Vue, тоді як серверна підтримує FastAPI, Django чи Node.js. Сумісність із сучасними розробницькими фреймворками забезпечує можливість кастомізації та адаптації під конкретні задачі.

SQL та бази даних — просторові розширення, такі як PostGIS, важливі для коректного збереження та обробки даних. Інтеграція з SpatiaLite або QGIS

підвищує точність обчислень, а оптимізація SQL-запитів впливає на продуктивність системи та швидкість обробки історичних даних.

Клієнт-серверна архітектура — визначає комунікаційні механізми, такі як REST API або WebSocket, що забезпечують реальну інтерактивність. Масштабованість рішень важлива для підтримки великих наборів геоданих, а динамічне завантаження гарантує ефективну роботу сервісу без зайвого навантаження на систему.

1.3.1 Інтерактивні карти та геоінформаційні системи

Інтерактивні картографічні сервіси поєднують на стороні клієнта потужні JavaScript-бібліотеки та фреймворки для відображення карт, а на стороні сервера — високопродуктивні API та географічні API - ендпоїнти, які забезпечують доступ до різноманітних просторових даних. Кожна платформа розроблена з урахуванням специфіки застосування: від легких віджетів до комплексних аналітичних систем.

Google Maps використовує JavaScript та TypeScript у клієнтському SDK, а серверні компоненти реалізовані на різних мовах, включаючи Java, .NET та Python. Хоча у платформі немає прямої підтримки SQL-запитів, вона забезпечує доступ до власних бекенд-сервісів через REST API та WebSocket для отримання реального трафіку та оновлень у режимі реального часу.

OpenStreetMap базується на серверному програмному забезпеченні, написаному переважно на C++, із клієнтськими інтерфейсами на JavaScript (Leaflet) та інструментами обробки на Python. Для розгортання локальних копій рекомендується використовувати PostgreSQL з розширенням PostGIS або легшу альтернативу SQLite-Spatialite, а комунікація між клієнтом і сервером побудована на REST-запитах через OSM API та складні вибірки — через Overpass API.

Mapbox поєднує клієнтську бібліотеку Mapbox GL JS на JavaScript із серверною частиною, реалізованою на Python, .NET та Java. Запити до векторних тайлів і стилів здійснюються через власний SQL API, а для відображення даних у реальному часі використовуються WebSocket-сервіси Mapbox Traffic.

ArcGIS Online надає розвинуті клієнтські SDK на JavaScript (ArcGIS JS API), Python (ArcGIS API for Python) та .NET (ArcGIS Runtime), а серверна інфраструктура спирається на Esri File Geodatabase та PostgreSQL з PostGIS через ArcGIS Enterprise. Клієнт–серверна взаємодія реалізована через REST API з можливістю стрімінгу даних через GeoEvent Server.

CARTO інтегрує на клієнті JavaScript-бібліотеку CARTO.js і SDK для Python та Ruby, а на сервері використовує PostgreSQL з PostGIS, до якого звертаються через SQL API. Архітектурно платформа надає REST- та GraphQL ендпоїнти, що дозволяють виконувати складні просторово-часові запити.

MangoMap спрощує картографічний процес для кінцевих користувачів, застосовуючи у фронтенді Vue.js, а у бекенді — Node.js із простим JSON орієнтованим сховищем. Для передачі даних між клієнтом та сервером використовується базовий REST API, який підтримує імпорт та експорт шарів у CSV і GeoJSON.

Детальне порівняння представлено на таблиці 1.

Таблиця 1 — Порівняння функціональних можливостей інтерактивних карт та геоінформаційних систем

Платформа	Ліцензія/вартість	Підтримка історії	Мови програмування	SQL/BD
Google Maps	Комерційна (Freemium)	Ні	JavaScript, TypeScript, Java, .NET, Python	Власні сервіси (без SQL)
OpenStreetMap	Open-source (ODbL)	Часткова	C++, JavaScript, Python	PostgreSQL + PostGIS, SQLite
Mapbox	Freemium/Платна	Ні	JavaScript, Python, .NET, Java	Mapbox SQL API
ArcGIS Online	SaaS (різні тарифи)	Так (через сервер)	JavaScript, Python, .NET	Esri File Geodatabase, PostgreSQL + PostGIS
CARTO	Freemium/Платна	Ні	JavaScript, Python, Ruby	PostgreSQL + PostGIS
MangoMap	Платна ($50/міс.)	Ні	JavaScript (Vue.js), Node.js	NoSQL/JSON

Таблиця 1 — продовження порівняння функціональних можливостей

Платформа	Архітертура клієнт — сервера	Основні функції програм
Google Maps	REST API, WebSocket	Маршрути, Street View, геокодування
OpenStreetMap	REST API (OSM), Overpass API	Відкриті шари, історія змін, експорти
Mapbox	REST API, WebSocket	Кастомні стилі, 3D-ландшафти
ArcGIS Online	REST API, WebSocket	Просторовий аналіз, колаборація
CARTO	REST API, GraphQL	Аналітичні панелі, ETL, Data Observatory
MangoMap	REST API	Редактор, WMS, тури через iframe

1.3.2 Веб-додатки з історичними даними про кордони

Веб-додатки, що працюють із історичними даними про кордони, є ключовим інструментом для дослідження територіальних змін. Вони не лише спрощують доступ до інформації, а й роблять її більш зрозумілою та інтерактивною. Завдяки таким платформам науковці, аналітики та звичайні користувачі можуть досліджувати, як змінювалися межі держав у різні історичні періоди, аналізувати геополітичні трансформації та простежувати зв’язок між територіальними змінами та соціально-економічними процесами.

Кожен веб-додаток має свою специфіку: структура даних, спосіб взаємодії користувача, доступні джерела імпорту — усе це впливає на зручність та ефективність роботи з історичною інформацією. Одні платформи базуються на традиційних ГІС-рішеннях і дозволяють глибоке картографічне моделювання, інші орієнтовані на швидку інтеграцію та легкий доступ до даних через API. Для дослідника важливо не тільки отримати картографічне зображення, а й мати змогу аналізувати просторово-часові аспекти змін. Інструменти, які підтримують таймлайни, архівні оверлеї, дають можливість глибше зрозуміти динаміку розвитку територій.

TimeMapper — це відкритий веб-додаток, що поєднує інтерактивну хронологічну стрічку та географічну карту для візуалізації історичних подій у часі та просторі. Його ключова особливість—простота використання. Користувачам достатньо заповнити таблицю в Google Sheets, а система автоматично генерує

таймлайн-карту, синхронізовану з відповідними геоданими. Це робить TimeMapper надзвичайно доступним навіть для тих, хто не має глибоких технічних знань.

Розробка базується на використанні відкритих бібліотек, що забезпечують інтерактивність та гнучкість. TimelineJS відповідає за візуалізацію хронології подій, Leaflet—за інтерактивну карту, а ReclineJS, Backbone і Bootstrap формують загальну структуру та стиль додатка. Це дозволяє створювати детальні, насичені візуалізації, що можуть бути легко інтегровані на веб-сайти завдяки можливості вбудовування HTML-коду. Технічно TimeMapper працює на основі Node.js та Express для серверної частини, а дані зберігаються у Google Sheets. Клієнтська взаємодія здійснюється через JavaScript, що гарантує швидкість та інтерактивність. Ця архітектура робить платформу ефективною, оскільки будь-які зміни у таблиці миттєво відображаються у кінцевому продукті.

Головні переваги TimeMapper—це простота використання, гнучкість та відкритий код. Його можуть застосовувати в освітніх проєктах, журналістських розслідуваннях і наукових дослідженнях, спрощуючи роботу з історичними даними. Однак платформа має певні обмеження: залежність від Google Sheets, обмежені можливості кастомізації та обмеження на кількість оброблюваних рядків. Для глибших аналітичних досліджень або роботи з великим масивом історичних даних можуть знадобитися додаткові інструменти та розширення[5]. Інтерфейс даної програми зображено на рисунку 1.5.

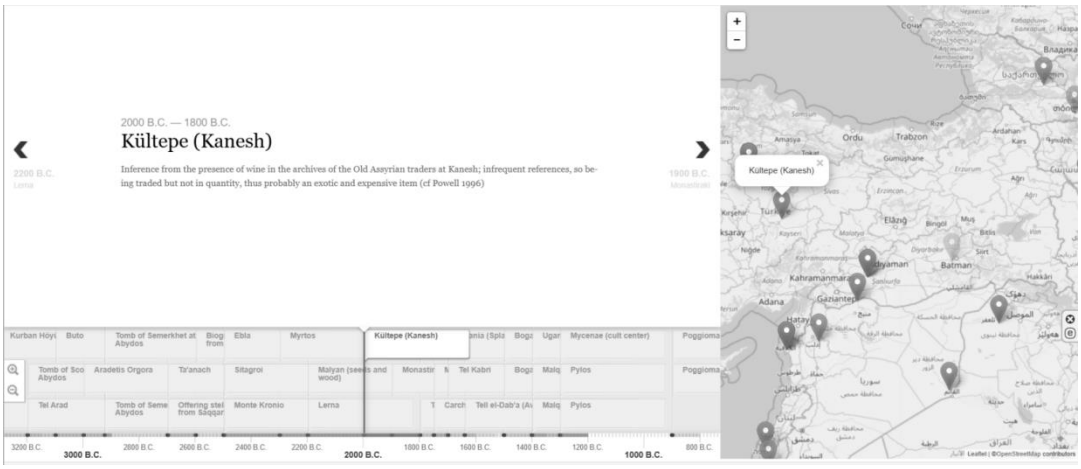

Рисунок 1.5 — Інтерфейс програми TimeMapper

Historypin — це унікальна платформа, яка відкриває нові можливості для вивчення історії через інтерактивні візуалізації. Вона дозволяє користувачам зі всього світу "прикріплювати" фотографії, відео, історичні документи та розповіді до карт, створюючи цифровий архів змін місць і подій. Це не просто інструмент для перегляду старих зображень, а спосіб взаємодії з минулим, дослідження локальної історії та збереження спадщини для майбутніх поколінь.

Однією з головних особливостей Historypin є її здатність геоприв’язувати контент. Користувачі можуть розміщувати матеріали відповідно до географічних координат і часу їх створення, що дозволяє вивчати зміни міст, районів та культурних ландшафтів. Завдяки такій інтеграції можна спостерігати, як еволюціонувала міська архітектура, як зникали чи перетворювалися історичні будівлі, як змінювалося суспільство через ключові події минулого.

Окрім простого розміщення фотографій, Historypin підтримує створення тематичних колекцій. Це дозволяє групувати матеріали за певними історичними періодами, подіями або регіонами, надаючи більш структуроване бачення історичних змін. Завдяки таким добіркам можна створювати навчальні ресурси, інтерактивні тури та дослідницькі проєкти. Крім того, платформа активно залучає спільноту: люди можуть коментувати, додавати додаткові відомості та брати участь у збереженні локальної історії.

Застосування Historypin виходить далеко за межі простого архівування. Вона використовується в освітніх програмах, музейних ініціативах та громадських проєктах, допомагаючи відновлювати культурну пам’ять. Можливість накладати старі фотографії на сучасні карти дозволяє краще зрозуміти трансформацію суспільства та взаємозв’язок між історією та сьогоденням. Інтерфейс Historypin зображено на рисунку 1.6.

Historic Borders (HistoryMap) – це веб-додаток, призначений для динамічної візуалізації змін державних кордонів протягом історії, починаючи з 123 000 року до н. е. і завершуючи 1994 роком, що дозволяє користувачеві досліджувати геополітичні трансформації у широкому хронологічному діапазоні. Інтерфейс додатку реалізовано за допомогою React та Mapbox GL JS, що забезпечує плавний

рендеринг як плоских карт, так і глобуса, а також інтерактивну панель із часовою шкалою, яка дозволяє миттєво переходити між обраними датами й переглядати відповідні кордони . Основою для відображення просторових об’єктів слугують векторові GeoJSON-шари, зібрані в репозиторії historical-basemaps, які оновлюються спільнотою й містять подробиці про регіональні кордони різних епох.

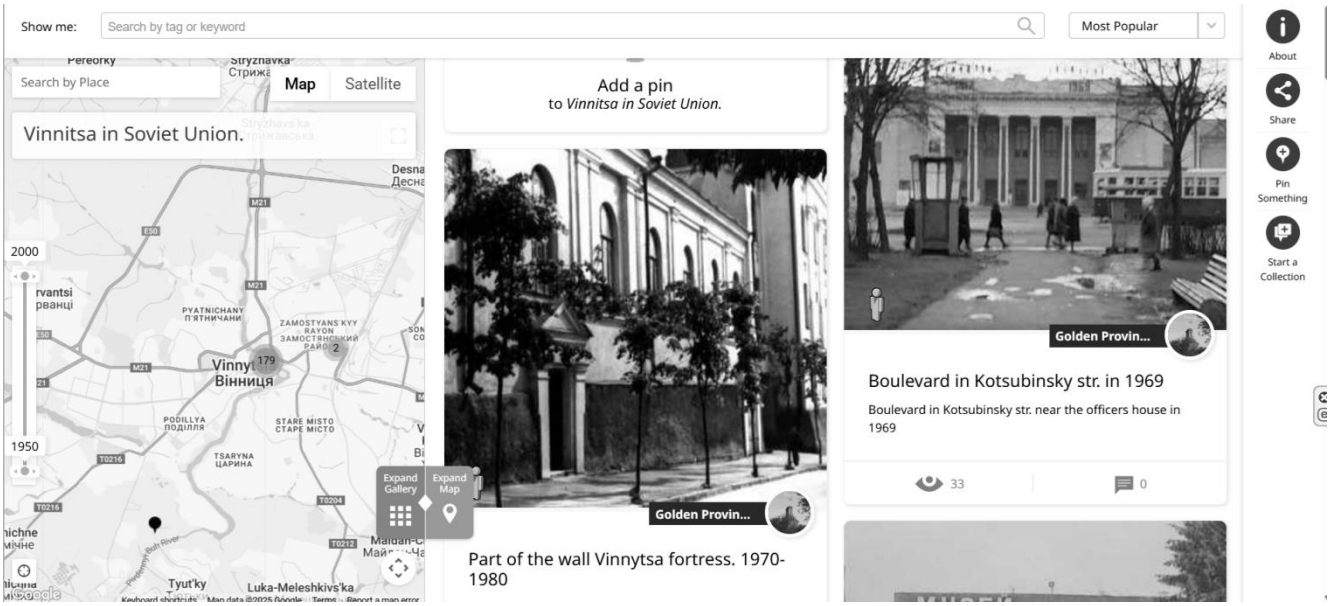

Рисунок 1.6 — Інтерфейс програми Historypin

Time slider відкриває доступ до режиму “globe view”, де зміни меж відображаються на тривимірній моделі Землі, що допомагає краще уявити просторові взаємозв’язки й розповсюдження державних утворень у часі . Користувацький досвід доповнюють інтерактивні інформаційні підказки, які відображають назву держави та ключові історичні події, пов’язані з обраною датою, а також гістограма інтенсивності змін території, що демонструє темпи розширення чи звуження меж у досліджуваному інтервалі. Нижче на рисунку 1.7 зображено інтерфейс Historic Borders.

Додаток підтримує пошук за назвою країни чи регіону, а також експортує статичне зображення карти або копію GeoJSON для подальшого аналізу в ГІС-середовищах. Архітектура платформи передбачає клієнт-серверну взаємодію через REST-ендпоінти Mapbox та власний бекенд на Node.js, який відповідає за

завантаження GeoJSON-шарів і сервісів тайлових карт. Historic Borders пропонує відкритий вихідний код під ліцензією MIT, що сприяє прозорості розвитку й залученню спільноти до покращення даних і функцій проекту . Завдяки простоті інтеграції й можливості працювати без встановлення додаткового ПЗ, додаток став популярним інструментом для викладачів, дослідників і ентузіастів цифрових гуманітарних наук, які прагнуть швидко та наочно дослідити еволюцію політичних меж у глобальному масштабі.

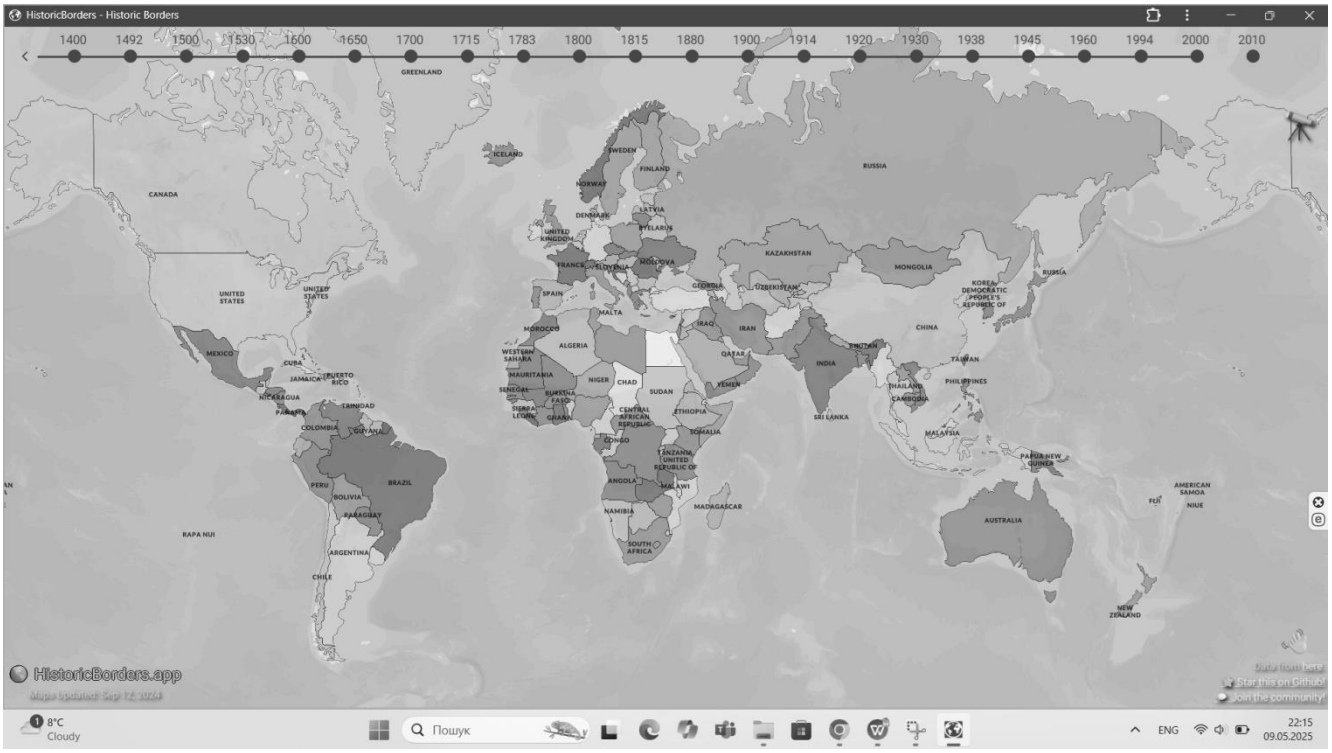

Рисунок 1.7 — Інтерфейс програми Historic Borders

GeaCron — це інтерактивний світовий історичний атлас, який охоплює період від 3000 року до н.е. до сучасності, надаючи користувачам можливість досліджувати геополітичні зміни через час за допомогою динамічної карти та налаштовуваних шкал часу. Створений Луїсом Мускісом, GeaCron поєднує знання з історії та інформаційних технологій, використовуючи OpenLayers, MySQL, PHP і Python для представлення історичних даних у геопросторовому форматі. Користувачі можуть переміщатися по карті, змінювати масштаб і переглядати політичні межі в будь-який момент історії, з кроками в 1, 10 або 100 років. Система дозволяє шукати країни, міста чи події, а також створювати

посилання для спільного використання конкретних історичних і географічних ситуацій. Додаткові функції включають відображення історичних подій, битв, міст та експедицій, а також можливість вбудовування карт у зовнішні веб-сайти за допомогою iframe. GeaCron доступний на декількох мовах, включаючи англійську, іспанську, французьку, португальську, китайську, італійську та німецьку, і підтримує мобільні пристрої на платформах iOS та Android. Завдяки своїй гнучкості та багатофункціональності, GeaCron є цінним інструментом для викладачів та ентузіастів історії, які прагнуть візуалізувати та аналізувати історичні дані в ГІС контексті[6]. На рисунку 1.8 інтерфейс GeaCron.

