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

Кваліфікаційна робота
на здобуття ступеня магістра
за освітньо-науковою програмою «Інформатика»
спеціальності 122 «Комп'ютерні науки»
на тему:

ІНФОРМАЦІЙНА СИСТЕМА МОНІТОРИНГУ ЯКОСТІ ПОВІТРЯ

Виконав студент 2-го курсу
Владислав КАПЛЮК

(підпис)

Консультант:
доцент, кандидат технічних наук
Олексій ТКАЧЕНКО

(підпис)

Науковий керівник:
асистент, кандидат технічних наук
Олексій ФЕДОРУС

(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

«08» травня 2023 р., протокол № 16
Завідувач кафедри
Микола НІКІТЧЕНКО

(підпис)

Київ – 2023

РЕФЕРАТ

Обсяг роботи 67 сторінок, 22 ілюстрація, 2 таблиці, 17 джерел посилань.

Об’єктом дослідження є архітектурний дизайн та імплементація системи моніторингу якості повітря. Предметом роботи є система для вирішення прикладної задачі отримання даних з джерел доступу до станцій моніторингу, знаходження найближчої станції моніторингу якості повітря та відображення інформації про його стан на основі місцезнаходження користувача.

Метою роботи є створення інформаційної системи для отримання актуальної інформації стану повітря на основі місцезнаходження.

Методи розроблення: розробка програмного продукту, архітектурні рішення.

Інструменти розроблення: операційна система - Windows 10, Visual Studio 2022, Amazon SQS, Amazon Lambda, Amazon EventBridge, .NET 6, Postman, MongoDB, Docker.

Результати роботи: спроектовано та створено розширювану масштабовану систему. Підхід до рішення це використання serverless технологій та розробка модульної структури системи що включає окремі компоненти для забору даних від джерел що мають доступ до станцій, передачі даних, обробки даних візуалізації та доступу до інформації. На основі цього вдосконалено програмний продукт «AirSnitch».

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

Система впроваджена у роботу та має близько двох тисяч користувачів.

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.

AWS – Amazon Web Services.

SQS – Simple queue service.

TPL – Task parallel library.

ВСТУП

Оцінка сучасного стану об’єкта розробки. В сучасному світі навігація стає все більш важливою через стрімке зростання міст і збільшення їх складності. Разом з розвитком міст, рівень забруднення повітря також збільшується через діяльність різноманітних підприємств та зростання кількості автотранспорту. Часто, прогулюючись вулицями, ми не замислюємося про те, наскільки корисним може бути наше перебування на вулиці. Вчені всього світу займаються проблемами зниження забруднення повітря. В даний час існує багато видів сенсорів для вимірювання забруднюючих речовин у повітрі, що варіюються від простих та доступних до високоточних та дорогих. Ці сенсори зазвичай забезпечують дані про концентрацію різних забруднюючих речовин у реальному часі. В багатьох країнах вже встановлені мережі стаціонарних сенсорів для моніторингу якості повітря. Однак, ці мережі можуть мати обмежений охоплення території або недостатньо часто оновлювати дані. В деяких випадках, дані з різних джерел та сенсорів можуть бути несумісними або складними для аналізу. Це ставить перед розробниками системи виклик інтеграції та стандартизації даних. Існують різні платформи та інструменти для візуалізації даних про якість повітря, проте не всі з них можуть бути легкими у використанні або доступними для широкого загалу користувачів.

Створення системи яка надавала б інформацію щодо якості повітря на основі геолокації об’єкта є практичною проблемою що має в собі декілька підзадач:

• збір даних про розташування та параметри станцій, що відслідковують якість повітря;

• приведення даних до єдиного формату за допомогою стандартизації;

• знаходження найближчих станцій до користувача і отримання даних про якісь повітря в цьому районі.

Актуальність роботи та підстави для її виконання. Забруднення повітря є серйозною проблемою, яка спричинює передчасну смерть мільйонів людей через різні захворювання, пов'язані з диханням. Україна також стикається з цією

проблемою, і згідно зі статистикою, вона є однією з найбільш забруднених країн у світі [1]. Пропонований додаток допоможе людям отримувати інформацію про якість повітря у своїй окрузі, що може допомогти збільшити усвідомленість про проблему та привернути увагу до неї. Актуальність розуміння стану якості повітря в умовах війни набуває особливого значення, оскільки військові дії можуть призвести до значних змін у рівні забруднення повітря. Такі зміни можуть бути результатом ряду факторів, таких як:

- знищення інфраструктури: військові дії можуть призвести до руйнування промислових об'єктів, енергетичних споруд, транспортних мереж та інших об'єктів, що можуть призвести до викиду шкідливих речовин у повітря;
- використання вогнепальної зброї та вибухів: стрілянина, вибухи та використання бойової техніки можуть спричинити викиди шкідливих речовин у повітря. Також можливе викидання хімічних, біологічних та радіоактивних речовин у результаті використання забороненої зброї;
- відсутність моніторингу: у зонах військових дій, системи моніторингу якості повітря можуть бути знищені або відключені, що ускладнює контроль за якістю повітря та підготовку вчасних заходів;
- руйнування екосистем: війна може призвести до значних змін у природному середовищі, які можуть негативно впливати на якість повітря. Зокрема, вирубка лісів, знищення водойм та інші форми деградації екосистем можуть знизити здатність природи самоочищатися;
- масові переміщення людей: війна може спричинити великі масові міграції населення, що може створити тимчасові табори з низьким рівнем санітарії та значним забрудненням повітря. У таких умовах, спалахи хвороб, пов'язаних з якістю повітря, можуть різко зрости, створюючи загрозу для здоров'я вже уразливого населення;
- складність реагування на екологічні кризи: у зонах конфлікту, екологічні кризи, такі як надзвичайні ситуації пов'язані з якістю повітря, можуть бути

ігноровані або недооцінені через відсутність ресурсів, інфраструктури та зосередження уваги на безпосередніх військових завданнях;

- підвищення рівня стресу та негативний вплив на здоров'я: постійний стрес та невизначеність, пов'язані з війною, можуть погіршити загальний стан здоров'я людей, що у свою чергу знижує їх здатність протистояти негативному впливу забрудненого повітря.

Враховуючи ці фактори, актуальність розуміння стану якості повітря в умовах війни стає критично важливою. Моніторинг якості повітря може допомогти у координації евакуацій, розподілі ресурсів для забезпечення здоров'я та запобігання виникненню екологічних криз. Відповідні системи також можуть сприяти підготовці та відновленню після війни, включаючи реконструкцію інфраструктури та забезпечення безпечного середовища для повернення населення.

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;
- розробити архітектуру інформаційної системи моніторингу якості повітря;
- вибрати оптимальні джерела даних для забезпечення системи актуальною інформацією про якість повітря;
- розробити алгоритми обробки даних про якість повітря з урахуванням геолокації об'єктів;
- розробити зручний інтерфейс для користувачів системи та її інтеграцію з іншими додатками та сервісами;
- розгортання системи на цільових платформах;
- розробити стратегію розвитку та модернізації системи моніторингу якості повітря

Завершивши виконання цих завдань, мета розробки інформаційної системи моніторингу якості повітря буде досягнута. Система дозволить забезпечити своєчасний доступ до актуальної інформації про стан повітря та допоможе виявляти та реагувати на можливі негативні наслідки для здоров'я людей та довкілля в умовах війни та інших надзвичайних ситуацій.

Об'єкт, методи й засоби розроблення. Об’єктом виконання роботи є процес отримання даних із різних джерел доступу до станцій моніторингу та пошук найближчих відповідно до координат користувача. Предметом роботи є розробка системи моніторингу якості повітря, яка забезпечує збір, обробку, візуалізацію даних про стан повітря в різних регіонах, зокрема в умовах війни, для попередження негативних наслідків для здоров'я населення та довкілля.

У якості інструментів розробки та тестування програмного засобу було обрано:

- Visual Studio 2019 - інтегроване середовище розробки.
- Платформа .NET - набір інструментів для розробки та виконання програмного забезпечення, який включає у себе віртуальну машину .NET (CLR), бібліотеки класів .NET та засоби розробки. Вона підтримує багато мов програмування, таких як C#, F# та VB.NET, та може використовуватися для розробки десктопних, веб- та мобільних додатків.
- C# - це об'єктно-орієнтована мова програмування, яка була розроблена компанією Microsoft із ціллю розробки додатків для платформи .NET.
- MongoDB – документо-орієнтована система керування базами даних з відкритим вихідним кодом, яка не потребує опису схеми таблиць.
- Amazon SQS (Simple Queue Service) - це сервіс керування чергами повідомлень, який дозволяє розподіляти та обробляти повідомлення між окремими компонентами програмного забезпечення.

- Amazon Lambda - це сервіс, який дозволяє виконувати код без необхідності управління серверами. Він дозволяє запускати функції (написані на різних мовах програмування) відповідно до певних подій або розкладу.
- Amazon EventBridge - це повністю керований сервіс, який дозволяє забезпечити публікацію, знаходження та обробку подій на різних джерелах та призначених для різних призначень. Він дозволяє обробляти та пересилати події між AWS-сервісами, а також зовнішніми додатками.
- Postman – потужний набір інструментів тестування API, зручний у використані командою через можливість використовувати спільні запити;
- Git – розподілена система контроля версій.
- Fork – клієнт для git з GUI.

Можливі сфери застосування. Можливі сфери застосування інформаційної системи моніторингу якості повітря включають:

Державний сектор:

- міністерства та державні установи, що займаються екологією та охороною навколишнього середовища;
- місцеві органи влади, які відповідають за контроль якості повітря на території своєї юрисдикції;
- військові та цивільні оборонні структури, які мають потребу в моніторингу якості повітря в умовах війни або інших надзвичайних ситуацій.

Промисловість:

- підприємства, що спричиняють забруднення повітря, які повинні контролювати та зменшувати свій екологічний вплив;
- компанії, які займаються розробкою та впровадженням екологічних технологій, які можуть скористатися даними про якість повітря для аналізу та прогнозування ефективності своїх рішень.

Наукові та дослідницькі установи:

- організації, які проводять дослідження в галузі екології, атмосферної хімії та кліматології, і можуть використовувати дані про якість повітря для аналізу та розробки нових методів зменшення забруднення;
- учбові заклади, які займаються підготовкою спеціалістів в галузі охорони навколишнього середовища, та можуть використовувати інформаційну систему моніторингу якості повітря в навчальному процесі.

Медичні установи та організації охорони здоров'я:

- лікарні, клініки та інші медичні заклади, які можуть використовувати дані про якість повітря для прогнозування та запобігання різних захворювань, пов'язаних із забрудненням повітря, таких як алергії, респіраторні захворювання та серцево-судинні проблеми;
- організації та асоціації охорони здоров'я, які можуть використовувати інформаційну систему моніторингу якості повітря для виявлення та вирішення проблем публічного здоров'я, пов'язаних з якістю повітря.

Сфера туризму та відпочинку:

- туристичні агенції, готелі та інші компанії, що працюють у сфері туризму, які можуть інформувати своїх клієнтів про якість повітря в певній місцевості, допомагаючи їм планувати подорожі та відпочинок, з урахуванням екологічної ситуації;
- організатори спортивних заходів та масових заходів на відкритому повітрі, які можуть враховувати якість повітря при плануванні та проведенні своїх заходів, для забезпечення комфорту та безпеки учасників та відвідувачів.

Розробники програмного забезпечення та інтегратори систем:

- компанії, що створюють мобільні додатки, веб-сервіси та інші програмні рішення, які можуть використовувати дані про якість повітря, надані інформаційною системою моніторингу, для реалізації функцій моніторингу та аналізу якості повітря в своїх продуктах;

- інтегратори систем, які можуть використовувати інформаційну систему моніторингу якості повітря для розширення функціональності та покращення екологічних показників вже існуючих систем управління містами, промисловими об'єктами та інфраструктурою.

Засоби масової інформації та громадські організації:

- ЗМІ, які можуть використовувати дані про якість повітря для інформування громадськості про екологічну ситуацію в регіоні, висвітлення проблем та залучення уваги до питань охорони навколишнього середовища;
- громадські організації та екологічні ініціативи, які можуть використовувати інформаційну систему моніторингу якості повітря для збору та аналізу даних, розробки та реалізації проектів з покращення екологічної ситуації в регіоні, а також для контролю дій органів влади та промислових підприємств у сфері екології.

Інформаційна система моніторингу якості повітря може мати широкий спектр застосувань у різних сферах суспільного життя, сприяючи підвищенню екологічної свідомості та забезпеченню безпеки та здоров'я людей у різних умовах, включаючи військові дії.

Тези даної роботи було представлено на науково-практичній конференції студентів і аспірантів «Теоретичні та прикладні аспекти розробки комп’ютерних систем 2023» [2].

РОЗДІЛ 1. ОГЛЯД ІСНУЮЧИХ НА РИНКУ СИСТЕМ

AirVisual

AirVisual - це глобальна платформа моніторингу якості повітря, яка надає дані з офіційних державних станцій моніторингу, а також від приватних вимірювачів. Користувачі можуть переглядати дані про якість повітря, AQI та прогнози на веб-сайті та мобільних додатках.

Plume Labs

Французька компанія Plume Labs розробила додаток Air Report, який надає користувачам інформацію про якість повітря, AQI та прогнози для міст у всьому світі. Додаток використовує дані з офіційних джерел моніторингу, а також власні алгоритми та моделі для аналізу якості повітря. Plume Labs також випустила персональний прилад для моніторингу якості повітря, Plume Flow, який дозволяє користувачам вимірювати рівень забруднення навколо них у режимі реального часу.

BreezoMeter

Це ізраїльська компанія, що розробила систему моніторингу якості повітря на основі аналізу даних з різних джерел, включаючи державні станції моніторингу, супутникові дані, метеорологічні моделі та інші параметри. BreezoMeter надає інформацію про якість повітря у вигляді AQI, концентрації забруднюючих речовин та прогнози через API, що дозволяє інтеграцію з різними системами та додатками

luftdaten.info

Це німецький веб-ресурс, що надає інформацію про забруднення повітря по всій Європі. Він містить детальні дані про кожну зі станцій моніторингу, які відображені на карті та вказані в легенді. Ресурс також включає метеорологічні дані, такі як показники вітру. Варто зазначити, що на сайті вказані представництва та контактна інформація організацій, відповідальних за проведення вимірювань.

