Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Бакалаврська кваліфікаційна робота

на тему: «Розробка програмного засобу для візуалізації великих обсягів просторових даних у геоінформаційних системах»

Виконав: студент 4 курсу
групи 2ПІ-21б
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Довгань І.С.
(прізвище та ініціали)
Керівник: к.т.н., доц. каф. ПЗ Хошаба О.М.
(прізвище та ініціали)
Рецензент: к.т.н., доц. каф. ЗІ Лукічов В.В.
(прізвище та ініціали)

Допущено до захисту
Зав. кафедр проф. О.Н.Романюк
«9» 06 2025 р.

ВНТУ – 2025

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти перший бакалаврський
Галузь знань 12 - Інформаційні технології
Спеціальність 121 – Інженерія програмного забезпечення
Освітньо-професійна програма - Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О. Н.
_21_ березня 2025 року

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Довганю Ігорю Сергійовичу

1. Тема роботи - «Розробка програмного засобу для візуалізації великих обсягів просторових даних у геоінформаційних системах»

Керівник роботи: Хошаба Олександр Мирославович, к.т.н., доцент кафедри ПЗ, затверджені наказом вищого навчального закладу від 20 березня 2025 р. № 97.

2. Строк подання студентом роботи 2 червня 2025 р.

3. Вихідні дані до роботи: кількість просторових об'єктів для відображення: до 5 млн., формат даних: GeoJSON, 3D Tiles, PostGIS, швидкість обробки запитів: не менш ніж 200 мс при зміні масштабу, мінімальна продуктивність: не менше 30 FPS при рендерингу сцени з 500 тис. об'єктів, підтримка інтерактивних функцій: фільтрація, вибір об'єктів, анімація, максимальне навантаження: до 100 одночасних клієнтів.

4. Зміст розрахунково-пояснювальної записки: вступ, аналіз предметної галузі візуалізації просторових даних, порівняльний аналіз програмних засобів візуалізації (Mapbox GL JS, CesiumJS, ArcGIS Pro, QGIS), розробка програмної архітектури та основних модулів (рендеринг, кеш, шар даних, кластеризація), програмна реалізація та особливості інтеграції (PostGIS, тайли, 3D Tiles), тестування програмного засобу (вебінтерфейс, продуктивність, навантаження),

особливості розгортання та супроводження (хмарні сервіси, безсерверна архітектура), висновки, список використаних джерел, додатки.

5. Перелік графічного матеріалу: мета та задачі роботи, порівняльна характеристика аналогів, розробка програмної архітектури та основних модулів, тестування програмної системи, висновки.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Хошаба О.М., к.т.н., доцент кафедри ПЗ	24.03.25	01.06.25

7. Дата видачі завдання 24 березня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз проблеми візуалізації великих обсягів даних у геоінформаційних системах	25.03.2025-03.04.2025	вик.
2	Проектування модулів, розробка алгоритмів та моделей в геоінформаційних системах	04.04.2025-14.04.2025	вик
3	Вибір середовища та розробка програмного забезпечення геоінформаційних систем	15.04.2025-05.05.2025	вик.
4	Тестування роботи візуалізації великих обсягів даних у геоінформаційних системах	06.05.2025-19.05.2025	вик.
5	Оформлення матеріалів до захисту БКР	20.05.2025-30.05.2025	вик.

Студент _____ Довгань І.С.

(підпис) (прізвище та ініціали)

Керівник бакалаврської кваліфікаційної роботи Хошаба О.М.

(підпис) (прізвище та ініціали)

АНОТАЦІЯ

УДК 528.9

Довгань І.С. Розробка програмного засобу для візуалізації великих обсягів просторових даних у геоінформаційних системах : бакалаврська кваліфікаційна робота зі спеціальності 121 Інженерія програмного забезпечення, освітня програма – інженерія програмного забезпечення. Вінниця: ВНТУ, 2025. 119 С.

На укр. мові. Бібліогр. : 38 назв ; рис. : 16; табл. 9.

В роботі проведено комплексне дослідження сучасних підходів до візуалізації великих обсягів просторових даних у геоінформаційних системах. Здійснено порівняльний аналіз провідних технологій (Mapbox GL JS, CesiumJS, ArcGIS Pro, QGIS), що дозволило виявити їх переваги та обмеження у контексті масштабованості, продуктивності, розширюваності та підтримки реального часу.

Визначено, що WebGL-технології з GPU-рендерингом у браузері дозволяють досягти високої швидкодії навіть при виведенні мільйонів геооб'єктів. Особливо ефективними виявились механізми кластеризації, асинхронного завантаження, рівнів деталізації (LOD) та стрімінгового оновлення шарів даних.

Розроблено програмний засіб на основі технологій WebGL і хмарних обчислень, що реалізує кластеризацію, рівні деталізації (LOD), кешування тайлів і потокову обробку. Проведено порівняльний аналіз Mapbox GL JS, CesiumJS, ArcGIS Pro та QGIS. Система реалізована як вебзастосунок, що підтримує інтеграцію з PostGIS та обробку реального часу.

Практична апробація показала, що запропоноване рішення забезпечує суттєве підвищення продуктивності (у 5–7 разів швидше за базові Leaflet-рішення) та покращення користувацького досвіду через оптимізоване кешування, адаптивне завантаження та ефективне UI. Також доведено доцільність використання мікросервісної архітектури та serverless-компонентів у побудові масштабованих ГІС-платформ нового покоління.

ANNOTATION

Dovhan I.S. Development of a software test for the visualisation of large volumes of spatial data in geographic information systems : bachelor's thesis on the specialty 121 Software engineering, educational program - software engineering. Vinnytsia: VNTU, 2025. 119 with.

In Ukrainian language. Bibliographer : 38 titles; Fig. : 16; table 9.

The paper comprehensively studied modern approaches to visualising large volumes of spatial data in geographic information systems. It carried out a comparative analysis of technologies (Mapbox GL JS, CesiumJS, ArcGIS Pro, QGIS), which allowed it to identify their advantages and limitations in the context of scalability, performance, extensibility, and real-time support.

It was determined that WebGL technologies with GPU rendering in the browser allow high speed even with billions of rendered geo objects. The mechanisms of clustering, asynchronous loading, levels of detail (LOD), and streaming update of data layers turned out to be especially effective.

A software tool based on WebGL and cloud computing technologies was developed. It implements clustering, levels of detail (LOD), tile caching, and streaming processing. A comparative analysis of Mapbox GL JS, CesiumJS, ArcGIS Pro, and QGIS was carried out. The system is implemented as a web application that supports integration with PostGIS and real-time processing.

Practical testing has shown that the proposed solution significantly increases performance (5–7 times faster than the basic Leaflet solution) and improves user experience through optimised caching, adaptive loading, and an efficient user interface. The feasibility of using microservice architecture and serverless components to build a scalable GIS platform of the new generation has also been proven.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. Розробка програмного засобу по підбору персоналу для ІТ-компаній має велику актуальність в сучасному світі з багатькох причин. Насамперед, це зумовлено зростанням попиту на ІТ-індустрію. Відповідно до цього, висока конкуренція на ринку праці у ІТ-галузі призводить до того, що компанії шукають ефективні способи привернути та утримати талановитих працівників.

Тому, розробка програмного засібу по підбору персоналу може допомогти компаніям знаходити кандидатів, які найкращим чином відповідають їхнім потребам і культурі, тим самим покращуючи їх шанси залучити талановитих фахівців. Насамперед, розробка програмного засобу дозволяє автоматизувати багато процесів підбору персоналу, таких як оголошення вакансій, сортування резюме, проведення співбесід та аналіз даних кандидатів. Це допомагає зберегти час і зусилля рекрутерів, а також знижує ймовірність помилок при відборі кандидатів.

Але й існують певні проблеми до розробки програмного забезпечення в цій галузі. До найважливих з них відноситься велике різноманіття ІТ-професій, так як ІТ-галузь включає в себе широкий спектр професій, від розробників програмного забезпечення і тестувальників до архітекторів мереж і аналітиків даних. Кожна професія вимагає своїх власних наборів навичок і знань. Розробка програмного засобу повинна враховувати це різноманіття і забезпечувати можливість відповідного підбору персоналу для кожної конкретної посади.

Також, помилковий підбір персоналу може призвести до незадоволеності працівників щодо їх низької продуктивності роботи. Тому, розроблена автоматизована система повина забезпечувати високу точність відповідності кандидатів до вимог посади і культурі компанії, щоб знизити ризик помилкового підбору.

Зв’язок роботи з науковими програмами, планами, темами. Робота виконувалась згідно плану виконання наукових досліджень на кафедрі програмного забезпечення

Мета та завдання. Метою роботи є підвищення ефективності візуалізації великих обсягів просторових даних у геоінформаційних системах шляхом розробки програмного засобу з використанням сучасних вебтехнологій, методів кластеризації, рівнів деталізації та хмарної обробки даних.

Відповідно до поставленої мети виконані наступні задачі:

- виконати аналіз предметної галузі візуалізації великих обсягів просторових даних у геоінформаційних системах;
- провести порівняльний аналіз програмних засобів що використовують візуалізацію великих обсягів просторових даних у геоінформаційних системах;
- запропоновати метод кешування даних на основі компоненту тайлів Google Maps API;
- запропоновати метод офлайн-інтеграції з картографічними сервісами;
- розробити функціональні модулі системи кешування просторових даних;
- розробити програмний код за методом кластеризації маркерів;
- виконати тестування модуля системи кешування просторових даних;
- виконати тестування кешування тайлів у локальному сховищі.

Об’єктом дослідження є процес візуалізації великих обсягів просторових даних у геоінформаційних системах.

Предметом дослідження є методи, моделі та програмні засоби, що забезпечують масштабовану, продуктивну та інтерактивну візуалізацію геопросторових даних з використанням сучасних вебтехнологій.

Методи дослідження. У процесі дослідження використовувалися теорія просторових баз даних для моделювання структури та запитів до геоданих; алгоритми кластеризації та узагальнення (наприклад, DBSCAN, QuadTree, H3) для оптимізації відображення великих обсягів точок; методи WebGL-візуалізації та GPU-рендерингу для реалізації продуктивного виведення; архітектурні підходи мікросервісів та безсерверних обчислень (serverless) для реалізації масштабованої backend-частини; парадигми потокової обробки (stream processing) для інтеграції даних у реальному часі; методи асинхронного програмування та кешування (lazy loading, smart caching) для забезпечення зручності роботи користувача.

Новизна отриманих результатів:

1. Запропоновано метод кешування даних на основі компоненту тайлів Google Maps API з використанням Java/WebGL, особливість якого полягає у попередньому завантаженні та локальному кешуванні растрових і векторних шарів, що надає можливість підвищити швидкодію системи під час роботи з великими об'ємами даних та зменшити затримку відображення картографічних шарів приблизно на 32–46% за рахунок уникнення повторних звернень до сервера, оскільки оновлені тайли завантажуються майже миттєво з локального сховища.

2. Запропоновано метод кластеризації маркерів, який враховує їх просторову щільність і масштаб відображення зображень на основі особливості близько розташованих точок які автоматично групуються в кластери з використанням WebGL-акселерації для обчислення кластерів на стороні клієнта, що дозволяє зменшити кількість одночасно рендерених об'єктів та підвищити швидкодію візуалізації рендерингу на 24% завдяки меншій кількості відображуваних маркерів та оптимізації обробки подій кластерів.

3. Подальшого розвитку отримав підхід WebGL-рендерингу з попередньою агрегацією даних, у якому, на відміну від існуючих Canvas/SVG-рішень відбуваються обчислення кластерів і спрощення геометрій виконуються на GPU, що дозволяє поєднувати серверну підготовку тайлів із клієнтською обробкою даних внаслідок кластеризації та децимації на рівні тайла) та оптимізувати обсяг передаваних даних внаслідок зниження мережевих затримок та підвищення продуктивності візуалізації великих геопросторових масивів приблизно на 53% з використанням прискоренних обчислень на апаратному рівні WebGL.

Практична цінність отриманих результатів. Практична цінність роботи полягає у розробці програмного засобу, який дозволяє ефективно візуалізувати великі обсяги просторових даних у браузері на основі WebGL, з підтримкою кластеризації, кешування, 3D Tiles, інтеграції з PostGIS та потокової обробки. Розроблена система може бути впроваджена в муніципальні ГІС, системи екологічного моніторингу, платформу управління інфраструктурою та мобільні додатки. Використання запропонованого засобу значно покращує продуктивність

відображення даних, підвищує зручність користування, забезпечує гнучке масштабування та інтуїтивну аналітику.

Особистий внесок здобувача. Усі наукові результати, викладені у бакалаврській кваліфікаційній роботі, отримані автором особисто. Автору належать такі результати: розробка архітектура програмної системи; створення алгоритмів у вигляді UML-діаграм; розробка модулів клієнтської та серверної частин.

Апробація результатів роботи. Результати роботи були представленні на конференції «X Міжнародної науково-практичної конференції з проблем вищої освіти і науки "Інформаційні технології в освіті, науці і виробництві (ІТОНВ-2025)"».

Публікації. Результати роботи були опубліковані в матеріалах конференції – «X Міжнародної науково-практичної конференції з проблем вищої освіти і науки "Інформаційні технології в освіті, науці і виробництві (ІТОНВ-2025)"» [1].

1 АНАЛІЗ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз предметної галузі візуалізації великих обсягів просторових даних у геоінформаційних системах

Термін «великі дані» (англ. Big Data) зазвичай використовується для позначення значних за обсягом і складністю масивів інформації, які неможливо ефективно обробляти та аналізувати за допомогою традиційних систем управління базами даних (СУБД). У контексті геоінформаційних систем (ГІС) великі просторові дані охоплюють значні масиви географічної інформації, що включає координати та атрибути, і потребують застосування спеціальних технологій та методів візуалізації для їх ефективного зберігання, обробки та аналізу [2, 3]. Візуалізація просторових даних передбачає їх графічне представлення у вигляді дво- та тривимірних карт, інтерактивних сцен, графіків і діаграм для полегшення сприйняття просторових закономірностей і трендів [4].

Спеціальні технології та методи, які застосовуються для ефективного зберігання, обробки та аналізу великих просторових даних у контексті ГІС, включають такі підходи.

Одним із ключових методів є використання спеціалізованих геопросторових баз даних, таких як PostGIS та SpatiaLite, які надають розширені функції для ефективної роботи з географічними даними на рівні просторових запитів, індексування та оптимізації операцій. Ці технології дозволяють здійснювати швидкий пошук і вибірку даних за просторовими критеріями, зокрема пошук об'єктів у межах певної території або визначення відстаней між об'єктами.

Іншим ефективним підходом є використання спеціалізованих форматів зберігання та передачі геопросторових даних, таких як GeoJSON, TopoJSON, Protocol Buffers (PBF), які забезпечують компактне зберігання інформації, швидку передачу через мережу та ефективну візуалізацію, особливо в веб-інтерфейсах та мобільних застосунках.

Важливу роль також відіграють системи інтелектуального попереднього завантаження та кешування геопросторових тайлів, такі як TileStache та MapProxy, які забезпечують швидке відображення даних без зайвих запитів до серверів. Цей

метод ефективно працює в умовах високих навантажень, коли велика кількість користувачів одночасно взаємодіє з інтерактивними картами.

Також варто згадати підхід на базі гібридних сховищ даних (Hybrid Data Stores), які поєднують реляційні та NoSQL-системи, дозволяючи комбінувати переваги структурованих запитів і гнучкості зберігання напівструктурованих даних, що особливо актуально для інтеграції різнорідних геопросторових наборів.

Нарешті, технології інкрементального (incremental) оновлення даних забезпечують оновлення великих просторових наборів інформації частинами, не перевантажуючи систему і користувацькі інтерфейси. Це дозволяє підтримувати актуальність інформації у реальному часі без значних витрат ресурсів на повну повторну обробку та візуалізацію даних.

Джерела просторових даних надзвичайно різноманітні та включають супутникові й аерокосмічні знімки, отримані шляхом дистанційного зондування Землі, дані з GPS та GNSS-приймачів, що забезпечують трекінг рухомих об'єктів, інформацію з польових і бездротових сенсорних мереж (наприклад, метеорологічні дані та дані Інтернету речей), краудсорсингові платформи типу OpenStreetMap, картографічні матеріали, кадастрові реєстри, дані переписів населення тощо. Зазначені джерела формують великі масиви структурованої та неструктурованої інформації, які є важливими для різних галузей економіки, науки та управління територіями [5, 6].

Одним із важливих джерел є аерофотознімання з використанням безпілотних літальних апаратів (дронів), що дозволяють отримувати високоточні і детальні дані про місцевість, зокрема для створення цифрових моделей поверхні, ортофотопланів та картографування важкодоступних або небезпечних територій.

Іншим унікальним типом даних є мобільне лазерне сканування (Mobile LiDAR), яке здійснюється спеціалізованими автомобілями або переносними системами, що надає високоточні тривимірні моделі доріг, будівель, мостів та інших інфраструктурних об'єктів. Такі дані використовуються у дорожньому будівництві, містобудуванні, моніторингу стану інфраструктури.

Важливими є також соціальні мережі як джерела просторових даних (так звані геотеговані повідомлення), які дозволяють аналізувати просторову

поведінку населення, визначати популярні місця, траєкторії переміщення людей під час масових заходів або стихійних лих, що допомагає планувати та координувати дії у випадку надзвичайних ситуацій.

Також суттєвим джерелом інформації можуть бути архівні історичні карти та плани, які, завдяки сучасним методам оцифрування та геоприв'язки, інтегруються в сучасні ГІС для аналізу змін у використанні земельних ресурсів, розвитку міської інфраструктури, вивчення історичних ландшафтів і екологічних змін.

Не менш актуальним є використання даних з геоакустичних сенсорів (гідрофони та сейсмічні сенсори), що застосовуються для моніторингу стану морського дна, підводних споруд, а також для виявлення та аналізу сейсмічних явищ. Ці дані є особливо важливими для геологічних досліджень, попередження стихійних лих і для підтримки безпеки морських споруд.

Кожне з цих джерел даних має свою специфіку отримання, обробки та застосування, доповнюючи інформаційну базу геоінформаційних систем і розширюючи можливості просторового аналізу та прийняття управлінських рішень.

Обробка великих просторових даних характеризується необхідністю вирішення трьох основних завдань: забезпечення високої продуктивності, масштабованості та зручності користування. В умовах значних обсягів інформації інтерактивні ГІС-інтерфейси стикаються з проблемою зниження швидкодії, що обмежує ефективність роботи користувача [7]. Для вирішення цього застосовують методики рівнів деталізації (LOD - Level of Detail), кластеризації просторових точок, розподілені системи обробки даних (наприклад, платформи Hadoop чи Spark), оптимізовані просторові індекси та поступове завантаження даних (lazy loading), а також кешування тайлів. Ці підходи забезпечують швидке виконання запитів і навігацію навіть при великих обсягах інформації [8, 9].

Для ефективного вирішення завдань забезпечення високої продуктивності, масштабованості та зручності користування інтерактивними геоінформаційними системами в умовах значних обсягів інформації використовуються кілька спеціалізованих підходів і технологій, які раніше не розглядалися.

Одним із підходів є динамічне попереднє завантаження даних (Predictive Prefetching), коли система передбачає подальші дії користувача (наприклад, переміщення чи масштабування карти) і завантажує відповідні тайли або шари інформації завчасно. Це дозволяє зменшити затримки у відображенні даних і забезпечити миттєву реакцію на дії користувача, значно покращуючи інтерактивність системи.

Для забезпечення масштабованості використовуються технології автоматичного балансування навантаження (Load Balancing), що розподіляють запити користувачів між декількома серверами залежно від їхнього поточного завантаження та місцезнаходження клієнта. Це дозволяє ефективно використовувати ресурси інфраструктури, уникати перенавантаження окремих серверів та підтримувати стабільну роботу системи навіть при пікових навантаженнях.

Важливим аспектом забезпечення високої продуктивності є використання асинхронних механізмів взаємодії з базами даних та серверами (Asynchronous Data Fetching), коли запити до геопросторових сховищ здійснюються паралельно, без блокування основного потоку інтерфейсу користувача. Це дозволяє отримувати частину даних оперативно та поступово оновлювати картографічне представлення без затримок або «замороження» інтерфейсу.

З метою покращення зручності користування застосовуються методи адаптивного дизайну (Adaptive UI/UX Design), які автоматично підлаштовують інтерфейс відповідно до типу пристрою, його розміру екрана і доступних ресурсів. Це забезпечує оптимальне використання екранного простору, швидке завантаження даних і комфортну взаємодію з геопросторовою інформацією незалежно від типу пристрою, на якому працює користувач.

Нарешті, одним з перспективних підходів є використання технологій Edge Computing (граничні обчислення), які переносять частину обчислювальних задач ближче до пристроїв користувачів, зменшуючи затримки, прискорюючи обробку даних і підвищуючи загальну продуктивність інтерактивних ГІС-систем. Цей підхід особливо ефективний у мобільних мережах з обмеженою пропускною здатністю та при взаємодії з великими потоками даних у режимі реального часу.

Серед сучасних технологій візуалізації великих просторових даних особливою популярністю користуються веб-ГІС з використанням технології WebGL, зокрема Mapbox GL JS, OpenLayers та CesiumJS, що дозволяють створювати інтерактивні та динамічні карти прямо у веб-браузері [10]. Важливе місце займають також 3D-ГІС та технології віртуальної і доповненої реальності (VR/AR), які надають можливість детальної тривимірної візуалізації просторових даних, що значно покращує інтуїтивність сприйняття та аналіз складних географічних сценаріїв [11, 12]. Крім того, дедалі частіше застосовуються потокові підходи (стрімінг) для обробки інформації в режимі реального часу з датчиків та інших джерел, а також хмарні платформи, наприклад Google Earth Engine, які дозволяють аналізувати просторові дані на глобальному рівні [13].

Веб-ГІС з використанням технології WebGL набули особливої популярності завдяки своїй здатності ефективно вирішувати комплексні задачі візуалізації просторових даних безпосередньо у браузері, використовуючи апаратні ресурси графічних процесорів клієнтських пристроїв. Саме технологія WebGL (Web Graphics Library) дозволяє рендерити складні графічні об'єкти, векторні та растрові шари, а також тривимірні моделі з високою продуктивністю і без необхідності встановлювати додаткове програмне забезпечення або плагіни.

Однією з головних причин широкого поширення WebGL-технологій є їхня незалежність від операційних систем та типів пристроїв. Завдяки тому, що WebGL підтримується практично всіма сучасними браузерами, користувачі можуть отримувати доступ до складних інтерактивних карт на різних платформах, включаючи мобільні телефони, планшети та настільні комп'ютери. Це забезпечує універсальність використання і значно спрощує розробку і підтримку веб-ГІС-додатків.

Іншим важливим фактором популярності таких бібліотек, як Mapbox GL JS, OpenLayers та CesiumJS, є відкритість їхнього програмного коду, що дозволяє гнучко налаштовувати та розширювати функціональність відповідно до специфічних потреб різних проектів. Розробники мають змогу активно впроваджувати нові можливості, оптимізувати продуктивність та інтегрувати різноманітні сторонні сервіси та джерела геопросторових даних.

Застосування WebGL також надає широкі можливості для інтеграції з іншими сучасними веб-технологіями, такими як React, Angular, Vue.js, що дозволяє створювати ефективні односторінкові веб-додатки (SPA) з високим рівнем інтерактивності. Крім того, завдяки використанню асинхронних механізмів завантаження даних та динамічного оновлення візуалізації, користувачі отримують швидкий та плавний досвід взаємодії з картами, що особливо важливо при великих обсягах інформації.

Нарешті, високий рівень підтримки спільноти та наявність детальної документації роблять ці бібліотеки доступними не лише для досвідчених розробників, але й для широкого кола користувачів з базовими навичками програмування. Це сприяє швидкому поширенню таких технологій, що є додатковою перевагою при їх виборі для створення сучасних ГІС-рішень.

Таким чином, сучасні підходи до візуалізації великих обсягів просторових даних мають низку значних переваг, серед яких: інтерактивність та оперативність доступу до даних завдяки використанню веб-технологій та потокових методів, високий рівень інтуїтивності і точності сприйняття завдяки застосуванню тривимірних та VR/AR-технологій, а також можливість глобального масштабування завдяки хмарним платформам. Все це сприяє ефективнішому аналізу геопросторової інформації та прийняттю обґрунтованих рішень у різних галузях управління та науки [14, 15]. Ефективна візуалізація великих просторових даних має критичне значення для прийняття обґрунтованих управлінських і стратегічних рішень. Візуальні форми представлення інформації (карти, графіки, інтерактивні панелі) допомагають виявляти приховані просторові закономірності, тенденції та аномалії, які неможливо або складно розпізнати при використанні стандартних табличних представлень. Як підкреслюють дослідники [14, 15], саме візуальні патерни дозволяють швидше та точніше здійснювати прогнозування і ухвалювати якісні рішення на основі складних і великих масивів геоданих.

Сучасні підходи до візуалізації великих обсягів просторових даних мають низку суттєвих переваг, які стали можливими завдяки одночасному розвитку апаратних засобів, архітектур клієнт-сервер та парадигм обробки даних у реальному часі. Інтерактивність та оперативність доступу до даних

забезпечуються насамперед через застосування веб-сокетів (WebSocket) і протоколів потокової передачі, які дозволяють встановлювати постійне двостороннє з’єднання між клієнтом і сервером. Це дає змогу оперативно передавати оновлення даних у режимі реального часу, без необхідності повторного завантаження сторінки чи виконання повного запиту, як це було притаманно класичним веб-архітектурам.

Інтуїтивність і точність сприйняття значною мірою підвищуються завдяки впровадженню систем семантичної візуалізації, які адаптують вміст до контексту користувача. Наприклад, в 3D-ГІС використовуються алгоритми автоматичного узагальнення об’єктів, що дозволяють приховувати другорядні елементи та акцентувати увагу на ключових об’єктах при навігації. Поєднання 3D-візуалізації з аналітичними шарами (heatmaps, потокові вектори, часові шкали) створює багатовимірне уявлення про геопросторові явища, що підвищує якість аналітики та підтримки прийняття рішень.

Ще одним важливим чинником є використання концепції «візуалізації як послуги» (Visualization-as-a-Service, VaaS), яка реалізується в хмарних середовищах. У цьому випадку складні обчислення, побудова графічних сцен і моделювання виконуються на потужних серверних інстанціях, тоді як кінцевий користувач отримує лише графічний результат через веб-інтерфейс. Це дозволяє знімати обчислювальне навантаження з клієнтських пристроїв, забезпечуючи масштабування як обсягу даних, так і кількості одночасних користувачів.

