КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра інтелектуальних прикладних систем

Кваліфікаційна робота на здобуття ступеня бакалавра

за спеціальністю 121 Інженерія програмного забезпечення
на тему:

ІНФОРМАЦІЙНА СИСТЕМА МОНІТОРИНГУ ЯКОСТІ ПОВІТРЯ

Виконав студент 4-го курсу
Микола МЕДИНСЬКИЙ

(Підпис)

Науковий керівник:
асистент, кандидат фіз.-мат. наук
Костянтин ЖЕРЕБ

(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 математичних основ комп'ютерних наук

«25» травня 2022 р.,

протокол № 10
Завідувач кафедри
О. І. Провотар

(Підпис)

Київ-2022

РЕФЕРАТ

API, ASP.NET, C#, HTTP, WEBSOCKET, WI-FI МОДУЛЬ, ДАТЧИК, ІНФОРМАЦІЙНА СИСТЕМА, МОДЕЛЬ АКТОРІВ, МОНІТОРИНГ, СЕРВЕР, ФРОНТЕНД, ЯКІСТЬ ПОВІТРЯ.

Об’єктом розробки є пристрій для вимірювання якості повітря та система для висвітлення усіх даних отриманих з пристрою.

Мета роботи полягає у розробці інформаційно-технічної системи, яка надасть інформацію про стан якості повітря.

Методи та інструменти розробки. Для розробки було використано бібліотеки для програмування плат Arduino, які були використанні при програмування Wi-Fi модуля який передає дані з датчика на сервер. Було реалізовано два веб-застосунки: у першому серверна частина була реалізована за допомогою мови програмування Scala, фреймворків Play Framework, Akka Classic Actors, в іншому веб-застосунку – C# та ASP.NET фреймворку. А для розробки фронтенду було використано API Mapbox та Plotly.

Новизна роботи полягає у розробці інформаційної системи, яка надає інформацію про стан якості повітря в місті Рівне.

В результаті було спроектовано та розроблено пристрій для визначення концентрації CO в повітрі, створено два веб-застосунки, а також було реалізовано передавання даних з пристрою до веб-застосунків.

Практичне значення одержаних результатів. Результати досліджень можуть бути використані для вдосконалення системи моніторингу якості повітря у різних населених пунктах зокрема у місті Рівне, прогнозування рівня забруднення повітря та запобігання ситуаціям, що призводять до погіршення екологічної обстановки у населених пунктах.

Першочерговою задачею у майбутньому буде удосконалення веб-застосунку проекту та пошуку оптимальних шляхів висвітлення даних. Також в перспективі розробка Телеграм бота для того, щоб можна було отримувати свіжі дані про стан повітря не виходячи з месенджера.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

MVC – архітектурний шаблон Model View Controller;

PPM – parts per million, одиниця вимірювання концентрації;

HTML – Hypertext Markup Language, мова розмітки гіпертексту;

HTTP – HyperText Transfer Protocol, протокол передачі гіпертекстових документів;

URL – Uniform Resource Located, єдиний вказівник на ресурс;

API – Application Programming Interface, прикладний програмний інтерфейс.

ВСТУП

Здоровий спосіб життя – це невід’ємна складова життєдіяльності сучасної людини. Ми відвідуємо тренажерні зали, щоб зберігати наше тіло в тонусі, обираємо лише натуральні та якісні продукти харчування, щоб забезпечити свій організм енергією. Попри це, ми не завжди знаємо про якість повітря, яким ми дихаємо, незважаючи на те, що воно має велике значення у нашому здоров'ї. Ми не можемо бути впевнені, що фабрика, за високим парканом поблизу нашої оселі, не забруднює повітря шкідливими речовинами, які отруюють наш організм.

Забруднене повітря також негативно впливає на клімат і погоду. Сьогодні забруднення повітряного середовища в нашій країні різними газоподібними і пиловидними речовинами сягає вкрай небезпечних розмірів. Щодня зменшується питома вага кисню в повітрі через його безконтрольне спалювання, внаслідок чого можуть бути серйозні зміни у навколишньому середовищі.

До забруднюючих речовин, що переважно викидаються в атмосферне повітря, належать озон (O_3), монооксид вуглецю (CO), діоксид сірки (SO_2), оксиди азоту (NO_X).

Відбувається збільшення кількості випадків перевищення встановлених нормативів гранично допустимих викидів забруднюючих речовин стаціонарними джерелами та внаслідок бойових дій. Основними причинами в умовах мирного часу, що зумовлюють незадовільний стан якості атмосферного повітря в населених пунктах, є недотримання підприємствами режиму експлуатації пилогазоочисного обладнання, нездійснення заходів із зниження обсягу викидів забруднюючих речовин до встановлених нормативів, низькі темпи впровадження новітніх технологій та значне збільшення кількості транспортних засобів, зокрема тих, що вичерпали строк придатності.

В інтернеті можна знайти сервіси (наприклад World's Air Pollution: Real-time Air Quality Index [1]) які висвітлюють дані щодо забруднення повітря в тому

чи іншому місті, але вони не покривають велику частину території населених пунктів.

Актуальність роботи зумовлена низькою кількістю джерел інформації щодо вмісту шкідливих речовин у повітрі; розробки інформаційно технічної системи, яка буде надавати дані щодо стану повітря.

Мета роботи полягає у розробці інформаційно-технічної системи, яка надасть інформацію про стан якості повітря.

Завдання:

1) проектування пристрою для аналізу повітря;

2) розробка пристрою для дослідження стану повітря;

3) розробка веб-додатку для перегляду даних щодо стану повітря в межах м. Рівного;

4) порівняння реалізацій виконаних на мові програмування Scala та C#;

5) дослідження повітря в межах міста Рівне.

Об’єктом розробки є пристрій для вимірювання якості повітря та система для висвітлення усіх даних отриманих з пристрою.

Предметом дослідження є пристрій для вимірювання якості повітря та система яка висвітлювала б усі дані отримані з пристрою.

Новизна роботи полягає у розробці інформаційної системи, яка надає інформацію про стан якості повітря в місті Рівне.

Практичне значення одержаних результатів. Результати виконаних досліджень можуть бути використані для вдосконалення та оптимізації системи моніторингу атмосферного повітря у м. Рівного, прогнозування рівня забруднення атмосферного повітря шкідливими домішками та запобігання ситуаціям, що призводять до погіршення екологічної обстановки у місті.

РОЗДІЛ 1. ОПИС ВИКОРИСТАНИХ ТЕХНОЛОГІЙ СИСТЕМИ

1.1 Play Framework

Для створення початкового проекту я використав фреймворк Play Framework, який був розроблений компанією Lightbend спільно з open-source ком'юніті. Play – це високопродуктивний фреймворк для розробки веб-додатків на Java та Scala, який об'єднує в собі компоненти та API які допомагають у розробці. Play був розроблений веб-розробниками для розробки веб-додатків. Play включає всі компоненти, необхідні для створення веб-додатків та REST сервісів, таких як інтегрований сервер HTTP, обробка форм, потужний механізм маршрутизації, підтримка I18n тощо. Ось чому саме цей фреймворк було взято для розробки сайту [2].

1.2 MVC

Одним з найпоширеніших архітектурних шаблонів для розробки веб-додатків є MVC (Model View Controller). MVC є архітектурним патерном програмного забезпечення для реалізації користувальницьких інтерфейсів. Він ділить програму на три з’єднані між собою частини, а саме: на модель, вид і контролер, для того щоб відокремити внутрішню обробку інформації від інформації, що представляється для користувача.

Складові MVC:

- Контролер (controller) становить клас (рис. 1.1.), що забезпечує зв’язок між користувачем і системою, представленням і сховищем даних. Він отримує дані введені користувачем і обробляє їх. І в залежності від результатів обробки відправляє користувачеві певну інформацію, наприклад, у вигляді якоїсь веб-сторінки.
- Вид (view) – це власне візуальна частина або призначений для користувача інтерфейс програми. Як правило, html-сторінка, яку користувач бачить, зайшовши на сайт.

- Модель (model) представляє клас, що описує структуру використовуваних даних.

Шаблон проектування MVC розділяє ці основні компоненти, що дозволяють ефективно повторно використовувати код і сприяє його паралельному розвитку. Традиційно він використовується для розробки десктопних додатків [3]. Також ця архітектура є популярною для розробки веб-додатків та навіть мобільних застосувань. Ось чому вона підходить для цього проекту.

```
class SensorApiController @Inject()(cc: ControllerComponents)(implicit val ec:
DatabaseExecutionContext) extends AbstractController(cc) {
  implicit val dataWrites: Format[Sensor.Sensor] =
    ( (JsPath \ "id").formatNullable[Int] and
      (JsPath \ "ownerId").format[Int] and
      (JsPath \ "name").format[String] and
      (JsPath \ "longitude").format[String] and
      (JsPath \ "latitude").format[String]) (Sensor.Sensor.apply,
unlift(Sensor.Sensor.unapply))

  def getAdminSensorsJson(status: String) = Action.async{
      val sensors = SensorService.getAdminSensors(status)
      sensors.map(sensor=> Ok(Json.toJson(sensor)))
  }

  def saveSensor = Action.async(parse.json) { request =>
    val sensorResult = request.body.validate[Sensor.Sensor]
    sensorResult.fold(
      errors => {
        Future {
          Ok(Json.obj("message" -> JsError.toJson(errors)))
        }
      },
      sensor => {
        SensorService.addSensor(sensor).map(x =>  Ok(Json.obj("message" -> (x)))
       )
      }
    )
  }
}
```

Рис. 1.1. Фрагмент коду контроллера

1.3 Scala

Для реалізації бекенд частини початкового веб-застосунку я вирішив обрати мову програмування Scala, так як вона відкриває дуже багато можливостей у програмуванні. Вона є мультипарадигмальною і поєднує в собі об’єктно-орієнтоване та функціональне програмування [4]. Scala дуже добре себе зарекомендувала у сфері Big Data та розподілених обчислень і так як в даній роботі для підвищення продуктивності початкової системи багато дій виконувались паралельно (уся логіка пов’язана з Акторами) тому було обрано

саме цю мову програмування. Також якщо буде потрібно якось аналізувати дані у майбутньому, то в будь-який час можна буде використати бібліотеки Python так як є бібліотека ScalaPy яка дозволяє використовувати будь-які Python функції у самій Scala [5]. Тому Scala є непоганим вибором для цього проекту так як в будь-який момент можна по-різному буде розширювати проект.

