НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис)

“__” _____2021 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія програмного забезпечення комп’ютерних систем»
спеціальності 121 «Інженерія програмного забезпечення»

на тему: «Програмна система визначення оптимального місця для занять спортом на основі показників стану міського довкілля»

Виконав (-ла): студент (-ка) 4 курсу, групи ІП-з72
(шифр групи)

Уваров Юрій Анатолійович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник к.т.н., доцент, Коган А. В. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант (нормоконтроль) _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент _____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.

Студент _____
(підпис)

Київ – 2021 р.

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалавр)

Освітньо-професійна програма

“Інженерія програмного забезпечення комп’ютерних систем”

спеціальності 121 “Інженерія програмного забезпечення”

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис)

“__” _____2021 р.

ЗАВДАННЯ
на бакалаврський дипломний проєкт студента

Уварова Юрія Анатолійовича

1. Тема проєкту *«Програмна система визначення оптимального місця для занять спортом на основі показників стану міського довкілля»*
керівник проєкту к.т.н., доцент, Коган А. В.
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від _____ 2021 року №_____
2. Термін здачі студентом закінченого проєкту *3 червня 2021 р.*
3. Вихідні дані до проєкту *технічна документація, теоретичні відомості, підсистема збору даних щодо показників стану міського довкілля, підсистема визначення оптимального місця для занять спортом*
4. Зміст розрахунково-пояснювальної записки (перелік питань, які розробляються)
Опис предметної області, дослідження існуючих фреймворків планування маршрутів, розробка архітектури системи, реалізація серверної частини системи на базі хмарного середовища Microsoft Azure, розробка клієнтської частини системи у вигляді web-сайту.

5. Перелік графічного матеріалу (з точним позначенням обов'язкових креслень): структурна схема системи, схема алгоритму взаємодії користувача з системою, діаграма класів, презентація до проекту.

6. Консультанта проєкту, з вказівкою розділів проєкту, які до них вносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль	Коган А. В., к.т.н., доцент		

7. Дата видачі завдання _____

Календарний план

№ п/п	Найменування етапів дипломного проєкту	Терміни виконання етапів проєкту	Примітки
1.	*Затвердження теми проєкту*	*08.01.2021-11.01.2021*	
2.	*Дослідження предметної області*	*12.01.2021-01.02.2021*	
3.	*Складання і узгодження технічного завдання*	*02.02.2021-10.02.2021*	
4.	*Розробка архітектури системи у хмарному середовищі*	*11.02.2021-28.02.2021*	
5.	*Розгортання базової версії системи (PoC) у хмарному середовищі Microsoft Azure*	*01.03.2021-13.03.2021*	
6.	*Розробка підсистеми взаємодії з фреймворком планування маршрутів*	*14.03.2021-07.04.2021*	
7.	*Розробка підсистеми збору даних про якість повітря на основі інформації від WAQI Project*	*08.04.2021-30.04.2021*	
8.	*Оформлення пояснювальної записки*	*01.05.2021-30.05.2021*	
9.	*Захист програмного продукту*	*15.05.2021*	
10.	*Передзахист*	*15.05.2021*	
11.	*Захист*	*17.06.2021*	

Студент-дипломник _____ Уваров Ю. А.
(підпис) (прізвище та ініціали)

Керівник проєкту _____ Коган А. В.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Дипломна робота виконана на 59 сторінках і містить 22 ілюстрацій, 1 таблицю, 3 додатки та 1 GitHub репозиторій. При розробці використано інформацію з 23 джерел.

Метою даної дипломної роботи є створення системи, яка допоможе потенційному користувачу спланувати маршрут для бігу на основі даних про якість повітря в місті Києві.

Для цього був реалізований клієнтський веб-додаток на основі фреймворку Angular та серверна частина на базі хмарного рішення Microsoft Azure. Був реалізований процес реєстрації та авторизації користувача та побудований інтерфейс взаємодії зі стороннім API. Також було реалізоване кешування запросів за допомогою Azure служби Cache for Redis.

Результатом дипломної роботи є програмна реалізація системи з використанням усіх перелічених сервісів та обраних фреймворків.

Ключові слова: здоров'я, індекс якості повітря, AQI, планування маршрутів, web-додаток, хмарне середовище, Azure, Active Directory, B2C.

SUMMARY

The Diploma project is made on 59 pages and contains 22 illustrations, 1 table, 3 appendices and 1 GitHub repo. 23 sources of information were used in writing the work.

The main idea of this project is to create a system that will help potential users to plan a route for running on the basis of data of air quality in the city of Kiev.

For this purpose, a client web application based on the Angular framework and a server part based on the Microsoft Azure cloud solution were implemented. The process of the user registration and authorization was implemented and the interface with third-party API was built. Query caching was also implemented using Azure's Cache for Redis service.

The result of this project is a software implementation of the application system using all the mentioned services and frameworks.

Keywords: health, air quality index, AQI, route planning, web-application, cloud environment, Azure, Active Directory, B2C.

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кіл. листів	№ екземпляра	Примітки
			Документація загальна			
	А4	ІАЛЦ.466200.002 ВП	Програмна система визначення	1		
			оптимального місця для занять			
			спортом на основі показників			
			стану міського довкілля.			
			Відомість проєкту			
	А4	ІАЛЦ.466200.003 ТЗ	Програмна система визначення	3		
			оптимального місця для занять			
			спортом на основі показників			
			стану міського довкілля.			
			Технічне завдання			
	А4	ІАЛЦ.466200.004 ПЗ	Програмна система визначення	59		
			оптимального місця для занять			
			спортом на основі показників			
			стану міського довкілля.			
			Пояснювальна записка			
	А4	ІАЛЦ.466200.005 Д1	Програмна система визначення	1		
			оптимального місця для занять			
			спортом на основі показників			
			стану міського довкілля.			
			Схема структурна			
	А4	ІАЛЦ.466200.006 Д2	Програмна система визначення	2		
			оптимального місця для занять			
			спортом на основі показників			
			стану міського довкілля.			
			Діаграма класів			
	А4	ІАЛЦ.466200.007 Д3	Програмна система визначення	1		
			оптимального місця для занять			
			спортом на основі показників			
			стану міського довкілля.			
			Алгоритм взаємодії			
			користувача з системою			

					ІАЛЦ.467200.001 ОА		
Зм.	Арк.	№ докум.	Підпис	Дата			
Розроб.		Уваров Ю.А.			Програмна система визначення оптимального місця для занять спортом на основі показників стану міського довкілля. Опис альбому	Літера	Аркуш. 1 / Аркушів 1
Перевір.		Коган А.В.					
Реценз.						НТУУ «КПІ ім. Ігоря Сікорського», ФІОТ	
Н. Контр.		Симоненко В.П.					
Затверд.		Стіренко С.Г.					

ВІДОМІСТЬ ДИПЛОМНОГО ПРОЕКТУ

до дипломної роботи

освітньо-кваліфікаційного рівня бакалавр

на тему: «Програмна система визначення оптимального місця для занять спортом на основі показників стану міського довкілля»

Київ – 2021 року

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кіл. листів	№ екземпляра	Примітки
			Завдання на дипломний проект			
	А4	ІАЛЦ.466200.003 ТЗ	Програмна система визначення	3		
			оптимального місця для занять			
			спортом на основі показників			
			стану міського довкілля.			
			Технічне проєкту			
	А4	ІАЛЦ.466200.004 ПЗ	Програмна система визначення	59		
			оптимального місця для занять			
			спортом на основі показників			
			стану міського довкілля.			
			Пояснювальна записка			
	А4	ІАЛЦ.466200.005 Д1	Програмна система визначення	1		
			оптимального місця для занять			
			спортом на основі показників			
			стану міського довкілля.			
			Схема структурна			
	А4	ІАЛЦ.466200.006 Д2	Програмна система визначення	2		
			оптимального місця для занять			
			спортом на основі показників			
			стану міського довкілля.			
			Діаграма класів			
	А4	ІАЛЦ.466200.007 Д3	Програмна система визначення	1		
			оптимального місця для занять			
			спортом на основі показників			
			стану міського довкілля.			
			Алгоритм взаємодії			
			користувача з системою			

Зм.	Арк.	№ докум.	Підпис	Дата	*ІАЛЦ.467200.002 ВП*		
Розроб.		Уваров Ю.А.			*Програмна система визначення оптимального місця для занять спортом на основі показників стану міського довкілля.* Відомість проєкту	*Літера*	*Аркуш.* 1 / *Аркушів* 1
Перевір.		Коган А.В.					
Реценз.						*НТУУ «КПІ ім. Ігоря Сікорського», ФІОТ*	
Н. Контр.		Симоненко В.П.					
Затверд.		Стіренко С.Г.					

ТЕХНІЧНЕ ЗАВДАННЯ

до дипломної роботи

освітньо-кваліфікаційного рівня бакалавр

на тему: «Програмна система визначення оптимального місця для занять спортом на основі показників стану міського довкілля»

Київ – 2021 року

Технічне завдання на дипломний проєкт

Зміст

1. НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Дане технічне завдання поширюється на розробку програмної системи визначення оптимального місця для занять спортом на основі показників стану міського довкілля. Область застосування: охорона здоров’я, контроль якості повітря, планування тренувань з бігу на основі даних системи.

2. ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на виконання роботи кваліфікаційно освітнього рівня «бакалавр інженерії програмного забезпечення», затверджене кафедрою обчислювальної техніки Національного технічного Університету України «Київський Політехнічний інститут ім. Ігоря Сікорського».

3. МЕТА І ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою розробки є надання потенційному користувачу не тільки засобу для контролю якості повітря, а ще й зручного інструменту для планування маршруту для бігу та інших видів тренувань на відкритому повітрі.

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелом розробки є технічна література з архітектурних особливостей побудови систем у хмарному середовищі, наукова література з моніторингу якості повітря та впливу забрудненого повітря на здоров’я людини під час спортивних навантажень та документація існуючих систем планування маршрутів.

5. ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до продукту

- Відображення на мапі широко доступної інформації з датчиків контролю якості повітря по місту Києву.
- Можливість описати бажаний маршрут на мапі.
- Побудова маршруту за вказаними користувачем точками.
- Збереження спланованих маршрутів в кабінеті користувача.

5.2. Вимоги до програмного забезпечення

- Наявність підписки та акаунту на платформі Microsoft Azure
- Наявність системної утиліти ngrok або її аналогу для забезпечення процесу тестування
- Бажано: наявність UNIX-подібної операційної системи

5.3 Вимоги до апаратного забезпечення

- Для локального розгортання Graphhopper необхідно принайні 4Гб оперативної пам’яті під час ініціалізації та створення графу на основі osm-даних місцевості

6. ЕТАПИ РОЗРОБКИ

Назва етапу	**Дата**
Затвердження теми проєкту	08.01.2021-11.01.2021
Дослідження предметної області	12.01.2021-01.02.2021
Складання і узгодження технічного завдання	02.02.2021-10.02.2021
Розробка архітектури системи у хмарному середовищі	11.02.2021-28.02.2021
Розгортання базової версії системи (PoC) у хмарному середовищі Microsoft Azure	01.03.2021-13.03.2021
Розробка підсистеми взаємодії з фреймворком планування маршрутів	14.03.2021-07.04.2021
Розробка підсистеми збору даних про якість повітря на основі інформації від WAQI Project	08.04.2021-30.04.2021
Оформлення документації дипломної роботи	01.05.2021-03.06.2021

ПОЯСНЮВАЛЬНА ЗАПИСКА

до дипломної роботи

освітньо-кваліфікаційного рівня бакалавр

на тему: «Програмна система визначення оптимального місця для занять спортом на основі показників стану міського довкілля»

Київ – 2021 року

					ІАЛЦ.467200.004 ПЗ			
Зм.	*Арк.*	*№ докум.*	*Підпис*	*Дата*				
Розроб.		Уваров Ю.А.			*Програмна система визначення оптимального місця для занять спортом на основі показників стану міського довкілля.* Пояснювальна записка	*Літера*	*Аркуш.*	*Аркушів*
Перевір.		Коган А.В.					*1*	*59*
Реценз.						*НТУУ «КПІ ім. Ігоря Сікорського», ФІОТ*		
Н. Контр.		Симоненко В.П.						
Затверд.		Стіренко С.Г.						

ЗМІСТ

СПИСОК УМОВНИХ СКОРОЧЕНЬ

API – Application Programming Interface (прикладний програмний інтерфейс).

AQI – Air Quality Index (індекс якості повітря).

AWS – Amazon Web Services (хмарного платформа від Amazoe).

Azure AD – служба Active Directory від Microsoft Azure.

Azure AD B2C – це служба управління посвідченнями Microsoft Azure.

B2C – business-to-customer (концепція “Бізнес для споживача”).

FaaS – Function-as-a-Service (архітектурний шаблон “Функція як послуга”).

GAHP – Function-as-a-Service (архітектурний шаблон “Функція як послуга”).

GDPR – загальний регламент про захист даних.

JSS – стилізація CSS на мові Javascript у декларативному стилі.

JSX – надбудова над JavaScript, яка дозволяє використовувати XML-подібний синтаксис у JavaScript.

OSM – Open Street Map (відкрита вулична карта) – картографічний сервіс та однойменний формат даних.

MSAL – Microsoft Authentication Library (Бібліотека аутентифікації Майкрософт).

Multi-tenant – режим колективної оренди – архітектурний підхід, при якому єдиний екземпляр додатків одночасно працює з декількома клієнтами.

PoC – Proof of Cocept (Доказ концепції) – демонстрація практичної здійсненності будь-якого методу, ідеї, технології, тощо.

SaaS – Software-as-a-Service (концепція “Програмне забезпечення як сервіс”).

SSO – Single Sign-On (Технологія єдиного входу) – дозволяє користувачеві переходити з одної системи в іншу без повторної аутентифікації.

ВСТУП

За статистикою 2020-го року Київ посідає 10-е місце в Європі за показниками забруднення навколишнього середовища згідно рейтингу Numbeo, а Україна – 4-те місце у світі за кількістю смертей через погану екологію згідно рейтингу The Global Alliance on Health and Pollution (GAHP). При цьому серцево-судинні захворювання займають перше місце серед усіх хвороб, а погана екологія та, зокрема, забруднене повітря лише погіршує ситуацію. Лондонський університет королеви Марії [1] провів дослідження щодо впливу забрудненого повітря на серцево-судинну систему. За словами експертів, якщо цілком здорова людина постійно дихає навіть трохи забрудненим повітрям, це може нашкодити її серцю, а зміни серця можна порівняти з тими, що відбуваються на ранніх стадіях серцевої недостатності. Такі речовини як оксид вуглецю (чадний газ, CO), озон (O3), тверді частинки (PM10), діоксид сірки (SO2) та діоксид азоту (NO2) вважаються найбільш небезпечними для здоров'я людини. Наприклад, окис вуглецю (CO) надає згубний вплив на здатність крові транспортувати кисень за допомогою еритроцитів. CO потрапляє в кров через легені і, умовно, «займає місце» кисню в еритроцитах.

У міру того, як кількість CO в крові збільшується, продуктивність людини постійно падає. Якщо людина при цьому ще й має проблеми астматичного або серцево-судинного характеру, то ні про який "оздоровчий біг" мова вже не йде.

Метою дипломної роботи є створення системи збору даних щодо якості повітря в конкретний момент часу та планування оптимального маршрут пробіжки, якого слід дотримуватися, щоб оминути усі зони з забрудненим повітрям.

З технічної точки зору було вирішено скористатися наступними даними та технологіями:

1. Використати геодані з сайту OpenStreetMap.

2. Обрати один з наявних фреймворків для планування маршруту на основі даних OpenStreetMap.
3. Побудувати програмний інтерфейс для отримання та обробки даних щодо показників індексу якості повітря (AQI).
4. На базі хмарного середовища Microsoft Azure взагалі та Azure Active Directory зокрема реалізувати контроль доступу до кабінета користувача з можливістю швидкої реєстрації останнього через популярні соціальні мережі.
5. Для забезпечення взаємодії користувача з системою та візуалізації маршруту створити за допомогою фронт-енд фреймворку Angular клієнтський web-додаток та інтегрувати отримані дані з останнім.

РОЗДІЛ 1

АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ І ПОСТАНОВКА ЗАДАЧІ

1.1 Вплив якості повітря на здоров'я людини під час фізичних навантажень.

