КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра теорії та технології програмування

Кваліфікаційна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інформатика»
спеціальності 122 «Комп'ютерні науки»
на тему:

ВИКОРИСТАННЯ REST API ТА ТЕХНОЛОГІЇ GEOHASH ДЛЯ РОЗВ'ЯЗАННЯ ПРИКЛАДНИХ ЗАДАЧ

Виконав студент 4-го курсу
Каплюк Владислав Олегович

(підпис)

Науковий керівник:
доцент, кандидат фіз.-мат. наук
Кузенко Володимир Федорович

_____
(підпис)

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент

(підпис)

Роботу розглянуто й допущено до захисту на засіданні кафедри теорії та технології програмування
«_21_» _ТРАВНЯ_____ 2021 р.,
протокол №_____
Завідувач кафедри
М.С. Нікітченко _____

(підпис)

Київ – 2021

РЕФЕРАТ

Обсяг роботи 51 сторінка, 24 ілюстрація, 1 таблиця, 18 джерела посилань.

Об’єктом дослідження є оптимізація та процес пошуку найближчих станцій моніторингу повітря використовуючи технологію Geohash. Предметом роботи є система для вирішення прикладної задачі знаходження найближчих станцій моніторингу якості повітря та відображення інформації про його стан на основі місцезнаходження користувача.

Метою роботи є створення публічного WEB API додатку для отримання актуальної інформації стану повітря на основі місцезнаходження.

Методи розроблення: розробка програмного продукту, архітектурні рішення.

Інструменти розроблення: операційна система - Windows 10, Visual Studio 2019, .NET 5, Postman, MongoDB.

Результати роботи: спроектовано та створено розширюваний масштабований додаток. Підхід до рішення це використання технології Geohash як спосіб пошуку найближчих станцій моніторингу повітря, та REST як підхід по написання публічного API. На основі цього розроблено програмний продукт «AirSnitch API».

За методами розробки та інструментальними засобами робота виконувалася сумісно з пошуком інформації в всесвітній мережі.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

Json – JavaScript object notification.

API – Application Programming Interface.

ПЗ – Програмне забезпечення.

ОС – Операційна Система.

DB – Data Base.

DTO – Data Transfer Object.

REST - Representational State Transfer

GUI – Graphical User Interface.

BFS – Breadth-first search.

ВСТУП

Оцінка сучасного стану об’єкта розробки. У наші дні потреба у навігації набула великого значення. Міста зростають великими темпами, і орієнтуватися становиться все складніше. Зі збільшенням міст збільшується і забруднення повітря внаслідок діяльності різних підприємств, зростаючої кількості автомобілів. Іноді, під час прогулянки на вулиці ми навіть не підозрюємо про те, чи дійсно корисною для нас вона виявиться. Проблемами зменшення забруднення повітря займаються вчені спільноти по всьому світі.

Аналіз якості повітря на основі місцезнаходження об’єкта є прикладною задачею, яка розподіляється на декілька підзадач:

- отримання місцезнаходжень та даних станцій моніторингу якості повітря;
- виконання пошуку найближчої станції до користувача та отримання даних

Розроблений Web API додаток надає публічний REST API до даних місця розташування та вимірів станцій, що дає змогу усім бажаючим інтегрувати знання про якість повітря в свої треті системи.

Актуальність роботи та підстави для її виконання. Забруднення повітря - це глобальна проблема. Воно є причиною передчасної смерті близько 7 мільйонів людей кожного року через гострі респіраторні інфекції, захворювання дихальної, серцево-судинної систем. Забруднення повітря є нагальною проблемою і для України. Відповідно до даних Всесвітньої організації охорони здоров’я, протягом року тут від забруднення повітря передчасно помирає 124 людини зі 100 тис. мешканців, що є вчетверо більше, ніж у п’яти найчистіших країнах планети. За співвідношенням смертності до кількості населення Україна – одна з країн із найбільш забрудненим повітрям зі 160 досліджених.[1] Запропонований мною додаток допоможе значно розширити усвідомленість суспільством даної проблеми та привернути увагу громадськості до цього питання. Будь-яка

стороння система буде мати змогу отримувати дані про якість повітря навколо себе.

Мета й завдання роботи. Метою роботи є розробка додатку для пошуку та отримання інформації щодо якості повітря з найближчих станцій моніторингу.

Для досягнення цієї мети поставлено такі завдання:

- дослідити існуючі на ринку застосунки;
- спроектувати архітектуру застосунку;
- проаналізувати та розробити оптимальний алгоритм пошуку;
- розробити програмний інтерфейс для взаємодії з додатком;

Об'єкт, методи й засоби розроблення. Об’єктом розроблення програмного засобу «AirSnitch API» є процес пошуку найближчих об'єктів. Предметом роботи є система, яка складається з 2-х рівнів: бази даних, сервера додатку.

У якості інструментів створення та тестування програмного засобу було обрано наступні інструменти:

- Visual Studio 2019 - інтегроване середовище розробки мовою програмування C#;
- MongoDB – документо-орієнтована система керування базами даних з відкритим вихідним кодом, яка не потребує опису схеми таблиць.
- Postman – потужний набір інструментів тестування API, зручний у використані командою через можливість використовувати спільні запити;
- Git – розподілена система контроля версій.
- SourceTree – клієнт для git з GUI.

Можливі сфери застосування. Додаток орієнтований на використання різними клієнтами, оскільки надає публічне API для цього. Це може бути мобільний додаток, веб додаток, клієнти у ролі ботів у різних месенджерах.

РОЗДІЛ 1. ОГЛЯД ІСНУЮЧИХ НА РИНКУ СИСТЕМ

SaveEcoBot

На даний момент це є один з перших продуктів, які були випущені на теренах нашої держави. Ця платформа позиціонує себе як єдиний екологічний чат-бот в Україні. Платформа агрегує дані зі станцій моніторингу по всій Україні та відображає це на веб сайті або в чат боті телеграм.

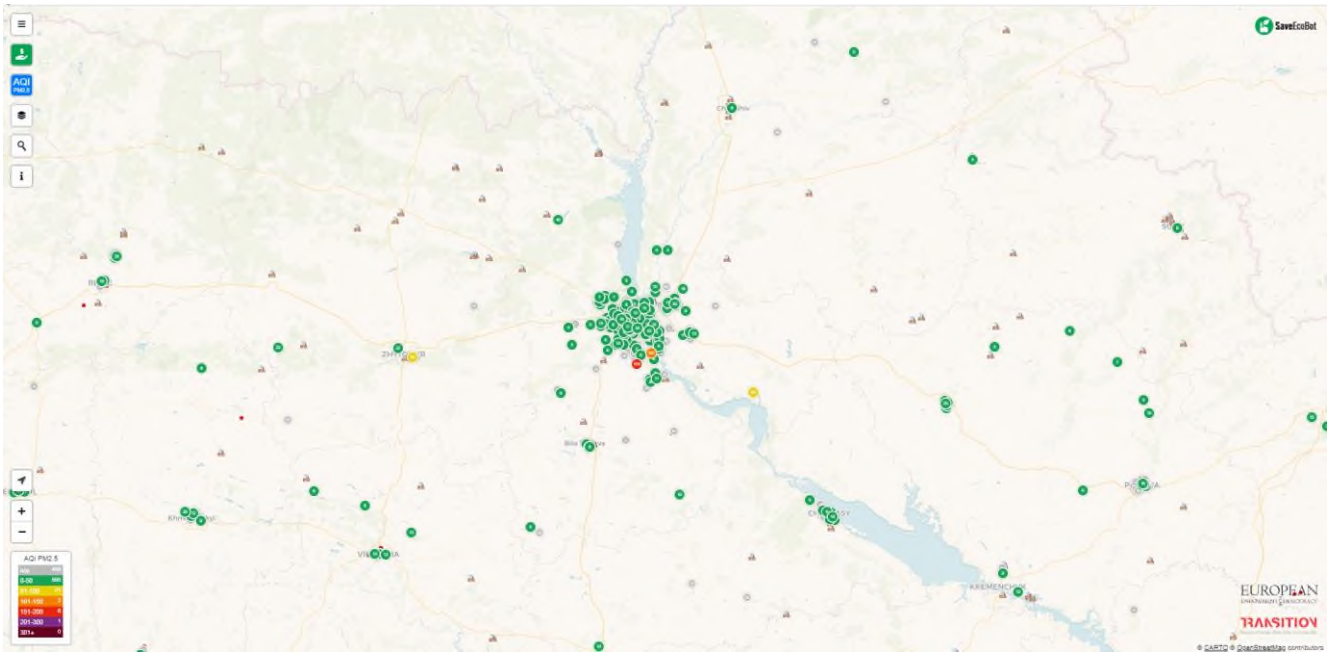

Рисунок 1.1 – Карти SaveEcoBot з мітками станцій.

Головна ціль організації - покращити якість повітря в країнах, що розвиваються, шляхом підвищення рівня екологічної обізнаності громадян шляхом надання їм безкоштовного та легкого доступу до індексу якості повітря та інших публічних екологічних даних. Також кожен охочий у разі виявлення якихось порушень з боку людей, громад чи організацій може подати скаргу на забруднення повітря, води, ґрунту, відходів чи сильного шуму.

luftdaten.info

Німецький ресурс який відображає інформацію щодо забруднень по всій Європі, детальна інформація по кожній зі станцій також позначені на легенді карти. Показники вітру та інші метеорологічні дані присутні на ресурсі. Також хочеться відмітити те, що позначаються представництва та їх контакти з яких беруться виміри рис.1.2.

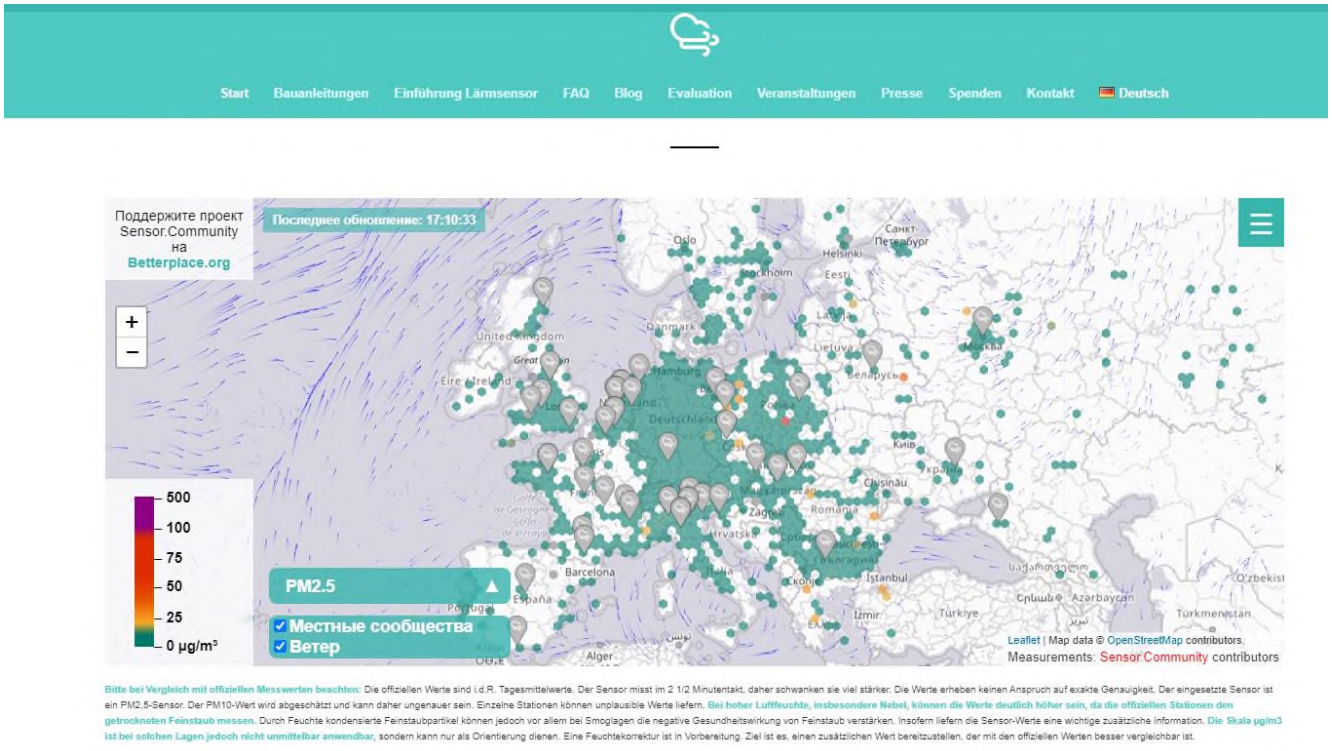

Рисунок 1.2 – Карта luftdaten.info з мітками станцій.

Aqualitybot

Телеграм бот Якість повітря у Києві NOW дозволяє обрати район зі списку та отримувати дані моніторингу є можливість отримування даних автоматично рис.1.3.

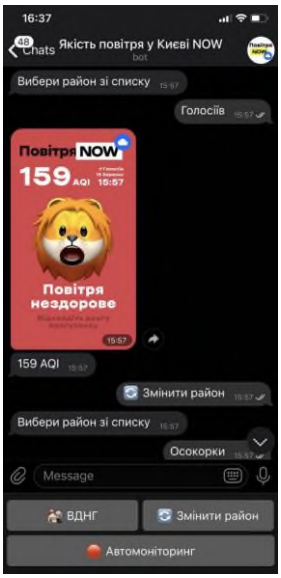

Рисунок 1.3 – інтерфейс телеграм боту Aqualitybot.

Після огляду та тестування кожного з ресурсів стало зрозуміло, що це нагальна тема яка у світі набирає неабиякого значення.

Проте, слід зазначити той факт, що зручного та прозорого публічного API не надають вищезгадані платформи.