1.4 Akka Actors

Akka – це бібліотека, що дає певні можливості для розробки високонавантажених систем. Робить вона це, використовуючи модель акторів [6]. Модель акторів – це концептуальна модель для роботи з одночасними обчисленнями. Вона визначає деякі загальні правила щодо того, як компоненти системи повинні поводитися та взаємодіяти між собою.

Основна ідея моделі акторів – все є Актор. А Актор – це сутність, яка приймає повідомлення, на які може відреагувати наступними діями:

- Надіслати N повідомлень іншим акторам (крім себе)
- Породити M нових акторів
- Змінити внутрішній стан (що позначиться на наступні реакції на вхідні повідомлення)

1.5 C# та ASP.NET MVC

Для створення іншого веб-застосунку було використано фреймворк ASP.NET та мову програмування C# (рис. 1.2). ASP.NET спирається на багатомовні можливості .NET, що дозволяє писати код сторінок на C#, VB, C/C++ та ін.

Для створення моделей та контролера була використана мова програмування C# (табл. 1.2), за допомогою якої створюють веб-додатки, і вона використовується в ASP.NET.

```
namespace Sensors.Controllers
{
    public class HomeController: Controller
    {
        public ActionResult Index()
        {
            return View();
        }
[Authorize(Roles = "admin")]
        public ActionResult Admin()
        {
            ViewBag.Message = "Sensor contact page.";
            return View();
        }
     }
}
```

Рис. 1.2. Фрагмент коду контроллера на C#

1.6 Entity Framework

Для роботи з даними в ASP.NET MVC було використано фреймворк Entity Framework. Великою перевагою цієї платформи є те, що вона дозволяє абстрагуватись від структури конкретної бази даних і взаємодіяти з даними через моделі. Дуже спростило роботу підхід Code First. Його суть полягає в тому, що спочатку пишеться код моделі C#, а потім за ним генерується база даних [7].

1.7 WebSocket протокол

У реалізації веб-застосунку на мові програмування Scala для преедачі даних з пристрою на сервер було використано протокол WebSocket. WebSocket є двонаправленим, повнодуплексним протоколом, який використовується в тому ж сценарії клієнт-серверної комунікації (рис. 1.3). Це протокол із збереженням стану, що означає, що з'єднання між клієнтом і сервером буде існувати, поки його не розірве будь-яка сторона (клієнт або сервер). Після закриття з’єднання клієнтом або сервером з’єднання припиняється з обох сторін [8].

Зазвичай WebSocket використовується у розробці:

- Веб-додатка який працює в режимі реального часу;
- Ігрового додатка;
- Чатів.

Рис. 1.3. WebSocket з’єднання [8]

1.8 HTTP протокол

Для передачі даних з пристрою до веб-застосунку написаного на мові програмування C# було обрано протокол HTTP. HTTP – це протокол передачі даних, який використовується для отримання таких ресурсів, як документи HTML. Він також використовується для комунікації між клієнтом і сервером.

HTTP – це протокол без зберігання стану, який працює поверх TCP, який є протоколом, орієнтованим на з'єднання, він гарантує доставку пакетів даних за допомогою методів тристороннього рукостискання та повторно передає втрачені пакети [8]. HTTP загалом працює наступним чином: клієнт надсилає запит на сервер, а сервер – відповідь на запит, і тоді з’єднання обривається (рис. 1.4).

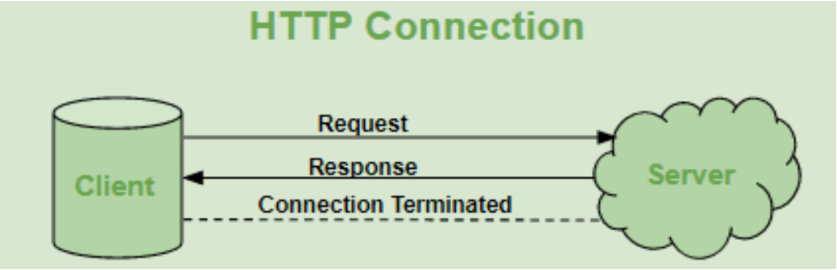

Рис. 1.4. HTTP з’єднання [8]

РОЗДІЛ 2. ПРОЕКТУВАННЯ ТА РОЗРОБКА СИСТЕМИ МОНІТОРИНГУ ПОВІТРЯ

У цьому розділі описується проектування апаратної частини проекту, визначення значення концентрації газу CO у повітрі, корегування даних отриманих з датчика, принцип роботи системи, реалізації веб-застосунків.

2.1 Вибір датчика

Напівпровідникові датчики газу серії MQ – це невеликі за розміром пристрої виміру вмісту газів у повітрі, в основі побудови яких лежить нагрівальний елемент з електро-хімічним сенсором.

MQ-7 – це напівпровідниковий датчик (рис. 2.1) для виявлення моноксиду вуглецю, метану і бутану [9].

Рис. 2.1. Датчик MQ-7

MQ-9 – напівпровідниковий датчик (рис. 2.2) для виявлення горючих газів та моноксиду вуглецю. MQ-9 чутливий до CO (чадний газ), CH_4 (метан), C_3H_8 (пропан), C_4H_{10} (бутан) [10].

Рис. 2.2. Датчик MQ-9

MQ-135 – це напівпровідниковий датчик (рис. 2.3), який реагує на такі речовини, як: аміак (NH3), оксид азоту (NOx), бензен (C_6H_6) та вуглекислий газ (CO_2), пари спирту, чадний газ (CO) [11].

Рис. 2.3. Датчик MQ-135

MQ-138 – це напівпровідниковий датчик (рис. 2.4) для виявлення в повітрі забруднюючих речовин, а саме: толуолу, випарів спирту, ацетону, формальдегідів [12].

Рис. 2.4. Датчик MQ-138

Після детального вивчення та аналізу джерел було вирішено, що краще всього обрати датчик MQ-7 (рис. 2.1), так як він найбільший чутливий до CO і тому можна буде отримати найточніші данні про CO в повітрі. Для калібрування датчика, його потрібно лишити увімкнутим протягом 24 годин при температурі 20 °C і відносній вологості 65%. Згідно з документацією похибка вимірювань становить ±5% [9].

2.2 Wi-Fi модуль ESP8266

Так як система розроблюється для того щоб в майбутньому бачити дані про стан повітря у будь-якій точці певного населеного пункту, нам потрібно щоб пристрої передавали данні на сервер бездротового підключення до самого сервера. У такому випадку ми можемо використати підключення до бездротової мережі Wi-Fi. Таким чином будемо мати вихід у мережу Інтернет і зможемо передавати дані. Для передачі даних з датчика на сервер було обрано Wi-Fi модуль ESP8266 (рис. 2.5) [13]**.**

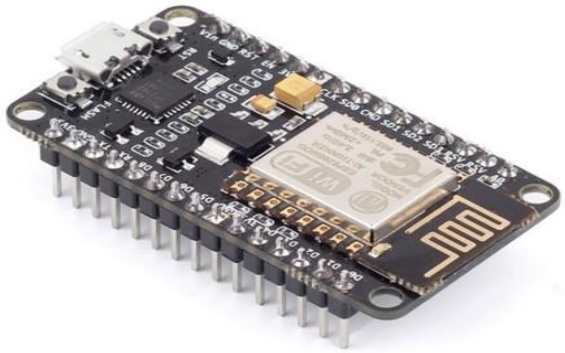

Рис. 2.5. Wi-Fi модуль ESP8266

2.3 Апаратна архітектура

Нижче зображено схему підключення Wi-Fi модуля з датчиком MQ-7 (рис. 2.6).

