ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка Telegram-боту мовою програмування C# для перегляду прогнозу погоди»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Олександр КОЖЕМ'ЯКІН
(підпис)

Виконав: здобувач вищої освіти групи ПД-41

Олександр КОЖЕМ'ЯКІН

Керівник: Олег ІЛЬЇН
д.т.н., професор

Рецензент: _____

Київ 2024

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
« ____ » _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Кожем'якіну Олександру Вікторовичу

1. Тема кваліфікаційної роботи: «Розробка Telegram-боту мовою програмування C# для перегляду прогнозу погоди»
керівник кваліфікаційної роботи д.т.н., професор Олег ІЛЬЇН,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «27» лютого 2024 р. № 36.

2. Строк подання кваліфікаційної роботи «28» травня 2024 р.

3. Вихідні дані до кваліфікаційної роботи: Вибір інструментів та технологій, реєстрація та налаштування бота у Telegram, отримання доступу до публічного API Open Weather Map, платформа .NET 8 та мова програмування C#.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Здійснити аналіз предметної області.
2. Дослідити існуючі програмні засоби, виявити ключові можливості.
3. Визначити функціональні та нефункціональні вимоги застосунку.

4. Проаналізувати доступні інструменти для розробки програмного продукту.
5. Розробити механізм отримання та обробки погодних даних.
6. Спроектувати та розробити Telegram-бот відповідно до визначених потреб та вимог.
7. Виконати тестування розробленого Telegram-боту.

5. Перелік графічного матеріалу: *презентація*

1. Таблиця аналізу аналогів.
2. Вимоги до програмного забезпечення
3. Логотипи програмних засобів реалізації
4. Діаграма варіантів використання.
5. Контекстна діаграма.
6. Діаграми діяльності.
7. Діаграми класів.
8. Діаграма розгортання.
9. Екранні форми.
10. Відео роботи застосунку.
11. Апробація результатів дослідження.

6. Дата видачі завдання «28» лютого 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	28.02-06.03.2024	
2	Аналіз та дослідження існуючих аналогів	07.03-13.03.2024	
3	Підбір інструментів розробки	14.03-20.03.2024	
4	Визначення вимог до застосунку	21.03-27.03.2024	
5	Проектування телеграм-боту для перегляду прогнозу погоди	28.03-10.04.2024	
6	Програмна реалізація застосунку	11.04-25.04.2024	
7	Тестування застосунку	26.04-29.04.2024	
8	Оформлення роботи: вступ, висновки, реферат	29.04-05.05.2024	
9	Розробка демонстраційних матеріалів	06.05-12.05.2024	
10	Попередній захист роботи	13.05-31.05.2024	

Здобувач вищої освіти _____ Олександр КОЖЕМ'ЯКІН
(підпис)

Керівник
кваліфікаційної роботи _____ Олег ІЛЬЇН
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: __55__ стор., ___3__ табл., __56__ рис., __9__ джерел.

Мета роботи – полегшення отримання погодної інформації за допомогою впровадження Telegram-боту.

Об'єкт дослідження – процес отримання погодної інформації.

Предмет дослідження – програмне забезпечення для отримання погодної інформації.

Короткий зміст роботи: В результаті аналізу предметної області було проаналізовано різні аналогічні застосунки. Виділені такі аналогічні застосунки: «Погода» на IOS/iPadOS, «Погода» на Android та застосунок AccuWeather. Завдяки проведеному аналізу було сформовано відповідні функціональні вимоги до застосунку: перегляд погодної інформації за назвою місця або з використанням поточного місцезнаходження, створення користувацьких погодних розсилок, перегляд створених розсилок, видалення розсилок та обробка виключень при виникненні помилок під час роботи застосунку.

Проведено аналіз існуючих програмних засобів та інструментів реалізації. Для реалізації Telegram – боту було обрано мову програмування C#, платформу .NET та фреймворк ASP.NET Core. Для зручної взаємодії з Telegram Bot API було обрано популярний клієнт для створення ботів Telegram.Bot. Клієнт підтримує всі наявні актуальні функції, які надає Telegram Bot API для проектування та розробки Telegram – ботів завдяки бібліотеці, що містить низку корисних методів.

Було спроектовано та розроблено бібліотеку класів, яка відповідає за взаємодію з API від Open Weather Map. Розроблена бібліотека відповідає всім нефункціональним вимогам та надає всі необхідні методи, необхідні для отимання погодної інформації.

Для збереження даних щодо створених користувачами підписок на погодні

розсилки, було розроблено мікросервіс, що базується на фреймворку ASP.NET Core із дотриманням принципів REST. Мікросервіс спроектовано з використанням трирівневої архітектури для розбиття застосунку на відповідні шари. Щоб забезпечити швидку та надійну роботи бази даних, обрано СУБД MySQL.

В подальших дослідження буде розширено доступний користувачеві функціонал. Додати можливість збереження локацій у список обраних з можливістю подальшого перегляду, редагування та видалення. Також необхідно додати розширене логування повідомлень для визначення поточного стану застосунку та його зручної підтримки.