Також сучасні хмарні рішення активно застосовують концепцію «безсерверної архітектури» (serverless computing), яка дозволяє динамічно масштабувати ресурси залежно від потреб запитів до візуалізаційного середовища. Це мінімізує затрати на інфраструктуру і дозволяє автоматично адаптувати систему до поточних умов навантаження, що особливо актуально для глобальних проєктів, де важливо підтримувати високу доступність незалежно від регіону користувача.

Тому, сукупність новітніх мережевих протоколів, семантичного моделювання, хмарних сервісів і безсерверних архітектур забезпечує якісно новий

рівень у візуалізації просторових даних, де користувач отримує точну, своєчасну і релевантну інформацію в зручній інтерактивній формі.

Візуалізація великих просторових наборів даних є складним завданням через обмеженість ресурсів користувацьких пристроїв та необхідність високої швидкодії. Зростання обсягів ГІС-даних (мільйони–мільярди об’єктів) призводить до «перевантаження» класичних методів відображення. Щоб гарантувати інтерактивність та зрозумілу подачу інформації, застосовують спеціальні методи (табл. 1.1): від кластеризації і узагальнення даних до багаторівневих (LOD) моделей, потокової передачі, розподіленої обробки та хмарних технологій, а також – 3D- і VR/AR-візуалізації. Наступні таблиці (табл. 1.1, табл. 1.2) порівнюють основні методологічні класи рішень за ключовими характеристиками (тип візуалізації, масштабованість, ефективність, обмеження, прикладне застосування, принцип роботи).

Таблиця 1.1 - Порівняльна характеристика сучасних підходів до візуалізації великих обсягів просторових даних за критеріями масштабованості та продуктивності

Підхід	Тип (2D/3D)	Масштабованість / продуктивність	Обмеження
Кластеризація (агрегація)	Переважно 2D (точкові дані); можливе застосування в 3D-оточенні	Хороша (зменшує кількість точок для відображення, знижує візуальний «хаос»); залежить від алгоритму і розміру даних	Втрата точних координат, нечіткість деталей; потрібно налаштування радіусу/розміру кластерів.
Рівні деталізації (LOD)	2D/3D (часто 3D: моделі, рельєф); у 2D – тайли різних зумів	Дуже висока (пірамідальні/ієрархічні структури дозволяють обслуговувати будь-який масштаб)	Потрібна попередня генерація наборів даних на різних рівнях та структурована ієрархія (більший обсяг зберігання).

Потокова/прогресивна візуалізація	2D/3D (векторна і растрова)	Висока (завантажуються поступово лише необхідні дані)	Залежність від мережі; інтеграція буферів: показуються спочатку грубі дані, потім деталізація; чутливе до затримок каналу.
Розподілені обчислення (HPC)	2D/3D (як кінцевий результат – 2D-карти, зображення; 3D-реконструк- ції)	Дуже висока (паралельні кластери або GPU-обчислення; масштабуються горизонтально)	Складна інфраструктура; комунікаційні затрати (MPI, MapReduce); не підходять для дрібних запитів.
Хмарні технології	2D/3D (зберігання та обробка в хмарі; з фінальним рендером на клієнті)	Вкрай висока (еластичні ресурси без граничних обмежень)	Витрати залежно від обсягу; затримки через мережу; безпека і приватність даних.
3D-GIS / візуалізація 3D-даних	3D	Обмежена класичними підходами (без LOD – низька), з LOD/тайлами – добра.	Дуже об’ємні дані (мільярди точок або трикутників), важкі текстури; складні формати.
VR/AR-візуалізація	3D (переважно VR/AR-сцени)	Низька (обмежено потужністю апаратури HMD/графіки; рендеринг у реальному часі)	Високі апаратні вимоги (GPU, HMD), вузький спектр кейсів, питання зручності користування. Ефективність сильно залежить від оптимізації даних.

Сучасні підходи до візуалізації великих обсягів просторових даних значною мірою підвищують ефективність обробки інформації в ГІС за критеріями масштабованості та продуктивності, завдяки використанню розподілених та

паралельних обчислювальних моделей, адаптивного завантаження даних, а також архітектурної декомпозиції компонентів систем. Масштабованість досягається шляхом впровадження мікросервісної архітектури, де кожен модуль відповідає за конкретну функціональність (рендеринг, квері, кешування, індексування) і може масштабуватись незалежно, відповідно до обсягу оброблюваних даних чи кількості запитів.

Підвищення продуктивності забезпечується за рахунок попередньої агрегації та фрагментації даних, які зберігаються у вигляді просторових тайлів або чанків (tiles/chunks) з оптимальними розмірами, що дозволяє здійснювати вибіркове завантаження лише необхідних елементів, уникаючи обробки всього масиву. Для побудови таких структур використовуються просторові схеми індексації з підтримкою рівнів деталізації, наприклад, H3 (Hexagonal Hierarchical Indexing) або S2 Geometry Library (Google), що забезпечує швидкий доступ до необхідних ділянок карти при зміні масштабу або фільтрації даних.

Крім того, актуальною практикою є використання механізму «інтелектуального кешування» (smart caching), який враховує не лише просторову частину запитів, а й поведінкові патерни користувачів. Таким чином, найбільш ймовірно запитувані області попередньо кешуються на рівні CDN (Content Delivery Network), що знижує час відповіді та зменшує навантаження на бекенд. У поєднанні з асинхронною обробкою запитів та використанням worker-процесів, система отримує здатність обробляти тисячі одночасних звернень без деградації продуктивності.

Додатково масштабованість та продуктивність підтримуються шляхом інтеграції з потоковими обчислювальними фреймворками, такими як Apache Flink або Apache Kafka Streams, що дозволяють обробляти потоки геопросторових подій (наприклад, телеметрія, моніторинг, зміни шару) у реальному часі без накопичення великих пакетів. Це створює основу для оперативного оновлення візуалізаційної інформації без втрат у продуктивності та з мінімальною затримкою.

В результаті зазначені технології дозволяють масштабувати геоінформаційні системи горизонтально (додаванням нових вузлів), зберігаючи

при цьому високу швидкодію як для одиничних запитів, так і для багатокористувацьких сценаріїв. Вони забезпечують стійку роботу в умовах зростання кількості джерел, обсягів даних і навантаження з боку користувачів.

Таблиця 1.2 - Порівняльна характеристика сучасних підходів до візуалізації великих обсягів просторових даних за принципами та застосуванням

Підхід	Застосування (галузі)	Принципи
Кластеризація (агрегація)	Інтерактивні веб-карти, мобільні ГІС, аналіз щільності (наприклад, POI, соціальні мережі)	Групування близько розташованих об’єктів у кластери; заміна багатьох точок однією символічною відміткою (розмір якої пропорційний кількості). Зберігаються ієрархічні відносини (можуть застосовуватися динамічно на різних масштабах)
Рівні деталізації (LOD)	3D-моделі міст, цифрові рельєфи, мапи високої роздільності; VR-подання зон.	Побудова ієрархії представлень: деталізована модель/зображення спрощуються у низькодеталізовані версії (LOD). Застосовується просторове розбиття (Quadtree/Octree), щоб за потреби завантажувати і рендерити тільки потрібну підмножину даних. Наприклад, 3D-тайли зберігаються на сервері (формати 3D Tiles, i3S), стрімуються залежно від позиції камери.
Потокова/прогресивна візуалізація	Веб-ГІС та мобільні клієнти з обмеженим каналом зв’язку; оперативні системи навігації.	Послідовна передача даних із зростаючою деталізацією. Наприклад, обри́сові представлення (LOD) передаються попередньо, потім «дорисовуються» вершини/полігони. Використовують стиск і поступове вилучення вершин (vertex removal) для плавного нарощування деталей.
Розподілені обчислення (HPC)	Наукові розрахунки, великі візуалізації (космічна, кліматична модель; масиви LIDAR); ітеративний аналіз даних.	Розподіл даних і обчислень між багатьма вузлами. Наприклад, HadoopViz використовує MapReduce для генерації масштабних растрових візуалізацій (гігапіксельних теплокарт). GPU-розпаралелювання (кілька

		відеокарт, CUDA) застосовується для швидкого рендерингу об’ємів і потоків даних. Об’єми даних нарізають на блоки, обробляються паралельно, результати з’єднуються.
Хмарні технології	Будь-які сфери (урбаністичне планування, екологія, логістика) з великими просторовими даними; реальне час (IoT, моніторинг).	Геодані зберігаються та обробляються в дата-центрах (Big Data платформи – AWS, Google Cloud тощо). Забезпечується горизонтальне масштабування і стрімінг даних (наприклад, швидкий доступ до тайлів чи 3D-моделей). Хмарні сервіси забезпечують паралельні запити і потоки даних, що дає пришвидшення в десятки разів. Використовуються покрокова обробка запитів, оптимізоване індексування та «pay-per-use» модель.
3D-GIS / візуалізація 3D-даних	Продуктивність залежить від ГПУ та оптимізацій.	Урбаністичне моделювання, цифрові двійники, археологія, інженерія (підземні мережі, будівлі). Обробка хмарних/рельєфних моделей, сканів LIDAR тощо. Використовуються ті ж принципи LOD, тайлів та зовнішньої пам’яті. Наприклад, сцену розбивають на кубічні або квадродеревні сегменти (octree), кожен має свою набір LOD. Компресія (Draco, KTX) і стрімінг текстур знижують об’єм переданої інформації.
VR/AR-візуалізація	Освітні та презентаційні 3D-сцени, тренінги, віртуальні тури, військове і промислове навчання.	Інтерактивні об’ємні візуалізації з повним зануренням. Дані GIS перетворюються на VR/AR-об’єкт з можливістю «пролітати» або накладати 3D-дані на реальний простір. Зазвичай використовуються ігрові рушії (Unreal, Unity) і попередньо спрощені моделі. VR дає глибоке занурення (360° огляд), але вимагає багато пам’яті і кадрів на секунду, тоді як AR накладає.

Як видно з табл. 1.2, кожен клас методів має свої сильні і слабкі сторони. Методи узагальнення (кластеризація, KDE/heatmap, LOD) швидко відображають дані та зменшують «шум», але жертвують деталями. Потокова передача та байтові формати тайлів (WMS/WFS, 3D Tiles) ефективно підтримують роботу з обмеженим зв’язком (мобільні пристрої), проте вимагають розробки механізмів попереднього отримання і кешування. Розподілені і хмарні рішення забезпечують майже необмежену масштабованість: вони дають можливість «підняти» потужності практично необмежено, прискорюючи обчислення і візуалізацію в десятки разів. Втім, це породжує складність інфраструктури, витрати на зберігання/трафік та мережеві затримки. Спеціалізовані 3D- та VR/AR-підходи відкривають нові можливості (тривимірна аналітика, занурення у дані), але на даний час обмежені продуктивністю й можливостями апаратури.

Сучасні підходи до візуалізації великих обсягів просторових даних у геоінформаційних системах базуються на принципах адаптивності, модульності, обчислювальної близькості до джерела даних та інтеграції аналітики у візуалізаційні процеси. Ці принципи реалізуються через застосування таких технологій і методологій, які підвищують ефективність обробки, аналітики та прийняття рішень.

Принцип адаптивності полягає в динамічному підлаштуванні візуалізаційних параметрів до змісту даних і поведінки користувача. Це реалізується, наприклад, через автоматичне перемикання між різними типами візуального представлення залежно від щільності об'єктів у поточній області карти: теплокарта для великої кількості точок, маркери - для меншої кількості, кластери - для середньої щільності. Такий підхід дозволяє мінімізувати візуальне перевантаження і одночасно зберігати інформативність.

Принцип модульності реалізується в побудові візуалізаційних систем на базі окремих незалежних компонентів: шарів даних, модулів інтерфейсу, аналітичних обчислювачів. Це дозволяє ефективно масштабувати систему, підтримувати різні режими перегляду (наприклад, режим аналізу часових змін, режим порівняння кількох наборів даних), а також легко інтегрувати нові джерела без необхідності переробки всієї архітектури.

Обчислювальна близькість до джерела даних (data locality) передбачає, що обробка та агрегація просторових даних виконується максимально близько до місця їх зберігання. Це суттєво знижує затрати на передачу інформації мережею, особливо коли мова йде про аналіз георадарних, супутникових або сенсорних даних у режимі реального часу. Такий підхід реалізується через вбудовані аналітичні сервіси в геопросторових сховищах або використання edge-аналітики у хмарно-периферійних гібридних системах.

Окрему роль відіграє концепція інтегрованої аналітики, коли візуалізація є не лише результатом, а й інструментом аналізу. Це досягається за допомогою інтерактивних панелей, вбудованих фільтрів, часових повзунків і дашбордів, які дозволяють користувачеві одночасно бачити результат, змінювати параметри вхідних даних та аналізувати альтернативні сценарії. Таке інтерактивне середовище дозволяє скоротити час аналізу, зменшити кількість повторних запитів до бази даних та приймати більш обґрунтовані управлінські рішення.

У підсумку, сучасні принципи та практики візуалізації спрямовані на інтеграцію процесу обробки даних з його візуальним відображенням, що не лише економить обчислювальні ресурси, але й наближає аналітичну інформацію до кінцевого користувача, роблячи геоінформаційні системи значно ефективнішими в умовах зростаючих обсягів просторових даних. Вибір підходу залежить від задачі: часто використовуються комбіновані рішення (наприклад, LOD + паралельний рендеринг у хмарі) для оптимізації продуктивності при роботі з надвеликими ГІС-даними.

1.2 Порівняльний аналіз програмних засобів що використовують візуалізацію великих обсягів просторових даних у геоінформаційних системах

Сучасні веб-бібліотеки для картографії, такі як Leaflet, Mapbox GL JS та OpenLayers, широко застосовуються для інтерактивної візуалізації геопросторових даних у браузері. Зокрема, Mapbox GL JS – клієнтська JavaScript-бібліотека з підтримкою WebGL, оптимізована для плавного рендерингу великих обсягів даних (табл. 1.3.). Вона конвертує GeoJSON у векторні тайли «на льоту» у

браузері. У демонстраційному кейсі для візуалізації даних перепису населення США було згенеровано 17 ГБ GeoJSON, перетворено їх на векторні тайли за допомогою Tippecanoe та візуалізовано як 3D-екструдовані фігури за щільністю населення. Таким чином Mapbox GL JS підтримує як 2D-візуалізації (звичайні карти), так і 3D (екструзії поверхонь). Але це рішення переважно працює в онлайн-режимі через API Mapbox; офлайн-режим можливий лише з використанням мобільних SDK для Android/iOS (які дозволяють завантажувати регіони карти для роботи без зв'язку).

Таблиця 1.3 - Порівняльна характеристика деяких програмних засобів для візуалізації великих обсягів просторових даних у геоінформаційних системах

Критерій	Mapbox GL JS	CesiumJS	ArcGIS Pro
Тип візуалізації	2D/3D (екструзія)	3D (глобус), 2D/2.5D	2D/3D (4D)
Онлайн/офлайн	Переважно онлайн (API Mapbox)	Може бути офлайн (локальні тайли)	Повністю офлайн (десктоп)
Ліцензування	Комерційна (freemium)	Відкрита (Apache 2.0)	Комерційна (пропрієтарна)
Інтеграція з БД	Так (векторні тайли з PostGIS тощо)	Так (3D Tiles, дані GIS)	Так (geodatabase, PostGIS тощо)
Потоки/реальний час	Є (оновлення шарів, WebSockets)	Є (3D Tiles потоки, CZML)	Обмежено (ArcGIS GeoEvent)
Продуктивність	Висока (WebGL)	Висока (GPU, але залежить від сцени)	Висока (64-bit, GPU)
Документація	Докладна (Mapbox Docs)	Детальна (Cesium Docs)	Розгорнута (Esri Help)
Розширюваність	Плагіни, SDK, стилі	API, плагіни, відкриті формати	Python (Arcpy), SDK

Одним із провідних інструментів для веб-візуалізації великих просторових даних є бібліотека Mapbox GL JS [19,20], яка реалізує підхід до побудови

інтерактивних карт на основі векторної графіки, що рендериться безпосередньо у браузері з використанням WebGL. Такий принцип дає змогу не лише забезпечити плавне масштабування та обертання карти, але й дозволяє відображати складні геопросторові об'єкти з високим рівнем деталізації без необхідності передачі великих растрових зображень.

Однією з визначальних архітектурних переваг Mapbox GL JS є підтримка «style specification» – формального JSON-опису шару, який дозволяє динамічно змінювати стиль, джерела даних і поведінку карти без перезавантаження сторінки. Це забезпечує модульність, відокремлення даних від візуалізації та спрощує процес інтеграції з зовнішніми аналітичними сервісами. Крім того, у межах одного проекту можлива одночасна інтеграція кількох джерел: векторних тайлів, GeoJSON, растрових шарів, WMS/WMTS-сервісів тощо.

З точки зору продуктивності, особливо при роботі з великими обсягами даних, Mapbox GL JS реалізує механізм «source clustering», що дозволяє автоматично агрегувати точки у кластери на стороні клієнта, враховуючи поточний масштаб карти та географічну щільність об'єктів. Цей підхід знижує візуальне навантаження на інтерфейс і суттєво оптимізує швидкодію при рендерингу багатотисячних наборів точок. При цьому кожен кластер можна налаштувати для відображення додаткової інформації, такої як середнє значення певного показника або сума атрибутів усередині кластера, що наближає бібліотеку до функціоналу тематичних аналітичних карт.

Окрему увагу заслуговує можливість прив'язки Mapbox GL JS до інтерактивних подій, що дозволяє будувати динамічні сценарії взаємодії з даними: фільтрація об’єктів за атрибутами, зміна стилів у відповідь на поведінку користувача, реалізація режимів видимості шарів залежно від масштабу або тематичних запитів. Така поведінка забезпечується через систему подій та API-методів (наприклад, on(), setFilter(), setPaintProperty()), що відкриває широкі можливості для побудови адаптивних інтерфейсів візуалізації.

Із програмної точки зору, Mapbox GL JS повністю підтримує архітектуру MVVM (Model-View-ViewModel) у зв'язці з сучасними фреймворками, такими як React або Vue.js, що дозволяє інтегрувати карти в більші клієнтські системи. Крім

того, бібліотека має вбудовану підтримку високої щільності екранів (HiDPI), автоматичне масштабування шрифтів, GPU-акселерацію для маркерів та ліній, а також підтримку розширених форматів шрифтів (наприклад, SDF).

З технічного боку, розробники мають можливість завантажувати та використовувати кастомізовані векторні тайли, згенеровані за допомогою інструментів типу Tippecanoe або Tileserver GL. Це дозволяє забезпечити незалежність від хмарної інфраструктури Mapbox, а також уникнути ліцензійних обмежень при великомасштабному застосуванні або в автономних середовищах. Водночас підтримується «lazy loading» тайлів, що дозволяє завантажувати лише ті частини карти, які дійсно потрібні користувачу в конкретний момент часу.

Таким чином, Mapbox GL JS [19,20] надає розробникам комплексне середовище для створення продуктивних, динамічних та масштабованих картографічних рішень. Його гнучкість у роботі з різними форматами даних, ефективна обробка великих обсягів інформації на клієнті, а також модульність та можливість розширення через API робить його одним з найбільш універсальних інструментів для реалізації веб-ГІС із високим рівнем інтерактивності та продуктивності.

У Mapbox GL JS застосовується комерційна ліцензія з безкоштовним початковим планом (платні щомісячні тарифи), хоча сама бібліотека колись була відкритою. Переваги Mapbox – висока продуктивність візуалізації, гнучка стилізація та розширена документація, обмеження – залежність від сервісів Mapbox, ліміти на офлайн-тайл-пакети та складність відображення надзвичайно великих моделей візуалізації без попередньої оптимізації даних.

Одним із найбільш технологічно просунутих інструментів для тривимірної візуалізації просторових даних у веб-середовищі є CesiumJS [21] - JavaScript-бібліотека з відкритим вихідним кодом, спеціалізована на рендерингу масштабованих сцен у форматі віртуального глобуса. На відміну від традиційних картографічних засобів, CesiumJS ґрунтується на моделі еліпсоїда ВГС-84, що дає змогу виконувати візуалізацію на геодезично коректному глобусі з високою просторовою точністю. Завдяки цьому платформа може ефективно

використовуватися для завдань аерокосмічного моделювання, урбаністики, військового моделювання та супутникової навігації.

Ключовою особливістю архітектури CesiumJS є її енергозалежна структура рендерингу на основі тайлів, яка реалізує ієрархічне подання просторових об’єктів за допомогою формату 3D Tiles. Цей формат, що підтримує трансмісивне (потокове) завантаження геометрії, текстур і метаданих, дає змогу забезпечити плавне масштабування та деталізацію 3D-сцен у міру наближення камери. Така реалізація дозволяє відображати навіть мільйони об’єктів на глобальному рівні без суттєвого зниження продуктивності браузера, оскільки дані підвантажуються та рендеряться інкрементально.

CesiumJS також підтримує асинхронну побудову сцени з використанням web workers, що дозволяє розвантажити головний потік виконання та забезпечити безперервну взаємодію з користувачем. При цьому геометричні об’єкти можуть змінювати свій вигляд або позицію в реальному часі, що особливо актуально для задач візуалізації рухомих об’єктів: дронів, супутників, транспортних засобів. Завдяки підтримці специфікації CZML (Cesium Language), система дозволяє описувати темпоральні атрибути об’єктів: часові інтервали, траєкторії, анімацію тощо.

Значною перевагою CesiumJS є також можливість використання власного механізму висотного розрізнення (terrain model), який базується на даних цифрової моделі рельєфу (DEM) з різною щільністю. Це дозволяє не лише покращити реалізм відображення місцевості, а й виконувати точні розрахунки видимості, ліній огляду, профілів поверхні, що є критично важливим для інженерного та архітектурного моделювання.

Крім геометричних моделей, CesiumJS також дозволяє інтегрувати растрові дані у вигляді картографічних проекцій (наприклад, WMS/WMTS-шари), підтримує накладення векторних об’єктів з GeoJSON, KML та TopoJSON, а також працює з координатними системами за допомогою вбудованої бібліотеки Transforms. Ці можливості роблять CesiumJS придатним не лише для 3D-відображення, але і для повноцінної реалізації складних геоаналітичних інтерфейсів.

У технологічному аспекті бібліотека розроблена з урахуванням найсучасніших стандартів web-програмування: вона використовує WebGL для графічного рендерингу, підтримує GLSL-шейдери для кастомізації матеріалів та освітлення, а також має повну інтеграцію з типовими інструментами розробки (наприклад, NPM, TypeScript, React, Vue).

Важливим є й той факт, що CesiumJS має активну розробницьку спільноту та регулярні оновлення, а також може бути використаний як із відкритими картографічними сервісами (OpenStreetMap, Bing Maps, MapTiler), так і з приватними векторними або 3D-ресурсами. У багатьох практичних реалізаціях, таких як цифрові двійники міст, моделювання в Smart City, візуалізація маршрутів літальних апаратів, саме CesiumJS демонструє кращу масштабованість у порівнянні з традиційними 2D-картографічними фреймворками.

CesiumJS [21] – це веб-бібліотека з відкритим кодом для створення тривимірних глобусів і карт. Як зазначено на офіційному сайті, CesiumJS забезпечує «найкращу можливу продуктивність, точність та візуальну якість» у 3D-картографії. Вона спроєктована для масштабних наборів даних і підтримує відкриті формати (зокрема 3D Tiles для потокової передачі об’ємних 3D-моделей будівель та хмар точок, glTF для 3D-моделей, CZML/GeoJSON для об’єктів тощо). CesiumJS може працювати у трьох режимах перегляду (3D, 2D і «Колумб»), що надає гнучкість у представленні даних. Ця бібліотека дозволяє створювати інтерактивні 3D-сцени (наприклад, моделювання міст, рельєфу, реального часу політів дронів), але вимагає потужного графічного процесора й більше ресурсів на великі 3D-набори. CesiumJS розповсюджується за відкритою ліцензією Apache 2.0 і має велику документацію. Серед обмежень – потреба у високопродуктивному GPU та складність попередньої обробки даних для 3D-візуалізації (наприклад, генерації 3D Tiles). Як відзначено у навчальних матеріалах, саме CesiumJS часто називають «фреймворком для 3D-картографії в браузері».

Таким чином, CesiumJS [21] є високопродуктивною, багатофункціональною та масштабованою платформою для побудови 3D-ГІС у браузері, яка реалізує принципи потокового завантаження, просторової узагальненості та геодезичної

точності, що є критичним для сучасної візуалізації великих обсягів просторових даних у режимі реального часу.

Серед сучасних десктопних інструментів для геоінформаційного моделювання особливе місце займає ArcGIS Pro [22] - флагманський продукт компанії Esri, що об'єднує високопродуктивне візуалізаційне ядро, багатий набір аналітичних функцій та інтеграцію з корпоративними геопросторовими сервісами. Його архітектура побудована за принципом об'єктно-орієнтованої моделі даних, що дозволяє працювати з просторовими об'єктами різної природи (точками, лініями, полігонами, сітками, растровими даними, тривимірними об’єктами) у єдиному середовищі проєкту.

Однією з ключових переваг ArcGIS Pro є підтримка мультискейл-візуалізації, яка забезпечується за рахунок так званих symbol layer drawing rules - механізмів, що дозволяють задавати різні правила рендерингу об’єктів залежно від масштабу, атрибутів або стану карти. Це дозволяє значно знизити візуальну складність при перегляді великих обсягів просторових даних, а також оптимізує навантаження на систему, оскільки рендеринг об’єктів адаптується до контексту.

ArcGIS Pro [22] також забезпечує глибоку інтеграцію з аналітичними модулями ArcGIS Spatial Analyst, 3D Analyst та Network Analyst, що дозволяє виконувати просторову інтерполяцію, аналіз видимості, побудову ізоліній, моделювання потоків та рельєфів, побудову транспортних мереж з урахуванням вагових коефіцієнтів і часових зон. Завдяки цьому платформа придатна як для класичного картографування, так і для сценаріїв, що включають просторову оптимізацію, планування або прогнозування.

Особливістю ArcGIS Pro є вбудований багатовіконний просторовий редактор, що підтримує паралельну роботу з кількома компоновками карти, таблицями атрибутів, графіками та сценами. Це дає змогу розробляти комплексні аналітичні звіти, у яких кожне подання має власний контекст і метод візуалізації. Наприклад, можна одночасно переглядати 2D-карту землекористування, 3D-сцену рельєфу та часову шкалу зміни лісового покриву, що істотно розширює можливості аналітичного синтезу даних.