РОЗДІЛ 2. ОГЛЯД ВИКОРИСТАНИХ ТЕХНОЛОГІЙ

2.1. Огляд середовища NoSQL

Для створення проекту було використано ОС Windows 10. Провайдером бази даних було обрано MongoDB. MongoDB — документо-орієнтована система керування базами даних (СКБД). MongoDB швидка та масштабована система, яка оперує даними в форматі ключ значення, на відміну від реляційних СКБД, які відрізняються своєю функціональністю та зручністю формування запитів. MongoDB підтримує зберігання документів в JSON-подібному форматі, має досить гнучку мову для формування запитів, може створювати індекси для різних збережених атрибутів, ефективно забезпечує зберігання великих бінарних об'єктів, підтримує журналювання операцій зі зміни і додавання даних в БД, може працювати відповідно до парадигми Map/Reduce, підтримує реплікацію і побудову відмовостійких конфігурацій. У MongoDB є вбудовані засоби із забезпечення шардінгу (розподіл набору даних по серверах на основі певного ключа), комбінуючи який з реплікацією даних можна побудувати горизонтально масштабований кластер зберігання, в якому відсутня єдина точка відмови (збій будь-якого вузла не позначається на роботі БД), підтримується автоматичне відновлення після збою і перенесення навантаження з вузла, який вийшов з ладу. Розширення кластера або перетворення одного сервера на кластер проводиться без зупинки роботи БД простим додаванням нових машин. [3]

Базу даних було розроблено в MongoDB Compass [4]. MongoDB Compass - графічний інтерфейс для MongoDB. Дозволяє візуально досліджувати дані, виконувати спеціальні запити за лічені секунди взаємодіяти зі даними з повною функціональністю CRUD, переглядати та оптимізовувати ефективність запитів. Доступний на Linux, Mac або Windows. Компас дає змогу приймати розумніші рішення щодо індексації, перевірки документації тощо. Також в ньому містяться вбудовані схеми візуалізації.

MongoDB Compass аналізує документи та відображає багаті структури у колекціях за допомогою інтуїтивно зрозумілого графічного інтерфейсу. Це дозволяє швидко візуалізувати та дослідити схему, щоб зрозуміти частоту, типи та діапазони полів у наборах даних.

2.2. Огляд платформи .NET Core

.NET Core - це модульна платформа для розробки програмного забезпечення з відкритим вихідним кодом. Сумісна з такими операційними системами як Windows, Linux і macOS. Була випущена компанією Microsoft. У платформи є власне співтовариство на GitHub. Підтримує наступні мови програмування: C #, Visual Basic .NET (частково) і F # .

.NET Core заснована на .NET Framework. Платформа .NET Core відрізняється від неї модульностью, кроссплатформенную, можливістю застосування хмарних технологій, і тим, що в ній відбувся поділ між бібліотекою CoreFX і середовищем виконання CoreCLR.

.NET Core - модульна платформа. Кожен її компонент оновлюється через менеджер пакетів NuGet, а значить можна оновлювати її модулі окремо, в той час як .NET Framework оновлюється цілком. Кожна програма може працювати з різними модулями і не залежить від єдиного поновлення платформи.

CoreFX - це бібліотека, інтегрована в .NET Core. Серед її компонентів: System.Collections, System.IO, System.Xml.

CoreCLR - це середовище виконання, що включає в себе RyuJIT (JIT-компілятор), вбудований збирач сміття та інші компоненти.[5]

2.3. Огляд протоколу HTTP

HTTP - це протокол , що дозволяє отримувати різні ресурси, наприклад HTML-документи. Протокол HTTP лежить в основі обміну даними в Інтернеті. HTTP є протоколом клієнт-серверного взаємодії, що означає ініціювання запитів до сервера самим одержувачем, зазвичай веб-браузером (web-browser). Отриманий підсумковий документ буде (може) складатися з різних піддокументів які є частиною підсумкового документа: наприклад, з окремо отриманого тексту, опису структури документа, зображень, відео-файлів, скриптів і багато чого іншого (рис.2.1).

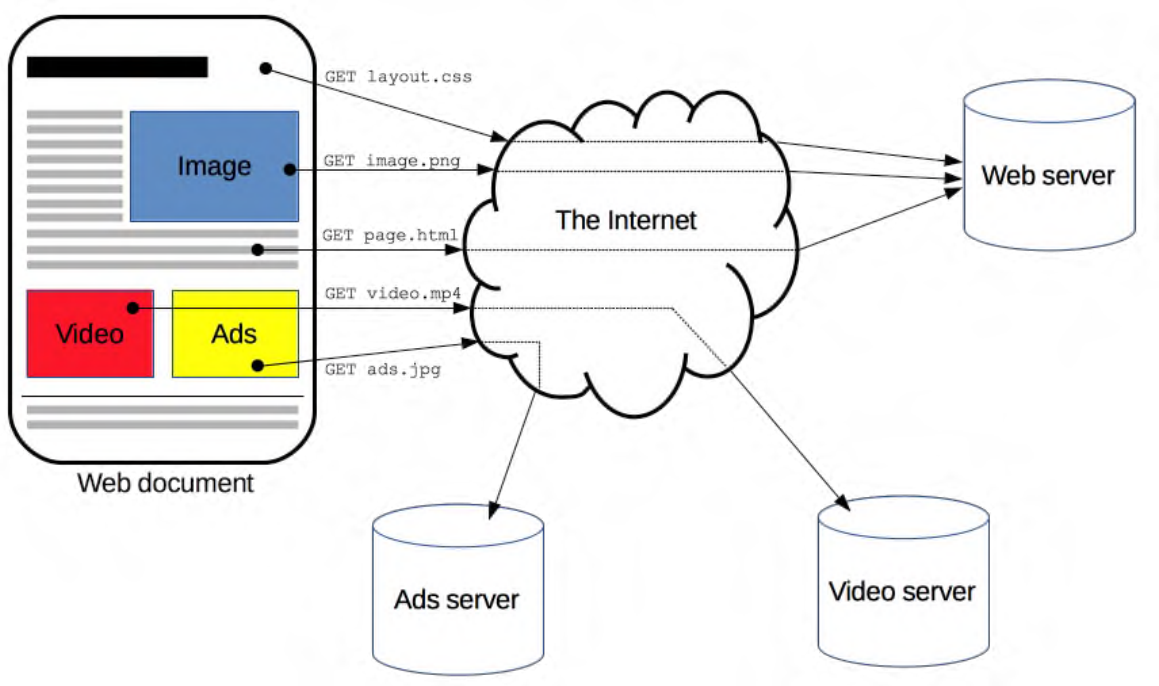

Рисунок 2.1 – Схема використання HTTP

Клієнти і сервери взаємодіють, обмінюючись поодинокими повідомленнями (а не потоком даних). Повідомлення, відправлені клієнтом, зазвичай веб-браузером, називаються запитами , а повідомлення, відправлені сервером, називаються відповідями.

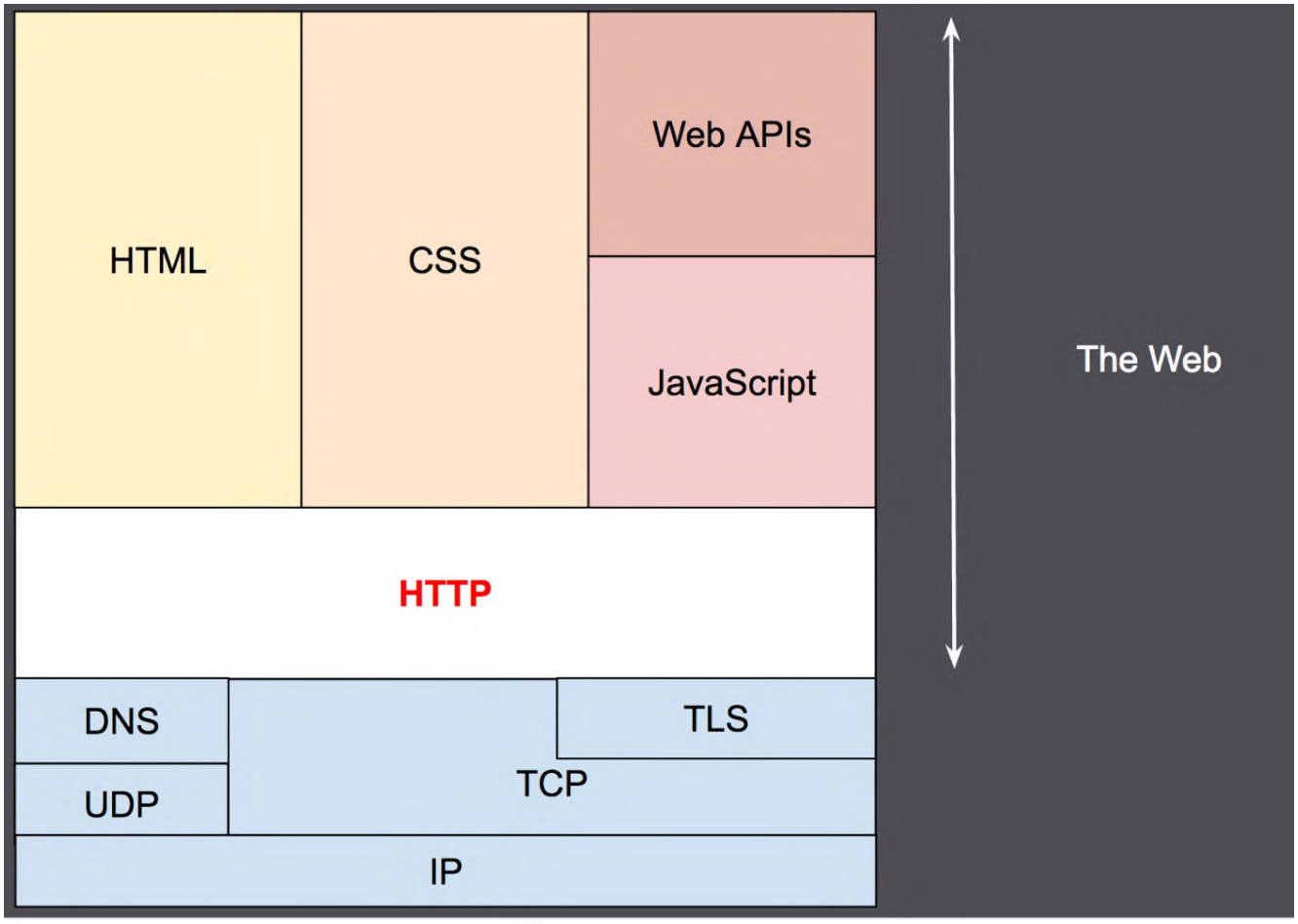

Рисунок 2.2 – Схема використання HTTP

Хоча HTTP був розроблений ще на початку 1990-х років, за рахунок своєї розширюваності надалі він весь час удосконалювався. HTTP є протоколом прикладного рівня, який найчастіше використовує можливості іншого протоколу – TCP або TLS захищений TCP (рис.2.2) - для пересилання своїх повідомлень, проте будь-який інший надійний транспортний протокол теоретично може бути використаний для доставки таких повідомлень. Завдяки своїй розширюваності, він використовується не тільки для отримання клієнтом гіпертекстових документів, зображень і відео, але і для передачі вмісту серверів, наприклад, за допомогою HTML-форм. HTTP також може бути використаний для отримання тільки частин документа з метою поновлення веб-сторінки за запитом (наприклад за допомогою AJAX запиту).[18]

2.3. Огляд архітектури REST

REST - це набір архітектурних обмежень, архітектурний стиль, який був визначений Роєм Філдінгом в своїй докторській дисертації 2000 року. API - це набір визначень та протоколів для побудови та інтеграції прикладного програмного забезпечення. Іноді це називають контрактом між постачальником інформації та користувачем інформації - встановлення контенту, що вимагається від споживача (дзвінок), та контенту, який вимагає виробник (відповідь).Розробники API можуть реалізувати REST різними способами. Для того, щоб API вважався RESTful, він повинен відповідати цим критеріям:

- Клієнт-серверна архітектура, що складається з клієнтів, серверів та ресурсів, із запитами, керованими через HTTP.
- Зв'язок клієнт-сервер без збереження стану, тобто інформація про клієнта не зберігається між запитами отримання, і кожен запит є окремим і не пов'язаним.
- Кешовані дані, що впорядковують взаємодію клієнт-сервер.
- Єдиний інтерфейс між компонентами, завдяки чому інформація передається у стандартній формі. Для цього потрібно:
    - запитувані ресурси можна ідентифікувати та окремо від подань, що надсилаються клієнту;
    - клієнт може маніпулювати ресурсами через представлення, яке він отримує, оскільки представлення містить достатньо інформації для цього;
    - самоописові повідомлення, що повертаються клієнту, мають достатньо інформації, щоб описати, як клієнт повинен це обробити;
    - доступний гіпертекст / гіпермедіа, що означає, що після звернення до ресурсу клієнт повинен мати можливість використовувати гіперпосилання для пошуку всіх інших доступних на даний момент дій, які він може зробити.

