УДК 004.855
DOI: 10.18523/2617-3808.2020.3.88-92

Жук М. А., Проценко В. С.

СУЧАСНІ ПІДХОДИ У ПРОЕКТУВАННІ ВИСОКОНАВАНТАЖЕНИХ МАП ДЛЯ ПОШУКУ ОБ'ЄКТІВ

У роботі описано ключові аспекти, пов'язані з прикладною розробкою високонавантажених вебмап на основі мапи пошуку нерухомості. Під час розроблення системи важливо розуміти ключові вимоги та адресувати їх в архітектурному рішенні. При проектуванні архітектурного рішення було враховано такі ключові аспекти: геокодування, кластеризація, вибір провайдера мапи, фільтрація. Відображення великої кількості об'єктів є одним із ключових завдань. У результаті запропоновано технічне архітектурне рішення з обґрунтуванням використаних елементів системи, зважаючи на можливі адаптації системи та економічну доцільність.

Ключові слова: мапа, пошук об'єктів, геокодування, кластеризація, фільтрація, OpenStreetMap, Mapbox.

Постановка проблеми

Важко уявити сучасне життя без мап. Сьогодні мапи є всюди: у комп'ютері, телефоні, машині, навіть у сучасному годиннику може бути мапа. А чи задумувались ви, наскільки це технологічно складний інструмент? У цій статті ми розглянемо кілька ключових аспектів, пов'язаних із прикладною реалізацією проектів із використанням мап: складність виведення великої кількості об'єктів, складність пошуку та фільтрації, а також витрати. Для цього буде вибрано реалістичний кейс для побудови високонавантаженої мапи зі списком функціональних і нефункціональних вимог. У результаті буде запропоновано архітектурне рішення для реалізації проекту.

Постановка завдання. Створити вебдодаток, на якому користувачі можуть швидко та зручно знайти будь-який об'єкт нерухомості в межах України, щоб купити або орендувати його.

Функціональні вимоги:

- на мапі мають відображатись усі об'єкти нерухомості України, які можливо купити або зняти в оренду;
- пошук об'єкта для покупки/оренди за назвою;
- пошук об'єкта інтересу за назвою;
- автодоповнення при пошуку об'єкта за назвою об'єкта за першими літерами;
- фільтрація об'єктів (ціна, площа тощо).

Нефункціональні вимоги:

- мапа має завантажуватись менш ніж за 3 с;
- оновлення результатів (фільтрація) має відбуватись менш ніж за 2 с;
- підказки автодоповнення мають з'являтись за менш ніж 300 мс;
- система має витримувати 1 млн сеансів на місяць;
- API автодоповнення має витримувати до 1млн запитів на місяць;
- витрати на карту не мають перевищувати 2 тис. доларів у рік.

Аналіз

Першу вебмапу представила компанія Xerox у 1993 р. [4]. Це було одне суцільне зображення. Компанія Google у 2005 р. запропонувала розбити зображення на плитки (tiles), які можна заздалегідь згенерувати [5]. За такого способу при зміщеннях вам лише необхідно завантажити кілька нових плиток. Для кожного рівня наближення мапи генерується свій набір плиток.

Є два типи плиток: растрові й векторні. Різниця між ними така сама, що й між растровими і векторними типами зображень. Із растровою картою важко реалізувати взаємодію [3], тому що з погляду браузера це лише картинка, і вирізнити на ній окремі елементи для взаємодії – достатньо нетривіальна задача. Векторні плитки запропонувала компанія Google у 2010 році [1]. Ідея полягала у тому, щоб зберігати на сервері не картинки, а опис об'єктів на мапі: полігони, лінії, точки. А також опис до об'єктів: назви вулиць, міст тощо. Найпоширеніший формат збереження даних – GeoJSON і Mapbox Vector Tile, який використовує бінарний протокол protobuf.

На ринку є більше ніж десяток *клієнтських бібліотек* [6]. Щоб звузити коло пошуку, можна скористатись такими ресурсами:

- npm – найбільший у світі репозиторій javascript бібліотек;
- google trends – аналітика пошукових запитів у всьому світі;
- github – вебпортал, у якому інтегровано безліч інструментів для розроблення коду, зокрема рейтингову систему;
- npm trends – вебпортал, який збирає статистику завантажень бібліотек.