Згідно з даними ВООЗ [2] більше 90% населення світу проживає у районах з рівнем забруднення атмосферного повітря, що перевищує допустимі межі, а дев’ять з десяти людей дихають повітрям, в якому присутня висока концентрація забруднюючих речовин. До основних забрудників відносяться :

1. Тверді частинки (PM10 і PM2.5) – це по суті, пил від доріг, продукти горіння (наприклад, пожеж) та автомобільні викиди [3]. Вони можуть проникати глибоко в легені і осідати в них, а найбільш руйнівного впливу завдають саме частинки діаметром менше 2,5 мікрон – вони, в свою чергу, можуть навіть потрапити в кровоносну систему. Хронічний вплив таких частинок посилює ризик розвитку серцево-судинних і респіраторних захворювань та раку легенів.
2. Діоксиду азоту (NO_2), основним джерелом якого є результат процесів згоряння (обігрів, робота двигунів, вироблення електроенергії, тощо). З впливом NO_2 пов'язують зниження функції легень, поглиблення симптомів астми і проблеми в роботі серця.
3. Діоксид сірки (SO_2), який представляє собою безбарвний газ з різким запахом. Як правило він утворюється при спалюванні різних викопних видів палива (вугілля, нафти). Це дуже актуально для великих міст, оскільки спалювання палива, що містить сірку, часто використовують для обігріву осель та вироблення електроенергії. SO_2 впливає на дихальну систему, функції легень і може викликати подразнення очей. В свою чергу, запалення дихальних шляхів призводить до появи кашлю, астми і розвитку хронічного бронхіту.

4. Оксид вуглецю (чадний газ, CO), наявність якого в повітрі може залежати від багатьох факторів, але все одно основним джерелом є викиди від автомобілів. Висока концентрація оксиду вуглецю може утворюватися через масове спалювання листя, трави, сміття, сухих залишків з полів, горіння болот (торфовищ) і т.д.
5. Озон (O3), який є одним з головних компонентів фотохімічного смогу. Останній утворюється в результаті реакції з сонячним світлом (фотохімічної реакції) оксиду азоту (NOx) і летючими органічними сполуками, які, як правило, виділяються транспортними засобами або викидаються в повітря машинами та промисловими підприємствами. Надмірна кількість озону в повітрі може викликати проблем з диханням, знизити функцію легень, спровокувати астму.

Саме масова концентрація PM 2.5 частинок в одиниці об'єму повітря є найважливішим і основним параметром оцінки якості повітря для кожної конкретної географічної локації. Всесвітня організація охорони здоров'я (ВООЗ) встановила, що гранична допустима концентрація дрібнодисперсних частинок для здоров’я людини складає:

- середньодобовий рівень PM2.5 в повітрі не повинен перевищувати 25 мкг / м.куб., а середньорічний – не більше 10 мкг / м.куб.
- середньодобовий рівень PM10 – не більше 50 мкг / м.куб, а середньорічний – не більше 25 мкг / м.куб.

В цілому, коли мова заходить про вплив забрудненого повітря на здоров’я людини під час занять спортом, то однозначності даних не існує, оскільки прямих тестів за участю певної цільової групи не проводилось. Проте існує ряд досліджень: дослідження до Лос-анджелеської Олімпіади 1984-го року [4], дослідження до Пекінської Олімпіади (яке є найбільш актуальним, оскільки проблема смогу над Пекіном є загальновідомою) [5], дослідження впливу забрудненого повітря на результати забігів [6], дослідження фізичної активності у середовищі з забрудненим повітрям [7].

Наприклад, автори дослідження [6] взяли дані 300 тис. марафонців, які брали участь в забігах в 37 містах Китаю з 2014 по 2015 рік включно. Параметрами дослідження був обраний їх фінішний час і показник якості повітря AQI, що складається з 6 основних показників забруднення (PM2.5, PM10, SO2, NO2, CO, O3), де мінімальне значення 0 – найчистіше повітря, до 100 – значення коефіцієнту якості без негативного впливу для здоров'я бігуна, і максимальне значення 500 – дуже небезпечно для здоров'я. При цьому дослідники підрахували, що кожне збільшення AQI в два рази збільшувало час фінішу марафону на 4.08%.

Автори дослідження [7], по суті, провели два окремих дослідження і при цьому в обох випадках були виявлені переваги занять при чистому повітрі в порівнянні з забрудненим. Під час першого дослідження тести проводилися на велосипедах: один тест був проведений в лабораторії з чистим повітрям, другий в загазованому місті в годину-пік. Тільки в першому випадку було відзначено зростання BDNF – білку, який відіграє особливу роль у синаптичній пластичності мозку. Остання, в свою чергу, особливо важлива для регулювання пам’яті, процесів навчання та апетиту. Під час другого дослідження учасники проходили 12-тижневу програму тренувань. Частина з них тренувалася в сільській місцевості, інша – в місті. Ті, хто тренувалися в місті гірше пройшли нейропсихологічний тест опісля закінчення програми.

Цікавим є дослідження до Пекінської Олімпіади [5], оскільки за результатами дослідження можна зробити висновок щодо впливу кожного з 6 основних показників забруднення на організм спортсмену. Наприклад, було встановлено, що збільшенні концентрації CO в крові призводить до збільшення частоти серцевих скорочень і зниження рівня максимального споживання кисню, що впливає на результативність роботи і в кінцевому підсумку призведе до погіршення спортивних результатів, особливо змагань на витривалість – на довгих дистанціях (10 км, напівмарафон, марафон).

В свою чергу вплив підвищених концентрацій озону призводить до появи таких симптомів, біль у грудях, як кашель, головний біль, роздратування очей і, що дуже важливо для спортсменів, зменшення обсягу повітря в видиху за одну секунду. При цьому фізичні вправи при підвищеній концентрації забруднюючих речовин включаючи озон або діоксид сірки, викликає великі ризики для здоров'я у порівнянні зі станом спокою. Через це навіть деякі спортсмени-марафонці були змушені відмовитися брати участь в Олімпіаді в Пекіні через можливу загрозу їх здоров'ю.

1.2 Існуючі джерела моніторингу якості повітря

Наразі існує ряд джерел для моніторингу якості повітря. Всі вони так чи інакше зводять показник якості повітря [8, 9] до значення від 0 до 500, де 0 – це найбезпечніший, а 500 – найнебезпечніший показник для здоров’я людини. AQI розраховується виходячи з чотирьох параметрів: рівня озону, твердих частинок, оксиду вуглецю і діоксиду сірки. AQI умовно ділять на шість рівнів.

Таблиця 1.1 – показник AQI за впливом на здоров'я

AQI значення	Якість повітря за впливом на здоров'я	Граничні значення pm2.5 частинок (мкг/м3)
0 - 50	Хороше	0 - 15.4
51 - 100	Середнє	15.5 - 40.4
101 - 150	Шкідливе для груп ризику	40.5 - 65.4
151 - 200	Шкідливе	65.5 - 150.4
201 - 300	Дуже шкідливе	150.5 - 250.4
301 - 500	Небезпечне	250.5 - 500.4

Сайт www.airindex.eea.europa.eu – це головний європейський сайт з моніторингу якості повітря, який збирає та візуалізує дані з сотень датчиків по всій Європі. Проблема в тому, що він не моніторить дані з України, але існують й інші джерела, які можна використати для моніторингу якості повітря в Україні та Києві зокрема.

www.iqair.com – на підставі даних саме цього джерела Київ рік тому потрапив в топ 3 найбрудніших міст світу. Це сайт швейцарської комерційної компанії, яка продає свої лічильники і, в якості реклами, розставила їх по всьому світу. Також в них є свої Android та iOS додатки.

Плюси використання цього джерела:

1. Дані оновлюються щогодини.
2. Відкрите безкоштовне API для некомерційної розробки (10 тис. запитів/місяць).

Мінуси:

1. Датчики розташовані лише в Києві.
2. Дуже мало датчиків в наявності.

www.saveecobot.com – найпоширеніший сайт з інформацією щодо якості повітря в Україні. По суті, коли говорять про AQI в Києві, то найчастіше мають на увазі саме дані з цього сайту. Дані збираються в 111 населених пунктах по всій Україні.

Плюси використання цього джерела:

1. Швидке оновлення даних.
2. Відкрите безкоштовне API для некомерційної розробки.
3. Не мала кількість датчиків.

Мінуси:

1. Немає даних щодо кожного з видів забруднення окремо.

aqicn.org – світовий індекс якості повітря, який є некомерційним проектом, започаткованим у 2007 році. Його місія – сприяти поінформованості громадян про забруднення атмосферного повітря та надавати уніфіковану та

загальносвітову інформацію про якість повітря. Проект надає доступну і прозору інформацію про якість повітря для більш ніж 130 країн, охоплюючи понад 30 тис. станцій у 2 тис. великих містах, через два веб-сайти: aqicn.org та waqi.info. На мою думку, “Світовий індекс якості повітря” – це найкращий варіант з інформацією щодо якості повітря в Україні і світі.

Плюси використання цього джерела:

1. Актуальне і швидке оновлення даних.
2. Відкрите безкоштовне API для некомерційної розробки.
3. Дуже велика кількість датчиків.
4. Дуже хороше покриття України та Києва.
5. Наявні дані щодо кожного з видів забруднення окремо.

Мінуси:

1. Немає можливості отримати всі необхідні дані одним запитом – необхідно робити запити за кожним датчиком окремо, щоб отримати розгорнуту інформацію по рівню кожного з видів забруднень.

Незважаючи на наявний мінус, було вирішено взяти за основну дані з останнього джерела, оскільки ці дані є найповнішими та найточнішими, а наявний мінус було вирішено нівелювати на програмному рівні під час реалізації системи додатків.

1.3 Постановка задачі

З названого вище можна виділити декілька основних проблем, на які потрібно звернути увагу:

1. Існуючі додатки з бігу можуть лише допомогти спланувати маршрут або показати вже пройдений маршрут постфактум, але не надають простої можливості створити його самостійно за маркерами, якщо користувач, наприклад, планує бігати лише в певному районі міста.

2. Існуючі системи можуть показати якість повітря в конкретній точці на мапі, але не можуть надати вичерпну інформацію щодо якості повітря вже в декількох метрах від вказаної точки.
3. Мною не було знайдено додатків, які б могли під час планування маршруту оминути небезпечні зони. Наприклад, зони з підвищеним рівнем забрудненості повітря.

Отже, двома головними питаннями, на які покликана відповісти дана система є:

1. Як людині отримати релевантну інформацію щодо якості повітря впродовж усього маршруту бігу, якщо існуючі джерела даних можуть лише надати дані про якість повітря в конкретній точці на мапі?
2. Як людині побудувати безпечний з точки зору якості повітря маршрут для бігу?

Задача, яку пробує вирішити дана робота – створення системи, яка дозволяє не тільки візуалізувати ситуацію з забрудненням повітря в конкретному районі, а й допомагає користувачу спланувати найкращий маршрут для пробіжки, який буде оминати усі небезпечні для здоров'я зони. Наприклад, зони з підвищеним рівнем забрудненості повітря (в майбутньому окрім якості повітря планується, що система буде зважати й на інші важливі для здоров'я людини фактори).

Основними вимогами до системи, що розроблюється, з точки зору бізнесу (користувача) є:

- Візуальне представлення якості повітря в певному районі в конкретний момент часу для оцінки стану довкілля;
- Можливість встановлення користувачем ключових точок (waypoint-ів) між якими повинен прокладатися маршрут для бігу;
- Планування маршруту, який проходить через встановлені користувачем маркери, а також оминає зони з забрудненим повітрям;

- Авторизація користувача та збереження історії створених маршрутів.

У свою чергу, серед вимог до архітектури системи можна виділити наступні:

- Підтримка мікросервісної архітектури;
- Використання хмарного середовища для розгортання проекту;
- Використання severless-рішень та застосування архітектурних шаблонів Function-as-a-Service (FaaS) та Software-as-a-Service (SaaS);
- Використання сучасного frontend-фреймворку;
- Реалізація Single Sign-On (SSO) на базі хмарного рішення задля подальшого масштабування проекту в сторону мульти-клієнтської системи додатків.

ВИСНОВКИ ДО РОЗДІЛУ 1

Суть першого розділу полягала в дослідженні впливу забрудненого повітря на організм людини під час занять спортом та зокрема бігу, параметрів, які характеризують рівень забруднення повітря, а також дослідження наявних джерел, дані щодо якості повітря яких можна брати за основу для створення системи.

Були проаналізовані декілька досліджень впливу забрудненого повітря на результати спортсменів-бігунів (в основному марафонців). Зокрема дослідження до Пекінської та Лондонської олімпіад, а також дослідження за 2014-2015-ті роки марафонів по всьому Китаю. Також було вивчено можливості трьох наявних джерел інформації щодо збору та візуалізації якості повітря та рівня забрудненості. За результатами дослідження як основне джерело даних для майбутньої систему був обраний сайт aqicn.org – світовий індекс якості повітря. Також була сформульована задача та основні бізнес-вимоги до системи з точку зору користувача та вимоги до архітектури системи.

РОЗДІЛ 2
ПІДСИСТЕМА ПЛАНУВАННЯ МАРШРУТУ

2.1 Фреймворки планування маршрутів

На практиці багато для пошуку найкоротшого маршруту між точками А та Б найчастіше використовуються алгоритми А* та Дейкстри. Не будемо зупинятися на цьому докладно, проте саме на ці два алгоритми, як правило, використовуються всіма сучасними фреймворками, про яке піде мова нижче.

У 2004 році з'явився проект з відкритим кодом, якому судилося назавжди змінити ландшафт маршрутизації і навігації – OpenStreetMap (OSM). OSM став ключовим компонентом розробки движків маршрутизації з відкритим кодом, оскільки світ отримав симбіоз високоефективних відкритих алгоритмів маршрутизації та аналогічних відкритих даних вулиць для маршрутизації. На сьогодні OSM [10] містить > 7,6 мільярда вузлів – світлофорів, дерев, дорожніх знаків і 200 мільйонів ділянок доріг. Це, можливо, найбільша в світі відкрита база даних фізичних об'єктів і найкраще джерело даних у багатьох країнах.

2.1.1 OSRM

OSRM – один з найперших фреймворків маршрутизації, побудований на базі даних з OSM [11, 12]. У 2019 активна розробка OSRM припинилась, і розробники сфокусувалися на розробці нового фреймворку Valhalla, про який піде мова нижче. За останні 2 роки активна розробка нових функцій безумовно значно впала, але все ще додаються нові функції.

Перевагами фреймворку є:

- Продуктивність: маршрутизація займає кілька мс навіть для довгих маршрутів > 1000 км. API може надати мільйони результатів за лічені секунди.

- Інфраструктура: навколо OSRM є гарна колекція пов'язаних проектів з відкритим кодом: клієнтські додатки, NodeJs та C++ бібліотеки з непоганою документацією, а також безліч посібників спільноти, надбудов і т.д.
- Гнучкість налаштування: профілі в OSRM визначаються як сценарії Lua, досить простої мови сценаріїв. Існує безліч готових прикладів та функцій для використання при визначенні нових профілів або адаптації існуючих. Це досить простий та ефективний спосіб зміни визначення профілів та атрибутів OSM.
- Трафік: OSRM може оновлювати записи швидкості на своєму графіку за допомогою простих файлів CSV, які можна використовувати для регулярних оновлення трафіку.

Недоліки OSRM:

- Вимоги до ОЗП: повний конвеєр попередньої обробки для файлу планети OSM займає > 250 ГБ ОЗП для кожного профілю (однак під час виконання, тобто при запуску запитів, розмір зменшується до ~ 35-40 ГБ).
- Гнучкість: OSRM – не найкращий вибір, якщо ви хочете встановити деяким атрибутам дороги різну вагу для кожного запиту (наприклад, щоб уникнути шосе або дорожніх зборів). Вся інформація про вагу вбудована в графік і не може бути призначена динамічно.

2.1.2 Valhalla

Про проект Valhalla говорить вже той факт, що Tesla вибрала [13] Valhalla в якості своєї автомобільної навігаційної системи завдяки задовільному часу роботи і унікальній гнучкості в оцінці витрат часу під час обробки сегментів дороги. Одним з основних переваг Valhalla є те, що всі профілі використовують один і той же граф, що стало можливим завдяки динамічній оцінці витрат часу виконання. При є можливість вплинути на пошук маршруту

з допомогою безлічі параметрів, як різні фактори/штрафи/витрати на пальне, тощо.

Переваги:

- Гнучкість: основним плюсом є параметризація динамічних запитів, якої не вистачає більшості інших механізмів маршрутизації, при збереженні хорошої продуктивності і невеликому обсязі пам'яті. Кожен запит маршруту може бути розрахований з різним набором витрат, штрафів і інших чинників крім статично визначених профілів.
- Вимоги до ОЗП: через рішення маршрутизації на мозаїчному ієрархічному графі (схожому, наприклад, на фрагменти карти) і розумних механізмів завантаження фрагментів [13], Valhalla може працювати на дуже скромному обладнанні, такому як бюджетний смартфон або Raspberry Pi для локальних наборів даних.
- Маршрутизація і трафік, що залежать від часу: Valhalla враховує час, і це досить унікально в цій області [13]. Можна вказати час відправлення або прибуття, і алгоритми Valhalla будуть враховувати тимчасові обмеження, а також поточні або історичні джерела даних про трафік при розрахунку маршруту.