Незважаючи на те, що REST API має ці критерії, він все ще вважається простішим у використанні, ніж встановлений протокол, такий як SOAP (Простий протокол доступу до об'єктів), який має конкретні вимоги, такі як обмін повідомленнями XML, та вбудовану безпеку та відповідність транзакціям, які роблять це повільніше і важче. На відміну від цього, REST - це набір керівних принципів, які можна впровадити за потреби, роблячи REST-API швидшими та легшими, з підвищеною масштабованістю - ідеально підходить для Інтернету речей (IoT) та розробки мобільних додатків.[16]

2.3. Огляд фреймворку ASP.NET Core

ASP.NET Core — вільне та відкрите програмне забезпечення каркаса вебзастосунків, з продуктивністю вищою ніж у ASP.NET, розроблена корпорацією Microsoft. Це модульна структура, яка працює як на повній платформі .NET Framework, так і на платформі .NET Core. Фреймворк являє собою повний перепис, який об'єднує раніше окремі ASP.NET MVC та ASP.NET Web API у єдину програмувальну модель. Не зважаючи на те, що це є новим фреймворком, побудованим на новому веб-стеку, ASP.NET Core має високий ступінь сумісності концепцій з ASP.NET MVC, який об'єднує функціональність MVC, Web API та Web Pages. В попередніх версіях платформи дані технології реалізовані окремо і тому містять багато дублювальної функціональності. Тепер це об'єднано в одну програмну модель ASP.NET Core MVC. Веб-форми повністю вийшли в минуле. Програми ASP.NET Core підтримують програмні версії, в якій різні програми, що працюють на одному комп'ютері, можуть орієнтуватися на різні версії ASP.NET Core. Це не можливо з попередніми версіями ASP.NET Core.[6]

2.5. Огляд технології Git

Git – на сьогоднішній день одна з найбільш популярних розподілених систем контроля версій. Система контроля версій – програмне забезпечення для полегшення роботи зі змінною інформацією. Система контроля версій дозволяє зберігати кілька версій одного документу та при необхідності повертатися до попередніх версій та визначати, хто зробив останні зміни у файлу. Дуже корисний для розробки ПЗ в командах для >1 чоловік.[8]

SourceTree – десктопний клієнт для git. Використовується для візуалізації і керування репозиторієм git.[9]

GitHub – один з найкрупніших веб-сервісов для хостингу проектів та спільної розробки. Веб-клієнт для git.[10]

GitFlow – строга модель розгалуження(створення гілок), у центрі якої знаходиться реліз. В основі моделі знаходяться наступні гілки: master, develop, features release та hotfix. Цикл розробки наступний: основна розробка ПО виконується у гілці develop, для кожної частини функціоналу(кожної feature) створюється окрема гілка з develop та по закінченню роботи відбувається merge у develop. Коли наближається дата релізу, з девелопа створюється гілка release, у якій новий функціонал все не додається, лише виправляються помилки. Під час релізу відбувається merge гілки release у гілку master та у гілку develop. Гілка release видаляється. Ще існує гілка hotfix, яка потрібна для швидкого виправлення помилок. Вона створюється напряму з master, та по закінченню роботи відбувається merge у master.[11]

2.6. Огляд Onion Architecture

Onion-архітектура являє собою поділ програми на рівні. При чому є один незалежний рівень, який знаходиться в центрі архітектури. Від цього рівня залежить другий рівень, від другого - третій і так далі. Тобто виходить, що

навколо першого незалежного рівня нашаровується другий-залежний. Навколо другого нашаровується третій, який також може залежати і від першого. Абстрактно це може бути виражено у вигляді цибулі, в якій також є серцевина, навколо якої нашаровуються всі інші верстви, аж до лушпиння. Термін "Цибулева архітектура" був запропонований Джеффрі Палермо (Джеффрі Палермо) ще в 2008 році.

Кількість рівнів може відрізнятися, але в центрі завжди знаходиться модель домену (Domain Model), тобто ті класи моделей, які використовуються в додатку і об'єкти яких зберігаються в базі даних. Схема аріхітектури такого додатку зображена на рис. 2.3

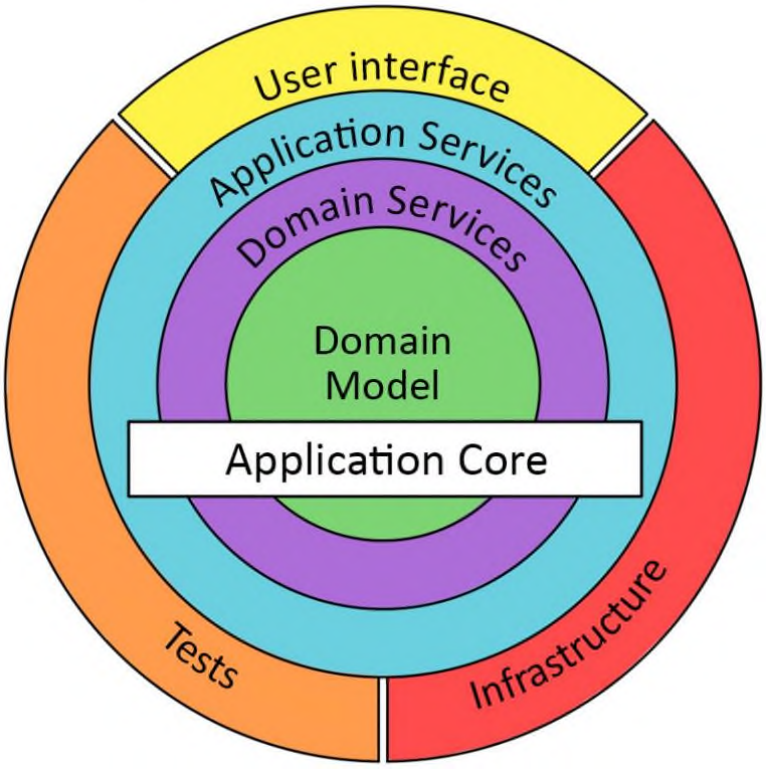

Рисунок 2.3 – Схема Onion architecture.

Перший рівень навколо моделі домену утворюють інтерфейси, які керують роботою з моделлю домену. Зазвичай це інтерфейси репозиторіїв, через які ми взаємодіємо з базою даних.

Зовнішній рівень являє такі компоненти, які дуже часто змінюються. Зазвичай зовнішній рівень утворюють призначений для користувача інтерфейс, тести, якісь допоміжні класи інфраструктури додатки. До цього рівня також належать конкретні реалізації інтерфейсів, оголошених на нижчих рівнях. Наприклад, реалізація інтерфейсу сховища, який оголошений на рівні Domain Services. Взагалі всі внутрішні рівні, які можна об'єднати в Application Core, визначають тільки інтерфейси, а конкретна реалізація цих інтерфейсів розташовується на зовнішньому рівні.

Також варто відзначити, що всі зовнішні сховища, як бази даних, файли, зовнішні веб-сервіси, від яких ми можемо отримувати дані - все це є зовнішнім по відношенню до архітектури.[13]

2.8. Огляд DDD

Предметно-орієнтоване проєктування (рідше проблемно-орієнтоване, англ. domain-driven design, DDD) - це підхід до моделювання складного об'єктно-орієнтованого програмного забезпечення. Переваги DDD полягають в наступному:

- Концентрація основної уваги на предметній області;
- Створення програмних моделей, які відображують глибоке розуміння предметної області.

Термін був вперше запроваджений Еріком Евансом в своїй книзі з однойменною назвою Domain Driven Design [17]

Основні визначення :

1. Домен: Предметна область, середовище, галузь. Предметна область, яку програміст використовує при створенні програмного забезпечення.
2. Модель: Система абстракцій, яка описує окремі аспекти предметної області.

3. Загальна мова: Мова побудована навколо моделі предметної області. Використовується як програмістами при написанні програмного забезпечення, так і іншими членами команди (експертами обраної галузі).
4. Контекст: Середовище, в якому предмет або дія означує своє значення.

2.9. Огляд автентифікації та авторизації API Key

Сьогодні основними типами автентифікації виступають JWT Bearer, OAuth, OAuth2 рідше можна зустріти Basic Auth та API Key. Проте, усі ці типи окрім останнього використовують в собі токени з дуже малим періодом життя, останній ж генерується раз і дається на користування на довгий час. Ключі API призначені для проектів, стандарта автентифікація - для користувачів. [9] Основна відмінність між цими двома:

- Ключі API ідентифікують проект, що викликає - програму чи сайт - що здійснює виклик API.
- Маркери (токени) автентифікації ідентифікують користувача - особу, яка використовує додаток або сайт.

Ключі API надають:

- Ідентифікацію проекту – можливість визначати програму або проект, який викликає цей API.
- Авторизація проекту – можливість перевірки, чи надано програмі, що викликає, доступ до виклику API.

Коли використовувати ключі API

API може обмежити деякі або всі свої методи вимагати ключів API. Це має сенс зробити, якщо:

- Ви хочете заблокувати анонімний трафік. Ключі API визначають трафік програми для виробника API на випадок, якщо розробник програми

повинен співпрацювати з виробником API, щоб налагодити проблему або показати використання програми.

- Ви хочете контролювати кількість дзвінків, здійснених на ваш API.
- Ви хочете визначити схеми використання у трафіку вашого API. Ви можете бачити використання додатків в API та послугах .
- Ви хочете фільтрувати журнали подій за ключем API.

Ключі API не можна використовувати для:

- Ідентифікація окремих користувачів - ключі API не ідентифікують користувачів, вони ідентифікують проекти.
- Безпечне авторизація.

2.10. Огляд ситеми геокодування Geohash

Geohash - система геокодування з відкритим доступом, винайдена у 2008 році Густаво Німейєром та (аналогічна робота в 1966 р.) Г.М. Мортон. Геохеш кодує географічне місце у короткий рядок літер та цифр. Це ієрархічна структура просторових даних, яка розділяє простір на сітку, що є одним із багатьох застосувань, що називається кривою Z-порядку, і взагалі кривими заповнення простору.

Основна частина алгоритму Geohash та перша ініціатива подібного рішення були задокументовані у звіті Г.М. Мортон в 1966 р. "A Computer Oriented Geodetic Data Base and a New Technique in File Sequencing". Робота Мортона була використана для ефективної реалізації кривої Z-порядку.[12]

geohash length	lat bits	lng bits	lat error	lng error	km error
1	2	3	±23	±23	±2500
2	5	5	±2.8	±5.6	±630
3	7	8	±0.70	±0.70	±78
4	10	10	±0.087	±0.18	±20
5	12	13	±0.022	±0.022	±2.4
6	15	15	±0.0027	±0.0055	±0.61
7	17	18	±0.00068	±0.00068	±0.076
8	20	20	±0.000085	±0.00017	±0.019

Рисунок 2.4 – Точність від кількості використаних бітів при кодуванні в Base32

2.11. Огляд кривої Мортона

У математичному аналізі та інформатиці крива Мортона, Z-порядок - це функція, яка показує багатомірні дані в одновимірні, зберігаючи локальність точок даних. Функція була введена в 1966 році Гаем Макдональдом Мортоном . Z-значення точок у багатовимірному просторі легко вибирає чередовані двомісні цифри свого координатного значення. Коли дані заповнюються у цьому порядку, вони можуть бути використані будь-якими однорідними структурами, такими як двійкові пошуки по дереву, B-дерева, списки з пропусками або хеш-таблицями (з відхиленням молодших бітів). Створений таким чином порядок можна еквівалентно описати як порядок, який можна отримати обходом в глибину дерева квадрантів.[15]

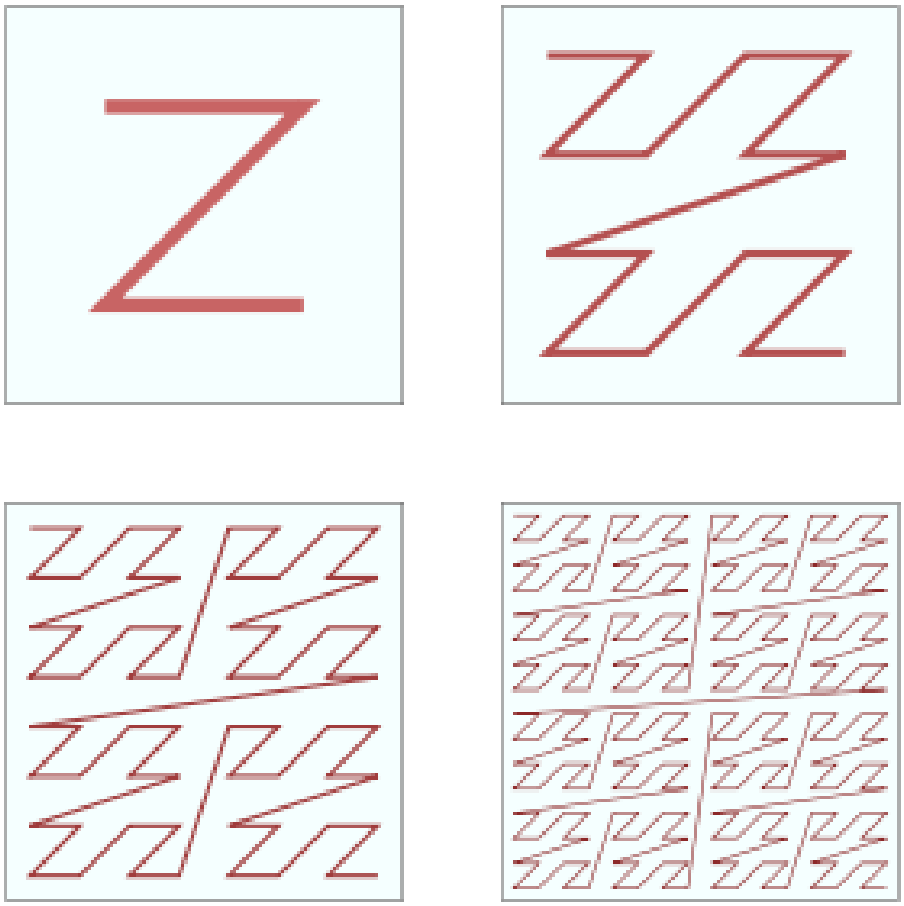

Рисунок 2.5 – Чотири ітерації кривої Мортона

РОЗДІЛ 3. ПРИЗНАЧЕННЯ ТА ЦІЛІ СТВОРЕННЯ СИСТЕМИ. ВИМОГИ ДО СИСТЕМИ

3.1. Призначення системи

Система призначена для використання іншими додатками, щоб отримувати дані про забруднення повітря відповідно до координат місцезнаходження об’єкта, знаходити інформацію про станції моніторингу та виміри безпосередньо по конкретним станціям.

3.2. Цілі створення системи

Систему було створено з метою надання повного публічного REST API для доступу до інформації про стан повітря.

3.3. Вимоги до системи в цілому

Система повинна мати масштабовану архітектуру, має бути логічно структурованою та однозначною.

У системі повинні бути передбачені наступні функціональні можливості:

- авторизація та автентифікація користувачів;
- логування подій у системі;
- отримання даних щодо якості повітря по координатам;
- отримання інформації щодо станцій моніторингу;
- реалізація RESTful API (Клієнт, що має доступ до REST сервісу, повинен отримати всі наявні сервіси та методи за допомогою гіперпосилань);
- реалізація resource expansion;

3.4. Вимоги до функцій, які виконуються системою

Функції, виконувані системою, мають бути простими у редагуванні для випадків змін або розширення функціоналу.