Проаналізувавши дані, ми обрали бібліотеку Mapbox GL GS, яка має такі переваги: одна з найпопулярніших бібліотек, opensource рішення, широкий функціонал, підтримка векторних і растрових плиток, кластеризація, доступ до API Leaflet, використовує графічне ядро для рендерингу.

Кластеризація та фільтрація. Одночасне відображення 100 тис. точок є затратним щодо ресурсів і дає мало користі для користувача, оскільки орієнтуватися у такій кількості точок буде дуже важко. Кластеризація – це процес групування близько розташованих об'єктів. При цьому можливо або просто відсікати частину точок і збільшувати їх кількість при збільшенні наближення на мапі, або об'єднувати згруповані точки у кластерний елемент і виводити кількість точок, що були згруповані. Є два підходи до кластеризації: клієнтська (у браузері) і серверна.

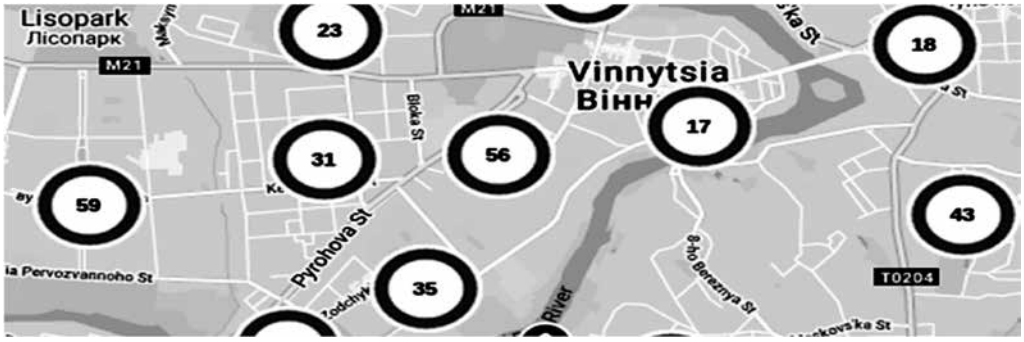

Рис. 1. Кластеризація

Перевага кластеризації на клієнті – це її простота реалізації і гнучкість, проте вона не зменшує кількість даних, які передаються на клієнт. Крім того, сам процес кластеризації потребує ресурсів.

Поєднання фільтрації за серверної кластеризації накладає деякі обмеження. Ми не можемо згенерувати кластери разом із нашими плитками, тому що генерування плиток займає багато часу, а наперед згенерувати кластери не вдасться, оскільки кількість об'єктів у кластері буде різною для різних наборів фільтрів. Якщо генерувати наперед окремі плитки для кожної комбінації фільтрів, то навіть якщо у нас два фільтри по п'ять опцій у кожному – це дасть 25 наборів плиток, і кожен наступний фільтр збільшуватиме цю кількість у геометричній прогресії.

Є кілька підходів, щоб розв'язати цю проблему: не показувати кількість об'єктів у кластері, генерувати кластери «на льоту». Перший підхід, звісно, простіший. При створенні плиток для мапи у місцях із великим накопиченням точок деякі точки просто відсікаються на певних рівнях наближення. Чим ближче користувач наближається, тим більше точок він бачить. Фільтрація у такому випадку забезпечується за рахунок стилізації мапи.

Другий варіант є складнішим у реалізації, але дає кращий досвід нашим користувачам. Для цього ми робитимемо мапу з декількох шарів. Перший шар – це плитки з основною інформацією яку ми бачимо на мапі: вулиці, дороги, будинки. Другий шар – це об'єкти, до яких можуть бути застосовані фільтри. Тож нам не потрібно генерувати «на льоту» всю мапу, а лише шар з об'єктами і кластерами відповідно до вибраних користувачем фільтрів [2]. Кластери потім можуть бути передані як geojson або конвертовані у векторні плитки.

