

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Комп'ютерних наук і технологій

(повне найменування факультету)

Комп'ютерні системи та мережі

(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

бакалавра

(ступінь вищої освіти)

на тему **РОЗРОБКА КОМП'ЮТЕРНОЇ СИСТЕМИ З МОНІТОРИНГУ
ДОРОЖНЬОГО ТРАФІКУ НА МІЖНАРОДНИХ, НАЦІОНАЛЬНИХ ТА
РЕГІОНАЛЬНИХ ДОРОГАХ УКРАЇНИ**

(назва теми)

Виконав: студент 4 курсу, групи КНТ-519

Спеціальності 123 «Комп'ютерна інженерія»

(код і найменування спеціальності)

Освітня програма (спеціалізація)

«Комп'ютерна інженерія»

Герасін Ю.В.

(Прізвище та ініціали)

Керівник Скрупський С.Ю.

(Прізвище та ініціали)

Рецензент Куляба-Харитонova Т.І.

(Прізвище та ініціали)

2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук та технологій
Кафедра "Комп'ютерні системи та мережі"
Ступінь вищої освіти бакалаврський
Спеціальність 123 "Комп'ютерна інженерія"
(код і найменування)
Освітня програма (спеціалізація) Комп'ютерна інженерія
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри Кудерметов Р.К.

« » 20 року

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

Герасін Юрій Володимирович

(Прізвище, ім'я, по батькові)

1.Тема проєкту (роботи) Розробка комп'ютерної системи з моніторингу дорожнього трафіку на міжнародних, національних та регіональних дорогах України

Керівник проєкту (роботи) к. т. н., доцент, Скрупський Степан Юрійович
(науковий ступінь, вчене звання, прізвище, ім'я, по батькові)

Затверджені наказом закладу вищої освіти від «24» березня 2023 року №71

2. Строк подання студентом проєкту (роботи) 31 травня 2023 року

3.Вихідні дані до проєкту (роботи) Технології створення клієнт-серверних додатків React, Angular, Vue.js, PHP, Java Servlets, Node.js, MySQL, PostgreSQL, MongoDB. Необхідність обрання найдоцільнішої для створення ефективної, надійної та простої розподіленої системи для зручного доступу до відеотрансляцій

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1)Аналіз сучасних технологій взаємодії з базою даних;

2) Розробка клієнтської та серверної частин системи;

3) Тестування розробленої системи.

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

Плакат 1 - Схема роботи системи, алгоритм роботи клієнтського додатку, UML-діаграма додавання маркерів на мапу;

Плакат 2,3 - Результати тестування системи

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-3	Скрупський С.Ю., к. т. н., доцент		
Нормоконтроль	Щербак Н.В., ст. викл.		

7. Дата видачі завдання «01» березня 2023 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Огляд наявних рішень комп'ютерного забезпечення платформ для моніторингу дорожнього трафіку, перегляд отриманих результатів, вибір альтернативи	01.03.2023 р.	
2	Розробка структури системи	15.03.2023 р.	
3	Розробка алгоритму роботи системи	01.04.2023 р.	
4	Тестування роботи системи для перегляду відеотрансляції	01.05.2023 р.	
5	Оформлення отриманих результатів у ПЗ	10.05.2023 р.	
6	Оформлення допоміжного матеріалу	17.05.2023 р.	

Студент(ка)


(підпис)

Юрій ГЕРАСІН
(Ім'я ПРІЗВИЩЕ)

Керівник проєкту (роботи)


(підпис)

Степан СКРУПСЬКИЙ
(Ім'я ПРІЗВИЩЕ)

РЕФЕРАТ

ПЗ: 51 с., 7 рис., 13 джерел

EXPRESS, JAVASCRIPT, JSON, MONGODB, NODE.JS, REACT, TYPESCRIPT, БАЗА ДАНИХ, ДОДАТОК, КЛІЄНТ, РОЗПОДІЛЕНА СИСТЕМА, СЕРВЕР

Об'єкт дослідження – розподілена система для моніторингу дорожнього трафіку.

Предмет дослідження – робота з веб-інтерфейсом – доступ до відеотрансляції дорожнього трафіку у режимі реального часу за допомогою камер відеоспостереження CCTV.

Метою роботи є створення розподіленої системи для перегляду через веб-браузер потокового відео дорожнього руху, що транслюються IP-камерами у глобальну мережу Інтернет.

Робота складається з трьох розділів.

Перший розділ присвячено аналізу технологій, які застосовуються при побудові системи даної роботи, а саме серверної та клієнтської частин додатку, проведенню їх зіставлення, в результаті чого зроблено висновки.

У другому розділі визначено вимоги до системи, описано процес побудови серверного та клієнтського додатків, застосовуючи обрані технології з прикладами коду, описано методи та правила наповнення бази даних додатку.

Третій розділ присвячено тестуванню системи з прикладом сценарію використання.

ЗМІСТ

Перелік скорочень та умовні позначки	6
Вступ	8
1 Аналіз сучасних технологій моніторингу дорожнього трафіку	10
1.1 Актуальність	10
1.2 Аналіз технологій для клієнтської частини системи	11
1.3 Аналіз технологій для серверної частини	15
1.4 Огляд існуючих рішень управління базою даних	18
1.5 Висновки аналізу технологій	21
2 Розробка клієнтської та серверної частини системи	23
2.1 Визначення функціональних вимог до системи	23
2.2 Визначення нефункціональних вимог до системи	24
2.3 Опис структури схеми	26
2.4 Розробка модулів клієнтської частини	29
2.5 Розробка модулів серверної частини	35
2.6 Розробка бази даних додатку	40
3 Тестування розробленої системи	42
Висновки	45
Перелік джерел посилання	46
Додаток А	48

Графічний матеріал:

Плакат 1 - Схема роботи системи, алгоритм роботи клієнтського додатку, UML-діаграма додавання маркерів на мапу;

Плакат 2,3 - Результати тестування системи.

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНІ ПОЗНАКИ

СКБД	—	система керування базами даних
HTML	—	мова розмітки гіпертексту, HyperText Markup Language,
CSS	—	каскадні таблиці стилів, Cascading Style Sheets
UI	—	інтерфейс користувача, User Interface
JSX	—	розширення синтаксису JavaScript, JavaScript Syntax Extension
DOM	—	об'єктна модель документа, Document Object Model
SPA	—	односторінковий застосунок, Single-page application
PEAR	—	сховище розширень і застосунків PHP, PHP Extension and Application Repository
ASCII	—	Американський стандартний код для інформаційного обміну, American Standard Code for Information Interchange
SQL	—	мова структурованих запитів, Structured query language
XSS	—	міжсайтовий скриптинг, Cross Site Scripting
XML	—	розширювана мова розмітки, Extensible Markup Language
ЦП	—	центральний процесор
ПЗ	—	програмне забезпечення
JSON	—	запис об'єктів JavaScript, JavaScript Object Notation
SSL	—	протокол безпечних з'єднань, Secure Sockets Layer
CCTV	—	замкнуте телебачення, Closed Circuit Television
API	—	прикладний програмний інтерфейс, Application Programming Interface
IP	—	інтернет протокол, Internet Protocol
UML	—	уніфікована мова моделювання, Unified Modeling Language

ЦОД	—	центр опрацювання даних
URL	—	уніфікований локатор ресурсів, Uniform Resource Locator
URI	—	уніфікований ідентифікатор ресурсів, Uniform Resource Identifier
BLOB	—	бінарний великий об'єкт, Binary Large Object
MSE	—	розширення медіаджерела, Media Source Extensions
REPL	—	цикл читання-обчислення-друку, Read-Eval-Print Loop
ЛІТ-компіляція	—	компіляція “на льоту”, Just-in-time compilation

ВСТУП

Обґрунтування актуальності роботи. Проаналізувавши поточну проблематику моніторингу дорожнього трафіку на території України, можна визначити необхідність у створенні загальнодоступного програмного продукту для відстежування і контролю дорожнього руху та подальшого прийняття ефективних рішень керування автомобільним транспортним засобом.

В останнє десятиріччя активно впроваджуються комп'ютерно-орієнтовані системи, що реалізують контроль за транспортною інфраструктурою і дорожнім трафіком шляхом використання різноманітних архітектур, які застосовують камери відеоспостереження, п'єзоелектричні датчики, пневматичні дорожні трубки, магнітні петлі. Зважаючи на це, тема даної дипломної роботи є актуальною.

Об'єкт бакалаврського дослідження – комп'ютерна система з моніторингу дорожнього трафіку.

Основна мета бакалаврського дослідження сформульована наступним чином: розробка оптимальних процесів моніторингу дорожнього трафіку за допомогою програмного продукту.

Реалізацію даного завдання покладено на відповідні технології.

Вхідними даними для розробки є загальні відомості про мережу автомобільних доріг країни та результати роботи фотовідеофіксації з відеокамер на автомобільних шляхах країни.

У зв'язку з повномасштабною війною Російської Федерації проти України, з міркувань безпеки відкриті трансляції з відеокамер на всій території України були вимкнені, тому країною для реалізації дипломної роботи була обрана Республіка Польща. Головними причинами такого вибору є подібність структури автомобільних доріг, швидкісних обмежень та рельєфу обидвох країн.

Наявні на сьогодні системи моніторингу дорожнього руху (Road Traffic Monitoring Systems) поділяють на наступні види за використанням [1]:

- виявлення подій, таких як ДТП і перевищення швидкості;
- регулювання дорожнього руху, щоб уникнути заторів.

Розроблювана комп'ютерна система представляє собою гібрид з цих двох видів з акцентом уваги на другий вид.

Предмет бакалаврського дослідження – засоби та методи спостереження і аналізу дорожнього трафіку в умовах використання інтернет-технологій.

1 АНАЛІЗ СУЧАСНИХ ТЕХНОЛОГІЙ МОНІТОРИНГУ ДОРОЖНЬОГО ТРАФІКУ

1.1 Актуальність

Провівши ґрунтовний порівняльний аналіз технологій (методів) моніторингу дорожнього трафіку, можна констатувати наступний факт: створення подібних систем спостереження шляхом застосування різноманітних датчиків та інших спеціалізованих засобів виявлення активності (пневматичні дорожні трубки, магнітні петлі і т.д.) є проблематичним з огляду на їхню доступність та ринкову ціну.

Оптимальним та загальнодоступним варіантом для отримання вхідних даних дорожнього руху є застосування ІР-камер відеоспостереження, які знімають відео та транслюють відеопотік у цифровому форматі з використанням мережевого протоколу, що забезпечує маршрутизацію пакетів. Тобто це означає, що існує можливість отримати доступ до необхідного контенту за допомогою обчислювальної машини із підключенням до глобальної мережі Інтернет.

Безпосередньо отримання результатів роботи відбувається через веб-інтерфейс комп'ютерної системи у браузері.

Розроблений веб-додаток представляє собою розподілений застосунок, що складається з двох частин – клієнтської та серверної. Обробка, пошук та оцінка використовуваних даних відбувається сервером із застосуванням системи керування базами даних (СКБД), яка надає контроль доступу до даних та управління ними.

1.2 Аналіз технологій для клієнтської частини системи

Для аналізу технологій, використаних у клієнтській частині веб-додатку, необхідно враховувати множину факторів вибору. Серед них найсуттєвішими є призначення, цільова аудиторія, шаблон побудови, підхід до проєктування веб-дизайну та продуктивність.

Традиційними засобами розробки front-end частини клієнт-серверного додатку є мова розмітки веб-сторінок HTML, CSS – для опису зовнішнього вигляду сторінок, і JavaScript – створення сценаріїв веб-сторінок.

Задля пришвидшення та спрощення розробки проєкту існує можливість використовувати фреймворки CSS та JavaScript.

Серед фреймворків CSS розглянемо наступні варіанти: Bootstrap, Foundation, Tailwind CSS.

У випадку із JavaScript проаналізуємо вибірку з React, Angular і Vue.js.

1.2.1 Bootstrap

Bootstrap – це безкоштовний фреймворк CSS із відкритим вихідним кодом, спрямований на адаптивну інтерфейсну веб-розробку, орієнтовану на мобільні пристрої. Він містить HTML, CSS і (опціонально) шаблони дизайну на основі JavaScript для типографіки, форм, кнопок, навігації та інших компонентів інтерфейсу.

За даними W3Techs, Bootstrap використовують 19,2% усіх веб-сайтів в глобальній мережі Інтернет.

Порівняно з іншими фреймворками CSS, найбільшою перевагою Bootstrap є величезний вибір додаткових шаблонів, тем і шаблонного коду. Це дозволяє легко та швидко створити складний веб-сайт без професійних навичок у сфері дизайну [2].

Головним потенційним недоліком використання Bootstrap є те, що він може обмежити ваш контроль над макетом і дизайном вашого веб-сайту: не мати

повного контролю над тим, як компоненти використовуються чи впорядковуються у вашому макеті.

1.2.2 Foundation

Foundation – це фреймворк CSS, призначений для створення респонсивного дизайну веб-сайтів. Він містить адаптивну сітку, що повністю налаштовується, плагіни JavaScript і повну підтримку спеціальних можливостей.

Характерною відмінністю від аналогів є його модульна структура. Він використовує такі надійні технології, як jQuery, HTML5 Boilerplate, Sass і Normalizr. Додатково поставляється з набором компонентів і плагінів, сумісних з більшістю браузерів і пристроїв. Крім того, Foundation може працювати практично з будь-якою серверною технологією.

Недоліками цього набору інструментів є його великий розмір, що може вплинути на час завантаження веб-додатку та вичерпність документації.