У табл. 1 наведено перелік функцій та задач, які підтримуються в даній версії системи.

Таблиця 1 - Перелік функцій та задач, які підтримуються в даній версії системи.

Функція	**Задача**
Робота з користувачами	Отримання користувачів (лише для користувачів у ролі адміністратора)

	Отримання пов’язаної станції з користувачем
	Отримання пов’язаного міста з користувачем
	Отримання пов’язаного провайдеру даних
	Отримання забруднення повітря що заміряється пов’язаною станцією
Робота з станціями моніторингу	Отримання станцій
	Отримання пов’язаних міст
	Отримання провайдерів даних
	Отримання актуальних вимірів забруднення по станції
Робота з моніторингом забруднення	Отримання даних про забруднення відповідно до місцерозташування
Робота з містами	Отримання всіх міст
Робота з провайдерами даних	Отримання всіх провайдерів даних
Робота з клієнтами API	Створити API ключ
	Видалити API ключ

3.5. Технічні вимоги

1. Application Server(.NET Core) підтримує наступні ОС: Windows, Linux, macOS. На машині повинен бути встановлений dotnet.exe.
2. DB Server(MongoDB) підтримує наступні ОС: Windows, Linux, macOS. На машині повинен бути встановлений MongoDB або можливе застосування хмарного сховища.

РОЗДІЛ 4. РЕАЛІЗАЦІЯ СИСТЕМИ

Під час визначення типу архітектури для сервера вибір зупинився на Onion architecture. Для розробки був обраний підхід DDD, предметною областю, яка має бути чітко визначена в цьому підході, став моніторинг якості повітря. Керуючись таким підходом, необхідно було спроектувати рішення, яке б містило в собі 3 абстрактних рівні: рівень ядра, у якому описаний весь домен та відповідні usecases, рівень інфраструктури, у якому реалізовані інтерфейси та логіка, що описана в ядрі, та рівень представлення API – зручний публічний доступ до ресурсів, обкладений архітектурними обмеженнями REST. Фінальна архітектура складається з 3 рівнів (рис.4.1):

- AirSnitch.Core
- AirSnitch.Infrastructure
- AirSnitch.Api

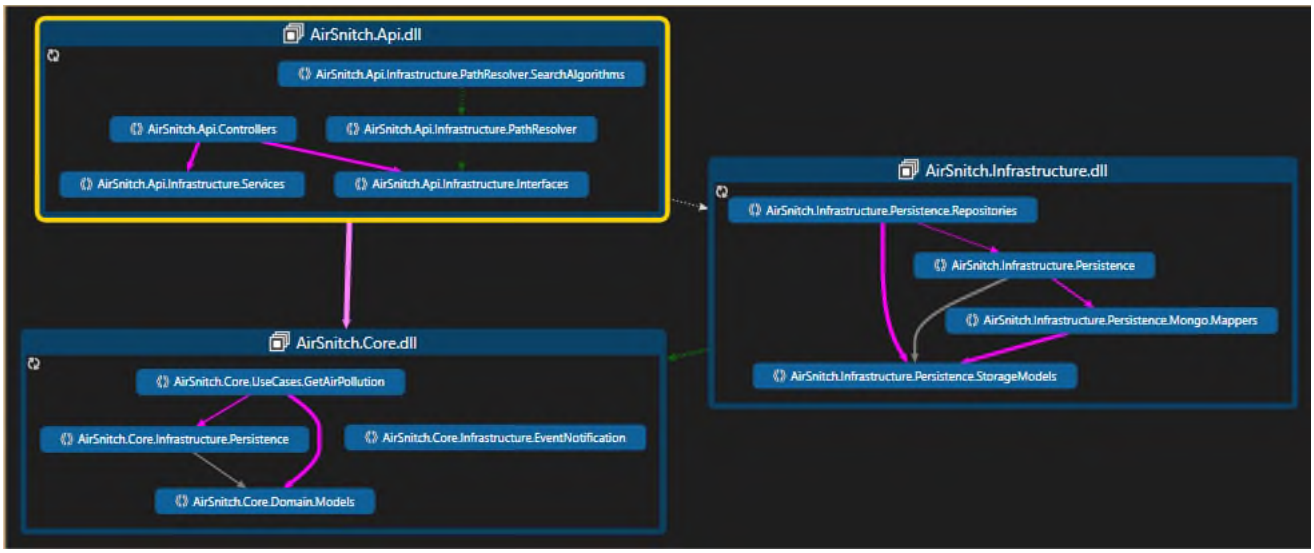

Рисунок 4.1 – кодова карта архітектури системи

4.1. Реалізація рівня ядра сервера

AirSnitch.Core (рис. 4.2) містить у собі всі інтерфейси репозиторіїв, доменні моделі та, відповідно, UseCase – вони виступають у ролі сервісів обмежених контекстом моделі, також міститься інтерфейс провайдерів даних. Оскільки виміри якості повітря беруться ззовні, необхідно було створити механізм, що забезпечив би легке розширення провайдерів даних. Закладений механізм подій, який допоможе легко розширювати додаток шляхом підписки на доменні події, без змін безпосередньо внесених в ядро системи.

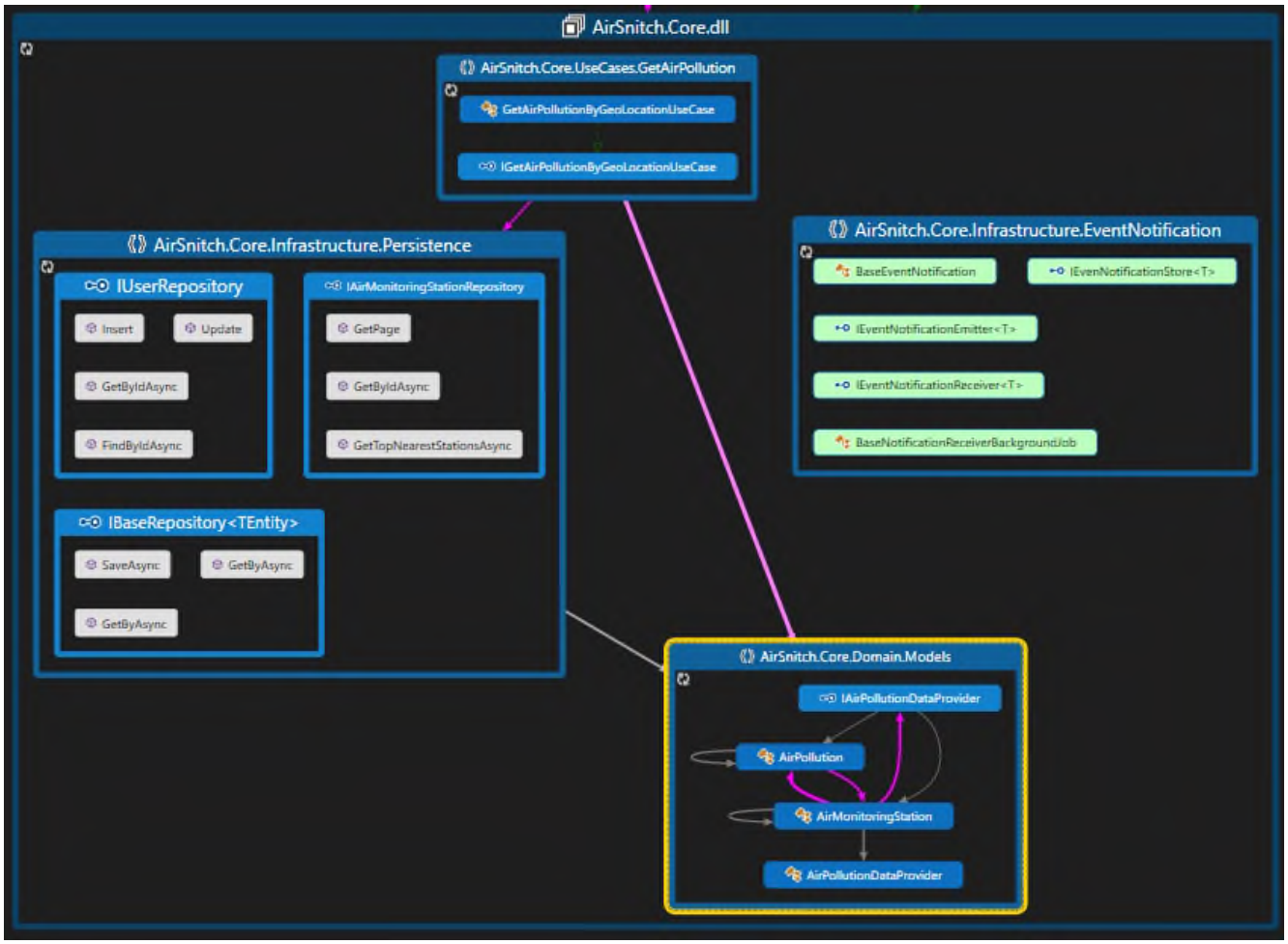

Рисунок 4.2 – кодова карта AirSnitch.Core

4.2. Реалізація рівня інфраструктури сервера

AirSnitch.Infrastructure (рис.4.3) містить у собі безпосередньо реалізацію персистентності відповідно до імплементації за контрактом з ядра доменної області (AirSnitch.Core). Тут визначається рівень з'єднання з базою даних, реалізація репозиторіїв доменних моделей, інтерфейси яких описані в ядрі, розроблені моделі сховища MongoDB та їх мапери в моделі домену. Додатково налаштована робота з логером та конфігурація для http клієнта, який буде використовуватись у провайдерах даних.

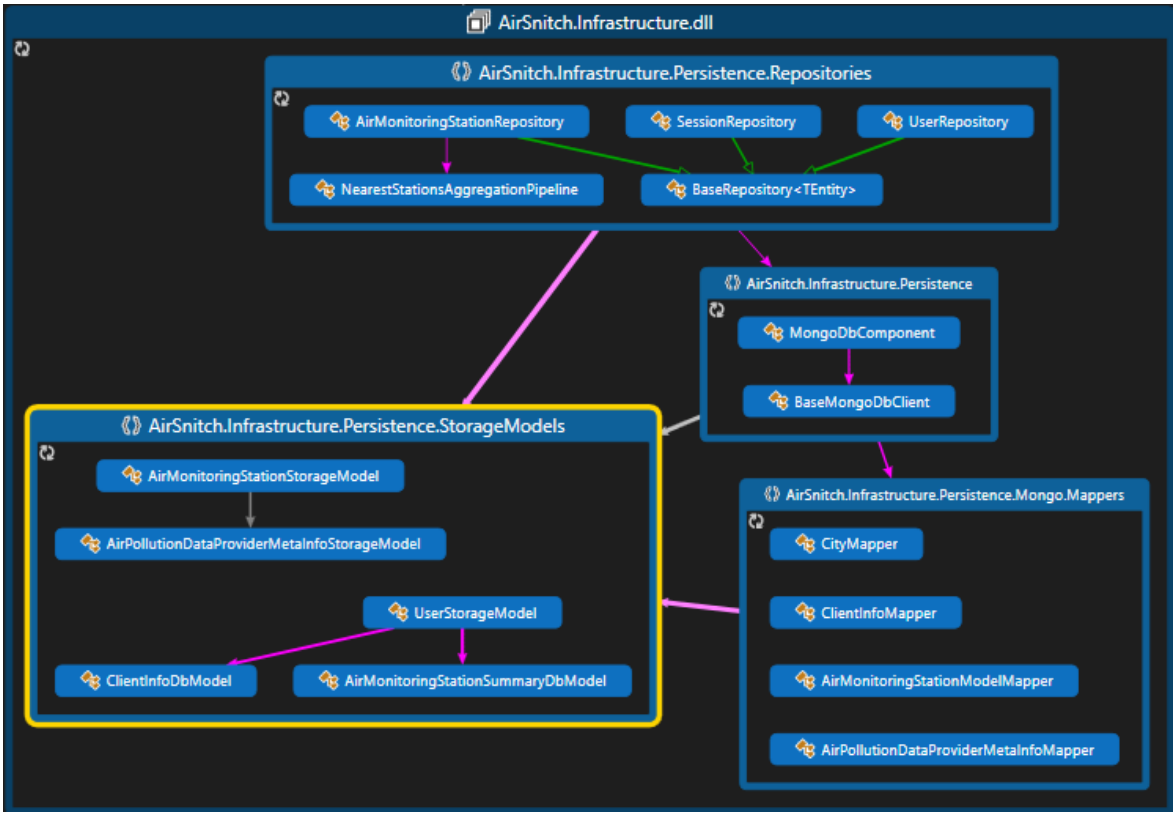

Рисунок 4.3 – кодова карта AirSnitch.Infrustructure

На етапі аналізу був обраний наступний підхід для вирішення проблеми пошуку найближчої станції серед уже існуючих в базі даних:

- зберігаємо геохеш у найкращій точності (максимальна точність – 52-бітне ціле число, що приблизно дорівнює радіусу пошуку в 0,5971 метра) в

таблиці станцій. У таблиці додатково побудований некластерний індекс по цій колонці, який значно пришвидшить швидкість виконання запитів;

- під час запиту на певний радіус за точкою або адресою буде відбуватись пошук сусідів за допомогою геохешу. Відповідно до радіусу поточні координати людини будуть перетворюватись у геохеш із певною точністю. Таким чином, ми користуємось однією з найбільш важливих властивостей геохешу: відсікаючи останні біти в числі, ми зменшуємо точність, тобто збільшуємо радіус пошуку;
- знаходимо сусідів цієї клітинки в такій самій точності та створюємо таким чином межі для пошуку;
- тепер межі для пошуку потрібно повернути в максимальну точність;
- далі робимо запит у базу на вибір значень геохешу в певних межах, які ми отримаємо з попереднього кроку;
- останній крок повернення знайдених елементів.