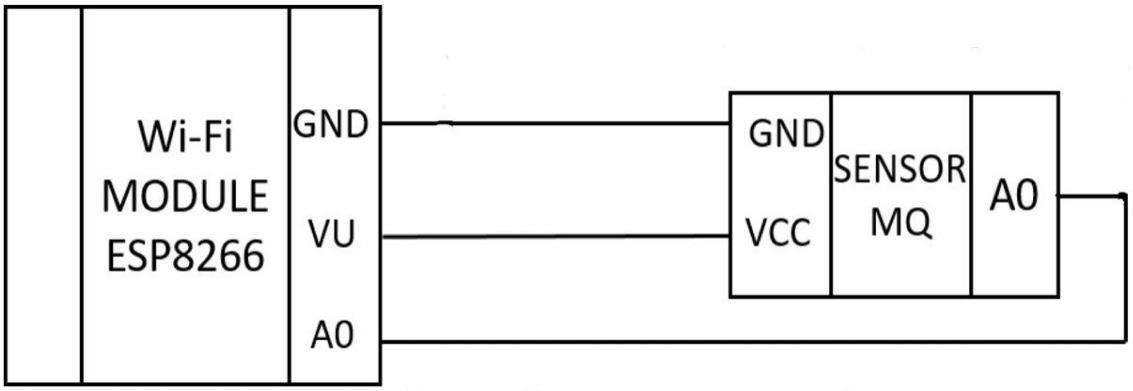

Рис. 2.6. Схема підключення пристроїв

Після проектування було сконструйовано прототип пристрою, який буде аналізувати концентрацію газу CO в повітрі (рис. 2.7).

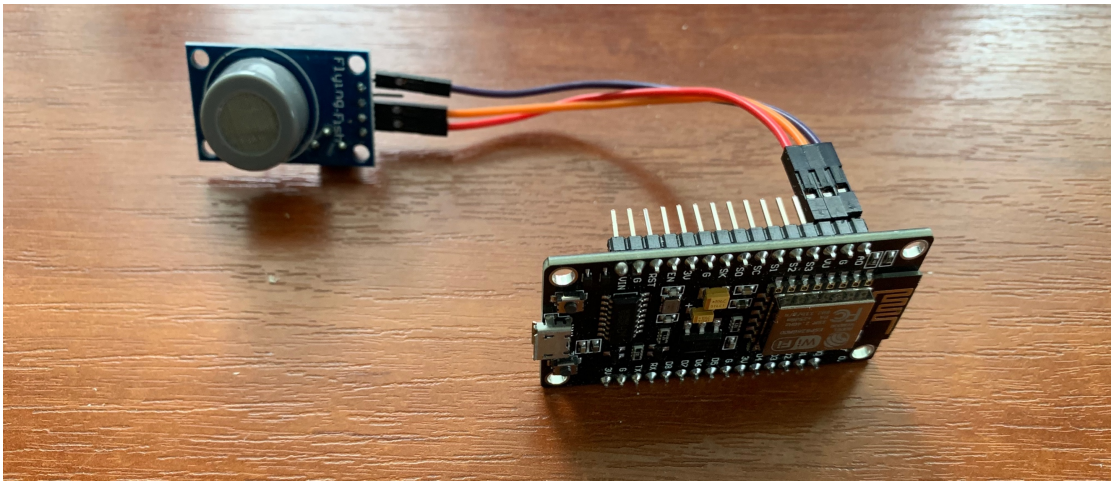

Рис. 2.7. Прототип пристрою

2.4 Програмування Wi-Fi модуля

Наступним важливим етапом у створенні геоінформаційної системи стало програмування Wi-Fi модуля. Для програмування Wi-Fi модуля було використано мову програмування C, так як зазвичай на цій мові програмуються продукти компанії Arduino. При написанні коду (рис. 2.8.), використано бібліотеку ESP8266WiFi, яка значною мірою полегшила створення зв'язку з Wi-Fi модулем, для передачі даних по WebSocket було використано методи бібліотеки ArduinoWebsockets.h, а по HTTP – ESP8266HTTPClient.h.

```
#include <ArduinoWebsockets.h>
#include <ArduinoJson.h>
#include <ESP8266WiFi.h>
const char* ssid = ""; //Enter SSID
const char* password = ""; //Enter Password
const char* websockets_server = "websocketServerUrl";
const char* websockets_server2 = " websocketServerUrl";//server adress and port
int analogValue = 0;
const int analogInPin = A0;
const int capacity = JSON_OBJECT_SIZE(2);
StaticJsonDocument<capacity> doc;
using namespace websockets;

WebsocketsClient client;
WebsocketsClient client2;
void setup() {
    Serial.begin(115200);
    // Connect to wifi
    WiFi.begin(ssid, password);
    // Wait some time to connect to wifi
    for(int i = 0; i < 10 && WiFi.status() != WL_CONNECTED; i++) {
        Serial.print(".");
        delay(1000);
    }
//    client.onMessage(onMessageCallback);
//    client.onEvent(onEventsCallback);
client.connect(websockets_server);
    client2.connect(websockets_server2);
}
```

Рис. 2.8 Приклад коду для програмування ESP8266

Wi-Fi модуль виступає у системі у вигляді передавача даних з датчика на сервер. Так як дані будуть надсилатись в режимі реального часу, для передачі даних у початковій реалізації на мові програмування Scala було обрано протокол передачі даних WebSocket. Тобто при запуску Wi-Fi модуль встановлює зв'язок з сервером по одній WebsSocket URL а потім вже у циклі, раз у 5 секунд, відбувається зчитування даних та їх надсилання до серверу за умови підключення до мережі Wi-Fi, доступу до Інтернету та підключення до серверу по WebSocket (рис. 2.9).

```
void loop() {
    client.poll();
    client2.poll();
    analogValue = analogRead(analogInPin);
    doc["id"] = 1;
    doc["value"] = analogValue;
    printf("Value: %d",analogValue);
    Serial.print(analogValue);
    client.send(doc.as<String>());
    analogValue = analogRead(analogInPin);
    doc["id"] = 2;
    doc["value"] = analogValue/2;
    client2.send(doc.as<String>());
    delay(5000);
}void loop() {
    client.poll();
    client2.poll();
    analogValue = analogRead(analogInPin);
    doc["id"] = 1;
    doc["value"] = analogValue;
    printf("Value: %d",analogValue);
    Serial.print(analogValue);
    client.send(doc.as<String>());
    analogValue = analogRead(analogInPin);
    doc["id"] = 2;
    doc["value"] = analogValue/2;
    client2.send(doc.as<String>());
    delay(5000);
}
```

Рис. 2.9. Приклад коду з передаванням даних по вебсокету

Але у реалізації альтернативної системи ми вирішили замінити протокол передачі даних з WebSocket на HTTP і передаємо дані за допомогою GET запитів.

2.5 Запуск пристрою та принцип роботи системи розробленої на мові програмування Scala

Спочатку підключаємо пристрій до джерела живлення – так запускається система (рис. 2.10). Після цього датчик починає аналізувати повітря та відповідно до його стану видає напругу в межах 0-5V і через аналогово-

цифровий перетворювач конвертує напругу в цифрове значення від 0 до 1023 та передає інформацію до Wi-Fi модуля. Wi-Fi модуль, в свою чергу при увімкненні, під’єднується до сервера та надсилає по вебсокету значення зчитане з датчика. На серверній частині значення спочатку потрапляє до контроллера який створює актора для подальшої обробки цього значення та актора який буде займатись передачею даних на фронтенд. Перший актор приймає значення з Wi-Fi модуля та зберігає дані в базу даних. Також цей перший актор надсилає дані іншому актору для того щоб той передав ці дані на фронтенд частину.

В свою чергу коли користувач заходить на веб-сайт, фронтенд частина під’єднується по вебсокету до сервера. При під’єднанні, з фронтенд частини надсилається тестове повідомлення і якщо воно було отримано сервером то значить зв’язок був успішно встановлений. Тут також при спробі під’єднатись по вебсокету створюється актор через який дані з сенсора будуть попадати на фронтенд частину.

На фронтенді показана карта з маркерами, які вказують де будуть знаходитись датчики які роблять заміри. Також на сторінці сайту показаний графік, який в режимі реального часу буде будуватись по отриманим даним.