1.2.3 Tailwind CSS

Аналогічно до вищенаведених прикладів Tailwind CSS є відкритим фреймворком мови CSS для швидкого створення користувацьких інтерфейсів.

Головною особливістю функціональності Tailwind CSS є його робота на низькому рівні, що, у свою чергу, надає усі будівельні блоки, необхідні для створення індивідуальних дизайнів. Окрім цього, існує можливість зв'язати Tailwind CSS з будь-якою платформою JavaScript.

Серед нюансів, які треба враховувати при застосуванні Tailwind CSS, є те, що Tailwind працює інакше, ніж більшість фреймворків CSS, тобто Tailwind безпосередньо включає правила стилю у HTML-файли, порушуючи основний принцип програмної інженерії – принцип “поділу відповідальності”.

1.2.4 React

React – це фреймворк і бібліотека JavaScript із відкритим кодом для створення динамічних та інтерактивних веб-додатків. Характерним для нього є використання колекції багаторазових фрагментів коду JavaScript (компонентів), які надалі застосовуються для побудови інтерфейсу користувача (UI).

До унікальних рис React слід віднести:

- JSX (розширення синтаксису JavaScript);
- віртуальний DOM;
- одностороння прив'язка даних;
- умовні розгалуження. JSX дозволяє записати в коді умовні оператори;
- компоненти.

JSX (розширення синтаксису JavaScript). JSX – це комбінація HTML і JavaScript. Ви можете вставляти об'єкти JavaScript в елементи HTML (подано на лістингу 1.1). JSX не підтримується браузером, тому компілятор Babel транскompілює JSX-код у код JavaScript.

Лістинг 1.1 – Фрагмент JSX-коду

```
function NumberDescriber(props) {
  let description;
  if (props.number % 2 == 0) {
    description = <strong>even</strong>;
  } else {
    description = <i>odd</i>;
  }
  return <div>{props.number} is {description} number</div>;
}
```

Віртуальний DOM. React використовує віртуальний DOM, який є точною копією реального DOM. Щоразу, коли у веб-програмі відбувається зміна, спочатку оновлюється весь віртуальний DOM і виявляється різниця між реальним і віртуальним DOM. Як результат, DOM оновлює лише ту частину, яка нещодавно змінювалася, і все залишається без змін.

Одностороння прив'язка даних. Дані в React направлені лише в одному напрямку, тобто передаються зверху вниз, від батьківських до дочірніх компонентів. Властивості в дочірньому компоненті не можуть повертати дані своєму батьківському компоненту, але можуть мати зв'язок із батьківськими компонентами для зміни станів відповідно до наданих вхідних даних.

Умовні розгалуження. JSX дозволяє записати в коді умовні оператори. Дані в браузері відображаються відповідно до умов, наданих усередині JSX, дивитися на лістингу 1.2.

Лістинг 1.2 – Використання умовних операторів у розширенні JSX

```
const speed = 75;
if (speed >= 50)
{
    <p>Speed limit is exceeded</p>;
}
else
{
    <p> {speed} km/h is normal speed </p>;
}
```

Компоненти. React ділить веб-сторінку на кілька компонентів. Кожен компонент є частиною дизайну інтерфейсу користувача (UI-дизайну), який має власну логіку та дизайн. Таким чином, логіка компонентів, написана на JavaScript, полегшує та пришвидшує роботу, і може використовуватися повторно.

1.2.5 Angular

Angular – це платформа та фреймворк для створення односторінкових клієнтських додатків (SPA) за допомогою HTML і TypeScript. Angular написаний на TypeScript. Він реалізує основні та додаткові функції як набір бібліотек TypeScript, які імпортуються в програми.

Платформа Angular містить:

- компонентну основу для створення масштабованих веб-додатків;
- набір добре інтегрованих бібліотек, які охоплюють широкий спектр функцій, включаючи маршрутизацію, керування формами, зв'язок клієнт-сервер тощо;
- набір інструментів розробника, які допомагають розробляти, запускати, тестувати й оновлювати код.

Процес розробки за допомогою Angular має власні переваги, а саме: добра документованість, підтримка Google, висока продуктивність, малий час виконання, створення шаблонів.

1.2.6 Vue.js

Vue – це фреймворк JavaScript для створення інтерфейсів користувача. Він створений на основі стандартних HTML, CSS і JavaScript і забезпечує

декларативну та компонентну модель програмування, яка допомагає вам ефективно розробляти інтерфейси користувача, прості чи складні.

Vue має 2 основні особливості:

- декларативне відображення: розширює стандартний HTML синтаксисом шаблону, який дозволяє декларативно описувати вивід HTML на основі стану JavaScript;

- реактивність: автоматично відстежує зміни стану JavaScript і ефективно оновлює DOM, коли відбуваються зміни.

Подібно до React, Vue.js оперує поняттям віртуального DOM.

З Angular Vue.js має багато схожостей. Директиви `v-if`, `v-for` – це аналоги `ngIf` та `ngFor` в Angular. Обидва фреймворки мають інтерфейс командного рядка для інсталяції проекту та його створення. VueJS використовує Vue CLI, у той час як Angular використовує Angular CLI. Обидва пропонують двостороннє зв'язування даних, рендеринг на сервері.

1.3 Аналіз технологій для серверної частини

Серверна частина додатку відповідає за безпосередню взаємодію користувача з функціоналом веб-додатку та його даними. Користувач не має права доступу до зміни серверної частини.

У контексті сучасних тенденцій розвитку цифрових та інтернет-технологій важливо забезпечити зручне, швидке та надійне обслуговування запитів клієнта.

Відповідно до вищеназваних вимог, проведемо аналіз серверних мов програмування та технологій: PHP, Java Servlets та Node.js.

1.3.1 PHP

PHP – це мова сценаріїв загального призначення, яка широко використовується як серверна мова для створення динамічних веб-сторінок.

Згідно з даними W3Techs, станом на сьогодні PHP використовується на 77,5% усіх веб-сайтів із відомою серверною мовою програмування. Найпопулярнішими версіями є PHP 7.4 (50,3%), PHP 7.3 (24,7%) і PHP 5.6 (12,8%), а всі інші версії разом складають 12,2%.

У мові PHP не існує обмежень на виведенням HTML або інших текстових файлів – можна створити будь-який формат документа. PHP має вбудовану підтримку для створення файлів PDF, зображень GIF, JPG і PNG, а також файлів формату Flash Video [3].

Однією з найважливіших особливостей PHP є широка підтримка баз даних. PHP підтримує всі основні бази даних (включаючи MySQL, PostgreSQL, Oracle, Sybase) [3].

Крім цього, PHP надає бібліотеку PHP-коду для виконання типових завдань, таких як абстракція бази даних, обробка помилок тощо, за допомогою PHP Extension and Application Repository (PEAR). PEAR – це структура та система розповсюдження багаторазових компонентів PHP [3].

Проте варто зауважити суттєвий недолік при імплементації проєктів на цій мові програмування, а саме – захищеність користувацьких даних. Протягом багатьох років низька безпека PHP була предметом критики. Оскільки PHP використовує концепцію відкритого вихідного коду, хакери можуть легко знайти та проаналізувати попередньо написаний код і текстовий файл ASCII. Якщо додаткові заходи захисту відсутні, зловмисники можуть проводити такі атаки, як: SQL ін'єкція, злом і захоплення сесії користувача, відкриття вихідного коду, XSS (Cross Site Scripting – “міжсайтовий скриптинг”).

1.3.2 Java Servlets

Сервлет – це спеціальний інтерфейс у мові програмування Java, який потрібен для обміну інформацією між серверною і клієнтською частиною програми. Сервлети перебувають під контролем іншої програми Java, яка називається Servlet Container. Сервлет розміщується на сервері, приймає запити від клієнта та надсилає на них відповіді.

Наведемо кілька переваг використання Java-сервлетів для реалізації функціональних можливостей веб-сервера:

- ефективний виклик сервлета. Після завантаження сервлета він залишається в пам'яті сервера як єдиний екземпляр об'єкта. Після цього сервер викликає сервлет для обробки запиту за допомогою простого виклику легкого методу. Кілька одночасних запитів обробляються окремими потоками, тому сервлети мають високу масштабованість [4];

- сервлети підтримують безпечні практики програмування на кількох рівнях. Оскільки вони написані мовою Java, сервлети успадковують надійну безпеку типів мови Java. Автоматичне збирання сміття в Java та відсутність покажчиків означають, що сервлети загалом захищені від проблем керування пам'яттю, таких як висячі покажчики, недійсні посилання на покажчики та витoki пам'яті [4];

- гнучкість. Сервлети також досить гнучкі у тому, як вони створюють вміст. Вони можуть генерувати простий вміст за допомогою операторів `out.println()` або вони можуть генерувати складний набір сторінок за допомогою механізму шаблонів. Для генерації HTML-сторінки існує 2 способи: розглядаючи її як набір об'єктів Java або виконуючи трансформацію XML-HTML.

1.3.3 Node.js

Node.js являє собою асинхронне середовище виконання JavaScript, призначення якого – створення масштабованих мережевих додатків.

На високому рівні Node.js поєднує механізм Google V8 JavaScript, однопотоковий неблокуючий цикл подій і низькорівневий API введення-виведення [5].

Відмінністю Node.js від аналогів є спосіб створення веб-серверів. У традиційному підході для кожного вхідного запиту або з'єднання сервер породжує новий потік виконання або навіть розгалужує новий процес для обробки запиту та надсилання відповіді.

Node.js імплементує інший підхід. Він запускає однопотоковий цикл подій, зареєстрований у системі для обробки з'єднань, і кожне нове з'єднання викликає запуск функції зворотного виклику JavaScript.

Функція зворотного виклику може обробляти запити з неблокуючими викликами вводу-виводу та, якщо необхідно, може породжувати потоки з пулу для виконання блокувальних або ресурсомістких операцій, а також для балансування навантаження на ядра ЦП.

Перевагою є той факт, що масштабування за допомогою функцій зворотного виклику потребує менше пам'яті для обробки більшої кількості з'єднань, ніж більшість конкурентоспроможних архітектур, які масштабуються за допомогою потоків, включаючи HTTP-сервер Apache, різні сервери додатків Java, IIS і ASP.NET і Ruby on Rails [5].

Значна частина утиліт Node надходить від його великої бібліотеки пакетів, доступ до якої здійснюється за допомогою команди `npm`. NPM, менеджер пакетів Node, є частиною стандартної інсталяції Node.js.

Реєстр NPM містить понад 1,3 мільйона пакетів безкоштовного коду (проектів) Node.js, що робить його найбільшим реєстром програмного забезпечення у світі.

1.4 Огляд існуючих рішень управління базою даних

Задачі управління та редагування вмісту бази даних можна виконати за допомогою багатьох технологій. Для поставленого завдання в межах бакалаврської дипломної роботи розглянемо 3 популярні СКБД, які найчастіше використовують у розробці додатків з архітектурою “клієнт-сервер”, а саме: MySQL, PostgreSQL, MongoDB.

1.4.1 MySQL

Одним з найпоширеніших рішень в оперуванні великими масивами інформації є MySQL – вільна система керування реляційними базами даних SQL з архітектурою “клієнт-сервер”. MySQL містить у своєму складі сервер SQL, клієнтські програми для доступу до сервера, інструменти адміністрування та програмний інтерфейс для написання власних програм [6].

Головним чинником використання цього програмного продукту в комерційній розробці ПЗ є забезпечення швидкого, багатопоточного, багатокористувацького і надійного сервера бази даних SQL, який призначений як для застосування в критично важливих виробничих систем із великим навантаженням, так і в програмному забезпеченні для масового розгортання [7].

Проте до істотних недоліків використання цього інструменту відносять такі незручності [8]:

- проблеми стабільності роботи. MySQL важко працювати з такими процедурами, як аудит, посилання та керування транзакціями;
- погане масштабування продуктивності. Цій системі важко виконувати кілька операцій одночасно;
- відтермінування майбутніх вдосконалень. Відтоді, як Oracle перейняла MySQL, прогрес зараз затримується, оскільки за останні сім-вісім років вийшов лише один великий випуск;
- залежність від плагінів. Порівняно з конкурентами йому не вистачає стандартної функціональності.

1.4.2 PostgreSQL

Аналогічно до MySQL, PostgreSQL представляє собою вільну об’єктно-реляційну СКБД, що базується на мові SQL, і станом на сьогодні широко використовується у веб-розробці. Про довіру до цієї технології з боку великих технічних компаній може свідчити той факт, що у 2010 році компанія Apple замінила MySQL на PostgreSQL як вбудовану базу даних у випуску Mac OS X Lion.

Серед характерних особливостей цього програмного рішення слід виділити наступне:

- підтримка великої кількості типів запису інформації. Окрім стандартного набору значень, PostgreSQL підтримує бітові рядки та мережеві адреси, масиви даних, у тому числі багатовимірні, композитні типи, формати обміну даними (XML, JSON), можливість роботи з NoSQL-базами;

- можливість роботи з великими обсягами даних. Максимальний розмір поля – 1 Гбайт, максимальна кількість полів у записі 250-1600 (залежно від типів полів), максимальний розмір запису – 1.6 Тбайт, максимальний розмір таблиці – 32 Тбайт;

- вбудована підтримка з'єднань SSL для шифрування зв'язків клієнт/сервер з метою підвищення безпеки. Для цього потрібно, щоб OpenSSL було встановлено як на клієнтській, так і на серверній системах, і щоб підтримка PostgreSQL була включена під час збирання даних.

