НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра автоматики та управління в технічних системах

До захисту допущено:

Завідувач кафедри

_____ Олександр РОЛІК

«___»_____20__ р.

Дипломний проєкт

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Програмне забезпечення інформаційно-комунікаційних систем»
спеціальності 121 «Інженерія програмного забезпечення»
на тему: «Система аналізу даних приватної метеорологічної станції»

Виконав (-ла):
студент (-ка) IV курсу, групи ІТ-61
Хижняк Олександр Сергійович _____

Керівник:
Старший викладач кафедри АУТС
Тимофєєва Юлія Сергіївна _____

Рецензент:
Посада, науковий ступінь, вчене звання,
Прізвище, ініціали _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____

Київ – 2020 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра автоматики та управління в технічних системах

Рівень вищої освіти – перший (бакалаврський)

Спеціальність - 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма «Програмне забезпечення інформаційно-комунікаційних систем»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Олександр РОЛІК

«___»_____20__ р.

ЗАВДАННЯ

на дипломний проєкт студенту

Хижняку Олександру Сергійовичу

1. Тема проєкту «Система аналізу даних приватної метеорологічної станції», керівник проєкту Тимофєєва Юлія Сергіївна, старший викладач кафедри АУТС, затверджені наказом по університету від «07» травня 2020 р. № 1081-с

2. Термін подання студентом проєкту: 9 червня 2020 р.

3. Вихідні дані до проєкту: мова програмування C#, середовище програмування Visual Studio, фреймворк .NET Framework версії 4.7.2, мікроконтролер ESP8266, датчики DHT22 та BMP280, хмарний ресурс Azure

4. Зміст пояснювальної записки: огляд існуючих рішень та аналіз вимог, технології та рішення використані при розробці, програмна реалізація системи, розгортання системи, графічний інтерфейс користувача, тестування додатку

5. Перелік графічного матеріалу: ER діаграма бази даних, діаграма загального алгоритму роботи програми, діаграма класів-моделей, діаграма класів-команд Telegram боту

6. Дата видачі завдання: 3 березня 2020 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	Вивчення рекомендованої літератури	10.04.2020	
2.	Аналіз існуючих рішень	13.04.2020	
3.	Розроблення структурної схеми системи аналізу даних домашньої метеорологічної станції	20.04.2020	
4.	Виготовлення макету системи збору даних погодних показників	24.04.2020	
5.	Розроблення функціональної схеми системи	26.04.2020	
6.	Реалізація програмної частини системи та подальше тестування	29.04.2020	
7.	Тестування програми	14.05.2020	
8.	Виконання графічних документів	17.05.2020	
9.	Оформлення пояснювальної записки	18.05.2020	
10.	Подання проєкту на основний захист	15.06.2020	

Студент Олександр ХИЖНЯК

Керівник Юлія ТИМОФЄЄВА

АНОТАЦІЯ

Хижняк О.С. Система аналізу даних приватної метеорологічної станції. КПІ ім. Ігоря Сікорського, Київ, 2020.

Проєкт містить 6 розділів, 61 сторінка, 35 рисунків, 7 таблиць, посилання на 18 літературних джерела, додатки та 4 кресленики.

Ключові слова: приватна, метеостанція, погодні показники, мікроконтролер ESP8266, датчик DHT22, датчик BMP280, Telegram, бот, Microsoft Azure, C#.

Об’єктом розробки є система аналізу даних для приватна метеостанція.

Мета розробки – створити застосунок зі зручним інтерфейсом, який буде аналізувати погодні показники минулих днів та годин.

У дипломному проєкті розроблено систему, яка в режимі реального часу збирає дані погодних показників, аналізує їх та виводить користувачу у вигляді тексту або у вигляді графіків за останні дні або 12 годин. Проведено ретельний огляд існуючих рішень і на основі цього виконано аналіз вимог, які б задовольняли всім потребам як звичайного користувача, так і садівника-любителя або ж працівника на господарстві. За апаратну частину було обрано мікроконтролер ESP8266 та датчики DHT22 і BMP280 для вимірювання температури, вологості повітря, атмосферного тиску та теплового індексу. Значну увагу було приділено серверній частині, яка розгортається в хмарному ресурсі Microsoft Azure.

Отримані результати можуть бути розширені масштабованістю апаратної частини та подальшою розробкою функціональної системи. Будуть корисними як звичайній, так і зацікавленій в господарстві людині.

SUMMARY

Khyzhniak O.S. Data analysis system for private weather station. Igor Sikorsky KPI, Kyiv, 2020.

The project contains 6 sections, 61 pages, 35 figures, 7 tables, links to 18 literary sources, annexes and 4 design documents.

Keywords: private, weather station, weather data, microcontroller ESP8266, sensor DHT22, sensor BMP280, Telegram, bot, Microsoft Azure, C#.

The object of the development is the data analysis system for private weather station.

The purpose of the development – create an application with user friendly interface for analyzing weather data of the last days and hours.

The graduation project developed the system, which collects weather data in real time, analyzes it and displays results like text or like charts for the last days or 12 hours. The thorough analysis of existing solutions and applications is a base of requirements that will satisfy all simple people, amateur gardener's or enterprise worker's needs. Microcontroller ESP8266 and sensors DHT22 and BMP280 were chosen like an embedded part of system for temperature, humidity, atmosphere pressure and heat index measurements. Considerable attention was paid to the server part, which is deployed in the cloud platform Microsoft Azure.

The results obtained can be expanded by scalability of embedded part and functional system future development. It will useful as for simple people as for interested in the enterprise person.

Поз.	Формат	Позначення	Найменування	Кількість аркушів	№ прим.	Примітки
1			Документація загальна			
2			Знову розроблена			
3						
4	А4	ІТ61.250БАК.004 ПЗ	Система аналізу даних приватної	61		
5			метеорологічної станції.			
6			Пояснювальна записка			
7	А3	ІТ61.250БАК.004 Д1	Система аналізу даних приватної	1		
8			метеорологічної станції. ER			
9			діаграма бази даних			
10	А3	ІТ61.250БАК.004 Д2	Система аналізу даних приватної	1		
11			метеорологічної станції. Діаграма			
12			загального алгоритму роботи			
13			програми			
14	А3	ІТ61.250БАК.004 Д3	Система аналізу даних приватної	1		
15			метеорологічної станції. Діаграма			
16			класів-сутностей			
17	А3	ІТ61.250БАК.004 Д4	Система аналізу даних приватної	1		
18			метеорологічної станції. Діаграма			
19			класів-команд Telegram боту			
20						
21						
22						
23						

					ІТ61.250БАК.004 ТП		
Змн.	*Лист*	*№ докум.*	*Підпис*	*Дата*			
Розроб.		Хижняк О.С.			Система аналізу даних приватної метеорологічної станції. Відомість технічного проєкту	*Літ.*	*Лист* 1 / *Листів* 1
Перевір.		Тимофєєва Ю.С.				КПІ ім. Ігоря Сікорського ФІОТ гр. ІТ-61	
Н. Контр.							
Затверд.							

Пояснювальна записка
до дипломного проєкту
на тему: «Система аналізу даних приватної метеорологічної станції»

Київ – 2020 року

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

БД – база даних.

Бот – спеціальна програма у вигляді окремого, спеціально створеного користувача для автоматизації прийому та обробки повідомлень.

Веб-браузер – програмне забезпечення для взаємодії з різноманітною інформацією в мережі Інтернету.

Десктопний (англ. Desktop software) – вид ПЗ, що встановлюється на комп'ютер, а не виконується в браузері.

Крос-платформеність – властивість програмного забезпечення працювати більш ніж на одній програмній або апаратній платформі.

ОС – операційна система.

ООП – об'єктно-орієнтоване програмування.

Хостинг (хостити) – послуга розміщення фізичної інформації на віддаленому сервері, що постійно перебуває в мережі інтернету.

API (англ. Application Programming Interface) – прикладний програмний інтерфейс.

Backend – програмно-апаратна частина сервісу, додатку.

BDD (англ. Behavior Driven Development) – розробка через поведінку.

Code First – підхід, в якому спочатку описуються моделі, а вже з них генеруються таблиці бази даних.

Embedded – апаратна частина системи.

ER діаграма (англ. Entity-Relationship) – модель даних, яка дозволяє описувати концептуальні схеми предметної області.

Frontend – клієнтська частина користувацького інтерфейсу системи.

GND (англ. Ground) – земля.

GPIO (англ. General-purpose input/output) – інтерфейс вводу/виводу загального призначення.

HTTP (англ. HyperText Transfer Protocol) – протокол передачі даних.

I^2C (англ. IIC) – послідовна двопровідна шина для міжмікросхемного управління.

IaaS (англ. Infrastructure-as-a-Service) – інфраструктура як сервіс.

IFTTT (англ. If This Then That) – сервіс, що за допомогою правил типу «якщо це, тоді те» дає змогу об'єднати різні додатки у взаємодії.

In-memory – підхід зберігання даних в оперативній пам'яті.

JIT-компіляція (англ. Just-in-time) – технологія збільшення продуктивності програмних систем.

JSON (англ. JavaScript Object Notation) – текстовий формат обміну даними на базі правил мови JavaScript.

Lazy loading – ліниве завантаження.

Messenger (далі месенджер) – сервіс для обміну текстовими повідомленнями.

Micro USB – один з варіантів роз'ємів типу USB.

ORM (англ. Object-Relational Mapping) — технологія, яка зв'язує бази даних з концепціями ООП, створюючи «віртуальну об'єктну базу даних».

PaaS (англ. Platform-as-a-Service) – платформа як сервіс.

REST (англ. Representational State Transfer) – архітектурний стиль взаємодії компонентів розподіленого додатку в мережі.

SaaS (англ. Software-as-a-Service) – програмне забезпечення як послуга.

SCL (англ. Serial CLock) – двонаправлена лінія синхронізації.

SDA (англ. Serial DAta) – двонаправлена лінія передачі даних.

SDK (англ. Software Development Kit) – набір засобів розробки.

SQL (англ. Structured query language) – мова структурованих запитів.

SPI (англ. Serial Peripheral Interface) – послідовний периферійний інтерфейс.

USB (англ. Universal Serial Bus) – універсальна послідовна шина.

Use-Case діграма – відображає відношення між акторами та прецендентами в системі.

VCC (або V_{cc}) – апаратних вхід для позитивної напруги.

XML (англ. Extensible Markup Language) – стандарт побудови мов розмітки ієрархічно структурованих даних.

ВСТУП

На сьогоднішній день, питання передбачення погоди або ж просто знання поточного стану погодних умов дуже гостро стоїть для всього людства, незалежно від їх роду діяльності та місцезнаходження. Наразі вже недостатньо просто подивитись у вікно і оцінити ситуацію. Можливо зараз дощу немає і небо світле, але через якусь одну годину може бути злива з грозою. Або ж сидячі влітку в прохолодній квартирі важко визначити чи потрібно брати з собою головний убір, чи слід взяти пляшку води, аби не отримати тепловий удар. Також будучи садівником-любителем можна випадково не помітити початок зливи, що може призвести до серйозних наслідків для деяких культур рослин.

Для того, щоб правильно оцінити погодні умови недостатньо виду з вікна, треба знати температуру, вологість повітря, атмосферний тиск, вирахуваний на основі перших двох показників, тепловий індекс, і дуже бажано ще знати динаміку їх зміни за останні 12-24 години. Різка зміна атмосферного тиску при світлому та безхмарному небі натякає на те, що слід взяти з собою парасольку чи прикрити чутливі культури рослин, овочів, ягід тентом або ж при відносно невисокій температурі влітку вологість могла в цей час збільшуватись, а отже і шанс отримати тепловий удар, навіть одягнувши головний убір, зростає. Якщо ж користувач працює на сільськогосподарському виробництві, то йому критично необхідно знати зміну цих параметрів за останні дні, тижні, місяці, щоб своєчасно спрогнозувати майбутні незручності, адже отримати наслідки, будучи садівником в себе на дачі — це одна справа, а відповідати за великий урожай, що коштує чималих грошей — справа абсолютно інших масштабів.

Проаналізувавши наявні засоби для задоволення даних потреб виявилось, що всі вони або достатньо легкі у використанні, але мають малий функціонал, або доволі складні, некомпактні і на додачу дорогі. Тому розробка продукту, яка б дозволяла, просто встановивши на потрібне місце компактну метеостанцію, маючи лише смартфон та підключення до інтернету, оперативно та в одне натискання кнопки отримувати і поточні, і проаналізовані за попередній проміжок часу у ви-

гляді графіків динаміки зміни погодні параметри, дасть можливість як звичайній людині, так і працівнику зацікавленого в цьому підприємства або господарства мати зручний та зрозумілий інтерфейс та знання для прийняття тих чи інших рішень, які дуже залежать від погодних умов.

Для вирішення цієї задачі було використано наступний ряд технологій: фреймворк ASP.NET, за допомогою якого було реалізовано серверну частину застосунку мовою програмування C#, мікроконтролер ESP8266 та датчики для збору погодних показників DHT22 та BMP280, які разом утворюють одну повноцінну метеостанцію, а в ролі графічного інтерфейсу, з яким взаємодіє користувач, виступає набираючий популярність месенджер Telegram з його функціональністю під назвою Бот. Боти дозволяють надати новий, що стає все популярнішим з кожним днем, спосіб взаємодії с програмним продуктом за допомогою текстових повідомлень. На відміну від інших аналогічних сервісів, Telegram пропонує зручне API для розробки ботів. Для того щоб забезпечити надійну та безперебійну роботу застосунку, було вирішено використати деякі рішення, що пропонує хмарна платформа Microsoft Azure.

1 ОГЛЯД ІСНУЮЧИХ РІШЕНЬ ТА АНАЛІЗ ВИМОГ

Різноманітних додатків, які б показували погоду, будь то за сьогодні, вчора чи завтра є дуже багато. Вони можуть бути у вигляді веб-сайтів, окремих додатків або віджетів (елемент графічного інтерфейсу операційної системи), але нас цікавлять саме окремі приватні системи для вимірювання погодних показників, що надають не просто програмне забезпечення, а окремий прилад, який можна встановити як в себе вдома або на подвір'ї, так і в саду, городі, теплиці, господарстві тощо і мати відповідні дані саме цієї області застосування.

1.1 Огляд існуючих рішень

1.1.1 Приватна метеостанція «Oregon Scientific Full Weather Station»

Дана модель приватної метеостанції є бездротовим десктопним рішенням для збору локальних даних погодних показників[1]. Як показано на рисунку 1.1, дана модель має окремий екран, на якому відображаються дані, і за один погляд можна побачити прогноз погоди, час, швидкість вітру та напрям, температуру зовні та всередині приміщення, вологість повітря та атмосферний тиск. Все це надається у вигляді тексту або гістограм як кругових, так і вертикальних.

Також має роз'єм USB, щоб підключатись до комп'ютеру на базі операційної системи Windows, даючи змогу зберігати, аналізувати та розповсюджувати погодні дані. Є можливість зберігати дані за останні 7 днів прямо в десктопній консолі.

Переваги:

— збирає дані великої кількості погодних параметрів;

— має окремий десктопний прилад (консоль) для нагляду за погодними показниками;

— вбудоване сховище даних на 7 днів.