Рисунок 1.8 — Інтерфейс програми GeaCron

1.4 Вимоги до клієнт-серверної архітектури для візуалізації даних

Проєктування клієнт-серверної архітектури для веб-програми з візуалізації історичних змін кордонів буде базуватися на комплексному врахуванні функціональних і нефункціональних вимог, що забезпечать ефективну роботу з просторово-часовими даними. В основу архітектури лягає трирівнева модель: клієнтський інтерфейс, прикладний сервер та сервер баз даних[7] із просторовими можливостями.

На рівні клієнтського інтерфейсу реалізується безпосередньо візуалізація картографічних шарів і часових ліній (таймлайнів), обробка взаємодії користувача та динамічне завантаження необхідних геометрій без повного перезавантаження сторінки. Інтерфейс має підтримувати адаптивність (responsive design) для різних типів пристроїв та забезпечувати високий рівень інтерактивності: зокрема, реалізовувати обробку подій зміни масштабу, панорамування карти, вибору часових зрізів і фільтрації тематичних шарів.

Серверна частина відповідає за обробку запитів клієнта, підготовку відповідних просторово-часових даних, фільтрацію, агрегацію інформації та передачу результатів назад у зручному форматі (наприклад, GeoJSON). Важливо, щоб сервер підтримував обробку декількох паралельних сесій і масштабувався в залежності від кількості користувачів. Також архітектура має враховувати можливість кешування даних для зменшення навантаження та прискорення роботи з популярними запитами.

Ключовим елементом є база даних, яка повинна зберігати як просторову (геометрію об'єктів), так і часову (дати існування, правки) інформацію. Для зберігання поліліній, полігонів[7] і точкових об’єктів має використовуватись просторове розширення з підтримкою геометричних типів та індексації. Крім того, база даних повинна забезпечувати версіонування об'єктів (наприклад, державні межі в різні історичні періоди) та підтримувати запити, які поєднують географічну і часову компоненти (наприклад, показати межі на 1916 рік).

Оскільки ми працюємо із веб-додатком у даному проєкті мають місце такі нефункціональні вимоги:

— масштабованість — це коли система повинна обробляти як невеликі навчальні проекти, так і великі історичні бази даних із багаторічними змінами;

— продуктивність — це коли відповідь сервера на картографічні запити не повинна перевищувати декількох сотень мілісекунд;

— доступність — це коли архітектура має підтримувати багатокористувацький доступ без конфліктів даних;

— безпека — це коли повинна бути реалізована базова автентифікація, захист арі-запитів та обмеження доступу до редагування.

Отже, вимоги до даної клієнт-серверної архітектури диктуються специфікою історичних даних, які мають складну часову структуру, значний обсяг просторової інформації та високі вимоги до точності подання. Такі системні особливості визначають структуру даного веб-додатку в якому закладена трирівнева модель з клієнтського інтерфейсу, прикладного сервера і сервера баз даних. Наявна архітектура архітектура гарантує високу продуктивність, точність та масштабованість, що є ключовими для інтеграції історичних даних у сучасні аналітичні платформи.

2 ПРОЕКТУВАННЯ ВЕБ-ДОДАТКУ

2.1 Вибір технологій для клієнтської частини

Після того як було виконано теоретичну роботу з даним проєктом, можна переходити на етап програмної реалізації. Для початку ми розглянемо та завантажимо програми які ми будемо використовувати для клієнтської частини даного програмного продукту. Клієнтська частина у веб-додатку виступає інтерфейсом взаємодії користувача з даними та візуалізацією. Для створення зручного, адаптивного й високопродуктивного інтерфейсу необхідно ретельно обрати інструменти та технології, які забезпечать швидку і злагоджену роботу розробника, а кінцевому користувачеві — комфортну роботу з картиною змін кордонів.

Основним середовищем розробки було обрано Visual Studio Code [8] — сучасний, кросплатформенний редактор із широкими можливостями налаштування та розширення. VS Code було розроблено компанією Microsoft. Дане програмне забезпечення підтримує такі операційні системи як macOS , Linux та Windows. Великою перевагою даного програмного середовища є підтримка численних мов програмування, включаючи Python , C++ , Go , JavaScript та Node.js. Окрім базових функцій, таких як згортання коду, підсвічування синтаксису та зіставлення дужок, інші функції можуть відрізнятися залежно від мови програмування. Користувачі можуть використовувати вільно доступне розширення для додаткової підтримки мов, тем та налагоджувачів. VS Code забезпечує:

— швидке завантаження та легкий запуск на будь-якій операційній системі;

— високий ступінь інтеграції з браузером через Live Server для автоматичного перезавантаження при змінах коду;

— можливість налаштування середовища за допомогою тем, команд і Snippet'ів, що прискорює розробку HTML/CSS/JS компонентів.

Таким чином, Visual Studio Code (VS Code) є провідним інструментом у сфері сучасної веб-розробки завдяки поєднанню легкості, гнучкості й розширюваності. Відкрита архітектура розширень, інтерфейс із багатовіконним редактором, інтегрований термінал та вбудована система контролю версій (Git) роблять VS Code оптимальним вибором як для фронтенд, так і для повноцінної full-stack розробки.

Після встановлення редактору ми повинні додатково встановити розширення для VS Code. Для клієнтської частини нам необхідно завантажити такі з них:

— HTML CSS Support (ECMEL);

— Live Server;

— Formatting Toggle;

— Prettier – Code formatter.

HTML CSS Support (ECMEL) — розширення додає інтелісенс (автодоповнення коду) для HTML та CSS: підказки за атрибутами, автодоповнення та швидкий доступ до документації. Це суттєво прискорює написання розмітки та стилів, гарантуючи коректність синтаксису та прискорюючи розробку адаптивного інтерфейсу.

Live Server — Плагін запускає локальний HTTP - сервер із автоматичним перезавантаженням сторінки при зміні файлів HTML, CSS або JavaScript. Завдяки йому розробник миттєво отримує зворотній зв'язок від нього ж — це критично для роботи з картографічними компонентами, де будь-яку зміну стилів або зміну коду потрібно негайно перевірити в браузері.

Formatting Toggle — дозволяє швидко вмикати чи вимикати автоматичне форматування що зручно при роботі над великими фрагментами коду або під час рефакторингу. Забезпечує баланс між строгим дотриманням стандартів та гнучкістю «ручних» змін.

Prettier – Code formatter — автоматичний форматер коду, що об’єднує стилі написання JavaScript, HTML та CSS за єдиними правилами. Налаштування Prettier гарантує послідовність відступів, розривів рядків і розміщення крапок з комою, що полегшує командну розробку та знижує ризик конфліктів при злитті гілок.

Нижче зображено рисунок 2.1, на якому видно встановлені додатки

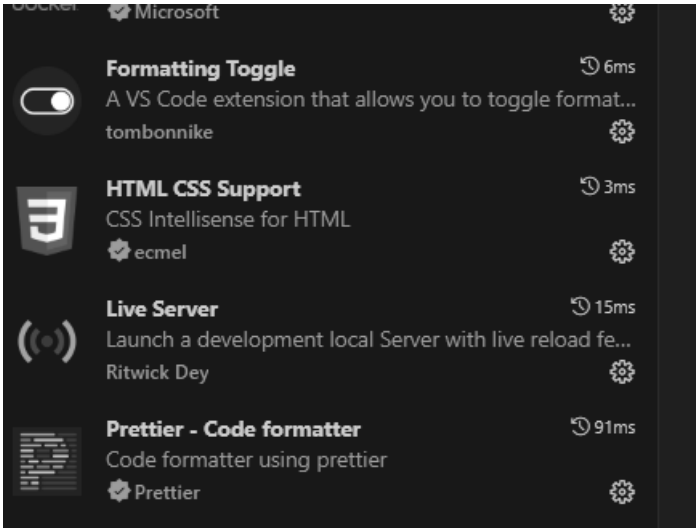

Рисунок 2.1 — Встановлення розширень для архітектури клієнта

Такий вибір інструментів є достатнім для розробки клієнтської бази. Наявне середовище та програми дозволяють швидко розробляти, тестувати й підтримувати клієнтський інтерфейс із високим рівнем адаптивності та інтерактивності. Багато необхідних елементів автоматично завантажені разом з VS Code. Зокрема, JavaScript, HTML,.

2.2 Вибір технологій для серверної частини

Після того як були встановлені необхідні розширення для клієнтської архітектури, можна переходити до встановлення архітектури сервера. Головним чином це стосується мови програмування пайтон на якій буде виконана серверна частина нашого додатку.

Серверна частина веб-додатку виконує ключову роль у взаємодії між клієнтом і базою даних. Саме на цьому рівні відбувається обробка запитів, пошук і фільтрація історичних просторових даних. Генерація відповідей у форматах, придатних для візуалізації у нашому випадку — GeoJSON. Також реалізація логіки доступу, кешування та безпеки. Зважаючи на це в даному проєкті було обрано мову Python, що на сьогодні є однією з найпоширеніших мов у галузі розробки веб-сервісів, геоінформаційних систем і наукових обчислень.

Python вирізняється простотою синтаксису, широкою екосистемою бібліотек та активною спільнотою, що дозволяє швидко інтегрувати найкращі практики у власну архітектуру. Зокрема, для побудови веб-сервісів на Python можуть використовуватися легкі фреймворки, такі як Flask або FastAPI, які забезпечують мінімалістичну, але потужну платформу для створення RESTful API. Вони дозволяють швидко налаштовувати обробники HTTP-запитів, реалізовувати маршрути, обробку параметрів URL та взаємодію з базою даних.

Крім фреймворку, у роботі з просторовими даними надзвичайно корисними є бібліотеки GeoPandas, Shapely, GDAL, Fiona, які дозволяють читати, обробляти та трансформувати географічні об'єкти у форматах GeoJSON, Shapefile, WKT. До прикладу Python-скрипт може зчитати полігони історичних кордонів із бази, виконати фільтрацію за часовим періодом, перетворити дані до формату GeoJSON і надіслати їх клієнтській частині для візуалізації на мапі. Вцілому мова добре підходить для побудови допоміжних інструментів — наприклад, для генерації тайлів, попереднього кешування, побудови часових зрізів.

Додатковою перевагою використання даної мови програмування є якісна інтеграція її з PostgreSQL, що у свою чергу дозволяє писати ефективні SQL-запити до просторової бази даних, зокрема до PostGIS. При цьому обробка запитів може бути асинхронною, що позитивно впливає на масштабованість і час відповіді сервера.

Загалом ця мова дуже гарно підходить для виконання даного проєкту. Також її необхідно для початку встановити на ноутбук. По — перше необхідно зайти на головну сторінку як показано на рисунку 2.2 та встановити версію пайтон для віндовс. Нижче зображено

Рисунок 2.2 — Офіційний сайт для завантаження мови пайтон

Після того як ми зайшли на сторінку необхідно натиснути на жовту кнопку завантаження і коли завантажиться пакет необхідно відкрити його .exe файл. Також перед встановленням власне програмного продукту внизу потрібно вибрати галочку напроти — Add Python to PATH, що показано на рисунку 2.3 і вже після цього повністю встановлювати всю програму.

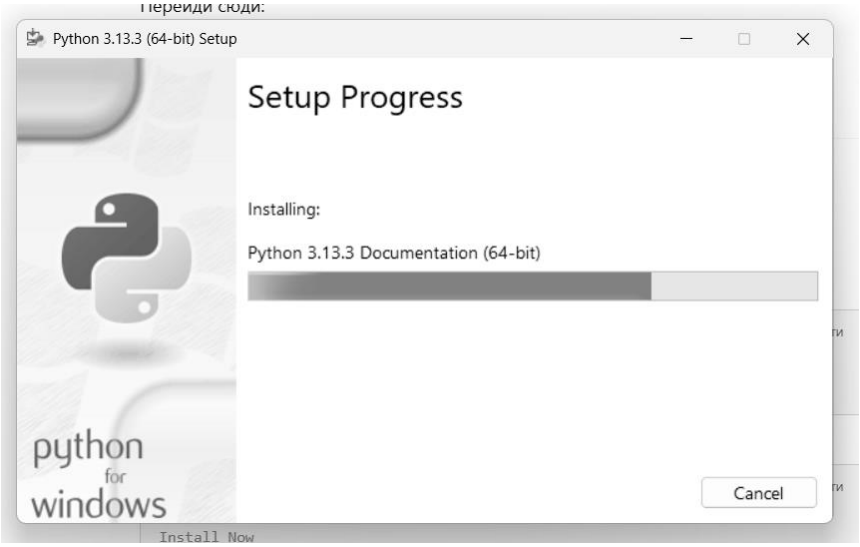

Рисунок 2.3 — Процес встановлення мови пайтон на ноутбук

Окрім цього кроку, нам потрібно встановити розширення для VS Code, як зображено на рисунку 2.4, щоб можна було в ньому програмувати на відповідній мові [9].

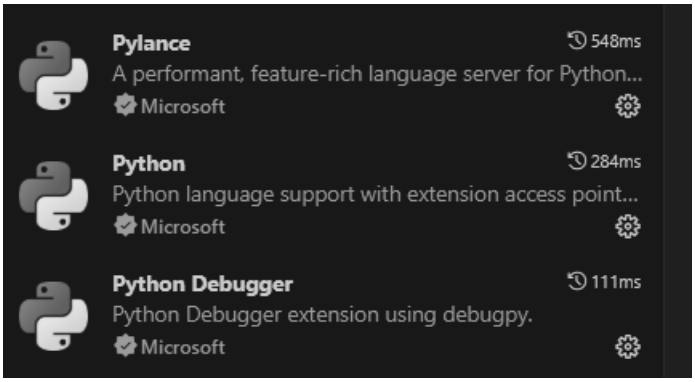

Рисунок 2.4 — Необхідні розширення для серверної частини веб-програми

Pylance — це сучасний Python Language Server від Microsoft, що інтегрується в VS Code через розширення та забезпечує високопродуктивний інтелектуальний досвід кодування. Завдяки Pylance стають доступні функції

рефакторингу, автозаповнення імпортів і детальні повідомлення про помилки на льоту, що значно прискорює писання чистого й безпечного коду.

Python (розширення) — це офіційне розширення Python для VS Code додає основні можливості для автозавершення (IntelliSense), підсвічування синтаксису та управління інтерпретаторами зі списку встановлених версій на наявній системі.

Python Debugger (debugpy) — це розширення що встановлюється разом із Python-розширенням і надає повноцінний відлагоджувач для серверного коду. Розширення дозволяє ставити точки зупину, крокувати по рядках, переглядати змінні та виконувати вирази у вікні Watch під час запуску серверного процесу або під’єднання до віддаленого інтерпретатора.

Після виконання необхідних кроків можна константувати факт, що в наявності є всі основні елементи для налаштування серверної частини програми або бекнеду.

Клієнтська частина веб-додатку реалізується у вигляді класичної HTML-сторінки з елементами JavaScript. Основним завданням цієї частини є візуалізація історичних змін державних кордонів на інтерактивній карті. Для реалізації цієї задачі використовується бібліотека Leaflet, яка дозволяє легко працювати з OpenStreetMap і просторовими даними у форматі GeoJSON.

Вся структура клієнтської частини організована у середовищі розробки Visual Studio Code. Користувач відкриває вебсторінку, бачить карту, поле для введення року, кнопку для завантаження даних та результат у вигляді відображених полігонів на карті. Такий підхід забезпечує мінімалістичний, але функціональний інтерфейс, що не вимагає додаткових дій для ознайомлення.
Карта створюється з фокусом на територію Європи. Базовий тайловий шар підключено із відкритих ресурсів OpenStreetMap, що дозволяє уникнути ліцензійних обмежень та забезпечує достатній рівень деталізації для географічного орієнтування. Завдяки цьому користувач може легко визначити контури держав, міста або природні об’єкти як підкладку під змінні кордони.

При взаємодії користувача з інтерфейсом відбувається зчитування введеного року та надсилання HTTP-запиту до REST API на стороні сервера.

Запит містить рік як параметр і спрямований на отримання просторових даних, що описують державні кордони у відповідному періоді. У відповідь клієнтська частина отримує GeoJSON — поширений формат, який підтримується більшістю геопросторових бібліотек і відображає геометрію полігонів, ліній або точок разом з атрибутивною інформацією.

У разі успішного отримання даних з сервера, попередній шар з картою кордонів видаляється, і новий додається замість нього. Це забезпечує коректне відображення лише актуальної інформації без візуального “нашарування” попередніх об'єктів. Карта автоматично масштабується так, щоб нові полігони були видимими у межах вікна, що підвищує зручність навігації.