Варто додати також той факт, що PostgreSQL неефективно використовувати, якщо питання швидкодії СКБД є критичним моментом, оскільки при простих операціях читання даних PostgreSQL може значно уповільнити сервер, виконання цієї операції відбувається довше, ніж в інших конкурентах, таких як MySQL [9];

Крім цього, проблематичним є вибір хостингу: інколи досить важко знайти хостинг, який підтримує цю базу даних.

1.4.3 MongoDB

MongoDB – це документно-орієнтована база даних NoSQL, яка використовується для зберігання великих обсягів даних. Замість використання таблиць і рядків, як у традиційних реляційних базах даних, MongoDB використовує колекції та документи. Документи складаються з пар ключ-значення, які є основною одиницею даних у MongoDB.

З-поміж традиційних об'єктно-реляційних СКБД MongoDB виділяється наступним:

- документо-орієнтована система: MongoDB зберігає основну тему в мінімальній кількості документів, а не розбиваючи її на кілька реляційних структур, як це робить об'єктно-реляційна система керування базами даних;
- індексування: без індексування база даних мала б сканувати кожен документ колекції, щоб вибрати ті, які відповідають запиту, що було б неефективним. Для ефективного пошуку індексування є обов'язковим, і MongoDB використовує його для обробки величезних обсягів даних за дуже короткий час;
- масштабованість: MongoDB масштабується горизонтально за допомогою шардингу (розподілу даних між різними серверами). Дані розділяються на блоки даних за допомогою ключа фрагмента, і ці блоки даних рівномірно розподіляються між шардами, які знаходяться на багатьох фізичних серверах;
- реплікація та висока доступність: MongoDB підвищує доступність даних за допомогою кількох копій даних на різних серверах. Забезпечуючи резервування, він захищає базу даних від апаратних збоїв;
- агрегація: операції агрегації обробляють записи даних і повертають обчислені результати. Це схоже на пропозицію GROUPBY у SQL. Кілька виразів агрегації: сума, середнє, мінімальне, максимальне тощо.

Окремо слід відзначити можливість використання хмарних обчислень і зберігання. Хмарне сховище є ефективним рішенням, але воно потребує легкого розподілу даних між декількома серверами для масштабування. Використання доступного апаратного забезпечення на місці для тестування, а потім для виробництва в хмарі – це призначення баз даних NoSQL.

1.5 Висновки аналізу технологій

Для прийняття остаточного рішення щодо вибору технологій у проєкті необхідно взяти до уваги специфіку функціоналу веб-застосунку – агрегатору камер відеоспостережень (CCTV cameras) на дорогах Республіки Польща.

Динамічна зміна контенту веб-сторінку відбувається шляхом взаємодії користувача з веб-інтерфейсом браузера: вибір локації на інтерактивній мапі, переключення між відеокамерами, навігації на інших розділах розроблюваної системи. Ці задачі потребують від веб-додатку високої швидкодії, гнучкості та доступності на різних платформах. Даним вимогам відповідає концепція SPA – Single Page Application – реалізація веб-програми, яка завантажує лише один веб-документ, а потім оновлює основний вміст цього окремого документа за допомогою API JavaScript. Це дозволить переключатися між сторінками без їхнього перезавантаження.

Щодо вибору стеку технологій:

- клієнтська частина – зв'язка фреймворку React для JavaScript разом з Tailwind CSS. React забезпечує динаміку та інтерактив у заповненні вмісту веб-документу, а Tailwind – можливість побудови індивідуального дизайну;
- серверна частина – платформа Node.js. Невибagliвість до апаратних ресурсів дозволяє запускати розроблений застосунок на слабких комп'ютерах та смартфонах. Варто також відмітити наявність NPM, менеджера пакетів Node. Цей реєстр дозволяє встановлювати сторонні Node-проекти, зокрема Express.js – серверного фреймворку, який підтримує кешування і уможливорює швидше завантаження веб-сторінок;
- управління базою даних – вибір зроблено на користь MongoDB. Суттєвими перевагами серед конкурентів є масштабованість і властивість цієї системи зберігати дані у JSON-форматі. Він набагато гнучкіший порівняно з реляційним форматом даних, який є фіксованим і незручним при зміні у схемах даних.

Обраний набір програмних технологій – це MERN (MongoDB, Express, React, Node) стек. Його архітектура дозволяє легко побудувати трирівневу архітектуру (візуальний інтерфейс, бекенд, база даних), повністю використовуючи JavaScript і формат обміну даних JSON.

2 РОЗРОБКА КЛІЄНТСЬКОЇ ТА СЕРВЕРНОЇ ЧАСТИНИ СИСТЕМИ

2.1 Визначення функціональних вимог до системи

Проектування та створення розподіленої системи вимагає розгляду питання вимог до програмного забезпечення, що описують внутрішню роботу системи: операцій над даними (обчислення, маніпулювання, обробка) ті інших специфічних функцій, які система зобов'язана вирішити в межах поставленого завдання. З погляду права, функціональні вимоги – це опис функцій, які реалізовані (пропонується реалізувати) у програмному продукті або посиляються на програмний продукт та обумовлюють його функціональні можливості, є підставою для технічних рішень при його створенні [13]. Визначення цих вимог відбувається на основі поставленого технічного завдання.

Функціональні вимоги складаються з двох частин: функції та поведінки. Функція – це виконувана дія системи. Поведінка залежить від підходу, як система це робить. Враховуючи особливість імплементованої в межах бакалаврського дослідження архітектури додатку – моделі “клієнт-сервер” – для кожної частини висувається свій окремий набір вимог.

Для клієнта висуваються наступні вимоги:

- надання користувальницького інтерфейсу. Враховуючи умови використання сторонніх API, виникає потреба в його ефективній взаємодії з іншими компонентами веб-інтерфейсу;
- формулювання запиту до сервера і його відправка. При взаємодії користувача з полем множинного вибору, картою, клієнт повинен сформулювати коректний HTTP-запит і надіслати його до сервера;
- отримання результатів запиту (запитів на додавання, оновлення або видалення даних) і формування на цій підставі веб-сторінки для перегляду.

У свою чергу, для сервера присутній свій список:

- зберігання, доступ, захист і резервне копіювання даних;
- обробка початкових даних з БД;

- обробка клієнтського запиту через відповідні HTTP-методи;
- відправлення результату (відповіді) клієнту.

2.2 Визначення нефункціональних вимог до системи

Для формування критеріїв оцінки якості ПЗ використовують нефункціональні вимоги.

Розглядаючи термін нефункціональних вимог, дамо його визначення. Нефункціональні вимоги – це обмеження або вимоги, що накладаються на систему, тобто формулюють саму сутність системи. Проблематикою охоплені питання масштабованості, зручності обслуговування, продуктивності, портативності, безпеки та надійності. Стосовно аспекту зберігання даних, то використання хмарного середовища суттєво знижує ризик системного збою даних, оскільки всі дані зберігаються та створюються резервні копії на зовнішньому пристрої (виділеному сервері). Окрім того, вживаються заходи щодо безпеки даних: аутентифікація, авторизація, шифрування.

Серед видів нефункціональних вимог розрізняють:

- вимоги до інтерфейсу (Interface Requirements);
- апаратні та програмні вимоги (Hardware/Software Requirements);
- операційні вимоги (Operational Requirements).

Враховуючи специфіку веб-додатку – клієнт-серверної програми в якій клієнтом виступає браузер, а сервером – веб-сервер – варто розглянути застосування інтерфейсів ПЗ. У даному випадку використовуються 2 типи мережних IP-протоколів: TCP/IP та HTTP. Протокол IP гарантує, що кожен комп'ютер, підключений до Інтернету, має певний серійний номер, який називається IP-адресою. TCP визначає, як відбувається обмін даними через Інтернет і як вони мають бути розбиті на IP-пакети. Він також переконується, що пакети містять інформацію про джерело даних повідомлення, адресат даних

повідомлення, послідовність, у якій дані повідомлення мають бути повторно зібрані, і перевіряє, чи повідомлення було надіслано правильно певному пункту призначення. HTTP використовується для передачі гіпертекстів через Інтернет і визначений `www` (всесвітньою павутиною) для передачі інформації. Цей протокол визначає, як потрібно формувати та передавати інформацію.

Апаратні та програмні вимоги (Hardware/Software Requirements) – опис апаратної та програмної платформ, необхідних для роботи (і підтримки) системи. Основний критерій у виборі надійного серверу — це вірогідна кількість запитів до серверу. Система розрахована на масове використання у зв'язку зі сферою застосування та призначенням. Тестування проводилось з використанням таких параметрів: 2 гігабайти ОЗП, пропускна здатність – 40 МБ/с та 5 гігабайт дискового простору. При таких середніх, на сьогодні, характеристиках, проблем у роботі серверу не спостерігалось. Зважаючи на це, рекомендовані характеристики системи будуть залежати лише від потоку користувачів. У свою чергу, клієнтська частина висуває вимоги, базуючись на складності передачі потокового відео, запитів до серверу та отримання картографічної інформації від сторонніх API. До мінімальних протестованих апаратних вимог можна віднести 2 ГБ ОЗП, процесор з частотою 1.8 ГГц. Вільний простір на диску, необхідний для стабільної роботи сервісу, становить 800 МБ. Середовищами тестування були обрані такі браузері: Google Chrome, Opera, Microsoft Edge та Safari. Під час тестування у роботі додатку не виявилось критичних проблем з швидкодією та коректністю завантаження ресурсів додатку.

Операційні вимоги (Operational Requirements). У цьому контексті суттєвою перевагою у користуванні розробленою комп'ютерною системою є універсальність вибору платформи: оскільки розподілена система працює, як на десктопній, так і на мобільній платформах, вибір операційної системи є довільним.

Інші види, такі як безпека та конфіденційність, надійність, відновлюваність, продуктивність, потенціал, збереження даних, правила перевірки, узгоджені стандарти задовольняються з боку MongoDB Atlas – багатохмарної служби баз

даних, що імплементує за основу концепцію MongoDB. У своїй роботі MongoDB Atlas користується підходом Database-as-a-Service (DBaaS), який дозволяє налаштовувати, розгортати та масштабувати базу даних у хмарі, не заглиблюючись у деталі налаштування продуктивності. Використання хмарного середовища суттєво знижує ризик системного збою даних, оскільки всі дані зберігаються та створюються резервні копії на зовнішньому пристрої (виділеному сервері). Окрім того, вживаються заходи щодо безпеки даних: аутентифікація, авторизація, шифрування.

У контексті нефункціональних вимог доречно враховувати думку О.Шипунової, яка описуючи вимоги до інформаційної системи, радить враховувати її функціональні можливості; сукупну вартість володіння; перспективи розвитку, підтримки і інтеграції та її технічні характеристики [11].

2.3 Опис структури схеми

Розроблювана розподілена система побудована на клієнт-серверній архітектурі, що складається з трьох компонентів: клієнт, сервер та база даних (рис. 2.1). Використання трьох рівнів, замість двох, дозволяє обробляти запити від великої кількості клієнтів та підвищує безпеку роботи додатку, оскільки тільки при зломі серверу злоумисник не матиме доступу до бази даних.

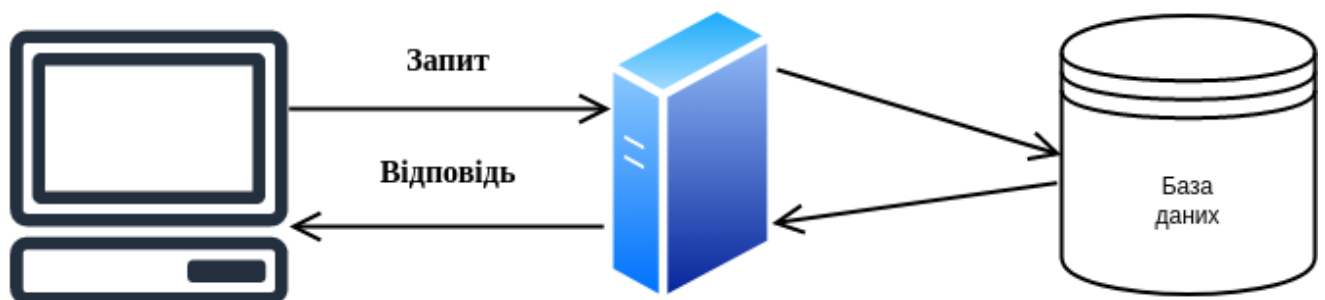


Рисунок 2.1 – Схема роботи системи

Клієнтська сторона додатку складається з вхідної точки App, що відповідає за маршрутизацію компонентів React-додатку, компонента Main, що ініціює підключення до API Google Maps та перенаправляє користувача до компонента Map. По суті, Map – основа додатку, оскільки на ньому відбувається рендеринг карти та накладення маркерів, відомості яких (назва локації, географічні координати, посилання на відеотрансляцію) сервер робить запит до БД та відправляє клієнту (рис.2.2).

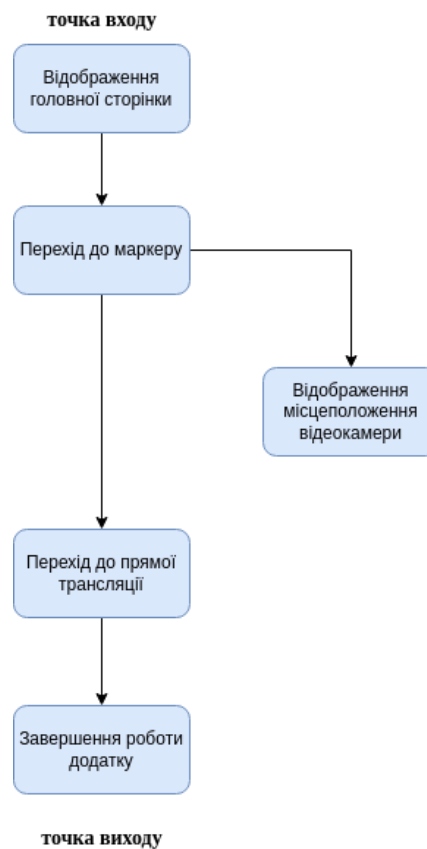


Рисунок 2.2 – Загальна схема роботи додатку

Процес вибору потрібної відеотрансляції відбувається шляхом взаємодії користувача з інтерактивною мапою Google Maps. Набір маркерів створюється методом попередньої ініціалізації запиту з боку клієнтської частини до серверу, який проводить запит на вибірку в сховищі даних.