SaveEcoBot

На сьогоднішній день цей продукт є одним з перших випробувань свого роду на території нашої країни. Він представляє себе як єдиний екологічний чат-бот в Україні. Ця платформа збирає інформацію з моніторингових станцій розташованих по всій Україні та відтворює ці дані на своєму веб-сайті або через Telegram-чат-бот. Основна мета організації полягає у покращенні якості повітря в країнах з розвиваючою економікою шляхом збільшення екологічної свідомості населення, надаючи безоплатний та простий доступ до інформації про індекс якості повітря та інші відкриті екологічні дані. Крім того, будь-яка зацікавлена особа може подати скаргу про порушення з боку людей, спільнот або організацій, що стосуються забруднення повітря, води, ґрунту, відходів або надмірного шуму.

Результати аналізу існуючих систем наведені в порівняльній таблиці 1.1 виділено наступні критеріїв:

- виміри на території України;
- безкоштовне API;
- провайдери даних;
- мобільний клієнт.

Таблиця 1.1 – Порівняльна характеристика існуючих систем

Критерій	Назва застосунку				
	Plume Labs	SaveEcoBot	luftdaten.info	BreezoMeter	AirVisual
Виміри з території України	+	+	+	+	+
Безкоштовне API	-	+	+	-	-
Провайдери даних	Lun Misto Air	SaveDnipro	luftdaten.info	BreezoMeter	AirVisual
Мобільний клієнт	-	+	-	+	+

РОЗДІЛ 2. ОГЛЯД ВИКОРИСТАНИХ ТЕХНОЛОГІЙ

2.1. Огляд середовища NoSQL

Проект був реалізований на основі операційної системи Windows 10 та використовував MongoDB як систему керування базами даних (СКБД). MongoDB - це швидка, масштабована, документо-орієнтована СКБД, яка працює з даними у форматі "ключ-значення" та відрізняється від реляційних СКБД своїми можливостями. MongoDB зберігає документи у JSON-подібному форматі, має гнучку мову запитів, дозволяє створювати індекси, ефективно зберігає великі бінарні об'єкти, підтримує логування, може працювати за парадигмою Map/Reduce, присутні можливості використання реплікацій та побудова відмовостійких конфігурацій. MongoDB також має вбудовані інструменти для шардінгу, що дозволяє створити горизонтально масштабований кластер зберігання без єдиної точки відмови та з автоматичним відновленням після збою. Розширення кластера може відбуватися без зупинки роботи БД шляхом додавання нових машин. [3]

База даних була створена за допомогою MongoDB Compass - візуального інструменту для роботи з MongoDB. Цей графічний інтерфейс дозволяє візуалізувати дані, швидко виконувати специфічні запити, керувати даними з повним набором CRUD-функцій, а також переглядати та оптимізувати продуктивність запитів. Програма сумісна з Linux, Mac та Windows і допомагає приймати обґрунтовані рішення щодо індексації, огляду документів та іншого. Вона також включає вбудовані можливості візуалізації схем.

MongoDB Compass аналізує документи і представляє складні структури колекцій через зручний графічний інтерфейс. Це сприяє швидкому візуальному дослідженню та аналізу схеми, що дозволяє зрозуміти частоту, типи та діапазони полів у даних.

2.2. Огляд платформи .NET Core

Платформа .NET - це відкритий і універсальний фреймворк розробки програмного забезпечення від компанії Microsoft, який підтримує ряд мов програмування, таких як C#, F# та Visual Basic. .NET спрямований на спрощення розробки, впровадження та управління програмами на різних платформах і пристроях.

Основні характеристики платформи .NET включають:

- Єдина система виконання (CLR - Common Language Runtime), яка забезпечує керування пам'яттю, безпеку та виконання коду для всіх мов, що працюють на платформі.
- Бібліотека класів .NET (BCL - Base Class Library) забезпечує загальну функціональність для різних типів програм, зменшуючи кількість коду, який необхідно написати вручну.
- Засоби для кросплатформної розробки, які дозволяють створювати застосунки, які працюють на різних операційних системах, таких як Windows, macOS, Linux, Android та iOS.
- Підтримка хмарної розробки та впровадження застосунків з використанням Microsoft Azure.
- Інтеграція з іншими технологіями Microsoft, такими як Entity Framework, ASP.NET та ML.NET.

.NET відрізняється своєю продуктивністю, гнучкістю та безпекою, що робить його привабливим вибором для розробників у різних галузях і сферах застосування.[5]

2.3. Огляд протоколу HTTP

Протокол HTTP (Hypertext Transfer Protocol) - це основний протокол передачі даних, який використовується в Інтернеті для обміну гіпертекстовою інформацією між клієнтами та серверами. HTTP базується на архітектурі клієнт-сервер і є застосуванням протоколу TCP (Transmission Control Protocol) на рівні транспорту.

Основні характеристики протоколу HTTP включають:

- Безстановість: HTTP не зберігає інформацію про попередні запити між сеансами, що робить кожен окремий запит незалежним від інших.
- Відкритий текст: HTTP-запити та відповіді передаються у вигляді зрозумілого для людини тексту, що спрощує розуміння та налагодження.
- Запит-відповідь: Клієнти (зазвичай веб-браузери) ініціюють запити до серверів, які надсилають відповіді назад. Запити та відповіді містять рядки заголовків, що описують характеристики передачі даних, і тіла, які містять самі дані.
- Методи: HTTP використовує методи (GET, POST, PUT, DELETE тощо) для визначення дій, які мають бути виконані на ресурсі (URL). Кожен метод має свою семантику та правила використання.
- Коди статусу: Відповіді HTTP містять коди статусу, що вказують на результат запиту (успіх, помилка, перенаправлення тощо).

Протокол HTTP є простим у використанні, але не надає вбудовану безпеку на рівні передачі даних. Ця проблема вирішується за допомогою протоколу HTTPS (HTTP Secure), який додає шар зашифрованого з'єднання SSL/TLS для захисту даних від перехоплення та зміни.

HTTP постійно розвивається і вдосконалюється з часом, що призводить до випуску нових версій протоколу. Наприклад, HTTP/2 впровадив покращення, такі як більш ефективне використання з'єднань, двостороння комунікація та компресія заголовків, що сприяє підвищенню продуктивності та швидкодії.

HTTP/3 - це найновіша версія протоколу, яка базується на протоколі QUIC замість TCP. QUIC використовує UDP (User Datagram Protocol) і включає вбудовану безпеку, низьку затримку та інші покращення, що роблять HTTP/3 ще швидшим та надійнішим для сучасного Інтернету.

В цілому, протокол HTTP є фундаментальним компонентом Інтернету, який забезпечує обмін даними між клієнтами та серверами. Завдяки своїй простоті та гнучкості, HTTP продовжує розвиватися та відповідати вимогам сучасного веб-середовища.

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.[18]

2.3. Огляд архітектури REST

REST представляє собою архітектурний підхід, створений Роєм Філдінгом у 2000 році, який включає в себе набір обмежень для розробки API. API - це засоби та правила, які дозволяють створювати та інтегрувати програмне забезпечення, виступаючи як посередник між надавачем та користувачем інформації.

Щоб API відповідало принципам REST, воно має задовольняти такі вимоги:

1. Клієнт-серверна структура, яка базується на ресурсах та HTTP-запитах.
2. Безстанова взаємодія, коли дані про клієнта не зберігаються між запитами.
3. Кешування даних для оптимізації спілкування між клієнтом та сервером.
4. Єдність інтерфейсу для забезпечення стандартного обміну інформацією між компонентами.

REST API відрізняється від більш формалізованих протоколів, таких як SOAP, своєю простотою, швидкістю та масштабованістю, що робить його відмінним вибором для Інтернету речей та мобільних застосунків.

Representational State Transfer став широко використовуваним архітектурним стилем для розробки веб-сервісів та API через свою простоту та гнучкість. Аспекти, які варто розглянути стосовно REST:

1. Гнучкість форматів даних: На відміну від SOAP, який вимагає використання XML, REST підтримує різні формати даних, такі як JSON, XML, HTML, текст тощо. Це дозволяє розробникам обирати найбільш підходящий формат даних для своїх потреб.

2. Відкриті стандарти: REST базується на відкритих стандартах, таких як HTTP, URI та MIME, що сприяє його широкому прийняттю та сумісності з іншими системами.

3. Легка інтеграція: REST API може легко інтегруватися з різними платформами, мовами програмування та фреймворками, завдяки своїй простоті та відкритим стандартам.

4. Масштабованість: REST підтримує горизонтальну масштабованість, дозволяючи системам легко розширюватися для підтримки збільшення кількості користувачів та запитів.

5. Контроль версій: REST дозволяє розробникам впроваджувати контроль версій для своїх API, що спрощує внесення змін та підтримку зворотньої сумісності з клієнтськими додатками.

REST, безумовно, пропонує багато переваг для розробки веб-сервісів та API. Однак, як і з будь-яким технологічним рішенням, слід розглянути специфічні вимоги та обмеження проекту, щоб визначити, чи є REST найкращим варіантом для вашого випадку.

Хоча REST пропонує чимало переваг, важливо також звернути увагу на деякі його недоліки та обмеження:

1. Відсутність стандартів безпеки: На відміну від SOAP, REST не має вбудованих стандартів безпеки, таких як WS-Security. Це означає, що розробники повинні самостійно реалізовувати механізми безпеки для своїх REST API.
2. Обмеженість методів: REST використовує лише чотири основні методи HTTP: GET, POST, PUT і DELETE. Це може призвести до обмеження функціональності API, коли справа доходить до більш складних операцій або сценаріїв.
3. Проблеми зі станом: Оскільки REST вимагає безстанового клієнт-серверного зв'язку, розробники можуть стикатися з труднощами у збереженні стану або контексту між послідовними запитами. Це може призвести до неефективності, особливо у ситуаціях, коли потрібно передавати великі обсяги даних.
4. Керування багатьма запитами: Залежно від дизайну API, розробники можуть стикатися з проблемами, коли клієнтські додатки повинні виконувати послідовність запитів для отримання повної інформації. Це може призвести до погіршення продуктивності та затримок.
5. Нестандартизовані відповіді: REST не має жорсткої структури відповідей, що може ускладнити роботу розробників з різними API, які використовують різні формати та структури відповідей.

Незважаючи на ці недоліки та обмеження, REST продовжує бути популярним вибором для розробки веб-сервісів та API через свою простоту, гнучкість та масштабованість. Щоб вирішити деякі з цих проблем, розробники можуть розглянути наступні рішення:

1. Використання стандартів безпеки: Розробники можуть впроваджувати додаткові механізми безпеки для своїх REST API, такі як OAuth, JWT (JSON Web Tokens) або HTTPS. Це допоможе забезпечити конфіденційність, цілісність та аутентифікацію даних, передаваних між клієнтами та серверами.
2. Використання налаштовуваних методів: Хоча REST використовує обмежений набір методів, розробники можуть створювати налаштовувані методи та дії для своїх API, щоб задовольнити специфічні потреби та сценарії.
3. Застосування механізмів збереження стану: Для випадків, коли потрібно зберігати стан або контекст між послідовними запитами, розробники можуть розглянути використання механізмів збереження стану, таких як сесії, токени або кешування.
4. Оптимізація кількості запитів: Щоб уникнути погіршення продуктивності, викликаної виконанням багатьох запитів, розробники можуть впроваджувати методи оптимізації, такі як злиття декількох запитів в один, використання полягання даних або підтримка GraphQL.
5. Стандартизація відповідей: Розробники можуть стандартизувати формати відповідей, використовуючи загальноприйняті практики та конвенції, такі як JSON:API або HAL (Hypertext Application Language), що спрощує інтеграцію різних API для клієнтських додатків.

Незважаючи на зазначені недоліки, REST продовжує залишатися популярним вибором для розробки веб-сервісів та API через свою простоту, гнучкість та масштабованість. Враховуючи переваги та обмеження REST, розробники можуть вибрати відповідні техніки та підходи для реалізації своїх проектів. В кінцевому підсумку, важливо обрати архітектурний стиль та технології, які найкраще відповідають конкретним потребам та цілям проекту.[16]

2.3. Огляд фреймворку ASP.NET Core

ASP.NET Core - це відкритий, кросплатформений фреймворк для розробки веб-додатків і серверних програм, розроблений компанією Microsoft. Він є наступником ASP.NET і було створено з метою покращити продуктивність, гнучкість і масштабованість веб-розробки. Ось деякі ключові особливості та переваги ASP.NET Core:

- кросплатформеність: ASP.NET Core підтримує різні платформи, такі як Windows, macOS та Linux, що дає розробникам можливість розробляти і розгортати веб-додатки на різних операційних системах;
- висока продуктивність: ASP.NET Core використовує Kestrel, швидкий і легкий веб-сервер, який забезпечує високу продуктивність веб-додатків;
- модульність: ASP.NET Core має модульну архітектуру, яка дозволяє розробникам вибирати лише ті компоненти, які вони потребують для своїх додатків. Це допомагає зменшити навантаження на додаток та покращити його швидкість;
- вбудовані засоби розробки: ASP.NET Core надає потужні засоби розробки, такі як підтримка тестування, діагностики, відлагодження та інструментів збірки;
- інтеграція з іншими технологіями: ASP.NET Core дозволяє легко інтегруватися з різними технологіями та бібліотеками, такими як Entity Framework Core, IdentityServer4, Blazor та інші;
- підтримка контейнерів: ASP.NET Core підтримує розгортання в контейнерах, таких як Docker і Kubernetes, що спрощує процес розгортання та масштабування веб-додатків;
- захист та безпека: ASP.NET Core має вбудовані засоби безпеки, такі як захист від міжсайтового скриптування (XSS), міжсайтового підроблення запитів

(CSRF) та захист від атак на контроль доступу. Це допомагає розробникам створювати безпечні веб-додатки без необхідності впровадження додаткових засобів захисту;

- розширюваність: ASP.NET Core дозволяє розробникам створювати власні компоненти та розширення, що можуть бути легко інтегровані з фреймворком. Це дає можливість створювати налаштовані рішення, які відповідають конкретним вимогам проекту;
- шаблони та компоненти: ASP.NET Core надає набір готових шаблонів та компонентів для підтримки різних сценаріїв розробки, включаючи веб-додатки, API, реактивні додатки та мікросервіси. Це спрощує процес розробки та прискорює час виведення продукту на ринок;
- спільнота та підтримка: ASP.NET Core має велику активну спільноту розробників та користувачів, яка допомагає вирішувати проблеми та поширює знання через блоги, конференції та онлайн-курси. Крім того, Microsoft активно розвиває фреймворк, надаючи оновлення та покращення, що допомагає забезпечити його надійність та стабільність.