У випадках, коли сервер не відповідає або виникає помилка у форматі даних, клієнтська частина передбачає повідомлення про помилку у вигляді спливаючого вікна, вигляд якого продемонстровано на рисунку 2.5 нижче.

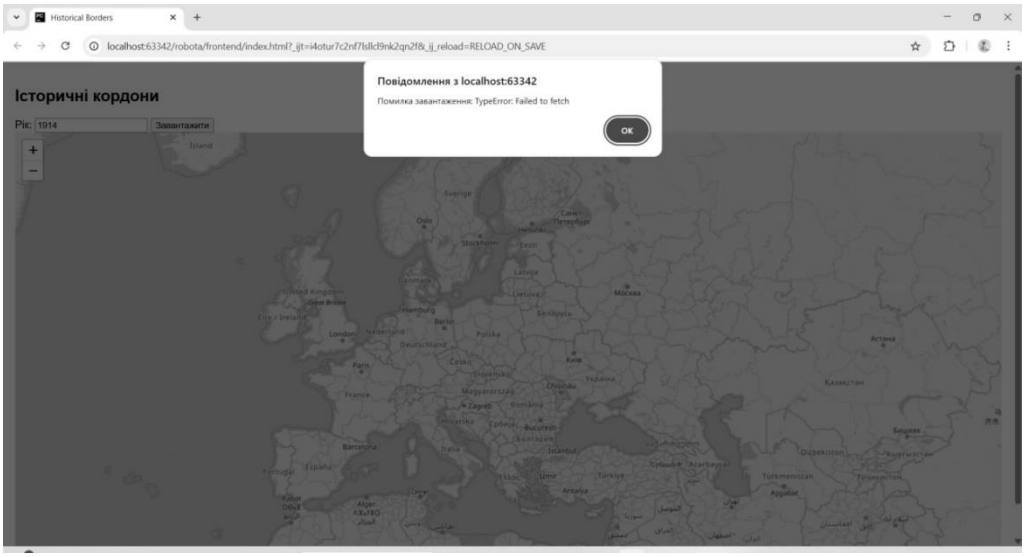

Рисунок 2.5 — Початковий інтерфейс розроблюваного веб додатка

2.3 Серверна частина: обробка запитів та зберігання даних

Серверна частина веб-додатку побудована на базі фреймворку FastAPI[10], який забезпечує ефективну та зручну розробку RESTful API. Основна її функція полягає в обробці запитів від клієнтської частини на отримання історичних кордонів держав за певний рік та поверненні відповідних геоданих у форматі GeoJSON.

Зберігання та обробка геопросторових даних здійснюється за допомогою PostgreSQL з розширенням PostGIS, яке додає підтримку географічних об'єктів. Параметри підключення до бази даних зберігаються в окремому словнику, що містить назву бази даних, ім'я користувача, пароль, хост та порт. Це дозволяє легко змінювати конфігурацію без необхідності редагування основного коду.

Основна функція обробляє GET-запити за маршрутом, де вказується рік, для якого користувач бажає отримати історичні кордони. Функція виконує наступні кроки: підключення до бази даних, виконання параметризованого SQL-запиту для вибору геометрій та назв з таблиці, де дата початку дії кордонів менша або дорівнює 1 січня вказаного року, а дата закінчення — більша або дорівнює 31 грудня того ж року. Отримані результати перетворюються у список об'єктів у форматі GeoJSON, кожен з яких містить тип, властивості (назва держави) та геометрію (координати кордонів). Після обробки результатів з'єднання з базою даних закривається, і функція повертає відповідь у форматі GeoJSON, яка містить тип колекції та список об'єктів. Це дозволяє клієнтській частині легко відобразити кордони на карті за допомогою бібліотеки Leaflet.

У разі виникнення помилки під час підключення до бази даних або виконання запиту, функція обробляє виключення та повертає відповідь з кодом 500 та повідомленням про помилку. Це забезпечує стабільність роботи додатку та інформування користувача про можливі проблеми.

Загалом, серверна частина додатку забезпечує ефективну обробку запитів на отримання історичних кордонів держав. Використання FastAPI дозволяє швидко створювати та масштабувати RESTful API, а PostgreSQL з розширенням PostGIS забезпечує надійне зберігання та обробку геопросторових даних. Завдяки чіткій структурі та обробці помилок, серверна частина є надійною основою для клієнтської частини, яка відображає отримані дані на інтерактивній карті.[11]

2.4 Організація бази даних для зберігання інформації про зміни кордонів

Організація бази даних для зберігання інформації про зміни кордонів є важливим завданням, яке потребує врахування як історичного, так і технічного

аспектів. Така база повинна містити не лише актуальний стан територіальних меж, а й зберігати всю історію їх змін. Для цього необхідно реалізувати систему версійності, тобто кожна зміна кордону має створювати новий запис, що містить дату початку та закінчення дії відповідного кордону. Таким чином, можна відстежити, як змінювалися межі певної території у різні історичні періоди.

Крім цього, доцільно фіксувати тип кожної зміни. Це може бути, наприклад, приєднання нової території, відокремлення певного регіону, об’єднання адміністративних одиниць або зміна назви чи статусу. Такий підхід дозволяє глибше аналізувати не лише самі зміни, а й їхні причини та наслідки.

У базі також важливо зберігати метадані — додаткову інформацію про джерело змін (наприклад, указ або закон), орган, що затвердив ці зміни, а також пояснення або коментарі до кожної зміни. Це надає контекст і підвищує достовірність даних.

Ще одним важливим аспектом є використання геометричних даних, які дозволяють здійснювати просторовий аналіз. Завдяки цьому можна будувати карти змін, визначати території, що мали спільні кордони, або порівнювати площу території до та після змін.

Оскільки така інформація може бути чутливою або мати правове значення, необхідно передбачити систему контролю доступу — визначити, хто має право переглядати, додавати або редагувати дані. Крім того, база має бути масштабованою, адже історичні зміни можуть охоплювати значний часовий період і великий обсяг інформації.

Для ефективного зберігання та обробки інформації про зміни кордонів держав або адміністративних одиниць необхідно організувати спеціалізовану базу даних, яка дозволяє не лише зберігати актуальні дані, а й вести історичний облік усіх змін. Така база повинна містити відомості як про самі географічні межі, так і про часові періоди, протягом яких ці межі залишалися незмінними. Це дозволяє відображати територіальний устрій певної країни або регіону на будь-який обраний момент часу.

Основна мета створення такої бази даних — забезпечити можливість зберігання просторової (геометричної) інформації разом із часовими характеристиками. Це дозволяє виконувати гнучкий аналіз та візуалізацію територіальних змін. Для цього доцільно використовувати сучасні системи управління базами даних, які підтримують просторові типи даних. Одним із найбільш популярних рішень у цій сфері є PostgreSQL з розширенням PostGIS, яке забезпечує зручну роботу з географічними об'єктами, такими як полігони, лінії та точки.

Ключовими елементами такої бази даних є декілька взаємопов’язаних таблиць. Таблиці для зберігання інформації про країни або регіони, у них міститься базова інформація, така як назва, унікальний ідентифікатор, тип адміністративної одиниці, а також додаткові атрибути, що можуть бути важливими для аналізу. У таблицях з геометрією кордонів кожен запис у цій таблиці відповідає окремому варіанту меж певної території у конкретний період часу. Важливо, щоб для кожної геометричної фігури було вказано початкову та кінцеву дату дії — це дозволяє точно визначити, коли саме та чи інша межа була чинною. Розглянемо таблиці для фіксації змін. У цих таблицях фіксуються події, пов’язані зі змінами кордонів — наприклад, розпад країни, об'єднання областей, передача територій. Тут зазначаються дата зміни, її тип (приєднання, поділ, зміна назви), а також супровідні пояснення або посилання на джерела змін, наприклад, офіційні укази чи міжнародні договори.

У структурі бази даних важливо передбачити низку технічних аспектів, які забезпечують правильне функціонування та повноцінне використання інформації. Насамперед необхідно реалізувати ідентифікатори, які дозволяють встановлювати зв’язки між різними таблицями. Завдяки використанню зовнішніх ключів забезпечується цілісність та логічна узгодженість даних, що дозволяє уникати дублювання та помилок під час обробки інформації.

Окрему увагу слід приділити полям, призначеним для зберігання геометричних об'єктів, які відображають форму та просторове розташування кордонів. Для цього застосовуються спеціальні просторові типи даних, наприклад,

`geometry(Polygon, SRID)`. Вони дають змогу точно описати контури територій і працювати з полігональними формами у географічному просторі.

Крім того, до структури таблиць необхідно включити поля для зазначення періоду дії кожного запису. Це, зокрема, поля типу `start_date` і `end_date`, які дозволяють фіксувати часові межі актуальності тієї чи іншої територіальної одиниці або її меж. Завдяки цьому база даних здатна зберігати не лише просторові характеристики, а й історичну динаміку змін.

Така організація дозволяє легко отримувати дані про вигляд кордонів на конкретну дату, виконувати порівняння стану територій у різні періоди, а також аналізувати динаміку територіальних змін. Наприклад, можна вивести карту світу станом на 1991 рік, а потім — на 1992-й і побачити, як розпад СРСР змінив політичну карту регіону. Завдяки використанню геоінформаційної бази даних, також з'являється можливість інтеграції з системами візуалізації — наприклад, за допомогою Leaflet або OpenLayers. Це дозволяє будувати інтерактивні карти, на яких користувач може обирати дату або період часу, а система автоматично відображатиме відповідний варіант меж[12].

Таким чином, правильно спроєктована база даних для зберігання інформації про зміни кордонів — це потужний інструмент для науковців, аналітиків, працівників державних установ, істориків і географів. Вона не лише фіксує минуле, а й дозволяє краще розуміти динаміку змін у геополітичному просторі.

У підсумку, грамотно побудована база даних для зберігання змін кордонів є не лише сховищем фактів, а й потужним інструментом для аналізу територіальних трансформацій у різні періоди історії, який стане в нагоді дослідникам, аналітикам, науковцям і державним установам.

2.5 Методи візуалізації змін кордонів на інтерактивних картах

Візуалізація змін кордонів є важливою складовою аналізу геополітичних процесів, історичних подій, адміністративних реформ та багатьох інших сфер. За допомогою інтерактивних карт можна не лише демонструвати сучасний стан меж,

але й відтворювати їхню динаміку у минулому, що робить такі інструменти незамінними для дослідників, освітян та аналітиків.

Основна перевага інтерактивних карт — це можливість взаємодії користувача з даними. На відміну від статичних зображень, користувач може змінювати масштаб, пересувати карту, обирати часові інтервали, переглядати додаткову інформацію про конкретні регіони. Це забезпечує гнучке налаштування під конкретні запити або завдання дослідження.

Одним із поширених методів є фільтрація за часовим інтервалом. Візуалізація за допомогою такого підходу дає змогу показати лише ті кордони, які були чинними у вибраний період. Наприклад, користувач може обрати рік 1914 і побачити геополітичну ситуацію на той момент, а потім змінити дату на 1945 — і одразу побачити відповідні зміни.

Іншим підходом є анімаційна візуалізація, яка дозволяє програвати послідовність змін меж як відео. Такий підхід дозволяє швидко оцінити динаміку змін протягом обраного періоду. Анімація може бути реалізована через таймер або за допомогою елемента "слайдер часу", де користувач вручну рухає повзунок по часовій шкалі.

Особливу увагу слід приділити методам візуального підкреслення змін. Наприклад, території, що змінили приналежність або були приєднані до інших регіонів, можуть автоматично виділятися іншим кольором, обводкою або напівпрозорістю. Це дозволяє зосередити увагу саме на зонах змін, не відволікаючи на інші об'єкти карти.

Крім змін просторових об'єктів, важливою є інформаційна складова. Інтерактивна карта може включати спливаючі вікна (pop-ups) з детальними поясненнями змін: дати прийняття рішень, законодавчі документи, історичні події, які призвели до зміни меж. Це поглиблює розуміння причин і наслідків територіальних змін.

Для ефективної реалізації візуалізації змін меж використовуються сучасні геоінформаційні технології. Найпоширенішими бібліотеками для веб-візуалізації є Leaflet.js, Mapbox GL JS, D3.js та OpenLayers. Вони дозволяють створювати

інтерактивні карти, завантажувати геодані з серверів, налаштовувати події взаємодії з користувачем, працювати з шарами та анімацією.

Карта може базуватись на векторних або растрових підкладках. Векторні підходи (як-от Mapbox Vector Tiles або GeoJSON-дані) дають змогу динамічно змінювати стиль, кольори, товщину ліній, підписів. Це дозволяє легко оновлювати вигляд карти без перезавантаження сторінки [13].

Інтеграція бази даних (наприклад, PostgreSQL з PostGIS) із фронтендом здійснюється через API або спеціальні сервіси, які обробляють запити до бази на основі вибраної дати чи іншого параметра. Це забезпечує динамічне оновлення геоінформації без необхідності повного перезавантаження карти.

Особливо перспективним напрямком є використання технологій тривимірної (3D) візуалізації та WebGL, що дозволяє створювати об'ємні моделі територій, показувати зміни на різних рівнях (національному, регіональному, місцевому) та відображати додаткові параметри, наприклад, кількість населення або площу, яка змінилася.

Крім того, розробники можуть включити історичні карти як фонові зображення, на які накладаються сучасні цифрові контури меж. Це дозволяє порівнювати історичну картографію з актуальними геопросторовими даними, що є цінним інструментом у краєзнавстві та історичних дослідженнях.

Сучасні візуалізації також підтримують мобільні пристрої, що дає змогу переглядати зміни кордонів у будь-якому місці та в будь-який час. Адаптивний дизайн і оптимізація для мобільних браузерів дозволяє залучати широку аудиторію користувачів.

Важливим аспектом є також юзабіліті та дизайн карти. Інтерфейс повинен бути інтуїтивно зрозумілим, доступним, із чіткими кнопками, легендою, підказками. Особливо це важливо для освітніх та публічних проєктів, де користувачами можуть бути школярі, студенти, громадські активісти.

Нарешті, інтерактивні карти змін меж можна інтегрувати в більші інформаційні системи, зокрема в портали відкритих даних, історичні архіви, наукові платформи. Це дозволяє поєднувати картографічну візуалізацію з

документами, аналітичними таблицями, фото- та відеоматеріалами, створюючи повноцінне інформаційне середовище.

Таким чином, візуалізація змін кордонів на інтерактивних картах — це не лише технічне, але й аналітичне завдання, яке потребує комплексного підходу: правильного структурування даних, використання сучасних технологій, врахування потреб цільової аудиторії та забезпечення високої якості подачі інформації.

2.5.1 Використання геопросторових бібліотек

Для реалізації візуалізації змін кордонів на інтерактивних картах важливу роль відіграють геопросторові бібліотеки. Вони надають інструменти для роботи з просторовими даними, їх обробки, візуалізації та взаємодії з користувачем.

Однією з найпопулярніших бібліотек є Leaflet.js. Вона легка, зручна для використання у веб-додатках і підтримує інтеграцію з GeoJSON, що робить її ідеальною для виведення меж адміністративних одиниць. Leaflet дозволяє додавати шари, налаштовувати стилі, працювати з подіями та створювати часові слайдери для перегляду змін меж у часі.

Інша потужна платформа — Mapbox GL JS, яка підтримує векторну візуалізацію на основі WebGL. Це забезпечує високу продуктивність, можливість деталізованої стилізації об’єктів та інтерактивної роботи з великими наборами просторових даних. За допомогою Mapbox можна реалізовувати складні сценарії анімації змін меж.

OpenLayers — ще одна бібліотека з потужним інструментарієм для роботи з картографією. Вона підтримує різноманітні формати геоданих (GeoJSON, KML, GML), дозволяє завантажувати тайли, працювати з проекціями та просторовими запитами, що особливо важливо при візуалізації змін, які залежать від конкретного регіону чи дати.

D3.js також може використовуватись для візуалізації просторових змін, особливо у поєднанні з часовими шкалами та анімацією. D3 забезпечує гнучке налаштування графіки та дозволяє будувати кастомні візуалізації змін меж у

вигляді ліній часу, діаграм або комбінованих карт.

Використання геопросторових бібліотек часто поєднується з серверними рішеннями — зокрема, PostGIS (розширення до PostgreSQL), яке дозволяє виконувати складні геопросторові запити та передавати результат у форматі GeoJSON для подальшої візуалізації на фронтенді. Таким чином, вибір конкретної бібліотеки залежить від поставлених цілей, складності задачі, обсягу даних, вимог до продуктивності та інтерактивності.

2.5.2 Алгоритми відображення часових змін

Візуалізація змін у часі передбачає не лише відображення геометрії, але й реалізацію алгоритмів, які враховують часові параметри даних. Основна мета таких алгоритмів — відобразити, як змінювались кордони у різні періоди, та забезпечити користувачу зручні інструменти для навігації в часі.

Найпростішим підходом є фільтрація даних за часовими мітками. У цьому випадку база даних або API передає лише ті геометрії, які мають діючу межу для вибраної дати. Це дозволяє швидко будувати карту на конкретну історичну дату.

Більш складний підхід — реалізація анімації змін меж у часі. У цьому випадку для кожної межі зберігаються часові межі її дії (start_date та end_date). Алгоритм анімації оновлює відображення карти по кадрах, змінюючи геометрію відповідно до зміни часу. Це дозволяє побачити, як змінювалась територія з року в рік.

Для побудови історичної лінії часу часто використовується так званий "time slider" — інтерфейсний елемент, що дозволяє вибрати конкретну дату або перетягувати повзунок і бачити зміну меж у реальному часі. У поєднанні з алгоритмами попередньої обробки даних це дозволяє забезпечити плавність і точність відображення.

Ще одним методом є накладання кількох меж одночасно, наприклад, шляхом використання різних кольорів, прозорості або стилів ліній. Це дозволяє порівняти стан меж у кількох часових точках, що зручно для аналітичного аналізу.

Алгоритми також повинні враховувати ситуації, коли межа перетворюється, розділяється або зникає. У таких випадках важливо мати механізми обробки змін типу “split”, “merge” або “dissolve”, а також можливість відобразити динаміку змін у структурі політичних або адміністративних утворень.

У реалізації часових алгоритмів важливим є ефективне зберігання та кешування геоданих, щоб уникнути навантаження на сервер та забезпечити швидку реакцію інтерфейсу при зміні дати.