Недоліки:

— не має доступу в інтернет;

— не має мобільного додатку;

— взаємодіє лише з операційною системою Windows;

— датчик вологості зазвичай показує дані на 5-15% нижчі.

Рисунок 1.1 — «Oregon Scientific Full Weather Station» та всі її складові [2]

1.1.2 Приватна метеостанція «BloomSky»

Наступним прикладом приватної метеостанції, що широко розповсюджена на ринку є «BloomSky». В даному випадку, у первинному вигляді метеостанція має вигляд як на рисунку 1.2, в яку, окрім як датчиків, додано камеру, що розташовується на вулиці і направляється в небо у необхідному напрямку. Випливаючи з побаченого на рисунку 1.2, камера та всі інші складові системи працюються за рахунок енергії, що надходить із сонячних панелей, тому непотрібно підключати їх до загальної мережі електропередачі. На додачу до вимірювання температури, вологості повітря та атмосферного тиску, камера, що інтегрована в BloomSky робить фото неба кожні 5 хвилин[3]. Проаналізувати дані можна за допомогою інтернету або з окремого додатку спеціально для BloomSky. Сумісність з IFTTT дає можливість використовувати дані, що збираються для запуску інших приладів з комплексу розумного дому.

Переваги:

— вбудована веб-камера, що дозволяє слідкувати за погодою в реальному часі;

— все працює на енергії від сонячних батарей;

— є можливість взаємодії з іншими системами розумного дому.

Недоліки:

— слабкий Wi-Fi зв’язок;

— камера не підтримує високоякісні знімки;

— показник температури різко зростає, при прямому контакті із сонячними променями;

— для вимірювання шансу опадів, швидкості та напряму вітру треба окремо придбати обладнання;

— не показує якою була погода за минулі дні;

— не відображає графіки динаміки зміни погодних показників.

Рисунок 1.2 — Камера, як одна з частин погодної системи «BloomSky» [4]

1.1.3 Приватна метеостанція «Netatmo Weather Station»

Ще одним прикладом системи для отримання погодних показників, що використовуються в розумних домах або на господарствах є система Netatmo

Weather Station[5]. Вона складається з переносної базової станції та з датчиків вологості повітря, температури, атмосферного тиску, сили та напряму вітру, об’єму вуглекислого газу в повітрі тощо, з якими вона пов’язана за допомогою бездротового з’єднання через інтернет. Як видно з рисунку 1.3 система датчиків даного продукту розбита на декілька частин, а не представляє собою одну компактну одиницю.

Рисунок 1.3 — Датчики системи «Netatmo Weather Station» [6]

Самі ж датчики можуть розміщуватись як на вулиці, так і в домі, залежно від потреб користувача. А результати відображаються в окремому додатку, який встановлюється на смартфон і на який повертаються дані в реальному часі.

Переваги:

— не потребує додаткового налаштовування, за це відповідає спеціальне програмне забезпечення;

— вимірює температуру, вологість повітря, атмосферний тиск, швидкість та напрям руху вітру, насичення вуглекислим газом повітря;

— можуть бути інтегровані в загальну систему розумного дому;

— дуже точне вимірювання температури, яке має максимальну похибку 0.01℃.

Недоліки:

— має нестабільно реалізований зв’язок до мережі інтернету;

— датчик вітру часто повертає значення швидкості повітря як занадто низьку;

— датчик вологості повітря може повертати хибне значення у вологому середовищі;

— графічний інтерфейс додатку на смартфоні (див. рисунок 1.4) повертає лише дані у текстовому вигляді за сьогоднішній день та наступні без графіків динаміки зміни погодних показників та даних за минулі дні.

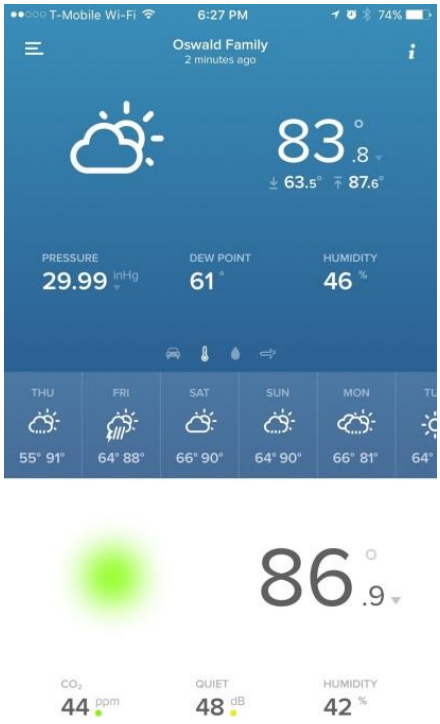

Рисунок 1.4 — Приклад графічного інтерфейсу системи [6]

1.2 Аналіз вимог до даного проєкту

Перш ніж починати реалізацію програмного продукту, необхідно визначитись з функціоналом, який повинен мати застосунок, щоб задовольнити потреби, які було поставлено задачею. Не зайвим буде представити майбутній функціонал у вигляді Use-Case діаграми використання, яку представлено на рисунку 1.5, що стисло описує можливості користувачів різних ролей. З рисунку видно, що однією

з важливих можливостей адміністратора є можливість підключити нову станцію збору даних для більшого розмаху системи.

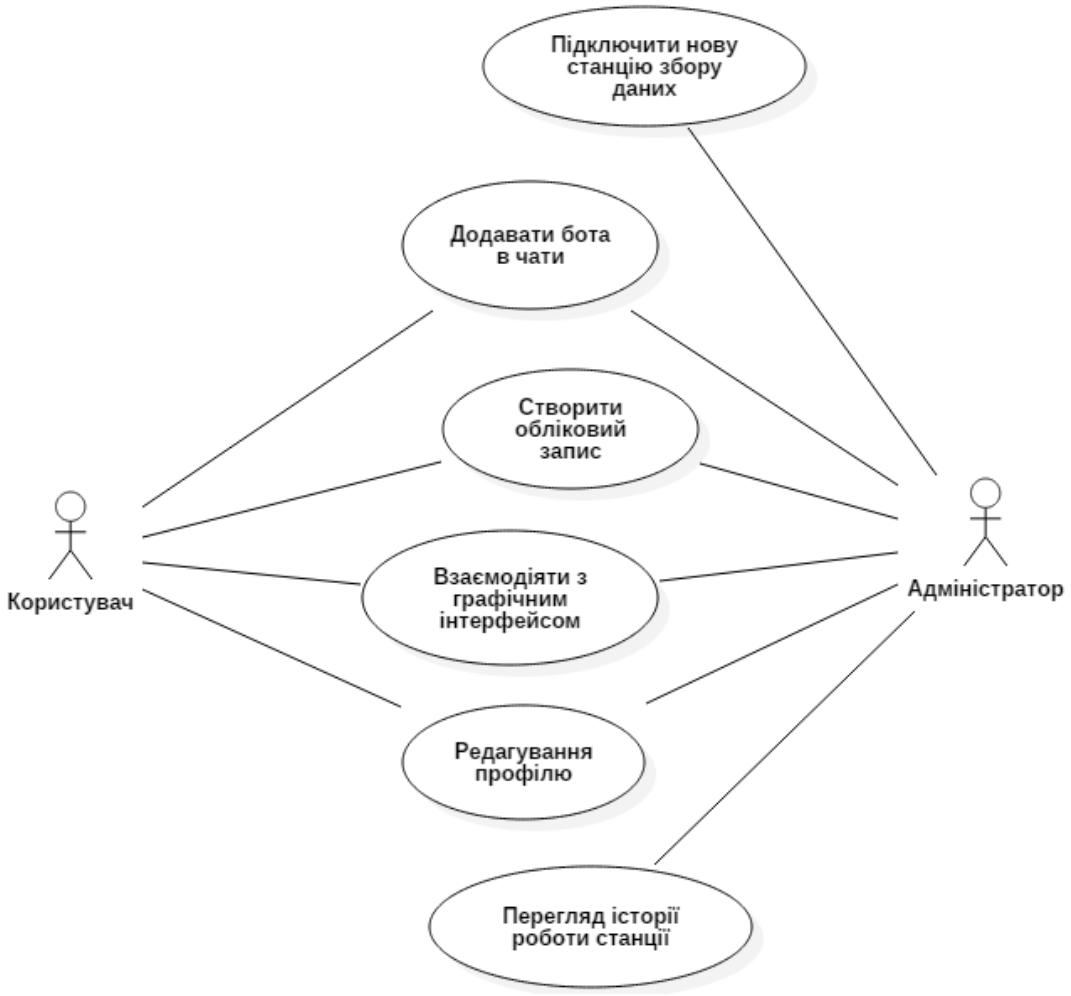

Рисунок 1.5 — Use-Case діаграма використання

Виходячи з проведеного аналізу існуючих рішень, їхнього функціоналу та на основі представленої вище Use-Case діаграми можна виокремити такий список бажаного функціоналу, з яким буде взаємодіяти звичайний користувач (в даному випадку команд Telegram-бота):

— виведення поточної температури;
— виведення поточної вологості повітря;
— виведення поточного значення атмосферного тиску;
— виведення поточного теплового індексу;

— відображення графіку динаміки зміни одного з чотирьох погодних показників описаних вище за останню кількість днів, обрану користувачем;

— відображення таблиці-помічника про значення теплового індексу;

— відображення графіку динаміки зміни атмосферного тиску та вологості повітря за останні 12 годин з подальшим оповіщенням про ризик опадів;

— відображення графіку динаміки зміни температури, вологості повітря та теплового індексу за останні 12 годин з подальшим оповіщенням про ризик отримання теплового удару.

1.3 Висновки до розділу

Проаналізувавши існуючі рішення можна дійти висновку, що деякі показують лише погодні показники без їхньої візуалізації у вигляді графіків динаміки зміни, деякі надають користувачеві перелік великої кількості показників, як от сила та напрям вітру, але інформації настільки багато, що відображається вона некомпактно і незручно для сприймання людським оком.

Продивившись велику кількість Telegram-ботів, які б показували погоду виявилось, що рішення зробити Telegram-бота є не тільки зручним з точки зору користувача, адже це наче повсякденне спілкування з іншою людиною за допомогою месенджера, але й сучасне та нове, бо насамперед їх не дуже велика кількість, а більшість з них є або одноденними розробками, або тестовими варіантами любителів, які роблять це виключно для себе без створення зручного інтерфейсу.

2 ТЕХНОЛОГІЇ ТА РІШЕННЯ ВИКОРИСТАНІ ПРИ РОЗРОБЦІ

2.1 Технології, використані при розробці серверної частини

Через специфіку програми та для більш зручного та легшого розгортання системи в хмарному ресурсі Microsoft Azure було прийнято рішення про написання серверної частини за допомогою платформи ASP.NET мови програмування C#. .NET — це безкоштовна платформа розробки програмного забезпечення будь-якої складності, що складається з інструментів, мов програмування та бібліотек для створення додатків різних типів. ASP.NET — це платформа розробки веб-додатків, до складу якої входять: веб-сервіси, програмна інфраструктура, модель програмування. Головною перевагою даної платформи є те, що вона підтримується компанією Microsoft та існує на ринку достатню кількість років, аби бути стабільним рішенням для розробки. ASP.NET входить до складу платформи .NET Framework, розширює її, і є розвитком більш старої технології Microsoft ASP[7].

Також ASP.NET має перевагу у швидкості в порівнянні зі скриптовими технологіями, такими як Python або JavaScript у вигляді Node.js, тому що при першому зверненні код компілюється і поміщується в спеціальний кеш, і згодом тільки виконується, не вимагаючи витрат часу на синтаксичний аналіз, оптимізацію тощо.

2.2 Microsoft Azure

Microsoft Azure — це одна з найпопулярніших на сьогоднішній день хмарних платформ, яка поступово завойовує ринок хмарних обчислень. Azure — це комплексна хмарна платформа, на якій можна розміщувати існуючі додатки та оптимізувати нову розробку сервісу[8]. Як і будь-яка інша хмарна платформа вона може виступати в якості хостинга для різноманітних додатків, а також надає доступ до великої кількості різних хмарних сервісів. Azure може навіть покращити локальні додатки. Воно інтегрує хмарні служби, необхідні для розробки, тесту-

вання та розгортання додатків, а також управління ними, одночасно використовуючи переваги хмарних обчислень.

В цілому, платформа надає всі моделі послуг хмарних обчислень: це і SaaS[9], і IaaS[10], і PaaS[11] (див. рисунок 2.4). Якщо брати окремо кожну модель, то в даній роботі використано щось з кожної з них. Більшість же сервісів взято з PaaS послуги даної платформи для розгортання додатку на віддаленому сервері для постійної роботи додатку та для зберігання і обробки даних. Платформа як послуга (PaaS) — це повноцінне середовище розробки та розгортання в хмарі з ресурсами, які дозволяють надавати будь-які додатки, від простих хмарних додатків до просунутих хмарних програм промислового класу. Як і у випадку з IaaS, PaaS включає інфраструктуру (сервери, сховища та мережеве обладнання), а також програмне забезпечення проміжного шару, засоби розробки, бізнес-аналітику, служби системи управління базами даних та багато іншого.

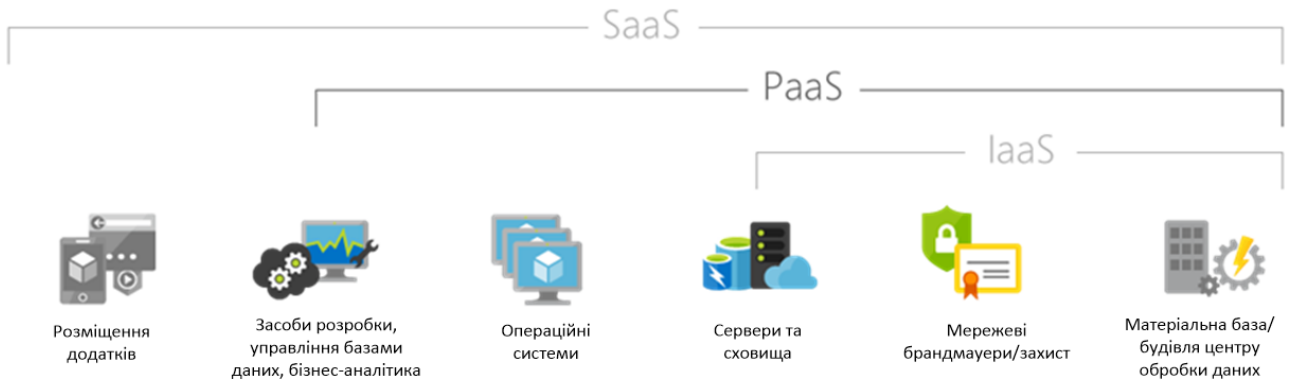

Рисунок 2.1 — Моделі хмарних обчислень та їхні можливості [9]

2.3 Технології, використані для роботи з даними

2.3.1 Entity Framework

Для зручності роботи з базою даних, а саме операції створення, читання, оновлення або видалення даних, маніпуляції в середині самого коду серверної частини було прийнято рішення обрати об’єктно-орієнтовану технологію доступу до даних Entity Framework.