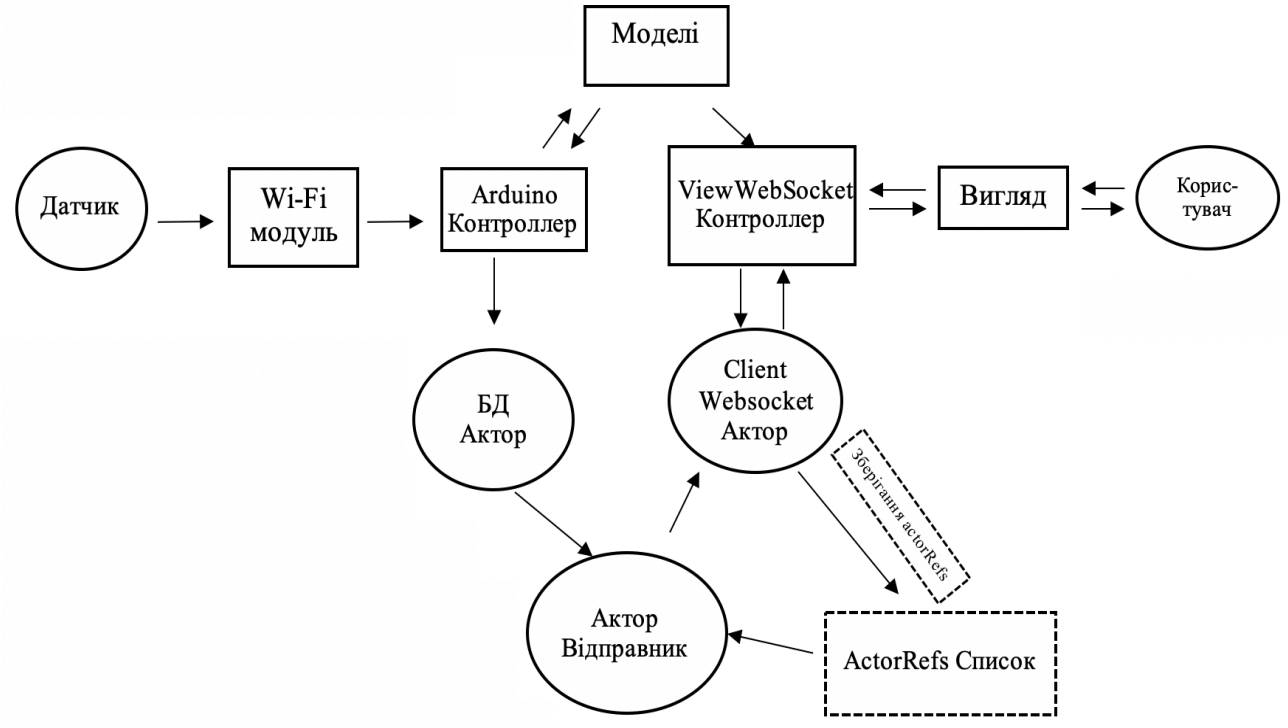

Рис. 2.10. Принцип роботи системи

2.6 Врахування параметрів навколишнього середовища

Згідно технічної документації (datasheet) до датчика MQ-7 бажано враховувати вологість та температуру повітря, що досліджується для досягнення більшої точності. Залежність від температури та вологості наведено на рисунку 2.11.

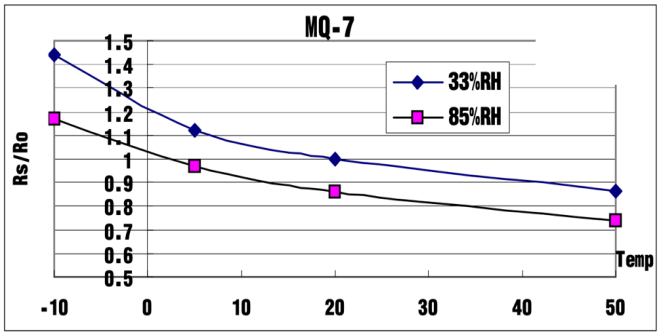

Рис. 2.11. Типова залежність датчика MQ-7 від вологості та температури [9]

Умовні позначення з графіку 2.11:

- R0 – опір датчика при 100ppm CO у повітрі при 33% вологості та 20 градусів.
- Rs – опір датчика при 100ppm CO при різних температурах і вологості.
- RH – відносна вологість

Алгоритм даної корекції вже успішно реалізований в бібліотеці Arduino library для MQ-135 [19]. В даній бібліотеці розробники вже описали необхідні формули для обчислення ppm значення з датчика MQ, тому залишилося ними лише скористатися у нашому рішенні. Наведемо основні кроки корекції показників датчика у відповідності до вологості та температури згідно з бібліотекою [19] на мові C++ та отримання значення ppm (parts per million – частин на мільйон):

1. Викликаємо метод getRzeroCO(), що на основі декількох констант дозволяє нам знайти R0, яке потім використовується як константа у знаходженні кількості чадного газу у повітрі.

```
float getRZeroCO() const {
  return getResistance()*pow((scaleFactorCO/ppm),(1/expomentCO))
}
//float scaleFactorCO = 98.54; float exponentCO = -1.521
```

2. Але якщо температура і вологість не дорівнюють 20 °C і 65% відповідно, то ми корегуємо дані за допомогою функції getCorrectionFactor(float t, float h).

```
float getCorrectionFactor(float t, float h) const { // t – температура, h –  вологість
  return CORA * t * t - CORB * t + CORC - (h-33.)*CORD;
}
//CORA = 0.00035; CORB = 0.02718; CORC = 1.39538; CORD = 0.0018
```

3. Після того ми отримуємо змінений опір за допомогою функції getCorrectedResistance(float t, float h)

```
float getCorrectedResistance(float t, float h) const {
  return getResistance()/getCorrectionFactor(t, h);
}
```

4. Заключний етап полягає в знаходженні значення CO в ppm і для цього використовуємо функцію getCO().

```
float getCO(float res) const { // res – це опір який ми отримуємо в методі
//getResistance() або в getCorrectedResistance(float t, float h)
  return scaleFactorCO * pow((res/r0CO), -exponentCO);
}
```

Однак ми не можемо безпосередньо використати даний алгоритм на стороні Wi-Fi пристрою, оскільки єдині дані, якими ми володіємо, це показники датчика MQ-7. Тому температуру та вологість будемо отримувати на серверній стороні із метеосайтів, оскільки знаємо місцерозташування кожного датчика.

Отже, на стороні сервера буде реалізовано спеціальний метод modifyAccordingToTempAndHumidity для описаного вище алгоритму, що підвищить точність результатів вимірювання.

2.7 Розробка та створення корпусу для пристрою

Для того щоб пристрій гарно виглядав та ним було зручно користуватись, потрібно було створити спеціальний корпус на 3d принтері. Але перед друком треба спроектувати 3d модель і для цього було використано програму AutoDesk Inventor (рис. 2.12). Ця програма має широкий функціонал для створення об’єктів різних величин (також використовується для створення моделей двигунів та запчастин до техніки) і є зручною у використанні. Проектування 3d моделі поділяється на 2 етапи:

1. Створення ескізу

2. Створення об’ємної моделі тіла

В ескізі створюємо 2d форму об’єкта (де вказуємо певні розміри), а вже в 3d моделі змінюємо під потрібний нам вигляд, об’єкт.

Після створення 3d моделі друкуємо на 3d принтері корпус.

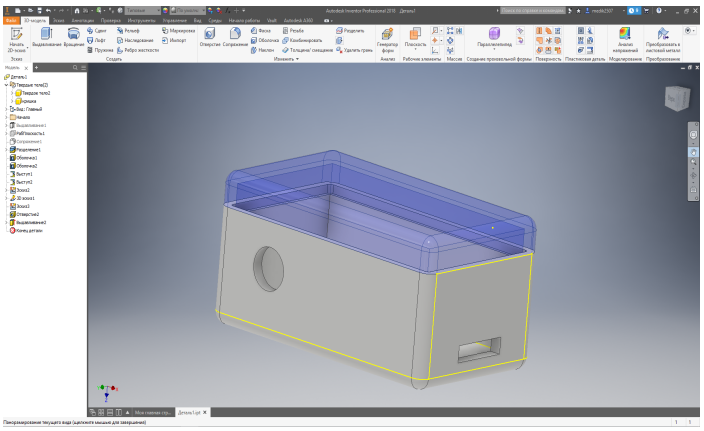

Рис. 2.12. Прототип корпусу розроблений у Autodesk Inventor

На рисунку 2.13 зображено наш пристрій у роздрукованому на 3D принтері корпусі:

Рис. 2.13. Пристрій у корпусі

2.8 Серверна частина веб-застосунку розробленого на Scala

Для реалізації серверної частини веб-додатку на мові програмування Scala було використано Play Framework і Akka Actors (Рис. 2.14).

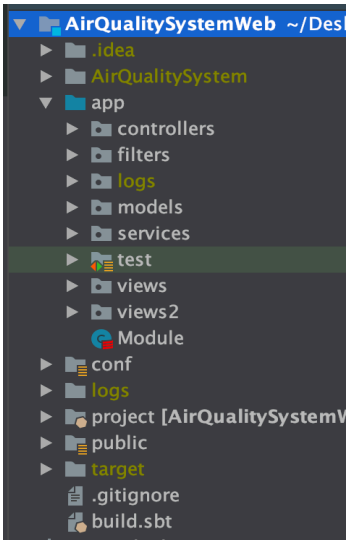

Рис. 2.14. Структура проекту

В папці models розташовані класи, що відповідають за структуру користувача, датчиків та даних з датчика.

У структурі користувача є наступні дані (рис. 2.15): id, email, пароль, статус (користувач може бути адміном або клієнтом), повне ім’я, юзернейм, websocket_url_in (вебсокет URL для надсилання даних на сервер з датчика; ця URL буде у відкритому доступі для користувача і він зможе використовувати її для своїх цілей), websockt_url_out (вебсокет URL яка використовується для виводу даних клієнта на його сторінці сайту).

```
Object User {
  case class User (id: Option[Int] = None, email: String, password: String, fullName:
String,
                   status: String, userName: String, websocketUrlOut: String,
websocketUrlIn: String)

   class UserTableModel(tag: Tag) extends Table[User](tag, "users") {


    def id = column[Int]("id", O.PrimaryKey, O.AutoInc)
    def email = column[String]("email")
    def password = column[String]("password")
    def status = column[String]("status")
    def fullName = column[String]("full_name")
    def userName = column[String]("user_name")
    def websocketUrlOut = column[String]("websocket_url_out")
    def websocketUrlIn = column[String]("websocket_url_in")
    override def * =
      (id.?, email, password, fullName, status,
        userName, websocketUrlOut, websocketUrlIn) <>(User.tupled, User.unapply)
  }
  val userTable = TableQuery[UserTableModel]
}
```