Клас GeohashHelper відповідальний за взаємодії з геохешем. У його основі закладені такі функції :

- отримання геохешу з певною точністю в бітах з координат по довготі та широті;
- знаходження координат, які обмежують даний геохеш;
- знаходження північного сусіда за геохешем або координатами;
- знаходження південного сусіда за геохешем або координатами;
- знаходження західного сусіда за геохешем або координатами;
- знаходження східного сусіда за геохешем або координатами;
- знаходження південно-східного сусіда за геохешем або координатами;
- знаходження південно-західного сусіда за геохешем або координатами;
- знаходження північно-західного сусіда за геохешем або координатами;
- знаходження північно-східного сусіда за геохешем або координатами.

4.3. Реалізація рівня представлення сервера

AirSnitch.Api (рис.4.4) зовнішній рівень у термінах Onion architecture – рівень презентації додатку. Тут зосереджена логіка побудови посилань, яка була необхідна для вирішення обмеження HATEOAS. Контролери та їх сервіси для взаємодії зі внутрішніми рівнями системи.

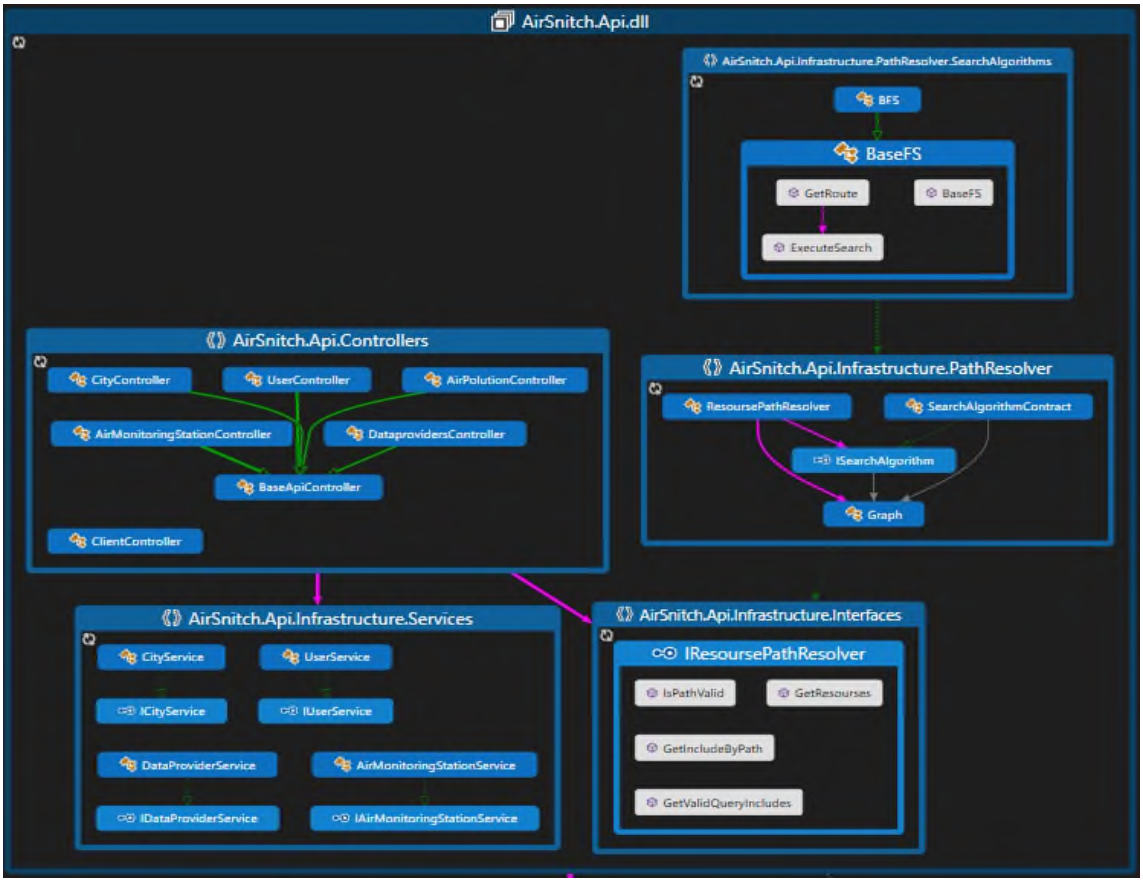

Рисунок 4.4 – кодова карта AirSnitch.Api

Також тут реалізована автентифікація та авторизація на основі API key (рис.4.5). Був обраний саме такий тип автентифікації та авторизації, оскільки за мету були взяті наступні критерії:

- Обмеження анонімного трафіку до ресурсу, вміння в майбутньому визначити схеми використання у трафіку API, наприклад, якщо розробник

програми повинен співпрацювати зі мною, щоб вирішити проблему або показати використання програми;

- Контролювання кількості викликів, здійснених на API;
- Бажання визначити схеми використання трафіку API;
- Бажання фільтрувати журнали подій за ключем API.

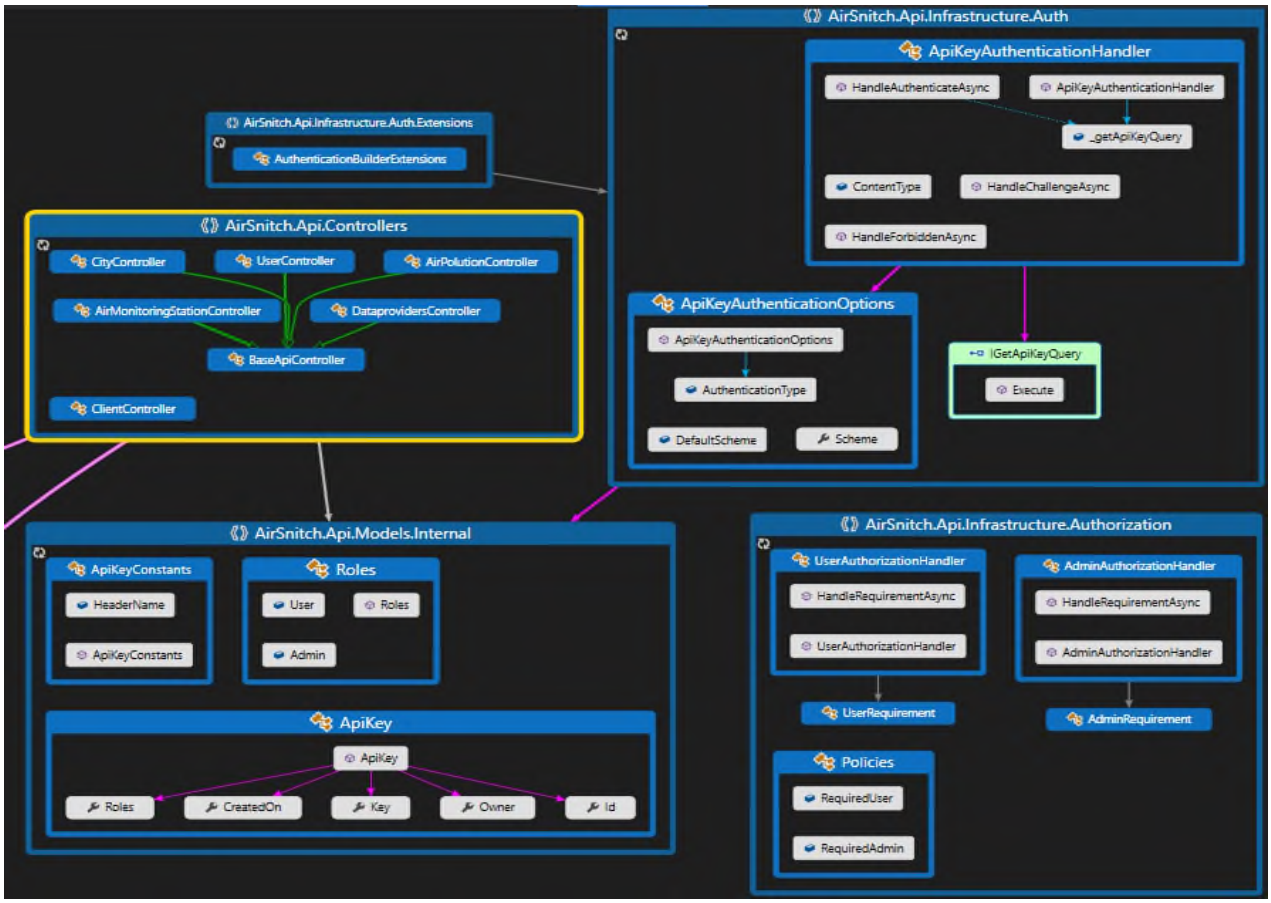

Рисунок 4.5 – кодова карта модулів авторизації та автентифікації

Для вирішення проблеми побудови пов'язаних ресурсів було використане представлення ресурсів у вигляді графа (рис.4.6).

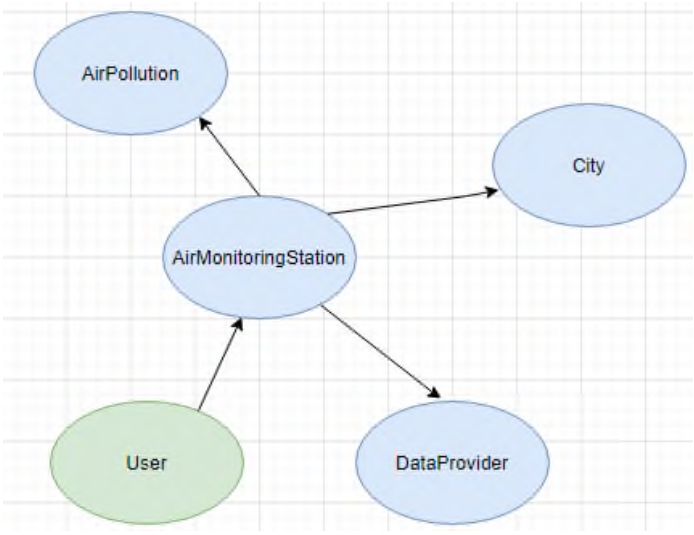

Рисунок 4.6 – граф ресурсів у системі

Головна ідея такого підходу – це одне універсальне розширюване рішення задля того, щоб задовольнити вимоги обмежень, які накладаються архітектурним стилем доступу до ресурсів REST. Було реалізовано програмні компоненти, які дозволяють розширювати дану модель досить легко. Атрибут IncludeResourse (рис. 4.7) відповідає за те, які ресурси потрібно включати до даного, використовується як додаткова мета інформація на контролерах ресурсів.

```
17  namespace AirSnitch.Api.Controllers
18  {
19      [Authorize(Policy = Policies.RequiredUser)]
20      [ApiController]
21      [IncludeResourse(ControllersRoutes.City)]
22      [IncludeResourse(ControllersRoutes.Dataprovider)]
23      [IncludeResourse(ControllersRoutes.AirPolution)]
24      [Route(ControllersRoutes.AirmonitoringStation)]
        1 reference | Каплюк Владислав Олегович, 19 hours ago | 1 author, 7 changes
25      public class AirMonitoringStationController : BaseApiController
26      {
27          private readonly IAirMonitoringStationService _airMonitoringStationService;
            0 references | Каплюк Владислав Олегович, 23 hours ago | 1 author, 1 change
28          public AirMonitoringStationController(IResoursePathResolver resoursePathResolver,
29              IAirMonitoringStationService airMonitoringStationService) : base(resoursePathResolver)
30          {
31              _airMonitoringStationService = airMonitoringStationService;
32          }
```

Рисунок 4.7 – Приклад використання атрибуту IncludeResourse.

Ідея метода полягає в тому, що під час старту системи будується граф ресурсів на основі додаткової метаінформації, представленої у контролерах

ресурсів. Цей граф вводиться за допомогою Dependency Injection у ResoursePathResolver разом з алгоритмом обходу ресурсів в ширину BFS. Далі він використовується для отримання пов’язаних ресурсів у відповідях. Контролери, які використовують ресурси, наслідувані від абстрактного класу BaseApiController. У ньому визначені методи для створення відповідей з ресурсами та пагінацією. Таким чином, якщо потрібно додати новий ресурс та пов’язати його з іншими, достатньо додати зв’язок з іншим через IncludeResource атрибут та наслідувати контролер від BaseApiController.

Для отримання даних моніторингу був написаний провайдер даних. Використовується інформація, надана організацією SaveDnipro, яку потрібно було дістати з мапи, відслідкувавши запити, які надходили в браузері для відображення значень з карти, був знайдений запит, який дозволяє діставати конкретну станцію. Дані щодо станцій були зібрані з публічного API SaveDnipro у вигляді дуже великого json за весь період, із якого і вибиралась інформація. Для першого знайомства з API був доданий опис сервісів із використанням Swagger. Для цього потрібно було додатково налаштувати його в файлі Startup.cs та імпортувати відповідний Nuget package.

РОЗДІЛ 5. ІНСТРУКЦІЯ КОРИСТУВАЧА

Коли користувач переходить за адресою сервера його перенаправляє за замовчуванням на сторінку зі Swagger (рис.5.1), де описані доступні запити, також тут можливо спробувати їх виконати.