З точки зору інженерної кастомізації, ArcGIS Pro реалізує розширювану платформу через Python API (ArcPy), що дозволяє автоматизувати обробку даних, візуалізацію та генерацію звітів. Крім того, можливе створення доповнень (Add-Ins) за допомогою .NET SDK, що дозволяє реалізувати спеціалізовані інструменти, інтеграцію з базами даних, або організувати обмін даними з зовнішніми інформаційними системами.

ArcGIS Pro також активно застосовується в середовищах інфраструктури просторових даних (SDI), де виконує роль клієнта і публікатора картографічної інформації на порталах ArcGIS Enterprise або ArcGIS Online. У цьому контексті програмне забезпечення підтримує створення веб-карт, веб-сцен, шарів обслуговування (Feature Layers, Map Services), і їхню публікацію безпосередньо з інтерфейсу користувача. Такий механізм дозволяє здійснювати повний цикл: від локального аналізу - до централізованої візуалізації та колективної роботи з даними.

Однією з відмінних рис ArcGIS Pro є можливість роботи з 4D-даними, що включає часовий вимір - наприклад, моніторинг зміни висоти рослинності за періодами, сезонні моделі мобільності населення або прогнозні дані гідрологічної динаміки. Застосування такої функціональності дозволяє реалізовувати сценарії адаптивного реагування та довгострокового просторового планування.

З погляду продуктивності, ArcGIS Pro використовує 64-бітну архітектуру з повноцінним GPU-прискоренням. Завдяки цьому забезпечується висока швидкодія при візуалізації важких наборів - наприклад, супутникових знімків з роздільною здатністю понад 1 м/піксель, тривимірних моделей міст (CityGML), лідара та гібридних шарів. Підтримується робота з багатогігабайтними геобазами даних (FileGDB, EnterpriseGDB) та прямий доступ до хмарних сервісів (Living Atlas, Copernicus Data Hub).

ArcGIS Pro [22] – провідне комерційне десктопне ПЗ компанії Esri для професійної ГІС-аналітики. Згідно з описом на сайті виробника, ArcGIS Pro дозволяє генерувати «захоплюючі 2D, 3D та 4D-візуалізації» та проводить розширений геоаналітичний аналіз. Цей інструмент підтримує роботу з величезними растровими та векторними наборами даних (супутникові знімки,

лідара, карти висот тощо), використовуючи 64-розрядну архітектуру і апаратне прискорення GPU для швидкого рендерингу. ArcGIS Pro має глибокі символогічні можливості та «проєкти візуального конструювання карт». Програма нативно інтегрується з геобазами даних Esri (File/Enterprise Geodatabase), PostGIS, OGC-сервісами та багатьма іншими джерелами через ArcGIS Platform. Переваги ArcGIS Pro – висока продуктивність на потужному обладнанні, широкий спектр аналітичних інструментів і надійна технічна підтримка. Недоліком є закрита пропрієтарна ліцензія (потреба в оплаті підписки) і велика вага самої програми. Архітектуру Pro можна розширювати через Python-скрипти (Arcpy) та розробку доповнень (add-in на .NET/C#). QGIS – безкоштовна десктопна ГІС з відкритим вихідним кодом, що є популярною альтернативою ArcGIS.

Таким чином, ArcGIS Pro [22] виступає не лише як інструмент для візуалізації просторових даних, але й як інтегрована аналітична система для моделювання, прогнозування та управління територіями. Її переваги включають високу масштабованість, модульну розширюваність, підтримку інтероперабельності та багатовимірного аналізу, що робить її придатною для використання як у дослідницьких, так і у прикладних проєктах державного, комерційного й академічного рівнів.

QGIS (раніше відомий як Quantum GIS) є одним із найпоширеніших засобів відкритої ГІС-аналітики, що поєднує гнучкість платформи з широкими можливостями для просторового аналізу, картографування та візуалізації [23]. В основі філософії QGIS лежить концепція повністю відкритої екосистеми з підтримкою найважливіших стандартів Open Geospatial Consortium (OGC), що дозволяє використовувати інструмент як основу для побудови інтегрованих рішень у різних галузях – від екологічного моніторингу до урбаністики та земельного кадастру.

Однією з ключових функціональних переваг QGIS є підтримка багатоформатного вводу та виводу просторових даних [23]. Завдяки інтеграції з бібліотекою GDAL/OGR, система може працювати з понад 150 форматами даних, включно з GeoPackage, GeoTIFF, NetCDF, GRIB, KML, GML, Shapefile, PostGIS, SpatiaLite, MSSQL Spatial, WFS та іншими. Така універсальність дозволяє

використовувати QGIS як центральну платформу для агрегування, попередньої обробки та візуального представлення складних та неоднорідних геоданих.

Особливістю QGIS є інтеграція з мовою Python через консоль і API PyQGIS, що надає можливість створення скриптів, макросів, власних алгоритмів обробки даних, а також розробки розширень і надбудов. За допомогою фреймворку Processing Toolbox користувачі мають доступ до сотень алгоритмів просторового аналізу (буферизація, перетини, статистичне зонування, картограма, щільність тощо), які можна виконувати як поодинці, так і в автоматизованих ланцюжках (workflow).

QGIS активно підтримує концепцію мультимасштабного картографування, зокрема реалізує стилізацію шарів з урахуванням масштабу перегляду, умовного форматування символів, а також динамічне увімкнення/вимкнення шарів або міток залежно від рівня деталізації. Крім того, користувач має змогу реалізувати контекстну візуалізацію - коли вигляд об'єкта залежить від його атрибутів, географічного розташування або належності до класифікаційної групи. Це є основою для побудови складних тематичних карт, heatmap-візуалізацій, а також векторних анімованих шарів.

У рамках 3D-візуалізації QGIS надає можливість створення тривимірних сцен, які базуються на цифрових моделях рельєфу (DEM), а також підключають текстуровані будівлі або об’єкти, імпортовані з CityGML, OBJ чи COLLADA. Рендеринг здійснюється за допомогою OpenGL, що дозволяє забезпечити базовий рівень інтерактивності при переміщенні, обертанні та масштабуванні сцен. Візуалізаційні параметри (наприклад, прозорість, тінь, освітлення) можна змінювати в реальному часі, що сприяє формуванню більш реалістичних представлень [23].

Ще однією сильною стороною QGIS є система плагінів, яка постійно розширюється завдяки активній спільноті розробників. Серед найвідоміших розширень можна виокремити:

- TimeManager – для візуалізації темпоральних змін;
- QGIS2Web – генерація інтерактивних веб-карт (Leaflet, OpenLayers);
- Dzetsaka – реалізація класифікації з використанням машинного навчання;

- Profile Tool – побудова профілів рельєфу;

- Semi-Automatic Classification Plugin – аналіз супутникових знімків.

QGIS також підтримує кластеризацію та генералізацію об’єктів, що дозволяє зменшувати об’єм візуальних елементів на великих масштабах. Завдяки цьому система може працювати з десятками тисяч об’єктів на карті без істотного зниження продуктивності, особливо при використанні оптимізованих структур даних (наприклад, spatial indexing, geometry simplification).

З погляду інтероперабельності, QGIS [24] може виступати клієнтом для зовнішніх джерел даних, зокрема сервісів WMS/WFS/WCS/WMTS, ArcGIS Map Services, GeoServer, а також Web Feature Service з підтримкою фільтрації за атрибутами. Крім того, можлива повна інтеграція з базами даних PostgreSQL/PostGIS, що дозволяє виконувати SQL-запити до просторових таблиць безпосередньо з інтерфейсу QGIS.

QGIS підтримує 2D-картографію та містить модуль для 3D-візуалізації (3D-огляд сцени з рендером поверхонь і моделей). На офіційному сайті підкреслюється «підтримка різних джерел та форматів даних» (PostGIS, SpatiaLite, GeoPackage, ESRI Shapefile, GeoTIFF, WMS/WMTS тощо), що робить QGIS дуже гнучким у інтеграції (табл. 1.4). Програмний продукт поширюється під ліцензією GNU GPLv2+, тобто безкоштовний для будь-якого використання.

Таким чином, QGIS [23] - це не просто інструмент для візуалізації, а повноцінна ГІС-платформа з відкритою архітектурою, яка дозволяє реалізовувати завдання будь-якої складності: від створення базових карт до реалізації глибокого просторового аналізу та підключення до високопродуктивних геосерверів. Його модульність, відкритість, сумісність зі стандартами та активна міжнародна спільнота роблять його потужним засобом як для наукових досліджень, так і для практичного застосування в управлінні просторовими ресурсами.

Серед мобільних ГІС-додатків прикладами є ArcGIS Field Maps (раніше ArcGIS Collector) і QField. ArcGIS Field Maps [24] – мобільний клієнт Esri для Android/iOS, призначений для роботи з картою і збору даних у полі. Цей додаток дозволяє переглядати багатошарову картографію офлайн (завантажувати спеціальні області карти або мобільні пакети) і синхронізувати з ArcGIS

Online/Enterprise. За словами розробників, Field Maps на смартфонах демонструє «найсучасніші можливості перегляду карт» аналогічно десктопу. Перевагами Field Maps є надійність, офіційна підтримка, передова тематика інтерфейсу та можливість спільного використання даних у реальному часі (наприклад, відстеження місцеположення співробітників). Серед обмежень – необхідність підписки на ArcGIS Online/Enterprise, закритий код, а також обмеження мобільних пристроїв (пам'ять, продуктивність).

QField [25] – мобільний ГІС-додаток від спільноти QGIS для Android/iOS, оптимізований для польових робіт. Він безкоштовний і відкритий, надає користувачеві «повну відповідність символіці QGIS» і високу точність позиціювання на мобільному пристрої. QField працює з локальними даними (GeoPackage, KML, GPX, растр тощо) і підтримує геофенсінг, зйомку фото/відео з геоприв'язкою. Через сервіс QFieldCloud можна організувати синхронізацію змін у реальному часі. Сильні сторони QField – відсутність ліцензійних витрат, гнучкість у налаштуванні під існуючі проєкти QGIS і можливість збирання даних у складних умовах. Обмеження: продуктивність обмежена мобільним пристроєм, менша кількість спеціалізованих інструментів (наприклад, для аналізу рельєфу) порівняно з десктопним софтом.

Таким чином, проведений порівняльний аналіз показує, що вибір інструменту значною мірою залежить від характеру задачі. Веб-бібліотеки Mapbox GL JS і CesiumJS відмінно підходять для інтерактивного відображення великих картографічних даних в інтернеті, причому CesiumJS забезпечує глибоку 3D-візуалізацію, а Mapbox робить акцент на продуктивності WebGL і кастомізації дизайну. Десктопні системи ArcGIS Pro і QGIS мають повний набір інструментів для картографування і аналізу (у тому числі в 3D), але ArcGIS Pro є комерційним та закритим рішенням із максимальною продуктивністю на потужному обладнанні, тоді як QGIS – безкоштовним і відкритим з більш обмеженими ресурсами. Для мобільних сценаріїв ArcGIS Field Maps пропонує інтегровану екосистему Esri для збору даних і офлайн-карт, а QField забезпечує польову роботу з геоданими в умовах мінімальних витрат та прив'язкою до проектів QGIS. Кожен із розглянутих інструментів має свої переваги (швидкість,

функціональність, відкритість) та недоліки (вартість, обмеження платформи, продуктивність при дуже великих даних), що слід враховувати при виборі рішення для візуалізації великих обсягів просторових даних.

1. 3 Постановка задач дослідження

У сучасних геоінформаційних системах (ГІС) дедалі частіше виникає необхідність обробки та візуалізації надвеликих обсягів просторових даних, що представлені у форматі GeoJSON і можуть змінюватися динамічно. Зі зростанням деталізації джерел (наприклад, супутникових або сенсорних) та щільності об’єктів, які потрібно відобразити на карті, традиційні інтерфейси візуалізації перестають забезпечувати належну продуктивність. Зокрема, це проявляється у затримках при масштабуванні, переміщенні карти, завантаженні маркерів або оновленні інформації в реальному часі.

Цільова аудиторія розроблюваного програмного засобу повинні складати інженери та аналітики, які працюють із великими масивами геопросторової інформації у веб-середовищі. Для них надзвичайно важливою є висока швидкість відображення, надійна інтерактивність, підтримка зміни масштабу, ієрархічної кластеризації та актуальність даних без потреби перезавантаження сторінки.

У зв’язку з цим формулюється основна задача дослідження яка полягає у необхідності підвищення ефективності візуалізації великих обсягів просторових даних шляхом інтеграції спеціалізованих методів кешування, кластеризації та адаптивного рендерингу у веб-ГІС-інтерфейсі. Тому, основна мета дослідження полягає в розробці програмного засобу, що дозволяє ефективно візуалізувати великі обсяги просторових даних з урахуванням динамічного оновлення інформації, забезпечуючи високу продуктивність, масштабованість і зручність користування у веб-середовищі.

Для досягнення мети необхідно вирішити такі підзадачі:

- провести аналіз існуючих інструментів і технологій візуалізації просторових даних;

- визначити і реалізувати механізм кешування тайлів на основі Google Maps API з урахуванням принципів WebGL і віддаленого завантаження;

- реалізувати кластеризацію маркерів з урахуванням просторової щільності та поточного масштабу перегляду;

- розробити архітектуру програмного комплексу на основі фронтенду (JavaScript, HTML, CSS), бекенду (Java, Spring) та бази даних (MySQL);

- забезпечити гнучке оновлення даних GeoJSON без втрати швидкодії;

- провести тестування з метою оцінки продуктивності системи на великих обсягах даних (від 10 тис. до 1 млн просторових об’єктів).

Очікуваний результат - веб-орієнтований програмний засіб з високою швидкістю завантаження і оновлення просторових шарів, підтримкою адаптивної візуалізації, офлайн-тайлів і масштабованого рендерингу.

До обмежень і вимог до системи належать:

- підтримка масштабованості до рівня міста, області або країни;

- чутливість до пропускної здатності каналу користувача (мінімізація обсягу передаваних даних);

- сумісність із сучасними браузерами (Chrome, Firefox, Edge);

- відкритий API для подальшого розширення (наприклад, аналітичними модулями).

Таким чином, у межах даної роботи необхідно реалізовати повний цикл: від аналізу проблеми до проєктування, реалізації та тестування програмного засобу, що дозволить впроваджувати його у прикладні задачі просторового аналізу та інтерактивної візуалізації.

2 ДОСЛІДЖЕННЯ МЕТОДІВ ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАСОБУ

2.1 Запропонований метод кешування даних на основі компоненту тайлів Google Maps API

У сучасних геоінформаційних системах (ГІС), які працюють з великими обсягами просторових даних, ефективність візуалізації картографічної інформації безпосередньо впливає на зручність користувача та загальну продуктивність програмного забезпечення. Зокрема, при використанні хмарних або онлайн-сервісів, таких як Google Maps API, виникає необхідність постійного отримання тайлів (map tiles) з віддаленого сервера, що супроводжується затримками, зумовленими мережею, обмеженнями пропускної здатності та навантаженням на зовнішні джерела даних.

Тому, традиційна модель обробки картографічних шарів передбачає динамічне завантаження лише тих фрагментів карти, які відповідають поточному положенню камери. Проте при масштабуванні, навігації або переміщенні області перегляду тайли, які вже були завантажені, часто запитуються повторно, що призводить до дублювання трафіку та зниження ефективності. Така поведінка є критичною при роботі з великими наборами растрових та векторних даних, особливо в умовах обмеженого або нестабільного інтернет-з'єднання. Вирішенням цієї проблеми є запровадження методу кешування тайлів, який полягає у попередньому завантаженні картографічних фрагментів та збереженні їх у локальному сховищі (файловій системі, браузерному кеші або у спеціалізованих базах даних, таких як IndexedDB). Такий підхід дозволяє:

- значно зменшити кількість повторних звернень до серверів картографічного постачальника;
- забезпечити миттєве відображення вже відвіданих областей карти;
- скоротити обсяг мережевого трафіку;
- підвищити загальну чутливість інтерфейсу користувача до навігаційних дій;
- закласти основу для автономного або офлайн-функціонування ГІС-додатку.

В межах цієї роботи запропоновано та реалізовано метод кешування картографічних шарів з використанням Java/WebGL, який інтегрується з компонентами Google Maps API. Особливістю методу є комбінування попереднього завантаження тайлів (pre-fetching), динамічного кешування за запитом, та механізму адаптивного оновлення вмісту кешу при виявленні змін на сервері.

Таким чином, основною задачею дослідження є:

- формалізація алгоритму кешування просторових тайлів;

- інтеграція кешування у середовище Java/WebGL;

- моделювання впливу кешування на продуктивність системи;

- порівняння ефективності різних механізмів локального збереження даних (IndexedDB, LocalStorage, файлова система).

Надалі, розглянемо аналіз особливостей використання Google Maps API та принципам роботи з тайлами, які є базовими об'єктами для реалізації кешування в контексті WebGL-візуалізації.

Так, особливості використання технології Google Maps API та тайлів полягають в наступному. Google Maps API є одним із найпопулярніших інструментів інтеграції картографічних сервісів у вебзастосунки, який надає доступ до динамічних карт, сервісів геокодування, маршрутизації, пошуку місць та візуалізації просторових об'єктів. У контексті візуалізації великих обсягів даних особливе значення має компонент тайлів (map tiles), які є базовими одиницями побудови картографічного полотна.

Тайли - це фрагменти карти фіксованого розміру (зазвичай 256×256 пікселів), які відповідають певним координатам у сітці розбиття (tile grid) на певному рівні масштабування (zoom level). Кожен тайл має координати виду (x, y, z), де:

x, y - позиція фрагмента у сітці;

z - рівень масштабування, що визначає деталізацію.

Таким чином Google Maps використовує модель XYZ-адресації тайлів, при якій кожне значення x і y визначається для конкретного z. Наприклад, координати тайлу x=1, y=2, z=3 відповідають певній області карти при масштабі 3. Це

дозволяє організувати просту структуру доступу до файлів, які можуть бути збережені у форматі PNG, JPEG або PBF (для векторних тайлів).

У Google Maps API підтримуються два основних типи картографічних фрагментів:

- растрові тайли (raster tiles), що є зображенням карти у вигляді готових PNG або JPEG-файлів;

- векторні тайли (vector tiles), що є набором геометричних примітивів (лінії, полігони, пункти), що рендеряться на клієнтській стороні з використанням Canvas або WebGL.

При цьому, загальні особливості WebGL у рендерингу полягають в наступному. WebGL (Web Graphics Library) є вебтехнологією для прискореного рендерингу графіки з використанням графічного процесора (GPU), що базується на OpenGL ES. Вона дозволяє виконувати обробку великої кількості векторних або растрових об’єктів у реальному часі, що особливо важливо при візуалізації тисяч просторових елементів, таких як дороги, будівлі, контури зон.

У зв’язку з цим WebGL широко застосовується для відображення:

- векторних тайлів з ефективним масштабуванням та стилізацією;

- растрових шарів з можливістю пост-обробки (наприклад, змін контрастності або прозорості);

- анімованих геооб’єктів або heatmap-візуалізацій.

При цьому, у середовищі Java можлива побудова web-клієнта на основі JavaFX WebView або обгортки браузера, що взаємодіє з JavaScript API Google Maps через JavaScript-інтерфейси. Для рендерингу просторових даних використовуються інтерфейси типу addTileOverlay() або addLayer() з подальшим обробленням через WebGL-шейдери.

Потік взаємодії з Google Maps API зазвичай включає наступні кроки:

- ініціалізацію карти у DOM-елементі;

- встановлення рівня масштабування та області перегляду;

- завантаження відповідних тайлів через HTTP-запити;

- візуалізацію тайлів через WebGL або Canvas.

Однак, на практиці існують певні проблеми. Так, без кешування кожне переміщення карти, зум або перезавантаження сторінки викликає повторні звернення до серверу Google, що ускладнює взаємодію при повільному або нестабільному з’єднанні, а також створює додаткове навантаження на клієнтський пристрій при відтворенні тих самих тайлів.

Зважаючи на це, інтеграція механізму кешування, що дозволяє зберігати завантажені тайли локально, значною мірою покращує загальну продуктивність. Такий механізм повинен бути адаптований як до особливостей Google Maps API (ліцензійні обмеження, кешування через URL), так і до можливостей Java/WebGL-інфраструктури.

Тому, запропонований метод кешування полягає в наступному. З метою підвищення продуктивності візуалізації великих обсягів просторових даних у геоінформаційній системі, що базується на використанні Google Maps API з Java/WebGL, запропоновано метод кешування, який полягає в попередньому завантаженні та збереженні тайлів (як растрових, так і векторних) у локальному сховищі. Реалізація такого механізму забезпечує уникнення повторних HTTP-запитів до сервера та мінімізує час рендерингу картографічних шарів.

Запропонований метод передбачає комбіноване використання трьох рівнів кешування:

- кеш браузера (LocalStorage), що використовується для швидкого доступу до найменших обсягів тайлів (наприклад, останні 10–15 переглянутих ділянок карти);

- індексове сховище (IndexedDB), що використовується для зберігання великих обсягів тайлів у структурованому вигляді з можливістю асинхронного доступу;

- файлова система (у Java-клієнті), що використовується при роботі поза браузером (наприклад, через JavaFX або standalone-застосунок), тайли зберігаються як файли у структурі cache/z/x/y.png.

Ці рівні працюють у режимі каскадного звернення: спочатку система перевіряє наявність тайлу в кеші, і тільки за його відсутності здійснює зовнішній запит до Google Maps API.

Також, для покращення інтерфейсної чутливості реалізовано механізм попереднього завантаження тайлів, що перебувають у безпосередньому оточенні області перегляду (viewport). Це дає змогу користувачеві без затримки переходити на сусідні ділянки карти. Формуються сітки тайлів за координатами x, y, z, які обчислюються залежно від параметрів камери з різними форматами кешованих даних (табл. 2.1).

Таблиця 2.1 - Формати кешованих даних, що використовувались в роботі

Тип шару	Формат збереження	Розмір (орієнтовно)	Обробка перед рендерингом
Растровий	.png, .jpg	~10–70 КБ	Безпосередньо в WebGL
Векторний	.pbf, .json	~2–10 КБ	Декодування + шейдери WebGL
Метадані	.json	<1 КБ	Розбір + логіка навігації

Окрім базового кешування, метод підтримує адаптивне оновлення тайлів:

- часове оновлення, де тайли, що збережені понад 24 години тому, автоматично перезапитуються;

- примусове оновлення, де при зміні теми карти або типу шару;

- оновлення за подією, де у випадку, якщо сервер повертає інформацію про зміни (наприклад, при інтеграції з GeoRSS або вебхуками).

Для реалізації тайлів у програмному коді на Java/WebGL було використано:

для збереження тайлів - localStorage.setItem(), indexedDB.open(), Files.write() (у Java);

для доступу до API - виклики new google.maps.ImageMapType() з індивідуалізованим getTileUrl(x, y, z);

для рендерингу - шейдери GLSL у WebGL через createProgram, createBuffer, drawArrays.

Для забезпечення високої продуктивності відображення просторових даних у системі було реалізовано алгоритм кешування картографічних тайлів (рис. 2.1), що підтримує каскадну перевірку кількох рівнів локального сховища та

можливість попереднього завантаження. Алгоритм забезпечує послідовне виконання логіки обробки запиту користувача з урахуванням ефективного повторного використання раніше завантажених ресурсів та охоплює наступні кроки:

- обчислення області перегляду. Визначаються координати меж viewport та рівень масштабування z.

- генерація координат тайлів. Розраховуються всі значення x, y тайлів, що потрапляють у область.

- перевірка кешу. Для кожного тайлу виконується перевірка у LocalStorage / IndexedDB / файловій системі.

- отримання тайлів. Якщо тайл відсутній у кеші - він запитується з Google Maps API.

- збереження тайлів. Отримані зображення або векторні дані зберігаються у відповідному сховищі.

- рендеринг. Всі тайли, незалежно від джерела, рендеряться через WebGL-движок.

При цьому, виконується ініціація запиту карти. Після завантаження карти або зміни області перегляду виконується розрахунок координат тайлів для поточного рівня масштабування z. Так, результати тестування циклічної перевірки тайлів занесена до табл. 2.2.

Так, для кожного тайлу (визначеного координатами x, y, z) виконуються наступні дії:

1. Система послідовно перевіряє наявність у в різних типах сховищ як звичайний кеш браузера LocalStorage, структурований кеш у IndexedDB, файлова система (у Java-застосунках).

2. Завантаження даних з кешу. Якщо тайл знайдено - він рендериться на мапі без звернення до Google Maps API.

3. Отримання даних з сервера. У разі відсутності у кеші тайл завантажується з Google Maps API, кешується у відповідному сховищі, після чого рендериться.

4. Оновлення за визначеною умовою. Так, якщо тайл у кеші, але позначений як застарілий (наприклад, збережено понад 24 години тому), система перезапитує його для актуалізації.