Рис. 2.15 Модель даних User

В свою чергу кожен сенсор (рис. 2.16) має дані про своє розташування, id та id власника. Моделі даних розроблялись з ідею подальшого розширення і в наступних версіях системи кожен користувач мав би змогу додавати свої датчики та моніторити стан повітря на територіях де вони ж встановили датчики на своїй сторінці сайту.

```
object Sensor {
  case class Sensor(id: Option[Int], ownerId: Int, name: String, longitude: String,
latitude: String)
  class SensorTableModel(tag: Tag) extends Table[Sensor](tag, "sensors") {
    def id = column[Int]("id", O.PrimaryKey,O.AutoInc)
    def ownerId = column[Int]("owner_id")
    def name = column[String]("name")
    def longitude = column[String]("longitude")
    def latitude = column[String]("latitude")
    override def * = (id.?, ownerId, name, longitude,latitude) <> (Sensor.tupled,
Sensor.unapply)
    def owner = foreignKey("fk_owner", ownerId, User.userTable)(_.id)
  }
  val sensorTable = TableQuery[SensorTableModel]
}
```

Рис. 2.16. Модель даних Sensor

Самі ж дані з сенсорів містять в собі id датчика, своє власне id, значення отримане з датчика та час коли значення було отримане на сервері (Рис. 2.17).

```
object SensorData {
  case class SensorData(id: Option[Int], sensorId: Int, value: Int, time:
String )

   class SensorDataTableModel(tag: Tag) extends Table[SensorData](tag,
"sensors_data") {

     def id = column[Int]("id", O.PrimaryKey, O.AutoInc)
     def sensorId = column[Int]("sensor_id")
     def data = column[Double]("data")
     def time = column[String]("time")
     def fkSensorId = foreignKey("fk_sensor_id", id, Sensor.sensorTable)(_.id)
     override def * =
      (id.?, sensorId, data, time) <>(SensorData.tupled, SensorData.unapply)
  }
  val SensorDataTable = TableQuery[SensorDataTableModel]
}
```

Рис. 2.17. Модель даних SensorData

Приклади застосування цих класів можна побачити у SensorApiController, UserApiController, ShowDataApiController, які приймають HTTP запити та видають певну інформацію. Наприклад:

- GET запит /getadminsensorjson – цей запит потрапляє в SensorApiController який задопомогою SensorSevice витягує усі датчики адмінів
- GET запит /getdata/:id – потрапляє в DataApiController який потім повертає дані з датчика.
- POST запит /savesensor – зберігає датчик в базу данних.

2.9 Фронтенд частина веб-застосунку розробленого на Scala

На фронтенд частині додатку, користувачу зображена карта на якій є маркери, які показують де знаходяться датчики та графік зміни якості повітря на тій чи іншій території (Рис. 2.18).

Для роботи з картою було використано API Mapbox [14] яка дозволяє користуватись функціями сервісу OpenStreetMap. За допомогою цієї API, я зміг додати карту на сайт та маркери на цій ж карті для висвітлення де будуть розташовані датчики CO. А для роботи з графіком було використано бібліотеку Plotly.

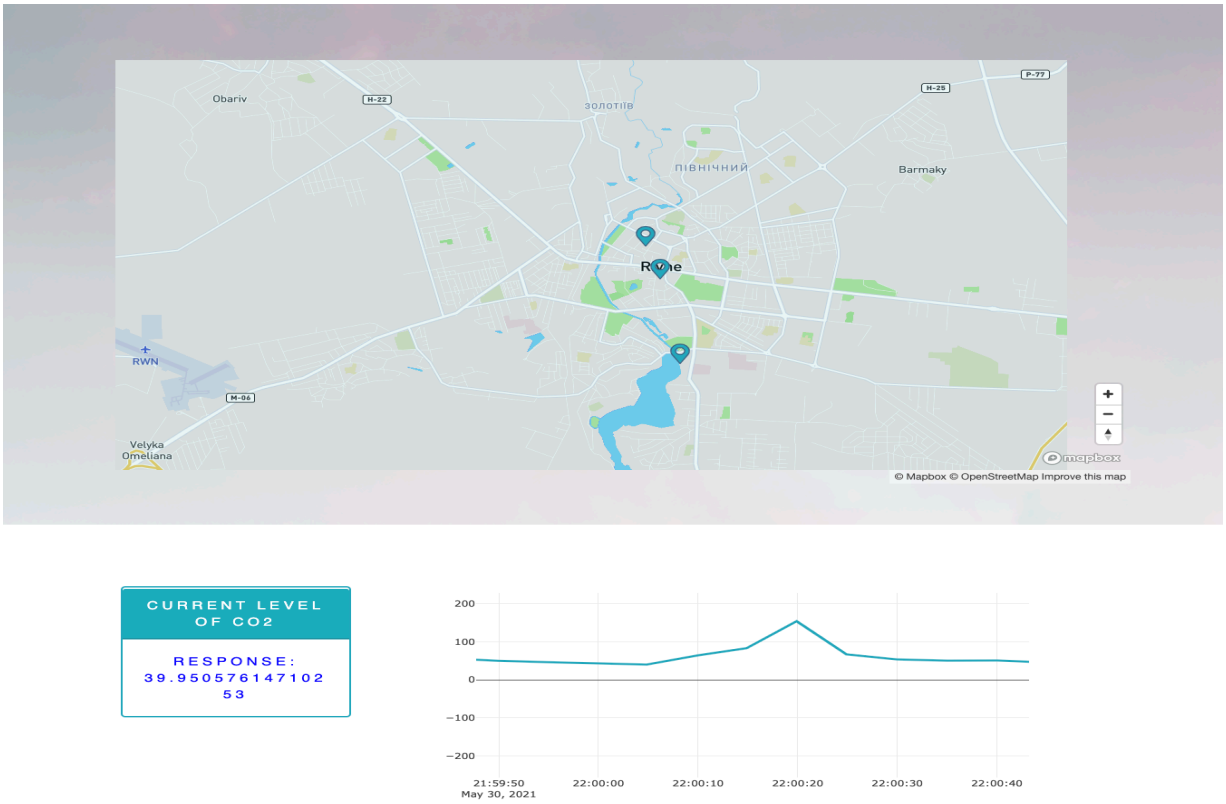

Рис. 2.18. Головна сторінка сайту

2.10 Реалізація веб-застосунку на мові програмування C#

Під час передачі даних по вебсокету дуже часто з’єднання обривалось та програмно на рівні Акторів було складно реалізувати коректне перепід'єднання і тому система була не надійною. Тому у даній дипломній роботі було вирішено також реалізувати інший веб-застосунок, але вже використовуючи протокол HTTP для передачі даних, мову програмування C# та ASP.NET фреймворк.

Було вирішено також трохи спростити веб-застосунок та переробити архітектуру. Як зазначалось вище, альтернативою для передачі даних з датчика на сервер було обрано HTTP-запити. Зазвичай для передачі даних відбувається за допомогою POST запитів, але в даній реалізації було вирішено надсилати дані за допомогою GET запитів у вигляді параметрів.

Нижче наведено приклад деяких запитів у новій реалізації:

- GET запит /data/set/1?level1= – цей запит використовується для передачі даних з датчика на сервер. Тут «1» виступає як айді датчика з якого передаються дані а level1 – це параметр під яким передається значення з датчика
- GET запит data/edit/id – цей запит використовується для зміни даних. В даному випадку id – це id запису даних.
- GET запит api/GetSensors – отримуємо список усіх датчиків

В основному, шляхи запитів у ASP.NET веб-застосукнку можна сформувати автоматично через конфігурації у класі RouteConfig, тому загальна форма шляху URL запитів виглядає так:

```
{controller}/{action}/{id}
```

Тут {controller} відповідає за певний контроллер, {action} за певну функцію з контроллеру, а {id} – параметри.

2.11 Серверна частина веб-застосунку розробленого на C#

Веб-застосунок розроблений на мові програмування C# також побудований за допомогою архітектурного шаблону MVC. Як згадувалось

раніше, згідно патерну MVC у нас є 3 основні частини в проекті – Models, Views та Controllers (рис 2.19).

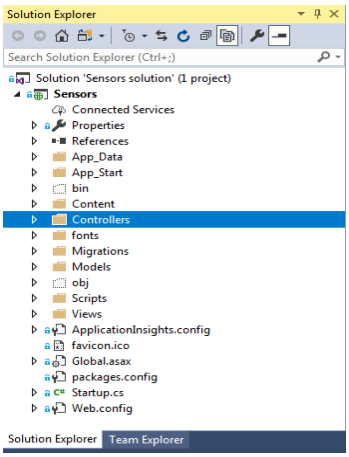

Рис. 2.19. Структура ASP.NET MVC проекту

В частині Models розташовані класи, що відповідають за датчики (рис. 2.20) та їхні дані (рис. 2.21).

```
public class Sensor
    {
        [DisplayName("ID")]
        public int Id { set; get; }
        [DisplayName("PhoneNumberID")]
        public string phoneNumberID { get; set; }
        [DisplayName("Location")]
        public string location { get; set; }
        [DisplayName("Description")]
        public string description { get; set; }
        public ICollection<Data> datas { get; set; }

        public Sensor()
        {
            datas = new List<Data>();
        }
    }
```