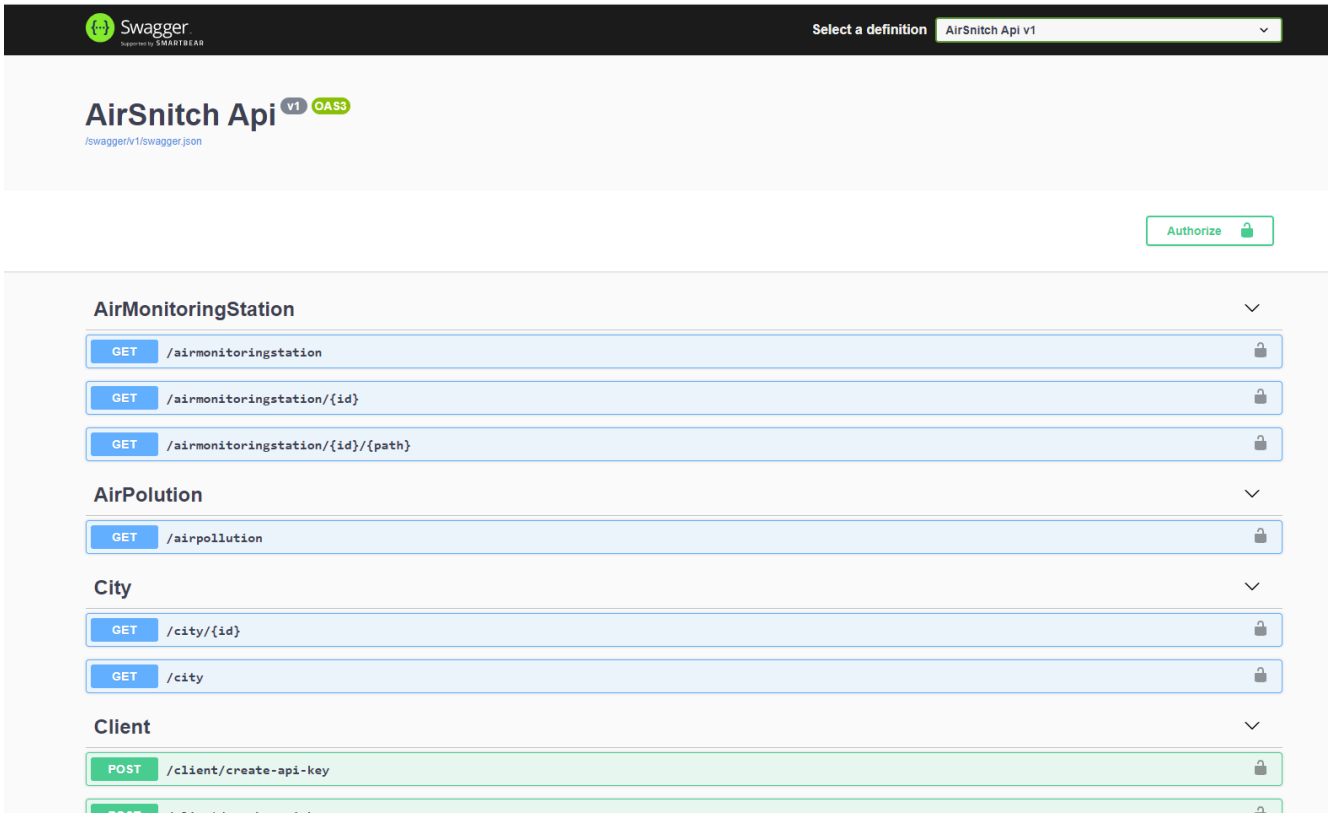

Рисунок 5.1 – Головна сторінка

Якщо розгорнути GET /airmonitoringstation (рис.5.2), можна побачити опис схеми, яка повертається, параметри запиту, такі як limit для обмеження вибірки, та offset для задання зміщення. За замовчуванням, якщо ці параметри не будуть указані, то замість limit буде підставлятися 10, а замість offset – 0. Аналогічно з іншими запитами на кожній вкладинці з них описана схема відповіді та параметри, які вони приймають.

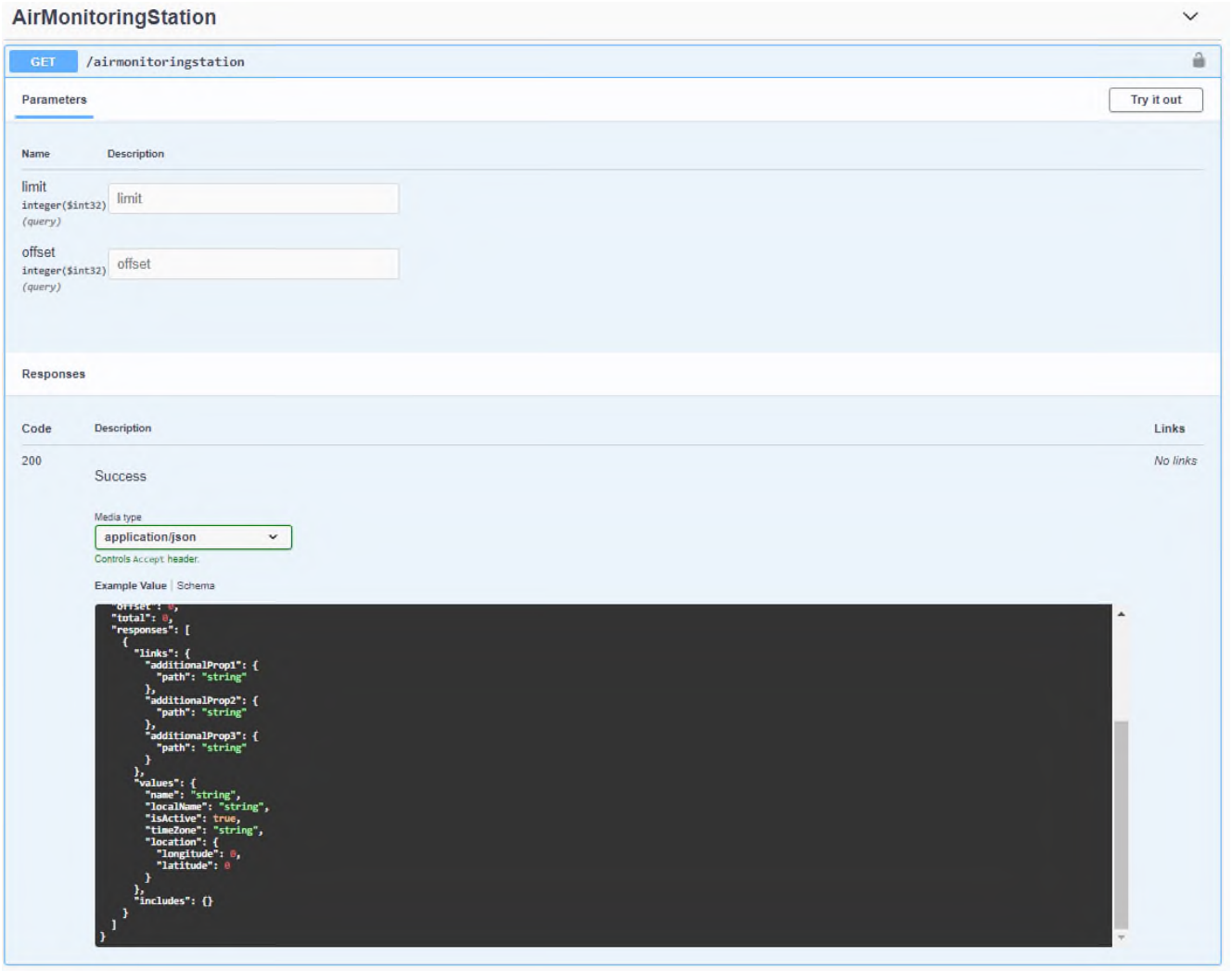

Рисунок 5.2 – вкладинка GET /airmonitoringstation

Якщо виконати запит, повернеться повідомлення із кодом 401 (рис. 5.3.), що означає «ми не авторизовані».

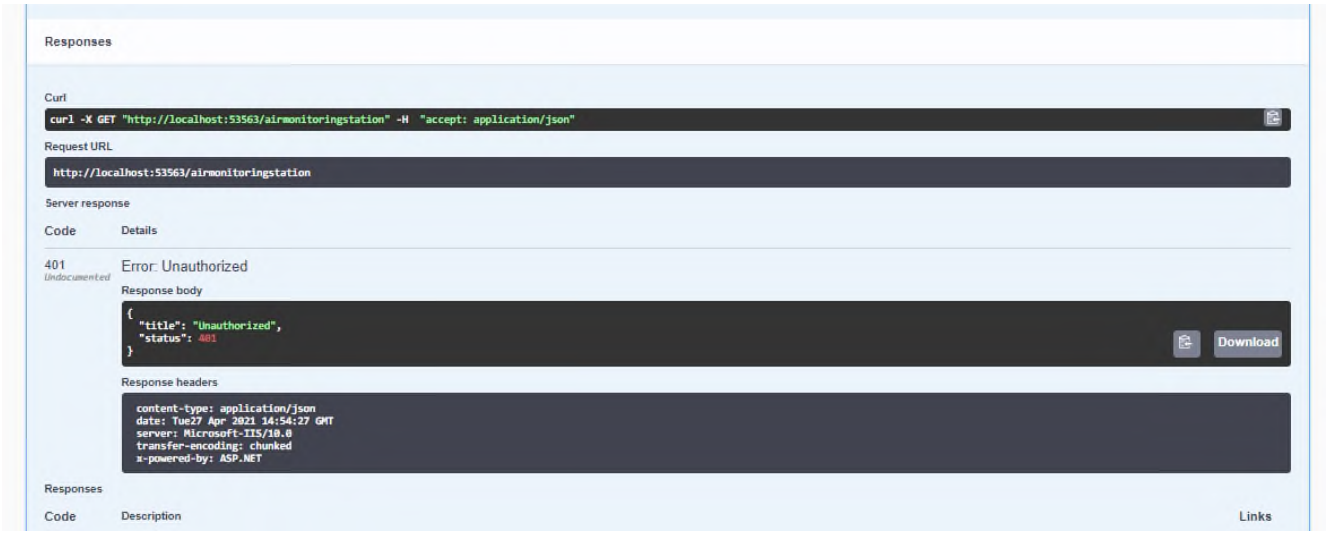

Рисунок 5.3 – результат GET /airmonitoringstation без авторизації

Для того, щоб авторизуватись, необхідно натиснути на кнопку Authorize, яка зобажена на рис.5.1, та ввести API ключ. Для отримання API key звернемось до POST /client/create-api-key (рис. 5.4).

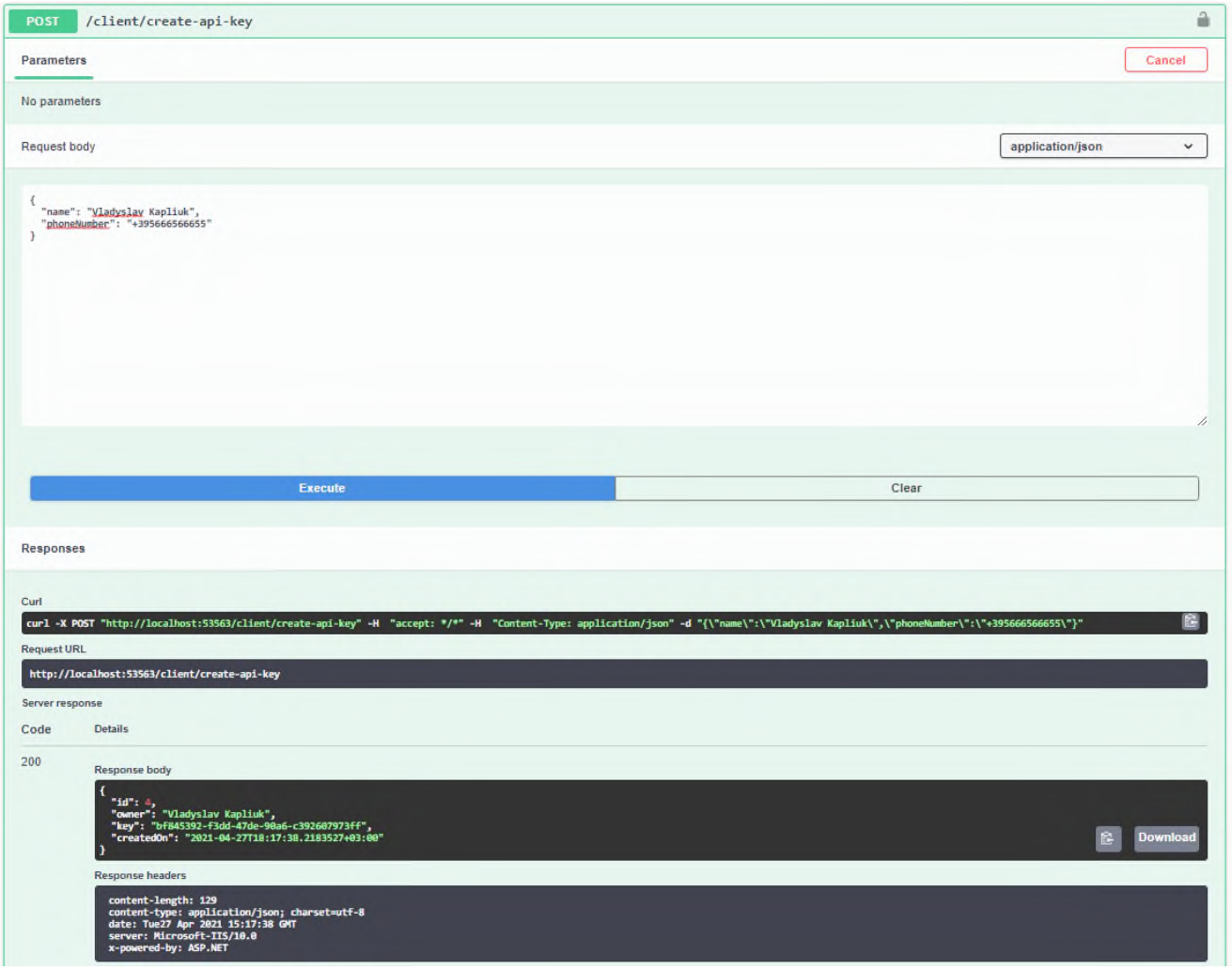

Рисунок 5.4 – результат POST /client/create-api-key отримання ключа

Тепер із цим ключем необхідно авторизуватись (рис. 5.5), і далі можна використовувати запити, які описані в Swagger. Хочу звернути увагу, що за межами цієї сторінки, для того, щоб користуватись API, як описано на рис.5.5, необхідно додавати цей ключ як значення в хедері API-Key при кожному запиті.