Entity Framework – це так зване об’єктно-реляційне відображення, що є рішенням від компанії Microsoft на базі платформи .NET Framework. Якщо традиційні засоби ADO.NET дозволяють створювати підключення, команди та інші об’єкти для взаємодії з базами даних, то Entity Framework являє собою більш високий рівень абстракції, який дозволяє абстрагуватись від самої бази даних та працювати з даними незалежно від типу сховища. Якщо на фізичному рівні ми оперуємо таблицями, індексами, первинними та зовнішніми ключами, то на концептуальному рівні, який нам надає Entity Framework, ми вже працюємо з об’єктами[12].

Центральної концепцією Entity Framework є поняття сутності. Сутність представляє собою набір даних, асоційованих з певним об'єктом. Тому дана технологія передбачає роботу не з таблицями, а з об'єктами і їх наборами.

Будь-яка сутність, як і будь-який об'єкт з реального світу, має низку властивостей. Наприклад, якщо сутність описує людину, то ми можемо виділити такі властивості, як ім'я, прізвище, зріст, вік, вага. Властивості необов'язково представляють прості дані чисельного типу, а й можуть представляти більш комплексні структури даних. І у кожної сутності може бути одна або кілька властивостей, які будуть відрізняти цю сутність від інших і будуть унікально визначати цю сутність. Подібні властивості називають ключами.

2.3.2 Microsoft Azure Cache for Redis

Redis — це in-memory сховище структур даних, яке використовується як нереляційна база даних, кеш або брокер повідомлень. Зберігання даних за допомогою даного сервісу відбувається у вигляді ключ-значення, тим самим Redis є чудово масштабованим по горизонталі, що зручно, якщо даних з кожним разом стає все більше, а їх типи стають різноманітнішими[13]. Однак дана версія призначена для локального використання. Так як backend частина додатку розгортається на віддаленому сервері середовища Microsoft Azure, а змоги встановити там локально Redis немає, то для таких випадків є аналог даного рішення — Azure

Cache for Redis. Функціонал дане рішення має той же самий, що і звичайний Redis.

В проєкті воно використовується для зберігання тимчасових даних, що надходять на сервер від мікроконтролеру для більш зручного оперування ними, коли дані ще не занесені в базу даних, щоб зменшити навантаження на базу та для більш швидкого відгуку додатку на запит користувача відобразити результат тої чи іншої функції.

2.3.3 Microsoft Azure SQL Server

Azure SQL Server є хмарним аналогом реляційної бази даних Microsoft SQL Server, оскільки все це є продуктами компанії Microsoft. Так само як і Microsoft SQL Server, Microsoft Azure SQL Databases використовує T-SQL в якості мови запитів. На відмінну від багатьох схожих сервісів інших виробників, які для доступу до сервісу з машини використовують протокол HTTP REST, даний сервіс під'єднується за допомогою протоколу Tabular Data Stream (TDS). Користувач може надсилати Transact SQL запити за допомогою TDS до сервісу Windows Azure SQL Databases, і це дозволяє додаткам використовувати сервіс так само, як вони б використовувати локальний SQL Server. Але оскільки Windows Azure SQL Databases є сервісом, його адміністрування має свої особливості. На відмінну від адміністрування локального SQL Server, Windows Azure SQL Databases розділяє логічний та фізичний аспекти адміністрування. Клієнт продовжує адмініструвати БД, керувати логінами, користувачами та ролями, але за обладнанням піклуються Microsoft[14]. Як і в попередньому пункті, локально використовувати базу даних звісно можна, але доведеться тримати комп'ютер постійно ввімкненим, адже він відіграє роль серверу, отже, можливість локального використання відпадає, а тому було прийнято рішення створити віддалений варіант реляційної бази даних з того ж середовища, що і технології для розробки серверної частини, а саме .NET, адже в такому випадку ризик несподіваних проблем з підключенням та взаємодією різко зменшується. А задля покриття поставлених вимог вирішено зберігати

дані окремо для кожного погодного показнику, а саме для температури, вологості повітря, атмосферного тиску та теплового індексу по 2 таблиці:

— дані за кожну годину за останні 24 години;

— середнє значення за день та за ніч кожної доби минулих днів.

Оскільки Azure SQL Database є реляційною базою даних, то створення нових таблиць з усіма необхідними правилами та типами даних не складає труднощів.

2.4 Технології використані при розробці клієнтської частини

2.4.1 Telegram

Задля зручності, швидкодійності та уникнення зайвих дій для отримання інформації було прийнято рішення використати сторонній безкоштовний продукт, яким є Telegram.

Telegram – це крос-платформовий месенджер, що дозволяє швидко та безпечно від прослуховування обмінюватись повідомленнями і медіа файлами багатьох форматів. Використовується пропрієтарна серверна частина із закритим кодом. Крім стандартного обміну повідомленнями в діалогах і групах, в месенджері можна зберігати необмежену кількість файлів різного типу, вести канали (мікроблоги) та створювати і використовувати ботів[15].

Використання саме цієї технології обумовлено низкою переваг та функціональних особливостей. По-перше, можна надсилати не лише текст, а й мультимедійні файли. По-друге, суттєвою перевагою є підтримка не тільки мобільних засобів, а й інших платформ, таких як Windows, Mac, Linux або ж звичайний веб-браузер. Усі повідомлення будуть автоматично синхронізуватись між пристроями. По-третє, за допомогою спеціального API можна створювати ботів, взаємодія з якими виглядає наче спілкування з іншою людиною. На сьогоднішній день, даний вид графічного інтерфейсу набирає популярність серед сторонніх розробників, що надає широке розмаїття навчального матеріалу та гарантії в стабільності роботи. І

на останок, зв’язка ASP.NET додатку та платформи-месенджеру Telegram є перевіреною часом та великою кількістю користувачів.

2.4.2 JavaScript

JavaScript - це легковісна, інтерпретуєма або JIT-компільована, об’єктно-орієнтована мова з функціями першого класу. Її використання зумовлено простотою використання, наявністю динамічної типізації, на відмінну від мов програмування C# та C++, великою кількістю бібліотек, які дозволяються будувати гнучкі графіки, для чого вона й була обрана, та виконанням коду у веб-браузерах, що значно полегшує подальшу розробку, генерацію графіків та їхню підтримку в майбутньому, адже не складає ніяких труднощів згенерувати графік за правилами розмітки веб-сторінки, а потім скопіювати отриманий результат та відправити його як відповідь бота. За рахунок використання саме цієї мови програмування не доведеться завантажувати великі пакети для перших двох мов, що займають велику кількість місця.

Оскільки дана мова не компілюється спочатку, а одразу інтерпретується – це дає змогу швидше виконати поставлену задачу для побудови графіків та відображення результату користувачу. Також підвищиться і швидкість розробки даних графіків, оскільки не потрібно описувати класи, об’єкти та їх використання так, як це відбувається в мовах C# та C++

2.5 Технології, використані при розробці апаратної частини

2.5.1 Мікроконтролер ESP8266

Даний мікроконтролер із серії ESP було вибрано у зв’язку із поширеним використанням у світі, багатим функціоналом, порівняно з іншими моделями в тій же ціновій категорії та за рахунок наявності вбудованого Wi-Fi модулю (див. рисунок 2.1), який необхідний для передачі даних з датчиків вимірювання погодних показників DHT22 та BMP280 через інтернет на сервер[16]. Для програмування

цього мікроконтролеру було використано середовище програмування Arduino Software IDE, яке надає зручний функціонал для написання коду та прошивки чіпу мікроконтролеру та засоби мови програмування C++. Підключення до живлення та прошивка відбуваються за допомогою роз'єму типу Micro USB з використанням інтерфейсу SPI.

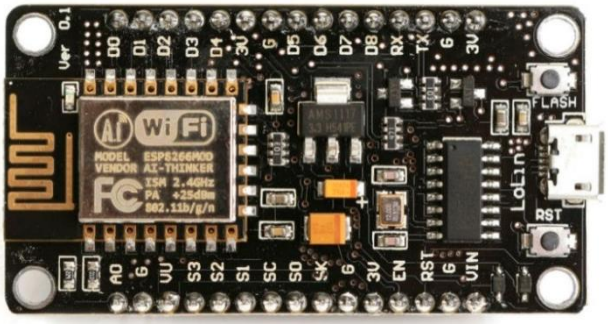

Рисунок 2.2 — Мікроконтролер ESP8266 [17]

2.5.2 Датчик DHT22

Датчик DHT22 / AM2302 використовується для вимірювання температури та вологості повітря із підвищеною точністю[18-19]. Дана модель має 4 апаратних входи/виходи (див. рисунок 2.2), один з яких використовується для живлення датчику з напругою в діапазоні 3.5 – 5.5В, другий — для передачі даних із використанням інтерфейсу GPIO, а четвертий — для заземлення. Третій апаратний вихід залишається неактивним. Датчик вимірює температуру в діапазоні -40…+80 градусів Цельсія з похибкою вимірювання ±0.5℃, а вологість повітря – 0-100% з похибкою ±2%.

DHT22 pins	
1	VCC
2	DATA
3	NC
4	GND

Рисунок 2.3 — Датчик DHT22 з апаратними входами/виходами [20]

2.5.3 Датчик BMP280

Модуль являє собою високоточний цифровий вимірювач атмосферного тиску та температури на базі мікро-чипу BMP280[21]. Особливістю даної моделі є наявність одразу двох послідовних інтерфейсів обміну даними: SPI та I^2C. В даному проєкті було реалізовано підключення за допомогою інтерфейсу I^2C. Також BMP280 може працювати в трьох режимах:

— Normal: в даному режимі модуль активується з відповідною періодичністю, виконує необхідні вимірювання і знову вимикається. Частота вимірювання задається програмним шляхом, а результат зчитується при необхідності. Саме цей режим було реалізовано;

— Sleep: режим максимально зниженого енергоживлення;

— Forced: цей режим дозволяє активувати модуль подачею зовнішнього управляючого сигналу. Після виконання вимірювань, модуль автоматично переходить в режим зниженого енергоживлення.

Дана модель має 6 апаратних входів/виходів (див. рисунок 2.3), але враховуючи, що підключення до мікроконтролеру виконання з використання інтерфейсу I2C, було використано лише перші 4. Вхід VCC — для живлення з напругою 1.7 – 3.6В, GND — для заземлення, SCL — двонаправлена лінія синхронізації, SDA — двонаправлена лінія передачі даних.

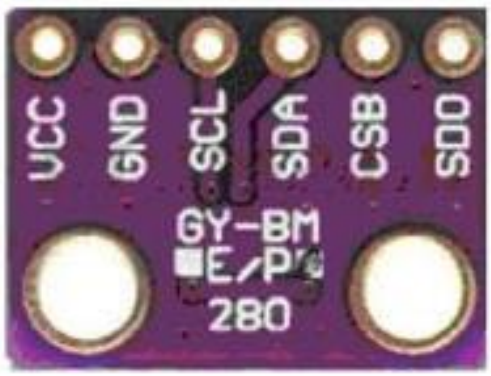

Рисунок 2.4 — Фронтальна сторона датчику BMP280 [21]

2.5.4 Живлення мікроконтролеру

Загалом, живити даний мікроконтролер можна від комп’ютеру за допомогою кабелю з роз’ємом типу Micro USB, але враховуючи той факт, що можливості тримати комп’ютер постійно увімкненим немає, постало питання щодо автономного джерела живлення. В якості рішення було використано бокс для 4 батарейок типу AAA, який не займає багато місця і дає змогу швидко та без зайвих зусиль замінити елемент живлення.

2.6 Висновки до розділу

В даному розділі було описано технології та рішення, які обрано для реалізації проєкту. Через специфіку тематики роботи тут присутні і Embedded технології, і хмарні рішення, і вже розроблений до цього сторонній продукт, як от месенджер Telegram, який на сьогоднішній момент набирає популярність та є другим месенджером за популярністю після Viber в країнах СНД, і одразу 3 різні мови програмування, які відрізняються не тільки підходами в написанні коду, але й значною різницею у наданому інструментарії.

За рахунок обраних технологій та рішень програма є масштабованою та може працювати не лише з одним екземпляром апаратної частини, як це показано в даному проєкті, а легко та без зайвих зусиль приймати дані від різних мікроконтролерів, незалежно від місцезнаходження, та вести історію наданих даних.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ

3.1 Структура системи

Через специфіку проєкту, а саме запланованої кількості окремих вузлів функціонування, маємо такі складові системи (див. рисунок 3.1): мікроконтролер з датчиками для вимірювання погодних показників, сервер, реалізований на платформі ASP.NET, розгорнутий в хмарній платформі Microsoft Azure, Azure Cache for Redis для зберігання тимчасових даних, що надходять з мікроконтролеру та швидкої реакції на запити користувача, Azure SQL Database для зберігання даних по кожній годині за останні 24 години та за добу та платформу-месенджер Telegram, на основі якої було створено бота для візуалізації результатів та взаємодії з користувачем. За рахунок крос-платформеності та наявності десктопної версії та веб-версії телеграму користувач може використовувати додаток за допомогою окремо встановленого Telegram на смартфоні, планшеті, комп'ютері та у веб-браузерах.

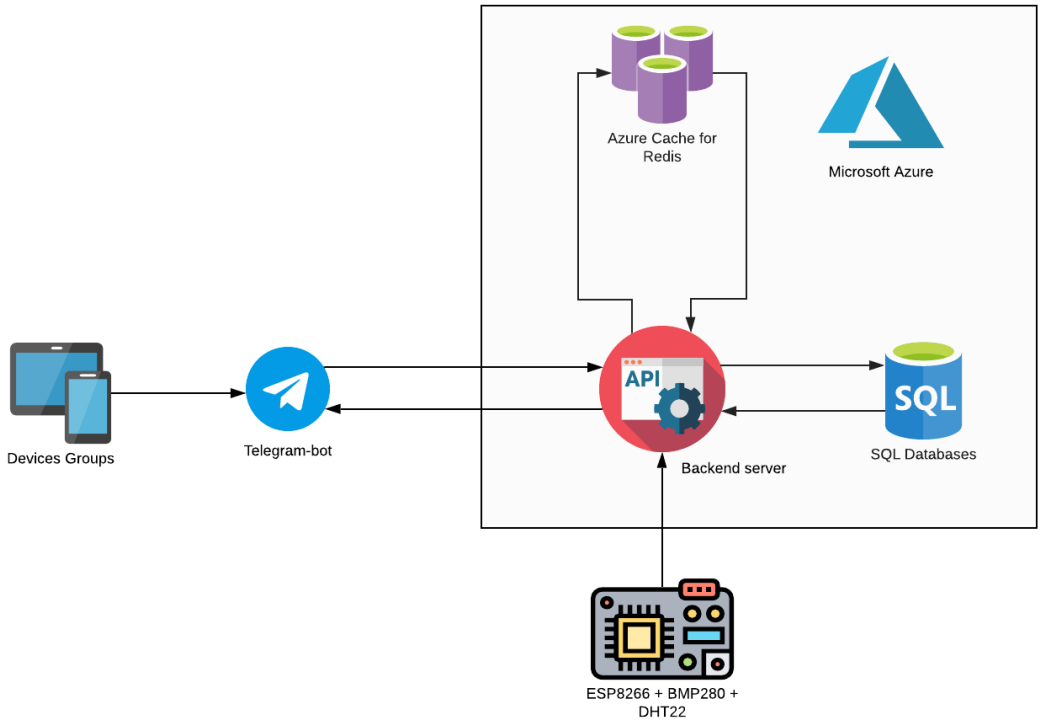

Рисунок 3.1 — Структура системи

На кресленику ІТ61.250БАК.004 Д2, що описує загальний алгоритм роботи застосунку від початку і до кінця, слід звернути увагу на те як багато присутньо умов, але незважаючи на це дані, що надходять з апаратної складової системи обов'язково будуть збережені в одне з двох сховищ даних, а коли користувач створить запит, обов'язково будуть проаналізовано та відображені відповідно до запиту чином. Що до умов, то вони теж відіграють важливу роль в роботі системи, оскільки у разі позитивного сценарію в той чи іншій умові дані будуть спочатку оброблені, а потім внесені до відповідної таблиці.