КЛЮЧОВІ СЛОВА: ТЕЛЕГРАМ, БОТ, .NET CORE, ASP.NET CORE, C#, BOTFATHER, ПОГОДА, API.

ЗМІСТ

ВСТУП

Telegram є однією з найпопулярніших месенджер-платформ, яку використовують мільйони користувачів з усього світу. Відкритий API та можливість створення ботів надають широкий спектр можливостей для створення різноманітних програмних застосунків для певних потреб. Telegram-боти стали важливим інструментом для надання різних послуг, таких як новини, сповіщення, підтримка покупців, проведення платежів та інше. Вони забезпечують зручний спосіб взаємодії з користувачами без необхідності використання інших застосунків.

В тому числі Telegram-бота можна використовувати для перегляду прогнозу погоди. Актуальність погодної інформації є завжди важливим фактором, що може кардинально змінити плани та справи. Особливо важливою актуальність погоди є для тих, чия діяльність безпосередньо залежить від кліматичних умов.

Проте, реалізація ботів програмними засобами може бути досить складною, адже кількість мов програмування, інструментів розробки та фреймворків може не аби як шокувати розробника, але існуючі засоби дозволяють створювати комплексні системи, які будуть виконувати різноманітні операції у вигляді боту у месенджері Telegram.

Об’єкт дослідження - процес отримання погодної інформації.

Предмет дослідження - програмне забезпечення для отримання погодної інформації.

Мета дослідження - полегшення отримання погодної інформації за допомогою впровадження Telegram-боту. Щоб реалізувати сформовану мету слід вирішити наступні задачі:

1. Здійснити аналіз предметної області

2. Дослідити існуючі програмні засоби, виявити ключові можливості.

3. Визначити функціональні та нефункціональні вимоги застосунку.

4. Проаналізувати доступні інструменти для розробки програмного продукту.

5. Розробити механізм отримання та обробки погодних даних.

6. Спроектувати та розробити Telegram-бот відповідно до визначених потреб та вимог.

7. Виконати тестування розробленого Telegram-боту.

8. Апробація результатів дослідження: робота успішно пройшла апробацію на Всеукраїнській науково-технічній конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях» за темами: «Аналіз засобів розробки Телеграм-боту для перегляду прогнозу погоди», «Використання RESTful WEB API для отримання погодної інформації щодо Телеграм-боту для перегляду прогнозу погоди», 24 квітня 2024 року, Київ, Державний університет інформаційно-комунікаційних технологій.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Значення погоди для людини

Погода відіграє важливу роль у житті кожної людини, впливаючи на її фізичний та психічний стан. Погода має прямий вплив на здоров'я людини. Наприклад, прохолодна погода може спричинити виникнення переохолодження та деяких захворювань. Крім того погода має вплив на емоційну складову людини. Сонячні дні зазвичай покращують настрій та підвищують рівень енергії, з іншого боку похмурі зимові дні можуть викликати сонливість на занепад настрою. Не слід нехтувати також і впливом на щоденні звички та діяльність. Існує безліч ситуацій, коли певні погодні умови руйнують плани або шкодять розпорядку дня. Але існує й інша сторона, гарна погода може сприяти появі нових ідей та звичок, які позитивно вплинуть на загальний стан людини.

Слід додати, що погодні умови впливають і на економічну діяльність. Такі великі галузі як сільське господарство напряму залежать від погодних умов. Кількість опадів, температура навколишнього середовища та інші кліматичні фактори впливають на кількість врожаю. Невчасно повідомлений прогноз погоди може завдати величезних економічних втрат.

Погода є невід'ємною частиною нашого життя, яка впливає на різні аспекти нашої діяльності та стану, тому важливо завжди мати на увазі актуальну погодну інформацію.

Програмні засоби, що забезпечують вчасне інформування щодо погодних умов стали невід'ємною складовою щоденного використання. Для того щоб дізнатися актуальну погодну інформацією можна скористуватися різними видами програмних засобів, наприклад: веб-застосунки, чат-боти та мобільні застосунки. Із загальною популярністю мобільних пристроїв якнайбільше користувачів

надають перевагу до різноманітних застосунків, які передбачають відображення інформації про погоду та чат-ботів.

1.2 Переваги використання програмних засобів для отримання погодної інформації

Різноманітні застосунки виконують важливу роль щодо інформування користувачів про погодні умови. Вони надають широку картину про погоду у навколишньому середовищі, стан якості повітря, кількість та вірогідність опадів у найближчий час, вологість повітря та інші дані. До переваг використання програмних засіб для отримання погодної інформації слід віднести такі пункти:

1. Зручність використання. Більшість застосунків мають зрозумілий інтерфейс, що сприяє виникненню позитивного досвіду використання зі сторони користувача.

2. Оперативність та доступність інформації. Застосунки для погоди надають актуальні дані щодо погодних умов в режимі реального часу або через певні інтервали. Користувачі мають змогу швидко отримати необхідно інформацію.