Рис. 2.20. Модель даних Sensor

На початкових етапах реалізації веб-застосунку на мові програмування C#, передача даних планувалася за допомогою GSM мережі. Тому в якості ідентифікатора датчика виступає телефонний номер. Однак в подальшому ми відмовилися від цієї ідеї та вирішили передавати дані за допомогою технології Wi-Fi використовуючи HTTP запити, але поле phoneNumberID і надалі використовується як ідентифікатор.

```
public class Data
    {
        public int Id { get; set; }
        public int SensorId { get; set; }
        [DisplayName("Data")]
        public string data { set; get; }
        [DisplayName("Date")]
        public DateTime? dateTime { set; get; }
        [DisplayName("Batery")]
        public int batery { get; set; }
        [DisplayName("Level")]
        public int level1 { get; set; }
    }
```

Рис. 2.21. Модель даних Data

Приклад застосування цих класів можна продемонструвати на методі Set у контролері DataController, що відповідає за отримання даних від WiFi модуля (рис. 2.22). Отримане значення якості повітря має модифікуватись згідно значень температури та відносної вологості. Створюється екземпляр класу Sensor на основі пошуку в базі даних датчика із певним ідентифікатором. Також створюється пустий екземпляр класу Data, поля якого поступово наповнюються потрібними даними. Насамкінець об’єкт data додається до контексту даних та зберігається в базі даних.

```
public ActionResult Set(int ? id, int ?level1, int? level2)
        {
            int value = level1 ?? 0;
            ModifyAccordingToTempHumidity(ref value);
            string phoneNumber = (id ?? 2).ToString();
            Sensor sensors = db.Sensors.Where(p => p.phoneNumber ==
phoneNumber).FirstOrDefault();
            Data data = new Data();
            data.Id = 20;
            data.SensorId = sensors.Id;
            data.dateTime = DateTime.Now;
            data.level1 = value;
            data.batery = level2 ?? 0;
            db.Data.Add(data);
            db.SaveChanges();
            return RedirectToAction("Index");
        }
```

Рис. 2.22. Обробка вхідних даних

2.12 Фронтенд частина веб-застосунку розробленого на C#

На стороні фронтенду за створення карти місцевості відповідають функції з main.js файлу. В ньому проводиться потрібна ініціалізація MapBox карти.

Загалом усі найцікавіші частини фронтенду висвітлені у Main.cshtml файлі. У цьому файлі скомбіновано висвітлення графіку зміни якості повітря, карту з маркерами які вказують на розташування датчиків, список датчиків та список записів отриманих даних з датчиків.

У main.js також реалізовано функцію showSensors, яка завантажує показники датчиків та виводить маркери різного кольору в залежності від показника забруднення (функція getSensorIconByLevel); showPlot малює графік даних протягом певного часу за допомогою бібліотеки plotly. Також, користувач може вибрати останні дані по певному датчику за певний період (останній день, 3 дні, тиждень та місяць) або взагалі можна спостерігати за надходженням даних від датчиків в режимі реального часу. Для швидкого пошуку датчика за координатами, місцерозташуванням або ідентифікатором, а також для сортування колонок таблиць датчиків та даних за зростанням чи спаданням, було використано плагін до JQuery dataTables.

2.13 Порівняння реалізацій

У кожній з цих реалізацій є свої переваги та недоліки. Головним недоліком у реалізації веб-застосунку на мові програмування Scala було те що передача даних по WebSocket була нестабільною і часто з’єднання саме по-собі обривалось і програмно було складно реалізувати перепід’єднання. Також через багато неправильних рішень на етапі проектування системи було дуже складно розширити функціонал використовуючи Play Framework. Тому було вирішено не переписувати уже існуючий додаток, а спроектувати та створити новий але уже за допомогою ASP.NET фреймворку, мови програмування C# та передачею даних за допомогою HTTP запитів. В результаті передача даних за допомогою HTTP запитів виявилась стабільнішою, хоча за попередніми спостереженнями передача даних по WebSocket була швидшою. ASP.NET фреймворк значно простіший у використанні ніж Play фреймворк тому було значно легше реалізовувати новий веб-застосунок та додавати новий функціонал. У реалізації

на C# було додано можливості користувачу з переглядом даних, редагуванням їх, та загалом був покращений вигляд користувацького інтерфейсу.

2.14 Опис аналогічних продуктів

Було проведено пошук аналогічних рішень, зокрема було знайдено інформаційні системи подібні до нашої. Одна з них – це система EcoCitizens (рис. 2.23). EcoCitizens – пристрій, що складається з плати з датчиками, плати обробки даних, батареї та екрану [15]. EcoCitizens вимірює склад повітря, а саме вміст CO_2 та NO_2, температуру, вологість, інтенсивність світла і рівень шуму. Далі пристрій передає весь потік даних за допомогою датчиків через Wi-Fi.

Рис. 2.23. Система EcoCitizens

Система розроблена на базі платформ Arduino або Raspberry Pi. Також передбачається структурування та обробка вже існуючих масивів даних про стан навколишнього середовища.

За кордоном має популярність система AirCasting (рис. 2.24), що дозволяє вимірювати не тільки CO_2, NO_2, температуру, вологість, інтенсивність світла і рівень шуму [16].

Рис. 2.24. Система AirCasting

Конкурентом можна вважати й систему smartcitizens (рис. 2.25), яка вимірює все те, що вимірює Air Casting. Але їхні датчики дорогі, система не працює в Україні, та вони мають малу кількість показників на своїй карті [17].

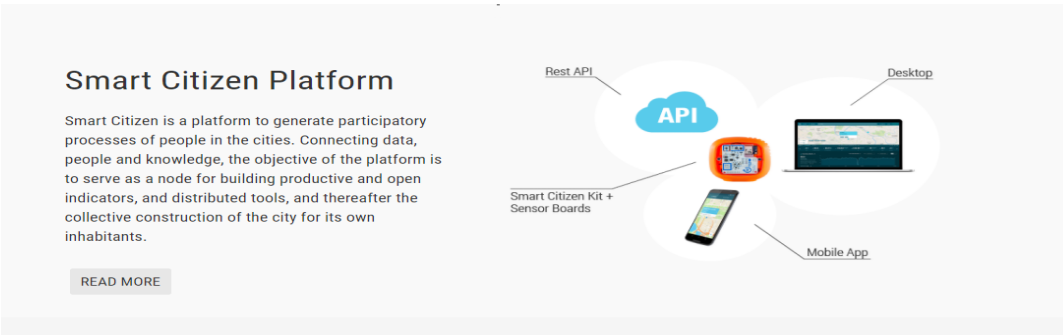

Рис. 2.25. Smart Citizens

Зовсім недавно з'явився на міжнародному рівні проект Breezometer (рис. 2.26), який дозволяє переглянути дані про стан повітря у будь-якій точці світу. Ця система бере інформацію про стан повітря за допомогою державних метеостанцій та відкореговує її відносно погоди та напряму вітру тієї чи тієї території. Після цього дані висвітлюються на сайті breezometer.com [18]. Але так як breezometer тримає свої алгоритми в таємниці, ми не можемо впевнитись у надійності тих даних, які висвітлені на їхньому сайті.

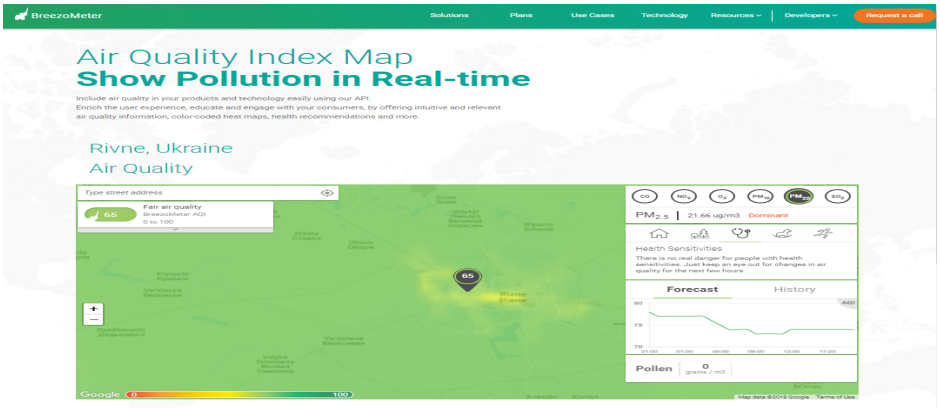

Рис. 2.26. Breezometer

Так як ecocitizen.online вже не доступний, air casting та smartcitizens не охоплюють територію України, а breezometer використовує дані державних установ й із незрозумілим та закритим алгоритмом, в нас є всі шанси успішно реалізувати наші дослідження і систему в межах України.

РОЗДІЛ 3. ВИКОРИСТАННЯ СИСТЕМИ ТА ЇЇ ПОДАЛЬШЕ УДОСКОНАЛЕННЯ

3.1 Визначення стану повітря

Так як чадний газ (CO) є одним з найбільш розповсюджених забруднювальних речовин у повітрі, було вирішено, що краще всього калібрувати пристрій під цей газ. Концентрація газу в повітрі вимірюється в ppm. У таблиці 3.1 висвітлені концентрації чадного газу в повітрі та їхні приблизні значення.

Табл. 3.1. Концентрації чадного газу та їхні приблизні значення

0 – 3 ppm	Чисте повітря
3 – 9 ppm	Нормальне повітря
9 – 35 ppm	Мало забруднене
>35	Забруднене

При обробці даних дуже важливо враховувати додаткові умови, такі, як вологість і температура. У разі необхідності можна провести калібрування. Для цього використовується інформація з документації [9], щоб правильно налагодити пристрій (рис. 3.1) та задати початковий опір (R0).