Важливо також враховувати візуальну відповідність між змінами: плавні переходи між межами, анімаційні ефекти, згасання старих кордонів і поява нових допомагають краще сприймати динаміку змін.

Отже, ефективна візуалізація часових змін на карті базується на поєднанні геоалгоритмів, інтерфейсних рішень та якісно структурованих даних у базі. Це дозволяє не лише показати, як змінювались межі, але й пояснити причини цих змін через взаємодію з користувачем.

3 РЕАЛІЗАЦІЯ ВЕБ-ДОДАТКУ

3.1 Налаштування середовища розробки

Загалом розробка такого проєкту розпочинається зі встановлення робочого середовища для написання коду. На момент отримання завдання з бакалаврської роботи вже було встановлено таке середовище як vs code. Власне маючи таке потужне та гнучке середовище для розробки коду і було вибрано таку тему проєкту для бакалаврської кваліфікаційної роботи.

Щодо встановлення такого IDE. По - перше необхідно перейти на на офіційний сайт VS Code, який зображений на рисунку 3.1 нижче, та завантажити інсталятор для відповідної операційної системи , у нашому випадку — Windows.

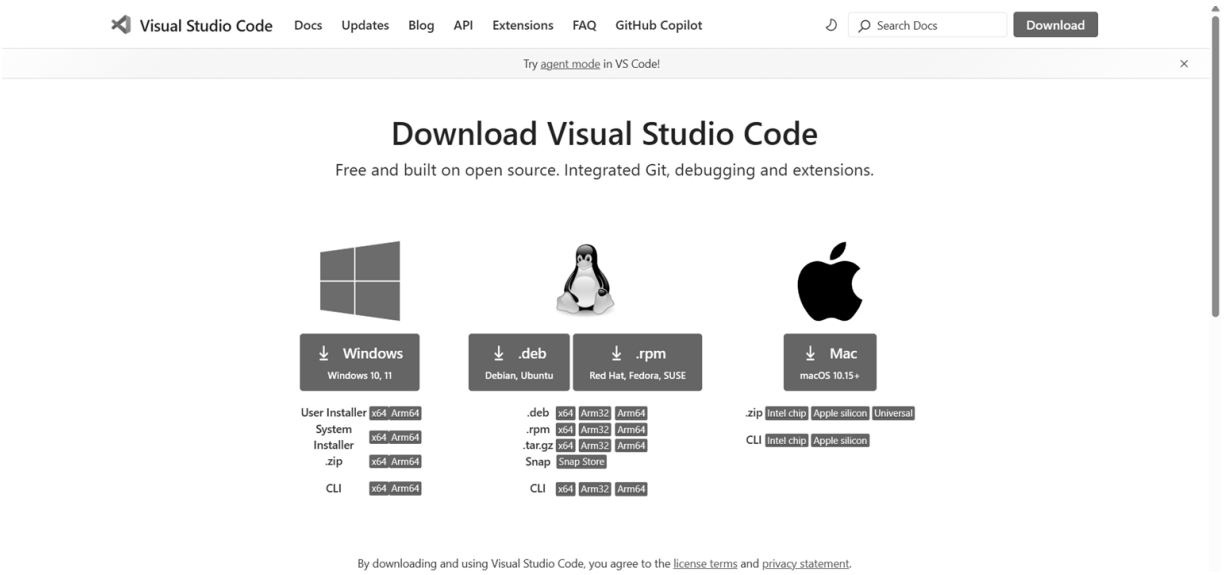

Рисунок 3.1 — Вигляд офіційного сайту для встановлення visual studio code

Після цього необхідно натиснути на пакет коли він завантажиться та погодитися на всі необхідні вказівки до моменту повного встановлення даного програмного середовища. В кінці даного процесу vs code буде доступний для користування, а його інтерфейс буде виглядати приблизно так як вказано нижче:

Як можна побачити на рисунку доступний ряд функцій, зокрема:

— створити новий файл;

— відкрити існуючий файл(який вже є на диску);

— відкрии існуючу папку;

— підключитися до — дозволяє підключатися до віддалених серверів, git-репозиторіїв;

— ввімкнути штучний інтелект від copilot.

Нижче вказано проекти які розроблялися на даному IDE, та їх місцезнаходження на ЕОМ, наприклад, на ноутбуці, на комп'ютері. З лівого боку також буде доступна необхідна колонка для різноманітних додатків, частину з яких вже завантажили. Даний редактор має автоматичну підтримку ряду мов програмування таких як Javascript, HTML, які нам допоможуть у розробці даної програми.

Після встановлення даної програми ми можемо переходити до створення власне самого кістятка проєкту. Для початку нам необхідно створити папку на комп’ютері і назвати її наприклад, robota. У папці robota в свою чергу ми створемо 3 папки і відповідно їх та назвемо — frontend, backend та data. Папка з назвою фронтенд відповідатиме за клієнтську частину проекту. Друга за серверну, а в третій будуть міститися необхідні координати для візуалізації.

Далі нам необхідно створити необхідні файли для кожної з папок де буде знаходитися наший проєкт. Для цього ми відкриваємо встановлений редактор та імпортуємо наший проєкт. У vs code він матиме такий вигляд:

Буква С, яка знаходиться коло папок на рисунку 3.2 була спеціально прописано мною, щоб показати як все відбувається на початкових етапах. Бо у вкладці recent ви можете побачити проект з папкою robota на диску D, яка і є справжнім веб - додатком, але щоб не зачіпати той проект, я вирішив показати як я виконував його на початку коли тільки починав його робити.

Коли вже було виконано початковий етап ми можемо розпочати процес створення необхідних програмних файлів для нашого проекту, де буде міститися код програми. Для цього ми заходимо на будь-яку з 3 папок і починаємо там створювати файл. Спочатку ми на нього натискаємо, щоб він підсвітився синім кольором. З рисунку 3.2 видно, що навпроти головного файлу robota видно 4 іконки. Необхідно натиснути на першу з них, яка підсвічуватиметься як New File. Це дозволить нам створити необхідний файл для коду. Коли ми на нього

натиснули під пакою утвориться пустий файл, якому необхідно надати ім'я і потім через крапку написати формат файлу доступний для тої чи іншої мови. Загалом прописати назву необхідно за такою структурою — ім'я.мова програмування. Для початку нехай це буде папка під назвою backend. Дана папка відповідатиме за серверну частину програми. Вона буде прописана на мові пайтон. Отже дамо файлу таку назву — main.py. Поки що цього буде достатньо. Також необхідно проробити це все ще з іншими папками. Кінцевий вигляд проєкту приблизно буде такий:

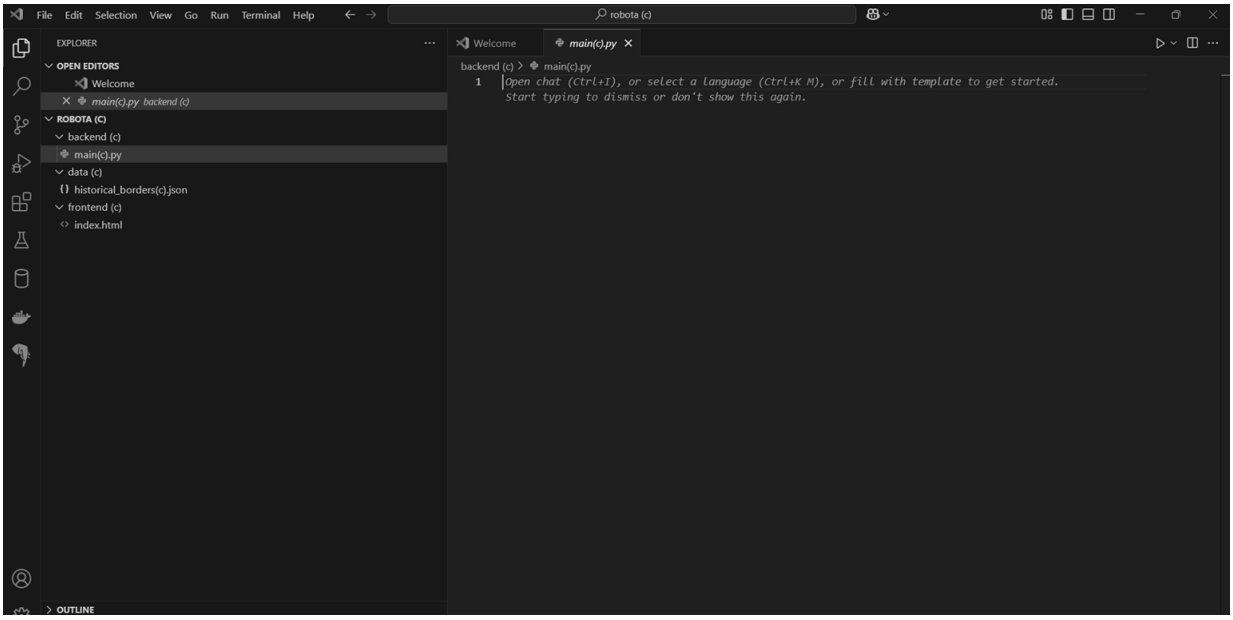

Рисунок 3.2 — Модернізований початковий вигляд структури додатку

Також після цього необхідно буде ще прописати requiments.txt. У двох стрічках окремо написати uvicorn та fastapi. Це знадобиться потім для налаштування мови пайтон для того щоб ефективно працювала клієнт-серверна архітектура.

На даний момент у нас вже необхідне програмне забезпечення для розробки, фронтенду та бекенду, оскільки ми його встановлювали раніше. Залишилося завантажити необхідні бази даних для географічних координат.

Для розробки роботи було обрано базу даних PostgreSQL найновішої версії, разом із розширенням PostGIS. PostgreSQL необхідно встановити першою. Для цього потрібно зайти на офіційний сайт на завантажити пакет інсталятора та

встановити дане середовище на ноутбук. Під час процесу встановлення необхідно слідувати всім наявним вказівкам. Зокрема можна обрати місце на пристрої куди встановлювати БД. У моєму випадку це диск D, що важливо в цьому випадку необхідно правильно прописати шлях та кінцеву папку, щоб потім не було конфлікту сертифікатів, коли буде відбуватися процес інтеграції PostgreSQL з vs code. Нижче вказано як приклад, одну з помилок в командному рядку ОС Windows: ERROR: Could not install packages due to an OSError: Could not find a suitable TLS CA certificate bundle, invalid path: D:\PostgreSQL17\ssl\certs\ca-bundle.crt.

Також необхідним буде, визначення правильної локалі для Бд у моєму випадку — це en_US.UTF-8, який підтримує всі символи Unicode. Та необхідно правильно створити собі пароль. У разі якщо пароль загубився, то доведеться заново все перевстановлювати. Вкінці, можуть бути проблеми з антивірусом, для цього необхідно буде додати програму у білий список. В іншому випадку можуть трапитися різні проблеми з PostgreSQL.

Після успіху з PostgreSQL[14] , необхідно переходити до інсталяції PostGIS. Процес потрібно виконувати через Stack Builder зображеному на рисунку 3.3 в завантаженому PostgreSQL, як зображено на рисунку нижче:

Рисунок 3.3 — Місцезнаходження Stack Builder

Процес встановлення подібний як і до інших програм, головне правильно прописати шлях. Необхідністю встановлення такого додатку є можливість підтримання географічних типів, полігонів, у основній БД.

Третьою програмою для інсталяції, буде QGIS вона необхідна для того аби завантажувати дані у форматі JSON з vs code, та імпортувати їх у PostgreSQL. Для цього необхідно зайти на офіційну сторінку програми показану на рисунку 3.4 і завантажити standalone версію, щоб не встановлювати її через пакет. Вона буде довго завантажуватися, однак це мінімізує потенційні ризики конфліктів з іншими програмними середовищами.

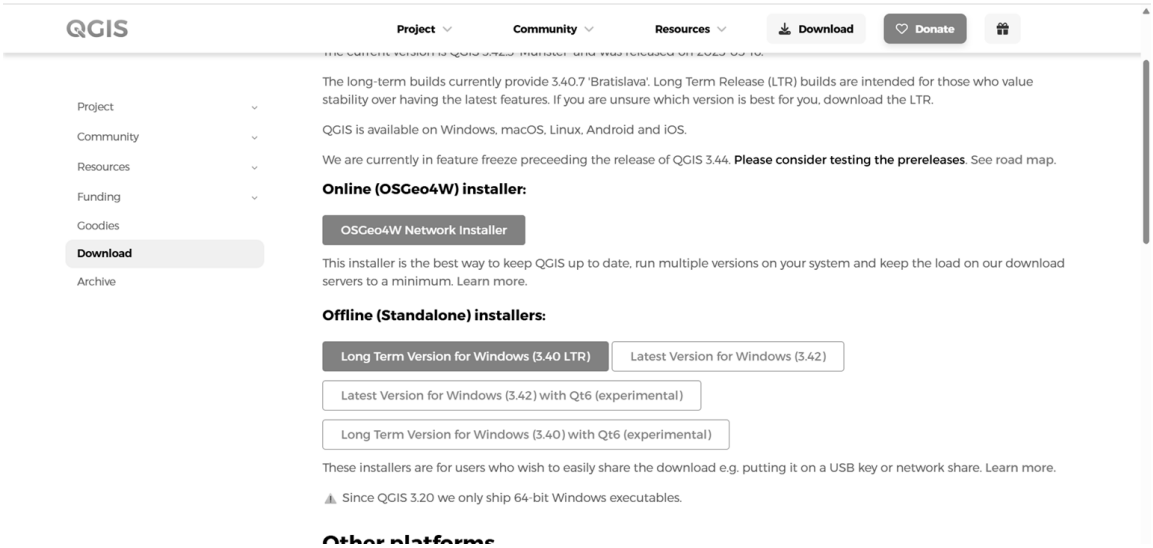

Рисунок 3.4 — Офіційна сторінка для завантаження QGIS

Загалом, було встановлено все необхідне програмне забезпечення, що і продемонстровано на рисунку 3.5.

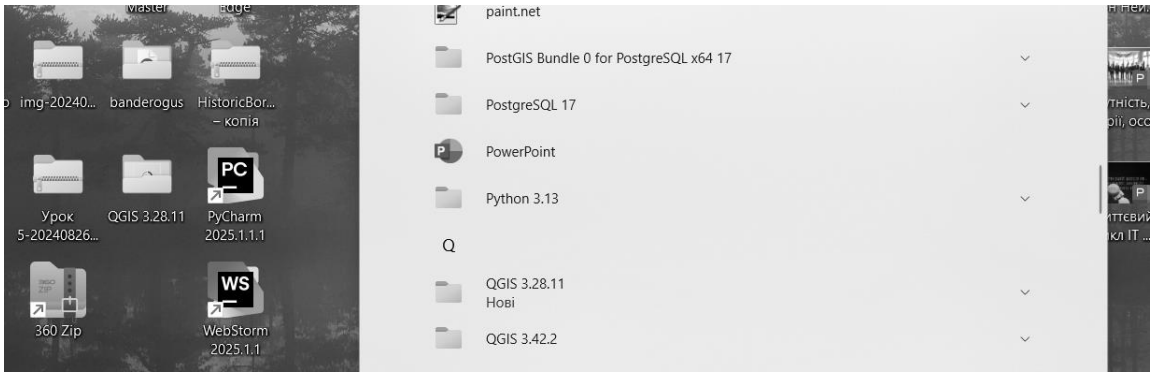

Рисунок 3.5 — Інстальоване програмне забезпечення для БД

3.2 Розробка серверної частини

Серверна частина веб-додатку реалізована за допомогою мови програмування Python із використанням веб-фреймворку FastAPI. Основним завданням серверної частини є обробка запитів від клієнта, надання геопросторових даних у форматі GeoJSON, а також інтеграція з базою даних для зберігання інформації про історичні зміни державних кордонів. У свою чергу Сервер відповідає за обробку HTTP-запитів, підключення до бази даних PostgreSQL, вибірку просторових даних, їх форматування у формат GeoJSON та передачу клієнтській частині.

Почати розробку серверного коду потрібно з імпорту необхідних бібліотек для роботи з БД та клієнтським сервером. Прописувати код необхідно у файлі main.py. Імпорт всіх необхідних бібліотек буде виглядати як на лістингу 3.1:

Лістинг 3.1 — імпорт фреймворку FastApi

```
from fastapi import FastAPI
from fastapi.middleware.cors import CORSMiddleware
from fastapi.responses import JSONResponse
import psycopg2
import json
```

Компонент FastAPI являє собою основу для створення та управління RESTful API. У наведеному прикладі створюється екземпляр застосунку app, до якого додається middleware CORS. Додавання CORSMiddleware дозволяє обробляти запити з будь-якого джерела, що може бути особливо корисно під час розробки або при роботі з фронтенд-клієнтами, які розміщені на інших доменах. Дані налаштування дозволяють уникнути типових проблем, пов’язаних із політикою крос-доменного доступу, сприяючи більш злагодженій інтеграції між різними компонентами системи, даний лістинг коду показано в лістингу 3.2.

Лістинг 3.2 — Додавання CORSMiddleware

```
app = FastAPI()
app.add_middleware(
   CORSMiddleware,
   allow_origins=["*"],
```

```
    allow_methods=["*"],
    allow_headers=["*"],
)
```

Після імпорту бібліотек необхідно здійснити ініціалізацію FastAPI(верхня стрічка коду) та CORS. дозволяє обробляти HTTP-запити з будь-яких джерел. У даній програмі така необхідність полягає в тому що, клієнт і сервер розміщені на різних портах. Щодо папраметрів підключення до бази, то вони зберігаються у вигляді словника показаному в лістингу 3.3:

Лістинг 3.3 — додавання словника

```
DB_CONFIG = {
    "dbname": "borders_proj",
    "user": "postgres",
    "password": "Mereth@12#",
    "host": "localhost",
    "port": 5432,
}
```

На даному етапі можливо переходити до створення ендпоінтів як на лістингу 3.4 :

Лістинг 3.4 — створення ендпоінтів

```
@app.get("/borders/{year}")
def get_borders(year: int):
    try:
            conn = psycopg2.connect(**DB_CONFIG)
        cur = conn.cursor()

         query = """
            SELECT ST_AsGeoJSON(geom), name FROM historical_borders
            WHERE valid_from <= %s AND valid_to >= %s
        """
        cur.execute(query, (f"{year}-01-01", f"{year}-12-31"))
```

Дана частина коду реалізує надсилання запитів та підключення до PostgreSQL. Також обирає межі країн для обраного року та забезпечує захист від SQL - ін'єкцій.