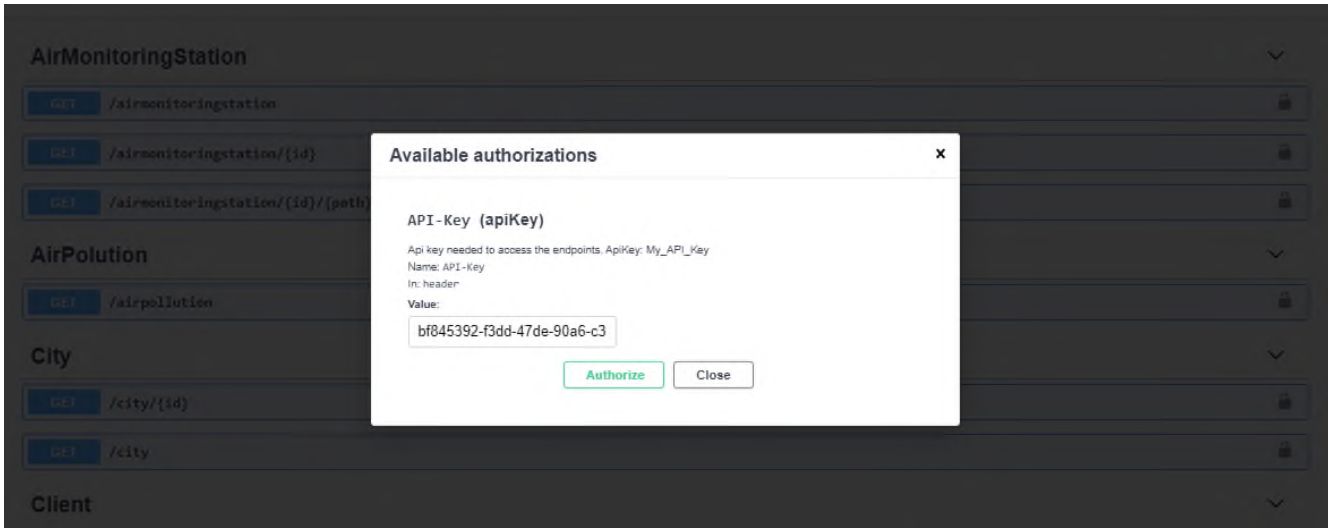

Рисунок 5.5 – Авторизація в Swagger

Приклад використання API в Postman, як було сказано вище: необхідно додати в Header ключ API-Key, а в якості значення – отриманий ключ у попередньому кроці.

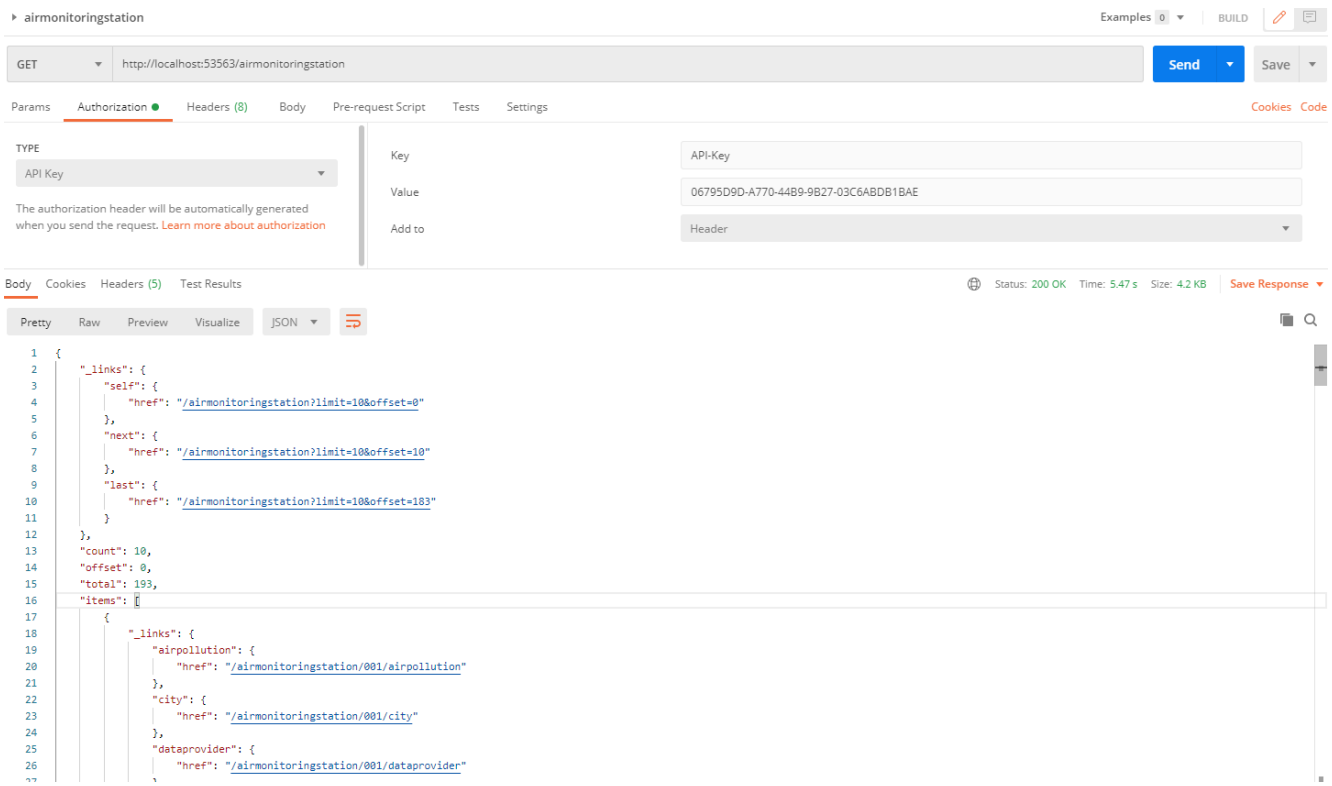

Рисунок 5.6 – використання API клієнтом Postman

У формі відповіді count (рис.5.6) кількість елементів, які повернув запит, offset – зсув, total – загальна кількість елементів по цьому ресурсу. В об’єкті _links прийшли відповідно посилання на наступні елементи з розрахованим зсувом, посилання на цей ж запит та посилання на запит, який забиратиме останню сторінку з даними. В середині масиву values зберігаються елементи станцій моніторингу, кожен із елементів має свої посилання відповідно до доступних ресурсів.

Якщо перейдемо за посиланням у середині одного елемента з масиву станцій, то отримаємо безпосередньо сам елемент (рис.5.8).

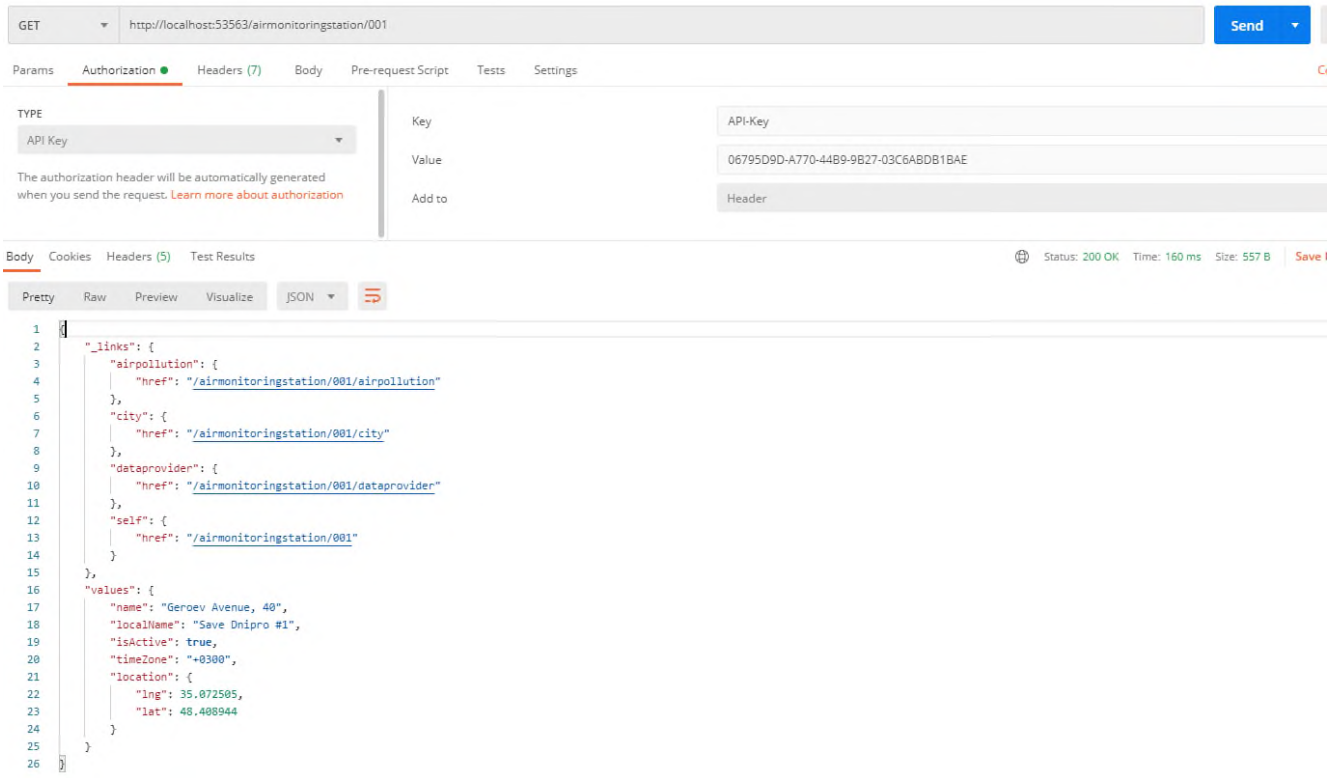

Рисунок 5.7 – доступ до ресурсу використовуючи посилання з попереднього запиту

Однією з вимог була реалізація resourse expantion. Для цього необхідно додати параметр запиту include=назва ресурсу 1, назва ресурсу 2. Тоді результатом буде розширена модель із відповідними включеннями. Ці включення доступні за посиланнями в об’єкті _links, назви полів об’єкту _links і є списком

можливих включень. Приклад цього ж запиту із відповідними включеннями із містом, провайдером даних і забрудненням можна переглянути на рис. 5.8

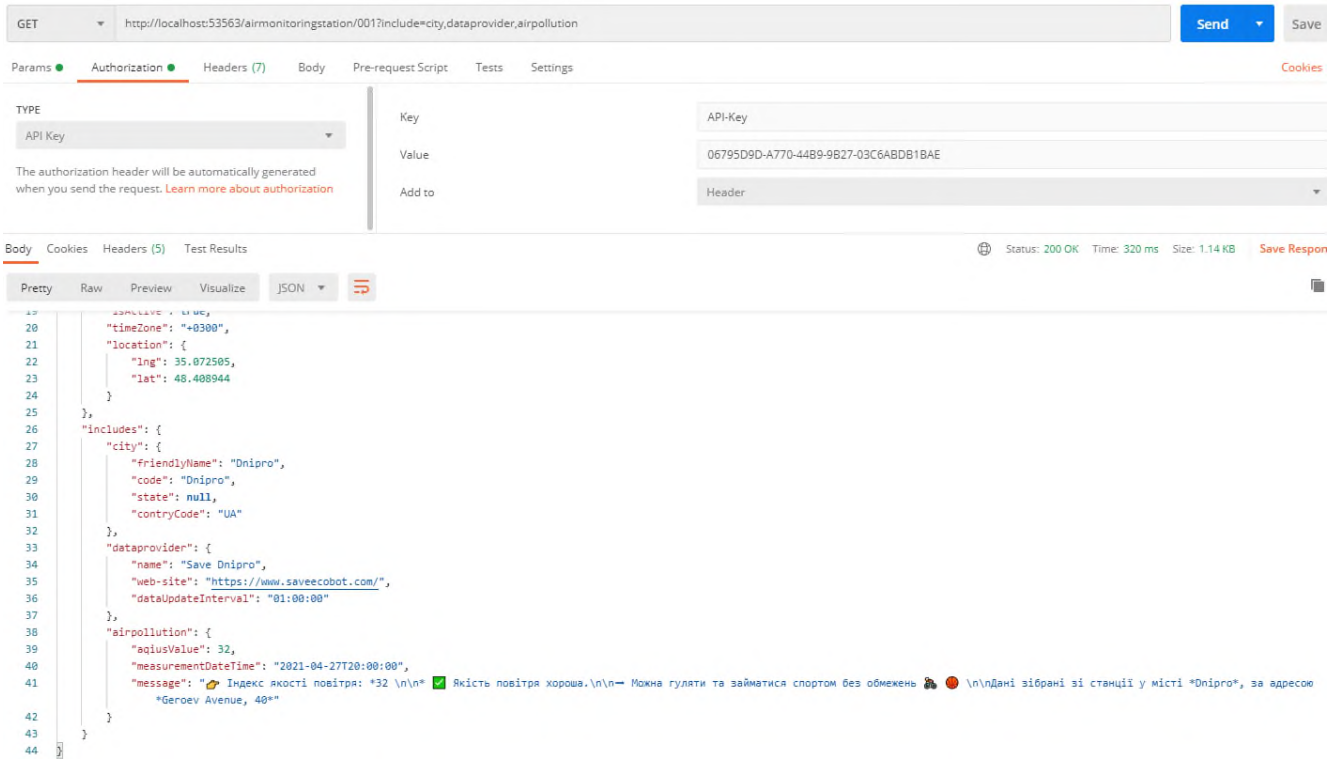

Рисунок 5.8 – демонстрація розширення ресурсу станції моніторингу

Звернення до ресурсу airpollution для отримання даних про якість повітря за координатами рис. 5.9