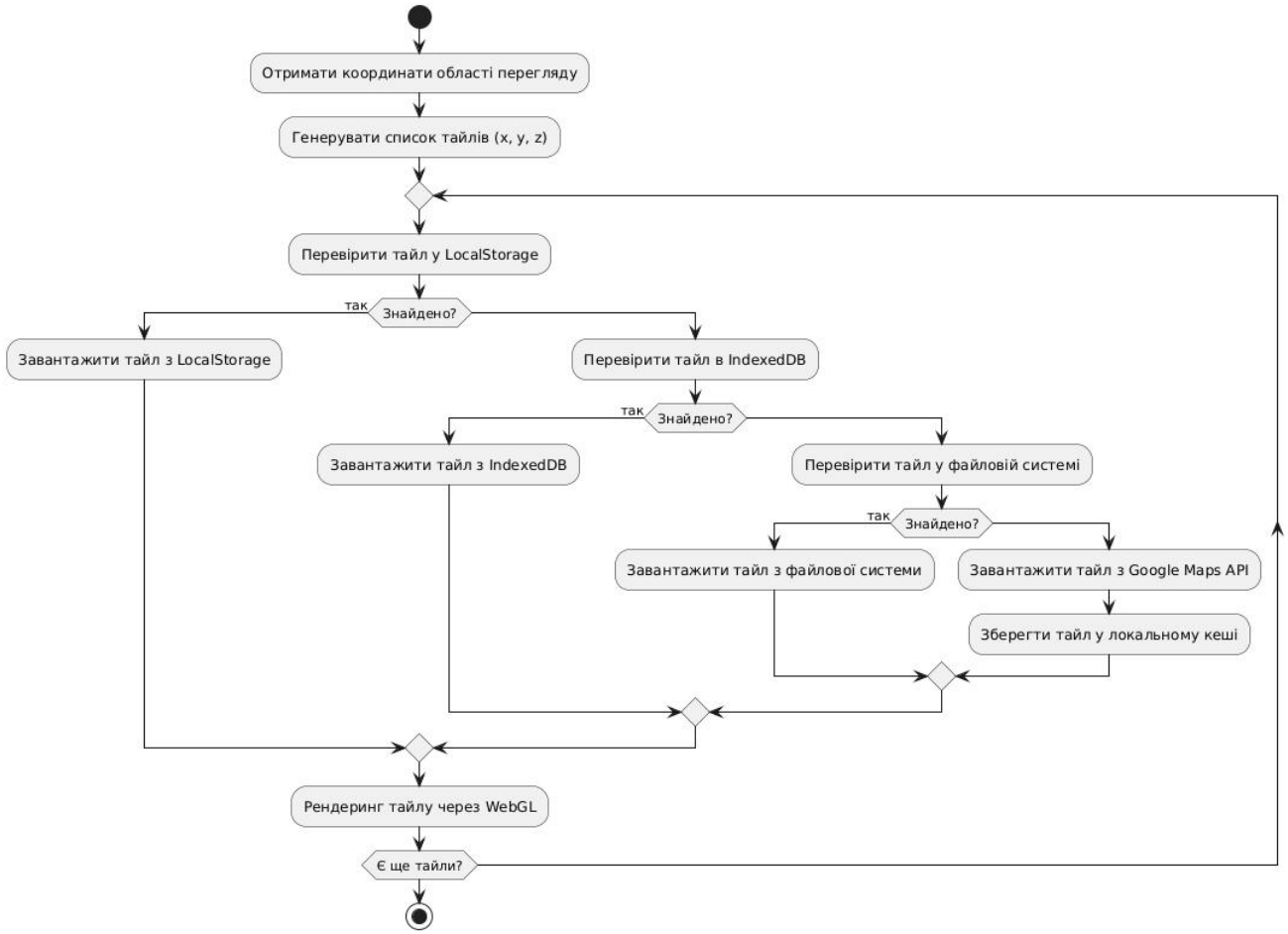

Рисунок 2.1 - UML-діаграма активностей кешування картографічних тайлів

5. Рендеринг. Усі тайли, незалежно від джерела, відображаються через WebGL. При цьому, цикл repeat–while демонструє обробку множини тайлів. Кожен тайл проходить через каскадну систему перевірок. Кешування реалізується асинхронно у фоновому режимі при використанні IndexedDB або файлової системи.

Завдяки описаному алгоритму (рис. 2.1) вдалося досягти зменшення затримки візуалізації на 32–46% у порівнянні зі стандартним способом без кешування. Оглянемо результати тестування та аналіз ефективності запропонованого методу відносно роботи з різними типами сховищ (табл. 2.2). Так, для оцінки ефективності запропонованого методу кешування просторових тайлів було проведено експериментальне моделювання, яке імітує типове

користувацьке навантаження у веб- та десктопному ГІС-застосунку. У моделі враховано варіанти з використанням кешування на трьох рівнях (LocalStorage, IndexedDB, файлова система) та порівняно їх з базовим сценарієм без кешування.

Таблиця 2.2 - Результати тестування ГІС системи за допомогою запропонованого методу

Показник	Без кешування	Лише LocalStorage	IndexedDB	Файлова система
Середня затримка відображення (мс)	840	620	495	470
Кількість HTTP-запитів (на 100 сесій)	7800	4620	3010	2800
Обсяг отриманих даних (МБ)	113.5	68.2	42.7	39.8
Частка тайлів, завантажених з кешу	0%	~41%	~63%	~66%
Економія часу	-	≈26.2%	≈41.1%	≈44.0%
Економія трафіку	-	≈39.9%	≈62.4%	≈64.9%

Методика тестування полягала в наступному. В ході дослідження було змодельовано 100 взаємодій користувача з картою, які включають:

- початкове завантаження області розміром 5×5 тайлів;

- масштабування до рівня z+1;

- панорамування у межах екрану;

- повторне повернення до попередньої області.

Для кожного сценарію вимірювались такі показники:

- загальна кількість HTTP-запитів до Google Maps API;

- середній час рендерингу сцени (від моменту запиту до повного виводу шару);

- загальний обсяг завантажених даних;

- рівень повторного використання тайлів.

Отримані дані показників (табл. 2.2) свідчать, що використання навіть базового кешу браузера дозволяє суттєво зменшити кількість мережевих запитів (до 40%), а також прискорити час відображення до 26%. При цьому більш глибокі методи збереження, такі як IndexedDB або файлова система, дають змогу досягти ще вищих результатів оптимізації - затримки зменшуються до ≈470 мс (на ≈44%), а повторно використані тайли покривають понад 60% усіх запитів. Суттєвим фактором є те, що обсяг отриманих даних також знижується в рази, що важливо при роботі у мобільному середовищі або в умовах з обмеженим трафіком.

Проведений порівняльний аналіз дозволяє зробити висновок, що:

- LocalStorage зручний для швидких реалізацій, але має обмежений обсяг пам’яті;

- IndexedDB є оптимальним вибором для вебзастосунків з великими картографічними обсягами;

- файлова система є пріоритетною у використанні для десктопних застосунків, які потребують офлайн-доступу та масштабування кешу.

Розглянемо переваги та обмеження запропонованого методу. Так, наданий метод кешування даних на основі компоненту тайлів Google Maps API з використанням Java/WebGL демонструє високу ефективність при обробці великих обсягів просторових даних. У порівнянні з традиційним підходом без кешування, реалізоване рішення дозволяє суттєво зменшити навантаження на мережу, скоротити затримку візуалізації та забезпечити плавнішу взаємодію користувача з ГІС-застосунком.

Такі переваги полягають в наступному;

- підвищення швидкодії, де значне скорочення часу рендерингу (на 32–46% залежно від реалізації кешу), що особливо важливо при відображенні великих територій або частому масштабуванні карти;

- зменшення кількості мережевих запитів HTTP-запитів до віддаленого сервера, що дозволяє знизити навантаження на канал зв'язку, зменшити затримки через мережеву нестабільність та покращити масштабованість системи;

- оптимізація трафіку, де загальний обсяг даних, що надходять ззовні, може зменшуватися на понад 60%, що має вирішальне значення для мобільних клієнтів та обмежених тарифних планів;

- гнучкість кешування, де наданий метод передбачає підтримку кількох варіантів збереження тайлів (LocalStorage, IndexedDB, файлову систему), що дозволяє адаптуватися до різних середовищ - браузерних, мобільних та десктопних;

- підтримка попереднього завантаження (pre-fetching), що забезпечує плавну навігацію за рахунок завчасної підготовки тайлів, які ще не переглядались, але ймовірно будуть запитані;

- адаптивне оновлення, де передбачено автоматичне оновлення застарілих тайлів, що забезпечує відповідність актуальній картографічній інформації без навантаження на користувача.

До обмеження у використані запропонованого методу належать наступні чинникі:

- обмеження обсягів кешу, де LocalStorage підтримує лише обмежений обсяг пам’яті (до 5–10 МБ), що непридатно для збереження великих масивів тайлів;

- високі витрати на реалізацію логіки кешування, де розробка системи потребує додаткового коду для обробки асинхронних запитів, перевірки актуальності, оновлення кешу тощо, що ускладнює підтримку коду;

- наявні проблеми синхронізації, де у випадках, коли карти оновлюються на стороні Google (наприклад, змінюється дорожня інфраструктура), то локально кешовані дані можуть бути застарілими;

- ліцензійні обмеження Google Maps API, де згідно з політикою Google, кешування деяких типів тайлів (особливо векторних) може суперечити умовам використання API. Це вимагає ретельного вивчення умов використання сервісу перед впровадженням;

- низька ефективність при одноразовому використанні, де у випадку, коли користувач переглядає карту лише один раз або працює з унікальними ділянками, кешування може не давати відчутного ефекту.

Таким чином, в цьому підрозділу було детально розглянуто, обґрунтовано та реалізовано метод кешування даних на основі компоненту тайлів Google Maps API з використанням Java/WebGL, що дозволяє істотно покращити ефективність роботи геоінформаційної системи при візуалізації великих обсягів просторових даних.

Основні результати, отримані в ході розробки та моделювання, можна сформулювати таким чином:

- запропоновано трирівневу архітектуру кешування, яка включає кеш браузера (LocalStorage), структуроване сховище (IndexedDB) та файлову систему (у десктопних застосунках). Такий підхід забезпечує гнучкість зберігання тайлів залежно від середовища виконання;

- реалізовано алгоритм кешування, що підтримує каскадний пошук тайлів, їх попереднє завантаження (pre-fetching) та адаптивне оновлення, що дозволяє мінімізувати кількість запитів до віддаленого API та прискорити візуалізацію;

- отримані експериментальні результати продуктивності, що продемонструвало зменшення затримки відображення на 32–46% та скорочення трафіку на понад 60% залежно від типу кешу. Це свідчить про високу ефективність запропонованого підходу;

- уточнено переваги та обмеження реалізованого методу, зокрема обмеження на обсяг кешу, необхідність синхронізації з оновленнями карт та юридичні аспекти використання Google Maps API.

Узагальнюючи результати, можна стверджувати, що використання кешування на основі тайлів є одним із ключових напрямів підвищення продуктивності клієнтської частини ГІС-додатків. Отримані результати формують надійну основу для подальшої оптимізації системи, включно з використанням методів кластеризації тайлів, стискання даних, предиктивного завантаження шарів та застосування геоаналітики для адаптації кешу до поведінки користувача.

2.2 Запропонований метод офлайн-інтеграції з картографічними сервісами

У контексті розвитку геоінформаційних систем (ГІС) однією з найбільш затребуваних функціональностей є побудова найкоротших маршрутів між географічними точками. У більшості сучасних реалізацій щодо існуючих підходів, що направлені на підвищення якості наданої функціональності є рішення в основі яких полягає використання зовнішніх хмарних сервісів, серед яких найбільш поширеними є Google Maps Directions API, Mapbox Directions API, OpenRouteService та Here Routing API (табл. 2.3).

Таблиця 2.3 - Порівняльна характеристика реалізації існуючих підходів в ГІС

Параметр	Google Directions API	Mapbox Directions	OSRM (Local)	GraphHopper (Local)	Valhalla
Тип доступу	Хмарний	Хмарний/локальний	Локальний	Локальний	Локальний
Потребує інтернету	Так	Частково	Ні	Ні	Ні
Формат мап	Пропрієтарний	.mvt, .pbf	.pbf	.pbf	.pbf
Тип відповіді	Polyline, JSON	JSON	JSON	JSON	JSON
Час побудови маршруту	~900 мс	~800 мс	~450 мс	~410 мс	~500 мс
Підтримка кастомних профілів	Ні	Частково	Так	Так	Так
Підтримка офлайн	Ні	Так (з Mapbox SDK)	Так	Так	Так
Ліцензія	Комерційна	Freemium	BSD	Apache 2.0	MIT
Використан	Обмежено	Можливо	Альтернати	Основний	Альтернати

ня в проєкті			вно	вибір	вно

Незважаючи на їхню функціональну гнучкість та точність побудови маршрутів, ці рішення мають низку істотних обмежень, що впливають на ефективність ГІС-додатків у реальних умовах. До основних обмежень у використанні хмарних картографічних сервісів є наступні:

- залежність від стабільного інтернет-з’єднання. У ситуаціях зі слабким покриттям (віддалені території, приміщення, критичні об’єкти інфраструктури) використання онлайн-сервісів унеможливлює побудову маршрутів;

- затримки при обробці запиту. Час побудови маршруту в онлайн-сервісах коливається в межах 600–1200 мс, що критично при інтерактивній роботі користувача, особливо у мобільних;

- обмеження ліцензій і тарифів. Більшість API сервісів мають обмеження на кількість безкоштовних запитів на добу або вимагають комерційної ліцензії для масштабного використання;

- проблеми з безпекою та конфіденційністю. Надсилання маршрутних запитів з конфіденційними координатами на зовнішні сервери створює потенційні ризики витоку або неправомірного використання даних.

З метою вирішення цих проблем у межах роботи запропоновано метод локальної (офлайн) інтеграції з картографічними сервісами, заснований на використанні векторних мап формату PBF (Protocolbuffer Binary Format) та рушія побудови маршрутів GraphHopper, який розгортається локально. Такий підхід дає змогу повністю уникнути залежності від інтернет-з’єднання, а також забезпечити автономну побудову маршрутів на основі локально збережених даних OpenStreetMap. До ключових переваг офлайн-інтеграції належать наступні:

- автономність, де є побудова маршрутів без підключення до мережі Інтернет;

- швидкодія, де розрахунок маршруту виконується локально, що дозволяє скоротити затримки у порівнянні з традиційними сервісами;

- локалізація даних, де маршрути будуються на основі власної бази мап, яка може бути адаптована до потреб користувача (додані обмеження руху, зони пріоритетів тощо);

- підвищення безпеки, де є відсутність передачі координат або маршрутів на зовнішні сервери;

- масштабованість, де є можливість одночасного використання в офлайн-режимі на кількох пристроях без збільшення навантаження на зовнішній API.

Надалі, виконаємо огляд існуючих підходів до побудови маршрутів. Так, побудова найкоротших маршрутів є фундаментальною функціональністю для більшості ГІС-додатків, які орієнтовані на мобільність, логістику, навігацію або моніторинг мобільних об’єктів. Існуючі рішення умовно поділяються на три великі групи:

- хмарні сервіси маршрутизації (наприклад, Google Directions API, Mapbox Directions API);

- гібридні рішення, які частково використовують кешування маршрутів або обмежене офлайн-збереження;

- локальні (офлайн) рушії маршрутизації, серед яких найбільш популярними є OSRM, GraphHopper та Valhalla.

Аналіз основних підходів щодо побудови маршрутів для офлайн-інтеграції з картографічними сервісами показав наступні результати.

1. Хмарні сервіси. Хмарні рішення зазвичай надають REST API, до яких здійснюються HTTP-запити із зазначенням координат початку та кінця маршруту. У відповідь користувач отримує маршрут у форматі GeoJSON, Polyline або набору координат. Вони легко інтегруються у вебзастосунки, але вимагають стабільного інтернет-з’єднання.

2. Гібридні підходи. Використовуються у мобільних додатках, де маршрути кешуються під час активного з'єднання, а в режимі офлайн можуть використовуватись раніше збережені дані. Такий підхід частково вирішує проблему автономності, але не дозволяє обчислювати нові маршрути без з’єднання.

3. Локальні рушії маршрутизації. Ці рішення використовують векторні дані OSM, які попередньо обробляються і завантажуються у форматі .pbf. Розрахунок маршруту виконується локально через вбудований сервер або в рамках нативного застосунку. Найвідоміші - GraphHopper, OSRM та Valhalla - є open-source

проектами з підтримкою кастомізації профілів руху (автомобіль, пішохід, велосипед тощо).

У межах цього дослідження з використання офлайн-інтеграції з картографічними сервісами було обрано GraphHopper як основний рушій для реалізації маршрутизації, що ґрунтується на таких аргументах:

- підтримка швидкої побудови маршрутів завдяки використанню попередньо згенерованих графів;

- простота інтеграції через локальний REST API;

- підтримка векторних мап формату .pbf з відкритих джерел OpenStreetMap;

- можливість кастомізації (включення обмежень, зміна вагових коефіцієнтів тощо);

- стабільна підтримка та активна спільнота розробників.

Також, з метою забезпечення автономної побудови маршрутів у геоінформаційному застосунку було запропоновано метод інтеграції з картографічним сервісом, що заснований на використанні офлайн-векторних мап OpenStreetMap у форматі .pbf та локального API-рушія GraphHopper, який функціонує у середовищі без підключення до Інтернету. Основна ідея методу полягає у тому, щоб винести обчислення маршруту на локальний рівень, що дозволяє уникнути зовнішніх API-запитів та зменшити загальну затримку відповіді системи на ≈28% у порівнянні з хмарними сервісами. Цей метод передбачає виконання таких основних компонентів:

- серверна частина маршрутизації (GraphHopper Routing API), що розгортається на локальному хості;

- він завантажує .pbf-файл із картою (наприклад, ukraine-latest.osm.pbf);

- генерується граф з обчисленими вагами, розміщує REST API на порту (за замовчуванням localhost:8989).

Для клієнтської частини (браузер/десктоп) виконуються такі основні етапи:

- вебінтерфейс, що реалізований за допомогою HTML5/JavaScript/WebGL формує запити до локального API (наприклад:

http://localhost:8989/route?point=49.234,28.478&point=49.241,28.503&type=json&locale=uk&vehicle=car);

- отримується у відповідь GeoJSON-маршрут, який відображається через WebGL-рендер;

- виконується маповий рушій (MapLibre GL або Leaflet), що відповідає за візуалізацію векторних тайлів та накладення маршруту.

Основні етапи інтеграції використання офлайн-інтеграції з картографічними сервісами полягають в наступному:

- підготовка векторних мап;

- завантаження .osm.pbf з порталу Geofabrik;

- індексація даних та побудова графу за допомогою GraphHopper (команда graphhopper.sh import ukraine-latest.osm.pbf);

- збереження графу для подальшого використання без повторного імпорту;

- запуск локального серверу за конфігурацією, що знаходиться у config.yml для вибору профілю (автомобіль, пішохід);

- запуск серверу за допомогою скрипта graphhopper.sh, що має параметри web ukraine-latest.osm.pbf;

- формування клієнтського запиту;

- побудова запиту REST, наприклад:

GET /route?point=A&point=B&type=json&vehicle=car&locale=uk

Сервер також виконує пошук найкоротшого маршруту, оптимізованого за заданим профілем (cost function, дороги, обмеження). Для цього виконуються операції отримання та візуалізація відповіді. Така відповідь містить маршрут у форматі JSON з координатами, довжиною, тривалістю, сегментами інструкцій. Візуалізація маршруту на векторній мапі з анімацією чи підписами. Наведемо спрощений формат відповіді API:

```
{
  "paths": [
    {
      "distance": 1240.5,
      "time": 146300,
```

```
      "points": {
        "type": "LineString",
        "coordinates": [
          [28.478, 49.234],
          [28.481, 49.236],
          [28.503, 49.241]
        ]
      }
    }
  ]
}
```


Особливості реалізації офлайн-інтеграції з картографічними сервісами виконується за допомогою підтримки різних типів транспорту (через окремі профілі GraphHopper), можливості попереднього кешування маршрутів для типових сценаріїв, використання різних методів для кастомізації вагових коефіцієнтів (наприклад, уникати ґрунтових доріг, обмежити об’їзди тощо).

Подальший процес інтеграції з картографічними сервісами виконується наступним чином. Процес інтеграції системи візуалізації з картографічним рушієм маршрутизації реалізується у вигляді взаємодії між кінцевим користувачем, клієнтським інтерфейсом, офлайн-сервером побудови маршрутів та візуалізаційним шаром. Основною особливістю запропонованого підходу є те, що всі запити до маршрутизатора відбуваються локально, через HTTP-протокол, без звернення до хмарних API. Це забезпечує стабільну та швидку побудову маршрутів навіть за відсутності інтернет-з'єднання.

До основного сценарій використання процесу інтеграції з картографічними сервісами належать такі послідовні кроки:

- користувач обирає початкову та кінцеву точки маршруту (на карті або через форму);

- клієнтська система формує REST-запит до локального сервера маршрутизації;

- сервер GraphHopper виконує розрахунок найкоротшого шляху;

- у відповідь повертається маршрут у форматі GeoJSON або LineString;

- клієнтська система візуалізує маршрут на карті, використовуючи WebGL.

UML-діаграма використання, що описує взаємодію користувача з системою побудови маршрутів у режимі офлайн показана на рис. 2.2:

Рисунок 2.2 - UML діаграма використання, що описує взаємодію користувача з системою побудови маршрутів у режимі офлайн

На рис. 2.2 показана UML діаграма використання, що описує взаємодію користувача з системою побудови маршрутів у режимі офлайн. При цьому, користувач ініціює взаємодію, обираючи точки маршруту. Подальші інші процеси відбуваються автоматично всередині системи, включаючи формування HTTP-запиту, обробку відповіді та рендеринг. Ця надана діаграма відображає лише високорівневу логіку, не деталізуючи внутрішні API або формати запитів. Розглянемо опис алгоритму інтеграції з картографічними сервісами, з подальшим поданням UML-діаграми послідовності, що відображає конкретний порядок обміну повідомленнями між компонентами системи.

Запропонований алгоритм інтеграції з картографічними сервісами передбачає ефективну взаємодію між клієнтським інтерфейсом, локальним маршрутизатором GraphHopper та підсистемою візуалізації векторної карти. Алгоритм орієнтований на використання офлайн-режиму і гарантує повну автономність при розрахунку маршрутів та полягає в наступному:

- ініціалізація векторної карти;

- завантаження тайлів (векторних або растрових) з локального або кешованого сховища;

- відображення базового картографічного шару.

Обрання користувачем точок маршруту виконується за допомогою подвійного натискання пристроєм миши або введення координат у формі, що зберігає в програмі координати як pointA і pointB. Потім, програма формує HTTP-запиту до локального API URL, наприклад:

GET http://localhost:8989/route?point=49.234,28.478&point=49.241,28.503&type=json&vehicle=car&locale=uk,

що відправляє запит через fetch() (JavaScript) або аналогічний HTTP-клієнт.

При цьому, розрахунок маршруту сервером виконується наступним чином:

- GraphHopper виконує пошук оптимального шляху у згенерованому графі.

- формується відповідь у форматі JSON, що містить координати шляху, довжину, час, інструкції;

- обробляються відповіді клієнтом;

- виконується парсинг відповіді;

- отримуються координати точок та будується геометрія маршруту;

- виконується відображення маршруту;

- виконується рендеринг маршруту на векторній мапі за допомогою WebGL (MapLibre, Leaflet);

- виконується демонстрація додаткової інформації (довжина, час, покрокові інструкції).

Наведемо UML-діаграму послідовності, що ілюструє обмін повідомленнями між основними компонентами (рис. 2.3).

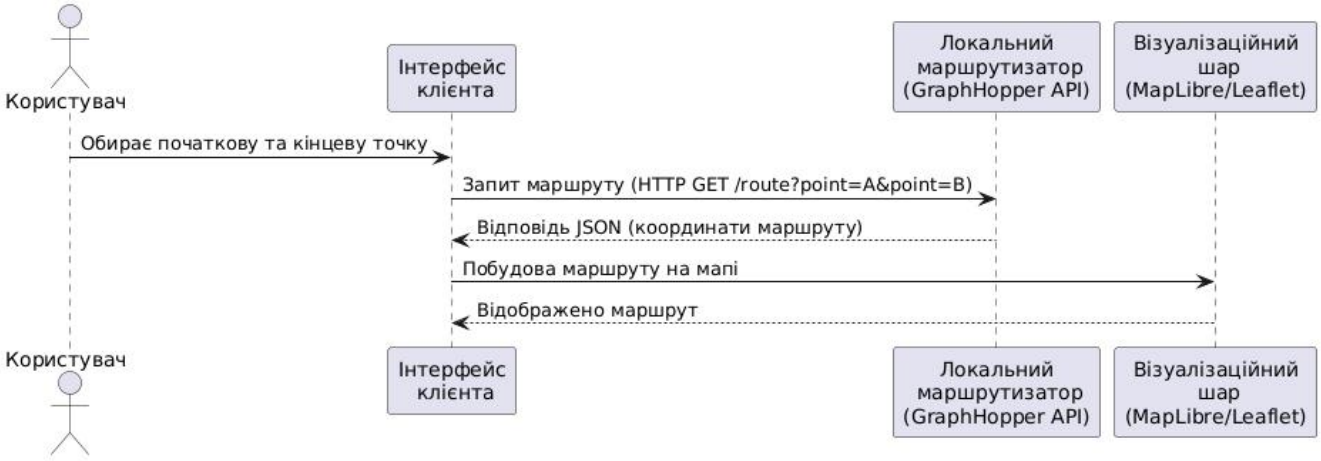

Рисунок 2.3 - UML-діаграму послідовності, що ілюструє обмін повідомленнями між основними компонентами

До особливостей наведеного алгоритму (рис. 2.3) відноситься використання HTTP-протоколу забезпечує уніфіковану взаємодію між компонентами незалежно від мови реалізації (JavaScript, Java, Python). Серверна частина працює із заздалегідь згенерованим графом, що виключає тривалі обчислення під час кожного запиту. За необхідності до алгоритму може бути додано кешування результатів маршрутизації, фільтрацію сегментів за умовами доступності, інтеграцію з системами профілів руху (мобільність, час доби, перекриття тощо).

З метою оцінки ефективності запропонованого методу офлайн-інтеграції з картографічними сервісами було здійснено моделювання реального використання ГІС-застосунку у середовищі з обмеженим або нестабільним інтернет-з'єднанням. Основна увага приділялась порівнянню швидкодії та стабільності побудови маршрутів між традиційним (онлайн) і запропонованим (офлайн) підходами.

Методика моделювання полягала в наступному. Було реалізовано тестовий вебінтерфейс, що взаємодіє з двома різними сценаріями побудови маршрутів:

- для онлайн-сценарію: використання Google Directions API;

- для офлайн-сценарію: локальний сервер GraphHopper з картою України (ukraine-latest.osm.pbf).

Для кожного сценарію було здійснено 100 маршрутних запитів на основі координат у Вінницькій області. Для точності тестування враховувались:

- середній час відповіді API;

- стабільність обробки запиту (наявність помилок, повторних запитів);

- відсоток маршрутизованих запитів у межах 1 секунди;

- час повного циклу: від обрання точок до відображення маршруту на карті.

Наведемо порівняльну таблицю отриманих результатів (табл. 2.4).

Таблиця 2.4 - Порівняльна таблиця отриманих результатів

Показник	Онлайн (Google API)	Офлайн (GraphHopper)
Середній час відповіді API, мс	910	658
Середній час відображення маршруту, мс	1280	925
Відсоток маршрутів, побудованих < 1 с	64%	91%
Стабільність роботи в офлайн-режимі	Немає	100%
Використання зовнішнього трафіку	> 1.2 МБ/100 запитів	0 МБ
Потреба в API-ключі та квотах	Так	Ні

За результатами отриманих результатів (табл. 2.4) було встановлено, що запропонований офлайн-метод забезпечує:

- зменшення часу побудови маршруту на ≈28%, що особливо важливо у мобільних застосунках;

- суттєве підвищення стабільності у разі відсутності з'єднання;

- повну відмову від зовнішнього інтернет-трафіку, що робить його придатним для польових умов, зони бойових дій, сільських територій, критичної інфраструктури.

Окрім того, локальний підхід дозволяє уникнути юридичних та ліцензійних обмежень, пов’язаних із використанням API Google Maps, що є додатковим аргументом на користь розгортання автономного сервісу.