Також з даного кресленику випливає, що проблему з даними не буде проігноровано, і у разі втрати даних або іншого збою в роботі програми користувач отримає повідомлення про тимчасові порушення в роботі застосунку.

3.2 Реалізація апаратної частина

Розберемо структурну схему підключення датчиків вимірювання погодних показників до мікроконтролеру. Як можна помітити з рисунку 3.2, обраний мікроконтролер має велику кількість портів входу/виходу, але не всі вони є можуть використовуватись для реалізації даного проєкту. Нараз потрібні порти для подачі напруги, заземлення та GPIO для роботи з даними.

Як показано на рисунку 3.2, датчик DHT22 використовує лише 3 апаратних виходи для взаємодії з мікроконтролером, а саме VCC — для подачі напруги на датчик, Data — для передачі даних, які збираються датчиком в аналогову вигляді, конвертуються у цифровий і надсилаються на мікроконтролер, та GND — для заземлення. Останній четвертий апаратний вхід/вихід для потреб даного проєкту непотрібен. Підключення до мікроконтролеру виконується через порт GPIO та портів напруги 3.3В (для VCC) та заземлення (для GND). В даному випадку, Data підключено до порту GPIO12. Також між VCC та Data встановлено резистор з опором 4.7 кОм для стабілізації роботи датчику та уникнення зайвої напруги при роботі.

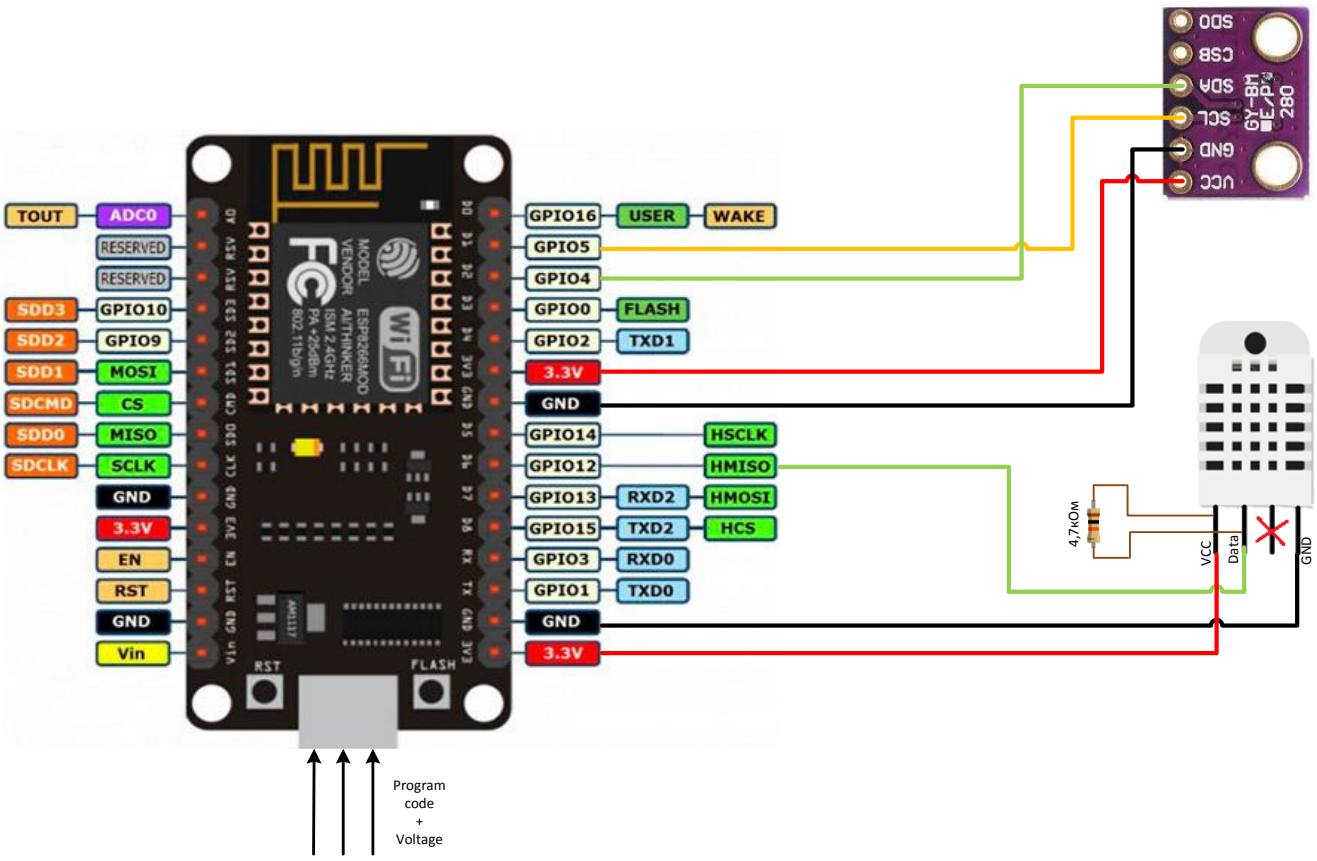

Рисунок 3.2 — Схема підключення датчиків до мікроконтролеру

Схожим чином підключено датчик BMP280. Але слід звернути увагу, що цей датчик має 6 апаратних входів/виходів. Це зумовлено двома послідовними інтерфейсами обміну даними, а саме SPI та I^2C. В цьому проєкті, обрано підхід I^2C, адже він покриває всі потреби в роботі з даним датчиком і паралельно не потребує особливо зайвих зусиль при роботі з ним. Через це було використано лише перші чотири (на рисунку 3.2 починаючи знизу) апаратних входи/входи, оскільки перші два обов'язкові до використання завжди, наступна двійка відноситься до I^2C, а третя двійка – SPI. Самі ж датчики для: VCC — для подачі напруги на датчик, GND — для заземлення, SCL — для синхронізації датчику та мікроконтролеру, SDA — для передачі даних, які також конвертуються з аналогового вигляду у цифровий і надсилаються на мікроконтролер. Цього ж разу, останні два виходи підключаються до мікроконтролеру через порти GPIO5 та GPIO4 відповідно. Це стандартні порти в моделі мікроконтролеру ESP8266 для роботи з I^2C інтерфейсом.

3.3 Програма-прошивка для мікроконтролеру

На сьогоднішній день, є багато засобів, за допомогою яких пишуться програми-прошивки та якими можна прошивати мікроконтролери. Насамперед все залежить від типу обраного мікроконтролеру. В даному випадку це мікроконтролер на базі плати NodeMCU V3. Цього разу було обрано середовище програмування Arduino Software IDE, яке є не лише зручним редактором коду, але й здатне заливати код на мікроконтролери, тим самим запрограмовуючи їх. Для написання самого ж коду використано мову програмування C++, як стандартний вибір в таких питання.

Для початку, дане середовище в своєму початковому варіанті не призначене для роботи з мікроконтролером ESP8266, тому спочатку слід встановити розширення для цієї моделі. Далі треба зібрати дані з датчиків, а для цього використано бібліотеки «DHT sensor library» та «Adafruit_BMP280», які є у вільному доступі. Дані бібліотеки надають методи, які позначені на рисунку 3.3, що повертають температуру (getTemperature), вологість повітря (getHumidity), атмосферний тиск (getPressure) та обчислюють тепловий індекс (computeHeatIndex).

```
float temperature = dht.readTemperature();
float humidity = dht.readHumidity();
float heatIndex = dht.computeHeatIndex(temperature, humidity, false);
float pressure = bmp.readPressure() / 133.33;
float temperatureBMP280 = bmp.readTemperature();
```

Рисунок 3.3 — Бібліотечні методи для отримання даних з датчиків

Після збору даних їх треба оформити у форматі JSON, оскільки даний формат є одним із загальноприйнятих стандартів передачі даних через інтернет. Він займає місця менше, ніж XML, та працювати з ним значно легше за рахунок його структури, так як вона нагадує звичайний об’єкт зі своїми полями. Створення JSON об’єкту та перетворення його в об’єкт класу SensorsData, який зображено на кресленику ІТ61.250БАК.004 Д3, мови C# відбувається набагато легше, ніж з XML файлом.

Спілкування між мікроконтролером та сервером відбувається за допомогою вбудованого до першого Wi-Fi модулю, який надсилає через інтернет дані з використанням протоколу HTTP. Для того щоб підключитись до Wi-Fi роутеру та отримати доступ в інтернет, насамперед необхідні логін та пароль для доступу, а також було використано бібліотеку «ESP8266WiFi», яка надає широкий спектр можливостей, що включає в себе й налаштування зв'язку по Wi-Fi з режимами роботи та шифруванням, а щоб сформувати HTTP запит із даними та зчитуванням статус коду й відправити його за відповідною адресою (http://weatheranalyzertelegrambotserver.azurewebsites.net/api/sensorData) взято бібліотеку «ESP8266HTTPClient». В рамках дипломного проєкту було вирішено надсилати нові дані кожні 5 хвилин. За необхідності достатньо змінити один рядок коду для встановлення іншого проміжку часу.

3.4 Структура проєкту

Серверну частину даного проєкту було виконано у вигляді окремого рішення в середовищі розробки Visual Studio 19. Оскільки платформою для цього було обрано ASP.NET, а завдяки особливостям роботи даної системи, а саме перехопленням даних від зовнішнього джерела в інтернеті, наявність великої кількості класів-моделей, які необхідні для взаємодії з базою даних та своєрідний метод запуску застосунку, шаблоном, за яким було створено проєкт є Model-View-Controller. За замовчуванням середовище створює проєкт даного шаблону з наступним списком директорій:

— App_Data (зазвичай тут зберігають всю необхідну інформацію або дані, з якими в подальшому працює розробник на стороні серверу, наприклад, локальні бази даних);

— App_Start (містить файли, які відповідають за налаштування запуску програми);

— Content (зберігає файли, що необхідні для роботи розробника на клієнтській стороні програми, а саме зображення, CSS файли тощо);

— Controllers (містить класи-контролери, які необхідні для прийняття та передачі даних);

— fonts (тут розміщені стилі для оформлення тексту на клієнтській частині застосунку);

— Models (зберігає всі класи-моделі для опису бізнес-логіки застосунку, а в даному випадку ще й для конвертації даних з JSON об'єкту в екземпляр класу мови C# та зберігання ключової інформації для пов'язання Telegram-боту, серверної частини та хмарного сховища);

— Scripts (містить JavaScript бібліотеки для роботи з клієнтською частиною);

— Views (безпосередньо веб-сторінки для відображення у веб-браузері).

З опису наведених директорій та описаної раніше структури системи випливає що деякі з них є зайвими та вимагають видалення, аби не збільшувати розмір проєкту та не забивати зайвим місцем структуру проєкту. Слід більше сказати про папку App_Data, оскільки в розділі 2.3.3 було сказано, що хай можливість розгортання системи локально й існує, в цьому випадку локальна БД буде зберігатись в цій папці, але з це не виконуватиме поставлені вимоги до системи.

Після відповідних маніпуляцій з видаленням зайвих та створенням нових директорій маємо наступну структуру:

— App_Start;

— Commands (містить класи-команди боту, в яких реалізовано логіку формування повідомлення-відповіді);

— Controllers;

— Migrations (після створення таблиць в базі даних за допомогою підходу Code First дана папка створюється сама і містить так звані міграції, які зберігають усю необхідні інформацію щодо структури БД);

— Models;

— Services (в цій папці зберігають класи-помічники для взаємодії з Entity Framework, як от клас WeatherContext (див. додаток А3), або для синхронізації Telegram-бота із серверною частиною, як от клас Bot).

3.5 Створення Telegram-боту

Оскільки бот — це окремий об’єкт в ресурсі Telegram’у, який має свою унікальну назву, унікальний ключ для взаємодії з ним, а команди, які буде викликати користувач, в даному проєкті реалізовано на стороні серверу за допомогою мови C#, то існує 2 етапи створення боту: створення самого боту та опис його поведінки (команди та відображення результатів) на сервері.

3.5.1 Створення боту на стороні Telegram

Для першого етапу створення боту слід скористатись спеціально створеним командою розробки Telegram ботом, який називається BotFather і застосовується виключно для створення нових та редагування існуючих ботів. Створення абсолютно всіх ботів, що створені не офіційною командою розробки, починалось саме з BotFather. На рисунку 3.4 наведено короткий опис цього боту в інформаційному меню.

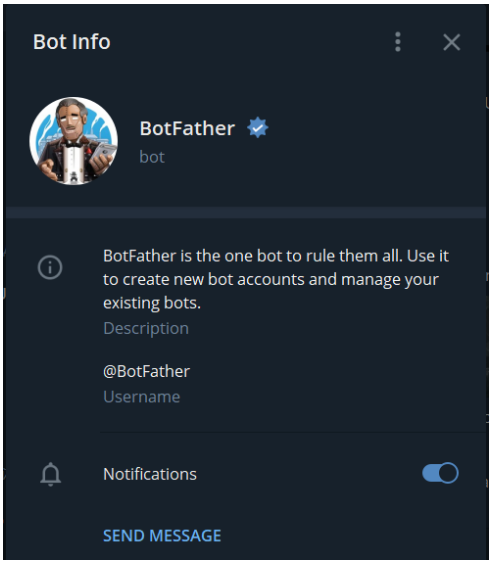

Рисунок 3.4 — Інформація про BotFather бот

Після старту роботи бот відображає список всіх своїх команд (див. рисунок 3.5), оперуючи якими можна створити нового бота або редагувати існуючого,

вмикнути функції, які змінять відображення команд вами створеного бота та створити гру в середовищі розробки, наданому командою Telegram. У випадку якщо API-ключ одного зі створених ботів було забуто, тут є можливість його відображення окремою командою.

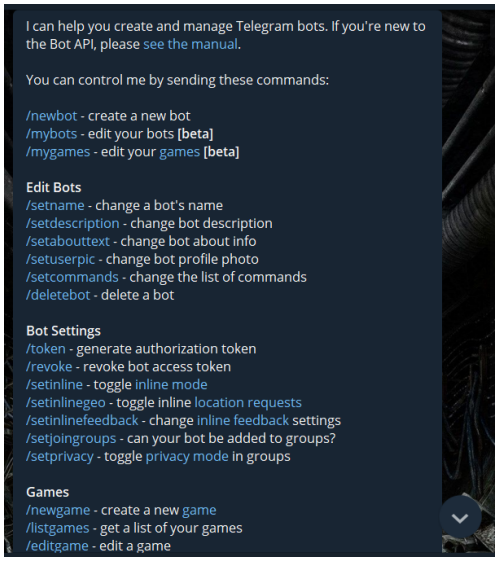

Рисунок 3.5 — Список всіх існуючих команд BotFather боту

Обравши назву свого бота та яке він буде мати кодове ім’я BotFather генерує API ключ (див. рисунок 3.6) для взаємодії вашої програми з описом команд із самим ботом. Слід обов’язково пам’ятати, що дана інформація є дуже важливою і повинна залишатись прихованою для всіх, окрім самого розробника.