Далі йде частина коду яка відповідає за власне обробку геоданих, закриття БД, обробку вийнятків, що не вилітав браузер, при неможливості отримати з бази даних інформацію.

Наступним кроком буде налаштування середовища для пайтон через утиліти терміналу або командного рядка Windows. Річ у тім що ми маємо справу з досить складно організованим проектом і тому потрібно багато чого налаштовувати, щоб все працювало.

Після написання коду буде ряд помилок які пов’язані з неможливістю імпорту ряду даних. Зокрема, власне пайтон, uvicorn, fastapi та psycopg2. Для початку необхідно встановити віртуальне середовище для пайтон. У файлі main.py у терміналі прописуємо таку команду python -m venv venv. Необхідно прописувати це у тому файлі, адже встановлення такого середовища, насправді починається з диску де знаходиться програма. У моєму випадку це диск D. отже сформується приблизний шлях D:\robota\backend, де і необхідно створити віртуальне середовище. Хоча в нас вже є встановлений пайтон, але він знаходиться глобально на всій системі, що може змусити його не активуватися через багато інших програм на ноутбуці та через ті ж БД. Коли налаштували віртуальне середовище, можна командою - pip встановити спочатку версію пайтон, а потім і інші додатки. Подібною командою необхідно завантажити додаткові компоненти для цього використаємо такі команди показані в лістингу 3.5:

Лістинг 3.5 — консольні команди для інсталяції необхідних пакетів для мови python

```
pip install fastapi
pip install uvicorn
uvicorn main:app --reload
pip install psycopg2
```

Всі, окрім 3 команди встановлюють необхідні доповнення для можливості доступу програми до клієнтської частини серевера, та бази даних. 3 Команда автоматично перезавантажувати застосунок при внесенні змін до коду, точніше за перезавантажування відповідає утиліта --reload, а застосунком є main:app, для

цього і було прописано в коді: app = FastAPI()[15]. Коли помилок не має, то серверна частина налаштована. Загалом папка backend матиме вигляд, зображений на рисунку 3.6:

Рисунок 3.6 — Фінальний вигляд архітектури сервера

3.3 Інтеграція з базою даних

Тепер здійснивши налаштування серверної частини програми, можна переходити до детальної інтеграції PostreSQL з середовищем розробки vs code. Для початку необхідно наповнити БД необхідними координатами. Координати будемо використовувати з інтернету. Точніше шукати будемо на сайті Historical Country Boundaries[16], де в хедері справа будуть координати у різних форматах, у даному випадку обираємо GeoJSON. Після того як відкриється сторінка можна буде побачити дуже великий та неохайний набір різних координат. Необхідно натиснути на автоматичне форматування аби код координат отримав зрозумілий вигляд, як на лістингу 3.6.

Лістинг 3.5 — форматований вигляд координат


```
{
 "type": "FeatureCollection",
 "numberReturned": 10,
 "numberMatched": 710,
 "timeStamp": "2025-06-09T08:58:17Z",
 "features": [
  {
   "type": "Feature",
   "id": 81,
   "geometry": {
    "type": "MultiPolygon",
    "coordinates": [
```

```
[
  [
    [9.5721124, 47.5389909],
    [9.4783216, 47.5758517],
    [9.2610956, 47.6628422],
    [9.0826376, 47.6854139],
    [9.0568986, 47.6869368],
    [9.0333354, 47.6883281],
    [9.0008354, 47.6808273],
    [8.9794446, 47.670136],
    [8.947774, 47.6574917],
    [8.9259727, 47.6518006],
```

Дані координати ми будемо використовувати для подальшої візуалізації історичних держав. Проте для цього необхідно буде створити папку формату .geojson, для імпорту цих координат. У архітектурі розроблюваного додатку передбачена папка data, де буде створений файл із подальшим імпортом координат, що зображено на рисунку 3.7:

Рисунок 3.7 — Створення файлу для збереження початкових даних

Файл з координатами буде початковим етапом для подальшого передання географічних даних у БД, і перед тим як здійснювати таку необхідно для початку ввести в даний файл потрібні координати та написати простий код до кожного кластера з координатами, щоб у кінці ми отримали правильну таблицю в БД.

Переглянувши всі доступні координати, було вирішено як для початкової програми, вибрати координати для Німеччини до першої світової та після першої світової війни. Оскільки, нам власне потрібні координати, то буде достатньо задати, масив координат, форму держави, та часові рамки для кожної з держав, код для першої держави буде починатися як у лістингу 3.6:

Лістинг 3.6 — код координат GeoJSON для кайзерівської Німеччини

```
{
  "type": "FeatureCollection",
  "features": [
```

```
{
 "type": "Feature",
 "properties": {
  "name": "Test Country 1914",
  "valid_from": "1886-01-01",
  "valid_to": "1914-12-31"
 },
 "geometry": {
  "type": "MultiPolygon",
  "coordinates": [
   [
    [
     [
      12.53337,
      54.4669495
     ],
     [
      12.5269447,
      54.4741629
     ],
     [
      12.4805543,
      54.4441632
     ],
     [
      12.4369446,
      54.3911083
     ],
     [
      12.4369381,
      54.3911006
     ],
     [
```


Де, FeatureCollection — це буде контейненром, для всіх координат(features) усіх можливих держав. Наступна частина відповідає за характеристику одного з наявних об’єктів, зокрема назва test country 1914 (хоча то Німеччина, до WWI). Наступна частина відповідає за геометрію, зокрема промальовування полігонів(ліній) на основі координат. Але оскільки, країна має дуже складні кордони з кількома анклавами, то було вибрано "type": "MultiPolygon", оскільки це дозволить якісніше зообразити кордони. Хоча з іншого боку доведеться в

наявні координати, додати ще по додатковій квадратній дужці, оскільки даний геометричний тип цього потребує. Подібна ж структура буде і для кордонів Німеччинни періоду після першої світової війни зображено у лістингу 3.7.

Лістинг 3.7 — код координат GeoJSON для міжвоєнної Німеччини

```
{
    "type": "Feature",
    "properties": {
     "name": "Test Country 1920",
     "valid_from": "1920-01-01",
     "valid_to": "1939-12-31"
    },
    "geometry": {
     "type": "MultiPolygon",
     "coordinates": [
       [
         [
           [
             19.2533272,
             54.2781911
           ],
           [
             19.2307632,
             54.3336731
           ],
           [
             19.2776362,
             54.3461091
           ],
```

Закінчується цей код із використанням вкладених дужок, який повністю закриває всі контейнери, вигляд яких подано у лістингу 3.8.

Лістинг 3.8 — завершення коду з координатами

```
[
             9.4147239,
             54.8333274
           ],
           [
             9.4453557,
             54.825402
           ],
           [
```

```
            9.4467521,
            54.8271448
           ],
           [
            9.4147239,
            54.8333274
           ]
          ]
         ]
        ]
       }
      }
     ]
    }
```


Загальний файл дуже великий, з приблизно десятком тисяч координат(приблизне число якщо виключити дужки/код), і це лише для 2 держав. Така велика кількість дозволить побудувати державу з високоякісними кордонами. Проте ми завершили лише оформлення даних.

Наступним кроком буде інтеграція даних з БД. Фактично це єдиний вихід як якісно працювати з такою великою кількістю цифрових даних. Для початку ми імпотуємо файл у QGIS. Це потім дозволить його якісно імпортувати в PostrgeSQL. Отже, відкриваємо програму та створюємо там проект, як зображено на рисунку 3.8 Коли ми його створимо його необхідно в правильному форматі зберегти.

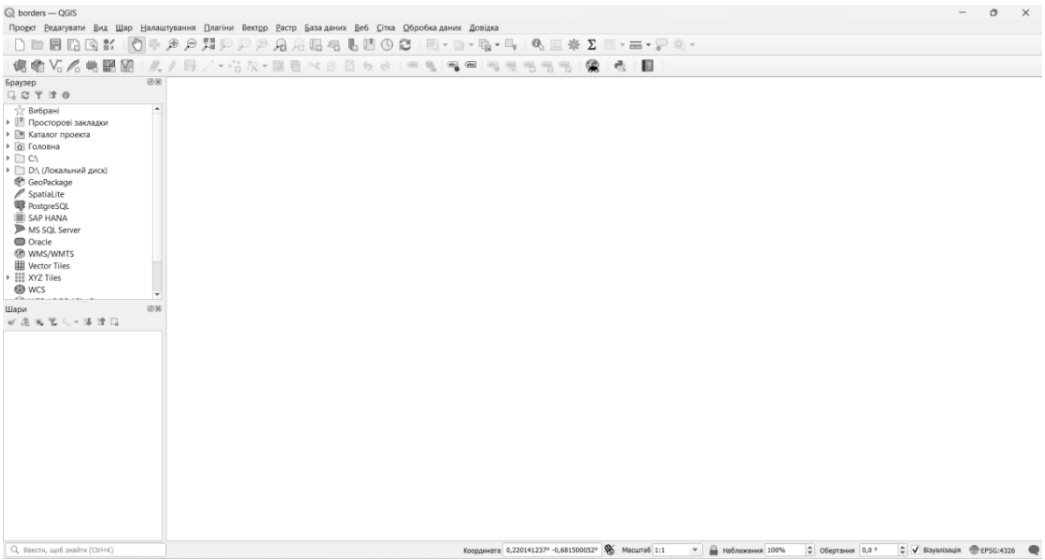

Рисунок 3.8 — Відкритий пустий проект в QGIS

Далі нам необхідно до нього підключити необхідні дані, для цього ми відкриваємо шар, Додати векторний шар і таким чином повинна появитися інтерфейс подібний до інтерфейсу на рисунку 3.9.

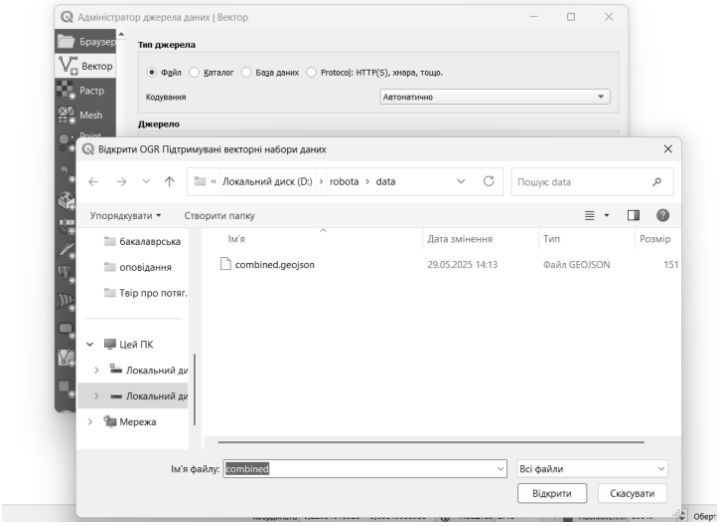

Рисунок 3.9 — Імпорт координат в QGIS

Точніше потрібно обрати шлях, звідки необхідно імпортувати отримані дані. В моєму випадку диск D, проект/папка проекту. Імпотруємо необхідний формат файлу і отримуємо такий результат зображений на рисунку 3.10.

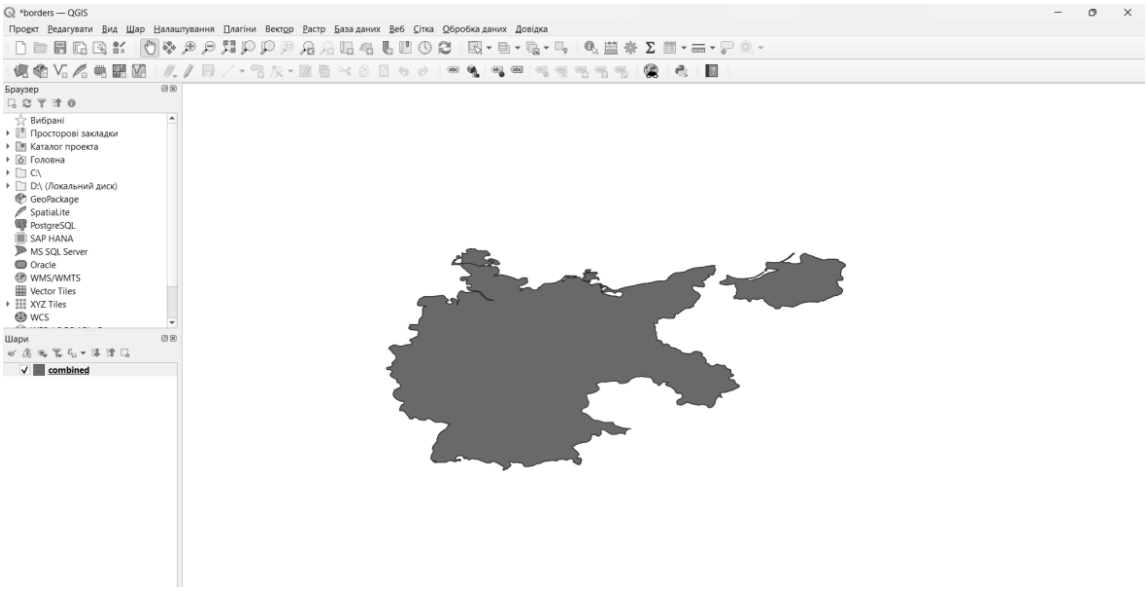

Рисунок 3.10 — Результат візуалізації координат

На рисунку зображено Німеччину, точніше дані кордони є суммішю обох з них. Тим не менше завдання з пошуку координат і початкові дії є виконаними[17].

Наступним етапом буде створенння даної таблиці з координатами в PostgreSQL. Найефективнішим способом буде використання командного рядка. Можна звісно і через програму, однак її функції часто не спрацьовують коректно і виникає конфліти імпорту. Тож імпортуємо дані в таблицю у БД через cmd, як на рисунку 3.11.

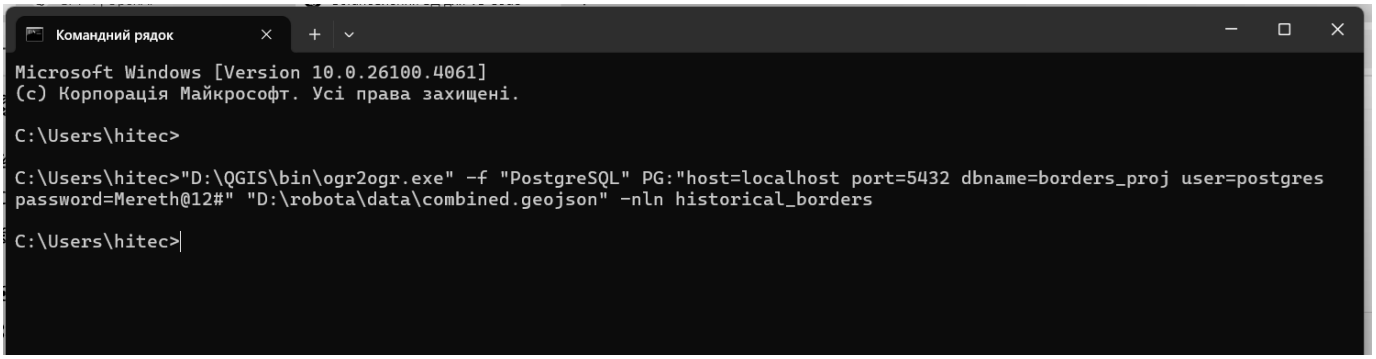

Рисунок 3.11 — Імпорт даних з QGIS в PostgreSQL

Тут ми використовуємо досить комплексну команду, основою якого буде утиліта ogr2ogr[18]. ogr2ogr — це утиліта з бібліотеки GDAL, яка використовується для конвертації та трансформації векторних геоданих між різними форматами. Функціонал даної програми є дуже широкий, але в цьому випадку необхідно конвертувати векторні величини Зокрема через ogr2ogr ми інтегруємо GeoJSON в PostgreSQL. Отже:

— початкова частина шляху "D:\QGIS\bin\ogr2ogr.exe" вказує на точний шлях утиліти;

— -f "PostgreSQL" вказує на формат призначення, точніше куди будуть спрямовані дані.

— PG:"host=localhost port=5432 dbname=borders_proj user=postgres password=Mereth@12#" це у нас параметри підключення до бази, вони були створені ще ще під час встановлення PostgreSQL.

— "D:\robota\data\combined.geojson" тут вказаний шлях, де шукати файл з даними.

— -nln historical_borders та створена таблиця необхідна для подальшої роботи.

Результатом цих маніпуляцій має стати новостворена таблиця в базі даних. Зайшовши до неї через PgAdmin 4, та натиснувши View all rows можна буде побачити таблицю та простий код для її виклику поданого на рисунку 3.12.