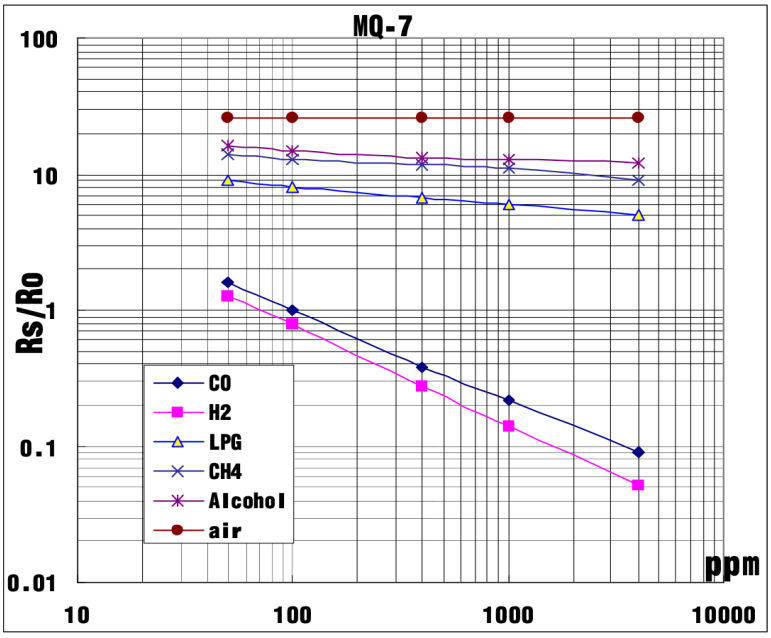

Рис. 3.1. Графік за допомогою якого проходить калібрування

Не менш важливим фактором є розміщення сенсора у просторі. Адже від правильності розташування пристрою, залежить коректність даних про вміст забруднюючих речовин. Пристрій необхідно встановлювати в місцях найбільшого скупчення людей або поблизу з передбачуваним джерелом забруднення. Однією з вимог для розміщення пристрою, є наявність Wi-Fi мережі з доступом до Інтернету, в іншому випадку дані не зможуть бути надіслані до веб-додатку.

Також, варто зауважити, що виміри, які проводяться пристроєм, є точковими, тобто сенсор не може визначати рівень забрудненості повітря на великій за площею території. Але ми можемо розбити ту територію на кілька секторів і провести декілька дослідів у цих секторах. За допомогою середнього арифметичного ми можемо знайти приблизне значення якості повітря на тій території.

Я провів дослідження на вулицях міста Рівне: на вулиці Соборній, на проспекті Миру та на вулиці Басівкутській. У таблиці 3.2 зображені дані щодо вмісту CO у повітрі (у ppm) на цих вулицях. Ці значення є середніми арифметичними даних які були отримані при 5ти хвилинних замірах на кожній з тих вулиць.

Табл. 3.2. Заміри концентрації газу CO на вулицях міста Рівне

Вулиця Соборна ppm	*Вулиця Басівкутська ppm*	*Проспект миру ppm*
25.6	14.5	4.2
18.7	12.3	7.9
19.3	9.5	6.8

Аналіз повітря в місті Рівне вказує на те, що найзабруднішими є вулиці Басівкутська та Соборна (значення коливаються в межах 9-26 ppm).

Скоріш за все це пов'язано з великим потоком наземного транспорту на тих вулицях. Спираючись на ці дані, можна зробити попередній висновок, що в межах міста не простежується перевищення норми (нормальною концентрація

газу CO вважається якщо вона має значення до 35 ppm [20]) і місто можна вважати не забрудненим.

3.2 Загальний опис користувацького інтерфейсу веб-додатку розробленого на C#

У альтернативній версії додатку було значно удосконалено користувацький інтерфейс та було додано більше функцій. На початковій сторінці додатку (рис. 3.2) висвітлено дані (1) з датчика, id (2) та час (3).

Маркери (4) на карті вказують на те, де ті дані були взяті, а їхні кольори – на чистоту повітря:

- зелений – чисте;
- жовтий – мало забруднене;
- оранжевий – забруднене;
- червоний – дуже забруднене.

Також на сайті організована функція, яка дозволить вибрати кількість даних для перегляду на одній сторінці (6). За допомогою графи “Пошук” можна знайти дані про стан повітря за певними координатами, або ж – знайти локацію за рівнем забрудненості (5).

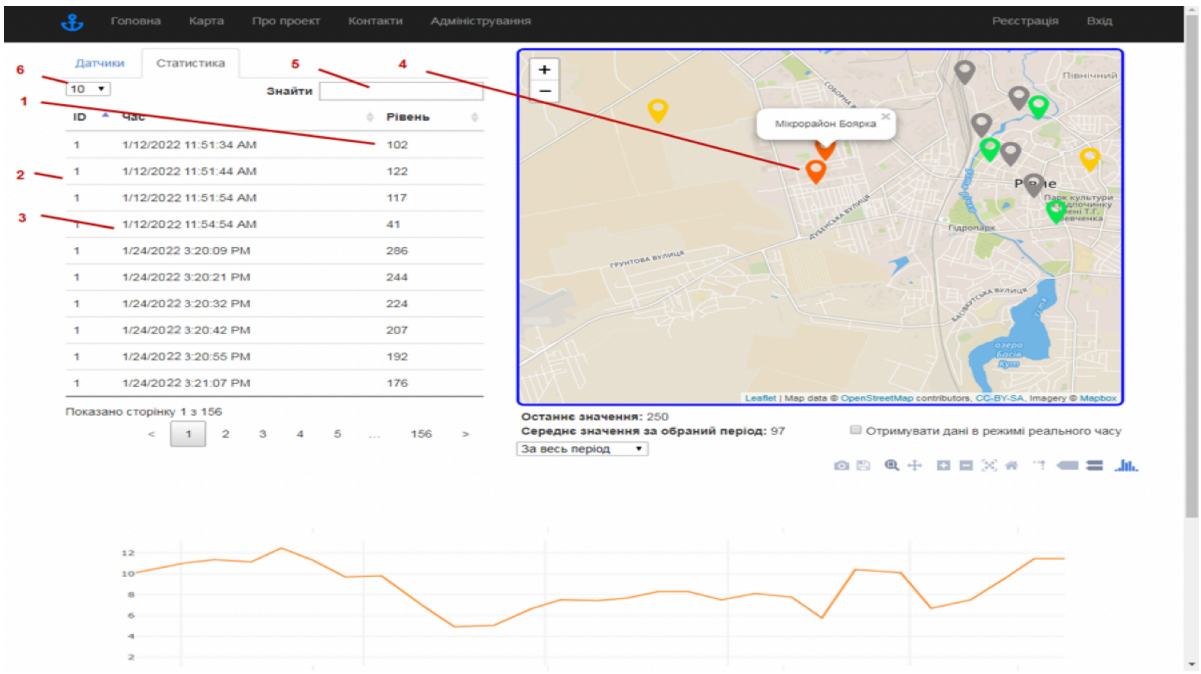

Рис. 3.2. Вкладка “Статистика”

Якщо змінимо вкладку з Статистика (2) на Датчики (рис. 3.3) (1), ми побачимо, що певному ID відповідає відповідна точка на карті, яка має свої координати. Також, якщо клацнути на будь-який датчик (3), то внизу з’явиться графік (4), який показує зміну забрудненості з певним проміжком часу.

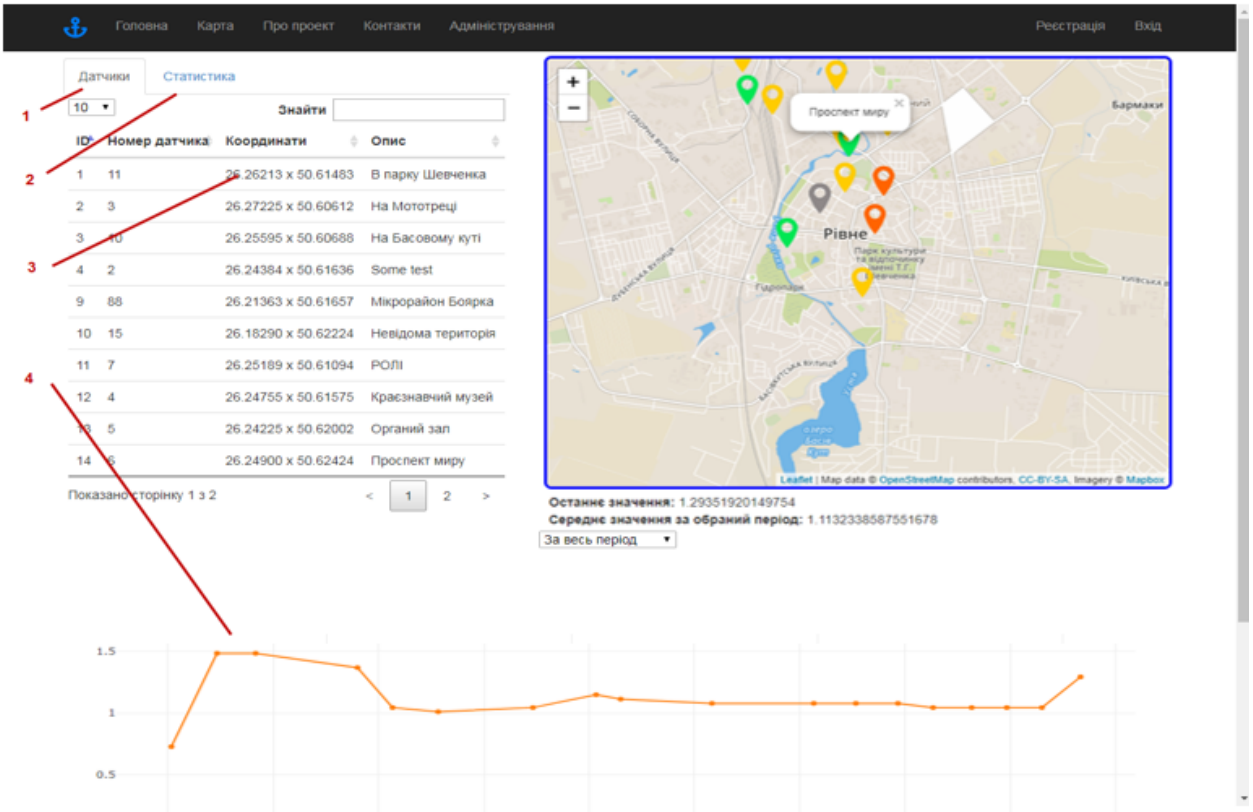

Рис. 3.3. Вкладка “Sensors”