Архітектура

Розглянемо структуру нашого вебсервісу докладніше на компонентній діаграмі. Користувач взаємодіє з системою через browser. У browser, із серверів, які займаються статичним контентом (Static Content) завантажуються необхідні ресурси: html, css, images, js та ін., і зокрема Mapbox GL GS.

Після завантаження Mapbox GL GS має підвантажитись основна підложка мапи з усіма об'єктами, крім наших об'єктів нерухомості. Генеруванням плиток для мапи займатиметься окремий сервіс (Tile Generator). Він братиме сирі

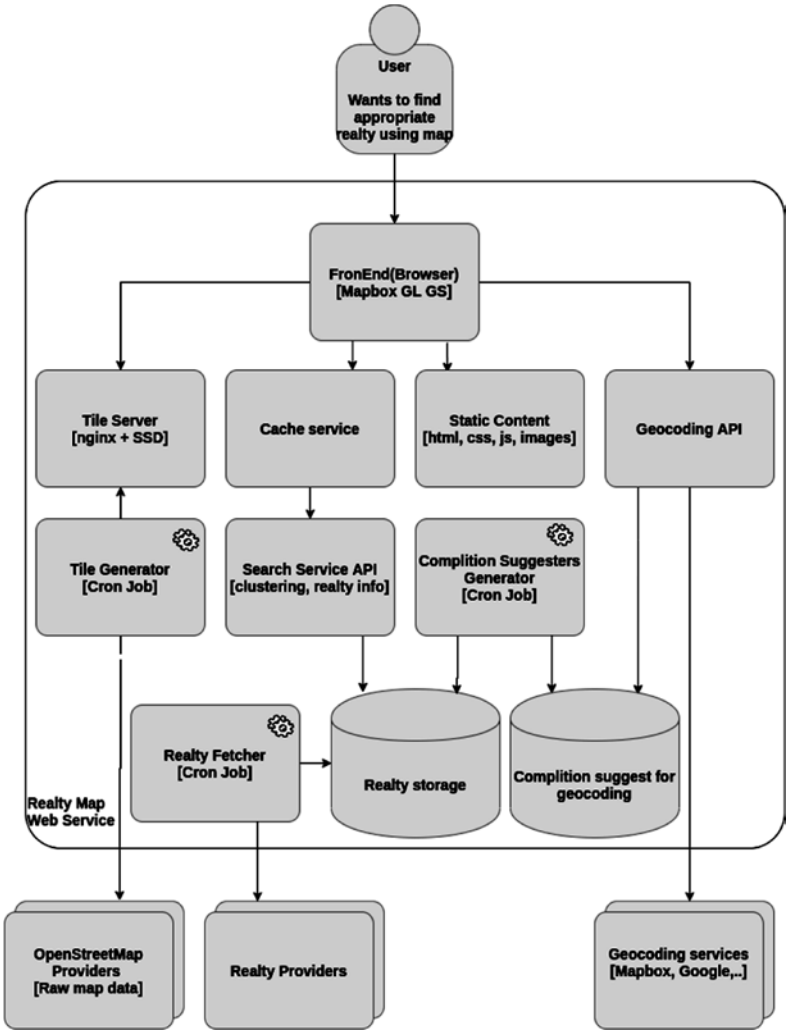

Рис. 2. Компонентна діаграма

дані з OpenStreetMap, генеруватиме плитки та складатиме їх на сервіс, який їх роздаватиме.

Mapnik opensource – продукт, який уміє генерувати як растрові, так і векторні плитки й має хорошу інтеграцію з іншими інструментами.

Крім підложки на мапі мають бути ще об'єкти нерухомості. За їх доставлення, фільтрацію та надання додаткової інформації відповідатиме Search Service. Він може передавати кластеризовані результати пошуку у вигляді GeoJson або векторних плиток. Дані він братиме з бази Realty Storage.

Операції пошуку у БД порівняно ресурсоємні й можуть стати слабким місцем за збільшення навантаження. Те саме можна сказати про обчислення кластерів і генерування векторних плиток на основі GeoJson. Для того щоб забезпечити максимальну продуктивність та стабільність, ми додали Cache Service.