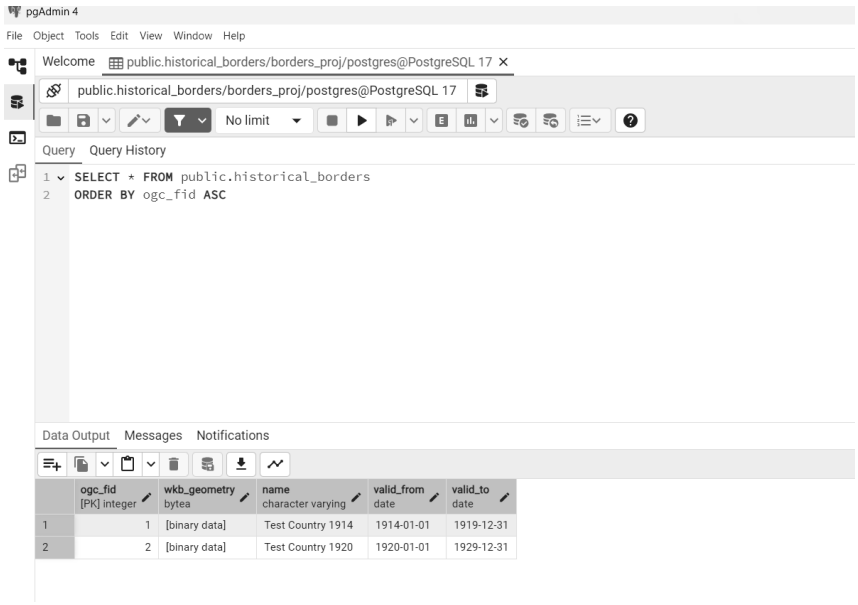

Рисунок 3.12 — Таблиця з координатами в PostgreSQL

Як бачимо в коді таблиці historical_borders на рисунку 3.13 , видно дану таблицю. Також в розділі SQL можна побачити такий код:

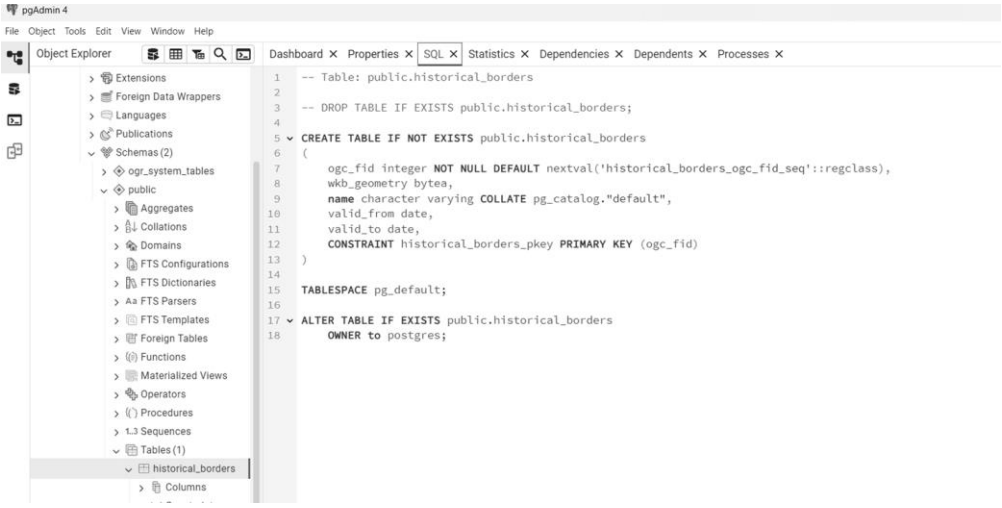

Рисунок 3.13 — SQL код таблиці historical_borders

Здійснивши налаштування і наповнення бази даних historical_borders постало завдання інтеграції цієї бази з основним середовищем розробки — Visual

Studio Code (VS Code). Для цього необхідно встановити такі додаткові розширення в програмному середовищі:

— SQLTools — універсальний менеджер підключень до баз даних.

— SQLTools PostgreSQL/Redshift Driver — драйвер, що забезпечує підтримку PostgreSQL у межах SQLTools.

— PostrgeSQL — необхідна для розробки база даних, точніше щоб приєднатися до неї.

Принцип встановлення їх такий самий як і інших розширень, у пошуковику в лівій вкладці прописуємо дані програмні додатки і їх вибираємо та встановлюємо.

Згодом у лівій колонці у нас додадуться дві іконки. Одна іконка, буде у формі циліндра, а інша в формі слона. Приблизний вигляд додатків зображено на рисунку 3.14 нижче.

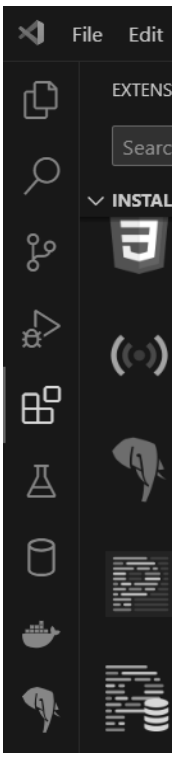

Рисунок 3.14 — Кінцевий вигляд встановлених розширень для роботи в vs code

Після інсталяції розширень необхідно буде підключити базу даних до програмного середовища. Для цього потрібно буде зайти в SQLTools та через неї

підключити PostreSQL. Можна звісно просто натиснути на іконку слона і його обрати, але це не найоптимальніший варіант, оскільки саме розширення PostreSQL не дуже поєднується з Windows 11, через це не завжди вдаватиметься запускати необхідну базу для ініціалізації картографічних даних. Тож потрібну бд обираємо через звичайний SQLTools. Там буде вибір між більш простим SQL і PostreSQL, обираємо другий і заповнюємо таблицю, приклад заповнення поданий на рисунку 3.15 [19].

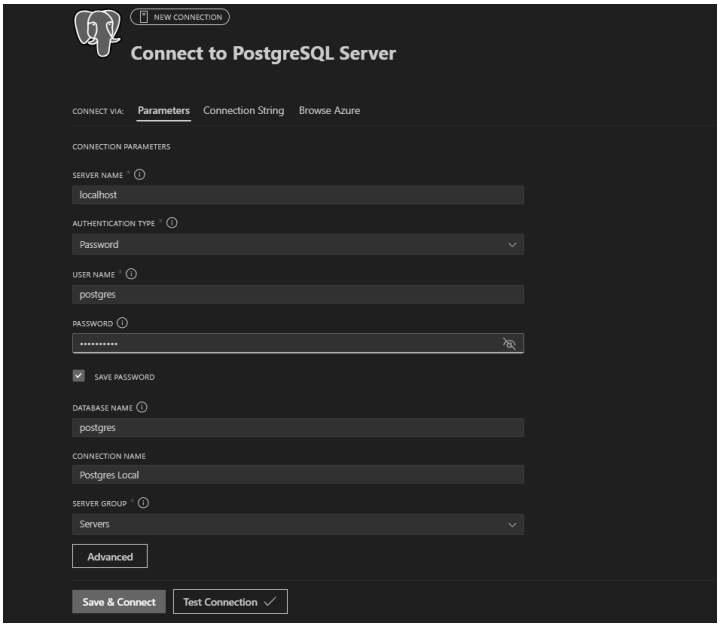

Рисунок 3.15 — Заповнення необхідної таблиці для підключення до БД

Під заповнення форми необхідно бути обережним, адже неправильне заповнення форми, нівелює всі зусилля і доведеться заново все перевстановлювати. Окрім, хіба що QGIS, де будуть зберігатися координати.

Після заповнення необхідно спочатку провести Test Connection, щоб зрозуміти чи база правильно підключилася, і лише тоді натискати на Save & Connect після цього в дереві програми у нас з’явиться .session.sql файл. У ньому ми можемо прописувати SQL - команди щоб взаємодіяти з базою даних. На рисунку нижче зображено.session.sql файл в vs code, який також і є самим файлом який зображений рисунках 3.16,3.17. Зліва на рисунку 3.16 видно структуру, яка повністю повторює структуру PostgreSQL. Друга колонка таблиці wbk_geometry — це цифрові дані на основі яких працювати має бекенд.

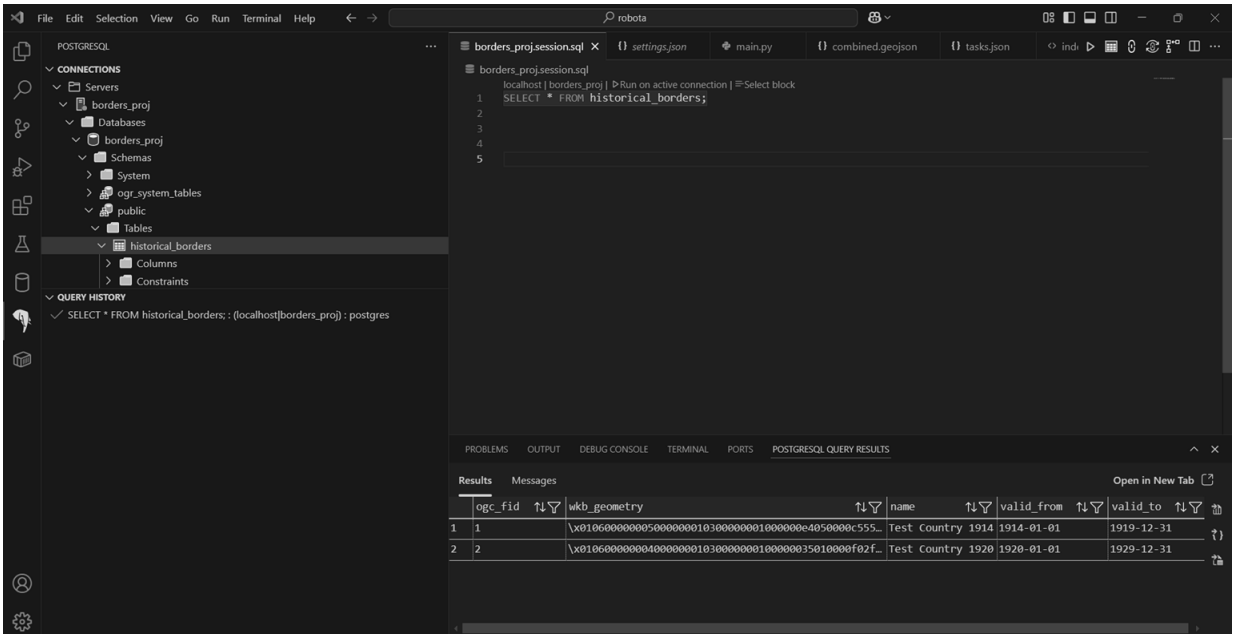

Рисунок 3.16 — Інтегрована база sql з vs code

3.4 Розробка клієнтської частини

Клієнтська частина веб-додатку реалізована за допомогою стандартних вебтехнологій — HTML, CSS і JavaScript. Цього буде достатньо, адже сама карта буде задавати дизайн сама по собі. Основною її метою є надання користувачу можливості обрати історичний період та переглянути відповідні зміни кордонів у вигляді інтерактивної карти. Для реалізації карти використовується бібліотека Leaflet.js, яка є зручною та легковаговою альтернативою іншим GIS-інструментам частина коду для завантаження бібліотеки вказана в лістингу 3.7.

Лістинг 3.7 — Встановлення бібліотеки Leaflet.js

```
<!DOCTYPE html>
<html>
<head>
   <meta charset="utf-8">
   <title>Historical Borders</title>
    <link rel="stylesheet" href="https://unpkg.com/leaflet@1.9.4/dist/leaflet.css" />
   <style>
      #map { height: 90vh; }
      body { font-family: sans-serif; padding: 10px; }
   </style>
</head>
```

Тут зображена початкова структура коду клієнта. Тут оголошується його тип. Встановлюється назва вкладки браузера між тегами title. Під цим же рядком реалізовано підключення до зовнішнього CSS-файлу бібліотеки Leaflet.js з CDN, що є необхідним для коректного вигляду карти. Між тегами <style> вказано розміщення карти у браузері, тут задано відступ зверху, карта займає 90 % висоти від видимого вікна, та шрифти з підписами, що вказано в лістингу 3.8.

Лістинг 3.8 — Лістинг місцезнаходження карти на сторінці браузера

```
 <h2>Історичні кордони</h2>
<label>Рік: <input type="number" id="year" value="1914" /></label>
<button onclick="loadBorders()">Завантажити</button>
<div id="map"></div>
```

Вище заданий інтерфейс користувача. Зокрема строфа для введення необхідного року, та кнопка для завантаження даних, тої чи іншої дати. У верхніх тегах заданий основний заголовок сторінки, а в лістингу 3.9 реалізовано підключення необхідної карти, для веб - додатку.

Лістинг 3.9 — Лістинг функціональних можливостей завантаженої карти

```
<script src="https://unpkg.com/leaflet@1.9.4/dist/leaflet.js"></script>
   <script>
      const map = L.map('map').setView([50, 30], 4);
      L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
         maxZoom: 18
      }).addTo(map);
      let layer;
```

Перший рядок підключає повнофункціональну бібліотеку leaflet.js. Дана бібліотека є ключовою в розробці інтерфейсу клієнта. Нижче створюється карта і центр звідки буде запускатися огляд карти, спираючись на координати це приблизно над Україною. L.tileLayer підключає шар з відкритих карт OpenStreetMap. Структура URL автоматично підставляється Leafle tзалежно від положення карти. Максимальний рівень масштабування встановлено як 18, що забезпечує достатню деталізацію. Метод .addTo(map) додає цей шар на карту. Остання стрічка відповідає за тимчасового зберігання шару меж, який буде завантажено з сервера на основі обраного року.

У розроблюваному веб-додатку передбачено можливість динамічного завантаження історичних кордонів на основі вибраного року. За це відповідає окрема функція loadBorders(), написана на мові JavaScript. Нижче зображений лістинг 3.10 :

Лістинг 3.10 — Лістинг завантаження історичних кордонів на карту

```
function loadBorders() {
        const year = document.getElementById('year').value;
        fetch(`http://localhost:8000/borders/${year}`)
          .then(res => res.json())
          .then(data => {
            if (layer) map.removeLayer(layer);
            layer = L.geoJSON(data).addTo(map);
            map.fitBounds(layer.getBounds());
          })
          .catch(err => alert("Помилка завантаження: " + err));
    }
```

Ця функція викликається кожного разу, коли користувач натискає кнопку “Завантажити”. Вона починається з зчитування значення року з елемента форми (getElementById). Далі використовується функція fetch(), яка здійснює HTTP-запит до API-сервера. У цьому рядку формується динамічне посилання типу http://localhost:8000/borders/1914, де рік підставляється у відповідне місце. Відповідно до серверної логіки, такий запит повертає географічні межі країн за вказаний рік у форматі GeoJSON, після чого відповідь перетворюється у формат GeoJSON за допомогою наступного методу .json. Наступні рядки відповідають за додавання кордонів на карті. Зокрема перевіряється наявність шару на карті, якщо так то за допомогою map.removeLayer(layer) старий шар видаляється і створюється новий. Новий шар виводиться на карту на що вказує такий рядок коду layer = L.geoJSON(data).addTo(map). Метод layer.getbounds відповідає за показ нового та масштабування нового шару. Останній рядок коду .catch(err => alert("Помилка завантаження: " + err));створений для обробки помилок, наприклад, наприклад, сервер не відповідає, або даних на такий рік немає. На рисунку 3.17 зображено з інтерфейсом розроблюваного додатку.

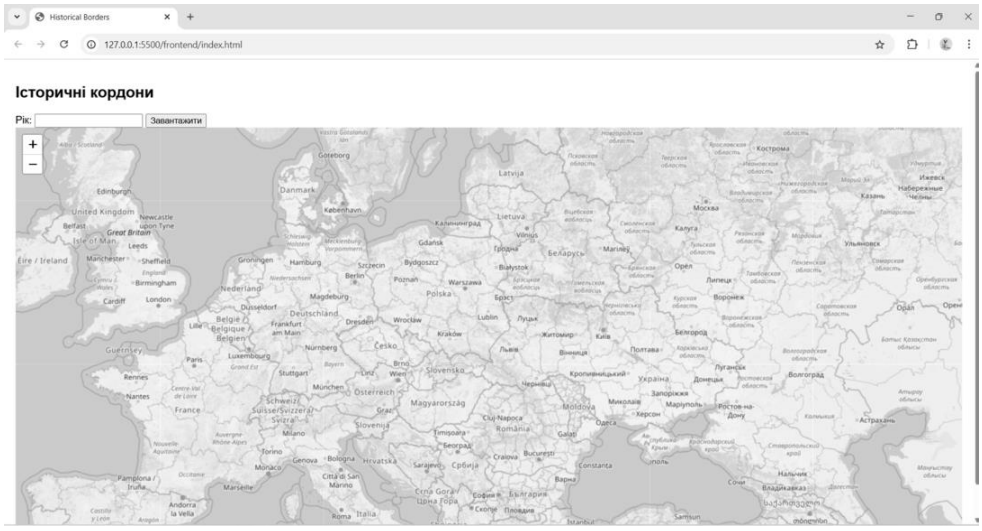

Рисунок 3.17 — Інтерфейс розроблюваного додатку

Загалом після розробки всієї архітектури програми дерево даного веб - додатку виглядатиме так, як вказано на рисунку 3.18:

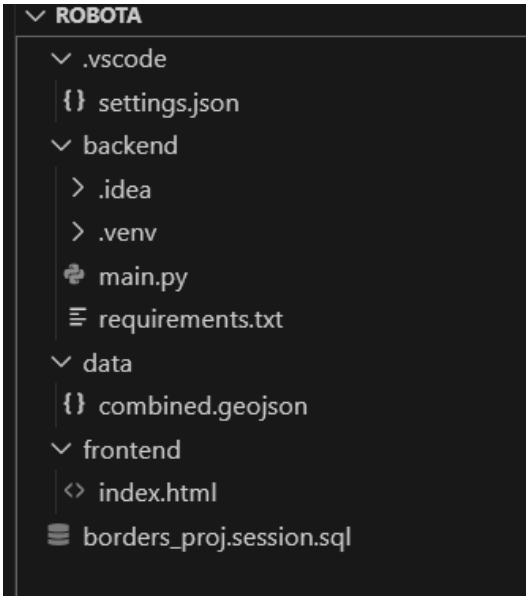

Рисунок 3.18 — Кінцева архітектура веб – додатку

3.5 Відображення динамічних змін кордонів

Спочатку активуємо браузер у vs. Code через go to live. На сторінці сайту, після активації браузеру повинна з’явитися карта та весь необхідний інтерфейс для взаємодії з нею і можемо відображати необхідні кордони держав. Для початку, якщо ми оберемо якусь дату наприклад, 1890р. як зображено на рисунку 3.19.

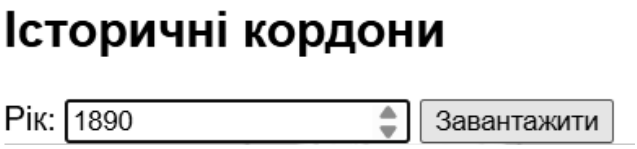

Рисунок 3.19 — обраний рік для візуалізації даних

Після введення 1890 р. необхідно натиснути на кнопку завантажити і в нас на карті з явиться зображення як подано на рисунку 3.20 з кордонами кайзерівської Німеччини:

Рисунок 3.20 — Відображення кордонів Німеччини до І світової війни

Наскільки вам відомо в нашій базі завантажено 2 кластери координат для однієї держави в різні часові рамки. Якщо ми оберемо дату між 1886 - 1914рр. То Отримаємо зображення подано на рисунку 3.24. Тепер зробимо ще один такий крок, але вже з Німеччиною у міжвоєнний період. Введемо наприклад, 1933 рік, як на рисунку 3.21 коли до влади прийшли нацисти.