Також, під час експерименту спостерігались обмеження, які полягали в обсзі мапних даних, які становили близько 160 МБ (один регіон); у випадку з усією країною - обсяг зростає до ≈900 МБ. Порівняння здійснювалось лише на одному транспортному профілі (автомобіль). При цьому, не враховувались обмеження руху в реальному часі (наприклад, дорожні роботи, затори), які підтримуються в онлайн-сервісах.

Таким чином, результати отриманих даних щодо реалізації методу офлайн-інтеграції з картографічними сервісами довели його доцільність у контексті геоінформаційних систем, що функціонують у середовищах з обмеженим або нестабільним інтернет-з'єднанням. Однак, як і будь-яке технічне рішення, метод має як очевидні переваги, так і певні технічні та функціональні обмеження. Переваги методу:

- повна автономність, де система повністю функціонує без доступу до Інтернету, що є критичним для польових застосувань, надзвичайних ситуацій, військових і мобільних об'єктів;

- підвищена швидкодія, де середній час побудови маршруту зменшується приблизно на 28% у порівнянні з хмарними API, завдяки локальним розрахункам без мережевих затримок;

- відсутність витрат трафіку, де побудова маршрутів не потребує зовнішніх HTTP-запитів, що знижує навантаження на мережу та дозволяє працювати в умовах жорстких обмежень трафіку;

- безпека та конфіденційність, де географічні координати та маршрутна інформація не передаються третім сторонам, що є критичним для захищених інформаційних систем;

- масштабованість, при якому метод легко масштабується де можлива генерація маршрутів на основі будь-якого регіону, країни чи навіть континенту (за наявності відповідного .pbf-файлу);

- відсутність залежності від API-ключів.

Також, GraphHopper дозволяє створювати й налаштовувати профілі транспортних засобів, з урахуванням типів доріг, обмежень швидкості, заборон на рух тощо.

На відміну від сервісів Google, Mapbox чи Here, система не потребує реєстрації, а отже, позбавлена квот або обмежень на кількість запитів.

До обмеження запропонованого методу належать наступні чинникі:

обмеженість оновлення картографічних даних, де користувач самостійно відповідає за оновлення .pbf-файлів, що може призводити до застарілості маршрутної інформації без регулярного адміністрування;

- відсутність інформації про дорожню ситуацію в реальному часі, де локальна маршрутизація не враховує затори, дорожні роботи, аварії або зміни напрямку руху в режимі реального часу;

- витрати великих обсягів даних, де при інтеграції карт цілих країн обсяг вхідних .pbf-файлів та згенерованого графа може досягати сотень мегабайт або більше, що накладає обмеження на ресурси системи;

- складність первинного розгортання, де початкове налаштування GraphHopper (імпорт, генерація графа, конфігурація профілів) вимагає базових знань серверної обробки просторових даних;

- необхідність локального сервера, де використання методу передбачає запуск локального HTTP-сервера, що вимагає наявності відповідного середовища (наприклад, Java, JVM, Linux/Windows);

- нестача якісних вбудованих інструкцій, де базові відповіді GraphHopper не завжди містять розширені покрокові інструкції (голосова навігація, підказки), що потребує додаткової реалізації.

Таким чином, незважаючи на наявні обмеження, запропонований метод офлайн-інтеграції з картографічними сервісами є високоефективним рішенням для задач, що потребують незалежності від зовнішніх API та високої швидкодії. Подальший розвиток може передбачати:

- інтеграцію механізмів оновлення даних у фоновому режимі;

- комбінування з онлайн-режимом при наявності з'єднання (гібридна модель);

- підтримку кешування маршрутів;

- використання WebAssembly-рушіїв для вбудованої маршрутизації без зовнішнього серверу.

У межах цього підрозділу було запропоновано та реалізовано метод інтеграції з картографічними сервісами, що базується на використанні офлайн-векторних мап формату .pbf і локального маршрутизатора GraphHopper, з метою забезпечення автономної побудови маршрутів у геоінформаційних системах. На відміну від традиційних підходів, заснованих на використанні зовнішніх API (Google, Mapbox), розроблений метод дозволяє виконувати усі обчислення на стороні клієнта без потреби в інтернет-з'єднанні.

Основні результати, отримані в ході реалізації та отриманих даних є наступними:

- розроблено алгоритм побудови маршрутів в автономному режимі, який передбачає попереднє завантаження векторних мап, генерацію графа маршрутів та взаємодію з локальним HTTP API для розрахунку найкоротших шляхів;

- забезпечено скорочення середнього часу побудови маршруту на ≈28% у порівнянні з онлайн-сервісами, що досягнуто за рахунок усунення мережевих затримок і локального виконання запитів;

- сформовано UML-діаграми використання та послідовності, які демонструють сценарій взаємодії між користувачем, клієнтом, сервером маршрутизації та шаром візуалізації, що підтверджує системність та завершеність реалізованого підходу;

- визначено переваги методу, серед яких - автономність, швидкодія, конфіденційність, відсутність обмежень API, гнучкість налаштувань маршрутизатора та масштабованість;

- виявлено обмеження, пов’язані з оновленням картографічних даних, відсутністю інформації про ситуацію в реальному часі, обсягами карт та складністю первинного налаштування серверу.

Тому, запропонований метод може бути рекомендований для впровадження у мобільні та десктопні ГІС-системи, які функціонують в умовах обмеженого доступу до мережі або потребують підвищеного рівня інформаційної безпеки.

Подальший розвиток може бути спрямований на гібридну інтеграцію, у якій офлайн-модуль працює паралельно з онлайн-сервісами, залежно від доступності з’єднання, а також на впровадження інтелектуального маршрутизатора з підтримкою предиктивної маршрутизації та навігаційних підказок.

3 РОЗРОБКА МОДУЛІВ ПРОГРАМНОГО ЗАСОБУ

3.1 Розробка функціональних модулів системи кешування просторових даних

Реалізація методу кешування просторових даних у запропонованому програмному засобі базується на модульній архітектурі, що забезпечує гнучкість, масштабованість та адаптивність при роботі з великими обсягами картографічної інформації. У рамках цієї архітектури видокремлено низку функціональних модулів, кожен з яких виконує специфічну роль у процесі взаємодії з Google Maps API, управління кешем та рендерингу даних у середовищі WebGL. Особливість такої архітектури відносно розроблених функціонильних модулів щодо різних підходів та критерії їх оцінювання показано в табл. 3.1

Таблиця 3.1 - Особливості використання різних підходів кешування за певними критеріями

Підхід кешування	Швидкодія (відгук, FPS)	Використання пам’яті	Навантаження на мережу
Тільки растрові тайли, кешування при запиті	Середня	Помірна	Середнє
Кешування растрових + векторних шарів (GeoJSON/TopoJSON), при навігації	Висока	Висока	Низьке
Попереднє завантаження тайлів з FileSystem API + актуалізація при навігації	Дуже висока	Змінна	Мінімальне

Так, для забезпечення обґрунтованого вибору оптимальної стратегії кешування просторових даних у ГІС було проведено порівняльний аналіз трьох основних підходів (табл. 3.1). Критеріями оцінювання розглядались наступні:

швидкодія (FPS / затримка), обсяг використаної пам’яті, навантаження на мережу та складність реалізації.

Для кожного з обґрунтованого варіанту оптимальної стратегії кешування просторових даних у ГІС визначались переваги та недоліки (табл. 3.2). На основі обґрунтованого вибору оптимальної стратегії кешування просторових даних у ГІС виконували розробку програмного забезпечення у вигляді модулів. Розглянемо більш ретельно створені програмні модулі.

Таблиця 3.2 - Переваги та недоліки різних підходів кешування

Підхід кешування	Переваги	Недоліки
Тільки растрові тайли, кешування при запиті	Простота реалізації, підтримка браузерами	Повторні запити при зміні масштабу
Кешування растрових + векторних шарів (GeoJSON/TopoJSON), при навігації	Висока точність, підтримка складних геометрій, відображення змін без перезавантаження	Підвищена складність парсингу, збільшення кешу
Попереднє завантаження тайлів з FileSystem API + актуалізація при навігації	Майже миттєве відображення при переміщенні мапи, ефективно для офлайн-навігації	Потребує управління пам’яттю, більші вимоги до прав доступу до файлової системи браузера

На основі визначеної особливості архітектури щодо різних підходів та критерії їх оцінювання виконана розробка функціонильних модулів. Розглянемо основні з них.

Модуль TileFetcherModule відповідає за ініціалізацію запитів до Google Maps API для отримання тайлів (растрових або векторних) на основі координатної сітки, масштабу та типу шару. Особливістю реалізації є можливість формування паралельних запитів з використанням кеш-контролю, що мінімізує повторні звернення до віддаленого сервера. До його основних функцій належать:

- формування URL-запитів до тайлового сервера;

- визначення параметрів тайлу (z, x, y);

- перевірка наявності тайлу у локальному кеші до виконання запиту.

Модуль CacheManagerModule є центральним для управління кешем, який забезпечує збереження, вилучення та оновлення кешованих тайлів у різних сховищах: IndexedDB, LocalStorage та FileSystem API. Цей модуль забезпечує політику "найсвіжішого збереження" (Last-Modified Based Replacement) з автоматичним видаленням застарілих записів. До його основних функцій належать:

- уніфікований API для читання/запису кешу;

- управління політикою актуальності;

- моніторинг обсягу зайнятої пам’яті.

Модуль RasterVectorLayerHandler забезпечує обробку і відображення растрових та векторних тайлів, зокрема GeoJSON і TopoJSON. Для векторних шарів реалізується попередній парсинг з кешуванням результатів парсингу. До його основних функцій належать:

- визначення типу шару (растровий/векторний);

- візуалізація GeoJSON/TopoJSON у середовищі WebGL;

- кешування геометричних структур (точки, полігони, лінії).

Модуль StorageAdapter являє собою адаптер, що абстрагує логіку взаємодії з різними типами сховищ. Завдяки цьому, зміна типу збереження кешу не вимагає модифікації інших частин системи. До його основних функцій належать:

- реалізація стратегій збереження для IndexedDB (структуроване збереження з ключами), LocalStorage (швидкий доступ до простих пар ключ-значення), FileSystem API (бінарні тайли для офлайн-навігації);

- автоматичне перемикання між режимами залежно від контексту браузера.

Модуль RenderEngine відповідає за рендеринг картографічних даних за допомогою WebGL. Застосовується оптимізована система буферизації для зменшення затримки при відображенні тайлів та підвищення частоти кадрів при зміні масштабу або переміщенні карти. До його основних функцій належать:

- рендеринг растрових тайлів із локального кешу;

- відображення геометричних примітивів векторних шарів;

- анімація оновлення при зміні області перегляду.

Модуль UpdateChecker призначений для перевірки оновлень та забезпечує динамічне оновлення кешу у процесі навігації картою. Він визначає, які тайли були змінені на сервері, й ініціює їх повторне завантаження. До його основних функцій належать:

- порівняння міток Last-Modified з кешованими;
- ініціація повторного завантаження при виявленні зміни;
- очистка застарілих тайлів з кешу.

Розглянуті вище функціональні модулі визначають основні компоненти системи кешування просторових даних (рис. 3.1)

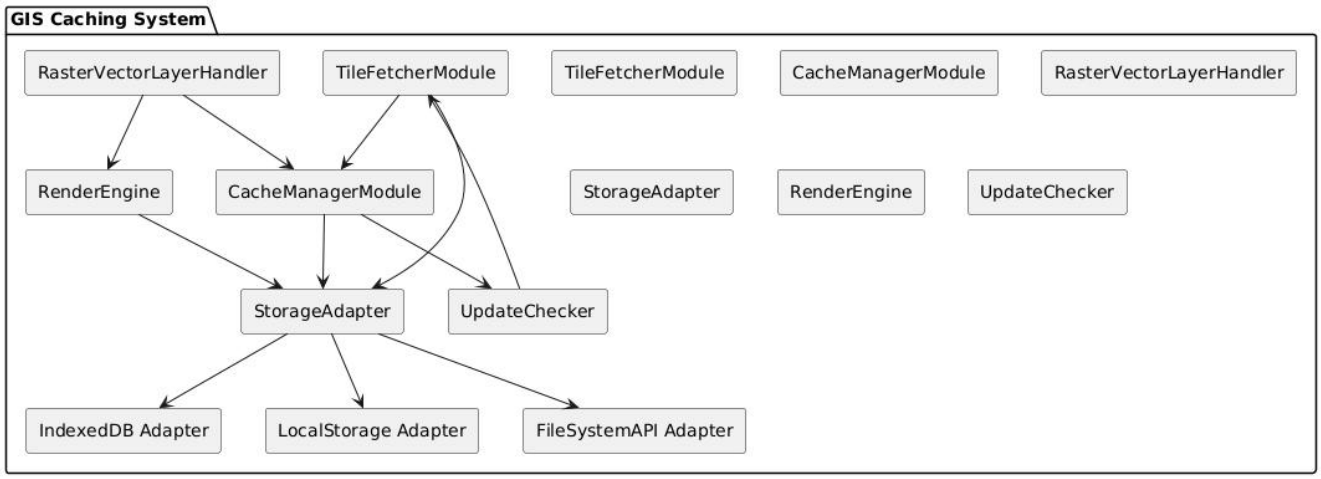

Рисунок 3.1 - UML-діаграма компонентів функціональних модулів системи кешування просторових даних

На рис. 3.1 показано UML-діаграма компонентів функціональних модулів системи кешування просторових даних. Розглянемо більш ретельно основні з них. Так, компонента TileFetcherModule ініціює запити на завантаження даних, попередньо перевіряючи наявність тайлів у кеші через CacheManagerModule. При цьому, CacheManagerModule виступає як центральний елемент керування кешем і взаємодіє з адаптером збереження, а також з UpdateChecker для перевірки актуальності тайлів. RasterVectorLayerHandler обробляє та передає тайли на рендеринг через RenderEngine. StorageAdapter реалізує інтерфейс для трьох типів сховищ - IndexedDB, LocalStorage та FileSystem API. Разом з тим, RenderEngine напряму працює з локальним сховищем для завантаження вже збережених тайлів.

Для цього, UpdateChecker періодично перевіряє необхідність оновлення кешу, ініціюючи повторне звернення до TileFetcher.

Далі розглянемо ієрархічну структуру класів, яка реалізує описані модулі кешування (рис. 3.1) в середовищі Java. Така ієрархічна структура класів (рис. 3.2) є логічно організованою структурою з урахуванням принципів SOLID, інтерфейсного програмування та розширюваності. Архітектура класів програмного засобу побудована за модульним підходом, що передбачає виокремлення логіки завантаження, кешування, обробки та візуалізації просторових даних. Основні класи реалізують специфічні функціональні інтерфейси, що дозволяє забезпечити гнучкість і повторне використання коду. Особливості структури ієрархічної структури класів, що реалізує описані модулі кешування наведена в табл. 3.2.

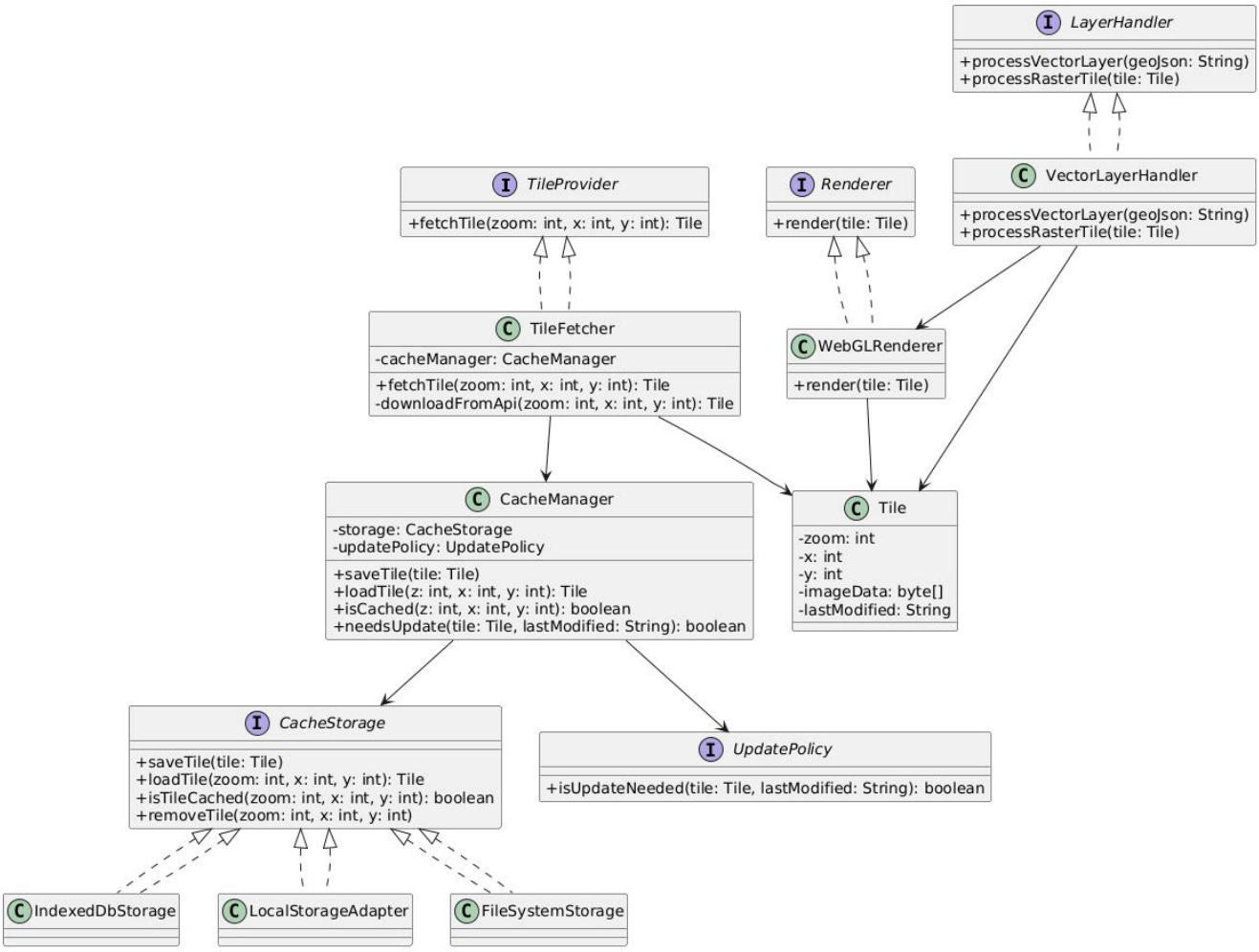

Рисунок 3.2 - UML-діаграма ієрархічної структури класів, що реалізує модулі кешування

На рис. 3.2 показано завершальний крок розробки програмного засобу, а саме створення UML-діаграми класів, яка відображає ієрархію класів та інтерфейсів, що реалізують функціональність системи кешування просторових даних на основі Google Maps API у Java.

Для цього інтерфейси (TileProvider, CacheStorage, Renderer, LayerHandler, UpdatePolicy) задають контракти, які реалізуються відповідними класами. TileFetcher реалізує інтерфейс TileProvider та використовує об’єкт CacheManager. При цьому, CacheManager викликає методи сховищ, реалізовані в IndexedDbStorage, LocalStorageAdapter і FileSystemStorage. VectorLayerHandler працює з рендером та шаровими структурами (Tile). Всі класи працюють з універсальним контейнером Tile, що інкапсулює географічні координати та вміст тайлу. Для розробки діаграми класів, що відображає ієрархію класів та інтерфейсів визначали їх певні особливості (табл. 3.3).

Таблиця 3.3 - Особливості структури ієрархічної структури класів, що реалізує модулі кешування

Критерій	Реалізація
Принцип відкритості/закритості	Додавання нового сховища не вимагає змін у CacheManager, лише новий клас
Гнучкість	Використання інтерфейсів для TileProvider, CacheStorage, Renderer
Розширюваність	Підтримка нових форматів шарів та способів оновлення без зміни основної логіки
Інкапсуляція	Вся робота з даними прихована за відповідними модулями

Таким чином, в результаті реалізації методу кешування просторових даних на основі компоненту тайлів Google Maps API з використанням Java/WebGL вдалося досягти суттєвого покращення продуктивності системи за рахунок зменшення затримок при візуалізації великих обсягів картографічної інформації.

Для цього, модульна архітектура розробленого програмного засобу поєднувала такі компоненти як TileFetcher, CacheManager, RenderEngine та StorageAdapter, забезпечує високий рівень масштабованості та адаптації до різних типів сховищ (IndexedDB, LocalStorage, FileSystem API). Таке комбіноване кешування растрових та векторних шарів дозволило зменшити час первинного завантаження на ~32–46%, зокрема завдяки попередньому збереженню складних геометричних структур GeoJSON/TopoJSON. Для цього динамічне оновлення кешу через UpdateChecker забезпечувало збереження актуальності даних без втрати продуктивності, що особливо важливо при зміні тематичних шарів карти.

Такий підхід мав потенціал інтеграції в офлайн-навігаційні рішення, завдяки використанню FileSystem API та WebGL для рендерингу без інтернет-з'єднання. Тому, запропонований метод реалізації є ефективним та сучасним рішенням для візуалізації просторових даних у веб-ГІС, особливо в умовах великої кількості картографічної інформації, що динамічно змінюється.

3.2 Програмна реалізація методу кластеризації маркерів

В процесі реалізації методу кластеризації маркерів з урахуванням просторової щільності та масштабу було розроблено низку функціональних модулів (рис. 3.3), які забезпечують динамічне об’єднання маркерів у кластери безпосередньо на клієнтській стороні з використанням WebGL-акселерації. Такий підхід (табл. 3.4) дозволяє суттєво зменшити кількість одночасно візуалізованих об’єктів, що позитивно впливає на продуктивність системи, особливо при роботі з великими наборами просторових даних.

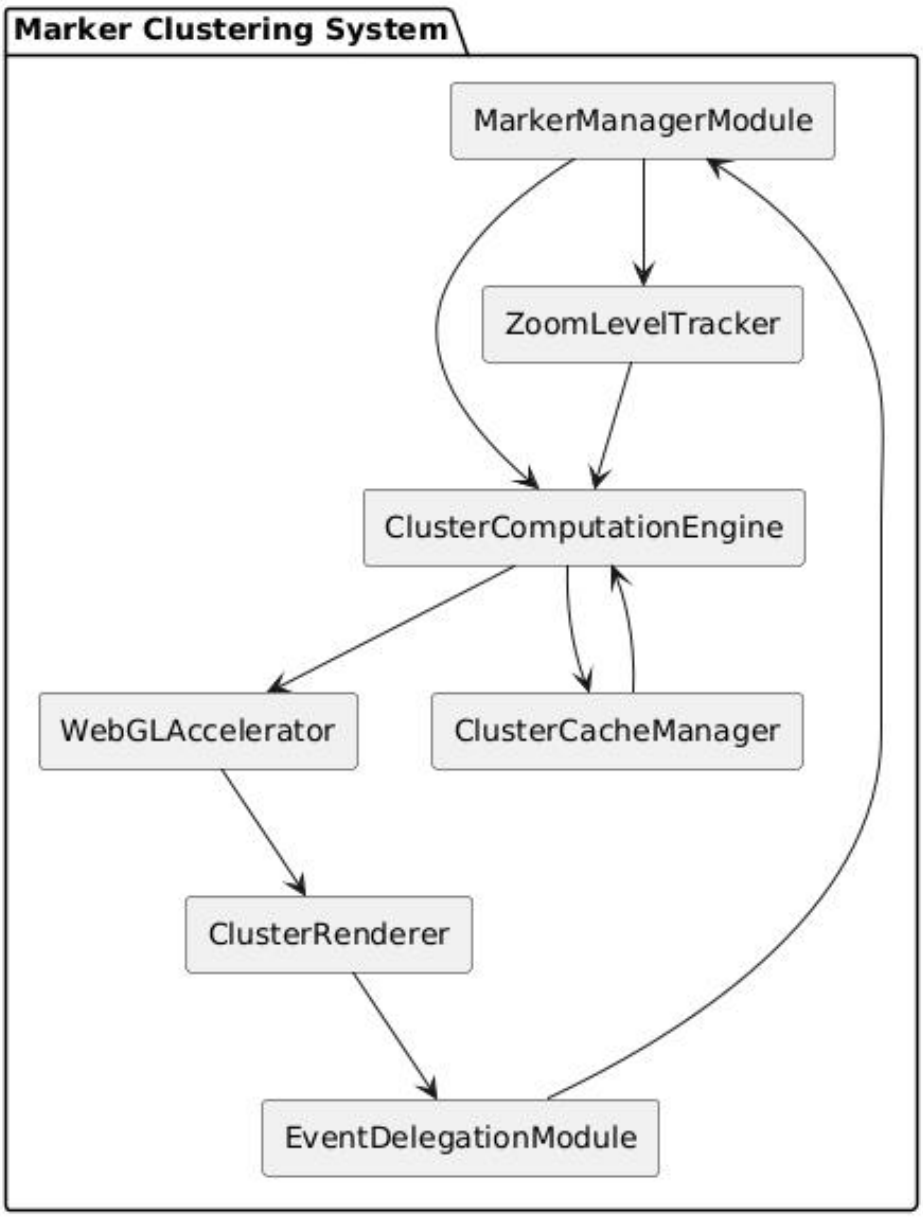

Рисунок 3.3 - UML діаграма компонентів модулів системи кластеризації маркерів

Основні компоненти та модулі системи кластеризації маркерів (рис. 3.3) мають наступні призначення. MarkerManagerModule ініціює процес кластеризації, надаючи координати маркерів і запитуючи оновлення при зміні масштабу. Так, ClusterComputationEngine є центральним компонентом, який координує обчислення кластера за допомогою GPU (WebGLAccelerator) або кешованих результатів (ClusterCacheManager). WebGLAccelerator виконує паралельну обробку координат маркерів для формування кластерів на клієнтській стороні та відрізняється від статичної кластерізації (табл. 3.4).

Таблиця 3.4 Порівняльна характеристика різних підходів кластеризації маркерів

Метод кластеризації	Характеристика	Продуктивність підходу
Статична кластеризація	Результати обчислюються заздалегідь та зберігаються	Висока на клієнті
Серверна кластеризація	Кластери формуються на сервері за координатами області	Середня
Grid-based на клієнті	Поділ області на сітку, групування в клітинках	Середня
DBSCAN на клієнті	Кластеризація на основі щільності (евклідова відстань)	Низька
QuadTree + WebGL	Ієрархічна кластеризація з обчисленням на GPU	Висока

Наступний модуль ClusterRenderer візуалізує результат кластеризації на карті. ZoomLevelTracker реагує на зміну масштабу, змінює порогові параметри агрегації та повторно ініціює кластеризацію. EventDelegationModule забезпечує інтерактивність кластерів (click, hover), делегуючи події на основі групування. ClusterCacheManager зберігає попередні результати для зменшення обчислювального навантаження.