Недоліки:

- Продуктивність: Valhalla утрималася від дорогої (додаткової) попередньої обробки і вирішила вбудувати деякі методи прискорення, що використовуються іншими фреймворками, в саму структуру графа. Однак це виявилося не так не так оптимізовано, як, наприклад, в OSRM. Хоча стандартна маршрутизація досить ефективна, матричні запити виконуються порівняно довго.
- Налаштування: документація [10] по налаштуванню Valhalla може бути досить громіздкою спочатку. Особливо це вірно якщо спробувати його запустити як загальнодоступний HTTP API. Проте

завдяки існуючому проекту в Docker першочергове розгортання Valhalla можна трохи спростити.

2.1.3 Graphhopper

Graphhopper [14] розпочався як особистий проект одного із засновників, проте він швидко перетворився в комерційний сервіс з відкритим вихідним кодом, в якому основна увага приділяється вирішенню дуже складних завдань маршрутизації транспортних засобів [13]. Graphhopper приділяє велику увагу підтримці і вдосконаленню свого високоякісного механізму маршрутизації для широкої публіки.

Переваги:

- Продуктивність: хоча Graphhopper трохи відстає від OSRM, він є одним з найшвидших механізмів маршрутизації [13, 14]. Навіть маршрути континентального масштабу не часто займають більше кількох сотень мс.
- Гнучкість: за останній рік Graphhopper зазнав значних змін щодо гнучкості під час виконання не стандартних профілів. Однак це має неабиякий вплив на продуктивність на довгих маршрутах. Їх так званий «швидкісний» режим такий же негнучкий (але майже такий же швидкий), як і найбільш продуктивний алгоритм маршрутизації OSRM.
- Транзит: Graphhopper підтримує маршрутизацію громадського транспорту через опубліковані GTFS канали. Ці канали повинні публікуватися місцевою мережею громадського транспорту і навіть можуть містити інформацію про транспортних лініях в реальному часі.

Недоліки:

- Вимоги до ОЗП: для кращої продуктивності етап попередньої обробки в Graphhopper вимагає досить багато ОЗП, хоча і не так

багато, як OSRM. Що стосується всієї планети, то тут цифри менш оптимістичні і треба бути готовим до 64-128 ГБ оперативної пам'яті для попередньої обробки для кожного профілю [13, 14]. Зберігання графіка в пам'яті набагато менш вимогливо і вміщується в <32 ГБ на профіль. З деякими опціями Graphhopper використовує менше 32 ГБ ОЗП: використовує mmap, а не будує орієнтири або ієрархії стиснення, однак значно жертвує швидкістю збірки або часом виконання.

- Гнучкість налаштування: підготовка додатковий профілів для Graphhopper досить накладна, оскільки вихідний код Java повинен бути написаний (тут немає Lua) з налаштуванням правил використання атрибутів OSM.
- Кілька функцій з закритим вихідним кодом: зокрема, Graphhopper вирішив закрити свій код API матриці через його критично важливість для бізнесу [14].

2.2 Критерії вибору фреймворку

При виборі фреймворку планування маршруту для проекту я керувався наступними критеріями:

- Швидкість розгортання.
- Простота налаштування.
- Наявність вбудованого API.
- Наявність можливості вносити зміни в існуючу OSM модель даних та створювати “мертві зони”, через які маршрут проходити не повинен.
- Хороша документація та приклади.
- Основна мова програмування високого рівня (C#, Java, JavaScript), якщо це можливо.

Всім цим пунктам повністю відповідає фреймворк Graphhopper, який написаний на Java, має дуже велике ком'юніті та чудову документацію з безліччю прикладів. Окрім того Graphhopper також має платну версію у вигляді серверної частини з налаштованим для широкого загалу API. В цій роботі я рухався шляхом open source і тому серверна частина Graphhopper була розгорнута локально.

Локальна open source версія Graphhopper нічим не відрізняється від платної, окрім того, що розробнику доведеться розгортати її самостійно. Програмний інтерфейс, алгоритми, конфігураційний файл та інші частини фреймворку абсолютно ідентичні та нічим не відрізняються від платної версії.

2.3 Особливості розгортання фреймворку Graphhopper

В цілому, можна назвати наступні ключові особливості при розгортанні локального серверу Graphhopper з певним пакетом даних:

1. При першому запуску фреймворк будує граф даних на основі переданого osm-файлу, тому до початку роботи з фреймворком необхідно завантажити з сайту проекту Geofabrik, де розробники збирають дані з Open Street Map в форматі файлів .osm, потрібний файл. В даному випадку – файл з даними по Україні "ukraine-latest.osm.pbf".
2. Далі для локального розгортання необхідно завантажити конфігураційний файл-приклад з сайту Graphhopper [14] та: або скомпілювати фреймворк з вихідного коду, або завантажити скомпільований jar-файл за адресою: https://graphhopper.com/public/releases/graphhopper-web-2.3.jar
3. Далі необхідно встановити JRE та створити конфігураційний файл для запуску Graphhopper, беручи за основу аналогічний файл-приклад з сайту розробника.
4. Далі виконати команду:

```
$> java -Ddw.graphhopper.datareader.file=ukraine-latest.osm.osm.pbf -jar
*.jar server config.yml
```

5. По суті, саме ця команда запускає процес аналізу osm-файлу та створю на його основі граф даних, про який йшла мова в першому пункті.
6. Після завершення побудови графу даних прикладний програмний інтерфейс буде доступний за локальною адресою http://localhost:8989/
7. Прикладний програмний інтерфейс фреймворку налічує багато GET та POST запитів і доступний для ознайомлення у документації [8], тому тут зупинимось тільки на одному ендпоінті: [GET] /route – запит для отримання маршруту між вказаними на мапі координатами. Також є можливість вказати “заблоковані зони” – координати, через які маршрут пролягати не повинен. Приклад запиту:

```
$> curl
"https://graphhopper.com/api/1/route?
point=51.131,12.414&point=48.224,3.867&vehicle=car&block_area=50.131,4.414,
1000;49.11,14.4,10"
```

ВИСНОВКИ ДО РОЗДІЛУ 2

Суть другого розділу полягала в дослідженні існуючих фреймворків маршрутизації та виборі фреймворку, який відповідає всім необхідним критеріям. За результатами дослідження був обраний фреймворк Graphhopper та розгорнута серверна частина з необхідним для подальшої роботи прикладним інтерфейсом. За допомогою програми типу Postman та допоміжних online web-додатків для розпізнавання та нанесення на Open Street Map фігур типу polyline була перевірена можливість оминання перешкод та “мертвих зон” на мапі. Результатом цього розділу є розгорнута локально серверна частина Graphhopper.

РОЗДІЛ 3
АРХІТЕКТУРА СИСТЕМИ ДОДАТКІВ

3.1 Опис структурної схеми та обґрунтування вибору технологій

Структурна схема система доступна у додатку 1. Всю систему можна умовно розділити на 3 логічних блока:

1. Інфраструктура Azure
2. Клієнтська частина
3. Сторонні прикладні інтерфейси

Кодова база проєкту доступна за посиланням на GitHub репозиторій: https://github.com/ottodranik/HealthyRunningProject.KPI.Diploma

3.1.1 Інфраструктура Azure

Наразі найпопулярнішими хмарними платформами є Microsoft Azure, Amazon Web Services та Google Cloud. В кожній з систем є свої плюси та мінуси.

Наприклад, якщо ми говоримо про гібридний підхід, то Microsoft завдяки своєму Azure Stack дуже довго була попереду конкурентів [15]. Платформа пропонує можливість розміщення хмарних сервісів Azure на локальних серверах обробки даних з відкритим порталом, кодом та API-інтерфейсами з метою простої інтеграції та сумісності. Пізніше, до цього напрямку приєднались AWS та Google Cloud. З точки зору обчислювальної потужності найпопулярнішими хмарними платформами, як було виявлено при дослідженні цієї теми, є Azure та AWS – їх обчислювальні потужності схожі, а перелік послуг регулярно збільшується. При цьому цінова політика всіх трьох платформ дуже різна. Наприклад, в Azure ресурси постачальника оплачуються з округленням у більшу сторону за кожну хвилину. Знижки надаються в залежності від обсягу використаних послуг. В той час як AWS пропонує три різні підходи до ціноутворення.

В цілому, хмарні обчислення [16] – це надання обчислювальних служб (у тому числі серверів, сховищ, баз даних, мереж, програмного забезпечення, аналітики, тощо) через мережу Інтернет. Такі служби прискорюють впровадження інновацій, підвищують гнучкість ресурсів, а також забезпечують економію через високу масштабованість. Платформа Microsoft Azure [16] пропагує модель SaaS – модель надання ліцензії на програмне забезпечення за підпискою. SaaS розшифровується як Software-as-a-Service – програмне забезпечення як послуга, також зустрічається переклад «інформація як сервіс».

В конкретному випадку, хмарні служби та використання serverless та Function-as-a-Service (FaaS) підходів дали змогу прискорити розробку даного проекту та не відволікатися на процеси оркестрування мікросервісів. Також використання додаткових служб, таких як Key Vault, дало змогу забезпечити достатній рівень безпеки даних та керування доступом, а використання служби хмарного Redis сервісу дало змогу зробити просте та потужне сховище для кешування запитів.

В цілому ж, мною було обрано саме Azure платформу для реалізації проекту. Перевагами платформи Microsoft Azure, які зіграли ключову роль при виборі, стали:

1. Наявність Azure Active Directory [17], що дозволило реалізувати процес авторизації користувача практично повністю “з коробки” та не займатися менеджментом ключів, паролів та особистих даних користувачів на стороні продукту, віддавши керування цими функціями в службу Azure Active Directory.
2. До реалізації цього проекту я вже мав досвід роботи з платформою Microsoft Azure та .NET CORE фреймворком і тому зв’язка Azure + C# для мене виглядала найбільш очевидним вибором.
3. Щодо цінової політики, Microsoft надає можливість річного безкоштовного використання майже всіх ключових сервісів (окрім

сервісу Azure Cache for Redis), які були потрібні для реалізації даного проекту у його початковій фазі.

У реалізованому проекті інфраструктура Azure представлена наступними сервісами (структурна схема у додатку 1):

1. Azure-функції.
2. Azure Active Directory.
3. Azure Cache for Redis.
4. Azure Key Vault
5. Azure Cosmos DB

Azure-функції є реалізацією FaaS підходу. Вони являють собою доступну за запитом хмарну службу, яка надає регулярно оновлювану інфраструктуру і всі ресурси, необхідні для запуску додатків. У дипломному проекті це є:

- Запит на побудову маршруту за вказаними маркерами та “мертвими” зонами.
- Запит для отримання даних від сторонніх сервісів щодо якості повітря.
- API-з’єднувачі.

Більш докладно ця частина представлена у схемі класів у додатку 2.

Azure Active Directory є рішенням для ідентифікації та управління доступом в Azure. Azure AD – це мультітенантний хмарний каталог і служба управління посвідченнями корпорації Майкрософт. Мультітенантність – це, по суті, можливість ізольовано обслуговувати користувачів з різних організацією в рамках одного сервісу. Azure AD об'єднує в собі базові служби каталогів, функції управління доступом до додатків і захист посвідчень.

Azure Cache for Redis можна назвати повністю керованим кешем для зберігання даних в пам'яті. Це спеціальна Azure служба, яка, в даній роботі, виконує функцію кешування запитів до стороннього API.

Azure Key Vault – це хмарна служба для безпечного зберігання секретів і доступу до них. В свою чергу, секрет – це все, що необхідно для управління доступом: ключі API, паролі, сертифікати або криптографічні ключі.

Azure Cosmos DB являє собою повністю керовану службу NoSQL баз даних для розробки додатків. Згідно зі специфікацію платформи Azure Cosmos DB має час відгуку менше десяти мілісекунд і рівень доступності 99,999% (згідно з угодами про рівень обслуговування). Також Cosmos DB забезпечує автоматичну миттєву масштабованість та надає API з відкритим кодом для MongoDB і Cassandra.

Вибір саме NoSQL бази даних обумовлений наступними властивостями проекту:

1. Нечіткі вимоги до даних та їх структури.
2. Проект може і буде коригуватися з часом, при цьому важлива можливість негайного початку розробки.
3. Швидкість обробки даних і масштабованість – одні з основних вимог до бази даних.

3.1.2 Клієнтська частина

У цій роботі клієнтська частина представлена одним web-додатком. Було вирішено реалізувати додаток на основі фреймворку Angular. На сьогоднішній день найпопулярнішими фреймворками для розробки клієнтських web-додатків все ще є фреймворки React, Angular та Vue. Нові клієнтські фреймворки поступово теж виходять на ринок, але перелічені фреймворки впевнено тримають лідерство.

Вибір впав на Angular через ряд суб’єктивних причин:

1. Досвід останніх років в роботі з Angular (з 6 до 11 версії).
2. Наявність TypeScript як основи фреймворку, що дає змогу уніфікувати підходи у написанні коду серверної та клієнтської частин, використовуючи принципи ООП.

3. Наявність Angular Material, що надає доступ до великої бази готових компонентів для стилізації UI.
4. Можливість дуже легко реалізувати Progressive Web Application (PWA) – такий собі гібрид сайту та нативного мобільного додатку.
5. Використання типових HTML-шаблонів, а не додаткових розширень типу JSX або JSS, що дає змогу при необхідності легко розділити бізнес логіку та представлення даних.

Все вищезгадане, а також наявність готових пакетів типу angular-leaflet для роботи з мапою та msal-angular для роботи з Microsoft Identity Platform, стали ключовими факторами при виборі фреймворку для клієнтської частини.

3.1.3 Сторонні прикладні інтерфейси

В даній роботі додатковими прикладними інтерфейсами є:

1. Graphhopper – система планування маршрутів з наявним API.
2. Air Quality Index – сторонній API для моніторингу якості повітря.

Про обидва інтерфейси вже йшла мова в розділах 1 та 2, тому тут слід тільки уточнити, що Air Quality Index API – це повністю сторонній сайт з відкритим API, який дозволяє отримувати низку корисної інформації щодо якості повітря, а Graphhopper API – це web-сервер з відкритим кодом, який можна як встановити локально та повністю відповідати за його хостинг самостійно, так і заплатити за ліцензію та використовувати платний web-сервіс Graphhopper API, надсилаючи запити до останнього через мережу Інтернет. В даному проекті було обрано перший спосіб роботи з Graphhopper – у вигляді локального web-серверу.

3.2 Особливості серверної частини

Як вже було сказано вище, прикладний програмний інтерфейс було вирішено побудувати з використанням можливостей платформи Microsoft

Azure. Зупинимось більш докладно на особливостях реалізації деяких частин системи та служб платформи Azure, які при цьому були використані.

3.2.1 Контроль доступу на основі Active Directory

Azure AD надає велику кількість переваг, як наприклад можливість створення єдиного входу (SSO) в пристрої, додатки і служби з будь-якого місця, щоб користувачі могли ефективно працювати де завгодно і коли завгодно. Як правило, наявність декількох керованих рішень для посвідчень створює проблему адміністрування не тільки для IT-фахівців, а й для користувачів, адже їм доведеться запам'ятати декілька паролів. Єдиний вхід можна включити за допомогою одного рішення для посвідчень для всіх додатків і ресурсів.

Також, в сучасному світі досить жорстко постає питання GDPR – регламенту в межах законодавства Європейського Союзу щодо захисту персональних даних усіх осіб у межах Європейського Союзу та Європейської економічної зони. Це створює додаткові проблеми зі зберігання чутливих особистих даних користувачів. Azure AD вирішує цю проблему і покладає зберігання цих даних на себе, а завдяки можливості створення AD каталогу в різних кінцях світу, нівелює також проблеми локального характеру зі зберігання даних користувачів в окремих країнах.