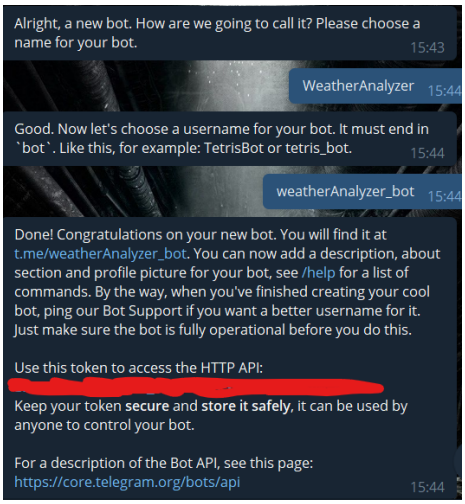

Рисунок 3.6 — Назва боту та згенерований API ключ

Після створення боту є можливість відредагувати його: змінити його опис (текст, який користувач буде бачити, вперше відкривши канал з ботом), додати текст в інформаційне меню про бота, як показано на рисунку 3.4, додати фото, яке буде виокремлювати його серед списку інших ботів, змінити відображення команд та результатів тощо. Меню команд для цього наведено на рисунку 3.7.

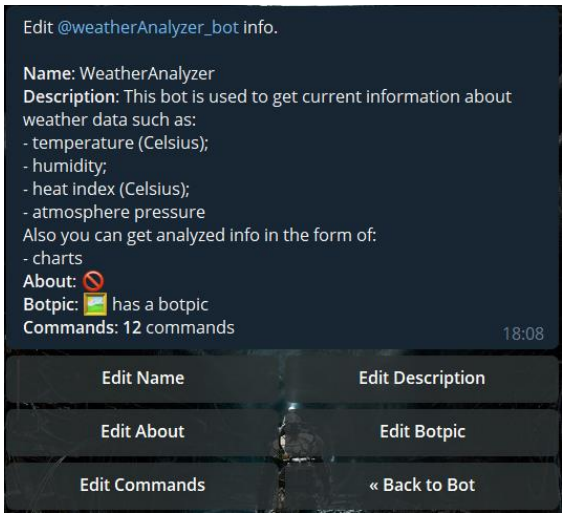

Рисунок 3.7 — Зручний інтерфейс для керування ботом

Для того, щоб користувач зміг обрати команду зі списку треба цей список створити. Достатньо обрати відповідну команду боту BotFather і, як показано на рисунку 3.8, а саме правило «назва команди – її опис», задати список команд.

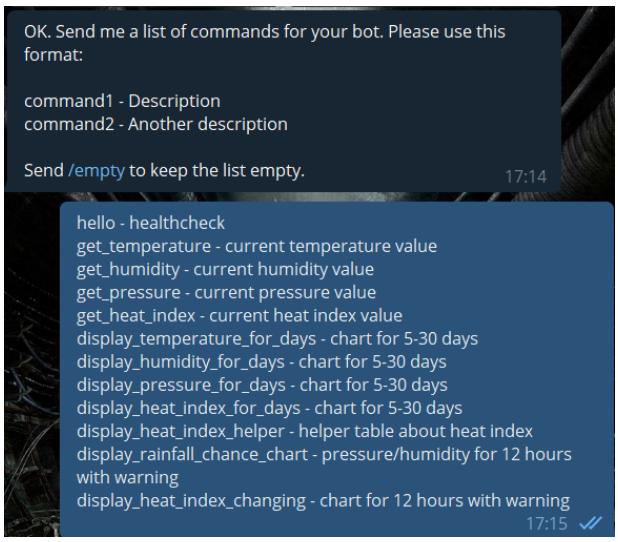

Рисунок 3.8 — Створення списку команд боту

3.5.2 Опис команд на стороні серверу

Після аналізу існуючих рішень та наявних вимог виокремилось дванадцять команд, з якими може взаємодіяти користувач. Чотири з них відповідають за відображення поточних погодних показників у текстовому вигляді, а інші 8 – за відображення графіків динаміки зміни цих самих погодних показників за останні 12 годин або кількість днів, яку зазначить користувач.

Для майбутньої взаємодії та синхронізації API із серверним кодом було створено клас ApiSettings (див. кресленик ІТ61.250БАК.004 Д3), який зберігає в собі назву даного боту, посилання на домен, який Microsoft Azure дає застосунку після його розгортання в хмарній платформі та сам API ключ.

Кожна команда має свій окремий клас в структурі проєкту, як зазначено в кресленику ІТ61.250БАК.004 Д4, де описано поле з назвою даної команди та метод з логікою виконання запиту Execute. У разі помилки або відсутності даних за тих чи інших причин команда поверне повідомлення про помилку. В залежності від обраної команди відрізняється й текст помилки. В додатку А1 можна побачити приклад коду класу однієї з команд.

3.6 Інтеграція з Azure Cache for Redis

Дані, які приходять на сервер кожні 5 хвилин записуються в Azure Cache for Redis у вигляді ключ-значення. Використання даного сервісу, враховуючи той факт, що реляційна база даних точно буде використовуватись, зумовлено тим, що оскільки Redis є нереляційною базою даних, то швидкість витягування даних з неї є більшою, ніж у звичної SQL бази даних. Це позитивно впливає на взаємодію з користувачем, адже час отримання відповіді все більше залежить лише від якості та швидкості підключеного інтернету.

Для того щоб не вносити нові дані в базу кожні 5 хвилин, адже оновлення таблиць в базі відбувається кожну годину або кожну добу через специфіку та побудовану архітектуру додатку, Redis у вигляді операційної пам’яті ідеально під-

ходить в даному випадку. Реалізацію алгоритму запису даних, що надходять з мікроконтролера, у сховища, одне з яких є Azure Cache for Redis відображено на рисунку 3.6.1.

Для виклику сховища Redis'у було використано підхід Lazy loading, який передбачає, що виклик сервісу та навантаження загального ресурсу відбувається лише в ті випадки, коли вони дійсно необхідні. Протягом всього іншого часу воно не навантажує систему. Реалізується це за допомогою приватного поля lazyConnection, яке можна побачити на прикладі коду в додатку А2.

Оскільки Azure Cache for Redis є стороннім сервісом, то інтеграція з ним потребує описаних правил підключення в самому проєкті програми. Для цього у файлі Web.config створюється окремий XML блок для рядку з'єднання з посиланням на сервіс та паролем, приклад якого наведено у додатку А4.

3.7 Інтеграція з Azure SQL Server

Дана база даних використовується для зберігання температури, вологості повітря, атмосферного тиску та теплового індексу окремо за кожну минулу добу та окремо за кожну годину останньої доби. Виходячи з наведеного у кресленику ІТ61.250БАК.004 Д1, бачимо що таблиць дев'ять: по дві таблиці на кожен погодний показник і одна таблиця для зберігання даних про різні мікроконтролери у випадку масштабованості додатку. Відповідно оновлення даних в таблиці для погодинних показників відбувається кожну годину таким чином, що останні записані дані в Azure Cache for Redis витягуються і заливаються в базу навпроти відповідної години, а в таблиці для подобових – кожного дня о шостій ранку для того, щоб охопити періоди останніх дня та ночі, взявши середнє значення за ці проміжки та додавши новий рядок в таблицю. Даний алгоритм зберігання даних проілюстровано на рисунку 3.9. Таким чином, при побудові графіків, користувач може задати період, за який відобразити динаміку зміни, в днях від п'яти до тридцяти днів, а система вже буде динамічно генерувати графіки. Щодо графіків по годи-

нам, то вони відображають динаміку лише за останні дванадцять годин, так це є оптимальним варіантом для прийняття рішення про поточну погоду.

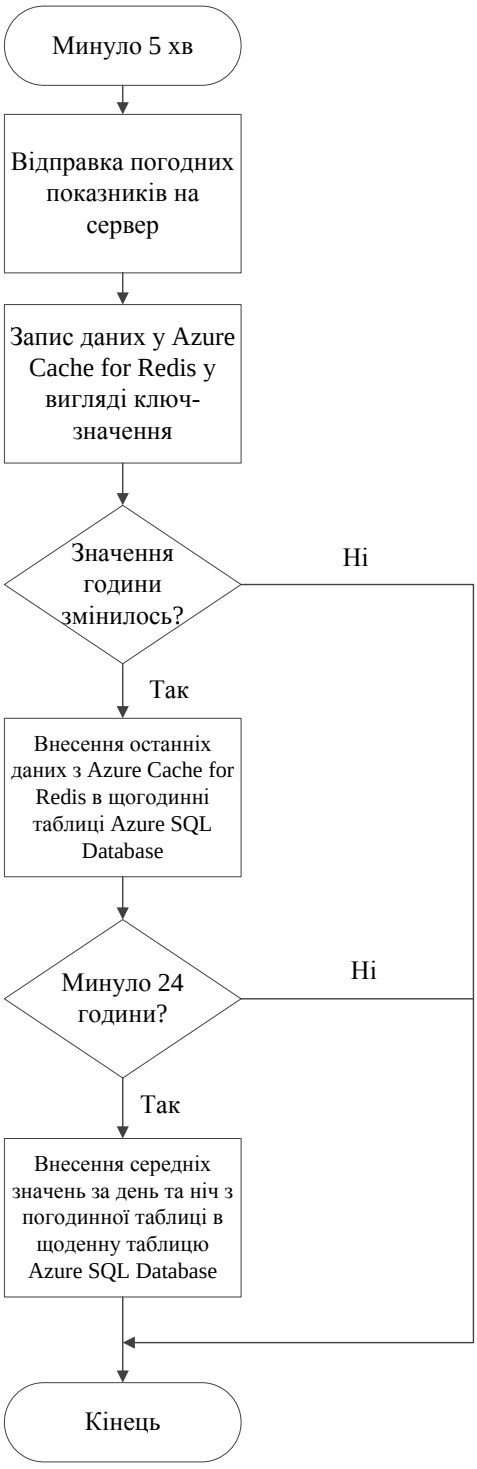

Рисунок 3.9 — Алгоритм отримання та запису даних в сховища

Для легшої взаємодії системи з базою даних було використано технологію Entity Framework, однією з особливостей якої є реалізація підходу Code First: коли

спочатку описуються класи-сутності, що описані в кресленику ІТ61.250БАК.004 Д3 (всі, що наслідуються від класу Base), а вже потім на їх основі Entity Framework створює таблиці з відповідними полями. Для роботи з цією ORM необхідно створити окремий клас-контекст, в якому будуть зберігатись посилання між сутностями і таблицями (див. додаток А3).

Як і з Redis, інтеграція програми з Azure SQL Server потребує відповідних правил підключення, які описані окремим XML блоком, приклад якого наведено в додатку А5, у файлі Web.config.

3.8 Побудова динамічних графіків

Користувач може запросити не тільки відобразити поточні погодні показники, але й графіки за вказаний ним проміжок часу. Для побудови цих графіків використовуються засоби мови програмування JavaScript, оскільки дана мова має динамічну типізацію, є зручною та швидкою в питаннях побудови графіків та має вже готові потужні рішення у вигляді бібліотек коду для цього. Однією з таких бібліотек є «Chart.js», яка надає легкий в розумінні та використанні інструментарій з можливістю побудови графіків різних типів, кольорів, форматів та оформлень. В рамках даного проєкту було обрано лінійний тип графіків, приклад якого наведено на рисунку 3.10.

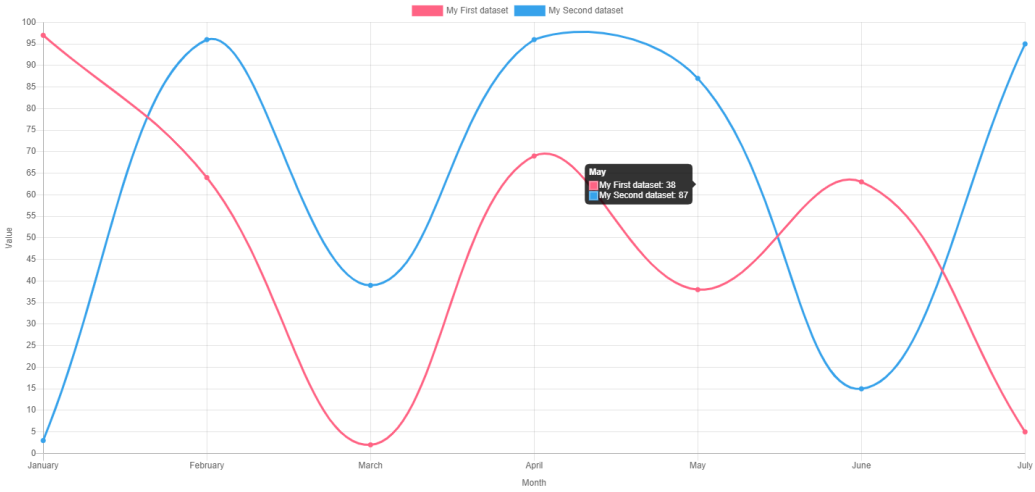

Рисунок 3.10 — Приклад лінійного графіку

3.9 Висновки до розділу

В цьому розділі було описано реалізацію проєкту, що, як і з чим взаємодіє, які бібліотеки та методи використовувались. Даний проєкт вміщує в собі і Embedded технології, тобто мікроконтролер та датчики вимірювання погодних показників, які програмуються та взаємодіють із сервером за допомогою написаного коду мовою C++, і ресурси хмарної платформи Microsoft Azure, такі як Azure Cache for Redis, Azure SQL Server та просто можливість розгортання на віддаленому сервері, і засоби мов програмування C#, JavaScript та фреймворку ASP.NET, і нової та швидко набираючої популярності платформи-месенджеру Telegram.

За рахунок прийнятого рішення про використання Telegram-боту для взаємодії з користувачем не потрібно відкривати важкий та вимогливий веб-браузер та переходити за посиланням на сайт додатку, достатньо просто відкрити месенджер і наче спілкуючись зі звичайною людиною отримувати дані та графіки на відправлені запити.

4 РОЗГОРТАННЯ СИСТЕМИ

4.1 Принцип розгортання

Як було сказано раніше, застосунок і всі його програмні частини розгортаються в хмарній платформі Microsoft Azure. Дана система та надані нею можливості спрощують розгортання, дають здатність тримати систему постійно працюючою не на власному комп'ютері, який в даному випадку виступає в якості серверу, а на віддаленому сервері з доступом до нього за веб-посиланням та забезпечують подальше масштабування системи в майбутньому.

В термінах Microsoft Azure, будь-що доступне в платформі для користувача називається ресурсом. Ресурси об'єднуються в логічні контейнери, їх називають ресурсними групами. Групи ресурсів можуть існувати лише в рамках деякої підписки. В даному випадку, підпискою слугує акаунт користувача хмарної платформи.