Узагальнюючи, ASP.NET Core - це потужний, гнучкий та продуктивний фреймворк для розробки веб-додатків та серверних програм. Він пропонує набір передових технологій, вбудованих засобів безпеки та підтримку різних платформ, що робить його привабливим вибором для розробників та організацій різного рівня та розміру [6]

2.5. Огляд технології Git

Git - це розподілена система керування версіями, яка дозволяє розробникам ефективно співпрацювати над проектами та відстежувати зміни в коді. Він був створений Лінусом Торвальдсом, автором ядра Linux, у 2005 році. Основні аспекти та переваги технології Git:

1. розподілена архітектура: Git - це розподілена система керування версіями, що означає, що кожен розробник має повну копію репозиторія на своєму комп'ютері. Це дозволяє розробникам працювати автономно, без постійного доступу до центрального сервера, та забезпечує резервне копіювання коду.
2. швидкість: Git відомий своєю високою швидкістю та продуктивністю, оскільки він забезпечує швидке збереження змін, галуження та злиття коду.
3. розгалуження та злиття: Git надає потужні інструменти для галуження (branching) та злиття (merging) коду, що дозволяє розробникам працювати над різними частинами проекту паралельно, а потім об'єднати їх з основною лінією розробки. Це сприяє більш гнучкому та ефективному процесу розробки.
4. відстеження змін: Git забезпечує відстеження змін та їх відновлення, що дозволяє розробникам відновлювати попередні версії коду, порівнювати зміни між версіями та відслідковувати проблеми.
5. стійкість: Завдяки своїй розподіленій архітектурі, Git має високу стійкість до відмов та пошкоджень. Навіть якщо одне робоче середовище зазнає пошкоджень, інші копії репозиторія зберігаються на комп'ютерах розробників, забезпечуючи відмінну відновлюваність.
6. колаборація: Git сприяє співпраці між розробниками, дозволяючи їм обмінюватися змінами, відгуками та внесками в код. Інтеграція з хостинговими сервісами, такими як GitHub, GitLab та Bitbucket, полегшує спільну роботу над проектами, організацію коду та відстеження проблем.
7. підтримка хуків: Git дозволяє використовувати хуки (hooks) для автоматизації різних процесів, пов'язаних з керуванням версіями, таких як тестування, перевірка коду або розгортання.
8. підтримка різних протоколів: Git підтримує різні протоколи передачі даних, включаючи HTTP, HTTPS, SSH та Git, що дозволяє розробникам вибрати найбільш підходящий спосіб обміну даними.

9. легкість навчання: Git має добре задокументовані команди та ресурси для навчання, які допомагають новим користувачам швидко освоїти його можливості та функціонал.
10. активна спільнота: Git має велику активну спільноту розробників та користувачів, яка допомагає вирішувати проблеми, розробляє нові інструменти та підтримує розвиток технології.

У підсумку, Git є надійною, швидкою та ефективною системою керування версіями, яка відповідає вимогам сучасних розробників та проектів різного рівня складності. Він сприяє колаборації, гнучкості розробки та стабільності коду, а також забезпечує відмінні можливості для резервного копіювання та відновлення версій коду. Завдяки своїй активній спільноті, підтримці різних протоколів та хорошим ресурсам для навчання, Git продовжує залишатися однією з найпопулярніших систем керування версіями для розробників по всьому світу.[11]

2.6. Огляд Onion Architecture

Архітектура Onion, відома також як "Архітектура шарів цибулі", є одним з підходів до структурування програмного забезпечення, розробленим Джеффрі Палермо. Основна ідея цієї архітектури полягає в тому, щоб розбити додаток на концентричні кільця або шари, кожен з яких має відповідальність за окремі аспекти програми. Основні принципи Onion Architecture включають:

- Внутрішні шари не мають знань про зовнішні шари. Замість цього, зовнішні шари залежать від внутрішніх.
- Код вищих рівнів завжди залежить від коду нижчих рівнів, а не навпаки.
- Застосування принципів зворотного зв'язку та ін'єкції залежностей для забезпечення гнучкості та розширюваності архітектури.

Архітектура Onion зазвичай містить такі шари:

- Domain Layer - центральний шар, який містить основну бізнес-логіку та сутності програми. Він не має знань про зовнішні шари та служить основою для інших шарів.
- Domain Services Layer - містить додаткові сервіси та логіку, які стосуються доменного шару, такі як валідація, обчислення та обробка даних.
- Infrastructure Layer - забезпечує підключення до зовнішніх ресурсів, таких як бази даних, файлові системи або веб-сервіси. Цей шар залежить від доменного шару та реалізує інтерфейси, визначені в доменному шарі.
- Application Layer - містить логіку для оркестрації операцій, викликів доменних сервісів та управління процесами роботи з даними. Цей шар може також включати моделі DTO (Data Transfer Objects) та мапери для конвертації між доменними сутностями та DTO.
- Presentation Layer - відповідає за взаємодію з користувачами та представлення даних. Це можуть бути веб-інтерфейси, мобільні додатки або настільні програми. Презентаційний шар взаємодіє з шаром застосунків для отримання та надсилання даних, а також відображення даних для користувачів.
- Шар тестування (Testing Layer): Містить юніт-тести, інтеграційні тести та інші тести для перевірки різних аспектів системи. Цей шар може бути вбудований в кожен з інших шарів, або ж відокремлений в окремий проект.

Приклад схеми архітектури такого додатку зображений на рис. 2.1

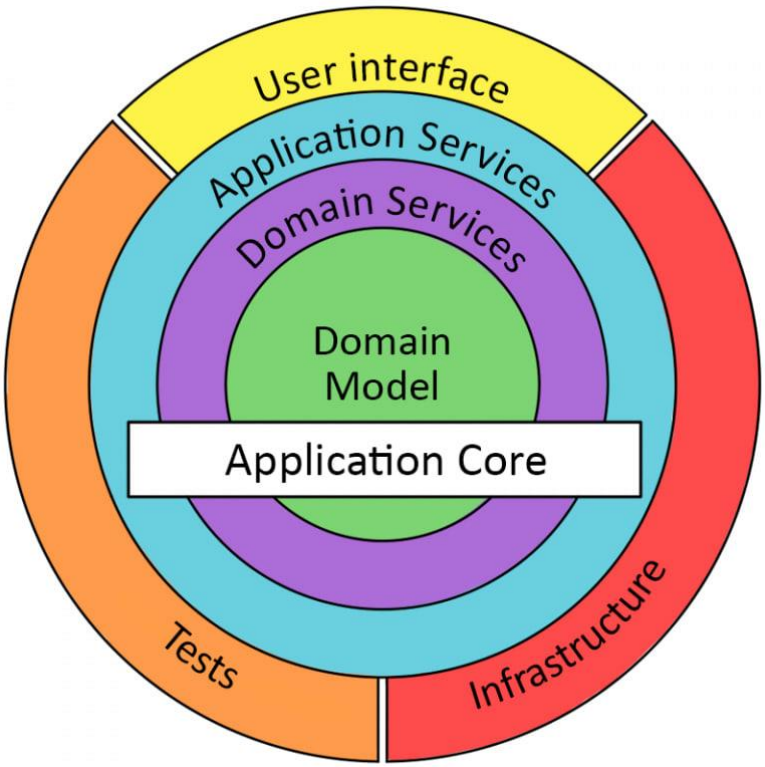


Рисунок 2.1 – Схема Onion architecture.

Переваги Onion Architecture:

- розподіл відповідальності - ця архітектура забезпечує чіткий розподіл відповідальності між різними шарами, що сприяє підтримці та розширенню програмного забезпечення;
- відмінна модульність - Onion Architecture підтримує модульність та гнучкість, що дозволяє легко замінювати окремі компоненти або реалізації без впливу на інші частини системи;
- сприяє тестуванню - завдяки принципам зворотного зв'язку та ін'єкції залежностей, Onion Architecture полегшує тестування програмного забезпечення на різних рівнях;

- підтримка Agile методологій - ця архітектура дозволяє легко розбити роботу над проектом на менші частини, що сприяє використанню Agile методологій розробки, таких як Scrum або Kanban, для більш ефективної роботи команди;
- незалежність від технологій - Onion Architecture не прив'язана до конкретних технологій або фреймворків, що робить її гнучкою та адаптованою до різних вимог розробки програмного забезпечення;
- простота вивчення та розуміння - архітектура Onion базується на простих принципах розробки та розподілу відповідальностей, що сприяє легкому вивченню та розумінню нею новими членами команди.

Деякі недоліки Onion Architecture:

- потрібно дотримуватися принципів - для досягнення всіх переваг Onion Architecture, розробники повинні дотримуватися її принципів. Це може вимагати додаткового часу та зусиль при розробці та підтримці програмного забезпечення;
- відносно складніша структура - в порівнянні з деякими іншими архітектурними підходами, Onion Architecture може мати відносно складнішу структуру, особливо для менших проектів, що може призвести до збільшення кількості коду та складності управління проектом;

Незважаючи на деякі недоліки, Onion Architecture вважається відмінним вибором для проектів різного рівня складності, що потребують чіткого розподілу відповідальності, високої модульності та гнучкості. Застосування Onion Architecture допомагає розробникам створювати стійкі, легко розширювані та легко підтримувані програмні продукти.

2.8. Огляд DDD

Domain-Driven Design (DDD) - це підхід до розробки програмного забезпечення, який спрямований на створення високоякісних та гнучких програмних систем шляхом зосередження на основній предметній області. Цей підхід був розроблений Еріком Евансом та описаний у його книзі "Domain-Driven Design: Tackling Complexity in the Heart of Software". DDD став широко відомим і популярним методом проектування програмного забезпечення.

Основні аспекти DDD:

- Убіквітарна мова - один з ключових аспектів DDD це створення єдиної мови, яку використовують розробники та доменні експерти для спілкування та обговорення предметної області. Убіквітарна мова включає терміни, які є спільними для всієї команди та відображають ключові концепції та моделі домену;
- створення доменних моделей - DDD спонукає розробників створювати доменні моделі, які відображають реальний світ та відтворюють основні концепції та правила предметної області. Доменні моделі допомагають розробникам краще розуміти та розробляти систему;
- розбиття системи на підсистеми - DDD рекомендує розбивати систему на менші підсистеми, які відповідають за певні аспекти предметної області. Це допомагає створювати модульні, гнучкі та легко розширювані програмні системи;
- використання шаблонів проектування - DDD включає використання ряду шаблонів проектування, таких як агрегати, сутності, значення, репозиторії та сервіси домену, для створення добре структурованої та зрозумілої архітектури програмного забезпечення. Ці шаблони допомагають розробникам ефективно моделювати та інкапсулювати логіку домену;

- відділення інфраструктурних аспектів - DDD акцентує увагу на важливості відділення інфраструктурних аспектів, таких як зберігання даних, від доменної логіки. Це дозволяє забезпечити гнучкість та незалежність компонентів системи, а також спрощує тестування та підтримку;
- фокус на колаборації - DDD ставить акцент на співпраці між розробниками та доменними експертами, що сприяє кращому розумінню предметної області та розробці більш точних та ефективних програмних рішень.

DDD допомагає розробникам створювати масштабовані, гнучкі та високоякісні програмні системи, які відображають реальний світ та враховують основні правила та концепції предметної області. Цей підхід може бути використаний у різних галузях та контекстах розробки програмного забезпечення, включаючи веб-розробку, розробку мобільних додатків, розподілені системи та інше.

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, стандартна автентифікація - для користувачів. [9] Автентифікація та авторизація є важливими аспектами розробки безпечних веб-додатків та API. API Key - це один із способів автентифікації, який дозволяє забезпечити контроль доступу до API.

- API Key: API Key - це унікальний ідентифікатор, який генерується та надається розробникам для автентифікації та ідентифікації користувачів або додатків, що використовують API. Зазвичай, API Key є довгим рядком символів, який містить букви та цифри. API Key може бути використаний як для публічних, так і для приватних API.

- Автентифікація: Автентифікація з допомогою API Key полягає у передачі ключа разом із запитом до API. Це може бути зроблено через заголовки HTTP, параметри URL або інші способи передачі даних. При отриманні запиту, API перевіряє наявність та відповідність API Key, і якщо ключ вірний, дозволяє доступ до ресурсів API.
- Авторизація: API Key також може використовуватися для авторизації доступу до певних ресурсів або дій. У такому випадку, API Key асоціюється з певним набором дозволів або ролей користувача, і сервер може обмежити доступ до ресурсів на основі цих дозволів.
- Обмеження API Key: Використання API Key має деякі обмеження. По-перше, API Key може бути викрадений або компрометований, що призведе до небажаного доступу до API. По-друге, API Key не містить інформації про автентифікацію користувача, тому його не можна використовувати для безпосередньої аутентифікації користувачів.
- Застосування: API Key часто використовується у випадках, коли не потрібна сильна безпека або коли він доповнює інші методи автентифікації. Деякі поширені сценарії включають:
    - Відкриті API: API Key може використовуватися для обмеження доступу до відкритих API, де не потрібна сильна аутентифікація або коли інформація не є конфіденційною.
    - Обмеження застосунків: Для контролю над тим, які додатки можуть використовувати ваше API, можна виділити окремий API Key для кожного додатка.
    - Моніторинг та аналітика: Завдяки API Key можна відстежувати використання API різними користувачами або додатками, що дозволяє забезпечити аналітику та моніторинг.
    - Альтернативи: Хоча API Key є простим способом автентифікації, він не завжди є найбезпечнішим. Для реалізації більш сильної безпеки

розгляньте інші методи автентифікації, такі як OAuth, JWT (JSON Web Token) або TLS-клієнтські сертифікати.

- Найкращі практики: Щоб забезпечити безпеку API Key, слід дотримуватися деяких найкращих практик, таких як регулярна ротація ключів, обмеження доступу до ключів лише для необхідних дій, використання HTTPS для передачі ключів та відстеження та моніторинг використання ключів.

API Key є простим та широко використовуваним методом автентифікації та авторизації, який дозволяє контролювати доступ до API на основі унікального ідентифікатора. Цей метод добре підходить для менш критичних до безпеки сценаріїв або для використання разом з іншими методами аутентифікації.

2.10. Огляд TPL Dataflow

Task Parallel Library Dataflow - це бібліотека, що дозволяє розробникам ефективно створювати складні системи обробки повідомлень та паралельного виконання завдань в .NET. TPL Dataflow включений у .NET Framework 4.5 і новіших версій, а також доступний як окремий пакет NuGet для попередніх версій.

Основні переваги TPL Dataflow:

1. легкість використання - TPL Dataflow надає високорівневий абстрактний API, що спрощує розробку паралельних програм;
2. гнучкість - TPL Dataflow дозволяє створювати складні оброблювачі повідомлень та контролювати потік даних за допомогою різних блоків, що можуть бути легко зв'язані разом;
3. висока продуктивність - TPL Dataflow розроблено для ефективного використання ресурсів, інтегрується з TPL (Task Parallel Library) для

керування паралельним виконанням завдань і може автоматично налаштовувати рівень паралелізму залежно від доступних ресурсів.