В даній системі було використане так зване B2C (business-to-customer) рішення [18]. Azure Active Directory B2C надає службу корпоративного управління посвідченнями для клієнтів. Користувачі отримають можливість єдиного входу в програми і API, використовуючи підтвердження соціальних мереж, підприємств або локальних облікових записів. Схема процесу авторизації представлена на рисунці 3.1

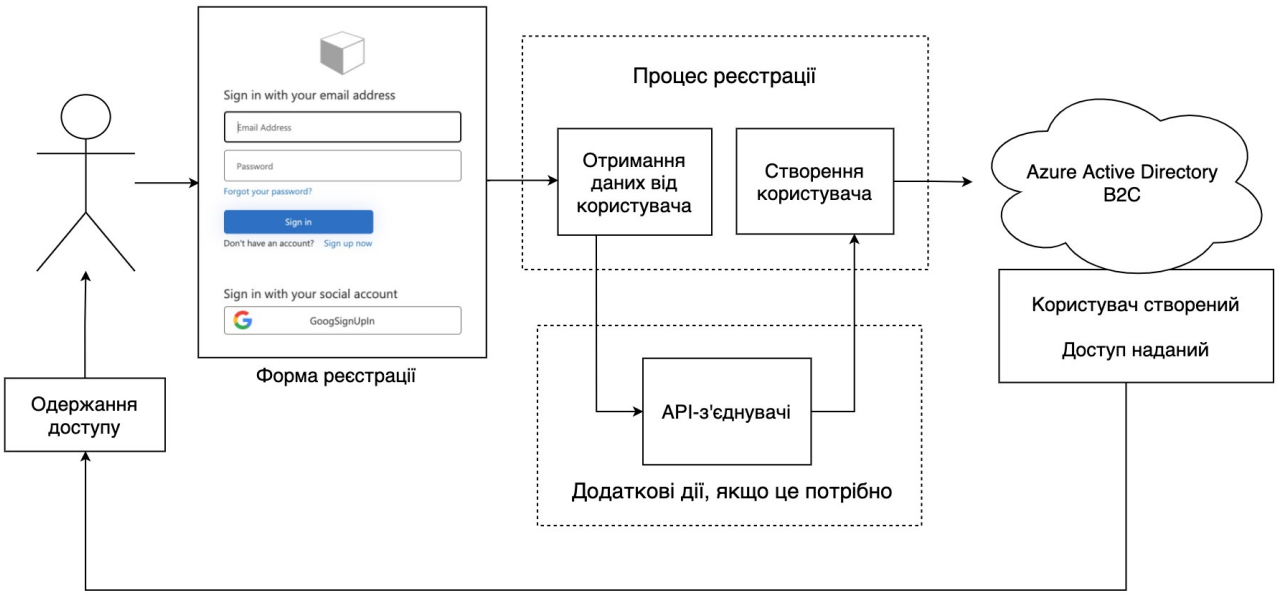


Рисунок 3.1 – Схема процесу реєстрації користувача в системі за допомогою Azure AD B2C

3.2.2 Azure B2C додаток та користувацькі потоки

Для того щоб надати користувачам можливість авторизуватися через Azure AD необхідно перш за все створити Azure B2C Application.

Щоб створити Azure B2C додаток необхідно [19]:

1. Обрати пункт “Реєстрація програм” і вибрати “Нова реєстрація.”
2. Ввести назву програми.
3. У розділі Підтримувані типи облікових записів обрати елемент Accounts in any identity provider or organizational directory (for authenticating users with user flows). Це дозволить будь-яким користувачам мати доступ до процесу авторизації в конкретному Azure AD B2C.
4. У полі URI перенаправлення обрати кінцеву точку, в яку сервер авторизації (Azure AD B2C в даному випадку) відправляє користувача після завершення взаємодії з користувачем і в яку відправляються маркер доступу або код авторизації після успішної авторизації.

5. Після завершення процесу реєстрації B2C додатка також необхідно не забути згенерувати secret key для новоствореного додатку.

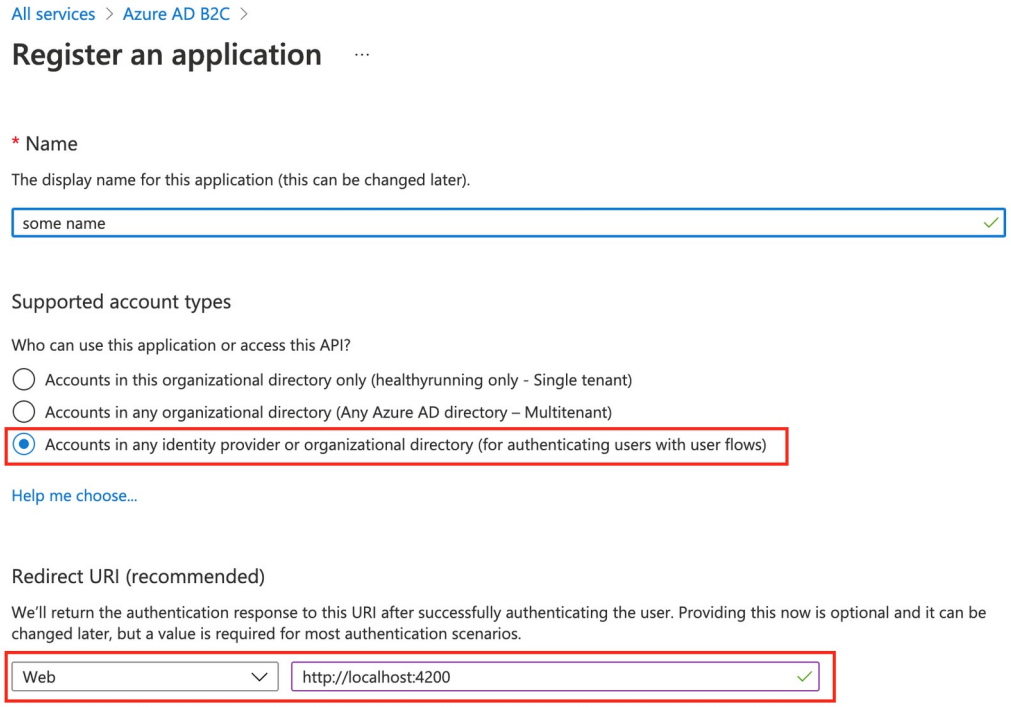


Рисунок 3.2 – Реєстрація Azure AD B2C додатку

У схожій манері створюємо user flow. Наразі, було вирішено обійтися простими стандартними user flow, та не використовувати розширені можливості.

Azure AD B2C надає просту у налаштуванні взаємодію з користувачем, яка повинна охоплювати 80% випадків. Для розширеної "ручної" конфігурації існує Identity Experience Framework [20], який надає розширені можливості конфігурації для налаштування потоку автентифікації та інтеграції зі сторонніми системами.

Спеціальні політики Azure AD B2C дозволяють настроїти потік аутентифікації за допомогою трьох основних рівнів:

1. Каталог користувачів, доступний за допомогою Microsoft Graph і який містить дані користувачів як для локальних облікових записів, так і для об'єднаних облікових записів (по суті, Azure AD)

2. Доступ до фреймворку Identity Experience Framework, який організовує по суті зв’язок між користувачами та організаціями та передає дані між ними для виконання ідентифікаційного завдання або управління доступом.
3. Служба маркерів безпеки (STS), яка видає маркери ідентифікатора, оновлює маркери та перевіряє їх для захисту ресурсів.

Після створення додатку та користувацького потоку процес інтеграції Azure AD буде майже завершено. Також Azure надає можливість модифікації стандартного user flow. Ми можемо змінити:

1. Html-шаблон та додати стилізації (проте не можемо змінити поля форми)
2. Можемо обрати які атрибути користувача ми повинні збирати в процесі реєстрації та які з них повернуться нам в процесі авторизації
3. Можемо встановити чи потрібно від користувача підтвердження email-адреси і якщо ні, то ми можемо відключити стандартний процес перевірки email-адреси і зробити це самостійно
4. Обрати можливість авторизації користувача через стороні сервіси: Facebook, Google, Github і т.д.

У підсумку ми отримаємо стандартну форму авторизації, на яку користувач буде перенаправлений у разі старту процесу авторизації/реєстрації.

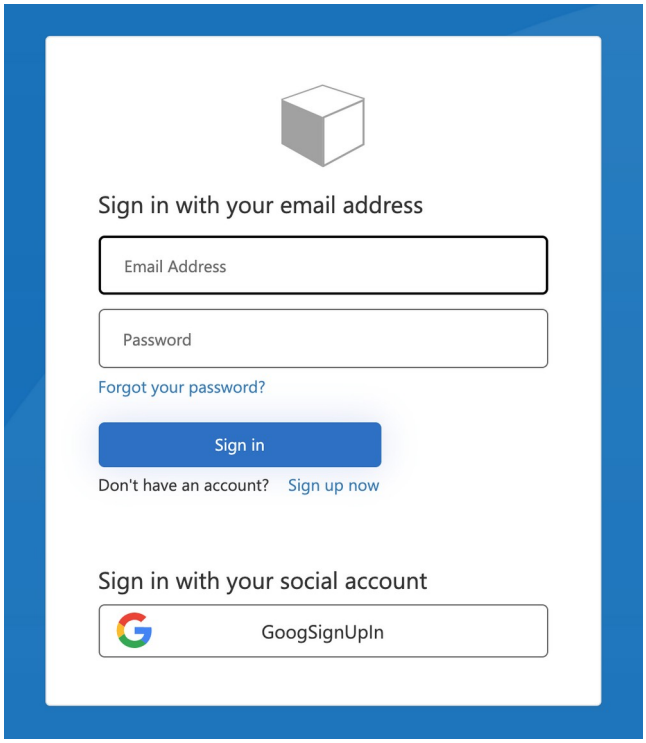


Рисунок 3.3 – Стандартна форма реєстрації через B2C додаток

3.2.3 Взаємодія з Graphhopper за допомогою Azure Functions

Програмний інтерфейс для взаємодії з фреймворком Graphhopper було вирішено побудувати на основі serverless Azure-функцій. Поняття serverless functions [16] або безсерверні функції можна вважати мікросервісами. Історично склалося так, що моноліти мають велику уніфіковану кодову базу, що вимагає повного розгортання всієї платформи для провадження будь-якого нового рішення. Безсерверні функції є автоматично масштабованими та нівелюють надлишкове розгортання коду. Провайдери хмарного хостингу називають цей продукт Functions-as-a-Service або скорочено FaaS – функція як послуга.

Платформа Microsoft Azure дозволяю розробляти Azure-функції на основі багатьох мов програмування. Для проекту було обрану мову C# та фреймворк .NET Core версії 3.1. Розробка проходила в програмному оточенні VSCode, а створення та завантаження функцій в Azure сховище – допомогою розширень Azure Functions, Azure Account та Azure Resources.

Для того щоб створити нову функцію прямо на Azure платформі, використовуючи додаток VS Code необхідно спершу обрати пункт “Створити нову Azure-функцію” (рисунок 1.2):

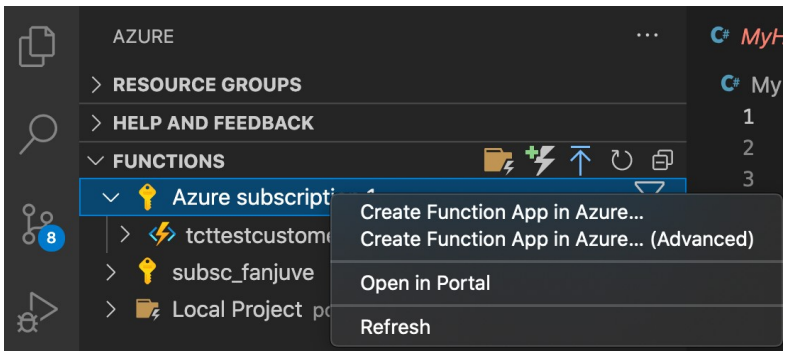


Рисунок 3.4 – Початок створення нового додатку Azure-функції

Потім – виконати наступні кроки, обравши унікальне ім’я функції, мову програмування (наприклад, C#) та версію фреймворку (в моєму випадку .NET Core 3.1). Далі необхідно обрати регіон, в якому будуть розташовані необхідні вам ресурси. Після чого дочекатися створення функції на платформі Azure.

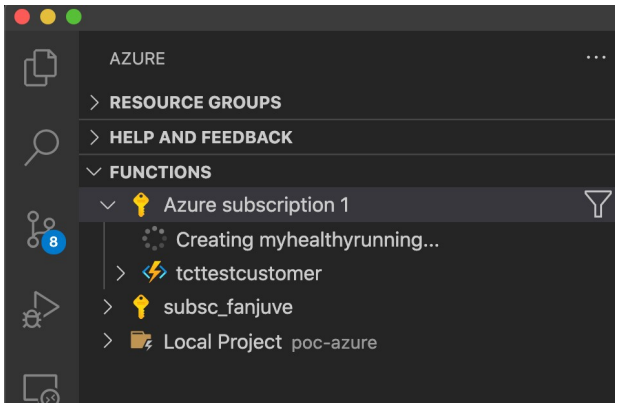


Рисунок 3.5 – Створення нового додатку Azure-функції

Коли процес створення функції буде завершено, на порталі Azure стане доступна новостворена функція, та допоміжні ресурси, які були також створені в процесі створення функції (рисунок 3.6).

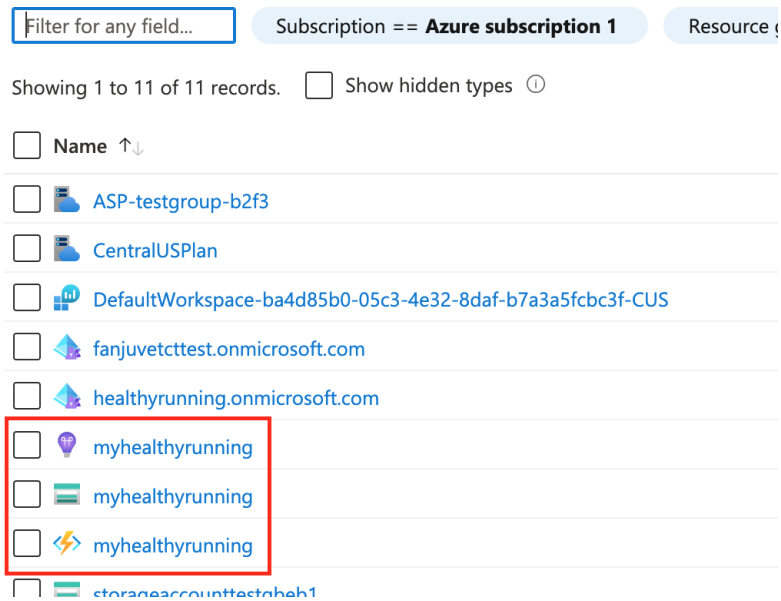


Рисунок 3.6 – Новостворений додаток Azure-функції на порталі

Далі за допомогою VS Code була створена сам функція. Спочатку необхідно обрати опцію “Створити нову функцію” та обрати тип функції. В моєму випадку це HttpTrigger, оскільки функція буде являти собою простий прикладний інтерфейс (рисунок 3.7).

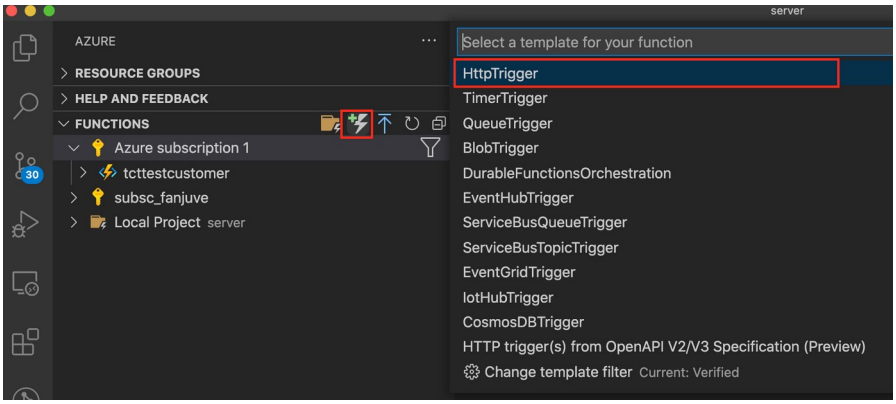


Рисунок 3.7 – Створення кодової бази Azure-функції

Далі, рухаючись за покроково обрати знову ж таки мову програмування, версію фреймворку, ім’я для базового класу та права доступу. Останні можуть бути трьох типів: Admin, Function та Anonymous. Вони необхідні для того, щоб

унеможливити доступ до функції сторонніх осіб, які не мають в своєму розпорядженні ключа. Сам запит при цьому буде мати вигляд:

```
https://<APP_NAME>.azurewebsites.net/api/<FUNCTION_NAME>?code=<API_KEY>
```

Для того щоб завантажити створену кодову базу функції на портал Azure необхідно вибрати опцію “Завантажити до додатку функції”, а потім обрати підписку та додаток функції, який був створений до цього. Через декілька хвилин функція буде завантажена у додаток функції на порталі Azure. Перевірити функцію можна за допомогою додатку типу Postman, відправивши запит до функції.

Кодова база Azure додатку функції складається з двох функцій типу HttpTrigger (GetEcoSources та BuildRoute), а також з допоміжних файлів роботи зі сторонніми клієнтами.

1. GetEcoSources:

- Доступ відбувається за ендпоінтом [POST] /api/getEcoSources
- Робить запит до API незалежної агенції з моніторингу якості повітря.
- Кешує дані запиту в Redis або повертає наявні дані з нього.