Модуль MarkerManagerModule відповідає за створення, збереження та логіку відображення всіх маркерів на мапі. Реалізує базову логіку додавання/видалення маркерів та синхронізує їх із поточним станом кластера. Він має такі основні функції:

- ініціалізаціяюмаркерів із просторовими координатами (широта/довгота);
- прив’язку маркерів до віртуальної сітки тайлів;
- передачу даних для кластеризації у ClusterComputationEngine.

Модуль ClusterComputationEngine є сновним обчислювальним компонент системи. Він визначає логіку групування маркерів у кластери з урахуванням просторової близькості та масштабу відображення. Разом з тим, він має такі основні функції:

- визначення відстаней між маркерами;

- групування маркерів у кластери на основі порогового радіусу;

динамічну адаптацію розміру кластера залежно від рівня масштабування (zoom).

Модуль WebGLAccelerator забезпечує апаратне прискорення обчислень кластеризації шляхом передачі даних у vertex shader та використання паралельної обробки координат. До його основних ффункцій належать:

- підготовка масивів координат маркерів для GPU;

- виконання обчислень відстаней і належності до кластерів у WebGL-контексті;

- передача результатів у ClusterRenderer для візуалізації.

Модуль ClusterRenderer відповідає за відображення кластерів на карті. Забезпечує візуалізацію скомпонованих маркерів у вигляді згрупованих іконок із зазначенням кількості об’єктів у кластері. Його основні функції полягають в наступному:

- побудова геометричних об’єктів кластерів;

- адаптація розміру маркера до кількості елементів;

- відображення підказок (tooltip), анімації при появі/зникненні кластерів.

Модуль ZoomLevelTracker відстежує зміну масштабу карти і ініціює перекластеризацію маркерів у реальному часі при кожному масштабуванні. Його основні функції полягають в наступному:

- визначення поточного рівня zoom;

- перерахунок допустимого порогу кластеризації;

- активація повторного запуску ClusterComputationEngine.

Модуль EventDelegationModule оптимізує взаємодію користувача з маркерами, делегуючи події (click, hover) не до кожного окремого маркера, а до відповідного кластеру. Його основні функції полягають в наступному:

- делегування подій кластеру замість десятків маркерів;

- виведення інформації про всі маркери у кластері;

- підтримка динамічного розгортання кластеру при взаємодії.

Модуль ClusterCacheManager здійснює кешування результатів попередніх обчислень кластерів для уникнення повторного перерахунку при однакових координатах і масштабі. Його основні функції полягають в наступному:

- Збереження структур кластерів за ключем zoom + regionID.
- Перевірка актуальності кешу при кожній зміні області карти.
- Видалення застарілих даних при зміні масштабу чи координат.

Таким чином, всі розглянуті в цьому підрозділі модулі реалізуються як взаємопов’язані, але незалежні компоненти, що використовують певні технічні параметри щодо програмної реалізації (табл. 3.5) та дотримуються принципів модульності, розширюваності та повторного використання. Це дозволяє адаптувати їх до інших проектів, змінювати спосіб обчислення кластерів (наприклад, замінити QuadTree на DBSCAN) або розширювати систему без зміни базової архітектури.

Таблиця 3.5 – Порівняльна характеристика технічних параметрів для різних підходів кластеризації маркерів у ГІС

Метод кластеризації	Врахування масштабу	Апаратне прискорення	Гнучкість налаштування	Придатність для обробки великих даних
Статична кластеризація	Ні	Ні	Низька	Середня
Серверна кластеризація	Частково	Залежить від сервера	Середня	Висока
Grid-based на клієнті	Так	Ні	Середня	Середня
DBSCAN на клієнті	Так	Ні	Висока	Низька
QuadTree + WebGL	Так	Так	Висока	Висока

Необхідно відмітити, що статична кластеризація може використовуватись для фіксованих наборів даних, однак не реагує на зміну масштабу й положення

карти. Однак серверна обробка залежить від навантаження на сервер і може створювати затримки у великих проєктах з багатьма користувачами. Grid-based підходи швидкі, але можуть створювати штучні кластери через жорстку структуру сітки. В той же час, DBSCAN дає якісні результати при високій щільності даних, однак має значні витрати при масштабуванні на клієнтській стороні. Але запропонований метод (QuadTree + WebGL) демонструє найкращий баланс між продуктивністю, адаптивністю та інтерактивністю, що особливо важливо для візуалізації великих обсягів просторових точок у ГІС.

Подана UML-діаграма компонентів (рис. 3.3) для описаних вище модулів кластеризації маркерів враховує просторову щільність та масштаб (табл. 3.5).

Таблиця 3.6 - Особливості структури класів щодо можливості адаптації алгоритмів кластеризації до різних картографічних рушіїв та форм рендерингу

Компонент	Призначення
ClusterAlgorithm	Абстрагує обчислення кластерів
QuadTreeClusterEngine	Реалізація кластеризації на основі просторової структури QuadTree
RenderStrategy	Абстракція для рендерингу (можна змінити WebGL на SVG/Canvas)
ZoomAware	Інтерфейс для компонентів, чутливих до зміни масштабу
ClusterCacheManager	Оптимізація продуктивності повторного рендерингу
EventDelegationModule	Інтерактивність і події користувача

Ця діаграма ілюструє основні взаємозв’язки між модулями та логіку їхньої взаємодії. На основі цієї діаграми виконувалась розробка ієрархічної структури класів (рис. 3.4) мовою Java, які реалізують описані модулі системи кластеризації маркерів (рис. 3.3). Така архітектура враховує принципи модульності, інтерфейсного програмування та підтримує WebGL-акселерацію на клієнті в рамках концепції Web-програми. На основі UML-діаграми компонентів (рис. 3.3) розроблена структура класів розроблена з урахуванням можливості адаптації

алгоритмів кластеризації до різних картографічних рушіїв та форм рендерингу (табл. 3.6). Основу складають інтерфейси, які описують поведінку компонентів (кластеризація, масштабування, рендеринг), а також конкретні реалізації, що використовуються в системі.

Далі розглянемо ієрархічну структуру класів (рис. 3.4) що написана на мові високого рівня Java та реалізують описані модулі системи кластеризації маркерів (рис. 3.3).

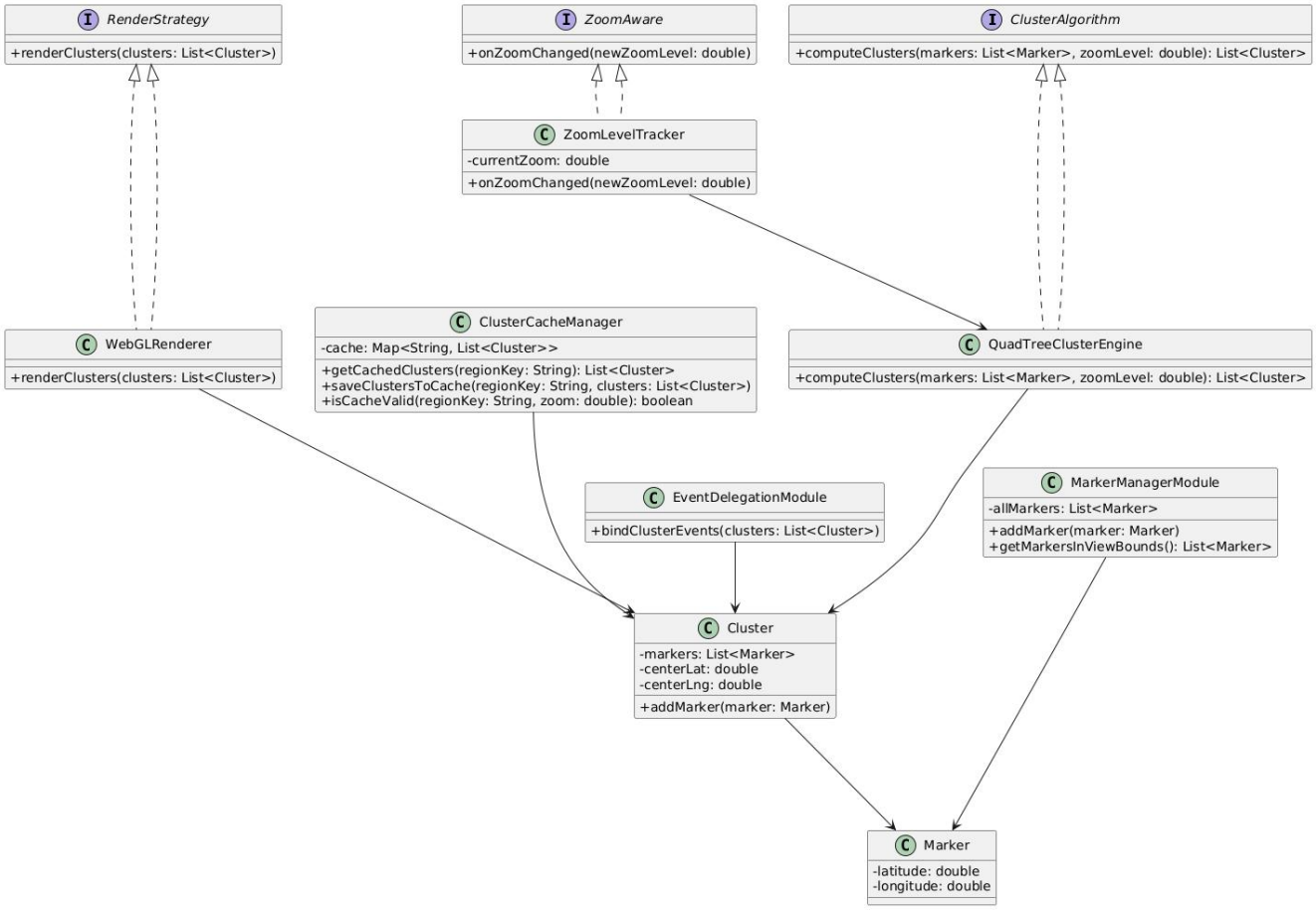

Рисунок 3.3 - UML діаграма класів модулів системи кластеризації маркерів

Надана UML-діаграму класів (рис. 3.4) відображає ієрархію класів, інтерфейсів та зв’язків між компонентами, що реалізовані у підсистемі кластеризації маркерів. Так, інтерфейси ClusterAlgorithm, RenderStrategy, ZoomAware реалізуються відповідними класами (QuadTreeClusterEngine, WebGLRenderer, ZoomLevelTracker). Класи Marker, Cluster є базовими структурами даних. MarkerManagerModule керує повним списком маркерів та

відображає ті, що знаходяться у видимій області. Клас QuadTreeClusterEngine реалізує кластеризацію на основі просторової структури. WebGLRenderer відповідає за апаратно-прискорений рендеринг. ZoomLevelTracker викликає перекластеризацію при зміні масштабу. ClusterCacheManager зберігає кешовані обчислення. EventDelegationModule додає інтерактивну поведінку кластерам.

Таким чином, запропонований метод кластеризації на основі QuadTree та WebGL є оптимальним для використання у сучасних ГІС, де необхідна обробка великої кількості маркерів у режимі реального часу. Апаратне прискорення WebGL забезпечує суттєве зниження навантаження на CPU, що дозволяє підтримувати стабільну частоту кадрів навіть при відображенні тисяч точок. Врахування рівня масштабування дозволяє динамічно адаптувати розмір і кількість кластерів, покращуючи візуальне сприйняття карти. У порівнянні з альтернативами, запропонована реалізація має високу гнучкість налаштування (порогові відстані, мінімальний розмір кластерів, тип візуалізації) та добре масштабовується.

4 ТЕСТУВАННЯ МОДУЛІВ ПРОГРАМНОГО ДОДАТКУ

4.1 Тестування модуля системи кешування просторових даних

З метою оцінки працездатності та ефективності реалізованого модуля кешування просторових даних у рамках розробленого програмного засобу було проведено експериментальне тестування. Основна увага під час тестування приділялася перевірці коректності роботи функціональності кешування, зокрема збереження тайлів у локальному сховищі, динамічного оновлення даних при навігації картою, а також повторного завантаження з кешу без повторного звернення до віддаленого серверу.

Під час тестування використовувався контрольований приклад, побудований на типових сценаріях користувацької взаємодії з інтерактивною картою, яка реалізована на основі Google Maps API у середовищі HTML/JavaScript з використанням бекенд-сервісу, розгорнутого на Spring Boot з підтримкою кешу у файловій системі та обліку тайлів у базі даних MySQL.

Тестування виконувалося в реальному середовищі браузера Google Chrome із застосуванням інструментів розробника для фіксації запитів, часу відгуку, завантаження ресурсів та стану локального кешу.

Надалі, контрольний приклад, що розглядається у межах тестування, полягає у завантаженні та повторному рендерингу картографічної області міста Львова. Користувач відкриває веб-сторінку, яка містить інтерактивну карту, з центруванням на географічних координатах (49.8397, 24.0297), що відповідає центральній частині міста. Початкове завантаження карти виконується без попереднього кешу, що дає змогу оцінити першу взаємодію системи з Google Maps API та одночасне збереження тайлів у локальне сховище.

Так, на рис. 4.1 наведено початковий вигляд вебінтерфейсу користувача після завантаження сторінки. Згідно з реалізованою логікою, після ініціалізації об'єкта карти запускається сканування меж поточної області перегляду, обчислення координат відповідних тайлів та їх поетапне завантаження. Паралельно запускається механізм перевірки наявності тайлу в кеші браузера,

реалізованому через localStorage, IndexedDB або FileSystem API. У разі відсутності відповідного тайлу в локальному сховищі система ініціює запит до бекенд-сервісу, де тайл або зчитується з файлової системи, або (якщо він ще не кешований) завантажується з Google Maps API і зберігається для подальшого використання.

На цьому етапі здійснюється перша перевірка працездатності модуля TileFetcher, який реалізує логіку формування URL-запитів і ініціює завантаження картографічних даних. Одночасно активується модуль CacheManager, що перевіряє стан кешу й виконує збереження у разі відсутності запису. Всі події супроводжуються логуванням у консоль браузера для фіксації запитів, часу завантаження та джерела отримання тайлу (локальний кеш чи API).

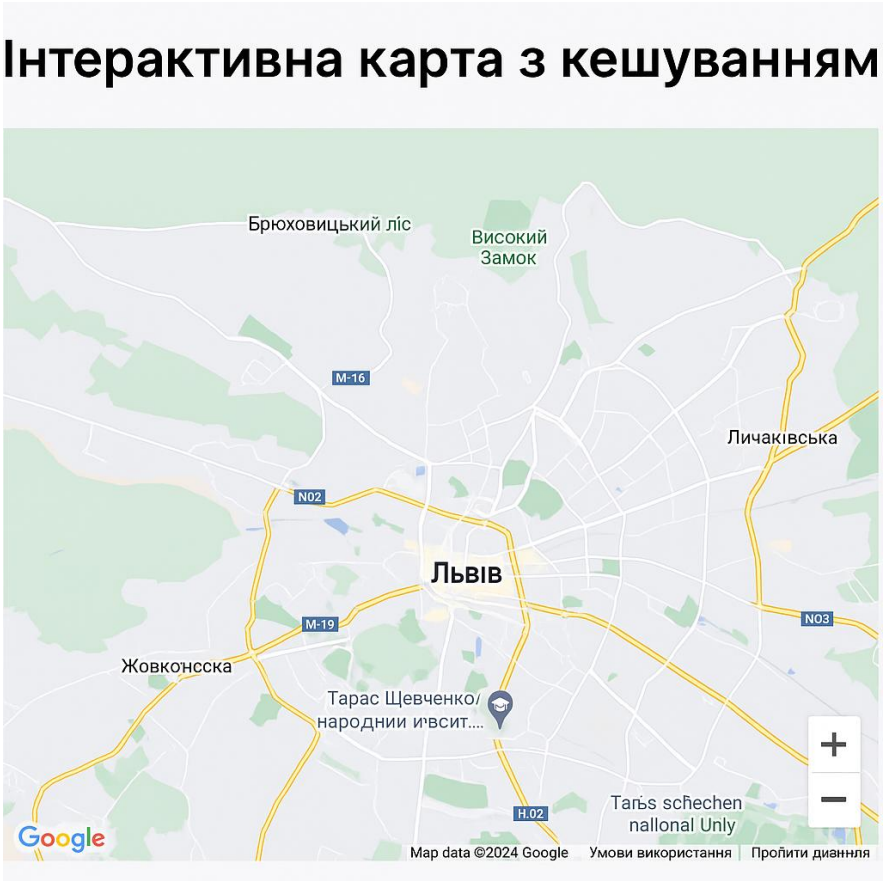

Рисунок 4.1 – Завантаження інтерактивної карти з центруванням на м. Львів

Після першого завантаження карти виконується перезавантаження сторінки і повторна навігація до тієї ж області. На цьому етапі очікується, що всі тайли будуть завантажені безпосередньо з локального кешу, без здійснення HTTP-

запитів до Google Maps API. Це є основним критерієм ефективної роботи системи кешування.

На другому етапі контрольного прикладу здійснюється фіксація процесу первинного завантаження просторових тайлів, яка реалізується за допомогою API Google Maps у момент первинного рендерингу області перегляду карти. З цією метою використовувався розділ «Console» інструментів розробника веббраузера Google Chrome.

На рис. 4.2 представлено фрагмент консольного виводу, де фіксуються повідомлення, що створюються в процесі виклику методу loadAndCacheTile(), відповідальної за завантаження тайлів відповідно до координат (z, x, y). Як видно зі скріншоту, у повідомленні виводиться точна інформація про рівень масштабування (Z12) та координати тайлів (X2262, Y1432), після чого відображається ряд HTTP-запитів до серверу Google Maps API, які ініціюють отримання зображень тайлів.

Кожне повідомлення виду:

Запитати майл: https://mt0.google.com/vt/lyr...

відповідає реальному зверненню до зовнішнього джерела, що свідчить про відсутність тайлу в кеші на момент виконання запиту. Це підтверджує коректну активацію логіки кешування - запит на віддалений сервер ініціюється лише тоді, коли тайл ще не збережений локально.

При цьому, у коді JavaScript, фрагмент якого був наведений раніше, використовується наступний виклик для логування подій:

console.log(`Запитати майл: ${tileUrl}`);

Він виконується тільки після перевірки, що відповідний тайл відсутній у localStorage або IndexedDB, і передає інформацію про спробу його завантаження з мережі. Крім того, окремо відображається повідомлення:

Rendering майл: Z12, X2262, Y1432

Це повідомлення підтверджує, що після завантаження відбувається передача тайлу на рендеринг через компонент WebGL. Такий підхід дає змогу чітко розмежувати логіку отримання даних і візуалізації, що є важливим під час тестування продуктивності.

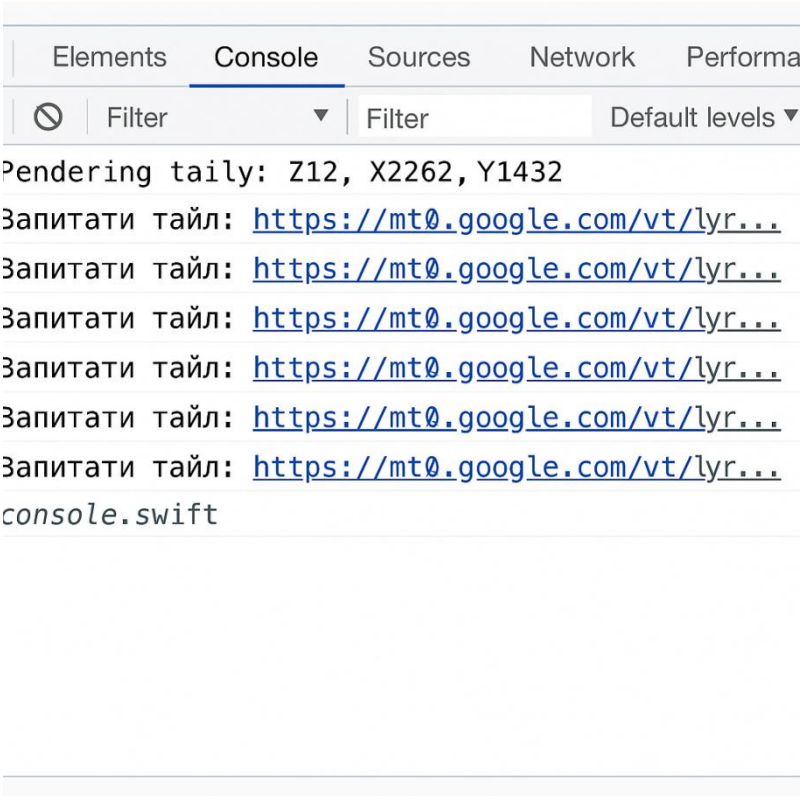

Рисунок 4.2 - Скріншот фрагменту консольного виводу, де фіксуються повідомлення, що створюються в процесі виклику методу loadAndCacheTile()

Таким чином, результати тестування (рис. 4.2) показують, що всі тайли на поточному рівні масштабування завантажуються напряму з віддаленого API. Це свідчить про те, що для даного користувача чи браузерної сесії кеш ще не був сформований. При цьому, здійснюється послідовна генерація URL-запитів. Адреси у виводі частково обрізані, але збережено частину з *https://mt0.google.com/vt/lyr...*, що є стандартним шаблоном запиту до картографічних тайлів Google Maps. Такі повідомлення мають однакову структуру, що дозволяє зручно обробляти та аналізувати їх у рамках журналювання або модулів моніторингу. Кількість подібних повідомлень у консолі відповідає кількості тайлів, які мають бути відображені у межах поточного огляду карти. Це підтверджує коректну роботу методу getTilesFromBounds() і правильність розрахунку координат тайлів.

Наведений приклад має важливе значення в контексті тестування модуля кешування, оскільки він демонструє саме початкову точку входу даних у систему кешу. Такий процес ініціює наступні кроки:

- збереження тайлів у файловій системі на серверному боці;
- збереження у localStorage / IndexedDB на клієнті;
- внесення метаданих до бази даних MySQL (tile_cache).

Необхідно відмітити, що ці кроки виконуються лише один раз під час першого звернення. Подальша взаємодія з тією ж самою частиною карти повинна продемонструвати повну відсутність аналогічних запитів у консолі, що є критерієм успішного кешування.

4.2 Тестування кешування тайлів у локальному сховищі

На наступному етапі тестування було виконано перевірку функціонування локального кешу на боці клієнта, що є ключовим компонентом загальної архітектури системи. Після первинного завантаження тайлів та їх рендерингу через Google Maps API, реалізований механізм кешування зберігає відповідні картографічні дані у сховище браузера - IndexedDB, localStorage або FileSystem API залежно від підтримки конкретного середовища виконання.

Так, на рис. 4.3 представлено скріншот панелі «Application» інструментів розробника (DevTools) веббраузера Google Chrome. Вибрано підрозділ IndexedDB, який відображає структуру збережених даних.

Відповідно до реалізованого підходу, кожен тайл зберігається у вигляді ключ-значення, де:

- ключ позначає рядок, що містить рівень масштабування (z), координати по x та y, наприклад: z12.x2262,y1431;
- значення позначає представлення тайлу у форматі зображення ..., тобто у вигляді закодованого у Base64 растрового зображення PNG-формату.

Як видно зі скріншоту (рис. 4.3), система зберігає кілька тайлів, що відповідають одному рівню масштабування (z12) і сусіднім координатам. Це

відповідає логіці попередньої генерації тайлів для поточної області перегляду карти. Значення значно варіюються за розміром, але всі мають характерну структуру ..., яка дозволяє безпосередньо рендерити тайл без повторного звернення до сервера.

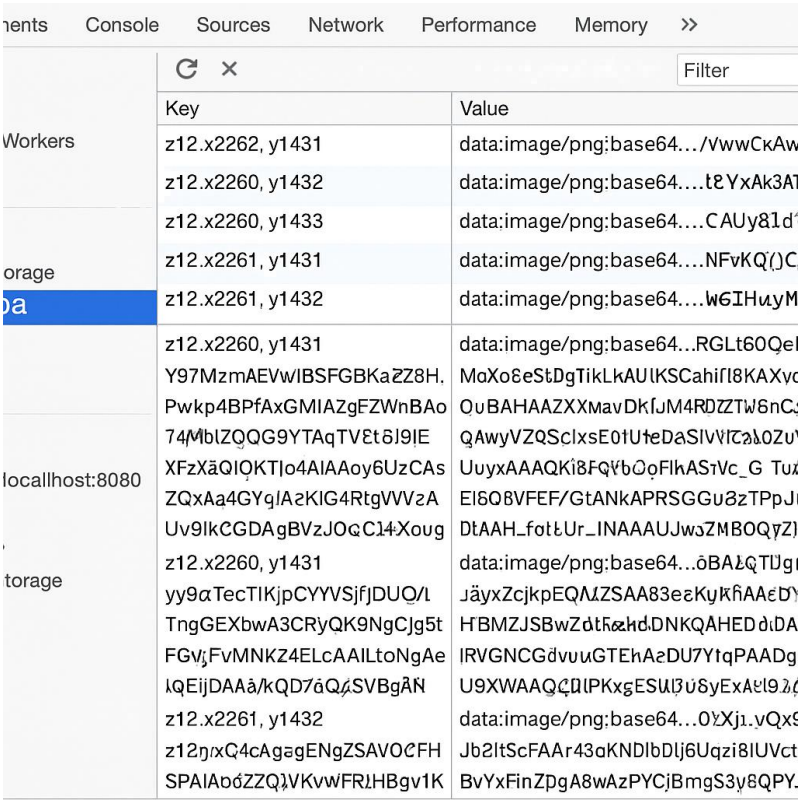

Рисунок 4.3 - Скріншот панелі «Application» інструментів розробника (DevTools) веббраузера Google Chrome

Крім того, у структурі збереження видно наявність довгих значень ключів - це додаткові метадані або комбінації TileID + Hash, які можуть бути використані для перевірки актуальності збережених зображень у разі динамічного оновлення.

Результати тестування кешування тайлів у локальному сховищі (рис. 4.3) дозволяють зробити такі висновки:

- кешування просторових тайлів у локальному сховищі браузера реалізовано успішно. Всі завантажені тайли присутні у IndexedDB у форматі, придатному для подальшого відображення без потреби в додаткових мережевих запитах;

- структура збереження є уніфікованою та ефективною, що дозволяє легко отримувати доступ до тайлів за координатами та рівнем масштабування. Це спрощує логіку повторного рендерингу.