Realty Fetcher – сервіс, який запускатиметься за потреби для того, щоб синхронізувати базу даних провайдера об'єктів нерухомості й нашу базу. Він потрібен для того, щоб не створювати надмірне навантаження на API провайдера об'єктів нерухомості. Також це покращить доступність нашої системи, оскільки наша система буде працездатною, навіть якщо провайдер якийсь час буде недоступний. Крім того, окрема база робить нас незалежними від функціональності API провайдера. Також ми можемо інтегруватись із різними провайдерами даних і виступати агрегатором.

Geocoding API Service зроблений для того, щоб не прив'язуватись до специфіки API провайдерів геокодування. Таким способом ми можемо комбінувати результати від різних провайдерів, змінювати провайдерів залежно від економічної доцільності або взагалі відмовитись від них. Також цей сервіс збагачує дані зі сторонніх сервісів нашими даними про об'єкти нерухомості. За основу можна взяти Elasticsearch, що дає можливість пошуку підказок з урахуванням контексту, а також геолокації, крім того, може виправляти помилки користувача при введенні. Оскільки база підказок зберігається в оперативній пам'яті, це гарантує високу швидкодію.

Ця архітектура не залежить від конкретного cloud провайдера, також може бути self hosted рішенням. Також архітектура легко масштабується під навантаження, що зростає. Це забезпечується за рахунок серверної кластеризації, попереднього генерування контенту, високоефективних серверів роздавання статичного контенту, кешуванню, in-memory баз даних, відсутності прямих залежностей від зовнішніх провайдерів.

Можливість швидко реалізувати першу робочу версію системи – дуже важливий фактор. Наприклад, цю систему можна реалізувати уже в першій ітерації, якщо не робити власний сервер плиток, а використовувати безкоштовний ліміт сервісу mapbox. Також можна для початку

обмежитись одним провайдером нерухомості й відмовитись від синхронізувальної бази. Геокодування може на першій ітерації не враховувати підказки з наших даних. Таким способом ми отримаємо робочу версію системи вже на першій ітерації.

Висновок

Сучасні підходи у проектуванні архітектури зумовлені динамічним розвитком індустрії та бізнесу. Agile методологія витісняє Waterfall. І проектування архітектури не є винятком. Хороша архітектура може швидко адаптуватись, реалізовуватись і спрямована передусім на задоволення бізнес-потреб. Тому проектування починається з виявлення основних функціональних і нефункціональних вимог і в результаті має показати, як саме вимоги будуть задоволені. Можливість швидко реалізувати першу робочу версію системи також має важливе значення. Бажано, щоб спершу витрати були мінімальними, це дасть змогу бізнесу з мінімальними ризиками випробувати нову систему. В хорошій системі витрати мають зростати повільніше, ніж навантаження на неї. У нашій архітектурі це реалізовано за допомогою використання opensource рішень, попереднього генерування даних, кешування, можливості зміни cloud провайдера, відсутності ресурсоємних компонентів.

У подальшому систему можна доповнити часовими лініями (ізохронами) та маршрутами. Адже зазвичай об'єкти нерухомості цікавлять своїм відносним розташуванням до інших об'єктів: час до роботи, школи, дитячого садка. Проте складність таких рішень виходить далеко за межі цієї статті.

«If change is the only constant in the universe, then software change is not only constant but ubiquitous» (Len Bass, «Software Architecture in Practice»).