Для того щоб створити новий ресурс і додати його в одну зі своїх ресурсних груп слід скористатись менеджером ресурсів, який надає декілька способів створення: ARM Template[22], спеціально розроблене для цього REST API, спеціалізований SDK або за допомогою графічного інтерфейсу на веб-сторінці платформи. Для повноцінного розгортання даного проєкту в хмарній платформі Microsoft Azure потрібно створити відповідні ресурси та провести ряд необхідних речей, а саме:

— створити ресурс для реляційної бази даних та під'єднати його до програми;

— створити ресурс для сервісу Redis, який грає роль оперативної пам'яті додатку та під'єднати його до програми;

— розгорнути backend частину додатку для постійної роботи та взаємодії з Telegram-ботом.

4.2 Вимоги до програмного та апаратного забезпечення

Для подальшої роботи з підтримки та розгортанню системи розробнику для комфортної та безперешкодної роботи необхідно мати:

— 8 гігабайт оперативної пам’яті;

— мінімум 2-х ядерний процесор з частотою 1.6 ГГц;

— HDD накопичувача буде достатньо;

— бажано 10 гігабайтів вільного місця та 20 гігабайт для встановлення всього необхідного програмного забезпечення (бібліотеки фреймворку розробки .NET Framework, NuGet пакети для додаткових сервісів);

— месенджер Telegram, встановлений на смартфон, комп’ютер у вигляді десктопної версії або як розширення у веб-браузері;

— щоб мати все необхідне під рукою та не потребувати сторонніх сервісів бажано використовувати середовище програмування Visual Studio версії не раніше 17;

— акаунт в платформі Microsoft Azure.

4.3 Реляційна база даних

Оскільки в даному проєкті використовуються можливості міграцій фреймворку Entity Framework, то для створення бази даних на стороні хмарної платформи з усіма необхідними таблицями та даними необхідно створити саму базу даних, тобто додати сервіс в свою ресурсну групу, і під’єднати її до програми за допомогою рядку підключення у файл Web.config. Уся структура бази даних буде створена автоматично при першому запуску веб-додатку.

Для створення одиничної бази даних в хмарній платформі найзручніше використовувати екземпляр SQL Database. Для цього необхідно виконати наступні кроки:

— вибрати створення нового ресурсу SQL Database;

— вибрати існуючу чи створити нову ресурсну групу;

— вибрати існуючий чи створити новий SQL Server;

— вибрати ім’я та початкову кількість ресурсів;

— налаштувати мережевий доступ.

4.4 Створення ресурсу Azure Cache for Redis

Створити хмарний аналог сервісу Redis в платформі Azure має приблизно той же рівень складності, ніж базу даних, оскільки необхідно кількість кроків залишається тою ж, а під’єднання до програми реалізовується за тим же принципом, відрізняється лише сам код рядку підключення. В загальному випадку порядок такий:

— вибрати створення нового ресурсу Azure Cache for Redis;

— вибрати існуючу чи створити нову ресурсну групу;

— вибрати ім’я та локацію центру даних, де буде розгорнуто сховище;

— вибрати правильну георепліkацію та налаштувати мережевий доступ.

4.5 Розгортання backend частини додатку

Якщо ж говорити власне за розміщення самого додатку на віддаленому сервері, то для цього необхідно в половину менше кроків, ніж для попередніх пунктів. А саме:

— вибрати існуючу чи створити нову ресурсну групу;

— вибрати ім’я та локацію центру даних, де буде розгорнуто сховище;

— вибрати унікальне ім’я, під яким буде розміщено застосунок в платформі.

Після того, як всі ці кроки буде виконано достатньо пари натискань на кнопки, щоб застосунок опинився на віддаленому сервері. Зручніше робити це за допомогою графічного інтерфейсу середовища програмування Visual Studio.

4.6 Висновки до розділу

В даному розділі наведено опис того як і яким чином розгортається програма на віддаленому сервері в хмарній платформі Microsoft Azure, а також які сервіси цієї ж платформи використовуються і як їх створити. Основну увагу приділено саме створенню сервісів Azure Cache for Redis та Azure SQL Database, оскільки вони потребують виконання більшої кількості кроків, ніж просте розгортання backend частини на віддаленому сервісі. Також описано за допомогою зображень графічного інтерфейсу середовища розробки Visual Studio як легко та швидко можна виконати розгортання.

Загалом існує 4 методи для створення нового ресурсу на хмарній платформі. Це ARM Template[22], спеціально розроблене для цього REST API, спеціалізований SDK або за допомогою графічного інтерфейсу на веб-сторінці платформи.

Також в цьому розділі міститься опис необхідного програмного та апаратного забезпечення для підтримки існуючого або розробки нового функціоналу даного проєкту.

5 ГРАФІЧНИЙ ІНТЕРФЕЙС КОРИСТУВАЧА

Як було багато разів сказано раніше, інтерфейсом, з яким буде взаємодіяти користувач та отримувати результати, було обрано месенджер Telegram з функціональністю ботів. Виглядає це наче спілкування з іншим користувачем месенджеру, різниця лише в тому, що користувач вводить назву необхідної команди, а бот, в даному випадку, повертає результат або у вигляді тексту, або у вигляді графіків.

5.1 Опис інтерфейсу

На рисунку 5.1 представлено вікно опису боту для користувача, який вперше відкрив чат з ним. В даному випадку тут описано для чого цей бот і які можливості він надає.

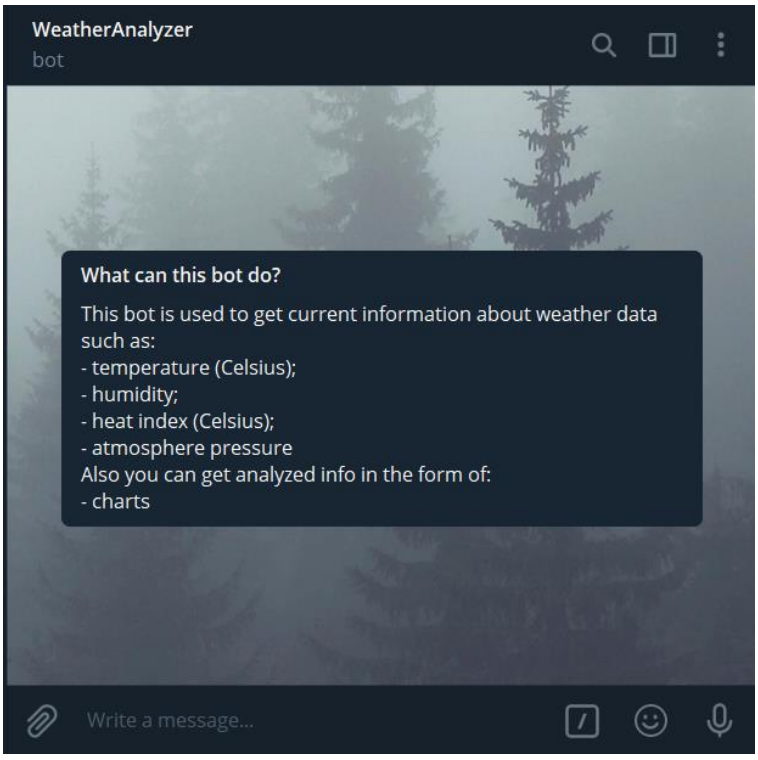

Рисунок 5.1 — Початкове вікно опису боту

Наступним кроком користувача буде обрати відповідну команду. На рисунку 5.2 представлені з першої по четверту команди повертають результат у вигляді

тексту, як би це писав співбесідник, про значення погодного показнику в даний момент.

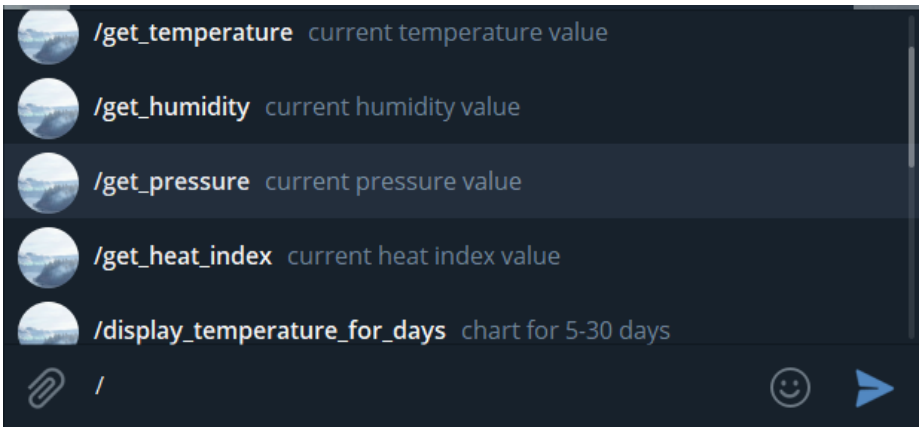

Рисунок 5.2 — Список команд для результату текстом

Наступними в списку йдуть команди (див. рисунок 5.3), які відображають графіки за обраний користувачем проміжок останніх днів, таблицю-помічник про тепловий індекс та графіки за останні 12 годин з оповіщенням про опади або над-високий тепловий індекс.

Рисунок 5.3 — Список команд для результату графіком

На рисунку 5.4 представлено приклад результату роботи перших чотирьох команд, що повертають дані поточних погодних показників у текстовому вигляді: Температура — в Цельсіях, вологість повітря — у відсотках, атмосферний тиск —

в міліметрах ртутного стовпчика, тепловий індекс — в Цельсіях. В даному випадку приклад наведено для температури.

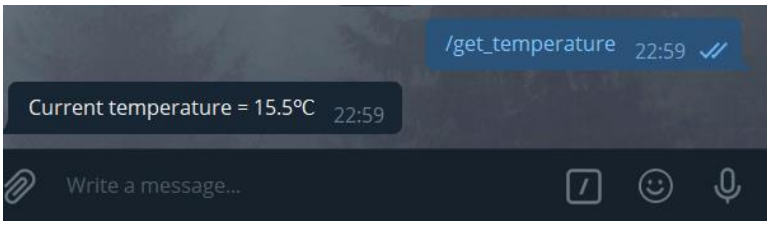

Рисунок 5.4 — Відображення повідомлення про поточну температуру

5.2 Опис графіків

Як було виокремлено в пункті 1.2 про аналіз вимог, дана система повинна ще й повертати графіки погодних показників у вигляді їх середніх значень за денний час та нічний протягом певного проміжку минулих днів та два графіки, що основною своєю метою ставлять аналіз ймовірності опадів або критичне підвищення теплового індексу, які негативно або ж навпаки позитивно впливають не тільки на людину, а й на рослинні культури, протягом останніх 12 годин, оскільки приблизно такий проміжок є оптимальним та раціональним для своєчасного прийняття подальшого рішення.

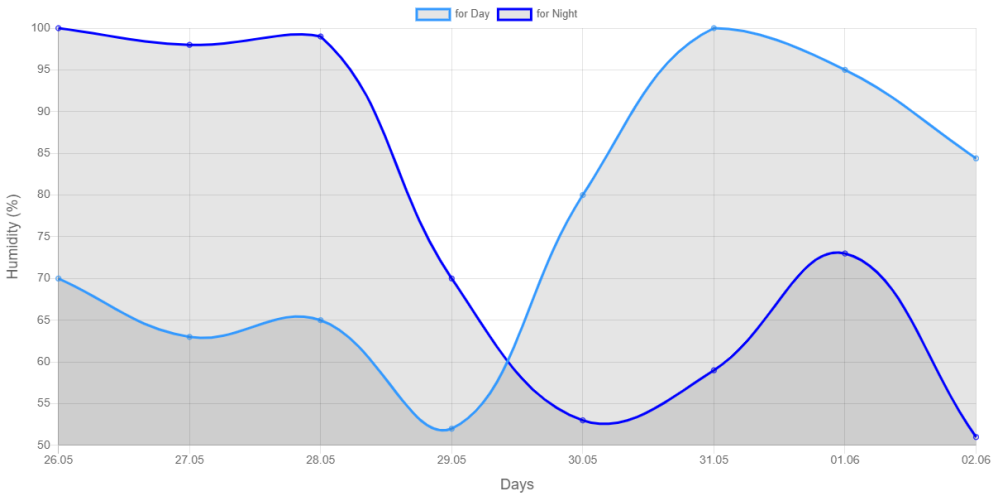

Рисунок 5.5 — Приклад графіку динаміки зміни середньої вологості повітря в день та ніч за останні 8 днів

На рисунку 5.5 представлено графік динаміки зміни вологості повітря, де зверху відображається легенда для розрізнення ліній для денного та нічного середніх значень. Ліворуч та знизу наведено назви осей, які в даному випадку відповідають за шкалу вологості, яка вимірюється у відсотках, та останніх восьми днів. Який проміжок останніх днів буде відображено обирає сам користувач, написавши боту додаткове повідомлення на відповідне питання. Обрати користувач може проміжок між 5 і 30 днями.

На рисунку 5.6 відображається графік динаміки зміни цього разу вже температури за проміжок останніх днів. Аналогічно до рисунку 5.5 маємо легенду про денний та нічний показники, а також підписи осей.

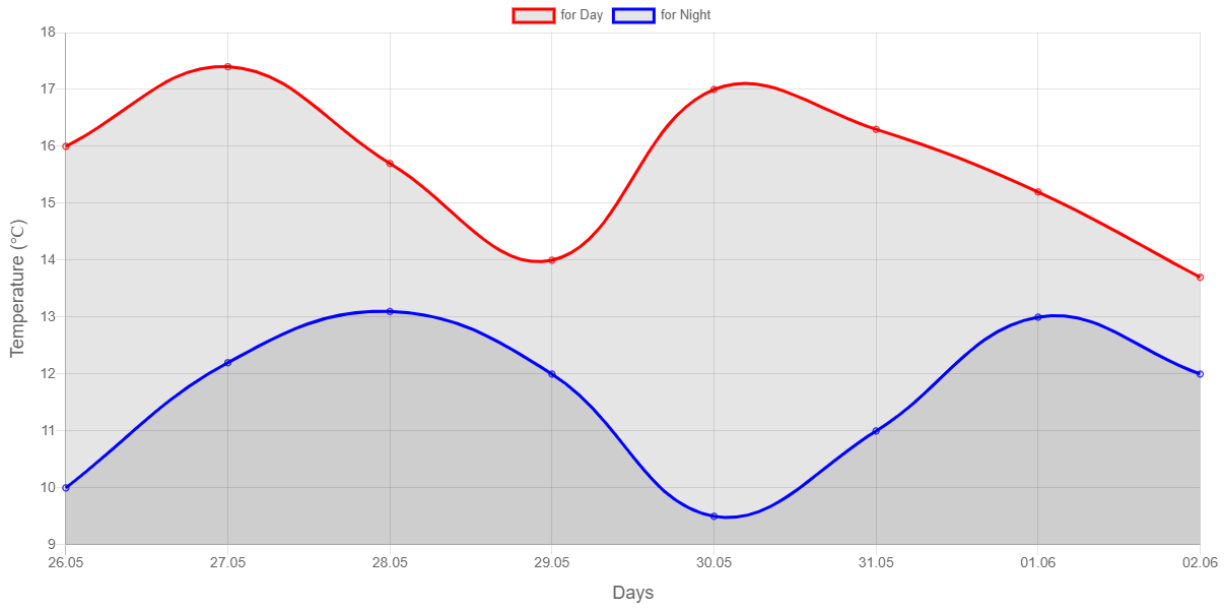

Рисунок 5.6 — Приклад графіку динаміки зміни середньої температури в день та ніч за останні 8 днів

Динаміка зміни атмосферного тиску та теплового індексу відображені на рисунках 5.7 та 5.8. Як можна помітити криві за денний та нічний показники відрізняються в кольорі від графіка до графіка. Це було зроблено для того, щоб користувач, який вже певний час користується даним додатком, міг з першого погляду визначити про який погодний показник зроблено саме цей графік.