Основні компоненти TPL Dataflow:

1. блоки - це основні будівельні блоки TPL Dataflow, які представляють операції обробки повідомлень та контролюють потік даних. Блоки можуть відправляти, отримувати або трансформувати дані, а також бути відповідальними за координацію між іншими блоками. Наприклад, TransformBlock, ActionBlock та BufferBlock.
2. зв'язки - використовуються для передачі даних між блоками, забезпечуючи ефективну обробку та розподіл роботи. Зв'язки можуть бути створені вручну або автоматично за допомогою методів LinkTo.
3. Графи - об'єднують блоки та зв'язки в більші структури, що дозволяє створювати складні структури обробки даних та координувати потік даних між різними частинами програми. Графи допомагають розробникам створювати модульні, перевикористовувані компоненти для різних сценаріїв обробки даних.
4. виконання - TPL Dataflow оптимізований для ефективного виконання завдань, інтегрується з Task Parallel Library та використовує паралельність для досягнення високої продуктивності. Він автоматично адаптується до ресурсів системи, забезпечуючи оптимальний рівень паралелізму та відгук на зміни в навантаженні.
5. контроль та моніторинг - TPL Dataflow надає інструменти для контролю та моніторингу потоків даних, що дозволяє розробникам відстежувати стан блоків, зв'язків та графів, а також ідентифікувати проблеми з продуктивністю та надавати можливість діагностики.

У цілому, TPL Dataflow - це потужна та гнучка бібліотека для побудови високопродуктивних, паралельних програм на платформі .NET. Вона відкриває нові можливості для розробки ефективних систем обробки повідомлень та паралельного виконання завдань, спрощуючи розробку та підтримку таких систем.

2.11. Огляд Amazon Lambda

AWS Lambda, - це безсерверний комп'ютинговий сервіс, що надається компанією Amazon Web Services (AWS). Він дозволяє розробникам створювати та запускати функції без необхідності управління інфраструктурою серверів. AWS Lambda автоматично масштабується, забезпечуючи виконання коду у відповідь на події, такі як зміна файлів у S3, повідомлення в Amazon SNS або запити через Amazon API Gateway.

Основні характеристики AWS Lambda:

1. Безсерверність - ви зосереджуєтеся на написанні коду, а AWS Lambda забезпечує управління ресурсами сервера, що звільняє вас від завдань з розгортання та масштабування.
2. Автоматичне масштабування - AWS Lambda масштабується автоматично, забезпечуючи кілька паралельних виконань функцій відповідно до вхідного навантаження.
3. Оплата за використання - ви платите лише за час виконання вашого коду, без додаткових витрат на розгортання сервера.
4. Підтримка множини мов програмування - AWS Lambda підтримує ряд мов програмування, таких як Python, Node.js, Java, C#, Go та Ruby, дозволяючи розробникам використовувати знайомі мови та інструменти.

5. Інтеграція з іншими послугами AWS - Lambda може легко інтегруватися з іншими послугами AWS, такими як S3, DynamoDB, API Gateway, SNS, SQS та Kinesis, що спрощує створення багатофункціональних додатків.
6. Подієвий відгук - Lambda може реагувати на події з різних джерел, таких як зміни файлів, HTTP-запити або сповіщення, що сприяє створенню швидко реагуючих додатків.

Загалом, Amazon Lambda - це потужний та гнучкий сервіс безсерверної архітектури, який дозволяє розробникам прискорити розробку та впровадження додатків, зосереджуючись на бізнес-логіці, а не на управлінні інфраструктурою серверів. Він масштабується автоматично та інтегрується з рядом інших послуг AWS. Використовуючи Amazon Lambda, розробники можуть створювати надійні, швидкі та економічно ефективні додатки, які легко реагують на події та забезпечують швидке відгук на запити користувачів.

2.12. Огляд Amazon SQS

Amazon Simple Queue Service (SQS) - це повністю керований сервіс черг повідомлень, який дозволяє розробникам безпечно і без зайвих зусиль координувати роботу компонентів розподілених систем. SQS розроблений для легкої інтеграції з іншими сервісами Amazon Web Services (AWS), що забезпечує надійне, масштабоване та просте використання рішення для передачі повідомлень.

Основні особливості AWS SQS:

1. Масштабованість: SQS автоматично масштабується, забезпечуючи необхідний рівень пропускної здатності для передачі повідомлень між компонентами системи.

2. Надійність: SQS зберігає повідомлення на декількох серверах, що гарантує їх доставку навіть у разі відмови окремих серверів.
3. Затримка: SQS дозволяє налаштувати затримку для повідомлень, що дає можливість гнучкого керування розподілом робочого навантаження.
4. Відмовостійкість: SQS може перерозподіляти навантаження між серверами, що забезпечує стабільну роботу системи при відмові одного або кількох серверів.
5. Безпека: SQS підтримує шифрування повідомлень, а також дозволяє налаштувати контроль доступу для різних користувачів та груп.

AWS SQS пропонує два типи черг: стандартні та черги зі збереженням порядку (FIFO). Стандартні черги забезпечують найвищу пропускну здатність, однак повідомлення можуть бути доставлені не в порядку відправлення. Черги зі збереженням порядку гарантують доставку повідомлень у порядку їх надходження, але мають меншу пропускну здатність порівняно зі стандартними чергами. AWS SQS легко інтегрується з іншими сервісами AWS, такими як Amazon S3, Amazon EC2, Amazon Lambda та іншими. Це дає розробникам можливість створювати складні та масштабовані архітектури без додаткового навантаження на управління інфраструктурою.

Amazon Simple Queue Service (SQS) є потужним інструментом для розробки розподілених систем, що дозволяє безпечно, масштабовано та надійно передавати повідомлення між компонентами. Завдяки своїй гнучкості та інтеграції з іншими сервісами AWS, SQS стає оптимальним вибором для передачі повідомлень в сучасних додатках та архітектурах.

2.13. Огляд AWS EventBridge

AWS EventBridge – це безшовний мостик подій, який дозволяє інтегрувати різноманітні додатки, системи та сервіси на основі подій. Він базується на популярному сервісі Amazon Web Services (AWS) і створений для спрощення обробки подій та оркестрації відкликань на ці події. EventBridge забезпечує зв'язок між вашими додатками та AWS сервісами, сторонніми сервісами та власними додатками.

Основні характеристики EventBridge:

- Широкий вибір джерел подій - EventBridge підтримує події від багатьох джерел, включаючи AWS сервіси (наприклад, S3, Lambda, EC2), сторонні додатки (наприклад, PagerDuty, Datadog, Zendesk) та власні додатки.
- Масштабованість і висока пропускна здатність - завдяки оптимізації EventBridge від AWS, сервіс масштабується автоматично, щоб обробляти велику кількість подій, які надходять від різних джерел.
- Гнучкість і кастомізація - EventBridge дозволяє створювати правила, які відповідають певним шаблонам подій або фільтрують події за допомогою атрибутів. Це дозволяє розробникам налаштовувати реагування на різні типи подій та контролювати, які події потраплять до кінцевих сервісів.
- Безпека і надійність - EventBridge забезпечує надійність та безпеку передачі подій. Він використовує AWS Identity and Access Management (IAM) для управління доступом до ресурсів та авторизації користувачів.
- Сумісність - EventBridge можна легко інтегрувати з іншими AWS сервісами, такими як Lambda, Step Functions, SNS, SQSта інші. Це спрощує розробку архітектур на основі подій та покращує міжсервісну комунікацію.

- Моніторинг і логування - завдяки вбудованому моніторингу та логуванню, розробники можуть легко відстежувати і аналізувати роботу EventBridge. AWS CloudWatch дозволяє відстежувати метрики, пов'язані з подіями, такі як кількість подій, час обробки та успішність доставки. Це допомагає виявляти та вирішувати проблеми з продуктивністю та надійністю.
- Інтеграція з іншими сервісами - EventBridge може співпрацювати з іншими AWS сервісами, а також зі сторонніми сервісами і системами, що відповідають протоколам HTTP, HTTPS, MQTT та іншим. Це робить його універсальним та гнучким рішенням для побудови різноманітних архітектур.
- SaaS інтеграції - EventBridge підтримує інтеграцію з різними SaaS-платформами, що дозволяє обмінюватися даними та подіями між різними SaaS-додатками та AWS сервісами. Це полегшує розробку та підтримку рішень, які використовують сторонні сервіси.

Завдяки цим можливостям, AWS EventBridge є потужним, гнучким та надійним рішенням для обробки подій та інтеграції різних додатків, систем і сервісів. Він допомагає розробникам створювати архітектури на основі подій, реагувати на зміни у середовищі, оптимізувати міжсервісну комунікацію та автоматизувати бізнес-процеси. Користуючись AWS EventBridge, команди можуть підвищити ефективність своїх додатків, забезпечити масштабованість та зменшити час на розробку та випуск нових продуктів.

2.14. Огляд pipeline architecture

Архітектура конвеєра (pipeline architecture) - це архітектурний підхід, який розбиває обробку даних або завдань на послідовність кроків, кожен з яких виконується незалежно від інших. Цей підхід сприяє ефективності,

масштабованості та модульності, оскільки кожен крок (або стадія) може оптимізуватися і розвиватися незалежно.[4]

Основні характеристики архітектури конвеєра:

1. Розділення обробки на стадії: завдання розбиваються на менші, послідовні кроки, які виконуються один за одним. Кожна стадія конвеєра забезпечує певну функціональність, таку як читання даних, обробка або запис.
2. Незалежні стадії: кожна стадія конвеєра може розроблятися, оптимізуватися та масштабуватися незалежно від інших стадій. Це полегшує відлагодження та забезпечує гнучкість в розробці.
3. Паралельне виконання: архітектура конвеєра дозволяє паралельне виконання стадій, що прискорює обробку даних та забезпечує кращу масштабованість.
4. Лінійність: архітектура конвеєра забезпечує лінійний потік даних від початку до кінця, що полегшує розуміння та відлагодження процесу обробки.

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

3.1. Призначення системи

Система призначена для використання через клієнт у вигляді телеграм-бота, а також для інтеграції з іншими застосунками за допомогою RESTful API. Вона дозволяє отримувати інформацію про рівень забруднення повітря на основі координат розташування об'єкту, а також транслювати дані про станції моніторингу та передавати інформацію про виміри асоційовані із певною станцією.

3.2. Цілі створення системи

- Забезпечення доступу до актуальної інформації про якість повітря - основна мета системи полягає у наданні точних, своєчасних і корисних даних про якість повітря, що допомагає громадянам, організаціям та урядовим структурам у прийнятті обґрунтованих рішень щодо охорони довкілля та здоров'я.
- Сприяння екологічній освіті та обізнаності - система має на меті збільшити обізнаність населення про проблеми забруднення повітря та їх вплив на здоров'я та довкілля. Це може стимулювати громадян до вжиття екологічно відповідальної поведінки та підтримки ініціатив з поліпшення якості повітря.
- Підтримка прийняття політичних рішень та регулювання - дані про якість повітря можуть слугувати основою для розробки та впровадження ефективних політичних рішень, стратегій та нормативів, спрямованих на зменшення забруднення повітря та покращення здоров'я населення.

- Агрегації даних із різних джерел доступу до станцій моніторингу для отримання якомога повнішої інформації про стан повітря в навколишньому середовищі

3.3. Вимоги до системи

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

У системі повинні бути передбаченій наступні функціональні можливості:

- Ведення журналів подій у межах системи: Система повинна автоматично зберігати всі події та дії, які відбуваються в процесі її роботи. Це допоможе відстежувати зміни, виявляти та усувати проблеми, а також покращувати безпеку.
- Надання RESTful інтерфейсу для доступу до ресурсів: Інтерфейс системи має бути простим у використанні та забезпечувати доступ до різних ресурсів через стандартні HTTP-методи. Це спрощує інтеграцію з іншими системами та розробку клієнтських додатків.
- Авторизація та автентифікація користувачів, які використовують API: Система має мати захист від несанкціонованого доступу до даних і функціоналу, забезпечуючи перевірку прав користувачів та контроль доступу до різних ресурсів.
- Можливість швидкого додавання нових джерел даних без змін у системі: Система повинна бути гнучкою і масштабованою, щоб легко інтегрувати нові джерела даних про якість повітря або розширювати можливості обробки і аналізу інформації.
- Підтримка розгортання у хмарному середовищі: Система має підтримувати хмарне розгортання, що дозволяє забезпечити високу доступність,

еластичність та швидкість відгуку, а також зменшити витрати на інфраструктуру.

- Горизонтальне масштабування системи з урахуванням окремих модулів: Архітектура системи повинна передбачати можливість горизонтального масштабування, що дозволить оптимально розподіляти ресурси та навантаження між різними моду

РОЗДІЛ 4. РЕАЛІЗАЦІЯ СИСТЕМИ

4.1. Огляд доступних підходів до проектування системи

Після завершення визначення вимог до проектованої системи, слід зосередитися на виборі архітектурних методів для створення рішення. Оскільки система повинна інтегруватись з хмарними середовищами та підтримувати горизонтальне масштабування, її архітектура має відповідати сучасним хмарним рішенням. Мережеві додатки мають два типи архітектур: монолітну та мікросервісну.

Монолітна архітектура - це єдине рішення з однією кодовою базою, що містить всі сервіси додатку. Цей підхід дозволяє швидко розробляти додатки за допомогою повторного використання коду, але зумовлює складність у підтримці та розширенні, а також у горизонтальному масштабуванні.

Мікросервісна архітектура - це модульний підхід до організації системи, який передбачає розділення додатку на незалежні сервіси з власними кодовими базами. Це полегшує підтримку, розширення та масштабування, а також забезпечує кращу доступність та відновлюваність системи.

Враховуючи нерівномірне навантаження на складові частини та потребу інтеграції з хмарними середовищами, було вирішено використовувати мікросервісну архітектуру для організації системи.

4.2. Опис основних складових частин системи

Відповідно до функціональних вимог і вибраної архітектурної стратегії, була розроблена архітектура інформаційної системи моніторингу якості повітря. В таблиці 4.1 представлено призначення компонентів системи.