Після отримання набору документів у форматі даних JSON сервер здійснює обробку полів географічних координат з метою їх перетворення із формату

“градуси, хвилини й секунди” (DMS) у формат “десяткових градусів” (DD). У подальшому відбувається зворотна передача набору колекції до клієнта.

Варто відмітити, що для коректної роботи проекту необхідно проініціалізувати користувацькі змінні середовища. З цією метою в серверній та клієнтській частині, в корені, створюється файл `.env`. Для клієнтської частини це API ключ для доступу до платформи Google Maps та географічні координати центру країни (у десятковому форматі), своєю чергою для серверної частини – це рядок підключення кластеру Atlas до середовища Node.js через драйвер MongoDB, назви затребуваних бази даних та колекції.

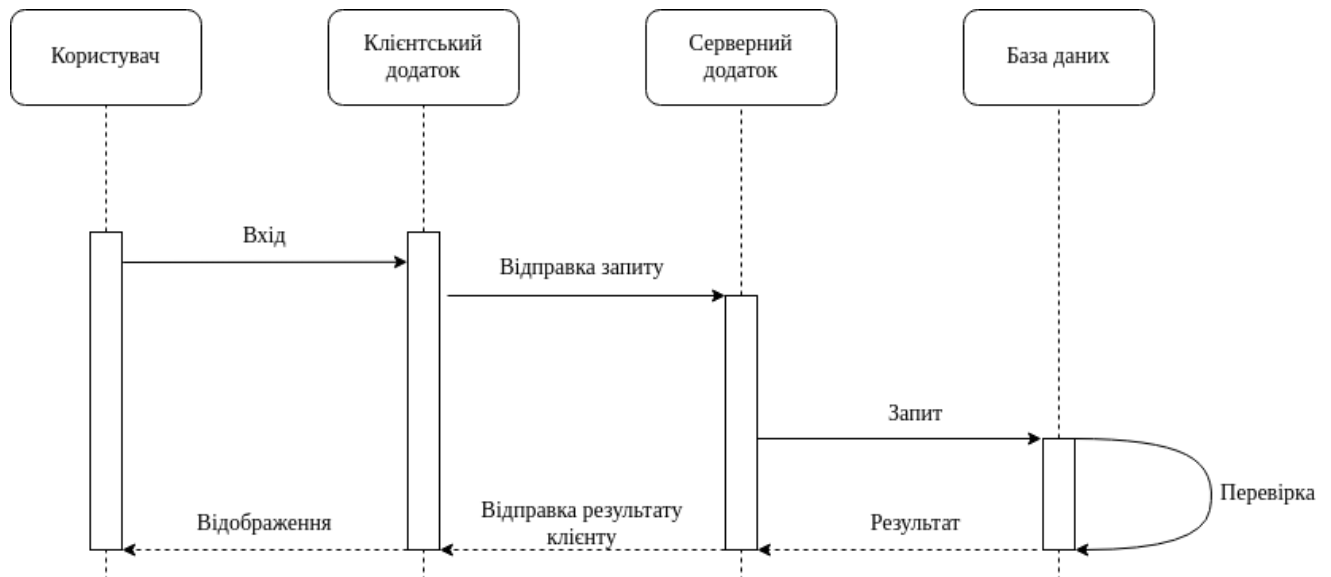


Рисунок 2.3 – UML-діаграма додавання маркерів на мапу

На рисунку 2.3 зображена UML-діаграма процесу додавання маркерів на мапу Google Maps. Першочергово, для роботи з додатком, користувач повинен перейти у компонент App. Клієнтський додаток відправляє запит на сервер, який робить запит до бази даних, сховище даних MongoDB Atlas зі свого боку перевіряє – чи збігається заданий сервером патерн документа з наявним у колекції. Після виявлення збігу сервер відправляє на клієнтський додаток відповідь у JSON-форматі.

Зазначимо, що у сервера немає графічного інтерфейса користувача, тобто в режимі headless-машини. На сьогодні даний вид комп'ютерної системи масово використовується в багатосерверних ЦОД.

2.4 Розробка модулів клієнтської частини

Для того, щоб почати працювати з Google Maps API, в додаток необхідно передати особистий ключ API. Ключ інтерфейсу прикладного програмування (API) – це код, який використовується для ідентифікації та автентифікації користувача. Ключі API доступні через платформи, такі як внутрішній ринок із білою міткою. Вони також діють як унікальний ідентифікатор і надають секретний маркер для цілей автентифікації.

У випадку з платформою Google Maps ключ API є послідовністю алфавітно-цифрових символів, що має довжину в 39 символів. Завантаження Google Maps API подано у лістингу 2.1.

Лістинг 2.1 - Завантаження Google Maps API

```
import React from "react";
import Map from "./Map";
import { useLoadScript } from "@react-google-maps/api";

const Main = () => {
  const { isLoading } = useLoadScript({
    //завантаження сценарію Google Maps API
    googleMapsApiKey: process.env.REACT_APP_GOOGLE_API_KEY
  });

  //вибір між екраном завантаження та Google-картою
  return (isLoading ?
    <Map/> :
    <h1>
      Loading...
    </h1>
  )
}

export default Main;
```

З метою отримання значень користувацьких змінних середовища у корені клієнтської частини створюється файл `.env`. Він містить перелік пар ключ-значення змінних середовища у форматі: `KEY=value`. Користувацькі змінні середовища клієнта подано у лістингу 2.2.

Лістинг 2.2 — Користувацькі змінні середовища клієнта

```
#символ * позначає облікові дані у формі ключа API до платформи
#Google Maps
REACT_APP_GOOGLE_API_KEY =
*****
REACT_APP_POLAND_GEO_CENTER = 52.2969935,16.6073242
```

У випадку з успішним завантаженням сценарію Google Maps API відбувається перехід до компоненту Map (див. лістинг 2.3), який відповідає за відображення Google-мапи та містить у своєму складі інші функціональні компоненти клієнтського додатку.

Лістинг 2.3 — Компонент карти

```
import { GoogleMap, Marker, InfoWindowF } from "@react-google-maps/api";
import styles from "../map_styles.css";
import { useMemo } from "react";
import { useEffect, useState } from "react";
import VideoStreamingWindow from "../VideoStreamingWindow";

const Map = () => {
  //ініціалізація змінних стану та функцій, що їх оновлюють
  const [markerCoordinates, setMarkerCoordinates] = useState([]);
  const [activeMarker, setActiveMarker] = useState(null);
  const [mapReference, setMapReference] = useState();
  const [infoWindowF, setInfoWindowF] = useState();
  const [videoStreamingWindow, setVideoStreamingWindow] =
    useState();
  const [isOpen, setIsOpen] = useState(false);
  const [isPreview, setIsPreview] = useState(false);
  const geo_center_Poland =
    process.env.REACT_APP_POLAND_GEO_CENTER.toString().split(',');

  //ініціалізація змінної, що відповідає за центр карти
  const geo_center = useMemo(() => ({
    lat:
      parseFloat(geo_center_Poland[0]),
    lng:
      parseFloat(geo_center_Poland[1])
  }), []);
```

Лістинг 2.4 — Оголошення функцій компоненту Map та хука ефекту useEffect()

```
//оголошення стрілкової функції, що обробляє одинарний
//клік на маркер
const handleMarkerClick = (id, title, location, lat, lng) => {
    mapReference?.panTo({lat, lng});
    setIsOpen(true);
    setIsPreview(false);
    setInfoWindowF({ id, title, location });
};

//оголошення стрілкової функції, що обробляє подвійне
//натискання на маркер
const handleMarkerDoubleClick = (id, livestream_availability,
link) => {
    setIsPreview(true);
    setVideoStreamingWindow({ id, livestream_availability,
link });
}

//отримання відповіді від серверу
useEffect(() => {
    async function getMarkerCoordinates() {
        const response = await
        fetch(`http://localhost:5050/`);
        if (!response.ok) {
            const message = `An error occurred:
            ${response.statusText}`;
            window.alert(message);
            return;
        }
        const coords = await response.json();
        setMarkerCoordinates(coords);
    }

    getMarkerCoordinates();
}, [markerCoordinates.length]);
```

За умови наявності обраного маркеру, через виклик компоненту VideoStreamingWindow та передачі посилання на відео, як об'єкта props, відбувається перехід до відеотрансляції (наведено на лістинг 2.5).

У класі GoogleMap наведено складне наслідування. Згідно з лістингу 2.6 відбувається формування множини маркерів на карті.

Лістинг 2.5 — Оголошення методу `return()` компоненту `Map`, фрагмент переходу до компоненту `VideoStreamingWindow`

```
return (
  <div className = "h-full w-1 text-3xl font-bold
  underline">
  {
    //перехід на компонент відеотрансляції
    isPreview &&
    <div>
      <VideoStreamingWindow
        link = {videoStreamingWindow.link}
      />
    </div>
  }
)
```

Лістинг 2.6 — Фрагмент оголошення класу `GoogleMap` та наслідованого класу `Marker` у методі `return()` компоненту `Map`

```
{
  //відображення Google-мапи
  !isPreview &&
  <GoogleMap
    center = {geo_center}
    zoom = {6.5}
    //обробка одинарного кліку на карті
    onClick = {(()) => setActiveMarker(null)}
    mapContainerClassName = "map-container"
  >
  {
    // відображення набору маркерів на карті
    markerCoordinates?.map(({ _id, title, location,
      latitude, longitude, livestream_availability,
      link }) => (
      <Marker
        key = {_id}
        position = {{lat: latitude, lng: longitude}}
        icon={{
          path:
            window.google.maps.SymbolPath.BACKWARD
              _CLOSED_ARROW,
          fillOpacity: 1,
          fillColor: "#fcfafa",
          strokeColor: "#f00c1b",
          scale: 5,
        }}
      //обробка одинарного кліку та виклик
      //відповідної функції
      onClick = {(()) => {
        handleMarkerClick(_id, title, location,
          latitude, longitude);
      }}
    </Marker>
  )
  )
}
```

```

    }}

    //обробка подвійного кліку та виклик
    //відповідної функції
    onDbClick = {() => {
        handleMarkerDoubleClick(_id,
                                livestream_availability, link);
    }}
>

```

Як видно з лістингу 2.7 за умови обрання маркеру відбувається показ інформаційної довідки.

Лістинг 2.7 — Продовження фрагменту оголошення класу GoogleMap та наслідованого класу Marker у методі return() компоненту Map

```

    {
        //показ інформаційної довідки про
        //маркер
        isOpen && infoWindowF.id === _id && (
            <InfoWindowF
                onCloseClick={() => {
                    setIsOpen(false);
                }}
            >
                <h2>{infoWindowF.title},
                {infoWindowF.location}</h2>
            </InfoWindowF>
        )
    }
    </Marker>))
}
</GoogleMap>
}
</div>
);
}

export default Map;

```

Звернімо увагу на те, що клієнт посилає запит до серверу на localhost-інтерфейс localhost, порт 5050, і очікує від нього відповіді. Безпосередньо заповнення масиву даних про маркери відбувається при виконанні side-effect'у: виходу за межі процесу потоку даних поточного компоненту та отримання даних JSON за допомогою інтерфейсу Fetch API.

Логічне зв'язування з наступним блоком (компонентом) додатку, VideoStreamingWindow (подано на лістингу 2.8), відбувається через стимулювання події DoubleClick з інтерфейсу MouseEvent. У якості вхідного параметру до нього передається посилання на відеотрансляцію з відеокамери.

Лістинг 2.8 — Блок трансляції потокового відео

```
import { useState, useEffect } from "react";
import ReactPlayer from "react-player";

const VideoStreamingWindow = (props) => {
  //ініціалізація змінної-посилання на відеострім
  const [ videoURL, setVideoURL ] = useState(props.link);

  return (
    <div>
      //показ відеоплеєра
      <ReactPlayer
        className = "react-player"
        url = {videoURL}
        width = "100%"
        height = "75%"
        controls
        playing
      />
    </div>
  );
};

export default VideoStreamingWindow;
```

У даному фрагменті використовується React Player, API розширень джерела медіа (MSE). Програвач заснований на відео- та аудіоелементах HTML5, щоб дозволити відтворення медіафайлів у браузерях. До нього передаються опціональні параметри, що відповідають за налаштування відеоплеєра.

2.5 Розробка модулів серверної частини

Серверна частина виконує зв'язуючу роль між клієнтом та базою даних, тобто його функціонал – обслуговування запитів клієнта та надання потрібного набору даних.

Особливістю реалізації серверного блоку розроблюваного веб-додатку є застосування мови програмування TypeScript, який є зворотно сумісним з JavaScript. TypeScript – це насправді дві різні, але пов'язані технології — мова та компілятор [12]:

- мова є багатофункціональною, статично типізованою мовою програмування, яка додає JavaScript справжні об'єктно-орієнтовані можливості;
- компілятор перетворює код TypeScript у JavaScript (транспіляція), але також надає програмісту допомогу у написанні коду з меншою кількістю помилок.