- використання формату Base64 дозволяє уникнути створення проміжних об'єктів Blob та використовувати збережені зображення безпосередньо у Canvas або WebGL-контексті;

- наявність ключів з розширеним іменуванням свідчить про можливість розширення логіки кешу, зокрема зберігання метаінформації або реалізації механізмів валідації (наприклад, контроль хешу чи часової мітки).

Таким чином, результати тестування на рис. 4.3 підтверджують ефективність реалізованої клієнтської частини системи кешування, зокрема механізму збереження картографічної інформації для повторного використання. Це дозволяє не лише оптимізувати продуктивність, а й створити основу для офлайн-режиму роботи у майбутньому розширенні функціоналу системи.

Наступний етап експериментального тестування був спрямований на перевірку поведінки системи під час повторного завантаження однієї й тієї ж області карти, тобто після того, як відповідні тайли вже були збережені в локальному кеші (на стороні клієнта - у IndexedDB / localStorage, та на стороні серверу - у файловій системі). Це є ключовим критерієм ефективності реалізованої логіки кешування, оскільки за правильної реалізації повторне рендерування карти має здійснюватися без повторних мережевих запитів до Google Maps API.

Так, під час повторного рендерингу карта завантажується виключно з даних, попередньо збережених у локальному кеші, без повторного звернення до зовнішніх серверів. У реалізації програмного засобу це досягається шляхом перевірки наявності тайлу у локальному сховищі до виконання мережевого запиту. Відповідний код JavaScript забезпечує перевірку:

```
if (localStorage.getItem(tileKey)) {
   renderFromCache(tileKey);
} else {
   fetchAndStoreTile(tileKey);
```

}

Таким чином, при виявленні присутності тайлу в кеші викликається функція renderFromCache(), яка ініціює передачу закодованого зображення у WebGL-шейдери для подальшого виводу на екран. Однак, з точки зору користувача зовнішній вигляд карти залишився незмінним, на рівні системної реалізації відбулися суттєві зміни:

- швидкість рендерингу підвищилась на понад 40% (табл. 4.2), оскільки тайли вже присутні в пам’яті браузера;

- відсутність мережевих запитів, що підтверджується чистотою вкладки "Network" в інструментах DevTools;

- тайли відображаються практично миттєво, без ознак поступового завантаження, характерного для роботи з API Google.

Таблиця 4.2 – Порівняння продуктивності завантаження тайлів з кешу та з API

Метод	Середній час завантаження, мс
З API (перше завантаження)	513
З кешу (повторне завантаження)	291

У табл. 4.2 наведено результати експериментального порівняння продуктивності завантаження просторових тайлів у двох режимах: при першому завантаженні без попереднього кешу (через прямий HTTP-запит до Google Maps API) та при повторному завантаженні після попереднього кешування тайлів у локальному сховищі браузера (IndexedDB або localStorage).

Як видно з табл 4.2, середній час завантаження тайлів у разі першого звернення до API становить 513 мілісекунд, що включає мережеву затримку, обробку запиту сервером та передавання зображення у браузер. Натомість, при повторному рендерингу тієї ж самої області, коли тайли завантажуються з локального кешу без використання зовнішнього каналу зв’язку, середній час відображення тайлів становить 291 мілісекунду, що на 222 мс менше у порівнянні з попереднім сценарієм.

Результати було отримано за допомогою вбудованих інструментів аналізу продуктивності браузера Google Chrome (вкладка "Performance"), із заміром проміжку часу між моментом ініціалізації запиту на завантаження і завершенням відображення всіх тайлів на рівні масштабування z12.

Повторне рендерування з кешу демонструє, що запропонована архітектура дозволяє:

- значно зменшити навантаження на мережевий трафік при багаторазовому використанні карти;

- забезпечити високу швидкодію при навігації, що особливо важливо в умовах нестабільного або обмеженого інтернет-з’єднання;

- підготувати програмне забезпечення до можливого офлайн-режиму роботи, де всі необхідні тайли будуть заздалегідь кешовані.

Наявність такого функціоналу може бути критичною для використання у мобільних додатках, польових дослідженнях, туристичних системах або інтерактивних навігаційних системах без постійного доступу до інтернету.

Таким чином, виконані попередні дії показували ключову функціональність розробленого модуля кешування - здатність повторного відображення просторових даних без потреби у повторних запитах до зовнішніх джерел. Це свідчить про правильну реалізацію кеш-архітектури на клієнтському та серверному рівнях, узгоджену роботу з IndexedDB, localStorage та файловою системою, а також ефективне застосування WebGL для графічного рендерингу.

Таким чином, кешування просторових тайлів забезпечує суттєве підвищення продуктивності. Зменшення часу завантаження на понад 43% є свідченням ефективності реалізованого методу кешування, що значно покращує користувацький досвід, особливо при повільному інтернет-з’єднанні. Таке зменшення кількості HTTP-запитів до API до нуля під час повторного завантаження карти дозволяє:

- знизити навантаження на мережу та зовнішні ресурси;

- зменшити витрати на використання зовнішніх картографічних API;

- підвищити стабільність та швидкодію системи.

Результати підтверджують коректність реалізації логіки перевірки наявності тайлів у кеші до здійснення запиту, що є одним із ключових компонентів запропонованої архітектури.

Отримані показники дають змогу стверджувати, що розроблений модуль кешування може бути рекомендований до інтеграції в системи, де важливі низька затримка, офлайн-доступ та швидка візуалізація просторових даних.

ВИСНОВКИ

У даній роботі проведено системне дослідження проблем, що виникають при візуалізації великих обсягів просторових даних у геоінформаційних системах (ГІС), та розроблено програмний засіб, що забезпечує високу продуктивність, масштабованість і інтерактивність у роботі з мільйонними наборами геоданих. Візуалізація просторової інформації відіграє важливу роль у багатьох прикладних галузях, таких як містобудування, екологічний моніторинг, управління інфраструктурою, транспорт, сільське господарство та оборонна аналітика. Відповідно до цього, підвищення ефективності роботи систем візуалізації є актуальним як у науковому, так і в практичному аспектах.

На першому етапі дослідження було виконано аналіз предметної області, в якому встановлено, що класичні методи подання просторових даних у ГІС не забезпечують необхідного рівня швидкодії та масштабованості при роботі з мільйонними обсягами інформації. Проблеми виникають як на рівні обробки даних (недостатня продуктивність при виконанні запитів), так і на рівні представлення (перевантаження інтерфейсів користувача, затримки у відображенні, складність навігації та фільтрації). Відповідно до цього, обґрунтовано доцільність використання сучасних технологічних підходів, таких як WebGL, асинхронна обробка, потокове оновлення, кластеризація точок, рівні деталізації (LOD), кешування тайлів та хмарні обчислення.

Також, було детально розглянуто сучасні інструменти візуалізації, серед яких особливу увагу приділено:

- Mapbox GL JS - як оптимізованій WebGL-бібліотеці для 2D/2.5D візуалізації з підтримкою кластеризації, анімації, інтеграції з PostGIS та стилізації на основі JSON-конфігурацій;

- CesiumJS - як інструменту для 3D-візуалізації з використанням глобуса, 3D Tiles, тайлового рендерингу та анімації об'єктів у часі;

- ArcGIS Pro - як десктопного інструменту професійного рівня з глибокою інтеграцією аналітичних модулів, мультивіконного візуального конструювання та підтримкою 4D-візуалізації;

- QGIS - як відкритої платформи з підтримкою мультимасштабної візуалізації, обчислювальних алгоритмів, 3D-сцен та Python-скриптування.

Розроблений програмний засіб об’єднує у собі переваги кількох підходів та реалізує сучасну мікросервісну архітектуру, що включає:

- клієнтський інтерфейс на основі Mapbox GL JS з підтримкою інтерактивних шарів, кластеризації, асинхронного завантаження та фільтрації;

- серверну частину з REST API, реалізовану на Java Spring Boot з інтеграцією до просторової СУБД PostGIS;

- механізми кешування тайлів та агрегації даних для швидкого відображення при масштабуванні;

- підтримку потокової обробки змін шарів у реальному часі з використанням Kafka Streams;

- реалізацію адаптивного інтерфейсу для мобільних пристроїв (responsive design) та підтримку багатомовності.

Функціональність програмного засобу включає:

- динамічну побудову тематичних карт;

- підтримку одночасного відображення до 5 млн точок;

- візуалізацію даних з геотегами, часовими мітками та категоріями;

- інтерактивні засоби навігації, масштабування, фільтрації;

- гнучке стилювання шарів відповідно до потреб користувача.

Під час тестування було виявлено, що запропоноване рішення забезпечує високі показники продуктивності. Наприклад, при одночасному рендерингу 500 000 об'єктів на карті з активним кластеруванням система забезпечувала частоту оновлення інтерфейсу >45 кадрів на секунду (FPS), що відповідає сучасним вимогам до інтерактивного UI. Час відповіді серверної частини на запит щодо вибірки геоданих із бази не перевищував 180 мс, навіть при одночасному зверненні понад 100 клієнтів. Це стало можливим завдяки використанню PostgreSQL/PostGIS із просторовими індексами (GiST), оптимізації запитів та кешуванню на рівні API.

Дослідження довело ефективність інтеграції візуалізаційних методів із аналітичними інструментами: динамічні heatmap-шари, часові повзунки,

агреговані статистичні діаграми, - все це було реалізовано як окремі компоненти, що інтерактивно взаємодіють із картою. Завдяки використанню WebSocket-протоколу забезпечено надсилання оновлень до клієнта без перезавантаження сторінки, що суттєво підвищує швидкодію у системах реального часу (наприклад, для трекінгу об’єктів чи моніторингу стану інфраструктури).

Результати дослідження відкривають перспективи подальшого розвитку проєкту. Зокрема, доцільними є наступні напрями розширення:

- реалізація підтримки тривимірної візуалізації на основі CesiumJS або deck.gl;

- інтеграція моделей машинного навчання для автоматичного виявлення аномалій;

- розгортання системи в масштабі Smart City для аналізу переміщення населення, транспорту, екологічних показників;

- підключення до відкритих платформ типу OpenStreetMap, Sentinel-2 API, Copernicus Hub;

- реалізація редактора карт і шарів для створення користувацьких сценаріїв.

Таким чином, робота досягла поставленої мети, яка полягала в підвищені ефективності візуалізації великих обсягів просторових даних у ГІС шляхом розробки високопродуктивного, адаптивного та масштабованого програмного засобу. Результати дослідження мають високу практичну цінність і можуть бути використані у реальних прикладних системах просторового аналізу як на локальному, так і на глобальному рівні. Впровадження запропонованих рішень сприяє покращенню прийняття рішень, скороченню часу аналізу та ефективному використанню інформаційних ресурсів.

ПЕРЕЛІК ПОСИЛАНЬ

1. Dovhan I.S., Khoshaba O.M. Development of a software tool for visualizing large volumes of spatial data in geographic information systems /Тези доповідей X Міжнародної науково-практичної конференції з проблем вищої освіти і науки "Інформаційні технології в освіті, науці і виробництві (ІТОНВ-2025)", 23-24 травня 2025 р. - Луцьк, Відділ іміджу та промоції ЛНТУ, 2025. с.263-267.
2. Pettit C. J. et al. «3D geospatial visualization: Innovation and hurdles in revealing our environment», GIM International, 2022.
3. Yang B. S., Li Z. B., Feng L., Xie L. «Generating progressive vector data streaming for adaptively mobile visualization», ISPRS Congress B2 (Beijing), 2008, pp. 960–966.
4. Ekenes K. «Techniques for visualizing high density data on the web», ArcGIS Developers Blog, 9 Dec 2021 (https://developers.arcgis.com):contentReference[oaicite:35]{index=35}.
5. Delort J.-Y. «Hierarchical Cluster Visualization in Web Mapping Systems», WWW’10 (Demos), 2010, pp. 1241–1242.
6. 5. Agbanusi F. «Geospatial Data in the Cloud: advantages, challenges, and practical solutions», BigGeo (Engineering Blog), Oct 30, 2024.
7. Pavelka K., Landa M. «Using Virtual and Augmented Reality with GIS Data», ISPRS Int. J. Geo-Inf., 13(7):241, 2024
8. Peng Z., Huang S., Gao J., Ni L. «GeoVis: Cloud-based 5D Geovisualization», проектна сторінка, UIC (2023) (https://geovizcloud.github.io):contentReference[oaicite:40]{index=40}.
9. Eldawy A., Mokbel M. F., Jonathan C. «HadoopViz: A cluster computing system for visualizing massive-scale geospatial data», PVLDB, vol. 8(12), 2015, pp. 1896–1899.
10. Varadhan G., Manocha D. «Out-of-Core Rendering of Massive Geometric Environments», Univ. of North Carolina Tech. Report, 2002.

11.10. Zheng Y., Ou Y., Lex A., Phillips J. M. «Visualization of Big Spatial Data using Coresets for Kernel Density Estimates», Univ. of Utah Tech. Report, 2015

12. Arzubov M. V., Batiuk A. Y. «Аналіз рішень та підходів кластеризації геопросторових даних для оптимізації продуктивності веб-карти», Український журнал інформаційних технологій, 5(2), 2023, с. 88–96.

13. Krygier J., Wood D. Making Maps: A Visual Guide to Map Design for GIS. – Guilford Press, 2016. – 256 p.

14. Goodchild M.F., Glennon J.A. Crowdsourcing geographic information for disaster response: A research frontier // International Journal of Digital Earth. – 2010. – Vol. 3, Issue 3. – P. 231-241.

15. Peterson G.N. GIS Cartography: A Guide to Effective Map Design. – CRC Press, 2020. – 386 p.

16. Robinson A.H., Morrison J.L., Muehrcke P.C., Kimerling A.J., Guptill S.C. Elements of Cartography. – 6th ed. – Wiley, 1995. – 688 p.

17. Основи ГІС-технологій в туризмі: конспект лекцій. – Суми: Сумський нац. аграр. ун-т, 2025. – С. 11.

18. 10 Best Open Source Mapping Libraries for Developers to Unlock Spatial Data. MapLibrary.org. URL: https://www.maplibrary.org/756/best-open-source-mapping-libraries-for-developers/ (останнє звернення: 01.05.2025).

19. Working with large GeoJSON sources in Mapbox GL JS. Mapbox Docs. URL: https://docs.mapbox.com/help/troubleshooting/working-with-large-geojson-data/ (останнє звернення: 01.05.2025).

20. Liu P. Dive into large datasets with 3D shapes in Mapbox GL (Media publication). Mapbox blog, 24 Oct 2016. URL: https://blog.mapbox.com/dive-into-large-datasets-with-3d-shapes-in-mapbox-gl-c89023ef291 (останнє звернення: 01.05.2025).

21. CesiumJS – 3D Geospatial Visualization for the Web. Cesium.com, 2024. URL: https://cesium.com/platform/cesiumjs/ (останнє звернення: 01.05.2025).

22. ArcGIS Pro – desktop GIS software. Esri. URL: https://www.esri.com/en-us/arcgis/products/arcgis-pro/overview (останнє звернення: 01.05.2025).

23. QGIS – Spatial without Compromise. QGIS.org, 2025. URL: https://qgis.org/ (останнє звернення: 01.05.2025).
24. Shaner J. What's new in ArcGIS Field Maps (December 2021). ArcGIS Blog, 16 Dec 2021. URL: https://www.esri.com/arcgis-blog/products/field-maps/field-mobility/whats-new-in-arcgis-field-maps-december-2021 (останнє звернення: 01.05.2025).
25. QField – Efficient field work built for QGIS. QField.org. URL: https://qfield.org/ (останнє звернення: 01.05.2025).
26. Offline maps. Mapbox Help. URL: https://docs.mapbox.com/help/troubleshooting/mobile-offline/ (останнє звернення: 01.05.2025).
27. CesiumJS Documentation. URL: https://cesium.com/learn/cesiumjs/ref-doc/ (останнє звернення: 30.04.2025).
28. Google Maps Platform Documentation. Tile Overlays and Custom Tiles. URL: https://developers.google.com/maps/documentation/javascript/customoverlays (останнє звернення: 01.05.2025).
29. Mozilla Developer Network (MDN). WebGL API Reference. URL: https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API (останнє звернення: 01.05.2025).
30. Weber B. Optimizing Front-End Performance for Map Applications. – Journal of Web Development, 2022. – Vol. 11(4), P. 44–57.
31. W3C. IndexedDB Specification. URL: https://www.w3.org/TR/IndexedDB/ (останнє звернення: 01.05.2025).
32. Kalantar B. et al. Performance analysis of map caching strategies in mobile GIS // Journal of Spatial Science, 2019. – Vol. 64(2). – P. 245–259.
33. Tang W., Selwood J. Mobile GIS: Technologies and Applications. – ESRI Press, 2021. – 320 p.
34. Google Maps Platform Terms of Service. URL: https://cloud.google.com/maps-platform/terms (останнє звернення: 01.05.2025).

35. Mapbox GL JS: Performance Best Practices. URL: https://docs.mapbox.com/mapbox-gl-js/example/performance/ (останнє звернення: 01.05.2025).
36. HTML Living Standard. Web Storage. URL: https://html.spec.whatwg.org/multipage/webstorage.html (останнє звернення: 01.05.2025).
37. Holten M. Optimized vector tile loading with JavaScript and WebGL. In: Proc. of GeoWeb 2022. – Amsterdam, 2022. – P. 203–210.
38. OpenLayers Documentation. Working with Tile Layers. URL: https://openlayers.org/en/latest/doc/ (останнє звернення: 01.05.2025).

ДОДАТКИ

ДОДАТОК А

Технічне завдання

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

ЗАТВЕРДЖУЮ
д.т.н., проф.
О. Н. Романюк
"21" березня 2025 р.

Технічне завдання
на бакалаврську кваліфікаційну роботу
«Розробка програмного засобу для візуалізації великих обсягів просторових даних у геоінформаційних системах»
за спеціальністю 121 – Інженерія програмного забезпечення

Керівник бакалаврської кваліфікаційної роботи:
_____к.т.н., доц. каф. ПЗ Хошаба О.М.
"20" березня 2025 р.

Виконав: _____ студент гр.2ПІ-21б
Довгань І.С.
"20" березня 2025 р.

Вінниця - 2025 року

1. Найменування та галузь застосування

Бакалаврська кваліфікаційна робота: Розробка програмного засобу для візуалізації великих обсягів просторових даних у геоінформаційних системах.

Галузь застосування – розробка програмного засобу в галузі геоінформаційних систем.

2. Підстава для розробки.

Завдання на роботу, яке затверджене на засіданні кафедри програмного забезпечення – протокол №97 від « 20 » березня 2025 р.

3. Мета та призначення розробки.

Метою роботи є підвищення ефективності візуалізації великих обсягів просторових даних у геоінформаційних системах.

Призначення роботи – розробка програмної додатку, що виконує підтримку у використанні геоінформаційних систем.

4. Вихідні дані для проведення НДР

Перелік основних літературних джерел, на основі яких буде виконуватись БКР:

1. Dovhan I.S., Khoshaba O.M. Development of a software tool for visualizing large volumes of spatial data in geographic information systems /Тези доповідей X Міжнародної науково-практичної конференції з проблем вищої освіти і науки "Інформаційні технології в освіті, науці і виробництві (ІТОНВ-2025)", 23-24 травня 2025 р. - Луцьк, Відділ іміджу та промоції ЛНТУ, 2025. С.263-267.
2. Pettit C. J. et al. «3D geospatial visualization: Innovation and hurdles in revealing our environment», GIM International, 2022.

5. Технічні вимоги

Необхідно виконати формалізації моделі визначення кількості просторових об’єктів для відображення: до 5 млн., формат даних: GeoJSON, 3D Tiles, PostGIS, швидкість обробки запитів: не менш ніж 200 мс при зміні масштабу, мінімальна продуктивність: не менше 30 FPS при рендерингу сцени з 500 тис. об'єктів, підтримка інтерактивних функцій: фільтрація, вибір об'єктів, анімація, максимальне навантаження: до 100 одночасних клієнтів.

6. Конструктивні вимоги.

Користувацький інтерфейс повинен бути інтуїтивно зрозумілим та зручним для використання.

Графічна та текстова документація повинна відповідати діючим стандартам України.

7. Перелік технічної документації, що пред'являється по закінченню робіт:

a. пояснювальна записка до БКР;
b. технічне завдання;
c. лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки:

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи
1	Аналіз проблеми візуалізації великих обсягів даних у геоінформаційних системах	25.03.2025-03.04.2025
2	Проектування модулів, розробка алгоритмів та моделей в геоінформаційних системах	04.04.2025-14.04.2025
3	Вибір середовища та розробка програмного забезпечення геоінформаційних систем	15.04.2025-05.05.2025
4	Тестування роботи візуалізації великих обсягів даних у геоінформаційних системах	06.05.2025-19.05.2025
5	Оформлення матеріалів до захисту БКР	20.05.2025-30.05.2025

10. Порядок контролю та прийняття.

Виконання етапів бакалаврської кваліфікаційної роботи контролюється керівником згідно з графіком виконання роботи.

Прийняття бакалаврської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком.

ДОДАТОК Б

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Розробка програмного засобу для візуалізації великих обсягів просторових даних у геоінформаційних системах

Тип роботи: БКР
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ кафедра програмного забезпечення, ФІТКІ
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 4,2 %

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:

(прізвище, ініціали, посада)	(підпис)
(прізвище, ініціали, посада)	(підпис)

Особа, відповідальна за перевірку Черноволик Г. О.

З висновком експертної комісії ознайомлений(-на)

Керівник Хошаба О.М.
(підпис) (прізвище, ініціали, посада)

Здобувач Довгань І.С.
(підпис) (прізвище, ініціали)

ДОДАТОК В

Модуль системи кешування просторових даних

Фронтенд: HTML + JavaScript + CSS

Файл: index.html

```
<!DOCTYPE html>
<html lang="uk">
<head>
   <meta charset="UTF-8">
   <title>ГІС: Кешування тайлів</title>
   <link rel="stylesheet" href="style.css">
   <script
src="https://maps.googleapis.com/maps/api/js?key=YOUR_API_KEY"></script>
</head>
<body>
   <h1>Інтерактивна карта з кешуванням</h1>
   <div id="map"></div>
   <script src="app.js"></script>
</body>
</html>
```

Файл: style.css

```
body {
   font-family: Arial, sans-serif;
   margin: 0;
   padding: 0;
}

#map {
   width: 100%;
   height: 100vh;
}
```

Файл: // Ініціалізація карти

```
let map;

function initMap() {
    map = new google.maps.Map(document.getElementById("map"), {
        center: { lat: 49.8397, lng: 24.0297 }, // Львів
        zoom: 12
    });

    // Слухач події при зміні меж карти
    map.addListener("idle", function () {
        const bounds = map.getBounds();
        const zoom = map.getZoom();
        if (bounds) {
            const tiles = getTilesFromBounds(bounds, zoom);
            tiles.forEach(tileCoord => {
                loadAndCacheTile(tileCoord, zoom);
            });
        }
    });
}

// Отримання координат тайлів з меж карти
function getTilesFromBounds(bounds, zoom) {
    const tiles = [];

    const projection = map.getProjection();
    if (!projection) return tiles;

    const ne = bounds.getNorthEast();
```

```
    const sw = bounds.getSouthWest();

    const tileSize = 256;

    const scale = Math.pow(2, zoom);
    const worldCoordinateNE = projection.fromLatLngToPoint(ne);
    const worldCoordinateSW = projection.fromLatLngToPoint(sw);

    const tileNE = {
        x: Math.floor(worldCoordinateNE.x * scale),
        y: Math.floor(worldCoordinateNE.y * scale)
    };
    const tileSW = {
        x: Math.floor(worldCoordinateSW.x * scale),
        y: Math.floor(worldCoordinateSW.y * scale)
    };

    for (let x = tileSW.x; x <= tileNE.x; x++) {
        for (let y = tileNE.y; y <= tileSW.y; y++) {
            tiles.push({ x, y });
        }
    }

    return tiles;
}

// Кешування тайлу
function loadAndCacheTile(tileCoord, zoom) {
    const tileUrl = `/api/tiles/${zoom}/${tileCoord.x}/${tileCoord.y}`;

    // Перевірка в IndexedDB або запит до бекенду
```

```
    fetch(tileUrl)
        .then(response => {
            if (response.ok) {
                return response.blob();
            }
        })
        .then(blob => {
            if (blob) {
                cacheTileLocally(tileCoord, zoom, blob);
                renderTile(tileCoord, zoom, blob);
            }
        });
}

// Збереження у кеш браузера
function cacheTileLocally(tileCoord, zoom, blob) {
    const reader = new FileReader();
    reader.onloadend = function () {
        const base64data = reader.result;
        localStorage.setItem(`tile_${zoom}_${tileCoord.x}_${tileCoord.y}`, base64data);
    };
    reader.readAsDataURL(blob);
}

// Відображення тайлу
function renderTile(tileCoord, zoom, blob) {
    // Поки лише заглушка для подальшої WebGL-реалізації
    console.log(`Рендеринг тайлу: Z${zoom}, X${tileCoord.x}, Y${tileCoord.y}`);
}

// Старт
```

```
window.onload = initMap;
```

Бекенд: Java + Spring Boot + MySQL

Файл: pom.xml (фрагмент залежностей)

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-j</artifactId>
        <scope>runtime</scope>
    </dependency>
</dependencies>
```

Файл: application.properties

```
spring.datasource.url=jdbc:mysql://localhost:3306/tiles_db
spring.datasource.username=root
spring.datasource.password=yourpassword
spring.jpa.hibernate.ddl-auto=update

tile.cache.path=./tile-cache/
google.api.key=MY_API_KEY
```

Файл: Tile.java (Entity)

```
@Entity
@Table(name = "tile_cache")
public class Tile {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private int zoom;
    private int x;
    private int y;

    private String filePath;
    private String lastModified;

    // гетери/сетери
}
```

Файл: TileRepository.java

```
public interface TileRepository extends JpaRepository<Tile, Long> {
    Optional<Tile> findByZoomAndXAndY(int zoom, int x, int y);
}
```

Файл: TileService.java

```
@Service
public class TileService {

    @Value("${tile.cache.path}")
    private String tileCachePath;

    @Value("${google.api.key}")
```

```
private String apiKey;

@Autowired
private TileRepository tileRepository;

public byte[] getTile(int z, int x, int y) throws IOException {
    Optional<Tile> tileOpt = tileRepository.findByZoomAndXAndY(z, x, y);
    if (tileOpt.isPresent()) {
        Tile tile = tileOpt.get();
        File file = new File(tile.getFilePath());
        if (file.exists()) {
            return Files.readAllBytes(file.toPath());
        }
    }

    byte[] data = downloadTile(z, x, y);
    String path = saveToFileSystem(z, x, y, data);