Таблиця 4.1 – Призначення компонентів системи

Компонент	Призначення
AWS EventBridge	Місток подій, який дозволяє інтегрувати різноманітні додатки, системи та сервіси на основі подій. Використовується задля ініціації виклику serverless лямбда функцій що виступають в ролі скраберів даних. В теперішній імплементації системи створено правило що керує періодичність старту за допомогою cron виразу.
AWS Lambda	Компонент обчислень без сервера, Lambda автоматично масштабується відповідно до навантаження і виконує код тільки тоді коли це необхідно. Провайдери даних станцій моніторингу написані з використанням AWS Lambda.
AWS CloudWatch	Централізоване рішення для моніторингу AWS, використовується для збирання, відстежування журналів подій. Використовується для отримання подій та журналу виконання провайдерів даних та черг повідомлень.
AWS SQS	Сервіс черги повідомлень. Використовується для того щоб виміри із станцій уже в стандартизованому вигляді передавались на подальше опрацювання відповідно до pipeline architecture.
AirSnitch.Worker	Сервіс що створений для швидкого опрацювання даних що надходять із черги і складає їх в MongoDB ReplicaSet

MongoDB ReplicaSet	Це група MongoDB серверів, які працюють разом для забезпечення високої доступності та додаткової надійності даних. Реплікація даних між серверами допомагає уникнути втрати даних у разі відмови одного сервера та забезпечує більшу продуктивність завдяки розподілу навантаження на читання між членами набору реплік. Використовується як основне сховище даних у системі.
AirSnitch.API	Сервер що відповідальний за REST API звертається за даними до бази даних у вигляді MongoDB
AirQualityBot	Сервер що відповідальний за функціональність телеграм-бота, сам він використовує REST API що надає AirSnitch.API
Grafana	Відкрите ПЗ для візуалізації моніторингу та аналізу метрик. Наразі отримує інформацію щодо метрик екземплярів AirSnitch.Worker та AirSnitch.API

Для розгортання системи використовуються AWS Console для створення та керуванням сервісами AWS, інші компоненти, що не є частиною AWS обгорнуті в Docker-контейнер і можуть розвертатись в хмарній інфраструктурі в залежності від реалізації оркестратора Kubernetes. На рисунку 4.1 зображена схема архітектури системи.

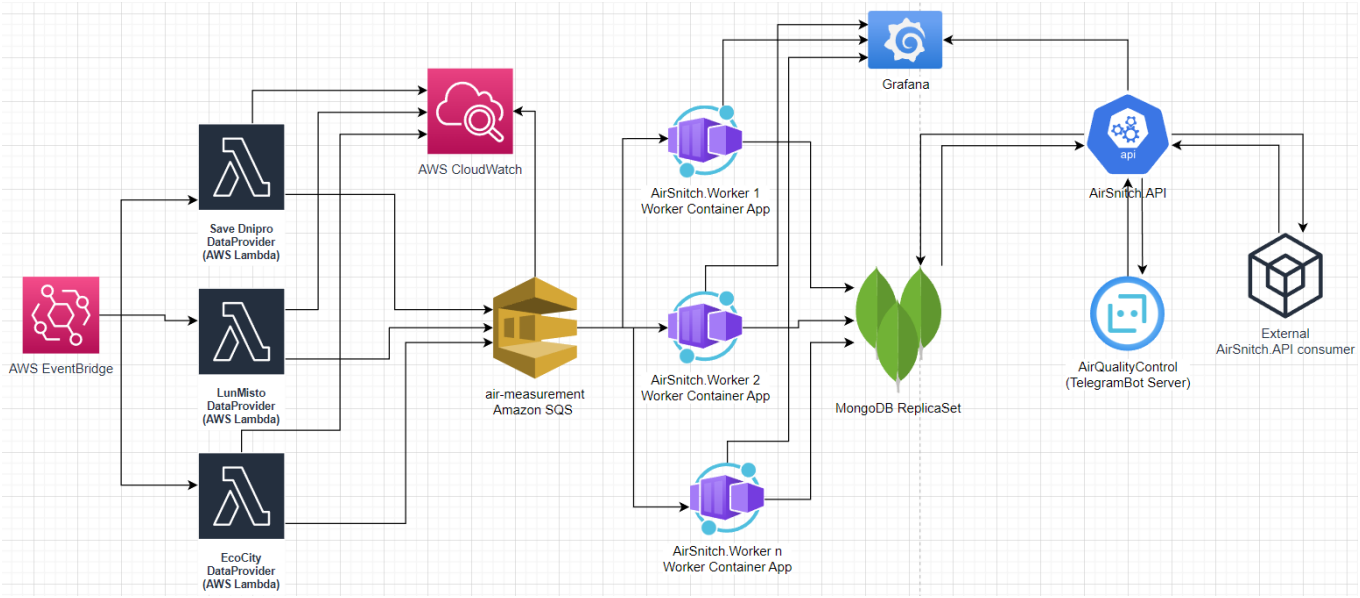


Рисунок 4.1 – Схема архітектури системи

4.3. Опис реалізації компонентів AWS Lambda

В ході проектування архітектури системи було вирішено зупинитись на serverless підході у вигляді AWS Lambda переваги такого архітектурного підходу:

- не потрібно керувати серверами, Lambda розгортає та запускає код автоматично;
- Lambda автоматично масштабується, відповідно до навантаження, для оптимальної продуктивності;
- Lambda може бути запущений відповідно до різних подій, таких як зміни у S3, запити API Gateway, повідомлення SNS або відповідно до розкладу (наприклад, за допомогою CloudWatch Events);
- Lambda підтримує декілька мов програмування, таких як Python, Node.js, Java, Ruby, Go та .NET.

Особливо хотілось би виділити підтримку різних мов програмування, таким чином для написання провайдерів даних можна використовувати будь-яку із підтримуваних мов, тобто цей компонент системи є language-agnostic.

Теперішня реалізація налічує три провайдери даних які реалізовані з використанням Lambda – це SaveDnipro, Lun Misto Air, та Eco City. На рисунку 4.2 зображена кодова карта провайдерів даних для станцій моніторингу відповідно.

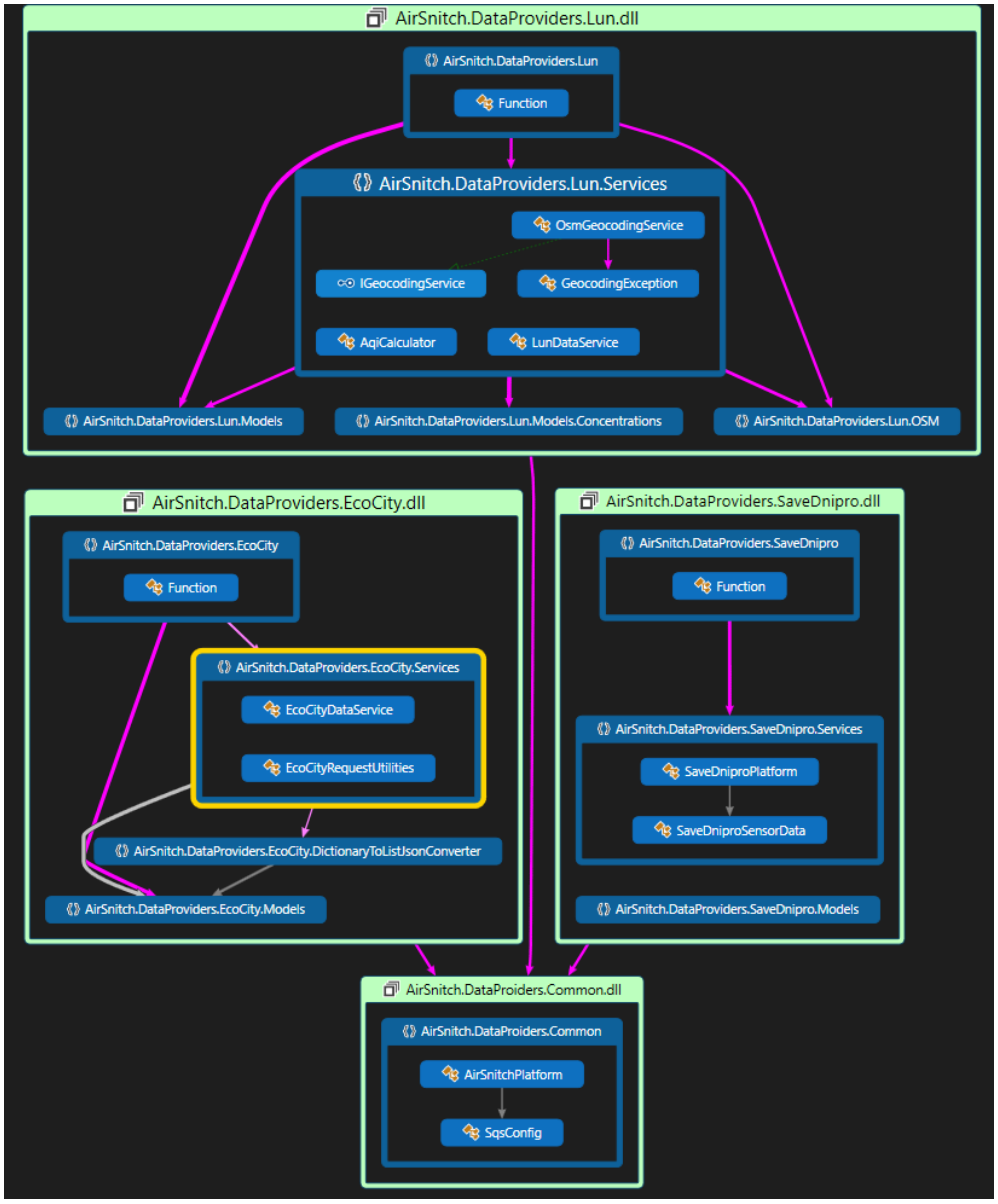


Рисунок 4.2 – Кодова карта провайдерів даних

Особливістю написання Lambda є те що єдина точка входу це є відповідний метод, який треба задекларувати в спеціальному config у моєму випадку для Lun Misto Air це AirSnitch.DataProviders.Lun:: AirSnitch.DataProviders.Lun.Function:: FunctionHandler, в який можна передати параметри функції та відповідно почати опрацювання логіки для отримання даних. Функції можуть повертати значення або виступати у ролі процедур що не повертають значення. У випадку теперішньої реалізації функції не повертають нічого, натомість в процесі роботи вони надсилають дані до черги AWS SQS у певному форматі. Для стандартизації формату повідомлень було створено рішення AirSnitch.SDK та опубліковано у Nuget package manager, що містить усі контракти даних які використовуються для роботи із чергою та вимірами станцій. Отриманий пакет є CLSComplient що означає - може бути використана мовою, що підпадає під Common Language Specification (CLS) що визначає функції, які має підтримувати будь-яка мова орієнтована на .Net, таким чином цей пакет може бути використаний і в F# і в Visual Basic .Net також. На рисунку 4.3 зображено кодову карту AirSnitch.SDK

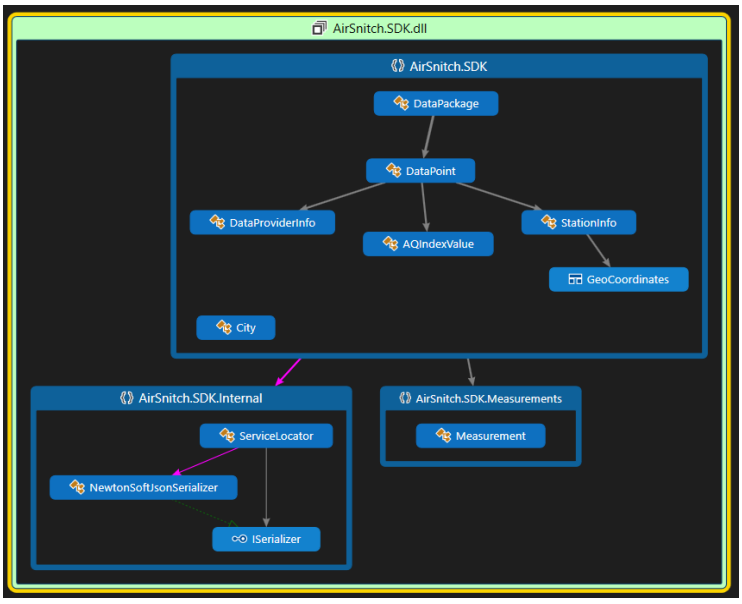


Рисунок 4.3 – кодова карта Nuget Package AirSnitch.SDK

4.4. Опис реалізації AirSnitch.Worker

AirSnitch.Worker виступає в ролі сервісу що створений для швидкого опрацювання даних що надходять із черги AWS SQS і складає їх в MongoDB ReplicaSet. Критично важливим для цього компоненту системи є швидкодія, саме через це було обрано реалізувати його роботу з використанням TPL Dataflow - це бібліотека для .NET, яка спрощує розробку паралельних та асинхронних програм з використанням ідіоми "акторів" та обміну даними між ними через буферизовані блоки. Бібліотека дозволяє створювати складні керівництва потоків і об'єднувати їх у вищі абстракції, що полегшує розробку масштабованих, відповідних та легко розширюваних програм. TPL Dataflow забезпечує використання обмежених ресурсів та підтримку потоків, що розподіляються на кілька ядер або процесорів. На рисунку 4.4 зображено конвеєр за яким рухаються дані після отримання із черги AWS.

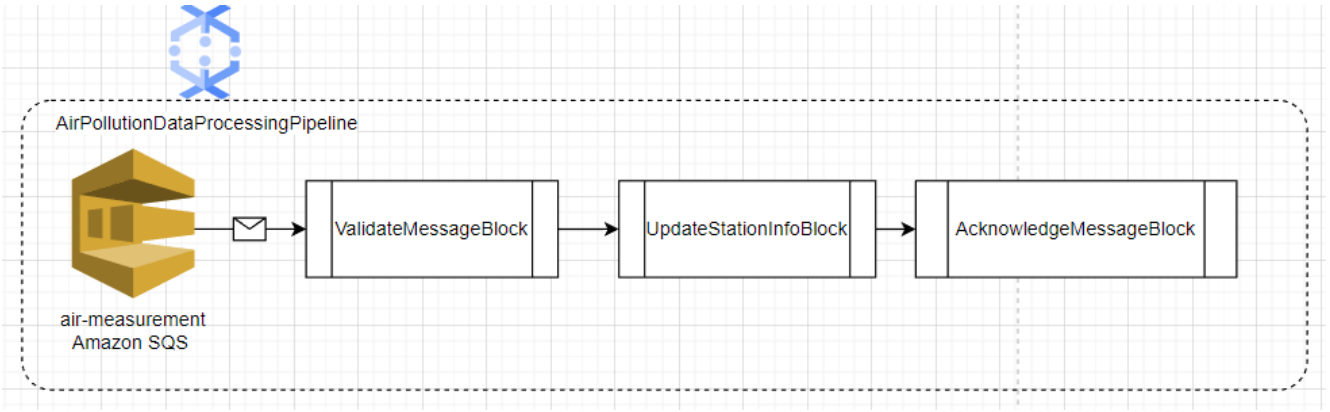


Рисунок 4.4 TPL Dataflow Blocks AirSnitch.Worker

ValidateMessageBlock – перевіряє вхідне повідомлення на коректність десереалізує та віддає сформований об’єкт станції моніторингу на наступний етап.