2. BuildRoute:

- Доступ відбувається за ендпоінтом [POST] /api/buildRoute
- Передає дані про обрані користувачем точки маршруту та дані про забрудненість повітря в конкретній місцевості у вигляді “заблокованих зон” до локального фреймворку Graphhopper
- Отримує дані стосовно побудованого маршруту від Graphhopper та передає ці дані на клієнт

При цьому, для зменшення ціни на етапі розробки була використана програмна консольна утиліта ngrok. Ngrok робить можливим проксування локального оточення через систему proxy-серверів та дає можливість доступу до конкретного локального оточення з інших місць.

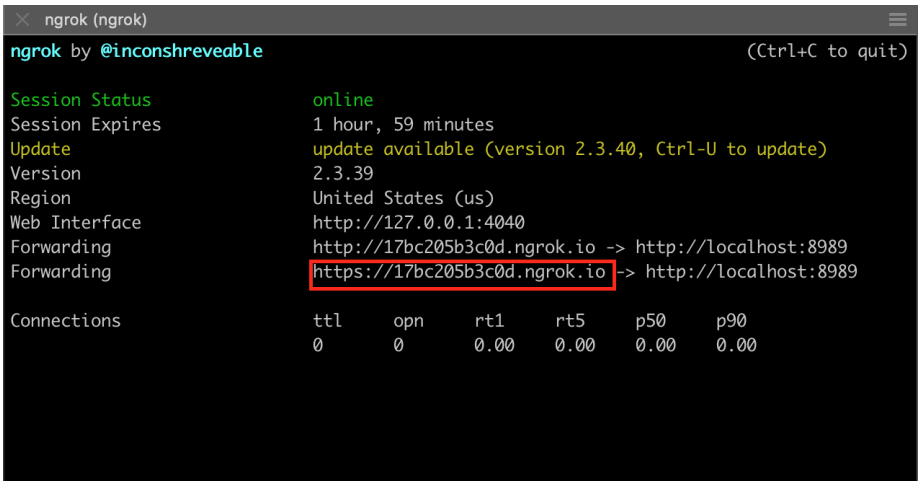


Рисунок 3.8 – Проксування локального серверу Graphhopper через ngrok

Це дає змогу використати його для проксування локального серверу Graphhopper та використання цієї згенерованої адреси під час роботи Azure-функції безпосередньо на порталі Azure, що дуже спрощує розробку.

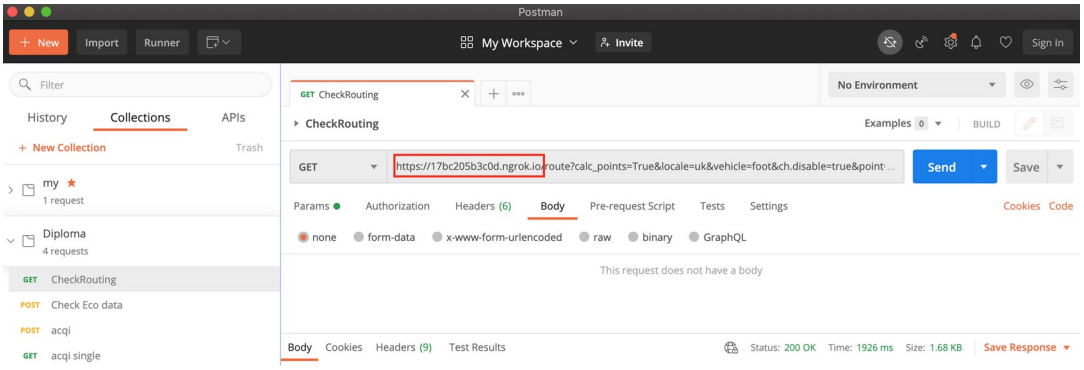


Рисунок 3.9 – Запит до локального серверу Graphhopper через проксі

3.2.4 Використання API-з’єднувачів

На структурній схемі можна побачити частину, яка відповідає за деякі додаткові дії, які можуть бути зроблені під час процесу авторизації. Це відбувається за допомогою так званих API-з’єднувачів [17]. Суть використання останніх зводиться до того, що під час процесу реєстрації та авторизації користувача через Azure Active Directory існує можливість перервати цей

процес, шляхом додавання додаткових запитів. Azure має наразі суворе обмеження з цього приводу і тому дозволяє робити запити лише через HTTPS-протокол.

В даній роботі програмно API-з'єднувач ніяк не відрізняється від стандартної Azure-функції та має на меті додавання деякої додаткової (не чутливої) інформації у базу даних Cosmos DB під час процесу реєстрації користувача в системі. В цілому ж, найчастіше API-з'єднувач можна використати для:

- Перевірки на відповідність некоректним чи недопустимим даними користувача. Наприклад, можна перевірити дані користувача з існуючими в зовнішньому сховищі даними або в списку заборонених значень. По суті, таким чином можна заблокувати профіль користувача "локально", без необхідності робити це в Azure AD.
- Перезапису атрибутів користувача. Наприклад, якщо користувач вводить case sensitive дані тільки малими або великими літерами, можна змінити цей атрибут.
- Перевірки особистості з використанням служби перевірки особистості, щоб додати додатковий рівень безпеки для прийняття рішень про створення облікових записів.
- Запуску додаткової бізнес-логіки та подій. Наприклад, можна активувати відправки push-повідомлень, поновлення корпоративних баз даних, управління дозволами, аудиту баз даних, настроїти самим процес підтвердження email-адреси, тощо.

3.2.5 Служба Azure Cosmos DB для зберігання даних

Коли мова заходить про зберігання даних, то постає питання забезпечення надійності, швидкості, безпеки даних, тощо. Дуже добре себе зарекомендували реляційні SQL бази даних, проте NoSQL бази не відстають від

них останнім часом, особливо з появою та все більшим розповсюдженням хмарних технологій почали потихеньку витісняти перших з ринку.

По суті, реляційні БД зберігають структуровані дані, які зазвичай представляють об'єкти реального світу. Скажімо, це можуть бути відомості про тварин, людей або про вміст кошика для товарів в магазині, з групуванням по таблицях. Як правило, формат даних заданий на етапі проектування сховища.

При цьому, нереляційні БД влаштовані інакше. Для прикладу, документо-орієнтовані бази зберігають інформацію у вигляді ієрархічних структур даних. В такому випадку мова може йти про об'єкт з довільним набором атрибутів, строкове представлення об’єкту у вигляді json-документу і т.д. Тобто те, що в реляційній базі даних буде розбито на декілька взаємопов'язаних таблиць, в нереляційних може зберігатися у вигляді цілісної сутності, кажучи грубо “в одній комірці даних”.

NoSQL бази даних мають певні можливості, які зробили їх достатньо популярними:

1. Зберігання великих обсягів неструктурованої інформації. NoSQL база не накладає обмежень на типи збережених даних, а при потребі в процесі роботи можна додавати нові типи даних.
2. Використання хмарних сховищ і обчислень. Хмарні сховища вимагають, щоб дані можна було легко розподілити між кількома серверами для забезпечення масштабування.
3. Швидка розробка. Якщо при розробці системи використовуються agile-методи, застосування реляційної БД здатне уповільнити роботу. NoSQL бази даних не потребують такого ж обсягу підготовчих процедур, які зазвичай потрібні для реляційних баз.

В якості бази даних було вирішено скористатися послугами Azure служби Cosmos Database.

Azure Cosmos DB [17] – це швидка доступна база даних NoSQL з можливістю автоматичного і швидкого масштабування, а також з API з відкритим кодом для MongoDB і Cassandra.

Основними перевагами Azure Cosmos DB при розробці застосунків є [17]:

1. Глибока інтеграція з ключовими службами Azure, що часто використовуються в сучасній розробці додатків у хмарному середовищі, включаючи Azure-функції, служба Azure Kubernetes, Центр Інтернету речей і багато інших.
2. Можливість обрати один з декількох API бази даних, включаючи власний API Core (SQL), або API для MongoDB, API Cassandra, API Gremlin і API таблиць.
3. Підтримка .NET, Java, Node.js і Python та інших драйверів для роботи з базою даних.
4. Потік змін спрощує відстеження та адміністрування змін в контейнерах баз даних, а також створює певні події-тригери, за допомогою Azure-функцій.
5. Служба для обробки даних без схеми Azure Cosmos DB автоматично індексує всі дані, незалежно від їх моделі, забезпечуючи достатньо швидку обробку запитів.
6. Cosmos DB має безкоштовний план (з обмеженням за кількістю даних та запитів).

В свою чергу, модель даних системи,що розроблювалась, має декілька особливостей:

1. Модель даних системи має дуже простий вигляд:
   - Колекція User, де зберігається дані користувачів.
   - Колекція RoutePath – тут зберігаються вже сформовані маршрути.
   - Ключі API, конфіги та інші secret-ключі.

- Кеш-дані запитів.

2. Неможливо чітко сформулювати критерії даних.
3. Дані приходять з середовища, про яке на момент створення архітектури та вибору бази даних не було майже нічого відомо.
4. Дані мають нечітку структуру та схильні до неконтрольованих змін, оскільки представляють собою дані від сторонніх сервісів.

Саме перелічені вище причини обумовили вибір NoSQL бази даних в якості основної (у кооперації з Azure Active Directory сховищем та Key Vault службою), оскільки використання реляційної бази даних при такій моделі могло призвести до ускладнення останньої надмірними реляційними зв'язками.

Використання Azure Cosmos DB під час розробки на порталі Microsoft Azure зводиться до кількох наступних дій.

По-перше, необхідно знайти службу Azure Cosmos DB, обрати необхідні параметри у конфігурації та додати сервіс на портал.

Спочатку слід обрати API, яке буде використане для встановлення зв'язку з базою даних. Тут в якості API було обрано MongoDB.

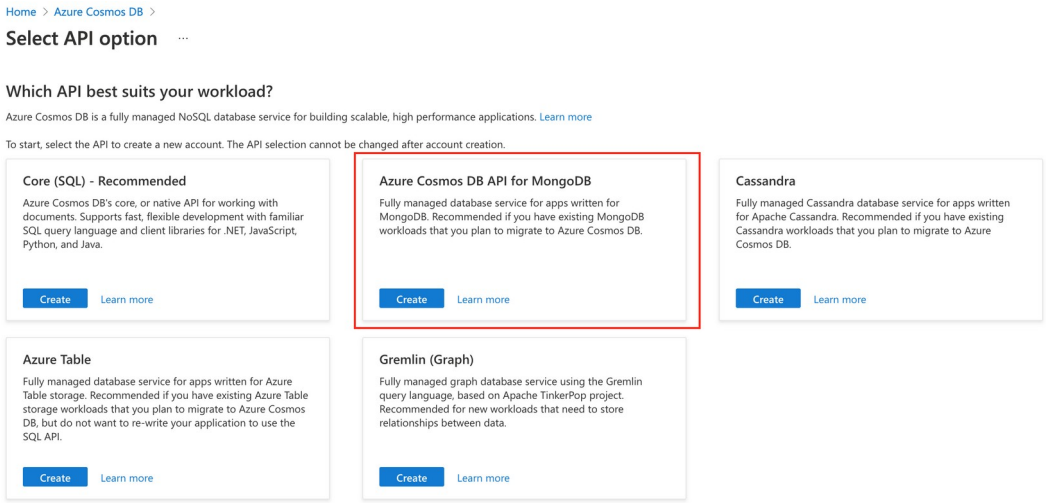


Рисунок 3.10 – Вибір API для встановлення зв'язку з базою даних

Також важливим моментом є те, що Microsoft надає багато можливостей для резервного копіювання даних. Їх політики у цьому напрямку дуже гнучкі, а можливості – величезні.

Наприклад, на наступному кроці можна побачити, що окрім стандартних “скільки буде зберігатися резервна копія даних”, “як часто потрібно робити резервну копію даних бази” та кількості копій, присутній ще один важливий пункт – backup storage redundancy або надлишкове резервне копіювання.

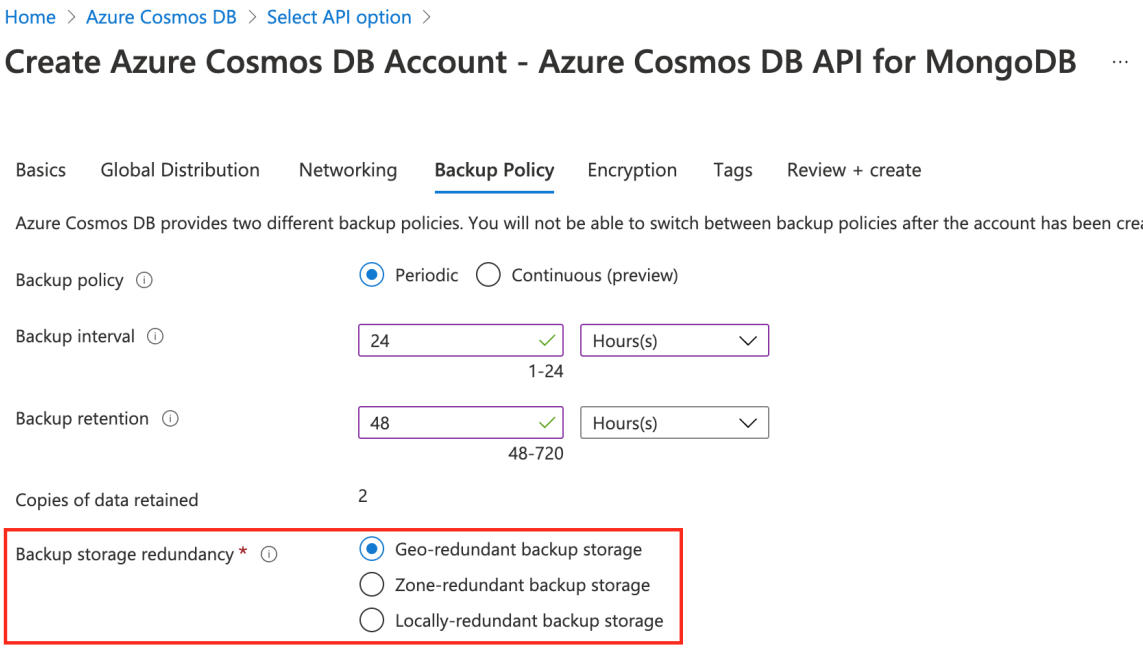


Рисунок 3.11 – Автоматичне геонадлишкове резервне копіювання

Автоматичне геонадлишкове резервне копіювання надає можливість зберігання даних в “геонадлишкових” великих двійкових об'єктах сховища, які реплікуються в парний регіон. Це допомагає захиститися від простоїв, що впливають на сховище резервних копій в основному регіоні, і дозволяє відновити сервер в інший регіон у разі аварії. В свою чергу, опція локальної надлишковості буде забезпечувати резервне копіювання бази даних у тому самому регіоні та навіть у тому самому датацентрі.

Остаточно, після деплою служби Cosmos DB, в налаштуваннях служби можна буде побачити MongoDB строку-з'єднання (connection-string), яку необхідно буде використати для підключення до бази даних.

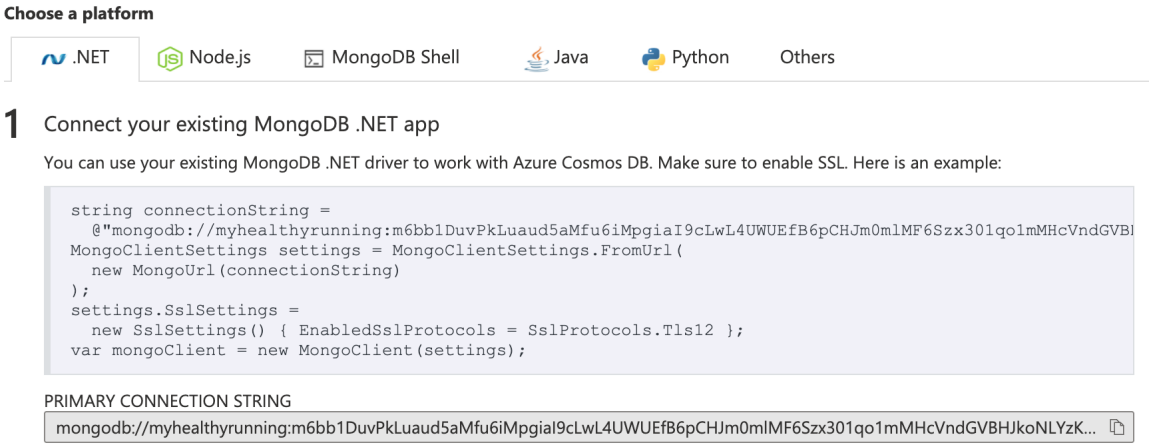


Рисунок 3.12 – MongoDB строка-з'єднання з базою даних

На другому кроці необхідно створити у коді файли-repository для відповідних колекцій даних та підключитися до бази даних за допомогою строки-з'єднання.

```
public class UsersRepository {

    1 reference
    private const string USERS_COLLECTION = "Users";
    6 references
    private readonly IMongoCollection<UserEntity> _usersCollection;
    2 references
    private readonly ILogger _logger;

    0 references
    public UsersRepository(ILogger<UsersRepository> logger) {
        _logger = logger;
        var client = new MongoClient(Environment.GetEnvironmentVariable("DB_CONNECTION"));
        var database = client.GetDatabase(Environment.GetEnvironmentVariable("DB_NAME"));
        _usersCollection = database.GetCollection<UserEntity>(USERS_COLLECTION);
        AddUniqueIndex("emailHash");
    }
```

Рисунок 3.13 – Підключення до MongoDB та створення колекції Users

При цьому DB_CONNECTION та DB_NAME представляють собою ключі, які були додані до змінних конкретного оточення. Самі ключі зберігаються в Azure Key Vault службі, про яку піде мова трохи згодом.