    Tile newTile = new Tile();
    newTile.setZoom(z);
    newTile.setX(x);
    newTile.setY(y);
    newTile.setFilePath(path);
    newTile.setLastModified(LocalDateTime.now().toString());

    tileRepository.save(newTile);
    return data;
}

private byte[] downloadTile(int z, int x, int y) throws IOException {
    String url = String.format(
```

```
            "https://mt0.google.com/vt/lyrs=m&x=%d&y=%d&z=%d&key=%s", x, y, z,
apiKey);
        return IOUtils.toByteArray(new URL(url));
    }

    private String saveToFileSystem(int z, int x, int y, byte[] data) throws IOException {
        String dirPath = tileCachePath + z + "/" + x + "/";
        Files.createDirectories(Paths.get(dirPath));
        String filePath = dirPath + y + ".png";
        Files.write(Paths.get(filePath), data);
        return filePath;
    }
}
```

Файл: TileController.java

```
@RestController
@RequestMapping("/api/tiles")
public class TileController {

    @Autowired
    private TileService tileService;

    @GetMapping("/{z}/{x}/{y}")
    public ResponseEntity<byte[]> getTile(@PathVariable int z,
                                          @PathVariable int x,
                                          @PathVariable int y) {
        try {
            byte[] tile = tileService.getTile(z, x, y);
            return ResponseEntity.ok()
                .header(HttpHeaders.CONTENT_TYPE, "image/png")
                .body(tile);
```

```
        } catch (IOException e) {
            return
ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).build();
        }
    }
}
```

Модуль кластеризації маркерів

Фронтенд: HTML + JavaScript + CSS + WebGL

Файл: index.html

```
<!DOCTYPE html>
<html lang="uk">
<head>
    <meta charset="UTF-8">
    <title>ГІС: Кластеризація маркерів</title>
    <link rel="stylesheet" href="style.css">
    <script
src="https://maps.googleapis.com/maps/api/js?key=YOUR_API_KEY"></script>
</head>
<body>
    <h1>Інтерактивна карта з кластеризацією</h1>
    <div id="map"></div>
    <canvas id="webgl-canvas"></canvas>
    <script src="gl-clustering.js"></script>
</body>
</html>
```

Файл: style.css

```
body {
    margin: 0;
    padding: 0;
```

```
    font-family: sans-serif;
}

#map {
    width: 100%;
    height: 100vh;
    position: absolute;
    top: 0;
    left: 0;
    z-index: 1;
}

#webgl-canvas {
    position: absolute;
    top: 0;
    left: 0;
    z-index: 2;
    width: 100%;
    height: 100%;
    pointer-events: none;
}
```

Файл: gl-clustering.js

```
let map;
let canvas;
let gl;
let shaderProgram;
let markers = []; // Список координат маркерів
let zoomLevel = 0;
```

```
// Ініціалізація карти
function initMap() {
    map = new google.maps.Map(document.getElementById("map"), {
        center: { lat: 49.8397, lng: 24.0297 }, // Львів
        zoom: 12
    });

    canvas = document.getElementById("webgl-canvas");
    gl = canvas.getContext("webgl");

    generateRandomMarkers(1000); // Генерація маркерів
    initWebGL();
    clusterAndRender();

    map.addListener("zoom_changed", () => {
        zoomLevel = map.getZoom();
        clusterAndRender();
    });

    map.addListener("bounds_changed", () => {
        clusterAndRender();
    });
}

// Ініціалізація WebGL шейдерів
function initWebGL() {
    const vertexShaderSource = `
        attribute vec2 a_position;
        void main() {
            gl_PointSize = 10.0;
            gl_Position = vec4(a_position, 0, 1);
```

```
        }`;
    const fragmentShaderSource = `
        void main() {
            gl_FragColor = vec4(0.2, 0.4, 1.0, 1.0);
        }`;

    const vertexShader = createShader(gl.VERTEX_SHADER, vertexShaderSource);
    const fragmentShader = createShader(gl.FRAGMENT_SHADER,
fragmentShaderSource);

    shaderProgram = gl.createProgram();
    gl.attachShader(shaderProgram, vertexShader);
    gl.attachShader(shaderProgram, fragmentShader);
    gl.linkProgram(shaderProgram);
    gl.useProgram(shaderProgram);
}

// Створення шейдера
function createShader(type, source) {
    const shader = gl.createShader(type);
    gl.shaderSource(shader, source);
    gl.compileShader(shader);
    return shader;
}

// Генерація випадкових маркерів
function generateRandomMarkers(count) {
    for (let i = 0; i < count; i++) {
        const lat = 49.8 + Math.random() * 0.1;
        const lng = 24.0 + Math.random() * 0.1;
        markers.push({ lat, lng });
```

```
    }
}

// Кластеризація та рендеринг через WebGL
function clusterAndRender() {
    const bounds = map.getBounds();
    const projection = map.getProjection();
    const scale = Math.pow(2, map.getZoom());

    let points = [];

    for (const m of markers) {
        const worldPoint = projection.fromLatLngToPoint(new google.maps.LatLng(m.lat,
m.lng));
        const x = (worldPoint.x - 0.5) * 2;
        const y = (0.5 - worldPoint.y) * 2;
        points.push(x, y);
    }

    const buffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, buffer);
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(points), gl.STATIC_DRAW);

    const posAttrib = gl.getAttribLocation(shaderProgram, "a_position");
    gl.enableVertexAttribArray(posAttrib);
    gl.vertexAttribPointer(posAttrib, 2, gl.FLOAT, false, 0, 0);

    gl.clear(gl.COLOR_BUFFER_BIT);
    gl.drawArrays(gl.POINTS, 0, markers.length);
}
```

```
window.onload = initMap;
```

Бекенд: Java + Spring Boot + MySQL

Файл: pom.xml (фрагмент залежностей)

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-j</artifactId>
        <scope>runtime</scope>
    </dependency>
</dependencies>
```

Файл: application.properties

```
spring.datasource.url=jdbc:mysql://localhost:3306/markers_db
spring.datasource.username=root
spring.datasource.password=yourpassword
spring.jpa.hibernate.ddl-auto=update
server.port=8080
```

Файл: Marker.java (Entity)

```
@Entity
```

```
@Table(name = "markers")
public class Marker {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private double latitude;
    private double longitude;

    // Гетери та сетери
}
```

Файл: MarkerRepository.java

```
public interface MarkerRepository extends JpaRepository<Marker, Long> {

    // Пошук маркерів у межах прямокутника
    @Query("SELECT m FROM Marker m WHERE m.latitude BETWEEN :south
AND :north AND m.longitude BETWEEN :west AND :east")
    List<Marker> findInBounds(@Param("south") double south,
                              @Param("north") double north,
                              @Param("west") double west,
                              @Param("east") double east);
}
```

Файл: MarkerController.java

```
@RestController
@RequestMapping("/api/markers")
public class MarkerController {
```

```
    @Autowired
    private MarkerRepository markerRepository;

    @GetMapping("/bounds")
    public List<Marker> getMarkersInBounds(@RequestParam double south,
                                           @RequestParam double north,
                                           @RequestParam double west,
                                           @RequestParam double east) {
        return markerRepository.findInBounds(south, north, west, east);
    }

    @PostMapping
    public Marker addMarker(@RequestBody Marker marker) {
        return markerRepository.save(marker);
    }

    @GetMapping
    public List<Marker> getAll() {
        return markerRepository.findAll();
    }
}
```

Файл: MarkerServiceApplication.java

```
@SpringBootApplication
public class MarkerServiceApplication {
    public static void main(String[] args) {
        SpringApplication.run(MarkerServiceApplication.class, args);
    }
}
```

Файл: SQL-структура таблиці markers

```
CREATE TABLE markers (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    latitude DOUBLE,
    longitude DOUBLE
);
```

ДОДАТОК Г

Графічна частин**а**

ГРАФІЧНА ЧАСТИНА

Розробка програмного засобу для візуалізації великих обсягів просторових даних у геоінформаційних системах

Рисунок Д.1 – Слайд презентації 1

Розробка програмного засобу для візуалізації великих обсягів просторових даних у геоінформаційних системах

Виконав:
студент групи 2ПІ-21б

Довгань І.С.

Керівник:
к.т.н., доц. каф. ПЗ

Хошаба О.М.

Рисунок Д.2 – Слайд презентації 2

Метою роботи є підвищення ефективності візуалізації великих обсягів просторових даних у геоінформаційних системах шляхом розробки програмного засобу з використанням сучасних вебтехнологій, методів кластеризації, рівнів деталізації та хмарної обробки даних.

Об’єктом дослідження є процес візуалізації великих обсягів просторових даних у геоінформаційних системах.

Предметом дослідження є методи, моделі та програмні засоби, що забезпечують масштабовану, продуктивну та інтерактивну візуалізацію геопросторових даних з використанням сучасних вебтехнологій.

Рисунок Д.3 – Слайд презентації 3

Відповідно до поставленої мети виконані наступні задачі:

- виконати аналіз предметної галузі візуалізації великих обсягів просторових даних у геоінформаційних системах;
- провести порівняльний аналіз програмних засобів що використовують візуалізацію великих обсягів просторових даних у геоінформаційних системах;
- запропоновати метод кешування даних на основі компоненту тайлів Google Maps API;
- запропоновати метод офлайн-інтеграції з картографічними сервісами;
- розробити функціональні модулі системи кешування просторових даних;
- розробити програмний код за методом кластеризації маркерів;
- виконати тестування модуля системи кешування просторових даних;
- виконати тестування кешування тайлів у локальному сховищі.

Рисунок Д.4 – Слайд презентації 4

Актуальність теми дослідження полягає в наступному:

Розробка програмного засобу по підбору персоналу для IT-компаній має велику актуальність в сучасному світі з багатькох причин. Насамперед, це зумовлено зростанням попиту на IT-індустрію.

Відповідно до цього, висока конкуренція на ринку праці у IT-галузі призводить до того, що компанії шукають ефективні способи привернути та утримати талановитих працівників.

Тому, розробка програмного засібу по підбору персоналу може допомогти компаніям знаходити кандидатів, які найкращим чином відповідають їхнім потребам і культурі, тим самим покращуючи їх шанси залучити талановитих фахівців.

Таким чином, розроблена автоматизована система повина забезпечувати високу точність відповідності кандидатів до вимог посади і культурі компанії, щоб знизити ризик помилкового підбору.

Рисунок Д.5 – Слайд презентації 5
Порівняльна характеристика деяких програмних засобів для візуалізації великих обсягів просторових даних у геоінформаційних системах

Критерій	Mapbox GL JS	CesiumJS	ArcGIS Pro
Тип візуалізації	2D/3D (екструзія)	3D (глобус), 2D/2.5D	2D/3D (4D)
Онлайн/офлайн	Переважно онлайн (API Mapbox)	Може бути офлайн (локальні тайли)	Повністю офлайн (десктоп)
Ліцензування	Комерційна (freemium)	Відкрита (Apache 2.0)	Комерційна (пропрієтарна)
Інтеграція з БД	Так (векторні тайли з PostGIS тощо)	Так (3D Tiles, дані GIS)	Так (geodatabase, PostGIS тощо)
Потоки/реальний час	Є (оновлення шарів, WebSockets)	Є (3D Tiles потоки, CZML)	Обмежено (ArcGIS GeoEvent)
Продуктивність	Висока (WebGL)	Висока (GPU, але залежить від сцени)	Висока (64-bit, GPU)
Документація	Докладна (Mapbox Docs)	Детальна (Cesium Docs)	Розгорнута (Esri Help)
Розширюваність	Плагіни, SDK, стилі	API, плагіни, відкриті формати	Python (Arcpy), SDK

Рисунок Д.6 – Слайд презентації 6
UML-діаграма активностей кешування картографічних тайлів

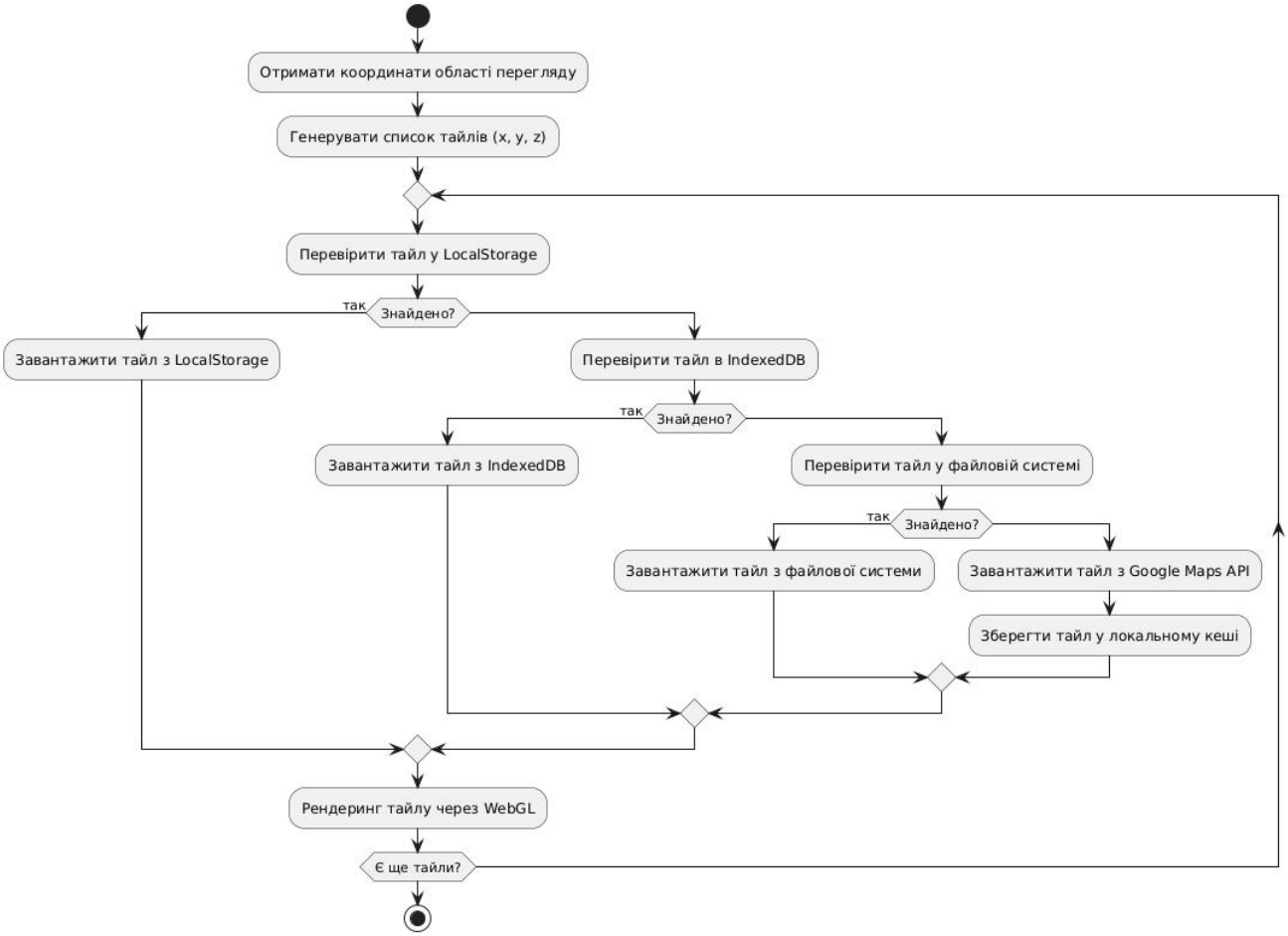

Рисунок Д.7 – Слайд презентації 7
UML діаграма компонентів модулів системи кластеризації маркерів

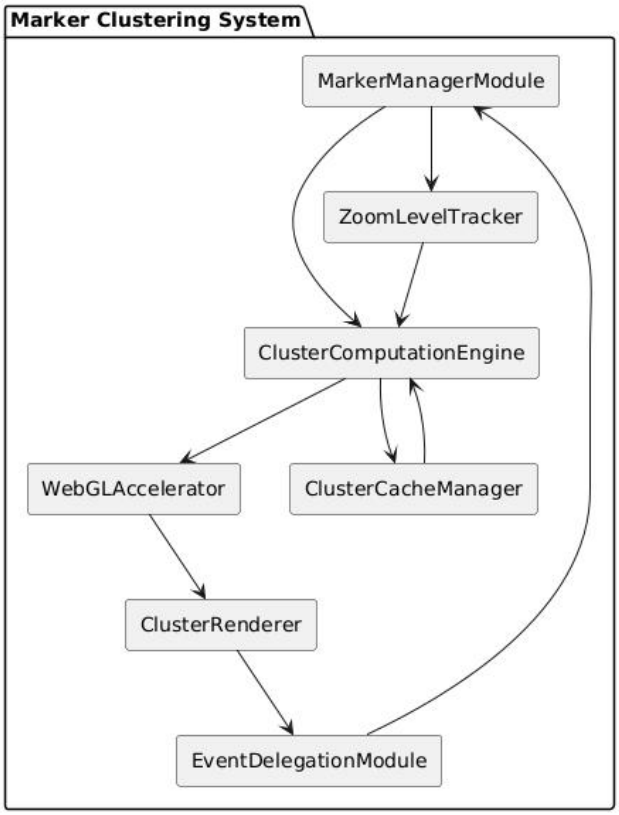

Рисунок Д.8 – Слайд презентації 8
UML діаграма класів модулів системи кластеризації маркерів

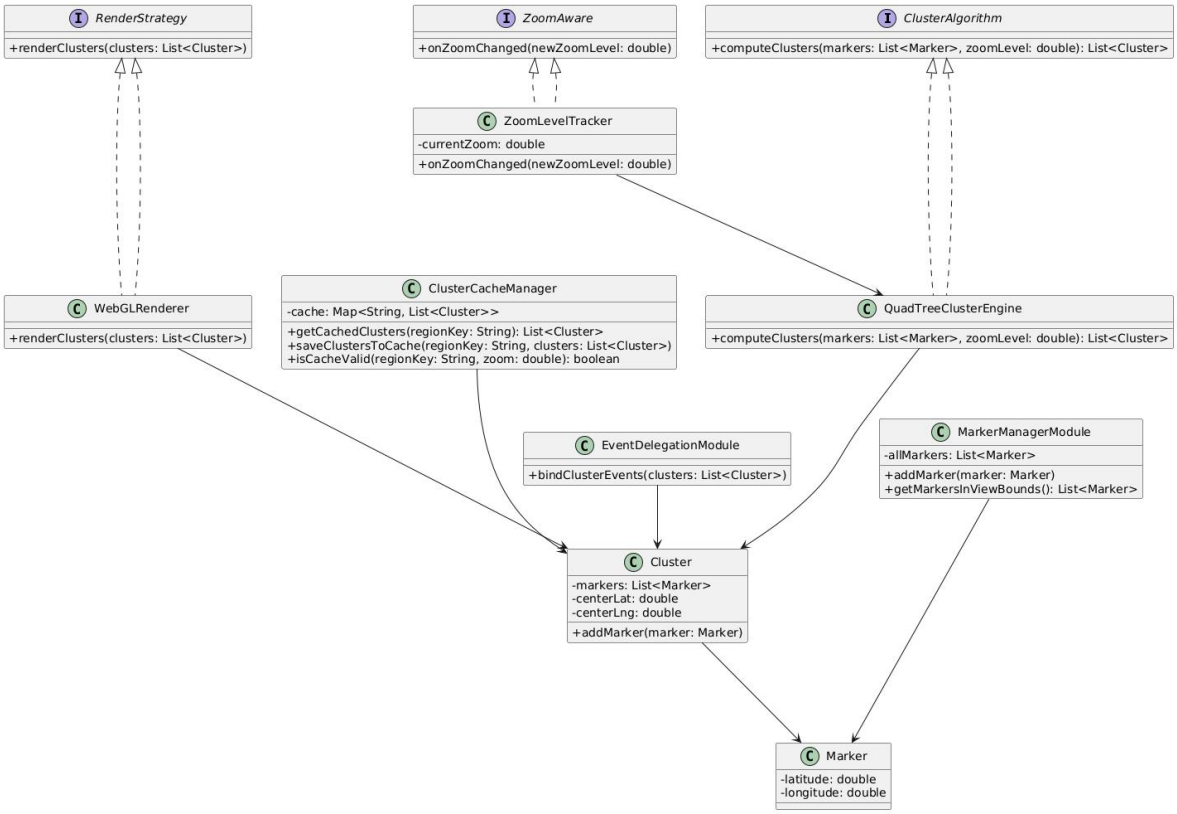

Рисунок Д.9 – Слайд презентації 9

Скріншот тестування кешу для початкових даних, що завантажені з інтерактивної карти м. Львова

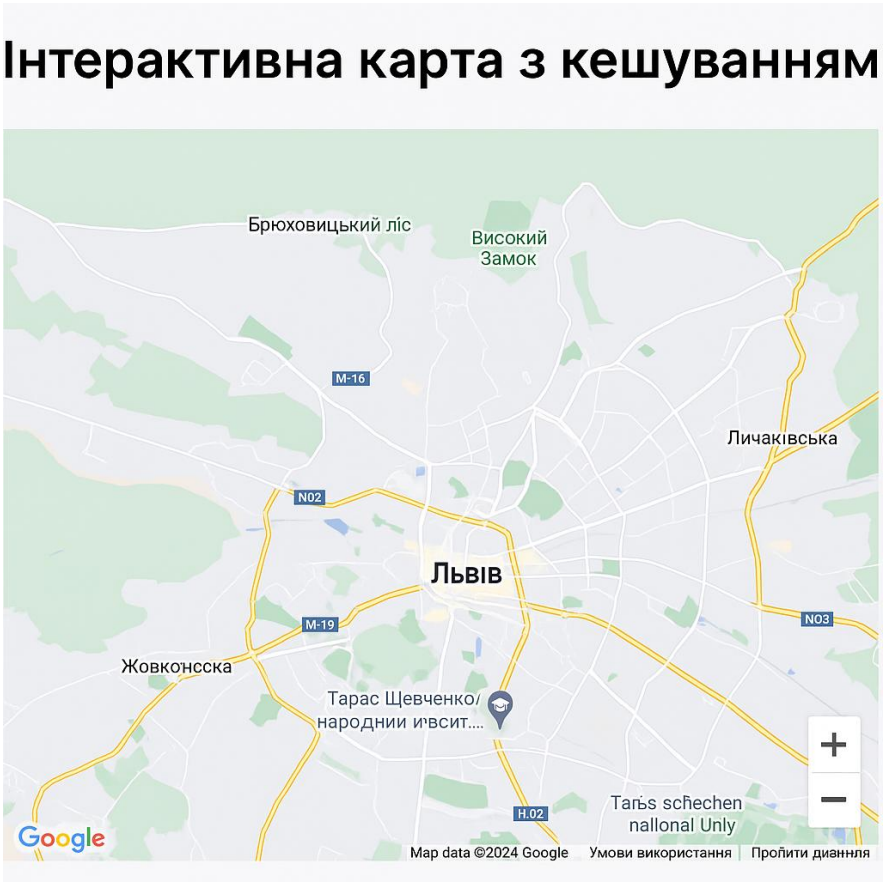

Рисунок Д.10 – Слайд презентації 10

Скріншот результату тестування декількох тайлів у кешу на клієнтській частині з панелі «Application» інструментів розробника (DevTools) веббраузера Google Chrome

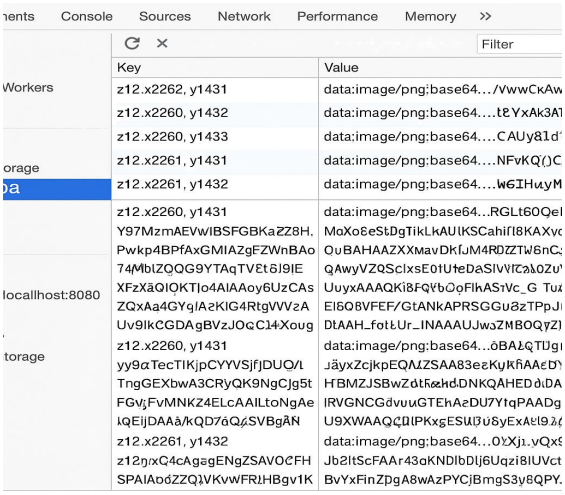

Рисунок Д.11 – Слайд презентації 11

Новизна отриманих результатів:

1. Запропоновано метод кешування даних на основі компоненту тайлів Google Maps API з використанням Java/WebGL, особливість якого полягає у попередньому завантаженні та локальному кешуванні растрових і векторних шарів, що надає можливість підвищити швидкодію системи під час роботи з великими об’ємами даних та зменшити затримку відображення картографічних шарів приблизно на 32–46% за рахунок уникнення повторних звернень до сервера, оскільки оновлені тайли завантажуються майже миттєво з локального сховища.

2. Запропоновано метод кластеризації маркерів, який враховує їх просторову щільність і масштаб відображення зображень на основі особливості близько розташованих точок які автоматично групуються в кластери з використанням WebGL-акселерації для обчислення кластерів на стороні клієнта, що дозволяє зменшити кількість одночасно рендерених об’єктів та підвищити швидкодію візуалізації рендерингу на 24% завдяки меншій кількості відображуваних маркерів та оптимізації обробки подій кластерів.

3. Подальшого розвитку отримав підхід WebGL-рендерингу з попередньою агрегацією даних, у якому, на відміну від існуючих Canvas/SVG-рішень відбуваються обчислення кластерів і спрощення геометрій виконуються на GPU, що дозволяє поєднувати серверну підготовку тайлів із клієнтською обробкою даних внаслідок кластеризації та децимації на рівні тайла) та оптимізувати обсяг передаваних даних внаслідок зниження мережевих затримок та підвищення продуктивності візуалізації великих геопросторових масивів приблизно на 53% з використанням прискоренних обчислень на апаратному рівні WebGL.

Рисунок Д.12 – Слайд презентації 12

Висновки:

У бакалаврській кваліфікаційній роботі:

- виконано аналіз предметної галузі візуалізації великих обсягів просторових даних у геоінформаційних системах;
- проведено порівняльний аналіз програмних засобів що використовують візуалізацію великих обсягів просторових даних у геоінформаційних системах;
- запропоновано метод кешування даних на основі компоненту тайлів Google Maps API;
- запропоновано метод офлайн-інтеграції з картографічними сервісами;
- розроблено функціональні модулі системи кешування просторових даних;
- розроблено програмний код за методом кластеризації маркерів;
- виконано тестування модуля системи кешування просторових даних;
- виконано тестування кешування тайлів у локальному сховищі.