UpdateStationBlock - етап що перевіряє чи є така станція, якщо нема, то створює її, та оновлює дані про виміри асоційовані із нею.

AsknowledgeMessageBlock – етап який сигналізує чергу AWS про успішне опрацювання повідомлення та можливість його видалення.

Кодова карта AirSnitch.Worker зображена на рисунку 4.5.

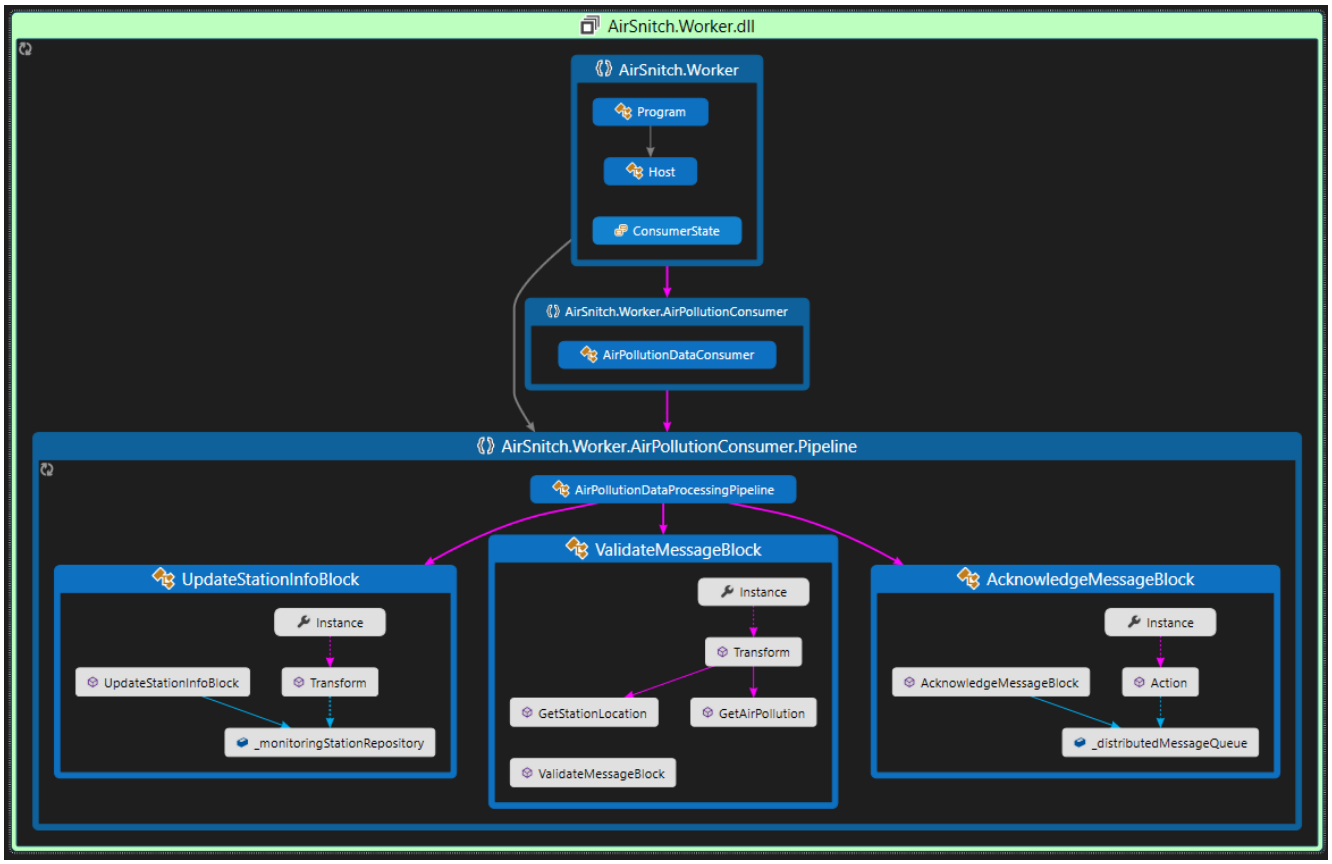


Рисунок 4.5 кодова карта AirSnitch.Worker

Усі операції в межах dataflow виконуються в конкурентному асинхронному режимі, результатом кінцевим є зміна даних асоційованих із вимірами всередині MongoDB ReplicaSet, кожен воркер що опрацьовує чергу пише дані саме у цю базу даних. Особливість використання такої конфігурації БД через її високопродуктивну роботу в режимі читання через доступ на читання з її реплік і внутрішнього балансування навантаження між ними, що є критично важливим для навантаженого API.

4.5. Опис реалізації AirSnitch.API

Тип архітектури сервісу - Onion architecture. Керуючись підходом DDD, предметною областю виступає моніторинг якості повітря. Спроектовано рішення що має 4 абстрактні рівні: рівень домену або ядра який не залежний від інших та усі інші залежать від нього, абстрактний рівень інфраструктури, у якому визначені інтерфейси для сервісів взаємодії із доменом, рівень інфраструктури у якому реалізовані інтерфейси та логіка, що описана в домені та абстрактному рівні інфраструктури, та рівень представлення API – публічний доступ до ресурсів. Фінальна архітектура складається з чотирьох рівнів (рис.4.6):

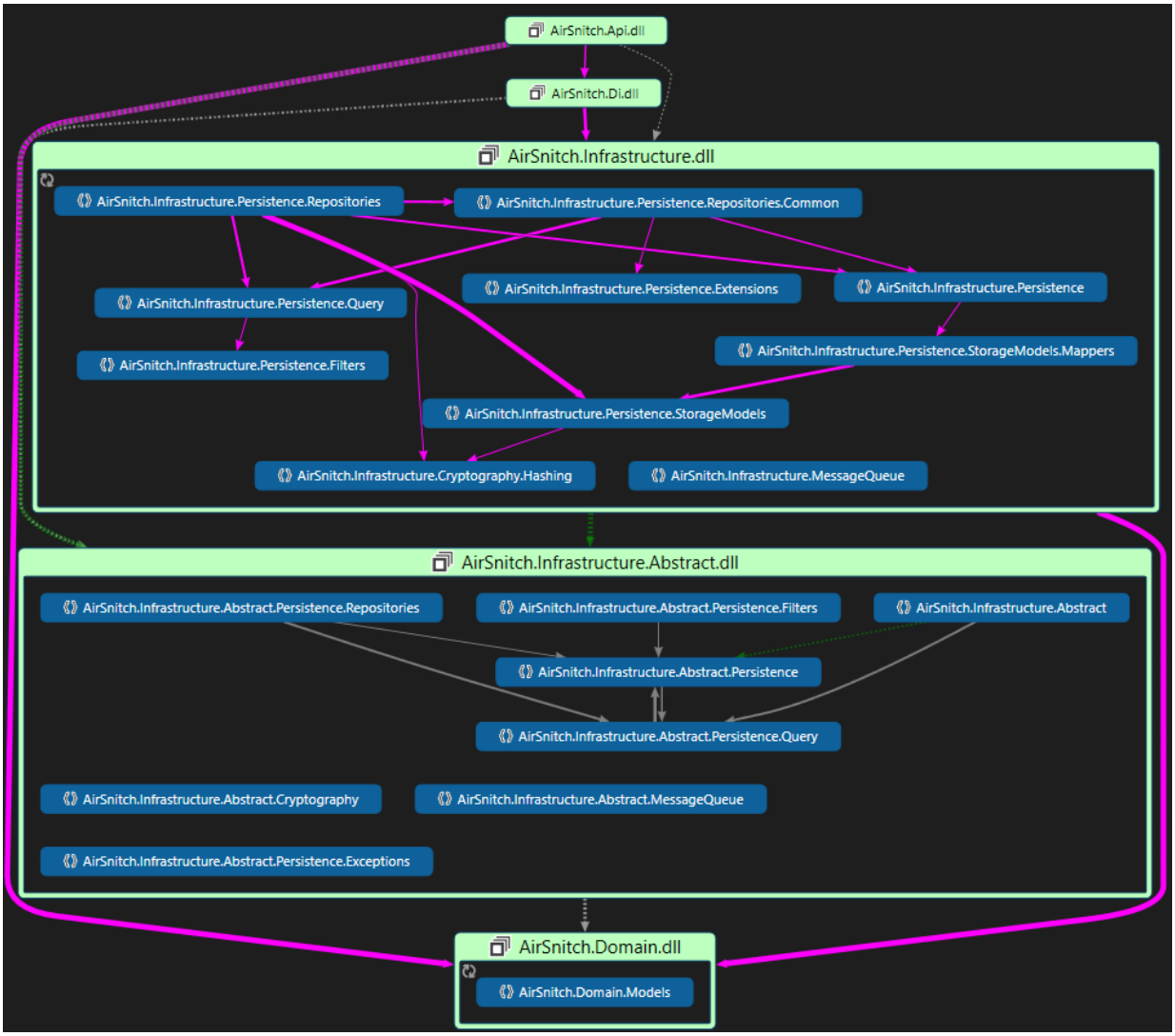


Рисунок 4.6 – кодова карта архітектури системи

AirSnitch.Domain містить доменні моделі.

AirSnitch.Infrastructure.Abstract включає інтерфейси репозиторіїв, маперів та абстрактних моделей запитів.

AirSnitch.Infrastructure відповідає за реалізацію персистентності згідно з контрактами з AirSnitch.Infrastructure.Abstract, включаючи з’єднання з базою даних, реалізацію репозиторіїв доменних моделей, моделі сховища MongoDB та мапери. Також налаштована робота з логером та конфігурація для http-клієнта, який використовується у провайдерах даних.

AirSnitch.Api представляє зовнішній рівень, де розміщена логіка побудови посилань для HATEOAS, контролери, сервіси, автентифікація та авторизація на основі API ключа. Для побудови пов'язаних ресурсів використано представлення ресурсів у вигляді графа.

4.6. Опис реалізації AirQualityControl

Ця частина системи призначена для створення клієнта у вигляді телеграм бота що б надавав інформацію про станції моніторингу відповідно до геолокації користувача. Для забезпечення даними додаток звертається до AirSnitch.API. На рисунку 4.7 зображено кодову карту AirQualityControl. Під час використання боту потрібно буде надати доступ до геолокації для отримання точки розташування користувача далі він передається на сервер для знаходження найближчої станції і відповідно отримання значення індексу AQI. Також на стадії тестування регулярні сповіщення по якійсь із станцій на яку користувач підпишеться, повідомлення о 7 ранку та 7 вечора будуть містити інформацію щодо індексу якості повітря по певній станції.

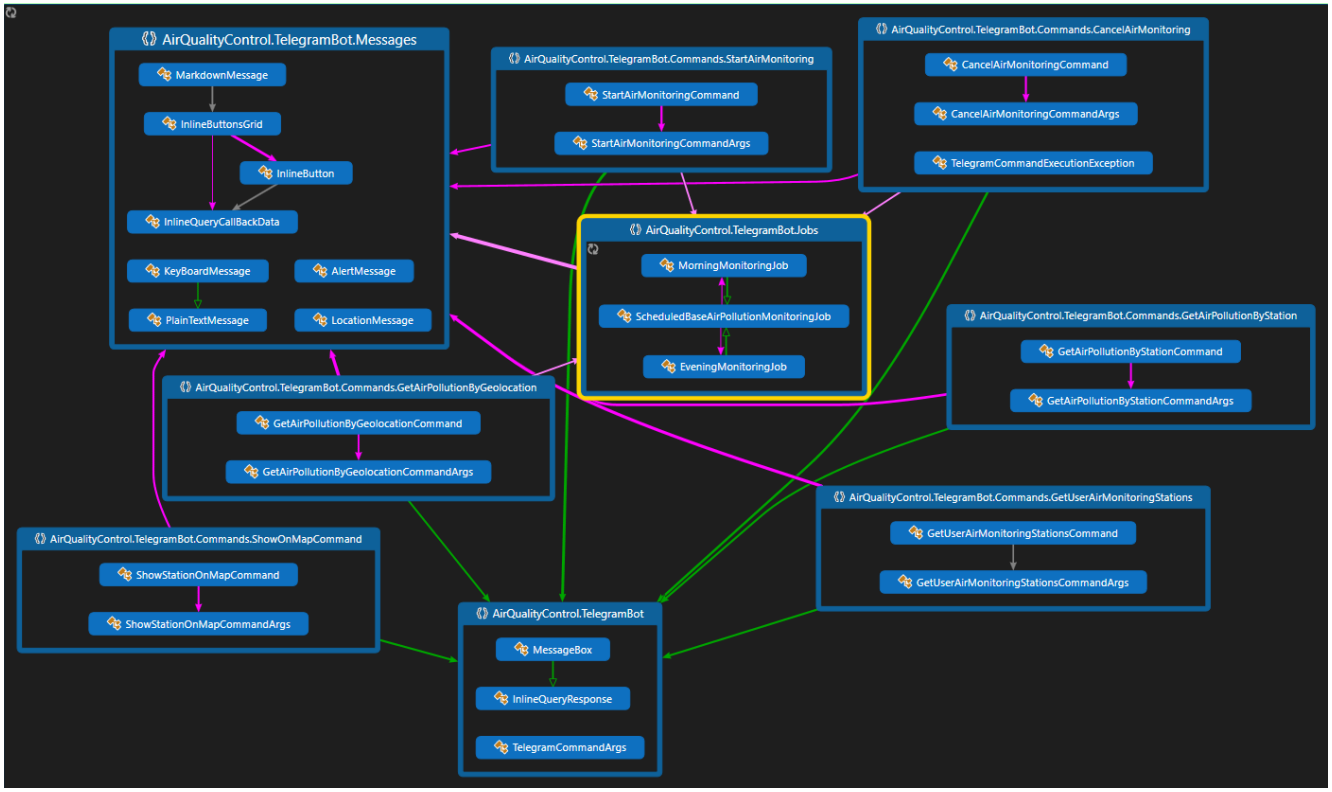


Рисунок 4.7 - кодова карта AirQualityControl

4.7. Моніторинг за системою

Для моніторингу за системою використовуються Grafana та AWS CloudWatch. Використання одразу двох ресурсів обумовлене тим, що нативні сервіси Amazon підтримують CloudWatch, в той час як моніторинг компонент що написані на платформі .Net – це AirSnitch.Worker, AirSnitch.API, можна розвертати не тільки на хмарній платформі AWS, а й у аналогічних інших провайдерів хмарних послуг, тому була додана підтримка моніторингу Grafana згаданих компонент. На рисунку 4.8 зображено діаграму Network I/O по AirSnitch.Worker. Видно деякі піки протягом 5-10 хвилин це пов’язано із отриманням повідомлень та їх опрацюванням. Повідомлення надходять в чергу із відповідних провайдерів даних і як тільки там з’являються відразу підхоплюються екземплярами сервісу AirSnitch.Worker.