Третій крок – створення класів-сутності UserEntity та RoutePathEntity з використанням ключових атрибутів моделі BsonId, BsonRequired та ін. однойменної бібліотеки. Після цього ми можемо вільно робити запити до бази даних.

```
public class UserEntity {
  [BsonId]
  [BsonRepresentation(BsonType.ObjectId), BsonRequired]
  0 references
  public string id { get; set; }

  [BsonElement("emailHash"), BsonRequired]
  0 references
  public string emailHash { get; set; }
}
```

Рисунок 3.14 – Використання Mongo атрибутів для опису моделі даних

Загалом, під час розробки системи та її взаємодії з даними, було вирішено використовувати шаблон сховища-служби (Repository-Service). Даний шаблон чудовий для ситуацій, коли потрібно запитувати дані зі складного сховища даних або потрібне чітко визначене розділення бізнес логіки між тим, що відбувається для окремих моделей, та комбінаціями моделей.

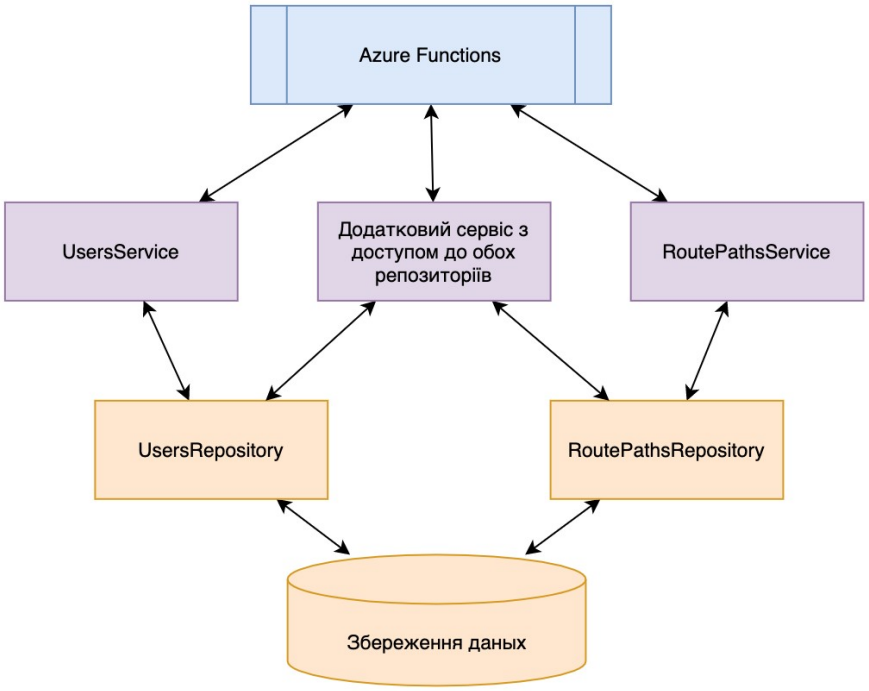


Рисунок 3.15 – Використання шаблону Repository-Service

Єдиними суттєвим недоліком даного шаблону є велика кількість, так би мовити, зайвого коду. Наприклад, під час створення даної системи з його досить простою моделлю даних необхідно було створити:

- клас UsersRepository
- клас UsersService
- клас RoutesPathRepository
- клас RoutesPathService

При цьому UsersService та RoutesPathService, по суті, виконують “прохідні” запити до відповідних сховищ. Це все може призвести до великої кількості надмірного коду, проте, якщо ми говоримо про масштабованість у майбутньому, такий підхід, з чітким розділенням моделі даних та сервісів, які з нею працюють, може стати плюсом.

3.2.6 Azure Key Vault та чутливі дані

Також говорячи про зберігання даних не можна оминути стороною службу Azure Key Vault, яка також використовується в цій роботі. Як було вказано в розділі 3.1, Azure Key Vault являє собою сховище для зберігання "секретів" – ключів від API, строк-з'єднання з базами даних та інших чутливих даних, потенційний доступ зловмисника до яких зможе призвести до нанесення невиправної шкоди системі та бізнесу в цілому.

Azure надає можливість використовувати для зберігання таких даних спеціальну службу Key Vault [17]. По суті, це просте сховище з даними у вигляді пари "ключ-значення". При цьому Azure Key Vault використовує апаратні модулі безпеки компанії Thales з реалізацією технології Hierarchical Storage Management (HSM). Особливість HSMs в тому, що вони не надають ключі напряму. Спочатку користувач створює або імпортує ключ в HSM, а потім, коли користувач передає в HSM певні дані, HSM виконує криптографічні операції з цими даними – шифрування, дешифрування, хешування і т.д. Подібні апаратні пристрої є досить дорогими, а Azure Key Vault дозволяє використовувати цей вид захист критичної інформації за невелику ціну. Це і є основною перевагою використання сховища ключів Azure.

Іншими словами, якщо віртуальна машина, була скомпрометована і при цьому на ній зберігалися ключі шифрування, то зловмисник отримав би ці ключі, що призвело б до набагато більших втрат, ніж злом самої віртуальної машини. Якщо ж на віртуальній машині є тільки ідентифікатор користувача і secret, то зловмисник має тільки облікові дані, але не самі ключі. І навіть з цими обліковими даними зловмисник не зможе отримати ключі від HSM. Це означає, що зловмиснику знадобиться виконати певні криптографічні операції з цими вкраденими обліковими даними, а це в свою чергу буде давати додатковий час адміністраторам системи, щоб розпізнаєте атаку та анулювати облікові дані.

В даній роботі в Azure Key Vault зберігаються наступні чутливі параметри системи:

- DB_CONNECTION – строка-з’єднання з базою даних MongoDB.
- DB_NAME – ім’я бази даних.
- ADB2C_TENANT_ID – ідентифікатор B2C Active Directory каталогу.
- ADB2C_TENANT_NAME – ім’я B2C Active Directory каталогу.
- ADB2C_APP_ID та ADB2C_APP_SECRET – ідентифікатор та секрет-ключ B2C Active Directory додатку для створення користувацьких потоків.
- ADB2C_EXTENSION_APP_ID – ідентифікатор B2C Active Directory додатку для роботи з додатковими blob-даними.
- ADB2C_USER_FLOW_SUSI – назва/ідентифікатор користувацького потоку для реєстрації/авторизації.
- ADB2C_USER_FLOW_RESET_PSWD – назва/ідентифікатор користувацького потоку для відновлення паролю.

3.2.7 Налаштування Azure Cache for Redis для оптимізації запитів

Redis (REmote DIctionary Server) – це нереляційна високопродуктивна СУБД. Redis зберігає всі дані в пам'яті, доступ до яких здійснюється по ключу. Опціонально копія даних може зберігатися на диску. Цей підхід забезпечує продуктивність, в десятки разів перевищує продуктивність реляційних СУБД, а також спрощує секціонування (шардінг) даних.

Платформа Microsoft Azure дає можливість підняти свій Redis додаток під назвою Azure Cache for Redis.

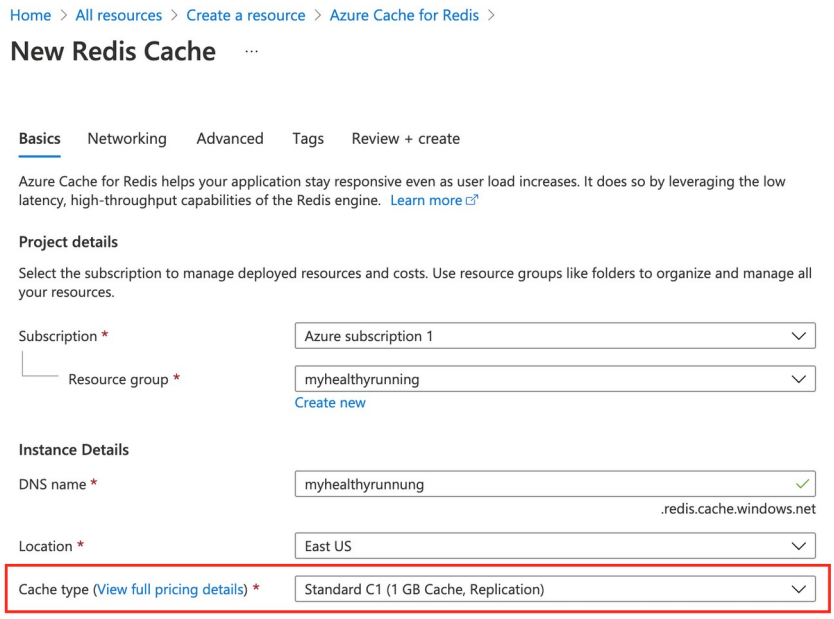


Рисунок 3.16 Створення Azure Cache for Redis

В цілому, область використання Redis це:

- Сховище сесій і профілів користувачів.
- Сервер черг.
- Кешування даних з можливістю відновлення кешу з диска.
- Місце для зберігання кількості користувачів онлайн, кодів капчі, різних прапорів, доповнень пошукових запитів.
- Система управління базами даних для невеликих додатків.
- Сховище проміжних результатів обчислень при обробці великих обсягів даних.

Конкретно в цій роботі Redis використовується для кешування запитів до serverless функції. Взагалі, принцип serverless означає на практиці “скільки запитав – стільки й заплатив”. Часті запити до Azure-функції можуть коштувати дорого. А оскільки в даному проекту Azure-функція по суті являє собою proxy-функцію для запиту до стороннього API і отримання даних щодо якості повітря, то такі запити можуть бути доволі частими. Тому було прийнято рішення зробити кешування даних через Redis. В такому випадку користувач

отримує дані з певного сектору міста за геоданими, після чого дані кешуються у Redis та у випадку повторних частих запитів даних цієї ж місцевості – повертаються з Redis, що дає змогу зменшити час надання відповіді до клієнтської частини з 1.5 секунди до 20-25 мс.

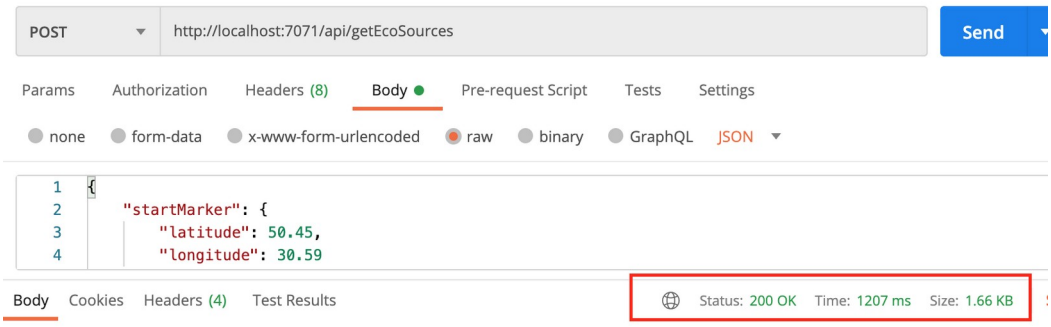


Рисунок 3.17 – Перший запит. Отримаємо дані від стороннього API

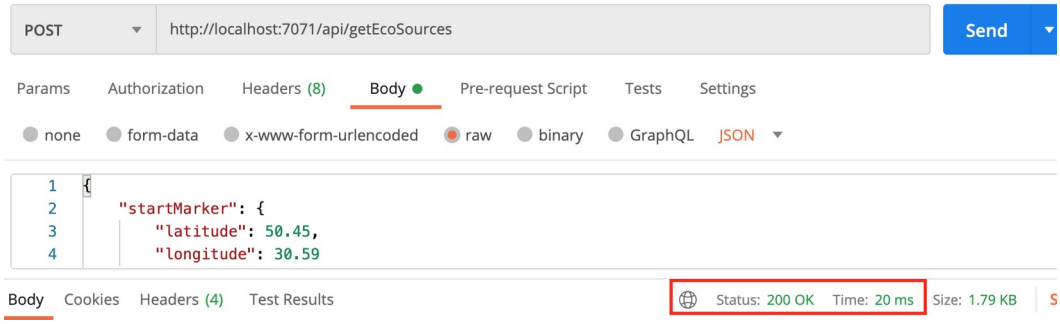


Рисунок 3.18 – Наступний запит. Дані повертаються з Redis

3.3 Особливості клієнтської частини

В цьому розділі зупинимось більш докладно на особливостях реалізації клієнтської частини. Схема алгоритму взаємодії користувача з системою представлена у додатку 3.

3.3.1 Опис клієнтського інтерфейсу

Інтерфейс клієнтської частини складається з двох основних частин: головна сторінка з привітанням та закликом до дії та сторінка з мапою для відображення якості повітря в конкретних районах та побудованого маршруту для бігу.

Існує багато бібліотек, які дозволяють працювати з мапами та геоданими. На практиці найчастіше використовуються Google Maps, OpenStreetMap та Mapbox. Кожна з цих бібліотек є повноцінним картографічним сервісом. Оскільки в цій роботі були використані osm-дані з порталу Geofabrik, то було вирішено також використати OpenStreetMap для відображення отриманих даних та згенерованих маршрутів.

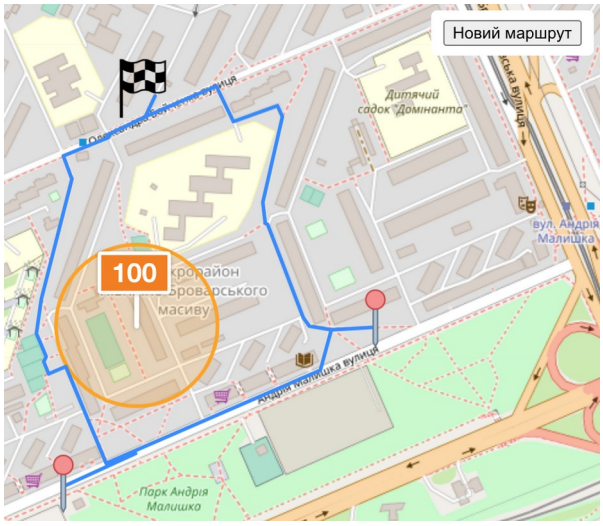
Для безпосередньої роботи з мапою було обрано бібліотеку leaflet, а саме npm пакет ngx-leaflet. Leaflet – це JavaScript API для створення інтерактивних карт в мережі Інтернет. Він може інтегруватися з безліччю інших джерел, таких як OpenStreetMap, Mapbox, Google Maps, тощо. Основна задача Leaflet – уніфікація роботи з різними картографічними сервісами та приведення їх від часткового вигляду до загального.

Алгоритм взаємодії з мапою передбачає наступні кроки:

1. Користувач, за допомогою маркера, обирає на мапі початкову точку маршруту, яка також є і кінцевою точкою маршруту.
2. Користувач обирає ще стільки допоміжних точок, скільки потрібно, але не менше однієї. Це потрібно для того, щоб побудувати круговий маршрут для бігу, оскільки для подібного маршруту необхідно не менше двох точок.

Рисунки 3.19 – Вибір трьох точок на мапі

3. Користувач завершує створення маршруту, шляхом натискання на кнопку “Готово”.
4. Враховуючи дані щодо якості повітря між обраними маркерами (waypoint-ами) будується маршрут в тій послідовності, в якій маркери були обрані.



Рисунки 3.20 – Побудова маршруту, огинаючи загазовану ділянку

The World Air Quality Index Project надає наступні значення:

- met.h – вологість повітря
- met.p – атмосферний тиск
- met.t – температура повітря
- co2 – кількість оксиду вуглецю
- pm10, pm2.5 – дрібнодисперсні частинки – значення забруднювачів, до складу яких входять як тверді мікрочастинки, так і дрібні краплі рідини.
- pm1 – молекули газів

Для того, щоб спростити розрахунки було вирішено, що небезпечна ділянка з завищеним значення AQI має форму кола з центром у точці, координати якої були отримані від сервісу The World Air Quality Index Project,

та радіусом, в залежності від виду забруднюючої речовини, значення якої виходить за межі норми.