Як альтернатива до транспіляції TypeScript-коду застосовується функціонал Node-пакету `ts-node` – механізм виконання TypeScript і REPL для Node.js. Його JIT-компіляція перетворює TypeScript на JavaScript, дозволяючи безпосередньо виконувати TypeScript на Node.js без попередньої компіляції. Це досягається шляхом підключення API-інтерфейсів завантаження модулів Node, що дозволяє безперешкодно використовувати його разом з іншими інструментами та бібліотеками Node.js.

Організація роботи серверу розподілена за загальними складовими: `db`, `helpers`, `routes` та `server`. Проаналізуємо роботу кожної складової.

Модуль `db`. Під'єднання до кластеру MongoDB Atlas потребує передачі рядка підключення кластеру Atlas до середовища Node.js через об'єкт класу `MongoClient()`. Обробка помилок під'єднання відбувається у блоці `try...catch`. Після цього отримуємо доступ до бази даних та колекції, визначивши їх імена через глобальну змінну `process.env`.

Вище описана послідовність дій наведена у формі програмного коду на лістингу 2.9.

Лістинг 2.9 — Під'єднання до бази даних

```
import * as mongodb from "mongodb";
import * as dotenv from "dotenv";

//визначення колекції cities
export const collections: { cities?: mongodb.Collection } = {}

export async function connectToDatabase () {
  dotenv.config();
  //визначення клієнта MongoDB
  const client: mongodb.MongoClient = new
mongodb.MongoClient(process.env.ATLAS_URI as string);
  try {
    //підключення до кластеру MongoDB Atlas
    await client.connect();
    console.log("Connected to the MongoDB Atlas Cluster");
  } catch(e) {
    console.error(e);
  }
  //визначення БД
  const db: mongodb.Db = client.db(process.env.DB_NAME as
string);
  //визначення колекції з БД
  const citiesCollection: mongodb.Collection =
db.collection(process.env.COLLECTION_NAME as string);
  //присвоєння колекції cities значенню отриманої колекції з
//БД
  collections.cities = citiesCollection;
  console.log(`Successfully connected to database:
${db.databaseName} and collection:
${citiesCollection.collectionName}`);
}
```

Аналогічно до клієнтської частини, проініціалізуємо користувацькі змінні середовища у файлі .env в корені серверного блоку, згідно з лістингу 2.10.

Лістинг 2.10 — Користувацькі змінні середовища сервера

```
#рядок підключення кластеру Atlas
ATLAS_URI =
mongodb+srv://*****:*****@cluster1.j1bhddc.mongodb
.net/?retryWrites=true&w=majority
DB_NAME = poland_videocameras #назва бази даних
COLLECTION_NAME = cities #назва колекції
```

Безпосереднє отримання даних з БД та зберігання їх на серверній частині відбувається через NoSQL-запити та визначення класу Videocamera, згідно з лістингу 2.11 та лістингу 2.12, поля якого є ідентичними до полів колекції.

Лістинг 2.11 — Клас Videocamera

```
import { ObjectId } from "mongodb";
//ObjectId є унікальним ідентифікатором для кожного документа

//створення індивідуального типу даних position_type
type position_type = {
  latitude: string;
  longitude: string;
}

//визначення класу Videocamera
export default class Videocamera {
  constructor(public title: string, public location: string,
    public position: position_type, public livestream_availability:
    boolean, public link: string, public _id?: ObjectId) {}
}
```

Лістинг 2.12 – Отримання даних з БД сервером

```
import { ObjectId } from "mongodb";
import { collections } from "../database.service";
import Videocamera from "../videocamera";

//створення нового типу даних, аналогу об'єкта в MongoDB
type position_format = {
  latitude: string;
  longitude: string;
}

//визначення інтерфейсу для заповнення масивів маркерів
interface marker_attributes {
  _id: ObjectId;
  title: string;
  location: string;
  position: position_format;
  livestream_availability: boolean;
  link: string;
}

export async function retrieve_data() {
  const markers = [];
  try {
    //отримання документів з колекції
```

```

        const results = (await
collections.cities?.find({}).toArray()) as unknown as
Videocamera[];
    for (const val of results) {
        //приведення до стандартизованого виду через
        //інтерфейс marker_attributes
        const marker: marker_attributes = {
            _id: new ObjectId(val._id),
            title: val.title,
            location: val.location,
            position: val.position,
            livestream_availability:
val.livestream_availability,
            link: val.link
        }
        //занесення отриманого значення до масиву
        //markers
        markers.push(marker);
    }
} catch (e) {
    console.error(e);
}
return markers;
}

```

Модуль `helpers` відповідає за обробку даних колекції та приведення їх у зручний для інтерпретації клієнтом вигляд, подано на лістингу A.1. Під цим розуміється перетворення географічних координат з формату “градусів, хвилин й секунд” (DMS) у формат “десяткових градусів” (DD). Потреба в даному перетворенні полягає у визначенні місцезнаходження маркера шляхом надання властивості `position`, який є парою числових значень, `latitude` (широта) та `longitude` (довгота).

Основна робота серверу полягає в коректній маршрутизації, — наданні правильної відповіді на запит клієнта згідно певної кінцевої точки. Ця кінцева точка складається зі шляху сторінки в межах хосту та HTTP-методу: GET, POST, PUT, DELETE. Маршрутизація серверу описана у модулі `routes`.

Комп’ютерна система використовує здебільшого метод GET.

Ефективна маршрутизація в серверному додатку досягається шляхом застосування проміжних запитів `express.Router`, оскільки вони дозволяють групувати обробники маршрутів для певної частини сайту та отримувати доступ до них через загальний префікс маршруту (лістинг A.2).

Ядром роботи серверу є функціонування модулю `server`. Згідно з ліст. 2.13 робота серверу починається після виконаного асинхронного підключення до БД MongoDB, що ініціалізує здійснення методу `then()` об'єкту `Promise`. Виявлення помилки під'єднання відбувається за допомогою методу `catch()`.

У разі успішного підключення викликаються методи з'єднання через призначені хост і порт. Окрему увагу слід приділити застосування описаного у модулі `routes` об'єкта `express.Router` у якості функції проміжного програмного забезпечення (`middleware`).

Лістинг 2.13 – Основа роботи серверу

```
import express from "express";
import cors from "cors";
import { connectToDatabase } from "../db/database.service"
import { videocamerasRouter } from "../routes/videocameras.router";

//визначення порту протоколу HTTP
//який використовується для підключення до сервера
const PORT = process.env.PORT || 5050;
const app = express();

//виклик middleware-функцій
app.use(cors());
app.use(express.json());

connectToDatabase()
  .then(() => {
    //налаштування маршрутизації HTTP запитів
    app.use("/", videocamerasRouter);

    //прослуховування на визначеному порті
    app.listen(PORT, () => {
      console.log(`Server started at
      http://localhost:${PORT}`);
    });
  })
  //обробка помилки підключення до БД
  .catch((error: Error) => {
    console.error("Database connection failed: ", error);
    process.exit();
  });
```

2.6 Розробка бази даних додатку

Для коректної роботи додатку було створено колекцію з 48 документів у СКБД MongoDB.

За аналогією до первинного ключа в реляційних базах даних, БД MongoDB містить унікальний ідентифікатор документа `ObjectId`, котрий являє собою 12-байтове шістнадцяткове число, з яких 4 байти займає позначка часу створення `ObjectId` (в секундах), 5 байтів — випадкове значення, яке генерується один раз на процес; 3 байти — інкрементний лічильник, ініціалізований до випадкового значення.

Кожний документ містить інформацію про відеокамеру у парах “ключ-значення” у форматі BSON (Binary JSON). Екземпляр такого документа наведений на лістинг 2.14.

Лістинг 2.14 — JSON-документ

```
{
  "_id": {
    "$oid": "64660010ad14bb6e54cd055e"
  },
  "title": "Королівський замок на Вавелі",
  "location": "Kraków",
  "position": {
    "latitude": "50°3'17\" N",
    "longitude": "19°56'2\" E"
  },
  "livestream_availability": true,
  "link": "https://player.webcamera.pl/krakow_cam_4c3e37"
}
```

Група документів об'єднуються у колекції. Для забезпечення цілісності даних та приведення їх до єдиного стандарту використовується валідація схеми. Правило валідації описано у лістинг 2.15.

Лістинг 2.12 — Схема валідації

```
{
  $jsonSchema: {
```

```

    required: [
      'title',
      'location',
      'position',
      'livestream_availability',
      'link'
    ],
    properties: {
      title: {
        bsonType: 'string',
        description: 'must be a string and is required'
      },
      location: {
        bsonType: 'string',
        description: 'must be a string and is required'
      },
      position: {
        bsonType: 'object',
        required: [
          'latitude',
          'longitude'
        ],
        properties: {
          latitude: {
            bsonType: 'string'
          },
          longitude: {
            bsonType: 'string'
          }
        }
      },
      livestream_availability: {
        bsonType: 'bool',
        description: 'must be a boolean and is required'
      },
      link: {
        bsonType: 'string',
        description: 'must be a string and is required'
      }
    }
  }
}

```

Документ — головна синтаксична одиниця БД, з якою взаємодіє сервер із застосуванням драйверу MongoDB Node.js.

Суттєвою перевагою застосування документо-орієнтованої БД MongoDB у розроблюваному застосунку є швидкодія NoSQL-запитів, в основному методів читання `find()` або `findOne()`, які відповідно до операторів запиту, що

застосовуються до певних полів, визначають набір вихідних документів. Аналогічний принцип дії застосовується до операцій запису, які дозволяють вставляти, змінювати або видаляти документи у базі даних MongoDB через відповідні програмні методи.

3 ТЕСТУВАННЯ РОЗРОБЛЕНОЇ СИСТЕМИ

З метою перевірки повної працездатності системи, існує потреба у моделюванні сценаріїв взаємодії користувача із розподіленим додатком.

Функціонал додатку передбачує вибір маркера відеокарти на території Республіки Польща шляхом маніпуляції з мишею:

- одинарне натискання відображає інформаційну довідку із зазначенням назви локації: об'єкт системи міського простору, місцеположення в межах населеного пункту, назва міста (польською чи англійською мовами);
- при подвійному кліку користувач переходить до сторінки з трансляцією потокового відео. Важливим зауваженням є формат потокового відео.

Стосовно формату файлів потокової передачі через інтернет-канали, то вона може бути 2 варіантів: у формі URL-посилання на зовнішній інтернет-ресурс, що містить плеєр потокового відео, або ж як схема blob URI, — схеми уніфікованого ідентифікатора ресурсу (URI), яка використовується для доступу до локально згенерованих даних через API.

Приклад blob URI наведений на лістинг 3.1.

Лістинг 3.1 — Екземпляр blob URI

```
blob:https://lublin.eu/896e92d2-d3af-4b80-b80e-feb55a5edaf5
```

У якості API-сервісів використовувались медіа-платформи польського провайдеру телекомунікацій, кабельного та цифрового телебачення TOYA та

міських порталів Республіки Польща. Вибіркова сукупність міських веб-сайтів складається на підставі критерію чисельності населення, а саме міста з чисельністю понад 250.000 жителів.

Етапи мапи сценарію користувача (user flow) покроково відображені на рис. 3.1 - 3.4