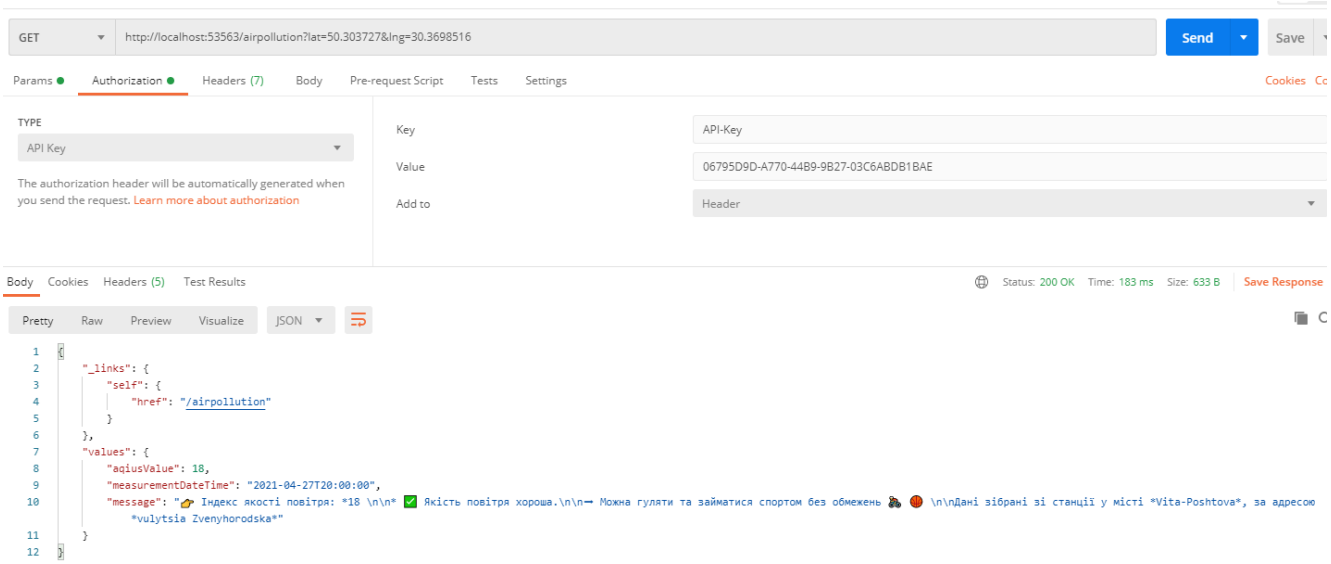

Рисунок 5.9 – запит якості на основі місцезнаходження

ВИСНОВКИ

При виконанні дипломної роботи було розроблено Web API додаток, який надає публічний RESTful API до даних місця розташування та вимірів станцій, це дає змогу усім бажаючим інтегрувати знання про якість повітря в свої системи. З цією метою було проведено огляд програм, платформ та систем, які розміщують інформацію про якість повітря, здійснено аналіз існуючих API. Система була реалізована, використовуючи предметно-орієнтований підхід проектування в домені моніторингу якості повітря, була розроблена масштабована система, яка дозволяє з легкістю розширювати проект, не змінюючи його. Реалізовані модулі, які дозволяють значно пришвидшити розширення ресурсів рівня презентації.

Під час розробки серверу додатку були поглиблені та закріплені знання використаних технологій. Як провайдер бази даних була використана нереляційна база MongoDB, а для написання серверної частина була обрана платформа .NET.

Проект містить в собі багато технологій та підходів, стратегій в розробці програмного забезпечення, успішне використання яких призвело до створення масштабованої системи, що задовольняє всі поставлені вимоги.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Статистичні данні про екологію України [Електронний ресурс] – Режим доступу до ресурсу: https://ecoaction.org.ua/diyalnist/povitria
2. ОС Windows 10 [Електронний ресурс] - Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Windows_10
3. MongoDB [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/MongoDB
4. MongoDB Compas [Електронний ресурс] – Режим доступу до ресурсу: https://www.mongodb.com/products/compass
5. .Net Framework [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/.NET_Framework
6. .Net Core [Електронний ресурс] – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/.NET_Core
7. ASP.NET Core [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/ASP.NET_Core
8. Git [Електронний ресурс] – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/Git
9. SourceTree [Електронний ресурс] – Режим доступу до ресурсу: https://www.sourcetreeapp.com/
10. GitHub [Електронний ресурс] – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/GitHub
11. GitFlow [Електронний ресурс] – Режим доступу до ресурсу: https://bitworks.software/2019-03-12-gitflow-workflow.html
12. Geohash [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Geohash

13. Onion architecture [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/@shivendraodean/software-architecture-the-onion-architecture-1b235bec1dec
14. API Key [Електронний ресурс] – Режим доступу до ресурсу: https://cloud.google.com/endpoints/docs/openapi/when-why-api-key
15. Крива Мортана [Електронний ресурс] – Режим доступу до ресурсу: https://domino.research.ibm.com/library/cyberdig.nsf/papers/0DABF9473B9C86D48525779800566A39/$File/Morton1966.pdf
16. Архітектурний стиль REST [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Representational_state_transfer
17. DDD [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Domain-driven_design
18. HTTP [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview

ДОДАТКИ

Додаток А. Лістинг ResoursePathResolver

```
public class ResoursePathResolver : IResoursePathResolver
    {
        private readonly ISearchAlgorithm _searchAlgorithm;
        private readonly Graph _graph;

        private readonly Dictionary<string, Dictionary<string, Resourse>>
_resourseResolutionCache;

        public ResoursePathResolver(ISearchAlgorithm searchAlgorithm, Graph
graph)
        {
            _resourseResolutionCache = new Dictionary<string,
Dictionary<string, Resourse>>();
            _searchAlgorithm = searchAlgorithm;
            _graph = graph;
        }


        public Dictionary<string, Resourse> GetResourses(string
controllerPath)
        {
            var result = new Dictionary<string, Resourse>();
            if (!_resourseResolutionCache.ContainsKey(controllerPath))
            {
                var currentVertex = _graph.Vertices.Single(item =>
item.ResourseName == controllerPath);
                var route = _searchAlgorithm.GetRoute(currentVertex,
_graph);
                foreach (var item in route)
                {

result.Add(_graph.Vertices[route[item.Key][^1]].ResourseName, new Resourse
                    {
                        Path = route[item.Key].Select(item => "/" +
_graph.Vertices[item].ResourseName).Aggregate((x, y) => $"{x}{y}"),
                    });
                }
                result.Add("self", new Resourse { Path = "" });
                _resourseResolutionCache.Add(controllerPath, result);
                return result;
            }

            return _resourseResolutionCache[controllerPath];
        }

        public bool IsPathValid(string controllerPath, string id, string
queryPath)
        {
            var currentResourse = new Resourse { Path = "/" + queryPath };
```

```
            if (!_resourseResolutionCache.ContainsKey(controllerPath))
            {
                var resourses = GetResourses(controllerPath);
                return resourses.Values.Contains(currentResourse);
            }
            return
_resourseResolutionCache[controllerPath].Values.Contains(currentResourse);
        }

        public string GetIncludeByPath(string controllerPath, string
queryPath)
        {
            var currentResourse = new Resourse { Path = "/" + queryPath };
            if (!_resourseResolutionCache.ContainsKey(controllerPath))
            {
                var resourses = GetResourses(controllerPath);
                return resourses.FirstOrDefault(item => item.Value ==
currentResourse).Key;
            }
            return
_resourseResolutionCache[controllerPath].FirstOrDefault(item =>
item.Value.Equals(currentResourse)).Key;
        }

        public string[] GetValidQueryIncludes(string controllerPath,
string[] includes)
        {
            if (!_resourseResolutionCache.ContainsKey(controllerPath))
            {
                var resourses = GetResourses(controllerPath);
                return includes.Where(item
=>resourses.ContainsKey(item)).ToArray();
            }
            return includes.Where(item =>
_resourseResolutionCache[controllerPath].ContainsKey(item)).ToArray();
        }


    }
```

Додаток Б. Лістинг GraphBuilder

```
    public class GraphBuilder
    {
        private class VertexMetaData
        {
            public string Resourse { get; set; }

            public List<string> OutEdges { get; set; }
        }

        public static Graph BuildResourseGraph()
```

```
        {
            var controllersMetaInformation =
Assembly.GetExecutingAssembly().GetTypes()
                .Where(myType => myType.IsClass && !myType.IsAbstract &&
myType.IsSubclassOf(typeof(ControllerBase)));

            var intermediateResult = new List<VertexMetaData>();
            foreach (var controllerType in controllersMetaInformation)
            {
                intermediateResult.Add(new VertexMetaData
                {
                    Resourse =
((RouteAttribute)controllerType.GetCustomAttribute(typeof(RouteAttribute)))
.Template,
                    OutEdges =
controllerType.GetCustomAttributes(typeof(IncludeResourseAttribute))
                        .Select(item =>
((IncludeResourseAttribute)item).Name).ToList()
                });
            }

            Dictionary<string, Vertex> vertecies = new();
            var graph = new Graph(intermediateResult.Count);
            foreach (var item in intermediateResult)
            {
                var vertex = new Vertex { ResourseName = item.Resourse };
                vertecies.Add(item.Resourse, vertex);
                graph.AddVertex(vertex);
            }

            foreach (var item in intermediateResult)
            {
                foreach (var vertexKey in item.OutEdges)
                {
                    if (vertecies.TryGetValue(vertexKey, out _))
                    {
                        graph.AddEdge(vertecies[item.Resourse],
vertecies[vertexKey]);
                    }
                    else
                    {
                        throw new ArgumentException($"You cannot use
IncludeResourseAttribute with Name: {vertexKey} which" +
                            " is not represented by controller with the
same RouteAttribute");
                    }
                }
            }
            return graph;
        }
    }
```

Додаток В. Приклади запитів та відповідей до API

```
// GET /airmonitoringstations HTTP/1.1
// 200 OK
// Content-Type: application/json

{
    "_links":{
        "self":{
            "href":"/airmonitoringstation?limit=4&offset=0"
        },
        "next":{
            "href":"/airmonitoringstation?limit=4&offset=4"
        },
        "last":{
            "href":"/airmonitoringstation?limit=4&offset=350"
        }
    },
    "count":4,
    "offset":0,
    "total":352,
    "items": [
           {
                "_links": {
                     "self":{
                          "href":"/airmonitoringstations/1"
                     },
                     "airPollution":{
                          "href":"/airmonitoringstations/1/airPollution"
                     },
                     "dataprovider":{
                          "href":"/airmonitoringstations/1/dataprovider"
                     },
                     "city":{
                           "href":"/airmonitoringstations/1/city"
                     }
                },
                "values": {
                     "id": "1",
                     "name": "My name",
                     "localName": "localName",
                     "isActive": 1,
                     "timeZone": "timezone",
                     "location": {
                          "lat": 15.16655,
                          "lng": 456.455
                     }
                }

           },
           {
                "_links": {
                     "self":{
                          "href":"/airmonitoringstations/2"
```

```
                },
                "airPollution":{
                      "href":"/airmonitoringstations/2/airPollution"
                },
                "dataprovider":{
                      "href":"/airmonitoringstations/2/dataprovider"
                },
                "city":{
                       "href":"/airmonitoringstations/2/city"
                }
          },
          "values": {
                "id": "2",
                "name": "My name",
                "localName": "localName",
                "isActive": 1,
                "timeZone": "timezone",
                "location": {
                      "lat": 15.16655,
                      "lng": 456.455
                }
          },

    },
    {
          "_links": {
                "self":{
                      "href":"/airmonitoringstations/3"
                },
                "airPollution":{
                      "href":"/airmonitoringstations/3/airPollution"
                },
                "dataprovider":{
                      "href":"/airmonitoringstations/3/dataprovider"
                },
                "city":{
                       "href":"/airmonitoringstations/3/city"
                }
          },
          "values": {
                "id": "3",
                "name": "My name",
                "localName": "localName",
                "isActive": 1,
                "timeZone": "timezone",
                "location": {
                      "lat": 15.16655,
                      "lng": 456.455
                }
          }

    },
    {
          "_links": {
                "self":{
```

```
                              "href":"/airmonitoringstations/4"
                        },
                        "airPollution":{
                              "href":"/airmonitoringstations/4/airPollution"
                        },
                        "dataprovider":{
                              "href":"/airmonitoringstations/4/dataprovider"
                        },
                        "city":{
                               "href":"/airmonitoringstations/4/city"
                        }
                  },
                  "values": {
                        "id": "4",
                        "name": "My name",
                        "localName": "localName",
                        "isActive": 1,
                        "timeZone": "timezone",
                        "location": {
                              "lat": 15.16655,
                              "lng": 456.455
                        }
                  }

            }
      ]
}

// GET
/airmonitoringstations/{stationId}?include=airPollution,dataprovider,city
HTTP/1.1
// 200 OK
// Content-Type: application/json
{
     "_links":{
         "self":{

"href":"/airmonitoringstations/{stationId}?include=airPollution,dataprovide
r,city"
         },
            "airPollution":{
             "href":"/airmonitoringstations/{stationId}/airPollution"
         },
            "dataprovider":{
             "href":"/airmonitoringstations/{stationId}/dataprovider"
         },
            "city":{
              "href":"/airmonitoringstations/{stationId}/city"
         }
     },
       "values": {
            "id": "1",
            "name": "My name",
            "localName": "localName",
            "isActive": 1,
```

```
            "timeZone": "timezone",
            "location": {
                  "lat": 15.16655,
                  "lng": 456.455
            }
      },
      "includes": {
            "airPollution" : {
                  "aquisValue": 1,
                  "measurementDateTime": "DATETIME",
                  "windSpeed": 50,
                  "humidity": 13,
                  "temperature": 20,
                  "aiqResult": "AIQ value: {AqiusValue},
MeasurementDateTime: {MeasurementDateTime}"
            },
            "city" : {
                  "code": 1234,
                  "countryCode": 3455,
                  "friendlyName": "friendly name",
                  "state": "state name"
            },
            "dataprovider" : {
                  "name" : "SaveDnipro",
                  "description" : "",
                  "web-site": "www.savednipro.org",
            }
      }
}


// GET /airPollution?lat=23&lng=45 HTTP/1.1
// 200 OK
// Content-Type: application/json
{
    "_links":{
        "self":{
            "href":"/airPollution?lat=23&lng=45&radius=5"
        }
    },
     "values": {
            "aquisValue": 1,
            "measurementDateTime": "DATETIME",
            "windSpeed": 50,
            "humidity": 13,
            "temperature": 20,
            "aiqResult": "AIQ value: {AqiusValue}, MeasurementDateTime:
{MeasurementDateTime}"
      }
}

// GET /dataproviders/{id} HTTP/1.1
// 200 OK
// Content-Type: application/json
{
```

```
    "_links":{
        "self":{
            "href":"/dataproviders/{id}"
        }
    },
     "values": {
           "name" : "SaveDnipro",
           "description" : "",
           "web-site": "www.savednipro.org",
     }
}
```