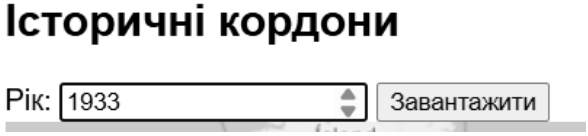

Рисунок 3.21 — наступний обраний рік для візуалізації даних

Після введення дати 1933р., отримаємо державу вже з дещо іншими кордонами як показано на рисунку 3.22. Другий кластер завантажених координат відповідає часовим рамкам 1920 - 1939рр, тобто відповідає кордонам Німеччини в період Веймарської республіки/Третього рейху. Хоча кордони схожі до першої держави, там вже є окупована судетська область Чехії та частина регіону східної Прусії у Польщі.

Рисунок 3.22 — Відображення кордонів Німеччини у міжвоєнний період

Якщо у вікно була введена дата, точніше рік, який не співпадає з вище обраними діапазонами держав, то появиться повідомлення про помилку. Оскільки база не зможе підключитися.

3.6 Тестування працездатності веб -додатку

Після завершення етапу розробки було проведено комплексне тестування працездатності веб-додатку з метою перевірки його функціональності, стабільності та коректної взаємодії між клієнтською та серверною частинами. Тестування охоплювало всі основні компоненти системи: від завантаження інтерфейсу користувача до обробки запитів та відображення змін кордонів на інтерактивній карті.

На початковому етапі було протестовано завантаження веб-додатку у сучасних браузерах, зокрема Google Chrome, продемонстрованому на рисунку 3.23. Карта та її базові шари відображались коректно, переміщення, масштабування та взаємодія з елементами інтерфейсу виконувались плавно, без візуальних артефактів або збоїв, приклади тестування наведенні на рисунках нижче.

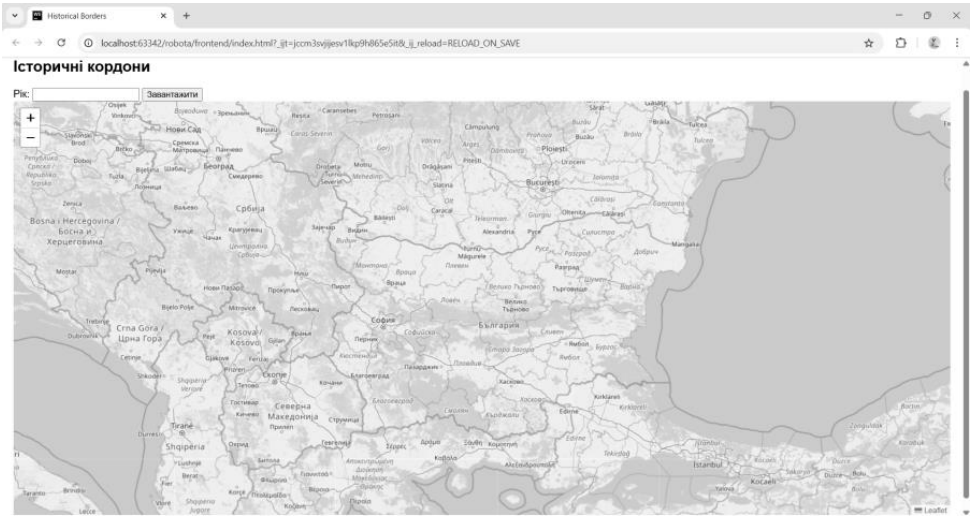

Рисунок 3.23 — Тестування працездатності веб - застосунку у браузері Google Chrome

На наступному етапі було протестовано завантаження додатку у браузері Microsoft Edge. Отримані результати зображені на рисунку 3.24 мали еквівалентний успіх по відношенню до першого браузера.

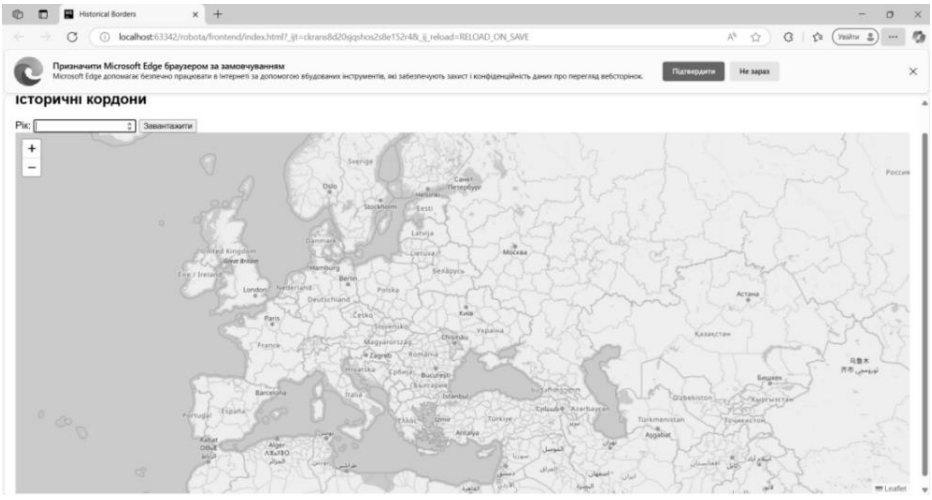

Рисунок 3.24 — Тестування працездатності веб - застосунку у браузері Microsoft Edge

Останнім етапом тестування була перевірка адаптивності інтерфейсу на різних розмірах екранів. Веб-додаток успішно підлаштовувався під мобільні пристрої та планшети, забезпечуючи зручну навігацію та взаємодію з картою. Результат показано на рисунку 3.25.

Рисунок 3.25 — Тестування працездатності веб - застосунку на мобільному пристрої

За підсумками тестування було встановлено, що веб-додаток функціонує стабільно, забезпечує швидку обробку даних, коректне візуальне відображення змін кордонів та ефективну взаємодію між усіма його компонентами.

4 ОЦІНКА ТА ОПТИМІЗАЦІЯ РОБОТИ ВЕБ-ДОДАТКУ

4.1 Тестування продуктивності клієнт-серверної взаємодії

Тестування продуктивності клієнт-серверної взаємодії є критично важливою складовою процесу розробки будь-якого веб-додатку, що передбачає передачу, обробку та візуалізацію значних обсягів даних. У цьому розділі було проведено всебічний аналіз функціонування системи у режимі реального часу, із врахуванням ключових показників продуктивності.

Перш за все, було протестовано затримку між запитом з клієнтської частини та отриманням відповіді від сервера. Для цього використовувались HTTP-запити, що імітували дії користувача при перемиканні між роками, зміні масштабування карти та активації різних шарів. Середній час очікування відповіді складав 200–300 мс при стабільному з’єднанні, що свідчить про ефективну організацію серверної логіки.

У рамках тестування була перевірена ефективність REST API, через яке клієнтська частина взаємодіє з базою даних. Було встановлено, що запити є оптимізованими, не перевантажують сервер зайвими операціями, а структура відповідей дозволяє мінімізувати обсяг переданих даних.

Проводилось також тестування продуктивності у сценаріях з одночасним підключенням кількох користувачів. За допомогою інструменту Apache JMeter було згенеровано навантаження у вигляді 50–100 паралельних запитів. Усі запити були оброблені без помилок, а середній час відповіді зростав незначно, що підтверджує масштабованість серверної частини.

Клієнтська частина також пройшла окреме тестування. Оцінювалась здатність браузера швидко відображати нові дані після отримання їх із сервера. Усі операції на інтерфейсі виконувались без помітних зависань або уповільнень, що вказує на використання ефективних механізмів рендерингу компонентів.

Було перевірено, наскільки швидко завантажуються різні набори географічних даних у форматі GeoJSON. Навіть великі файли, що містять кілька тисяч полігонів,

оброблялись за 1, 2 секунди. Це стало можливим завдяки застосуванню методів попередньої агрегації та оптимізації формату даних.

Використання браузерного кешу також дало позитивні результати. При повторному зверненні до тих самих даних час завантаження скорочувався вдвічі, що дозволяє зменшити навантаження на сервер та підвищити загальну продуктивність.

Оцінка ефективності взаємодії клієнта із сервером проводилась у різних браузерах: Chrome, Firefox та Edge. Результати свідчать про кросбраузерну сумісність додатку та стабільність обміну даними в усіх протестованих середовищах [20].

Під час тестування враховувались і нетипові ситуації, наприклад, повільне інтернет-з'єднання або втрати пакетів. У таких умовах система залишалась працездатною, що демонструє надійність реалізації обробки помилок і повторного запиту даних.

Додатково були перевірені сценарії зі зміною часових періодів, які потребують оновлення великої кількості геооб'єктів на карті. Завдяки використанню бібліотеки Leaflet, відображення нових меж здійснювалося без перезавантаження всієї карти, що значно покращує UX.

Особлива увага приділялась перевірці механізмів обробки запитів до бази даних PostgreSQL з розширенням PostGIS. Було переконанося, що всі запити мають індексацію за ключовими полями, що дозволяє швидко фільтрувати дані за роками.

Було протестовано передачу даних у форматах JSON та GeoJSON. Обидва формати виявилися придатними для реалізації задачі, однак GeoJSON забезпечив кращу інтеграцію з картографічною бібліотекою без додаткової трансформації.

Під час навантажувального тестування з імітацією пікової активності система продемонструвала стабільність роботи. Сервер не переходив у стан перевантаження, а затримки залишались у межах допустимих значень.

У результаті проведених досліджень можна стверджувати, що клієнт-серверна архітектура реалізована якісно, з урахуванням принципів оптимізації

взаємодії, продуктивності та відмовостійкості. Наявна структура дозволяє легко масштабувати додаток та додавати новий функціонал без втрати швидкодії. Таким чином, тестування підтвердило здатність веб-додатку ефективно функціонувати у реальних умовах використання, забезпечуючи швидке та коректне відображення змін кордонів у різних часових періодах.

4.2 Оптимізація швидкості завантаження даних

Однією з ключових цілей при розробці веб-додатку для візуалізації змін кордонів є забезпечення максимальної швидкодії завантаження та обробки даних. Зважаючи на обсяги геопросторової інформації, які можуть охоплювати декілька сотень мегабайт. Оптимізація швидкості завантаження даних є дуже важливою, оскільки навряд чи користувач захоче користуватися продуктом, який довго грузиться. З протидії таким проблемам було проведено цілий ряд заходів з оптимізації передачі, зберігання та візуалізації даних.

Першочергово було впроваджено серверну фільтрацію даних. Замість надсилання повного масиву меж усіх періодів, система формує запит до бази даних лише за обраним користувачем роком або діапазоном. Це дозволило зменшити обсяг даних, що передаються через мережу, у 5–10 разів.

Для забезпечення швидкої вибірки інформації використовувалась індексація в базі даних PostgreSQL з використанням розширення PostGIS. Геоіндекси були застосовані до просторових полів, що суттєво скоротило час виконання запитів.

У клієнтській частині застосовано динамічне завантаження даних (lazy loading). Географічні об'єкти не завантажуються всі одразу, а підвантажуються порціями відповідно до області перегляду карти. Завдяки цьому уникається перевантаження оперативної пам’яті та браузера при масштабних запитах.

Дані передаються у форматі GeoJSON, який добре підтримується картографічними бібліотеками. Проте перед надсиланням на клієнт GeoJSON стискається за допомогою GZIP, що зменшує розмір файлів у середньому на 70%. Компресія відбувається на сервері автоматично, а браузери її підтримують без додаткових налаштувань.

Також було реалізовано кешування відповідей сервера. Для статичних даних (наприклад, меж за певний рік) встановлено тривалий час кешування. Це дозволяє повторно використовувати вже завантажені файли без нових запитів до сервера.

Окрему увагу приділено оптимізації структури GeoJSON-файлів. Було видалено всі зайві властивості, скорочено назви полів та усунуто дублікати геометрій. Це зменшило розмір вихідних файлів без втрати корисної інформації.

Застосовано асинхронну обробку запитів на клієнті за допомогою fetch із async/await. Це дозволяє продовжувати взаємодію з інтерфейсом, поки дані завантажуються у фоновому режимі, створюючи враження високої швидкодії.

Було проведено аналіз вузьких місць при завантаженні через інструменти Chrome DevTools. Основними були ідентифіковані великі обсяги необроблених даних, дублікати запитів та зайві рендери на карті. Після усунення цих проблем середній час повного завантаження сторінки зменшився з 5 секунд до 1,5 секунди.

Ще одним кроком стало використання пагінації та API з параметрами. У запити до сервера додано параметри limit і offset, що дозволяє дозовано запитувати частини даних при потребі — наприклад, у списках змін чи у панелях атрибутів.

На рівні карти реалізовано кластеризацію об'єктів. Це дозволило не лише візуально спростити вигляд карти при масштабі, а й уникнути одночасного рендерингу сотень полігонів [21].

Було проведено оптимізацію стилів і скриптів: усі JS-файли об'єднано та мініфіковано за допомогою Webpack. Це значно зменшило розмір клієнтської частини та скоротило час першого завантаження.

Для швидшої ініціалізації карти було реалізовано попереднє завантаження базових тайлів, які не потребують авторизації або обробки даних. Таким чином, користувач бачить перші результати практично миттєво.

Загальна оцінка продуктивності, проведена з використанням Lighthouse та GTmetrix, підтвердила високі показники: швидкість відповіді сервера — менш як

500 мс, час інтерактивності — до 1,8 с, що відповідає сучасним вимогам до оптимізованих веб-додатків.

У результаті реалізованих заходів було досягнуто суттєвого покращення в часі відгуку системи, зменшено навантаження на клієнтську частину та забезпечено ефективне використання ресурсів як на сервері, так і на фронтенді.

4.3 Оцінка зручності використання

У процесі розробки особлива увага приділялася простоті та зрозумілості користувацького інтерфейсу як показано на рисунку 4.1. У верхній частині розташовано заголовок сторінки, поле для введення року та кнопку для завантаження даних. Нижче представлена інтерактивна карта з можливістю навігації (масштабування та перетягування). Така структура є інтуїтивно зрозумілою для користувачів, які погано ладнають з використанням комп’ютерів, ноутбуків.

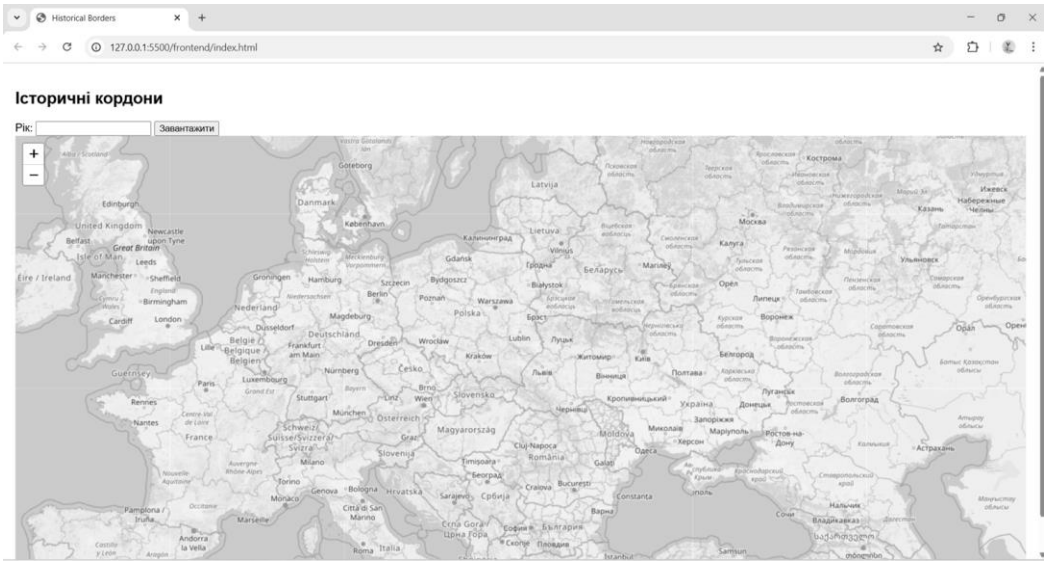

Рисунок 4.1 — Інтерфейс розроблюваного додатку

Форма для введення року реалізована у вигляді простого числового поля, що дає змогу швидко змінювати параметр часу. Кнопка "Завантажити" однозначно вказує на дію, яку необхідно виконати для оновлення даних на карті.

Завдяки використанню бібліотеки Leaflet карта працює стабільно, плавно масштабується та реагує на введення без затримок.

Оцінюючи загальну зручність, інтерфейс можна вважати вдалим для першої версії додатку. Він не перевантажений елементами, легко адаптується до ширини вікна браузера та забезпечує виконання головного завдання — перегляду історичних змін кордонів. У подальшому, для покращення взаємодії, можна передбачити додаткові інструменти, такі як слайдер часу, підказки при наведенні на полігони, або динамічну легенду. Проте навіть у наявному вигляді інтерфейс є функціональним і дружнім до користувача. Єдине, що зверху можна написати підказку, які роки доступні для використання.

ВИСНОВКИ

У ході виконання бакалаврської роботи було реалізовано повнофункціональний веб-додаток, який дозволяє інтерактивно візуалізувати зміни державних кордонів у різні історичні періоди. Для цього було розроблено клієнтську частину з використанням бібліотеки Leaflet, яка забезпечує зручне та інтуїтивно зрозуміле відображення просторових даних на основі відкритих карт OpenStreetMap. Серверна частина була реалізована на базі фреймворку FastAPI та взаємодіє з базою даних PostgreSQL, розширеною за допомогою PostGIS для зберігання та обробки географічної інформації.

Основну увагу було приділено реалізації клієнт-серверної архітектури та пошуку необхідних даних завдяки чому вдалося забезпечити динамічне завантаження даних про кордони відповідно до запиту користувача, а також масштабування карти завдяку чому користувачу легко орієнтуватися у просторі та часі. Впроваджено простий інтерфейс, який дозволяє швидко змінювати рік і переглядати відповідні геополітичні зміни. Уся інформація надається у форматі GeoJSON, що є стандартом для просторових вебдодатків і гарантує сумісність з іншими сервісами.

Під час роботи було вирішено низку задач: від аналізу існуючих технологій і вибору оптимальної архітектури до безпосередньої реалізації інтерфейсу та бази даних. Проведено оцінку зручності використання додатку, яка засвідчила простоту взаємодії з користувачем і ефективність відображення даних. У перспективі можливе розширення функціональності додатку, зокрема додавання анімації змін у часі, відображення додаткових історичних подій, інтеграція з іншими джерелами даних.