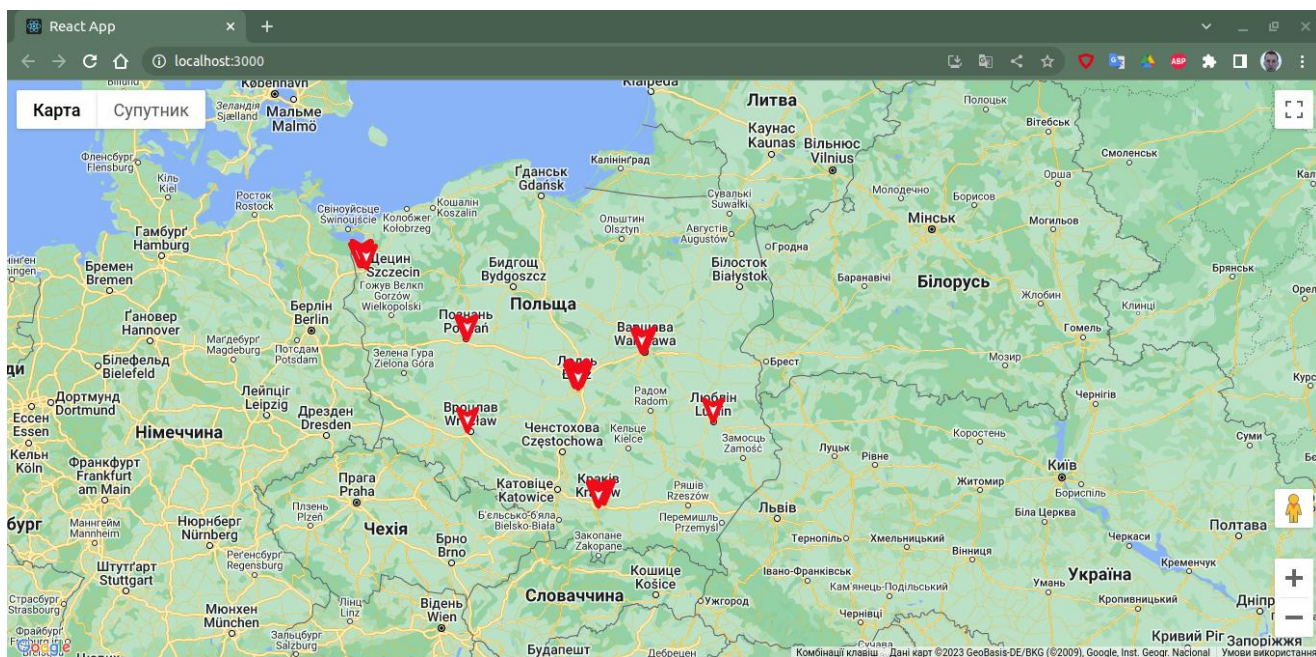


Рисунок 3.1 — Нанесення маркерів на Google-мапу

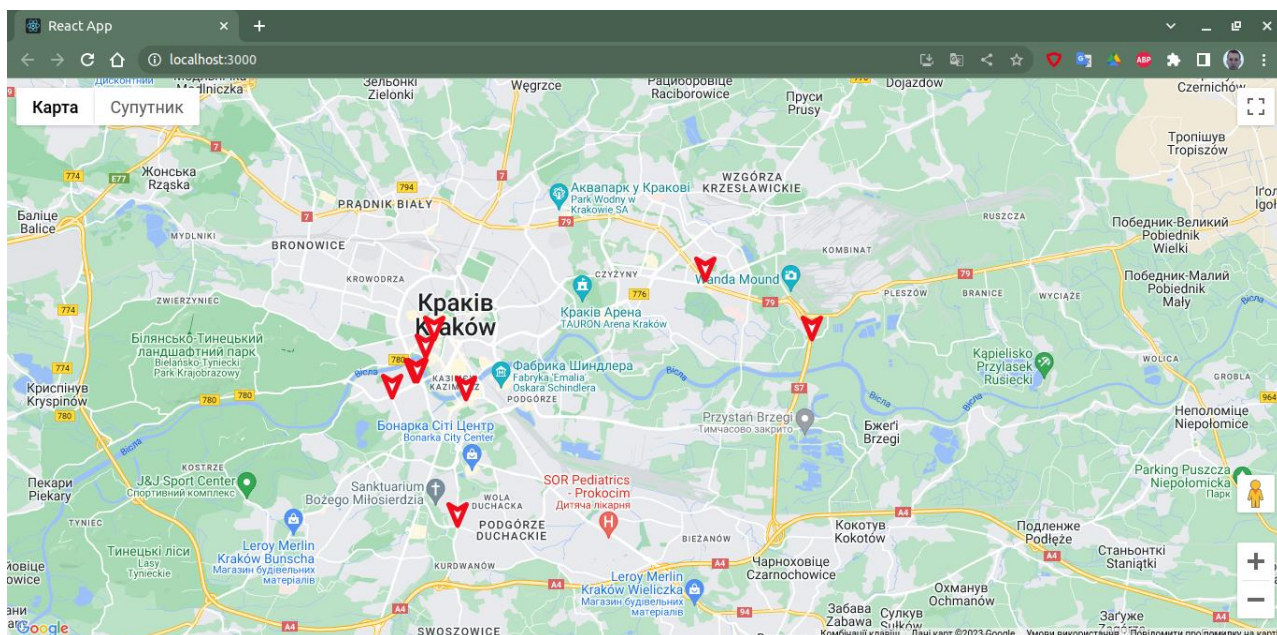


Рисунок 3.2 — Місцезнаходження маркерів в межах одного міста

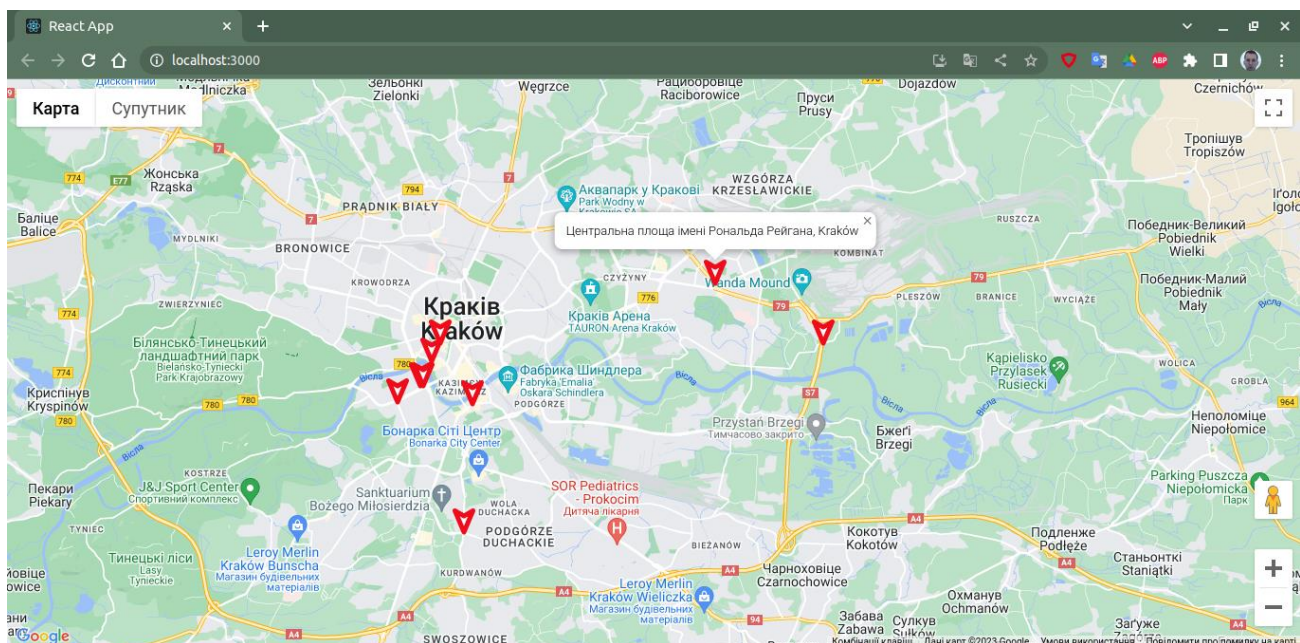


Рисунок 3.3 — Отримання інформаційної довідки

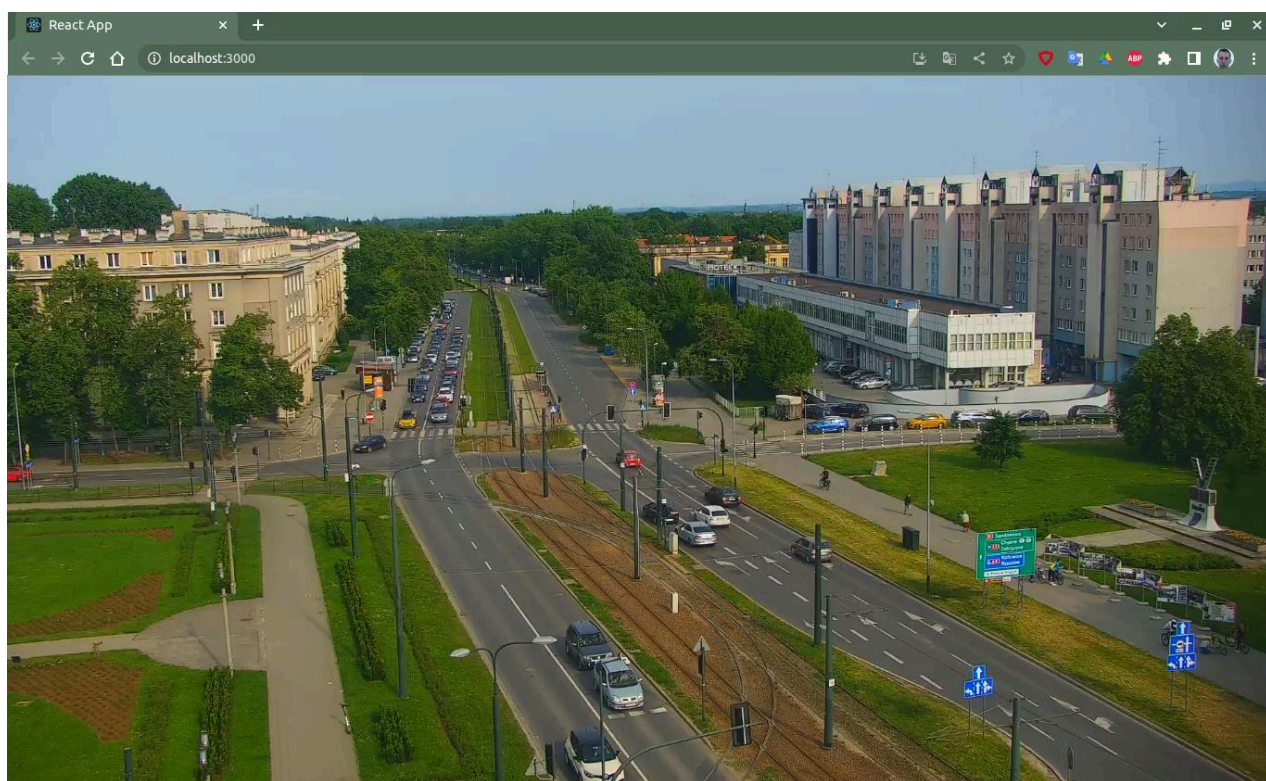


Рисунок 3.4 — Перегляд потокового відео з веб-камери

Таким чином, було проведено тестування додатку з використанням сценарію доступу до відеокамери.

ВИСНОВКИ

У ході проведення роботи був зроблений короткий огляд конкурентоспроможності існуючих систем спостереження дорожнього руху. На підставі цього зроблено вибір застосованого методу моніторингу дорожнього трафіку — трансляція потокового відео, переданого у мережу за допомогою IP-камер відеоспостереження.

Було проведено порівняльний розгляд інформаційних технологій для побудови клієнтської, серверної частин та сховища опрацьованих даних: дано їх визначення, описано особливості імплементації та ефективний контекст використання, виділено їх переваги та недоліки та обрано найефективніший варіант з урахуванням специфіки функціоналу веб-застосунку та особливостей його архітектури.

Було виконана програмна реалізація розроблюваної системи, проведене тестування роботи вихідного продукту, на основі якого зроблена оцінка задоволення користувацьких критеріїв.

Проведено тестування системи на прикладі сценарію взаємодії користувача з веб-інтерфейсом додатку: доступу до трансляції потокового відео з камери.

З урахуванням проведеного обсягу роботи, можна зробити підсумок та зазначити, що поставлене в межах бакалаврського дослідження розробки оптимальних процесів моніторингу дорожнього трафіку за допомогою програмного продукту було виконано.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

- 1 M. Elloumi, R. Dhaou, B. Escrig, H. Idoudi and L. A. Saidane, "Monitoring road traffic with a UAV-based system," 2018 IEEE Wireless Communications and Networking Conference (WCNC), Barcelona, Spain, 2018, pp. 1-6, doi: 10.1109/WCNC.2018.8377077. URL: <https://ieeexplore.ieee.org/abstract/document/8377077> (дата звернення 27.04.2023).
- 2 Krause, J. (2020). Introduction to Bootstrap. In: Introducing Bootstrap 4. Apress, Berkeley, CA. URL: https://doi.org/10.1007/978-1-4842-6203-0_1 (дата звернення 30.04.2023).
- 3 Rasmus Lerdorf, Kevin Tatroe. Programming PHP: O'Reilly Media, 2002. 524 с.
- 4 Jason Hunter, William Crawford. Java Servlet Programming: Help for Server Side Java Developers (Java Series) Second Edition: O'Reilly Media, 2001. 782 с.
- 5 Martin Heller. What is Node.js? The JavaScript runtime: InfoWorld. URL: <https://www.infoworld.com/article/3210589/what-is-nodejs-javascript-runtime-explained.html> (дата звернення 02.05.2023).
- 6 Paul DuBois. MySQL: Pearson Education, 2008. 1224 с.
- 7 Michael Widenius, David Axmark, Kaj Arno. MySQL Reference Manual: Documentation from the Source: O'Reilly Media, 2002. 816 с.
- 8 Justin Alexander F. Ravag. Comparison of MySQL and MS SQL Server. Lyceum of the Philippines University. Laguna, Philippines. С.1.
- 9 Stanislav Zablotskiy. ADVANTAGES AND DISADVANTAGES OF USING POSTEGRESQL IN CLOUD TECHNOLOGIES: Міжнародна науково-практична конференція молодих вчених, аспірантів та студентів — Інформаційні технології в сучасному світі: дослідження молодих вчених: тези доповідей, 12 – 13 березня 2015 р. – Харків: ХНЕУ ім. Семе́на Кузне́ця, 2015. С.16.
10. Zave P., Jackson M. Four Dark Corners of Requirements Engineering // ACM Transactions on Software Engineering, January 1997.– № 1.

11. Шипунова О.В. Критерії вибору інформаційної системи / О.В. Шипунова // Вісник УБС НБУ. - №2(11). - 2011. - С.173-175
12. David Choi. Full-Stack React, TypeScript, and Node: Build cloud-ready web applications using React 17 with Hooks and GraphQL: Packt Publishing, 2020. 648 с.
13. Рішення Національної комісії з цінних паперів та фондового ринку від 02.10.2012 №1342 // Інформаційно-правові системи LIGA:ZAKON // Інформаційне агентство "ЛІГА:ЗАКОН". URL: https://ips.ligazakon.net/document/re22072?an=1&ed=0000_00_00 (дата звернення 07.05.2023).