За даними ВООЗ дрібнодисперсні частинки pm10 та pm2.5 , а особливо останні, мають найбільший вплив на здоров'я людини. При цьому pm10 частинки є більш важкими і тому як правило осідають в парі сотень метрів від джерела забруднення, в той час як pm2.5 частинки є більш легкими і тому більш небезпечними, адже можуть переноситись на відстані до 1 кілометру (і навіть до 2 кілометрів і більше у вітряну погоду при сильному забрудненні) [21].

В даній роботі для спрощення візуалізації був обраний радіус у 500 метрів від точки вимірювання починаючи з помаранчевого рівня загрози та до 3км при небезпечному рівні забруднення.

3.3.2 Взаємодія з Azure Active Directory за допомогою MSAL

Процес авторизації на клієнтській стороні відбувається за допомогою бібліотеки Microsoft Authentication Library (MSAL) [22]. А конкретно, за допомогою npm пакету @azure/msal-angular [23].

Алгоритм взаємодії користувача з клієнтською частиною відбувається наступним чином:

1. Користувач заходить на сайт і бачить головну сторінку сайту
2. Користувач повинен авторизуватися для подальшої роботи з сайтом.
3. MSALGuard – це клас бібліотеки msal-angular, який відповідає за можливість доступу користувача в ту чи іншу частину сайту.
4. MSALService – клас, що відповідає за запити до Azure AD.
5. Під час переходу на закриту для гостей частину сайту, відбувається перевірка наявності id_token параметру в Local Storage браузера, і якщо його немає, то відбувається виклик функції:

```
loginFunction() {
    MSALSerivce.loginPopup()
```

```
        }
```

що є простим перенаправлення на [APP_NAME].microsoft.co та початком виконання потоку реєстрації/авторизації користувача (Sign Up user flow).

6. Коли користувач успішно авторизувався або зареєструвався, відбувається перенаправлення назад, на вказане в B2C-додатку посилання.
7. Після цього необхідні параметри заносяться в Local Storage браузера і надалі не будуть запитані протягом деякого проміжку часу. Те, як це виглядає у браузері можна побачити на рисунці 3.12.
8. Microsoft надає можливість перевірити актуальність створеного id_token параметру за допомогою web-додатку jwt.ms. Це проста web-сторінка, яка реалізує декодування id_token, що є дуже корисним при тестуванні процесу реєстрації/авторизації користувача (рисунок 3.13).

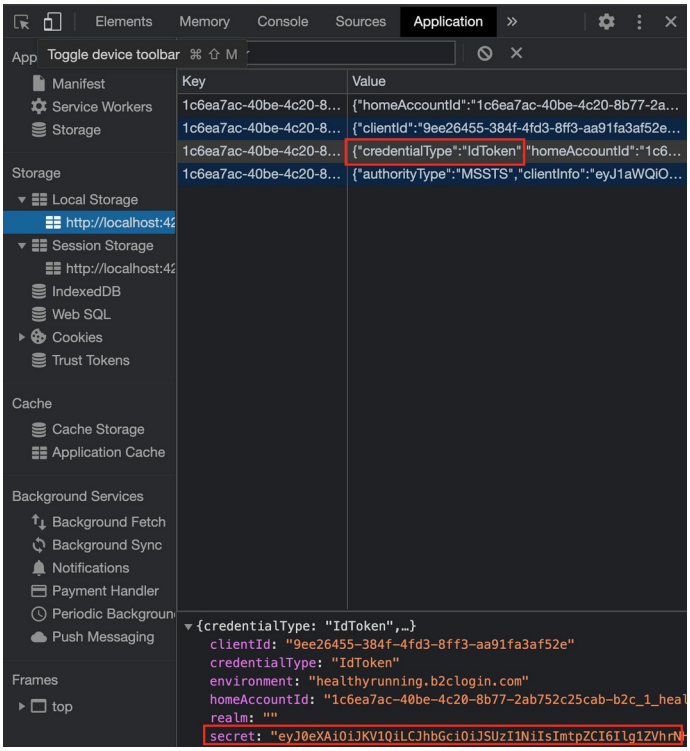


Рисунок 3.21 – IdToken у LocalStorage браузера після повернення на сайт

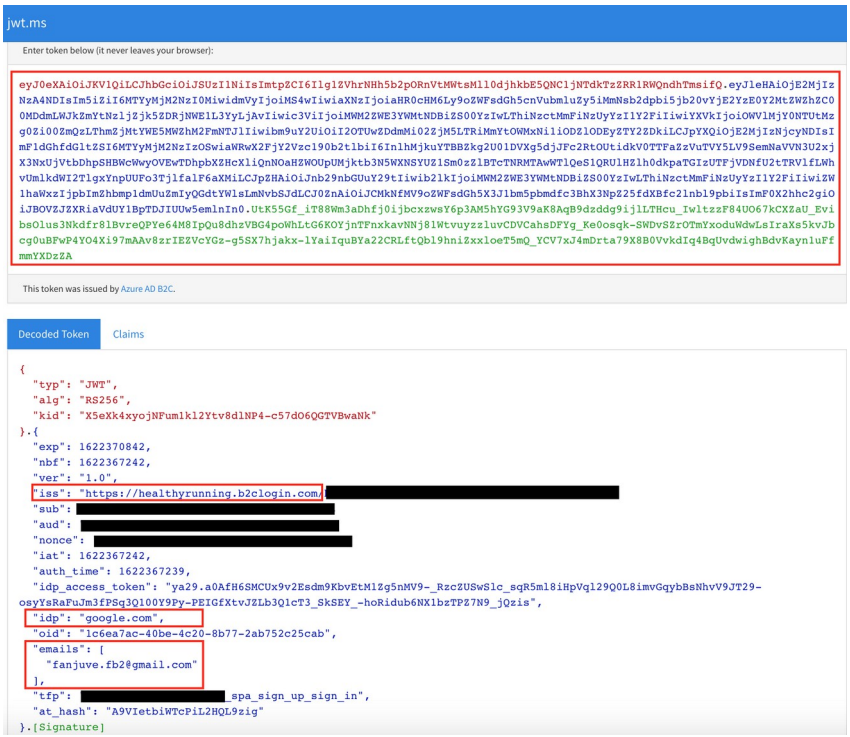


Рисунок 3.22 – Перевірка параметру IdToken за допомогою jwt.ms

ВИСНОВКИ ДО РОЗДІЛУ 3

Суть третього розділу полягала в створенні необхідних Azure додатків, які будуть використані в процесі авторизації користувачів та для отримання даних від сторонніх сервісів: Graphhopper та Air Quality Index API. За результатами роботи були створені Azure B2C додаток, декілька Azure B2C користувацьких потоки для реєстрації та авторизації користувача, зміни паролю, тощо, а також реалізований процес клієнтської авторизації.

Результатом цього розділу є розгорнуті локально, а також завантажені на портал Azure-функції для взаємодії зі сторонніми сервісами для отримання даних стосовно рівня забруднення повітря та планування маршруту та реалізований програмний інтерфейс також у вигляді Azure-функції для взаємодії з інтерфейсом серверної частини Graphhopper. Також було розгорнуто Azure Redis для оптимізації запитів до Graphhopper.

З клієнтської сторони був реалізований додаток, який використовується для візуалізації отриманих від сторонніх сервісів та фреймворку Graphhopper даних, авторизації користувачів в системі через клієнтську бібліотеку MSAL та для надання користувачу безпосередньої можливості взаємодії з системою.

ЗАГАЛЬНІ ВИСНОВКИ

В даній роботі була спроектована та реалізована система збору даних щодо якості повітря та планування безпечного маршруту для бігу та занять спортом на свіжому повітрі на основі цих даних.

В роботі було використано інформацію з сайту The World Air Quality Index Project, який надає достатньої точну та своєчасну інформацію стосовно ситуації зі станом повітря по всьому світу.

Також був проведений аналіз декількох фреймворків планування маршруту та обраний один – Graphhopper, який відповідає більшості шуканих критеріїв. Даний фреймворк можна доволі швидко розгортати, його просто налаштувати, в ньому присутні вбудоване API та хороша документація, також існує можливість динамічно вносити зміни в існуючу OSM модель даних та створювати “мертві зони”, через які маршрут проходити не повинен.

Прикладний програмний інтерфейс для взаємодії зі сторонніми сервісами та з клієнтським додатком був реалізований на основі платформи Microsoft Azure. Були використані такі сервіси Azure як: Azure Functions, Azure B2C, Azure Cache for Redis, тощо.

З точки зору клієнтської частини був розроблений web-додаток та реалізована клієнтська частина процесу авторизації за допомогою бібліотеки MSAL та візуалізація даних, отриманих від Graphhoper та прикладного інтерфейсу Air Quality Index.

Як результат, було побудовано систему, при взаємодії з якою користувач може як отримати інформацію про якість повітря в конкретний момент часу, так і спланувати маршрут для пробіжки або інших занять спортом на, справді, свіжому повітрі.

З наявних мінусів такої реалізації можна назвати те, що типовий фреймворк для планування маршруту виявився не на 100% відповідне рішенням, оскільки головна ідея подібних фреймворків – надання користувачу

можливості якогомога швидко попасти “з пункту А до пункту Б”. При цьому при побудові маршруту для бігу найкоротший шлях не є самою ціллю, а планування найчастіше відбувається у манері “з точки А у точку А” з вказанням кілометражу (наприклад, через 5 кілометрів) або часу при плюс-мінус відомій швидкості (наприклад, через 30 хвилин зі швидкістю 10-12км/год).

Проте наявні фреймворки планування маршруту не дають безпосередньої можливості отримати подібні дані. В даній роботі цей мінус був зменшений за рахунок надання користувачу можливості самостійно обирати ключові точки (waypoints) при складанні маршруту і, таким чином, створювати потрібний круговий маршрут, оминаючи всі небезпечні для здоров’я ділянки.

СПИСОК ЛІТЕРАТУРИ

1. Сайт BBC News Україна. “Даже легкое загрязнение воздуха вредит сердцу. Как уберечься от этого?” [Електронний ресурс] — Режим доступу: https://www.bbc.com/ukrainian/features-russian-45056218
2. Офіційний сайт ВООЗ. “Качество атмосферного воздуха и здоровье”. [Електронний ресурс] — Режим доступу: https://www.who.int/ru/news-room/fact-sheets/detail/ambient-(outdoor)-air-quality-and-health
3. Частицы PM2.5: что это, откуда и почему об этом все говорят. [Електронний ресурс] — Режим доступу: https://habr.com/ru/company/tion/blog/396111/
4. Roy J. Shepard. “Athletic performance and urban air pollution”. [Електронний ресурс] — Режим доступу: http://europepmc.org/backend/ptpmcrender.fcgi?accid=PMC1483256&blobtype=pdf
5. Giuseppe Lippi, Gian Cesare Guidi, Nicola Maffulli. “Air Pollution and Sports Performance in Beijing”. [Електронний ресурс] — Режим доступу: https://www.researchgate.net/publication/5339128_Air_Pollution_and_Sports_Performance_in_Beijing
6. Fu, Shihe and Guo, Mengmeng. “Running with a Mask? The Effect of Air Pollution on Marathon Runners’ Performance”. Southwestern University of Finance and Economics. 2017 [Електронний ресурс] — Режим доступу: https://mpra.ub.uni-muenchen.de/79473/1/MPRA_paper_79473.pdf
7. S. Elavsky, V. Jandačková, L. Knapová, V. Vašendová, M. Sebera, etc. “Physical activity in an air-polluted environment: behavioral, psychological and neuroimaging protocol for a prospective cohort study”. [Електронний ресурс] — Режим доступу: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7801866/

8. San Salvador, El Salvador. Air Quality Communication Workshop. April 16-17, 2012. [Електронний ресурс] — Режим доступу: https://www.epa.gov/sites/production/files/2014-05/documents/zell-aqi.pdf
9. Матеріал з Вікіпедії. “Індекс якості повітря”. [Електронний ресурс] — Режим доступу: https://uk.wikipedia.org/wiki/Індекс_якості_повітря
10. Офіційна документація сервісу Open Street Map. [Електронний ресурс] — Режим доступу: https://wiki.openstreetmap.org/wiki/Main_Page
11. “Разворачиваем сервис построения маршрутов OSRM”. [Електронний ресурс] — Режим доступу: https://habr.com/ru/post/224731/
12. Офіційна документація проекту OSRM. [Електронний ресурс] — Режим доступу: http://project-osrm.org/docs/v5.5.1/api/#general-options
13. Nils Nolde. “Open Source Routing Engines – An Overview”. [Електронний ресурс] — Режим доступу: https://gis-ops.com/open-source-routing-engines-and-algorithms-an-overview/
14. GraphHopper Directions API docs: [Електронний ресурс] — Режим доступу: https://docs.graphhopper.com/
15. “AWS vs Azure vs Google Cloud: Сравнение”. [Електронний ресурс] — Режим доступу: https://dinarys.com/ru/blog/comparison-of-aws-vs-azure-vs-google-cloud
16. Кристофер Беннэйдж. “Руководство по архитектуре облачных приложений”. Microsoft Press. Подразделение корпорации Microsoft One Microsoft Way. Redmond, Washington, 2017 г. – 319 с.
17. Документація Azure. [Електронний ресурс] — Режим доступу: https://docs.microsoft.com/

18. Документація Azure. “Azure Active Directory B2C”. [Електронний ресурс] — Режим доступу: https://docs.microsoft.com/
19. Jithesh Chandrasekharan. “Practical Azure: Secure a .NET Core Web API using Azure AD B2C”. *Codeburst.* [Електронний ресурс] — Режим доступу: https://codeburst.io/practical-azure-secure-a-net-core-web-api-using-azure-ad-b2c-fde234a8e819
20. Nicolas Yuen. “Azure AD B2C Custom Policy with REST API”. [Електронный ресурс]: https://nicolas-yuen.medium.com/azure-ad-b2c-custom-policy-with-rest-api-e4dc560b7245
21. X Li, Y J Feng and H Y Liang. Published under licence by IOP Publishing Ltd. “The Impact of Meteorological Factors on PM2.5 Variations in Hong Kong”. [Електронний ресурс] — Режим доступу: https://iopscience.iop.org/article/10.1088/1755-1315/78/1/012003/pdf
22. Документація Azure. “Use MSAL.NET to sign in users”. [Електронний ресурс] — Режим доступу: https://docs.microsoft.com/en-us/azure/active-directory/develop/msal-net-aad-b2c-considerations
23. Jithesh Chandrasekharan. “Practical Azure: Authenticate Angular App using Azure AD B2C”. *Codeburst.* [Електронний ресурс] — Режим доступу: https://codeburst.io/practical-azure-azure-ad-b2c-and-angular-a30e3c2225ee

ДОДАТОК 1

Програмна система визначення оптимального місця для занять спортом на основі показників стану міського довкілля

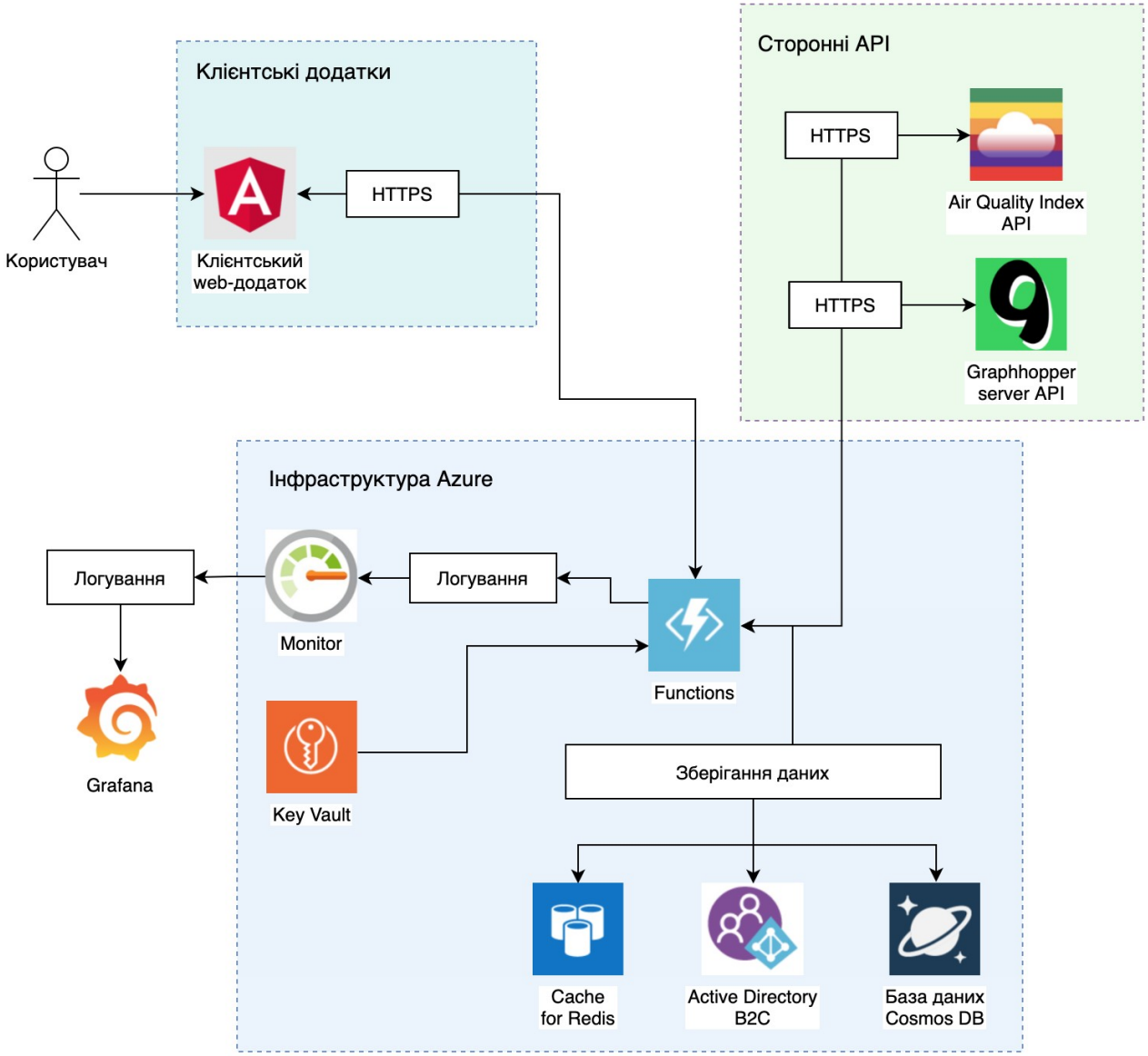
Програмна система визначення оптимального місця для занять спортом на основі показників стану міського довкілля