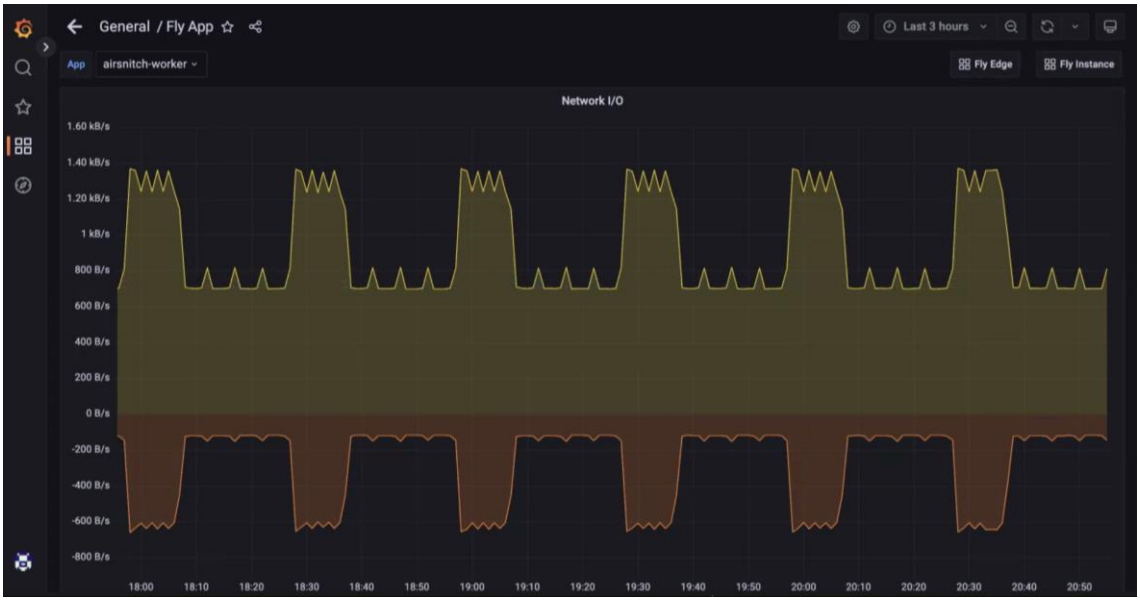


Рисунок 4.8 діаграма Network I/O

На рисунку 4.9 зображено діаграму використання оперативної пам’яті сервісом AirSnitch.Worker. Бачимо зростання утилізації ресурсу оперативної пам’яті, що обумовлене тривалим процесом імпорту даних. TPL DataFlow дає

виграш у швидкодії (паралелізму за даними), проте, натомість ми дещо більше використовуємо пам’яті.

Рисунок 4.9 використання оперативної пам’яті екземпляром AirSnitch.Worker

РОЗДІЛ 5. ІНСТРУКЦІЯ КОРИСТУВАЧА

5.1. Приклад використання API в Postman

Якщо виконати запит, повернеться повідомлення із кодом 401 (рис. 5.1.), що означає «ми не авторизовані».

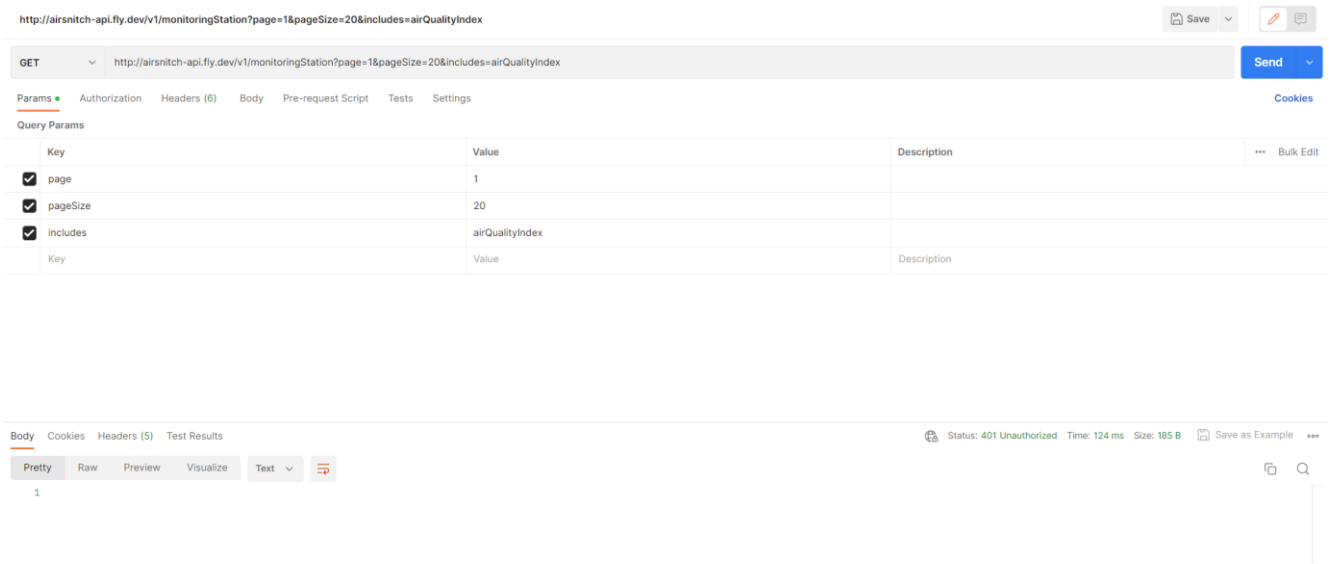


Рисунок 5.1 – результат GET /monitoringStation без авторизації

Для успішного проходження даного процесу необхідно включати API-ключ як значення у заголовку X-API-KEY при кожному запиті. На рисунку 5.2 ілюструється коректна відповідь сервера, отримана внаслідок надання користувачем ключа API. Відповідь містить такі елементи: pageSize - кількість елементів, що повертаються запитом, currentPageNumber - номер поточної сторінки, що повертається запитом, та lastPageNumber - номер останньої сторінки.

В об'єкті _links містяться відповідні посилання на наступні елементи, посилання на поточний запит, а також посилання на запит, який отримуватиме останню сторінку з даними. Всередині масиву values знаходяться елементи станцій моніторингу.

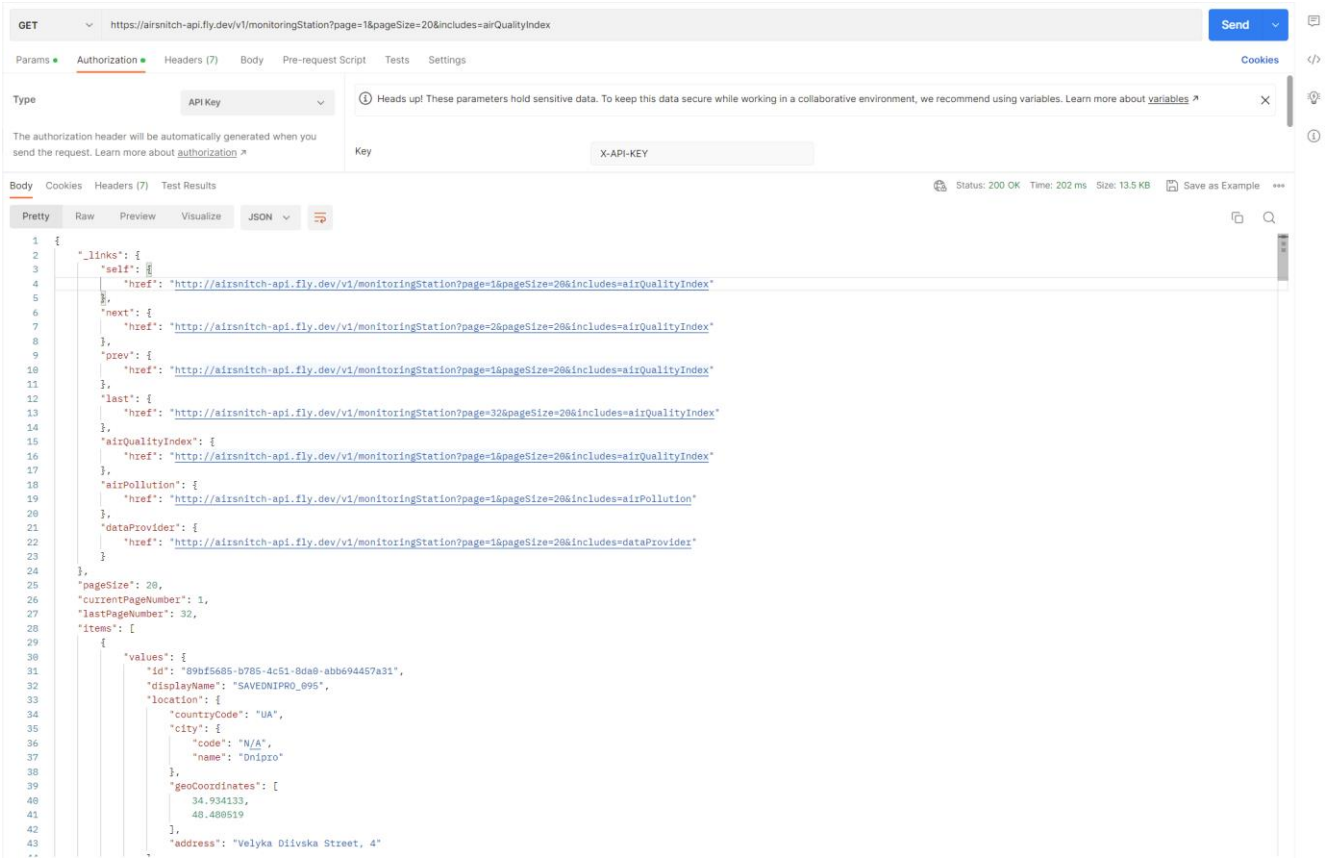


Рисунок 5.2 - результат GET /monitoringStation з авторизацією

Реалізація розширення ресурсів вимагає додавання параметра запиту, що містить включення необхідних ресурсів, які мають вигляд: include=назва ресурсу 1, назва ресурсу 2. В результаті отримується розширена модель із відповідними включеннями. Ці включення доступні через посилання в об'єкті _links, де назви полів об'єкту _links являють собою перелік можливих включень. Рисунок 5.3 демонструє приклад такого запиту із відповідними включеннями, які стосуються провайдера даних та забруднення.

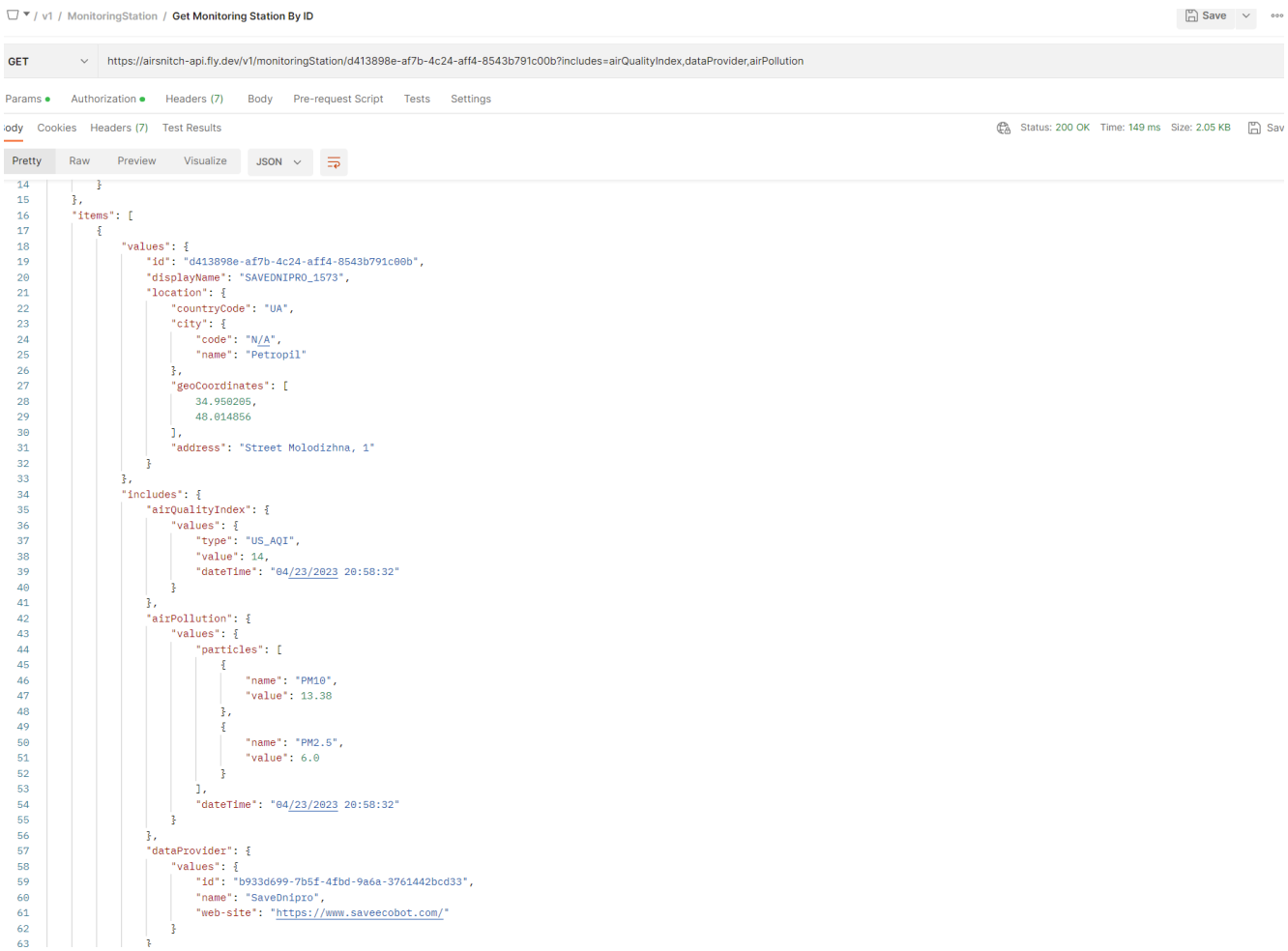


Рисунок 5.3 – демонстрація розширення ресурсу станції моніторингу.