Функції сторінки «Адміністрування» (рис. 3.6) доступні тільки адміністраторам сайту. Зокрема, це перегляд та видалення помилкових показників датчиків, перегляд та редагування місця розташування датчиків та їхніх ідентифікаторів.

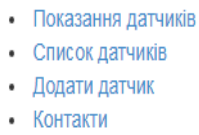

Рис. 3.6. Сторінка “Адміністрування”

На цій сторінці можемо змінювати ідентифікатор датчика (id) та його місцезнаходження, а також додавати нові датчики.

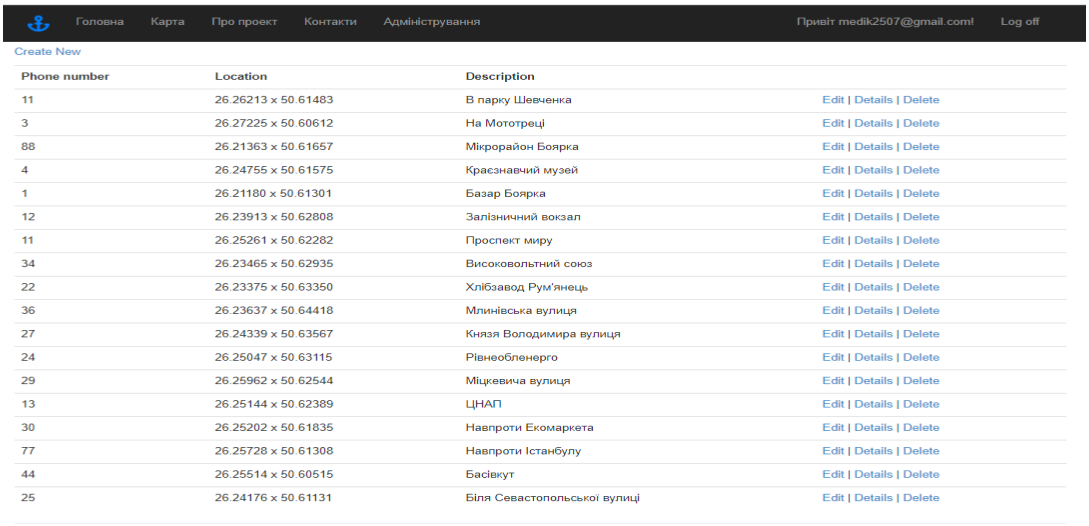

Рис. 3.7. Редагування датчиків

3.3 Ідеї удосконалення системи

На даний момент дані про температуру та вологість є константами. Ці дані не беруться з жодних сервісів, але в майбутньому першочерговою ціллю буде взяття цих даних з сервісу OpenWeatherMap та використання цих даних для корегування значення яке описує якість повітря.

Наступним етапом буде розробка телеграм бота адже, зазвичай, користувачам навіть простіше не виходячи з меседжера дізнатись потрібну їм інформацію. А ще на сторінці сайту буде викладений туторіал для розробки пристрою та його використання щоб кожен користувач міг з легкістю моніторити якість повітря у потрібних йому місцях у будь-якій точці світу.

ВИСНОВКИ

Постійний розвиток виробничої діяльності призводить до виникнення нових виробництв, більшість із яких у процесі свого функціонування та після його завершення стають джерелами забруднення навколишнього середовища і становлять екологічну загрозу.

У зв'язку з низькою кількістю джерел інформації щодо вмісту шкідливих речовин у повітрі було розроблено інформаційно-технічну систему, яка надає інформацію про стан якості повітря.

Спроектовано прототип пристрою, який вимірює рівень забрудненості повітря в м. Рівне. Подано схему взаємодії елементів пристрою сканування якості повітря, де датчик MQ-7 взаємодіє з Wi-Fi модулем ESP8266.

Було використано не дорогі матеріали для створення системи спостереження за станом повітря. Датчик з Wi-Fi модулем обійшовся мені у ~7$.

Розроблено дві онлайн системи для перегляду даних щодо стану повітря за допомогою технології Play Framework та ASP.NET. Було порівняно методи передачі даних з датчика на сервер у цих реалізаціях.

Проведено експеримент з моніторингом якості повітря в межах міста Рівне.

Після аналізу повітря в межах м. Рівне не простежується сильне перевищення норми, і місто Рівне можна вважати не забрудненим.

Комерційна привабливість проекту може полягати у продажі або наданні аналітичної інформації ріелторам, для того щоб дізнатись в якому районі краще всього жити і відповідно змінювати від цього ціну на житло. Також така система може бути використана у «розумних містах».

У подальшій роботі вбачаю перспективним удосконалення створеної системи моніторингу за станом повітря, використовуючи інші технології, шукаючи найоптимальніші компоненти та способи їх використання.

Майбутні плани передбачають зміну архітектури, що дозволить під'єднувати пристрої від різних користувачів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. World's Air Pollution: Real-time Air Quality Index [Електронний ресурс]. – https://waqi.info. – Назва з екрана.
2. Andy Patrella. Learning Play! Framework 2 / Patrella Andy. – Birmingham: Packt Publishing, 2013. – 290p.
3. Metanit [Електронний ресурс]: [Інтернет-портал]. – Режим доступу: https://metanit.com/sharp/mvc5/1.1.php.
4. Martin Odersky, Lex Spoon, Bill Veners – Programming in Scala – artima – 2016 – 592p.
5. ScalaPy [Електронний ресурс]. – Режим доступу: https://scalapy.dev.
6. Загальна інформація про Akka Actors [Електронний ресурс]. – Режим доступу: https://medium.com/@maximsidorov/akka-о-чем-все-эти-акторы-и-зачем-вам-akka-cad22a30747 .
7. Entity Framework and Code First [Електронний ресурс]. - https://docs.microsoft.com/en-us/ef/ef6/modeling/code-first/workflows/new-database .
8. What is web socket and how it is different from the HTTP? [Електронний ресурс]. – https://www.geeksforgeeks.org/what-is-web-socket-and-how-it-is-different-from-the-http/ . – Назва з екрана.
9. MQ-7 Semiconductor Sensor for Carbon Monoxide [Електронний ресурс]: [сайт]. – Режим доступу: https://www.sparkfun.com/datasheets/Sensors/Biometric/MQ-7.pdf
10. MQ-9 Semiconductor Sensor for CO/Combustible Gas [Електронний ресурс]: [сайт]. – Режим доступу: http://www.haoyuelectronics.com/Attachment/MQ-9/MQ9.pdf.
11. Технічна документація MQ-135 [Електронний ресурс]: [сайт]. – Режим доступу: https://www.olimex.com/Products/Components/Sensors/SNS-MQ135/resources/SNS-MQ135.pdf.

12. MQ-138 Gas sensor [Електронний ресурс]: [сайт]. – Режим доступу:https://www.mysensors.org/dl/57c3ebeb071cb0e34c90057a/design/MQ-138.pdf.
13. Wi-Fi module ESP8266 [Електронний ресурс]. – Режим доступу: https://arduino-ua.com/prod1492-Wi-Fi-modyl-nodemcu-esp8266.
14. Mapbox API [Електронний ресурс]. – Режим доступу: https://docs.mapbox.com/api/overview/
15. DOU Проектор: EcoCitizens – система, що попереджує екологічні катастрофи [Електронний ресурс]. – Режим доступу: https://dou.ua/lenta/articles/dou-projector-ecocitizens/.
16. Air quality system [Електронний ресурс]: [Веб-сайт]. – Режим доступу: http://aircasting.org/.
17. Smart Citizen [Електронний ресурс]: [Веб-сайт]. – Режим доступу: https://smartcitizen.me.
18. Breezometer [Електронний ресурс]: [Веб-сайт]. – Режим доступу: https://breezometer.com.
19. Using Correction factor [Електронний ресурс]. – Режим доступу: https://github.com/GeorgK/MQ135/blob/master/MQ135.cpp . – Назва з екрана.
20. What is the average level of carbon monoxide in homes? [Електронний ресурс]. – Режим доступу: https://www.epa.gov/indoor-air-quality-iaq/what-average-level-carbon-monoxide-homes. – Назва з екрана.