Схема структурна

ІАЛЦ.467200.005 Д1

Аркушів 1

Київ – 2021 року



					ІАЛЦ.467200.005 Д1	
Зм.	*Арк.*	*№ докум.*	*Підпис*	*Дата*		
Розроб.		Уваров Ю.А.			*Програмна система визначення оптимального місця для занять спортом на основі показників стану міського довкілля.* Схема структурна	*Літера* / *Аркуш.* 1 / *Аркушів* 1
Перевір.		Коган А.В.				
Реценз.						*НТУУ «КПІ ім. Ігоря Сікорського», ФІОТ*
Н. Контр.		Симоненко В.П.				
Затверд.		Стіренко С.Г.				

ДОДАТОК 2

Програмна система визначення оптимального місця для занять спортом на основі показників стану міського довкілля

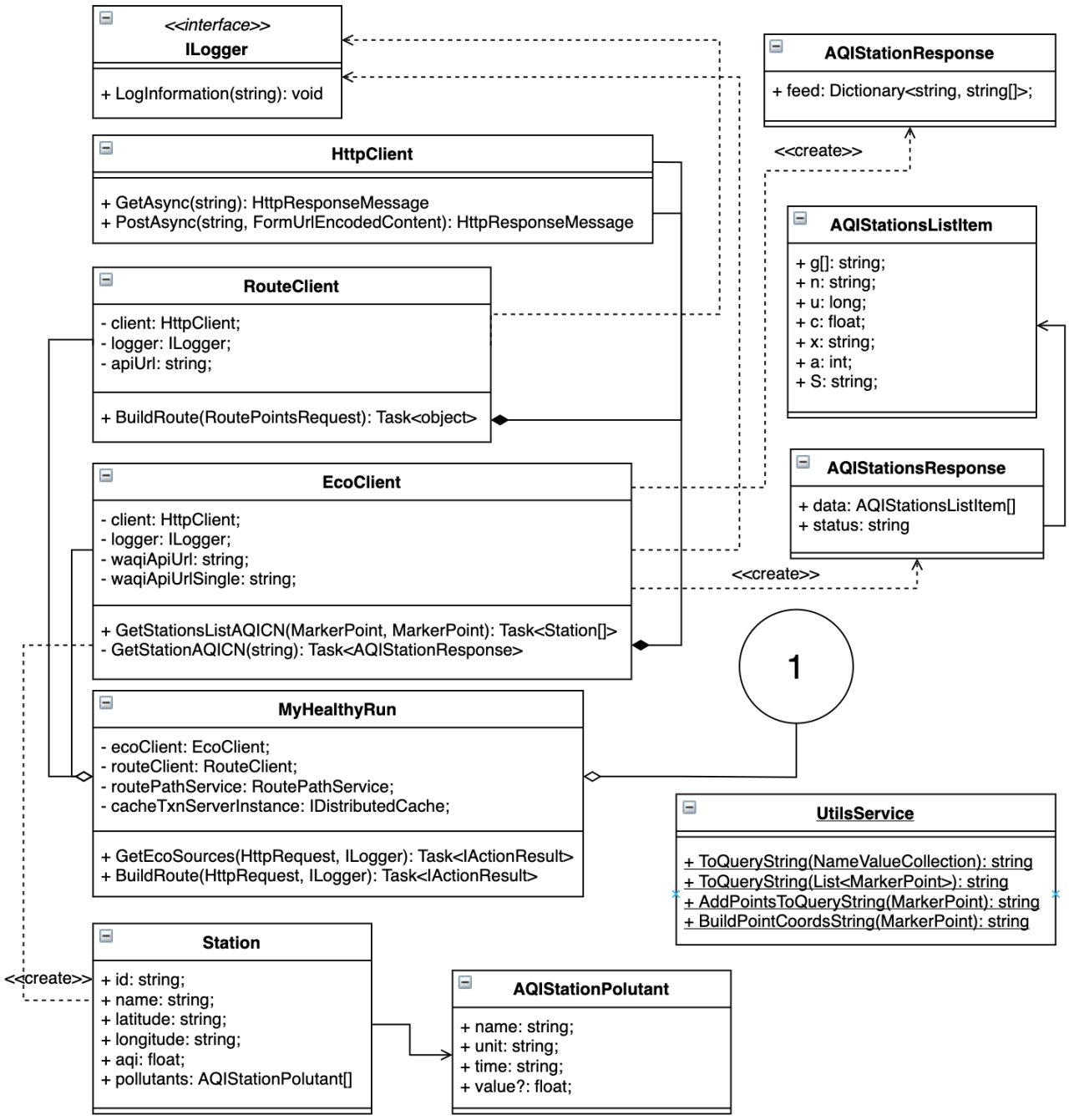
Програмна система визначення оптимального місця для занять спортом на основі показників стану міського довкілля

Діаграма класів

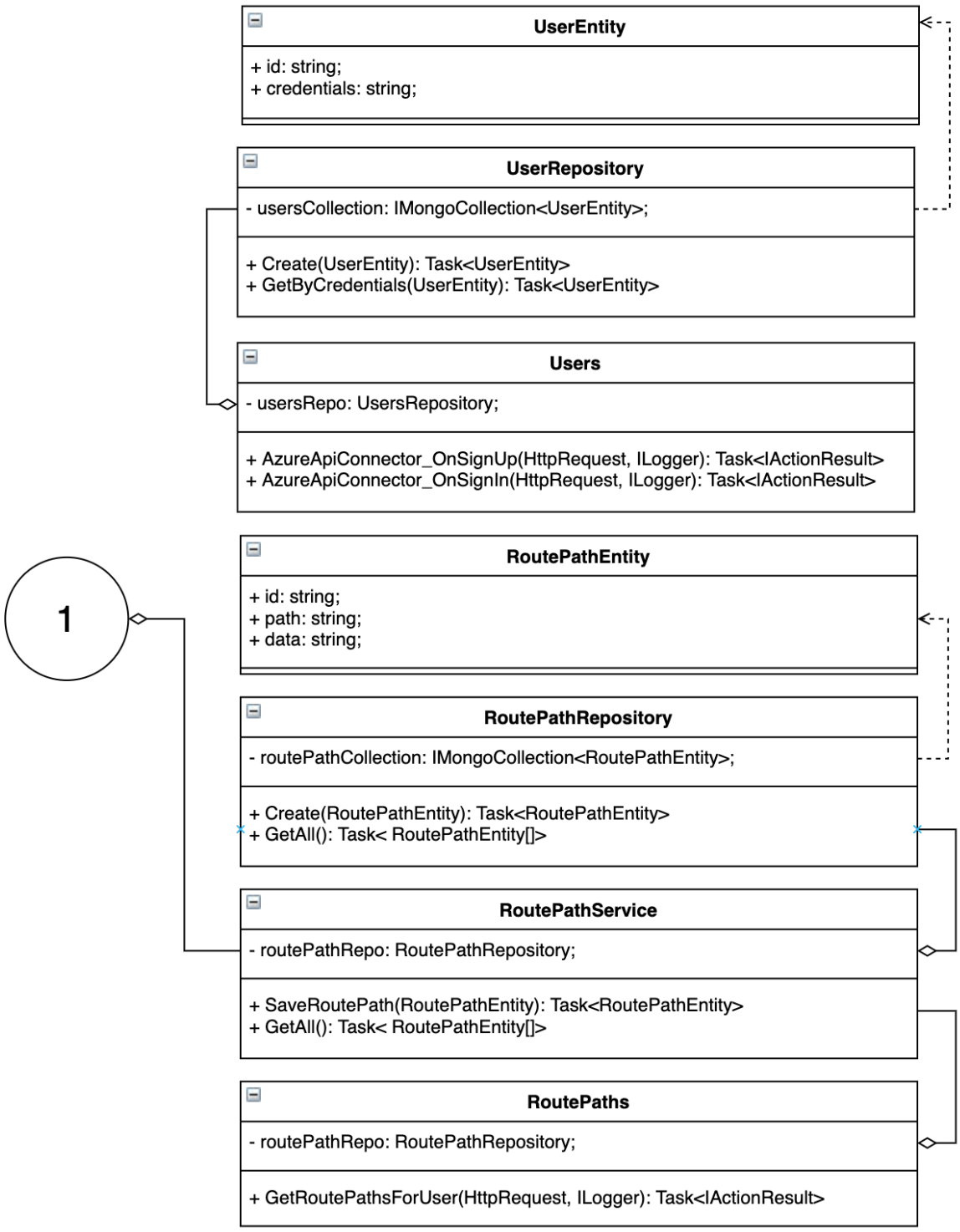
ІАЛЦ.467200.006 Д2

Аркушів 2

Київ – 2021 року



					ІАЛЦ.467200.006 Д2	
Зм.	Арк.	№ докум.	Підпис	Дата		
Розроб.		Уваров Ю.А.			Програмна система визначення оптимального місця для занять спортом на основі показників стану міського довкілля. Діаграма класів	Літера: — / Аркуш.: 1 / Аркушів: 2
Перевір.		Коган А.В.				
Реценз.						НТУУ «КПІ ім. Ігоря Сікорського», ФІОТ
Н. Контр.		Симоненко В.П.				
Затверд.		Стіренко С.Г.				


UserEntity
+ id: string;
+ credentials: string;
UserRepository
- usersCollection: IMongoCollection<UserEntity>;
+ Create(UserEntity): Task<UserEntity>
+ GetByCredentials(UserEntity): Task<UserEntity>
Users
- usersRepo: UsersRepository;
+ AzureApiConnector_OnSignUp(HttpRequest, ILogger): Task<IActionResult>
+ AzureApiConnector_OnSignIn(HttpRequest, ILogger): Task<IActionResult>
1
RoutePathEntity
+ id: string;
+ path: string;
+ data: string;
RoutePathRepository
- routePathCollection: IMongoCollection<RoutePathEntity>;
+ Create(RoutePathEntity): Task<RoutePathEntity>
+ GetAll(): Task< RoutePathEntity[]>
RoutePathService
- routePathRepo: RoutePathRepository;
+ SaveRoutePath(RoutePathEntity): Task<RoutePathEntity>
+ GetAll(): Task< RoutePathEntity[]>
RoutePaths
- routePathRepo: RoutePathRepository;
+ GetRoutePathsForUser(HttpRequest, ILogger): Task<IActionResult>

ДОДАТОК 3

Програмна система визначення оптимального місця для занять спортом на основі показників стану міського довкілля

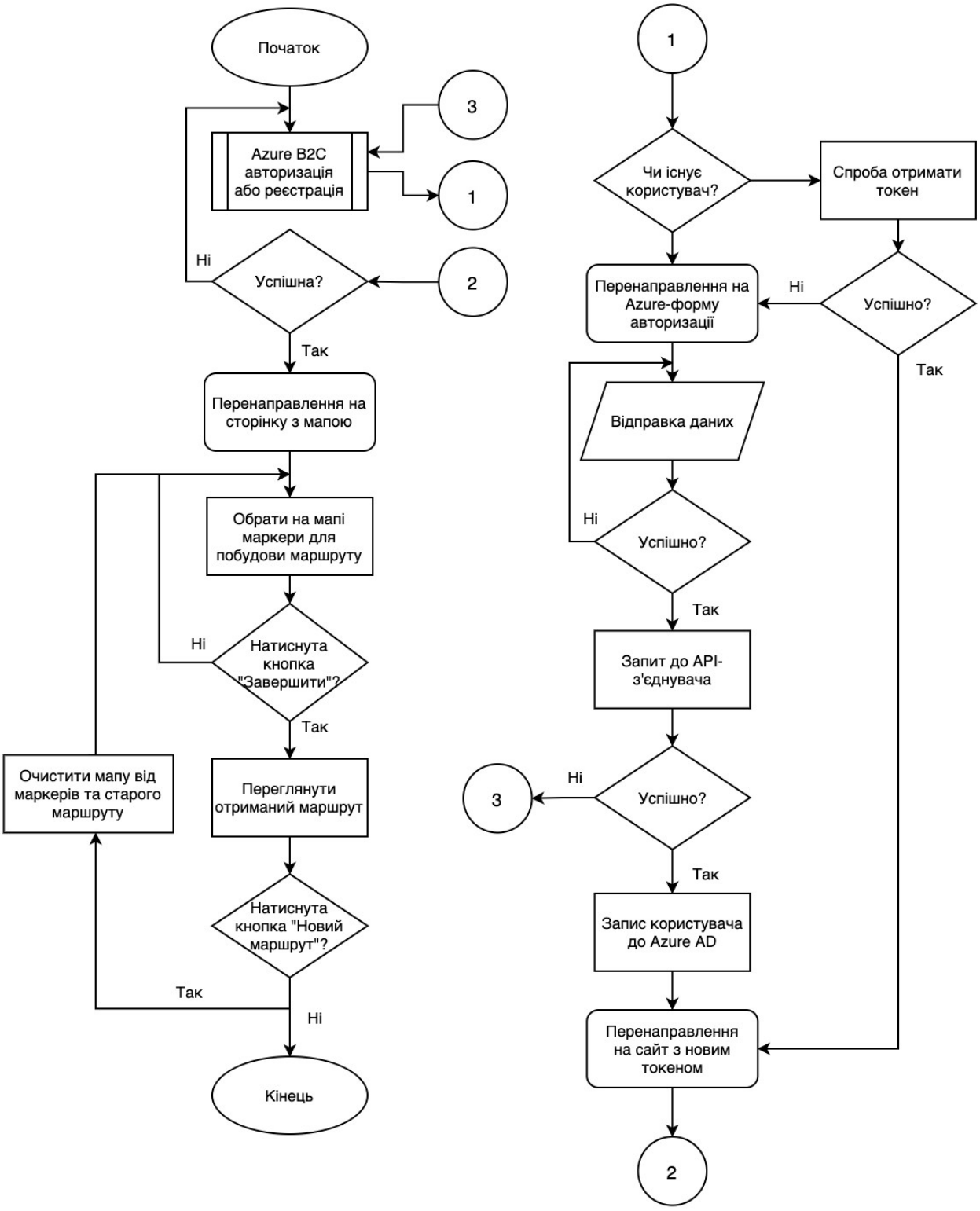
Програмна система визначення оптимального місця для занять спортом на основі показників стану міського довкілля

Алгоритм взаємодії користувача з системою

ІАЛЦ.467200.007 Д3

Аркушів 1

Київ – 2021 року



					ІАЛЦ.467200.007 ДЗ	
Зм.	*Арк.*	*№ докум.*	*Підпис*	*Дата*		
Розроб.		Уваров Ю.А.			*Програмна система визначення оптимального місця для занять спортом на основі показників стану міського довкілля.* Алгоритм взаємодії з системою	*Літера* / *Аркуш.* 1 / *Аркушів* 1
Перевір.		Коган А.В.				
Реценз.						*НТУУ «КПІ ім. Ігоря Сікорського», ФІОТ*
Н. Контр.		Симоненко В.П.				
Затверд.		Стіренко С.Г.				