Додаток А

Лістинг А.1 – Обробка інформації зі сховища даних

```

import { ObjectId } from "mongodb";
import { retrieve_data } from "../db/retrieve";
//визначення інтерфейсу подання даних про відеокамери
interface videocamera_format {
  _id: ObjectId,
  title: string,
  location: string,
  latitude: number,
  longitude: number,
  livestream_availability: boolean,
  link: string
}
//оголошення функції, яка перетворює географічні координати з
//формату градуси/хвилини/секунди у формат десяткових градусів
function DMS_to_DD (DMS_coordinate: string): number {
  const units = ['°', '\'', '"'];
  let processed_DMS_coordinate = DMS_coordinate;
  let sum = 0.0;
  let fact = 1.0;
  for (let y = 0; y < 3; y++) {
    sum
    parseFloat(processed_DMS_coordinate.split(units[y])[0])/fact; +=
    fact *= 60;
    processed_DMS_coordinate
    processed_DMS_coordinate.split(units[y])[1]; =
  }

  if (processed_DMS_coordinate[processed_DMS_coordinate.length -
1] == 'S' ||
processed_DMS_coordinate[processed_DMS_coordinate.length - 1] ==
'W')
    sum *= -1.0;
  return parseFloat(sum.toFixed(6));
}

export async function coordinates_transformation() {
  //отримання стандартизованих даних про маркери
  const initial_videocameras = await retrieve_data();
  const final_videocameras = [];
  for (let i = 0; i < initial_videocameras.length; i++) {
    const videocamera: videocamera_format = {
      _id: initial_videocameras[i]._id,
      title: initial_videocameras[i].title,
      location: initial_videocameras[i].location,
      //перетворення широти

```

```

        latitude:          DMS_to_DD(initial_videocameras
[i].position.latitude),
        //перетворення довготи
        longitude:         DMS_to_DD(initial_videocameras
[i].position.longitude),
        livestream_availability: initial_videocameras
[i].livestream_availability,
        link: initial_videocameras[i].link
    };
    //занесення до масиву даних про відеокамери
    final_videocameras.push(videocamera);
}
return final_videocameras;
}

```

Лістинг А.2 – Маршрутизація запитів клієнта

```

import express, { Request, Response } from "express";
import dotenv from "dotenv";
import { ObjectId } from "mongodb";
import { collections } from "../db/database.service";
import Videocamera from "../db/videocamera";
import { coordinates_transformation } from
"../helpers/coordinate_transform";

//налаштування маршрутизації та обробка даних у форматі JSON
export const videocamerasRouter = express.Router();
videocamerasRouter.use(express.json());

//об'єкт, що представляє проаналізований файл .env.
dotenv.config();

//отримання даних колекції через HTTP-метод GET та надсилання
//відповіді клієнту
videocamerasRouter.get("/", async (req: Request, res: Response)
=> {
    const videocameras = await coordinates_transformation();
    res.send(videocameras).status(200);
});

//отримання документу відеокамери через HTTP-метод GET та
//надсилання відповіді клієнту
videocamerasRouter.get("/:id", async (req: Request, res:
Response) => {
    const id = req?.params?.id;
    try {
        const query = { _id: new ObjectId(id) };
        const videocamera = await (await
collections.cities?.findOne(query)) as unknown as
Videocamera;
        if (videocamera) {
            res.status(200).send(videocamera);
        }
    }
}

```

```

    } catch (error) {
        res.status(404).send(`Unable to find matching
        document with id: ${req.params.id}`);
    }
});

//створення документу відеокамери через HTTP-метод POST та
//надсилання відповіді клієнту
videocamerasRouter.post("/", async (req: Request, res:
Response) => {
    try {
        const newVideocamera = req.body as Videocamera;
        const result = await
collections.cities?.insertOne(newVideocamera);

        result
        ? res.status(201).send(`Successfully created a new
videocamera with id ${result.insertedId}`)
        : res.status(500).send("Failed to create a new
videocamera.");
    } catch (error) {
        console.error(error);
        if (error instanceof Error)
            res.status(400).send(error.message);
    }
});

//заміна документу відеокамери корисним навантаженням запиту
//через HTTP-метод PUT та надсилання відповіді клієнту
videocamerasRouter.put("/:id", async (req: Request, res:
Response) => {
    const id = req?.params?.id;

    try {
        const updatedGame: Videocamera = req.body as
Videocamera;
        const query = { _id: new ObjectId(id) };
        const result = await
collections.cities?.updateOne(query, { $set: updatedGame
});

        result ? res.status(200).send(`Successfully updated
videocamera with id ${id}`)
        : res.status(304).send(`Videocamera with id: ${id}
not updated`);
    } catch (error) {
        if (error instanceof Error) {
            console.error(error.message);
            res.status(400).send(error.message);
        }
    }
}

```

```

    });

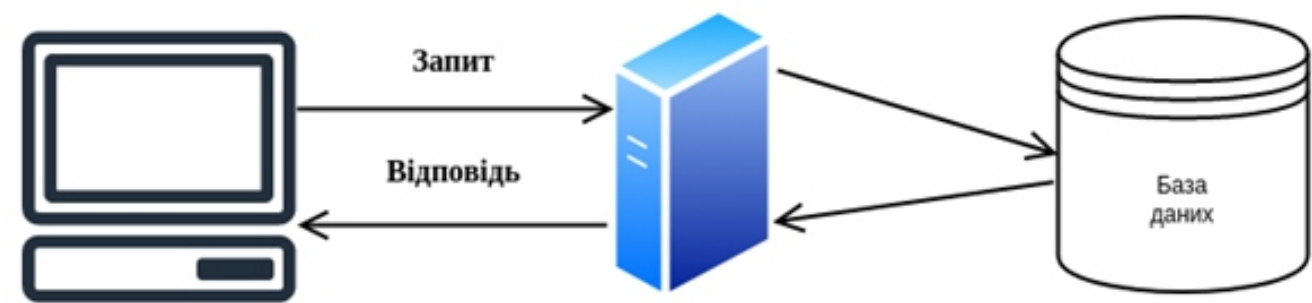
//видалення документу відеокамери через HTTP-метод DELETE та
//надсилання відповіді клієнту
videocamerasRouter.delete("/:id", async (req, res) => {
    const id = req?.params?.id;

    try {
        const query = { _id: new ObjectId(id) };
        const result = await collections.cities?.deleteOne(query);

        if (result && result.deletedCount) {
            res.status(202).send(`Successfully removed
            videocamera with id ${id}`);
        } else if (!result) {
            res.status(400).send(`Failed to remove videocamera
            with id ${id}`);
        } else if (!result.deletedCount) {
            res.status(404).send(`Videocamera with id ${id} does
            not exist`);
        }
    } catch (error) {
        if (error instanceof Error) {
            console.error(error.message);
            res.status(400).send(error.message);
        }
    }
});