Список літератури

1. Antoniou V. Tiled Vectors: A Method for Vector Transmission over the Web [Electronic resource] / V. Antoniou, J. Morley, M. M. Haklay // Proceedings of the International Symposium on Web and Wireless Geographical Information Systems, Maynooth, Ireland, 7–8 December 2009. – Springer : Berlin/Heidelberg, Germany, 2009. – Pp. 56–57. – Mode of access: http://link.springer.com/10.1007/978-3-642-10601-9_5.
2. Nebel Paul. Dynamic Server-Side Clustering for Large Datasets [Electronic resource] / Paul Nebel. – 2018. – Mode of access: https://geovation.github.io/dynamic-server-side-geo-clustering.
3. Noskov A. Computer Vision Approaches for Big Geo-Spatial Data: Quality Assessment of Raster Tiled Web Maps for Smart City Solutions [Electronic resource] / A. Noskov. – Bulgarian Cartographic Association: Sofia, Bulgaria, 2018. – Mode of access: http://dx.doi.org/10.5281/zenodo.1346671.
4. Sample J. T. Tile-Based Geospatial Information Systems. Principles and Practices / J. T. Sample. – Springer : Berlin/Heidelberg, Germany, 2010.
5. Stefanakis E. Map Tiles and Cached Map Services [Electronic resource] / E. Stefanakis // GoGeomatics. Magazine of GoGeomatics Canada. – 2015. – Mode of access: http://www2.unb.ca/~{}estef/papers/go_geomatics_stefanakis_december_2015.pdf.
6. Stepnov Eugene. Top Javascript Maps Api And Libraries [Electronic resource] / Eugene Stepnov. – 2016. – Mode of access: https://flatlogic.com/blog/top-javascript-maps-api-and-libraries.

References

Antoniou, V., Morley, J., & Haklay, M. M. (2009). Tiled Vectors: A Method for Vector Transmission over the Web. In *Proceedings of the International Symposium on Web and Wireless Geographical Information Systems, Maynooth, Ireland, 7–8 December 2009*. Springer: Berlin/Heidelberg, Germany. Retrieved from http://link.springer.com/10.1007/978-3-642-10601-9_5.

Nebel, Paul. (2018). Dynamic Server-Side Clustering for Large Datasets. Retrieved from https://geovation.github.io/dynamic-server-side-geo-clustering.

Noskov, A. (2018). *Computer Vision Approaches for Big Geo-Spatial Data: Quality Assessment of Raster Tiled Web Maps for Smart City Solutions*. Bulgarian Cartographic Association: Sofia, Bulgaria, 2018. Retrieved from http://dx.doi.org/10.5281/zenodo.1346671.

Sample, J. T. (2010). *Tile-Based Geospatial Information Systems. Principles and Practices*. Springer: Berlin/Heidelberg, Germany.

Stefanakis, E. (2015). Map Tiles, and Cached Map Services. In *GoGeomatics. Magazine of GoGeomatics Canada*. Retrieved from http://www2.unb.ca/~{}estef/papers/go_geomatics_stefanakis_december_2015.pdf.

Stepnov, Eugene. (2016). *Top javascript maps API and libraries*. Retrieved from https://flatlogic.com/blog/top-javascript-maps-api-and-libraries.

M. Zhuk, V. Protsenko

MODERN APPROACHES IN DESIGNING HIGHLY LOADED MAPS FOR OBJECT SEARCH

It is hard to imagine our life without maps. Maps are everywhere, even modern watches can have a map. Essentially maps are very complicated from the engineering perspective. Displaying thousands of objects, filtering, clustering, and making it possible on low-end devices are tough tasks.

We will investigate modern engineering approaches using a realistic case and provide all key details and motivation behind the architecture. The case will be to develop a web map for searching property all over Ukraine. Key functional requirements: all objects should be on map, filters, text search. Key non-functional requirements: total number of objects, load time, filter time, search speed, amount of users, the budget.

There are a lot of map libraries on the market, our research shows one most suitable for us. The map consists of different layers. The basic layer can be provided as a raster image or in vector format. Clustering is a key feature that makes it possible to display a huge amount of points. Clustering can be performed either in the browser or on the server-side. Server-side clustering is the most scalable solution but it creates additional challenges for filtering that is why we should provide different layers for the base layer and for clusters.

Baseline architecture consists of the Static Content Server, Tile Server, Search Service, Geocoding Service. Tiles for Tile Server generated from OpenStreetMap data. Search Service uses Realty Database to generate clusters and provide additional information. Realty Storage filled by Realty Fetcher Service who takes them from open sources. Geocoding Service combines data from different providers(internal and external) in order to provide a reach user experience.

Modern engineering approaches have distinguishable features: business-oriented, customer-oriented, heavily using open source to meet strict deadlines and budgets, scaleable, adaptable for changes, and fast releases.

Keywords: map, object search, geocoding, clustering, filtering, OpenStreetMap, Mapbox.

Матеріал надійшов 31.05.2020