5.2. Приклад використання telegram bot

Потрібно перейти в пошуку за https://t.me/AirQualityControlBot на мобільному додатку telegram. Перед користувачем відкриється дві кнопки як зображено на рисунку 5.4.

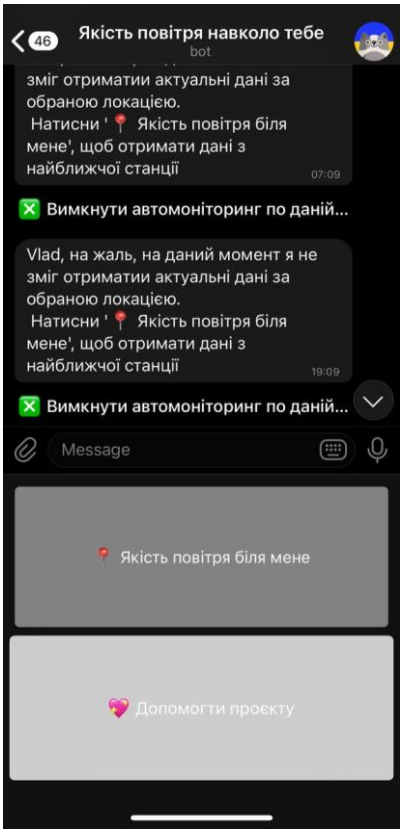


Рисунок 5.4 - Зовнішній вигляд кнопок боту.

На рисунку 5.5 зображено результат натискання кнопки якість повітря біля мене. Результатом цієї дії є отримання вимірів з найближчої станції моніторингу та відповідно короткий їх аналіз на предмет того чи можна проводити час на подвір’ї без шкоди здоров’ю.

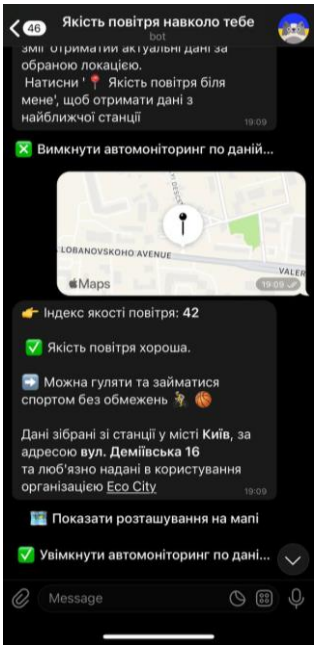


Рисунок 5.5 - результат отримання від станції моніторингу.

Якщо натиснути кнопку показати розташування на мапі, то як зображено на рисунку 5.6 прийде повідомлення із міткою на карті що і буде місцерозташуванням станції моніторингу. Натиснувши на мітку на карті можна буде обрати бажаний додаток для перегляду карт та відповідно ознайомитись із позицією мітки.

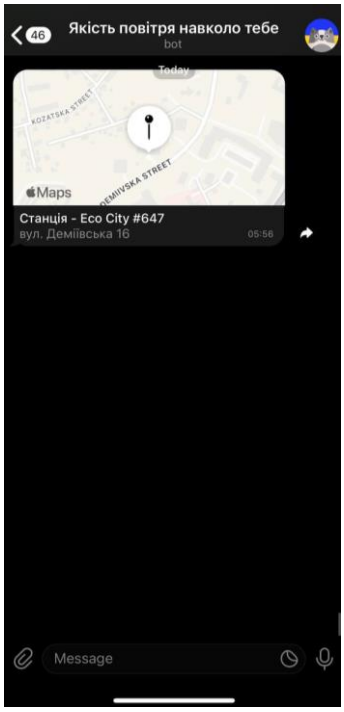


Рисунок 5.6 – Відповідь бота із міткою станції моніторингу.

Якщо на попередньому кроці натиснути кнопку увімкнути автомоніторинг по даній станції тоді результат виконання буде зображений на рисунку 5.7. Тепер щодня о 7 ранку та о 7 вечора бот надсилатиме інформацію щодо стану повітря на обраній станції.

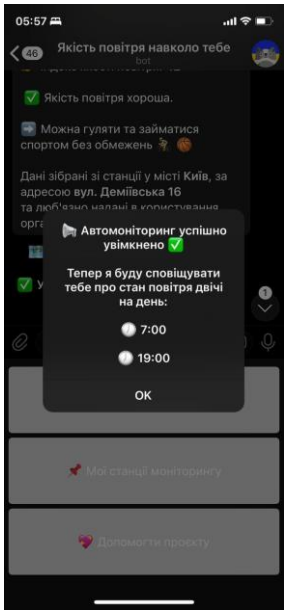


Рисунок 5.7 – увімкнення автомоніторингу за станцією.

По кнопці мої станції моніторингу відобразиться перелік станцій з можливістю вимкнення автомоніторингу як зображено на рисунку 5.8.

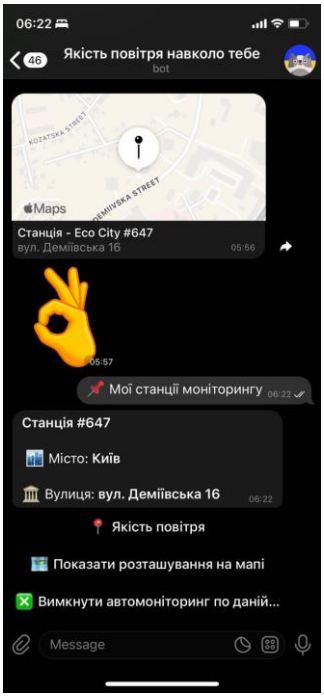


Рисунок 5.8 – Перелік станцій моніторингу з увімкненим автомоніторингом

ВИСНОВКИ

Актуальність проблеми забруднення повітря та необхідність моніторингу якості повітря підкреслює важливість розробки інформаційної системи моніторингу якості повітря. Аналіз існуючих систем на ринку показав, що різні системи пропонують різний функціонал та рівень доступу до інформації про якість повітря, але є потреба у більш інтегрованому та користувацьки-орієнтованому рішенні. Реалізована архітектура системи передбачає використання сучасних технологій: serverless технологій, хмарної інфраструктури, а також масштабування. Інформаційна система моніторингу якості повітря сприятиме підвищенню екологічної обізнаності населення, поліпшенню якості повітря та зменшенню негативного впливу забруднень на здоров'я людей. Під час виконання дипломної роботи було розроблено та втілено систему моніторингу якості повітря, яка надає RESTful API для доступу до даних про місцезнаходження та виміри станцій моніторингу. Це дозволяє різним зацікавленим сторонам інтегрувати дані про якість повітря в свої системи, а також створено телеграм-бот, який надає можливість отримувати виміри від найближчої до користувача станції моніторингу. З метою досягнення цієї цілі було проведено дослідження програм, платформ та систем, які публікують дані про якість повітря, а також проаналізовано наявні API. Під час написання системи були поглиблені та закріплені знання використаних технологій.

Проект охоплює широкий спектр технологій, методів та стратегій розробки програмного забезпечення, а ефективне використання цих інструментів допомогло створити гнучку та масштабовану систему, яка задовольняє всі поставлені вимоги.

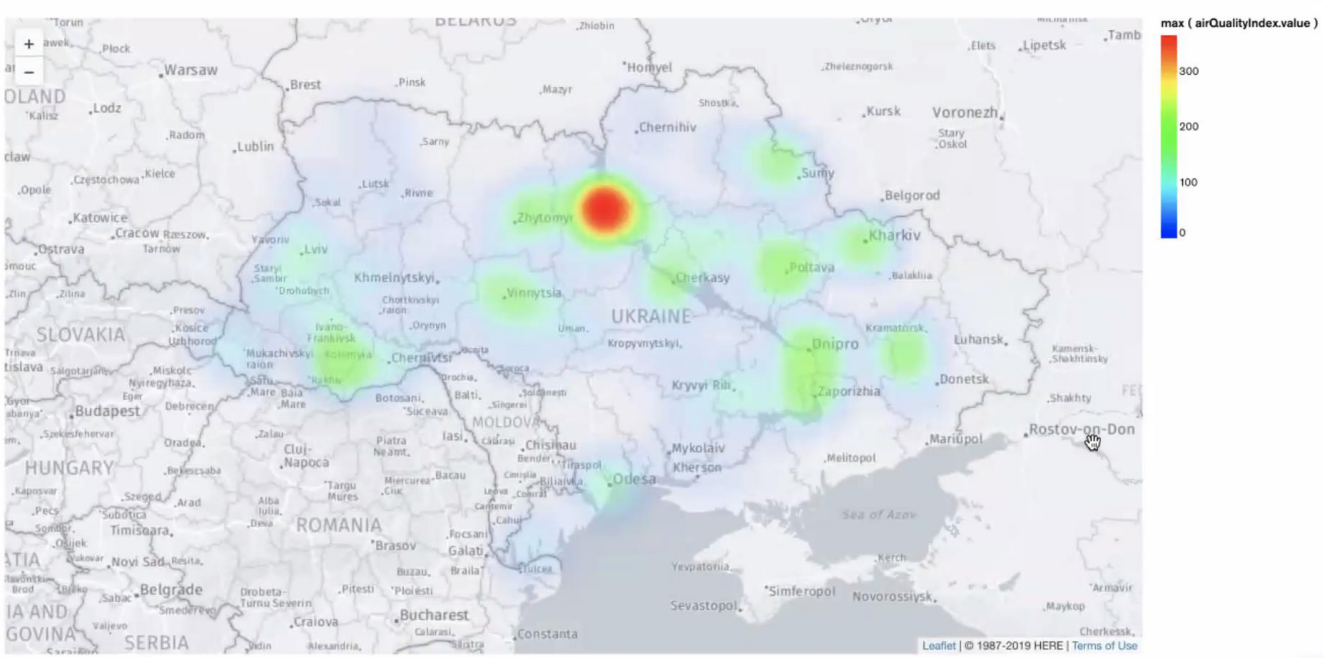
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

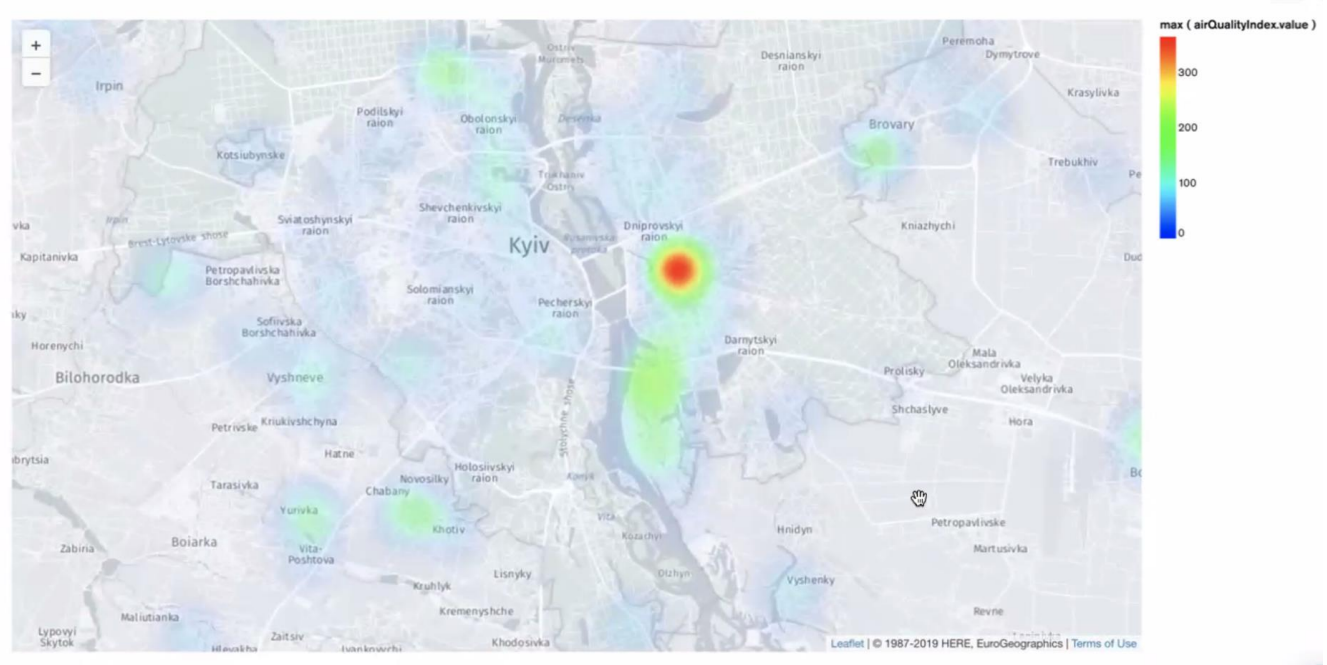
1. Статистичні данні про екологію України [Електронний ресурс] – Режим доступу до ресурсу: https://ecoaction.org.ua/diyalnist/povitria
2. Каплюк В. Інформаційна система моніторингу якості повітря: матеріали наук.-практ. конф. студентів і аспірантів «Теоретичні та прикладні аспекти розробки комп'ютерних систем 2023», м. Київ, 26-27 квітня 2023 року
3. MongoDB [Електронний ресурс] – Режим доступу до ресурсу: https://www.mongodb.com/docs/manual/replication/
4. M. Kleppmann Designing Data-Intensive Application / Kleppmann // O'Reilly. – 2017. – p.439-464.
5. MongoDB Compas [Електронний ресурс] – Режим доступу до ресурсу: https://www.mongodb.com/products/compass
6. .Net Core [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/dotnet/core/introduction
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://git-scm.com/
9. AWS EventBridge [Електронний ресурс] – Режим доступу до ресурсу: https://aws.amazon.com/eventbridge/
10. AWS SQS [Електронний ресурс] – Режим доступу до ресурсу: https://aws.amazon.com/sqs/
11. AWS Lambda [Електронний ресурс] – Режим доступу до ресурсу: https://aws.amazon.com/lambda/
12. AWS CloudWatch [Електронний ресурс] – Режим доступу до ресурсу: https://aws.amazon.com/cloudwatch/

13. TPL DataFlow [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/dotnet/standard/parallel-programming/dataflow-task-parallel-library
14. Onion architecture [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/@shivendraodean/software-architecture-the-onion-architecture-1b235bec1dec
15. API Key [Електронний ресурс] – Режим доступу до ресурсу: https://cloud.google.com/endpoints/docs/openapi/when-why-api-key
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. Рисунки Heatmap забруднення повітря





Додаток В. Покриття станціями території України реалізованими провайдерами даних