Таким чином, поставлену мету досягнуто — було створено адаптивний вебдодаток, що дозволяє вивчати зміни державних кордонів у зручному інтерактивному форматі. Робота має практичну цінність у контексті цифрової гуманітаристики, освіти, історичних досліджень та інформування широкої аудиторії.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. М.О Зубар А.В. Кожем’яко. Веб-додаток для візуалізації змін кордонів з використанням клієнт-серверної архітектури. (2025) ВНТКП ВНТУ. Факультет інформаційних технологій та комп'ютерної інженерії. Отримано з https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2025/paper/view/23974/19790

2. Boundary Changes : вебсайт. URL: https://atlas.co/gis-use-cases/boundary-changes/(дата звернення: 21.03.2025)

3. Visualizing the Past: Mapping, GIS, and Teaching Historical Consciousness : вебсайт. URL:https://activehistory.ca/blog/2016/07/13/visualizing-the-past-mapping-gis-and-teaching-historical-consciousness/(дата звернення:23.03.2025)

4. Six (6) Best GIS web and online or cloud Mapping platforms : вебсайт. URL: https://orbital.co.ke/the-5-best-gis-web-and-online-mapping-platforms/(дата звернення: 26.03.2025)

5. Посилання на сайт TimeMapper вебсайт. URL: https://timemapper.okfnlabs.org/ (дата звернення: 04.04.2025)

6. Посилання на сайт GeaCron: вебсайт. URL: https://geacron.com/home-en/(дата звернення: 04.04.2025)

7. Посилання на сайт chapter 4 Data Management : вебсайт. URL: https://postgis.net/docs/using_postgis_dbmanagement.html (дата звернення: 08.04.2025)

8. Visual Studio Code : вебсайт. URL: https://www.webopedia.com/definitions/visual-studio-code/(дата звернення: 12.04.2025)

9. Python in Visual Studio Code : вебсайт. URL: https://code.visualstudio.com/docs/languages/python(дата звернення: 12.04.2025)

10. FastAPI features : вебсайт. URL: https://fastapi.tiangolo.com/features/(дата звернення 12.04.2025)

11. Using GeoJSON Geometry data to draw borders with TomTom Maps : вебсайт. URL: https://developer.tomtom.com/blog/build-different/using-geojson-geometry-data-draw-borders-tomtom-maps/(дата звернення: 14.04.2025)

12. Chapter 11 Spatial and Temporal Data Management: вебсайт. URL: https://www.richardtwatson.com/open/Reader/_book/spatial-and-temporal-data-management.html (дата звернення: 14.04.2025)

13. Dynamic web maps with Mapbox Tiling Service and mapgl: вебсайт. URL: https://walker-data.com/mapboxapi/articles/dynamic-maps.html(дата звернення: 14.04.2025)

14. Install PostgreSQL on Windows: вебсайт. URL: https://neon.com/postgresql/postgresql-getting-started/install-postgresql(дата звернення: 15.04.2025)

15. FastAPI Tutorial in Visual Studio Code : вебсайт. URL: https://code.visualstudio.com/docs/python/tutorial-fastapi (дата звернення: 15.04.2025)

16. Посилання на сайт для пошуку координат GeoJSON : вебсайт. URL: https://demo.ldproxy.net/cshapes/collections/boundary/items?limit=10&offset=80 (дата звернення: 17.04.2025)

17. Connecting QGIS to Postgres and PostGIS : вебсайт. URL: https://www.crunchydata.com/blog/connecting-qgis-to-postgres-and-postgis (дата звернення: 20.04.2025)

18. GDAL documentation ogr2ogr : вебсайт. URL: https://gdal.org/en/stable/programs/ogr2ogr.html (дата звернення: 21.04.2025)

19. Getting Started with the PostgreSQL Extension for VSCode : вебсайт. URL: https://ryanhutzley.medium.com/getting-started-with-the-postgresql-extension-for-vscode-d666c281ec72 (дата звернення: 23.04.2025)

20. Performance Optimizations for Geospatial Dash Apps : вебсайт. URL: https://plotly.com/blog/performance-optimization-geospatial/ (дата звернення: 25.04.2025)

21. Analyze runtime performance: вебсайт. URL: https://developer.chrome.com/docs/devtools/performance (дата звернення: 25.04.2025)

ДОДАТОК А

Технічне завдання

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки

ЗАТВЕРДЖУЮ

Завідувач кафедри ОТ

д.т.н., проф. Азаров О.Д.

«21» березня 2025 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання бакалаврської кваліфікаційної роботи

«Веб-додаток для візуалізації змін кордонів з використанням клієнт-серверної архітектури»

08-54.БКР.028.00.000 ТЗ

Науковий керівник: к.т.н., доц. каф. ОТ

Кожем'яко А. В.

Студент групи 2КІ-21б

Зубар М. О.

ВНТУ 2025

1 Підставою для виконання бакалаврської кваліфікаційної роботи (БКР)

1.1 Попит суспільства на використання географічних даних у освітніх та гуманітарних цілях. Традиційні джерела подання історичних кордонів, такі як паперові атласи або статичні зображення, обмежені у функціональності та не дозволяють здійснювати гнучкий аналіз змін у часі та просторі, для більш ефективного освоєння матеріалу. Сучасні історичні веб - додатки з візуалізації кордонів не є дуже досконалими у своїй точності.

1.2 Наказ про затвердження теми БКР

2 Мета БКР і призначення розробки

2.1 Мета роботи — створення клієнт-серверного веб-додатку для візуалізації змін державних кордонів у часі з використанням відкритих географічних даних та інтерактивної мапи на основі OpenStreetMap.

2.2 Призначення розробки — створення простого, але якісного веб - додатку для візуалізації змін кордонів історичних держав в часі, призначеного для використання у навчальному процесі, історичних дослідженнях, популяризації знань з геополітики.

3 Вихідні дані для використання в БКР

3.1 Аналіз сучасних додатків з технологіями використання геоінформаційних даних.

3.2 Відкриті геопросторові дані про історичні кордони держав у форматі GeoJSON.

3.3 Технологічна база для реалізації проєкту, а саме: бібліотека Leaflet для клієнтської візуалізації, FastAPI як фреймворк для серверної частини, PostgreSQL з розширенням PostGIS для зберігання геоданих, середовище розробки Visual Studio Code та QGIS для підвищення ефективності баз даних

3.4 Реалізація веб - додатку на основі трьох попередніх пунктів

4 Вимоги до виконання БКР

Робота повинна бути виконана згідно затвердженої теми та наказу про затвердження теми БДР. Робота повинна включати в себе технології клієнт - серверної архітектури та бути реалізованою з використанням геоданих в форматі GeoJSON. Веб - додаток має бути розроблений у середовищі visual studio code.

5 Етапи БКР та очікувані результати

Етапи виконання БКР заданні у таблиці А.1 нижче

Таблиця А.1 — Календарний план виконання БКР

№	Назва та зміст етапу	Термін виконання		Примітка
		початок	закінчення	
1	Вибір, узгодження та затвердження теми БДР	21.03.2025	24.03.2025	
2	Освоєння необхідних мов програмування для розробки проєкту	25.03.2025	2.04.2025	
3	Розробка кінцевого змісту для записки	8.04.2025	11.04.2025	
4	Розробка тези та визначення з кінцевими технологіями використання	12.04.2025	20.04.2025	
5	Початок пошуку літературних джерел для детального розуміння можливих функцій програми	21.04.2025	26.04.2025	
6	Початок розробки перших 2 пунктів записки	26.04.2025	29.04.2025	
7	Програмна реалізація додатку	30.04.2025	12.05.2025	
8	Попередній захист БДР та доопрацювання всіх аспектів	13.05.2025	10.06.2025	
9	Нормоконтроль та рецензування БДР	10.06.2025	10.06.2025	
10	Захист БДР	13.06.2025	13.06.2025	

6 Матеріали, що подаються до захисту БКР

До захисту подаються: пояснювальна записка БКР, графічні і ілюстративні матеріали, протокол попереднього захисту БКР на кафедрі, відгук наукового керівника, рецензія, анотації БКР українською та іноземною мовами, довідка про відповідність оформлення БКР діючим вимогам.

7 Порядок контролю виконання та захисту БКР

Виконання етапів графічної та розрахункової документації БКР контролюється керівником згідно зі встановленими термінами. Захист БКР відбувається на засіданні Екзамеційної комісії, затвердженої наказом ректора.

8 Вимоги до оформлювання та порядок виконання БКР

8.1 При оформлюванні БКР використовуються:

— ДСТУ 3008: 2015 «Звіти в сфері науки і техніки. Структура та правила оформлювання»;

— ДСТУ 8302: 2015 «Бібліографічні посилання. Загальні положення та правила складання»;

— міждержавний ГОСТ 2.104-2006 «Єдина система конструкторської документації. Основні написи»;

— методичні вказівки до виконання бакалаврських кваліфікаційних робіт зі спеціальності 123 – «Комп’ютерна інженерія» (освітня програма «Комп’ютерна інженерія»). Кафедра обчислювальної техніки ВНТУ 2022;

— документами на які посилаються у вище вказаних.

ДОДАТОК Б

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: _____

Тип роботи: _____ бакалаврська кваліфікаційна робота _____

(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ _____кафедра обчислювальної техніки_____

(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism ___1____%

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

✓Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту

☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

_____Азаров О.Д., зав. каф. ОТ _____ _____

(прізвище, ініціали, посада) (підпис)

____Крупельницький Л.В., доц. каф ОТ_____ _____

(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ ___Захарченко С.М.____

(підпис) (прізвище, ініціали)

З висновком експертної комісії ознайомлений(-на)

Керівник _____ _____

(підпис) (прізвище, ініціали, посада)

Здобувач _____ _____

(підпис) (прізвище, ініціали)

ДОДАТОК В

ГРАФІЧНА ЧАСТИНА

ОРГАНІЗАЦІЯ АРХІТЕКТУРИ ВЕБ-ДОДАТКУ ДЛЯ ВІЗУАЛІЗАЦІЇ ЗМІН КОРДОНІВ З ВИКОРИСТАННЯМ КЛІЄНТ-СЕРВЕРНОЇ АРХІТЕКТУРИ

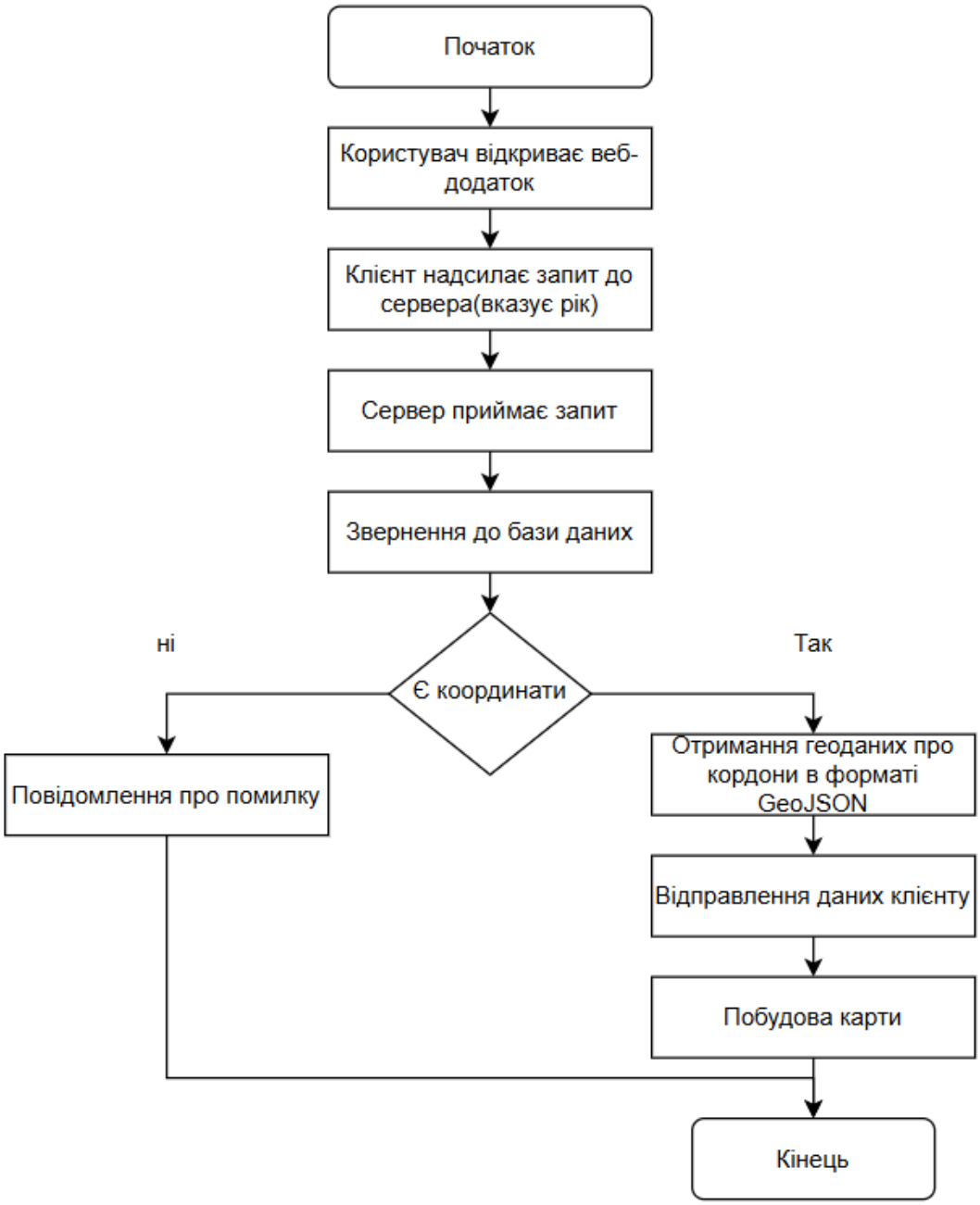

Рисунок В.1— Блок-схема алгоритму візуалізації кордонів держав

ДОДАТОК Г

ІЛЮСТРАТИВНА ЧАСТИНА

ДЕМОНСТРАЦІЯ РОБОТИ ВЕБ-ДОДАТКУ ДЛЯ ВІЗУАЛІЗАЦІЇ ЗМІН КОРДОНІВ З ВИКОРИСТАННЯМ КЛІЄНТ-СЕРВЕРНОЇ АРХІТЕКТУРИ

Рисунок В.1— обраний рік для візуалізації кордонів держави

Рисунок В.2 — візуалізовані кордони держави станом на 1890р.

ДОДАТОК Д

Лістинги програми

Лістинг Д.1 — Клієнтська частина програми

```
<!DOCTYPE html>
<html>
<head>
   <meta charset="utf-8">
   <title>Historical Borders</title>
    <link rel="stylesheet" href="https://unpkg.com/leaflet@1.9.4/dist/leaflet.css" />
   <style>
      #map { height: 90vh; }
      body { font-family: sans-serif; padding: 10px; }
   </style>
</head>
<body>
   <h2>Історичні кордони</h2>
   <label>Рік: <input type="number" id="year" value="1914" /></label>
   <button onclick="loadBorders()">Завантажити</button>
   <div id="map"></div>
   <script src="https://unpkg.com/leaflet@1.9.4/dist/leaflet.js"></script>
   <script>
      const map = L.map('map').setView([50, 30], 4);
      L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
         maxZoom: 18
      }).addTo(map);
      let layer;
      function loadBorders() {
         const year = document.getElementById('year').value;
         fetch(`http://localhost:8000/borders/${year}`)
   .then(res => res.json())
   .then(data => {
      console.log("Received data:", data);
      if (layer) map.removeLayer(layer);
      layer = L.geoJSON(data).addTo(map);
      map.fitBounds(layer.getBounds());
   })
   .catch(err => console.error("Fetch error:", err));
      }
   </script>
</body>
</html>
```

Лістинг Д.2 — Серверна частина програми

```
from fastapi import FastAPI
from fastapi.middleware.cors import CORSMiddleware
from fastapi.responses import JSONResponse
import psycopg2
import json

app = FastAPI()

app.add_middleware(
    CORSMiddleware,
    allow_origins=["http://localhost:8000"],
    allow_methods=["GET"],
    allow_headers=["*"],
)

DB_CONFIG = {
    "dbname": "borders_proj",
    "user": "postgres",
    "password": "Mereth@12#",
    "host": "localhost",
    "port": 5432,
}

@app.get("/borders/{year}")
def get_borders(year: int):
    try:

        conn = psycopg2.connect(**DB_CONFIG)
        print("Database connection successful")
        cur = conn.cursor()


        query = """
            SELECT ST_AsGeoJSON(geom), name FROM historical_borders
            WHERE valid_from <= %s AND valid_to >= %s
        """
        cur.execute(query, (f"{year}-01-01", f"{year}-12-31"))


        features = [
            {
                "type": "Feature",
                "properties": {"name": row[1]},
```

```
                    "geometry": json.loads(row[0]) if row[0] else None,
                }
                for row in cur.fetchall()
            ]

            cur.close()
            conn.close()
            return JSONResponse(content={"type": "FeatureCollection", "features":
features})

        except psycopg2.Error as e:
            print(f"Database connection failed: {e}")
            return JSONResponse(status_code=500, content={"error": f"Database
connection failed: {e}"})
```

Лістинг Д.3 — Координати в PosrgreSQL


```
{
 "type": "FeatureCollection",
 "features": [
  {
   "type": "Feature",
   "properties": {
    "name": "Test Country 1914",
    "valid_from": "1886-01-01",
    "valid_to": "1914-12-31"
   },
   "geometry": {
    "type": "MultiPolygon",
    "coordinates": [
     [
      [
       [
        12.53337,
        54.4669495
       ],
       [
        12.5269447,
        54.4741629
       ],
       [
        12.4805543,
        54.4441632
       ],
       [
        12.4369446,
        54.3911083
       ],

...
       [
        9.7925002,
        55.0750001
       ]
      ]
     ]
    ]
   }
  },
  {
```

```
      "type": "Feature",
      "properties": {
        "name": "Test Country 1920",
        "valid_from": "1920-01-01",
        "valid_to": "1939-12-31"
      },
      "geometry": {
        "type": "MultiPolygon",
        "coordinates": [
          [
            [
              [
                19.2533272,
                54.2781911
              ],
              [
                19.2307632,
                54.3336731
              ],
              [
                19.2776362,
                54.3461091
              ],
              [
                19.3761092,
                54.3525001
              ],

...
              [
                9.4467521,
                54.8271448
              ],
              [
                9.4147239,
                54.8333274
              ]
            ]
          ]
        ]
      }
    }
  ]
}
```