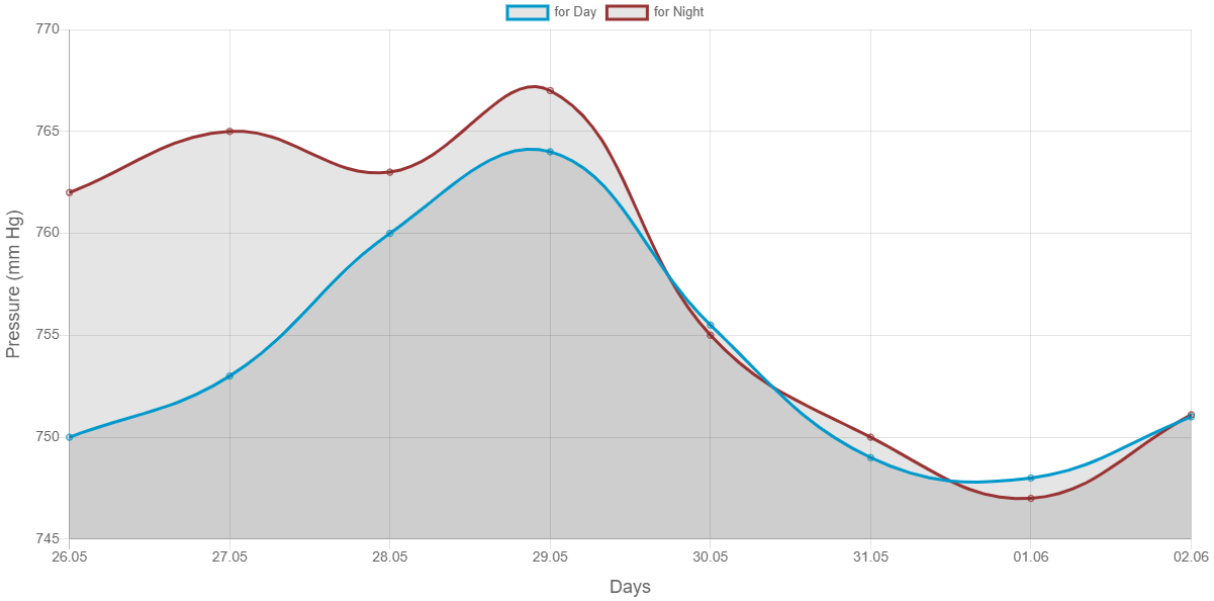

Рисунок 5.7 — Приклад графіку динаміки зміни атмосферного тиску за останні 8 днів

Також кожен з них має сітку перетину значень на осях для більш зручного орієнтування та точки на кривих, які слугують перетином конкретного значення з відповідним днем. Як можна помітити, кожна крива має свою сіру зону для більш зручного зчитування руху кривих і при перетині цих кривих зони не заважають подальшій роботі, як це продемонстровано на рисунку 5.8.

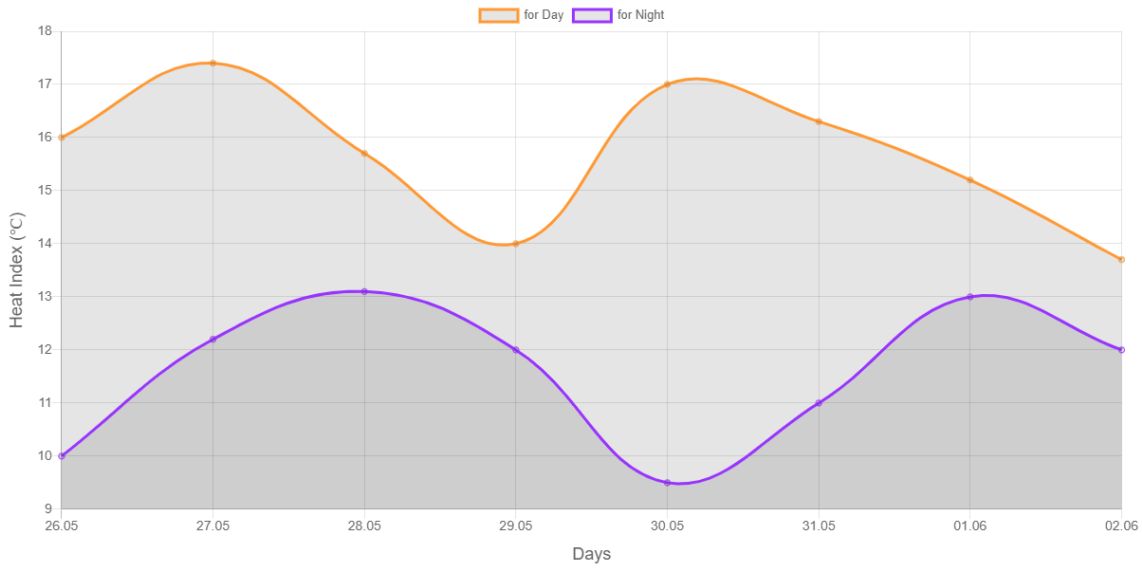

Рисунок 5.8 — Приклад графіку динаміки зміни теплового індексу за останні 8 днів

Наступним прикладом графіку даного додатку, що зображено на рисунку 5.9, є таблиця-помічник, яка необхідна для швидкого визначення поточного стану теплового індексу, адже не кожна людина знає або ж просто в змозі запам’ятати яким чином змінюється даний показник і який його рівень є критичним. В даному випадку присутні чотири кольори для позначення рівня критичності теплового індексу. Перший відповідає за помітне відчуття в’ялості та послаблення організму, другий є сигналом прийняття відповідних заходів для убезпечення себе або культур, які бояться надмірного тепла від подальших наслідків, третій – це рівень, коли в будь-який момент може трапитись тепловий удар, а четвертий є критичним і в такі моменти рекомендується не покидати приміщення або ж організувати потік холодного повітря для культур, що бояться надмірного тепла.

Ліворуч наведено шкалу зміни температури, а зверху – вологості повітря і можна помітити, що при збільшенні температури значення теплового індексу змінюється різкіше, ніж при зміні вологості. Якщо значення відносної вологості достатньо мале, а, отже, повітря може "прийняти" досить багато колишніх вологих шматочків людської шкіри, температура шкіри охолоджується за рахунок випаровування. Якщо ж вологість підвищена і, до того ж, повітря попередньо нагріли, до тіла прилипають гарячі мікрокрапельки води або іншої рідиннопарофазной субстанції. Природньо, при цьому шкірний покрив додатково нагрівається.

Temperature, °C	Relative humidity, %																				
	0	5	10	15	20	25	30	35	40	45	50	55	60	65	70	75	80	85	90	95	100
21	18	18	18	18	19	19	19	19	20	20	21	21	21	21	21	21	22	22	22	22	22
24	21	21	21	22	22	22	23	23	23	23	24	24	24	24	25	25	26	26	26	26	27
27	23	23	24	24	25	25	26	26	26	27	27	27	28	28	29	30	30	31	31	32	33
30	26	26	27	27	28	28	29	29	30	31	31	32	32	33	34	35	36	37	39	41	42
32	28	29	29	30	31	31	32	33	34	35	36	37	38	39	41	43	45	47	50		
35	31	31	32	33	34	35	36	37	38	40	42	43	46	48	51	54	58				
38	33	34	35	36	37	38	40	42	44	46	49	52	56	59	62						
41	35	36	38	39	41	43	45	48	51	54	57	61	65								
43	37	39	41	42	44	47	51	54	58	62	66										
46	39	42	44	46	49	53	57	62	66												
49	42	44	47	51	54	59	64														
Weariness				Very bad						Heatstroke						Jump to sea!					

Рисунок 5.9 — Таблиця-помічник відносно значення теплового індексу

Останнім графічним продуктом слугують два графіки, які вміщують в собі більше, ніж один погодний параметр для більш наглядного процесу аналізу та

статичний проміжок в 12 останніх годин. Це рисунки 5.10 та 5.11. Майже все, що було на перших чотирьох рисунках присутнє й тут, але криві наразі відповідають за окремий погодний показник, а до осі Y приєдналась ще одна така ж, оскільки, наприклад, атмосферний тиск та вологість повітря мають різні ваги значень.

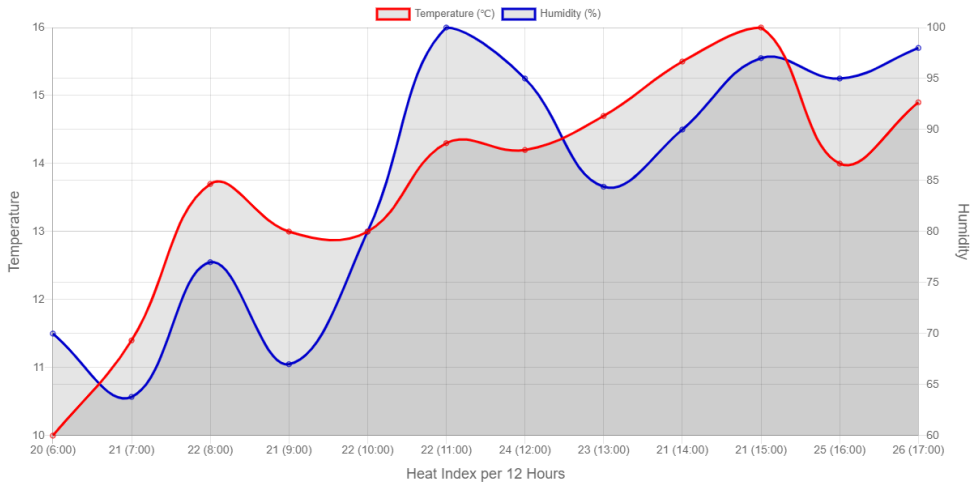

Рисунок 5.10 — Приклад графіку динаміки зміни теплового індексу за останні 12 годин

Разом з цими двома графіками користувачу буде відображено повідомлення з попередженням про вірогідність опадів або рівнем теплового індексу для швидшого реагування на поточну ситуацію.

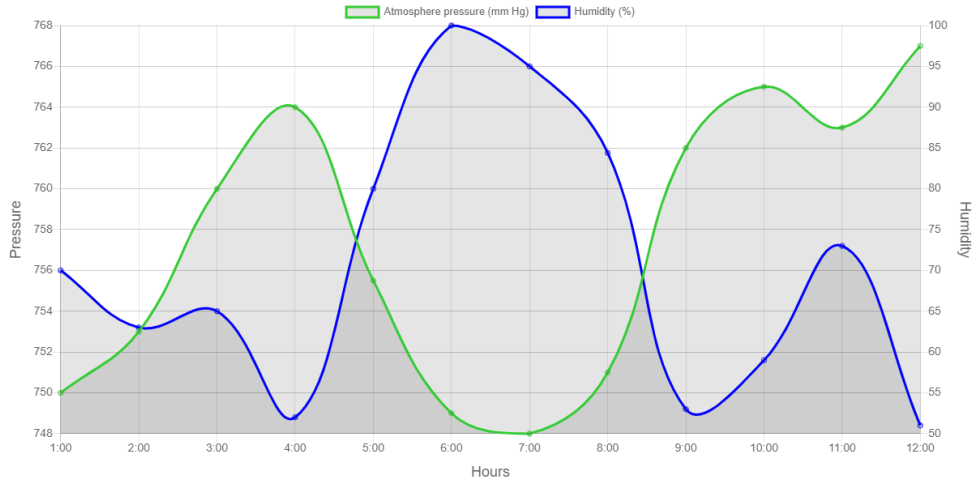

Рисунок 5.11 — Приклад графіку динаміки зміни вологості повітря та атмосферного тиску за останні 12 годин для передбачення опадів

5.3 Висновки до розділу

Даний розділ було присвячено графічному інтерфейсу користувача в середовищі месенджера Telegram, із зображеннями чату з ботом, його опису роботи і для чого він, коли користувач вперше відкриває чат з ним, зображень його команд, як для результату в текстовому вигляді, так і для повідомлень з графіками і звісно приклади самих графіків з описом їх легенди, використання, призначення та функціональності.

Графіки наведені в цьому розділі є динамічними, оскільки зовнішній вигляд результату залежить від даних, що збираються, агрегуються та надходять з бази даних, а також для деяких у користувача є можливість обрати проміжок останніх днів від 5 до 30, за який згенеруються та відобразяться погодні дані. Або ж існують інші два графіки, які хай і генеруються на динамічних даних, але завжди відображають їх за останні 12 годин, оскільки це є оптимальним та раціональним проміжком для своєчасного прийняття подальшого рішення. Разом з ними користувач отримує повідомлення про шанс випадіння опадів або критичність рівня теплового індексу.

6 ТЕСТУВАННЯ ДОДАТКУ

Типів та підходів тестування є велика кількість. Вони відрізняються глибиною тестування, типом модуля, який тестують, ступенем залученості різних частини програми, знанням коду або ж навпаки незнанням, тими, хто проводить тестування тощо. В даному розділі буде виокремлено 4 види тестування: unit-тестування або модульне, інтеграційне тестування, End-2-End тестування або тестування на основі сценарії користувача від початку і до кінця та тестування навантаження.

Важливо ще розуміти що саме треба тестувати, аби уникнути зайвого тестування. Наприклад, API для роботи з Telegram-ботом тестувати не потрібно, оскільки команда розробки месенджеру вже це зробила до нас. Або роботу сервісів хмарної платформи Microsoft Azure, як от Azure Cache for Redis та Azure SQL Database теж не варто тестувати, бо відповідне тестування було проведене задовго до нас командою розробки.

6.1 Unit-тестування

Unit-тестування або як його ще називаються модульне тестування має на меті перевірити правильність роботи та отриманого результату окремих модулів програми, роль яких грають публічні методи. Тести для них описані в таблицях 6.1.1 та 6.1.2.

Даний підхід є дуже важливим, оскільки якщо на етапі unit-тестування є проблеми та непройдені тести, то йти на рівень вище немає ніякого сенсу, оскільки взаємодія з цими модулями вже некоректна.

В даному проєкті, такі методи використовуються в класах-командах боту, контролерах, які відповідають за прийняття та запис даних, а також окремий клас Bot, який має метод, що повертає всі існуючі команди і синхронізує їх з тим, що виводиться користувачу в списку команд.