```

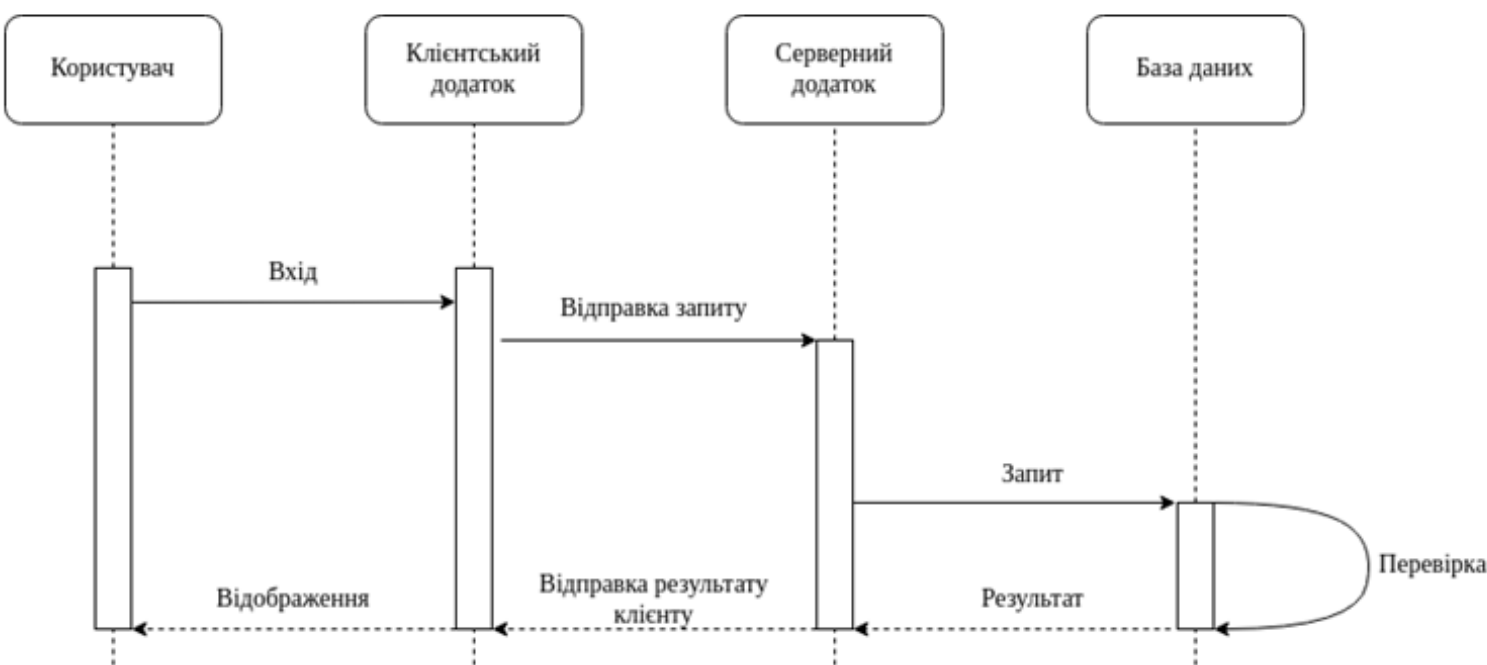
Схема роботи системи



Загальна схема роботи додатку

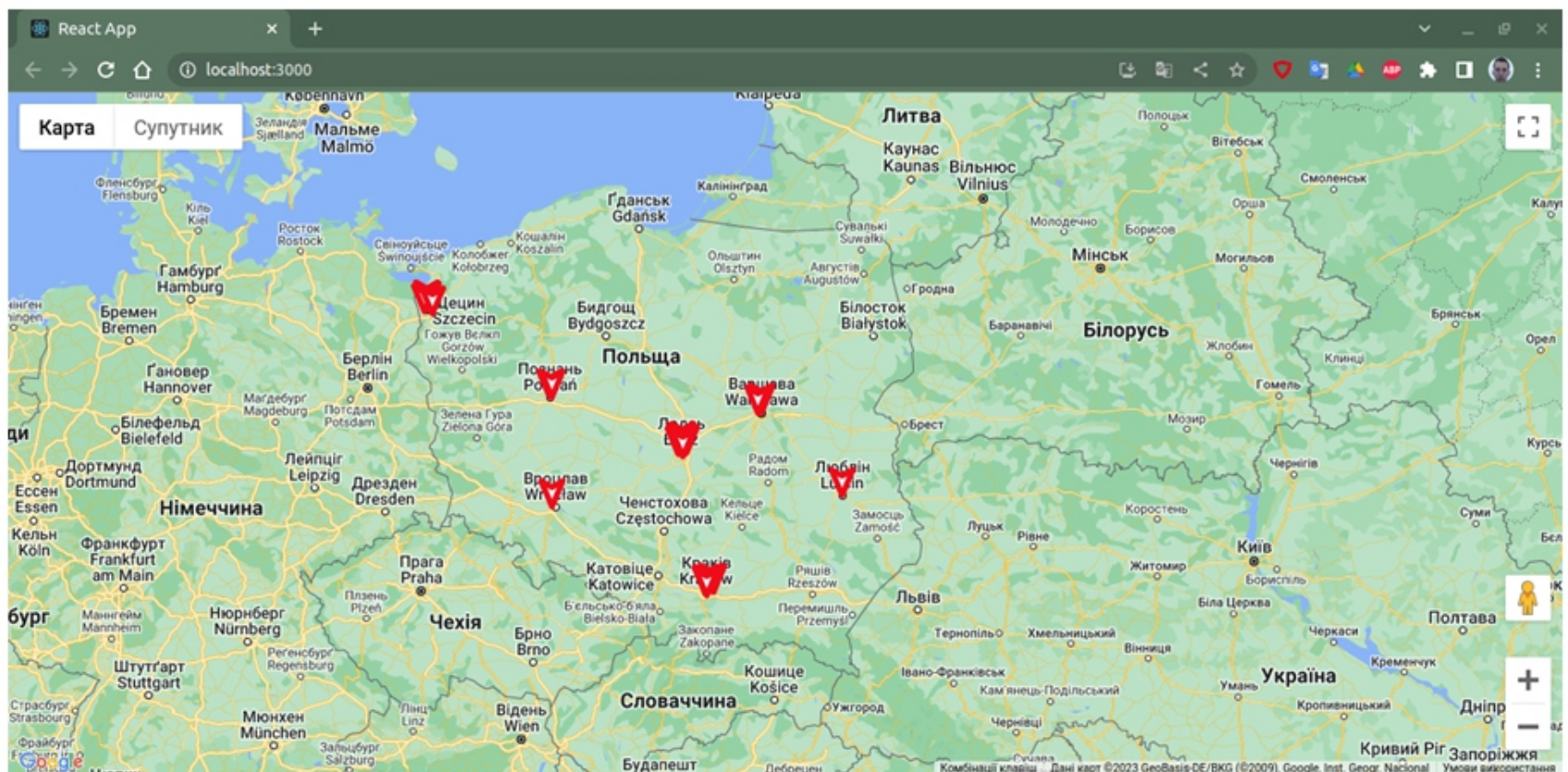


UML-діаграма додавання маркерів на мапу

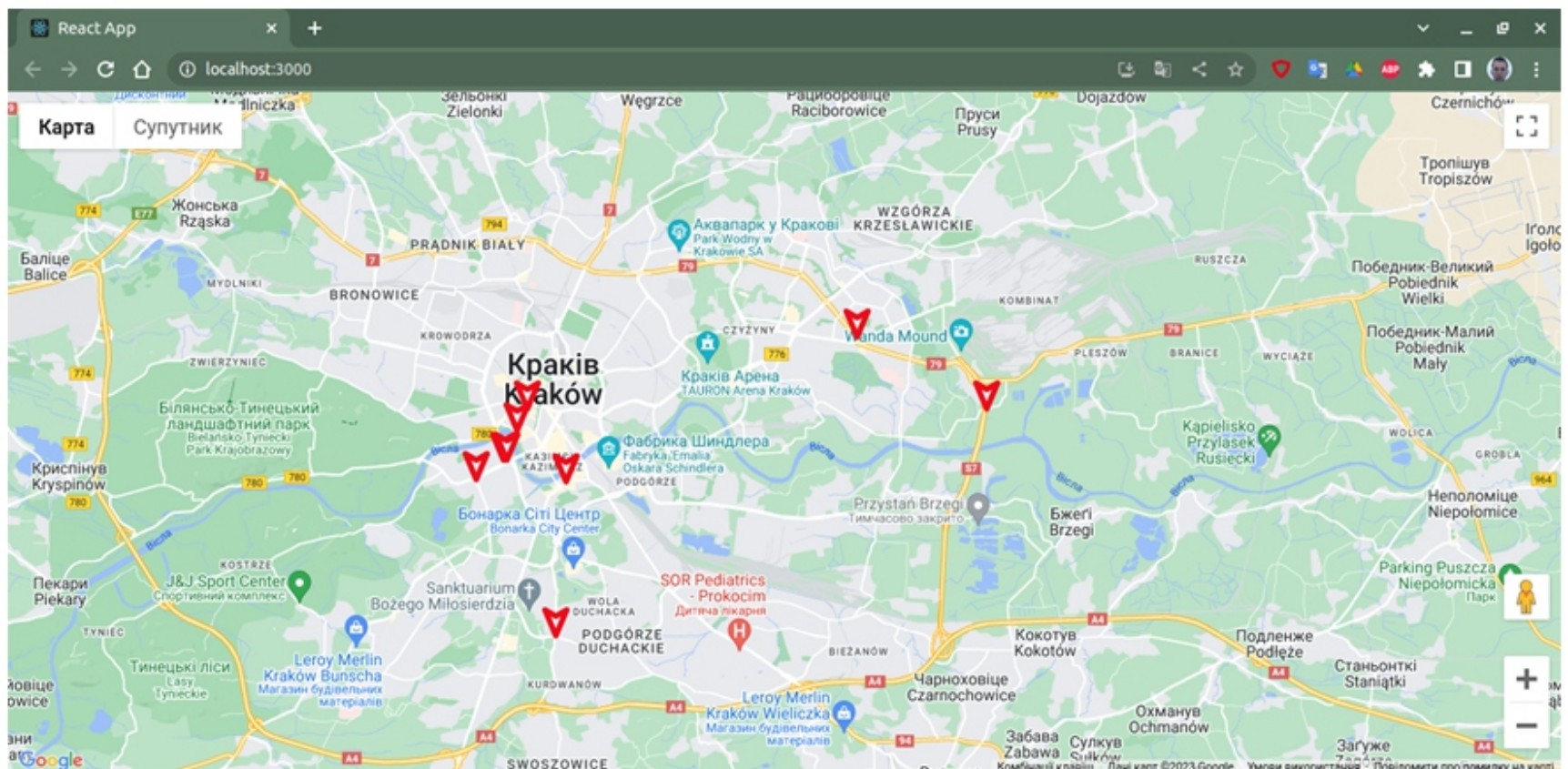


					13.02070849.51904 ПЛ1								
Зм.	Лист	№ документа	Підпис	Дата	Розробка комп'ютерної системи з моніторингу дорожнього трафіку на міжнародних, національних та регіональних дорогах України Схема роботи системи, загальна схема роботи додатку, UML-діаграма додавання маркерів на мапу					Літ.		Лист	Листів
Розробив		Герасін Ю.В.										1	1
Перевірів		Скрупський С.Ю.											
Н. контр.		Щербак Н.В.											
Затвердив		Кудерметов Р.К.								НУ «Запорізька політехніка», група КНТ-519			

Нанесення маркерів на Google-мапу

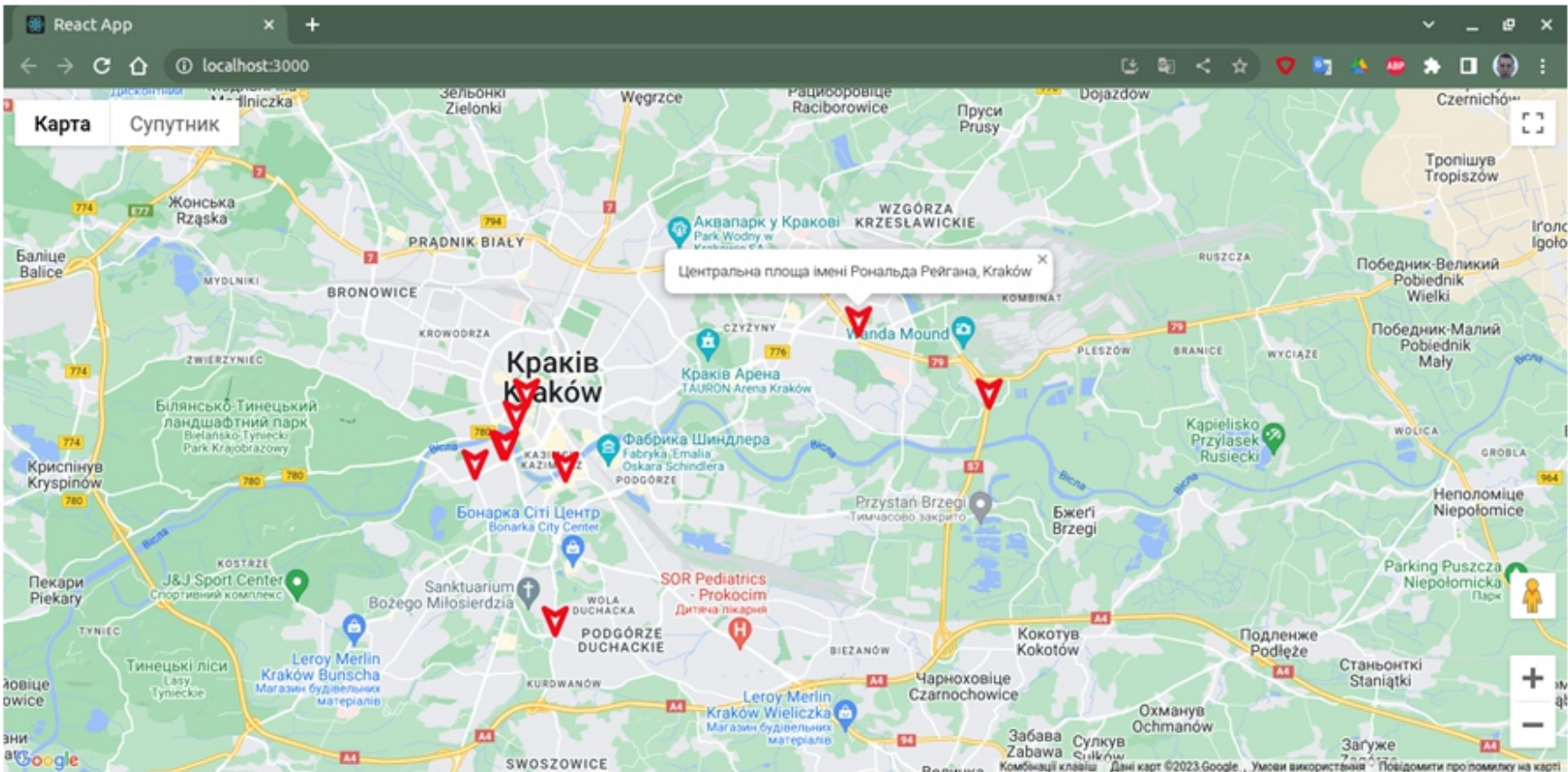


Місцезнаходження маркерів в межах одного міста

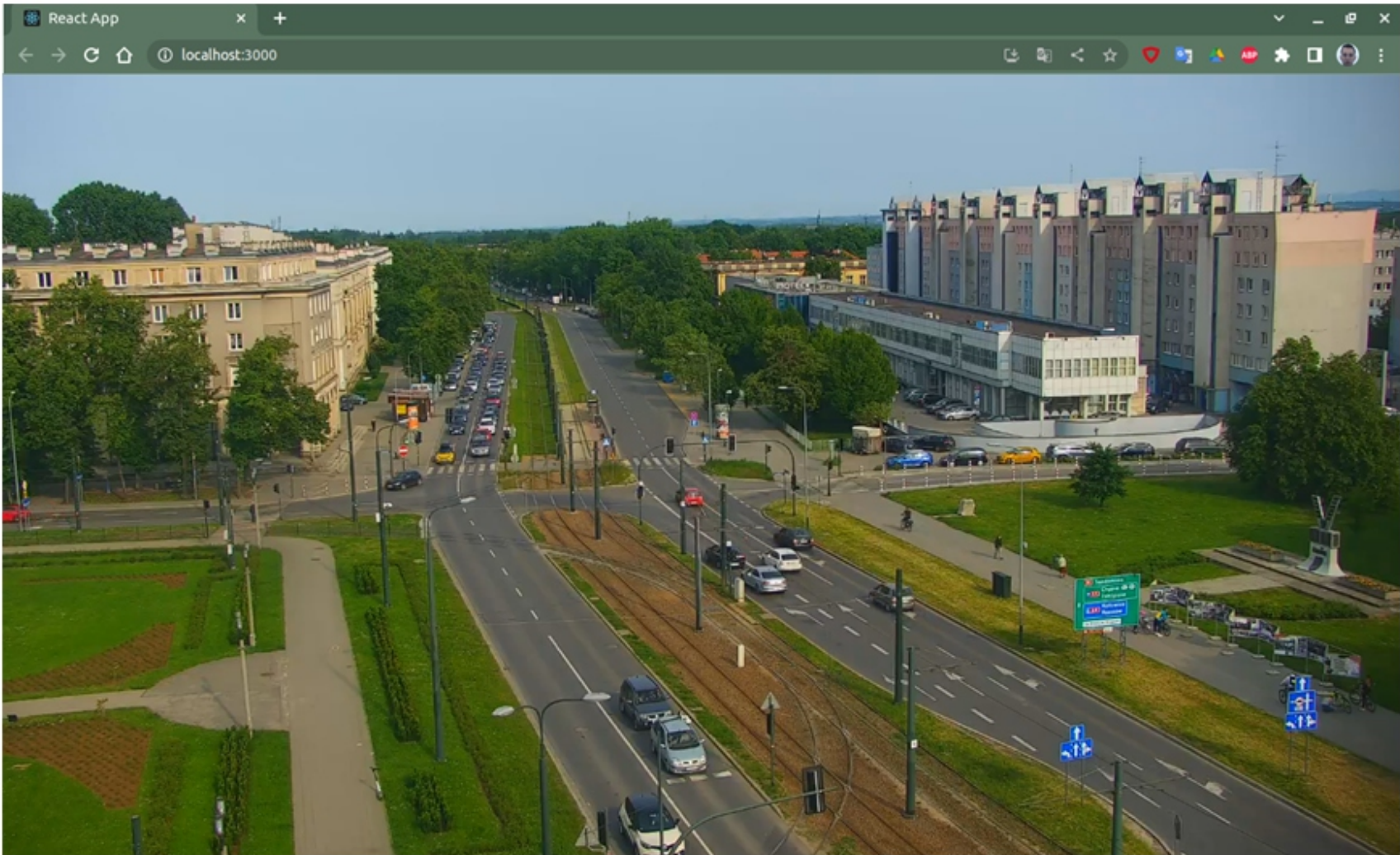


					13.02070849.51904 ПЛ2							
Зм.	Лист	№ документа	Підпис	Дата	Розробка комп'ютерної системи з моніторингу дорожнього трафіку на міжнародних, національних та регіональних дорогах України Нанесення маркерів на Google-maps, місцезнаходження маркерів в межах одного міста	Літ.		Лист		Листів		
Розробив		Герасін Ю.В.						1		1		
Перевірив		Скрупський С.Ю.										
Н. контр.		Щербак Н.В.										
Затвердив		Кудерметов Р.К.				НУ «Запорізька політехніка», група КНТ-519						

Отримання інформаційної довідки



Перегляд потокового відео з веб-камери



					13.02070849.51904 ПЛЗ						
Зм.	Лист	№ документа	Підпис	Дата	Розробка комп'ютерної системи з моніторингу дорожнього трафіку на міжнародних, національних та регіональних дорогах України Отримання інформаційної довідки перегляд потокового відео з веб-камери	Літ.		Лист		Листів	
Розробив		Герасін Ю.В.						1		1	
Перевірив		Скрупський С.Ю.									
Н. контр.		Щербак Н.В.									
Затвердив		Кудерметов Р.К.					НУ «Запорізька політехніка», група КНТ-519				