Таблиця 6.1.1 — Тестування окремих модулів серверної частини програми

Назва методу	Очікуваний результат	Отриманий результат	Статус тесту
WriteRedisData	Дані, що було отримано в рамках цього методу, записано в Redis у вигляді ключ-значення з назвою ключа, що відповідає назві погодного показнику	Дані, що було записано в Redis за допомогою методу під відповідними ключами дійсно відповідає тим, що надійшли	Пройдено
UpdateHoursTable (позитивний тест)	Даний метод повинен записати дані в погодинну таблицю навпроти переданої поточної години, оскільки пройшло менше 5 хвилин з початку цієї години	Дані, що було передано в цей метод, дійсно записуються в погодинну таблицю навпроти переданої години лише в тому випадку, якщо хвилин минуло не більше 5	Пройдено
UpdateHoursTable (негативний тест)	Даний метод не повинен записувати дані в погодинну таблицю, оскільки хвилин з поточної години пройшло більше 5, а тому свіжі дані за цю годину вже записано	Дані, що було передано в цей метод, дійсно не записуються в погодинну таблицю, бо хвилин пройшло більше 5, і в таблиці напроти поточної години зберігаються дані, що були записані останніми	Пройдено
UpdateDaysTable (позитивний тест)	Цей метод записує в щоденну таблицю середні дані за останні день та ніч, агреговані на основі даних з погодинної таблиці, коли настає 6 ранку	Агреговані дані з погодинної таблиці дійсно записуються в щоденну таблицю, коли спрацьовує подія по настанню 6 години ранку	Пройдено

UpdateDaysTable (негативний тест)	Метод не повинен записувати агреговані дані з погодинної таблиці в щоденну таблицю, якщо не настала 6 години ранку	Даний метод дійсно не записує дані в таблицю, поки не спрацює подія по настанню 6 години ранку	Пройдено
GetClient	Даний метод повинен повертати екземпляр класу TelegramBotClient, що використовується в подальшому для взаємодії з API боту, з полем webhook, яке синхронізує програму і бота	Метод дійсно повертає результатом виконання даного методу екземпляр класу TelegramBotClient з полем webhook всередині нього	Пройдено
Execute (для текстового повідомлення)	Якщо команда боту повинна повертати результат у вигляді тексту, як от значення поточної температури, то і даний метод повинен повертати відповідний текст з даними з Redis	Метод повертає текст у вигляді “Current [назва погодного показнику] = [дані, що витягуються з Redis]”, а дані збігаються з поточними даними з Redis	Пройдено
Execute (для графічного повідомлення)	Якщо команда боту повинна повертати результат у вигляді графіку, то і даний метод повинен повертати графік з даними з відповідної таблиці	Метод дійсно повертає графік з даними з щоденної таблиці, якщо запит на відображення даних за обрану кількість останніх днів або з даними з погодинної таблиці, якщо запит на графік за останні 12 годин	Пройдено

Таблиця 6.1.2 — Тестування модулів коду апаратної частини програми

setJsonObject	Цей об’єкт приймає погодні дані і повинен створювати JSON об’єкт, який в подальшому буде надсилатись на серверну частину	Об’єкт дійсно створюється з полями саме тих назв, що необхідні при прийнятті серверною частиною та даними, що передавались в метод	Пройдено

6.2 Інтеграційне тестування

Даний вид тестування вже націлений на перевірку коректності роботи взаємодії декількох модулів або підсистем програми (див. таблицю 6.2.1). Інтеграційне тестування є важливим етапом, який часто оминають, оскільки вважається що і End-2-End тестування буде достатньо для перевірки роботи системи. Але як і з модульним тестуванням, якщо є непройдений тест — це означає, що якась частина системи вже не працює коректно і переходити на наступний рівень зарано.

Таблиця 6.2.1 — Інтеграційне тестування між апаратною та серверною частинами

Назва	Передача даних з апаратної частини на серверну
Опис	Код, яким прошивається апаратна частина системи, формує JSON об’єкт з даними, який передається в метод GetSensorData і конвертується в об’єкт класу SensorsData
Очікуваний результат	Апаратний код повинен формувати JSON об’єкт таким чином, щоб назви його полів та назви полів класу SensorsData збігались без урахування регістру, і вдало відправляти цей об’єкт на серверну частину з даними погодних показників, що відображаються на екрані порту, до якого підключено мікроконтролер
Отриманий результат	JSON об’єкт надходить в метод GetSensorData, екземпляр

	класу SensorsData формується без проблем, дані в полях цього екземпляру відповідають тим, що відображається на екрані порту, до якого підключено мікроконтролер
Статус тесту	Пройдено

6.3 End-2-End тестування

Для окремих людей-тестувальників, e2e тестування (як його ще пишуть) є напевно найважливішою та найпоказовішою частиною тестування в цілому, оскільки саме цей вид тестів працює так, наче це звичайний користувач виконує дії в роботі з додатком. В рамках цього тестування можна і навіть бажано перевіряти не окремі модулі функціоналу (кнопки, панелі), а проходити шлях від однієї точки додатку до іншої, при цьому виконуючи багато різних дій та залучаючи різноманітні модулі. Саме цей вид тестування стоїть на найвищому рівні і може точно сказати чи працює функціонал коректно для користувача. На відмінну від інших видів тестування, де тести так і називаються тестами, в End-2-End тести називають сценаріями.

В даному розділі тести будуть описані за допомогою методології BDD, де сценарій розділяють на три частини, які характеризуються ключовими словами Given, When, Then та And (див. таблицю 6.3.1, 6.3.2 та 6.3.3).

Таблиця 6.3.1 — Тестування загальної роботи додатку при відображенні текстового результату

Назва сценарію	Бот виконує команду відображення текстового повідомлення
Опис та очікуваний результат	**Given** Я як звичайний користувач **When** Відкриваю месенджер Telegram **And** Відкриваю чат з ботом WeatherAnalyzer **And** Обираю команду для відображення поточної температури

	Then Бот відповідає на команду повідомленням у вигляді “Current temperature = 20℃”
Отриманий результат	Бот відобразив повідомлення з текстовою відповіддю про поточну температуру
Статус тесту	Пройдено

Таблиця 6.3.2 — Тестування загальної роботи додатку при відображенні графічного результату за обрану кількість останніх днів

Назва	Бот виконує команду відображення графічного повідомлення за обрану кількість останніх днів
Опис та очікуваний результат	**Given** Я як звичайний користувач **When** Відкриваю месенджер Telegram **And** Відкриваю чат з ботом WeatherAnalyzer **And** Обираю команду для відображення графіку динаміки зміни температури за останні дні **And** Вводжу кількість останніх днів від 5 до 30 **Then** Бот відповідає на команду повідомленням, що містить зображення графіку зміни температури за вибрану кількість днів
Отриманий результат	Бот відобразив повідомлення, що містить зображення графіку зміни температури за вибрану кількість днів
Статус тесту	Пройдено

Таблиця 6.3.3 — Тестування загальної роботи додатку при відображенні графічного результату за останні 12 годин

Назва	Бот виконує команду відображення графічного повідомлення за останні 12 годин
Опис та очікуваний результат	**Given** Я як звичайний користувач **When** Відкриваю месенджер Telegram

	And Відкриваю чат з ботом WeatherAnalyzer **And** Обираю команду для відображення графіку зміни вологості повітря та атмосферного тиску за останні 12 годин **Then** Бот відповідає на команду повідомленням, що містить зображення графіку зміни вологості повітря та атмосферного тиску за останні 12 годин **And** Окремим повідомлення бот попереджає користувача про ймовірність опадів
Отриманий результат	Бот відобразив повідомлення, що містить зображення графіку зміни вологості повітря та атмосферного тиску за останні 12 годин та окреме текстове повідомлення про ймовірність опадів
Статус тесту	Пройдено

6.4 Тестування навантаження

Даний вид тестування є окремим та несхожим на всі попередні види (див. таблицю 6.4.1), оскільки він перевіряє не функціонал додатку, а його витримку та здатність переживати навантаження великою кількістю користувачів.

Оскільки даний проєкт може бути корисним великій кількості людей, то і максимальний поріг навантаження він теж повинен мати відповідний, а тому дане тестування необхідно провести з метою визначення цього максимального порогу.

Таблиця 6.4.1 — Тестування навантаження користувачами на систему

Назва	Проведення запиту на генерацію графіку
Опис	Команда на відображення графіку створює найбільше навантаження на систему
100 користувачів одночасно створюються запит на генерацію графіку	Система без видимих проблем та просідання по продуктивності генерує графіки для цих користувачів та відображає їх

200 користувачів одночасно створюються запит на генерацію графіку	Система без видимих проблем та просідання по продуктивності генерує графіки для цих користувачів та відображає їх
300 користувачів одночасно створюються запит на генерацію графіку	На даному рівні навантаження система вже має проблеми із швидкою генерацією графіків та відображенням їх одразу після генерації

6.5 Висновки до розділу

Даний розділ було присвячено тестуванню додатку за допомогою таких підходів як: unit-тестування, інтеграційне тестування, End-2-End тестування та тестування навантаження. Всі описані вище тести система пройшла, а отже здатна виконувати свою роботу коректно. При цьому протестовано було не лише окремі модулі або робота додатку в цілому, а й взаємодія апаратної частини із серверною, що є ключовим елементом програми.

Тестування навантаження показало, що система при 200 користувачах ще здатна адекватно працювати та видавати результат без проблем, а при 300 — вже виникають з цим труднощі. Отже, максимальний поріг навантаження знаходить десь між 200 та 300 користувачами.

ВИСНОВКИ

В рамках роботи над дипломним проєктом було сформовано, описано та реалізовано ідею програмно-апаратної системи, яка являє собою систему аналізу погодних показників для приватної метеорологічної станції. Дана система збирає погодні показники, такі як температура, вологість повітря, атмосферний тиск та тепловий індекс, аналізує їх, зберігає погодинно та на кожен день в базі даних і повертає проаналізовані дані у вигляді текстового повідомлення або графіку динаміки зміни. Дану систему розроблена таким чином, що за замовчуванням вона хоститься на хмарній платформі на віддаленому сервері, але за необхідності може бути розгорнута локально.

Як і розробка будь-якої ідеї, весь процес було розділено на декілька етапів. Перш за все було проведено аналіз існуючих рішень, щоб на основі отриманих знань визначити функціональні можливості та мати первинний погляд на те, які технології було б бажано використати для майбутньої системи.

Після цього було безпосереднє визначення технологій та рішень, які б задовольнили поставленим вимогам та дозволяли б розробити систему без зайвих зусиль. Обрані рішення дозволили не тільки мати можливість легкого розширення функціоналу, але й змінення масштабованості системи, що дає змогу залучити більше точок збору погодних показників.

Коли задача була постановлена, було обрано та обґрунтовано вибір ряду технологій та підходів для виконання задачі з розробки системи, наступним етапом слідує реалізація застосунку. В даній записці описано програмну реалізацію системи з детальним описом усіх підсистем та їх структури, наведено набір діаграм для опису побудованої системи у графічному вигляді, описано процес розгортання застосунку на хмарній платформі та графічний інтерфейс користувача з демонстрацією відповідних елементів інтерфейсу.

Не менш важливим етапом розробки програмного забезпечення є тестування, яке в даному випадку представлено у вигляді модульного, інтеграційного,

End-2-End та тестування навантаження. За його результатами стає очевидно, що система готова та здатна працювати адекватно і без перешкод, а тестування навантаження показало приблизну максимальну кількість користувач, при якій система здатна без перебоїв в роботі реагувати на створені запити та відображати результат.

Слід зазначити, що для отримання більш точних даних з різних точок необхідної користувачеві площі, необхідно мати більше одного набору апаратної частини, але завдяки впровадженим рішенням та подальшим вкладенням в проєкт це стає можливим. Також у випадку розгортання системи в хмарній платформі необхідно доплачувати за використовувані звідти ресурси. Кількість же користувачів, що здатна одночасно працювати з додатком, якщо порівнювати із сьогоднішніми рішеннями на ринку, є доволі низькою.

Зрештою, побудована система є сучасною, відповідає поставленим на початку роботи вимогам як в рамках функціоналу, так і в технічному аспекті, має сучасну та набираючу популярність складову інтерфейсу користувача, а легкість в розробці дозволяє швидко додавати новий функціонал.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Oregon Scientific Pro Weather Station WMR89A review URL: https://www.techhive.com/article/3074232/oregon-scientific-pro-weather-station-wmr89a-review-a-solid-option-for-casual-observers.html (Last accessed: 15.04.2020)
2. Всі складові системи «Oregon Scientific Pro Weather Station» URL: https://www.techhive.com/article/3074232/oregon-scientific-pro-weather-station-wmr89a-review-a-solid-option-for-casual-observers.html (Дата звернення: 15.04.2020)
3. BloomSky weather station/webcam review URL: https://www.techhive.com/article/3095958/bloomsky-weather-station-review.html (Last accessed: 15.04.2020)
4. Апаратна складова «BloomSky weather station» URL: https://www.techhive.com/article/3095958/bloomsky-weather-station-review.html (Дата звернення: 15.04.2020)
5. Netatmo Weather Station URL: https://www.netatmo.com/en-us/weather/weatherstation (Last accessed: 16.04.2020)
6. Датчики системи «Netatmo Weather Station» URL: https://www.androidpit.com/netatmo-weather-station-review (Дата обращения: 16.04.2020)
7. What is ASP.NET? URL: https://dotnet.microsoft.com/learn/aspnet/what-is-aspnet (Last accessed: 20.04.2020)
8. Microsoft Azure URL: https://docs.microsoft.com/ru-ru/azure/guides/developer/azure-developer-guide (Last accessed: 22.04.2020)
9. What is SaaS? URL: https://azure.microsoft.com/ru-ru/overview/what-is-saas/ (Last accessed: 23.04.2020)
10. What is IaaS? URL: https://azure.microsoft.com/en-us/overview/what-is-iaas/ (Last accessed: 23.04.2020)

11. What is PaaS? URL: https://azure.microsoft.com/en-us/overview/what-is-paas/ (Last accessed: 23.04.2020)

12. Что такое Entity Framework URL: https://metanit.com/sharp/entityframework/1.1.php (Last accessed: 25.04.2020)

13. Redis URL: https://redis.io/ documentation (Last accessed: 26.04.2020)

14. What is Azure SQL Database? URL: https://docs.microsoft.com/ru-ru/azure/azure-sql/database/sql-database-paas-overview (Last accessed: 28.04.2020)

15. Что за Telegram? URL: https://tlgrm.ru/faq (Дата обращения: 24.04.2020)

16. Знакомство с микроконтроллером ESP8266 URL: https://tproger.ru/articles/about-esp8266/ (Дата обращения: 13.04.2020)

17. Зовнішній вигляд датчику ESP8266 URL: https://voltiq.ru/shop/lolin-v3-nodemcu-lua-wifi-esp8266/ (Дата звернення: 13.04.2020)

18. DHT11, DHT22 – датчик температуры и влажности URL: http://studrobots.ru/dht11-dht22-temperature-arduino/ (Дата обращения: 13.04.2020)

19. Подключение датчика DHT11 или DHT22 к Ардуино URL: https://arduinomaster.ru/datchiki-arduino/datchiki-temperatury-i-vlazhnosti-dht11-dht22/ (Дата обращения: 13.04.2020)

20. Зовнішній вигляд датчику DHT22 та його апаратні входи/виходи URL: https://prom.ua/p410962794-dht22am2302-termometr-gigrometr.html (Дата звернення: 13.04.2020)

21. Огляд модулю атмосферного тиску BMP280 URL: https://3d-diy.ru/wiki/arduino-datchiki/sensor-bmp280/ (Дата звернення: 13.04.2020)

22. Understand the structure and syntax of ARM templates URL: https://docs.microsoft.com/en-us/azure/azure-resource-manager/templates/template-syntax (Last accessed: 30.04.2020)