1.3 Telegram-бот як засіб для здобуття інформації про погоду

Telegram-бот – це спеціальний програмний засіб, який базується на месенджері Telegram та надає різні можливості для створення автоматизованих систем для різних цілей.

Для взаємодії з користувачем через бота надається велика кількість засобів, які можуть значно покращити досвід роботи. Завдяки набору доступних засобів, які надає API Telegram можна створити інтуїтивно зрозумілий користувачу інтерфейс та набір функцій, необхідних для певних задач.

Telegram-бот дозволяє створювати комплексні рішення всередині месенджеру телеграм для виконання різноманітних завдань або взаємодії з іншими системами через інтеграції.

Створення Telegram-боту для перегляду прогнозу погоди дозволяє користувачеві використовувати лише один застосунок для отримання різної інформації та даних. Великий набір можливостей, які надає Telegram розробникам через різні API відкриває широкий спектр різноманітних завдань, які можна реалізувати завдяки створенню боту. Насамперед Telegram-бот для перегляду прогнозу погоди є зручним доповненням до месенджеру, який дозволяє користувачеві отримувати користувачеві актуальну погодну інформацію.

1.4 Застосунок «Погода» на пристроях з операційними системами IOS та iPad OS.

Застосунок «Погода» на пристроях з операційними системами IOS та iPadOS – це вбудований програмний засіб, який надає користувачам інформацію про поточні погодні умови, прогнози на найближчі дні та іншу корисну інформацію щодо метеорологічних даних.

Застосунок пропонує стандартні функції щодо перегляду погодної інформації, а зрозумілий та зручний інтерфейс допоможе користувачеві швидко зорієнтуватись та знайти для себе необхідну інформацію.

Для оновлення інформації використовується надійне джерело метеорологічних даних, The Weather Channel, що гарантує достовірність.

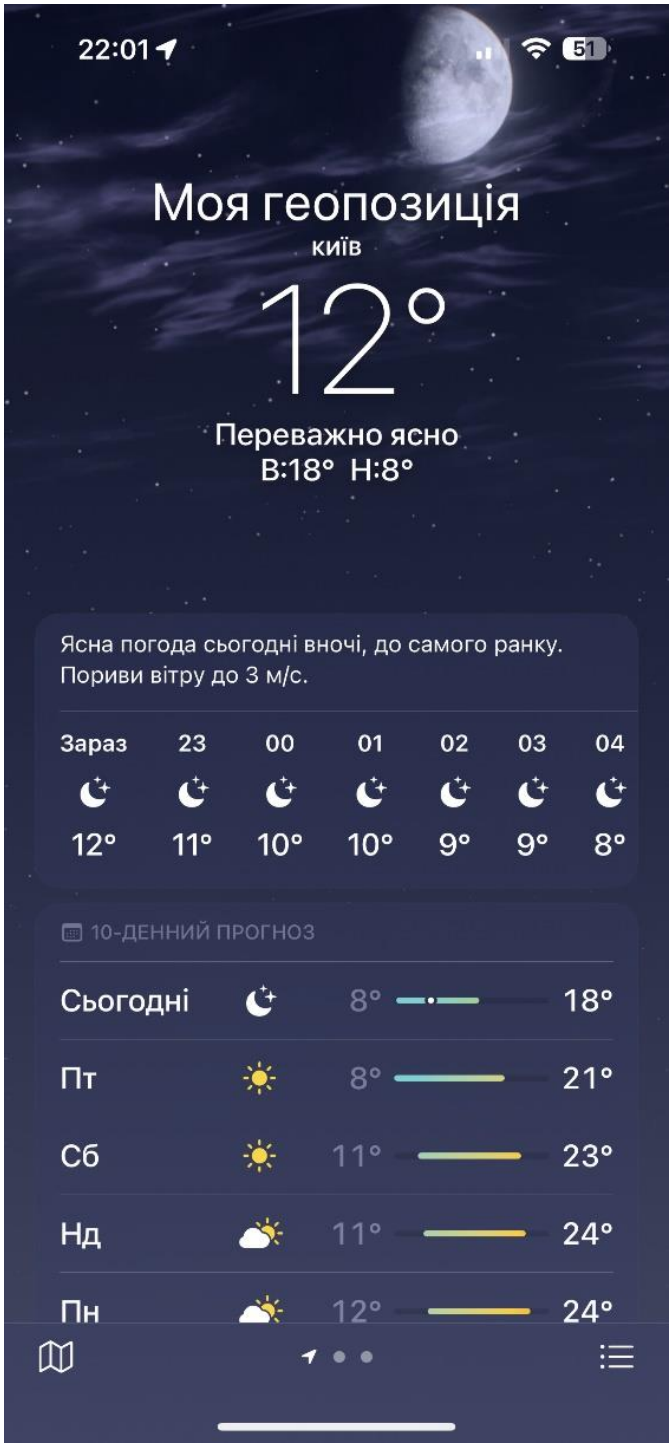


Рис. 1.1 Застосунок "Погода" на пристроях з IOS/iPadOS

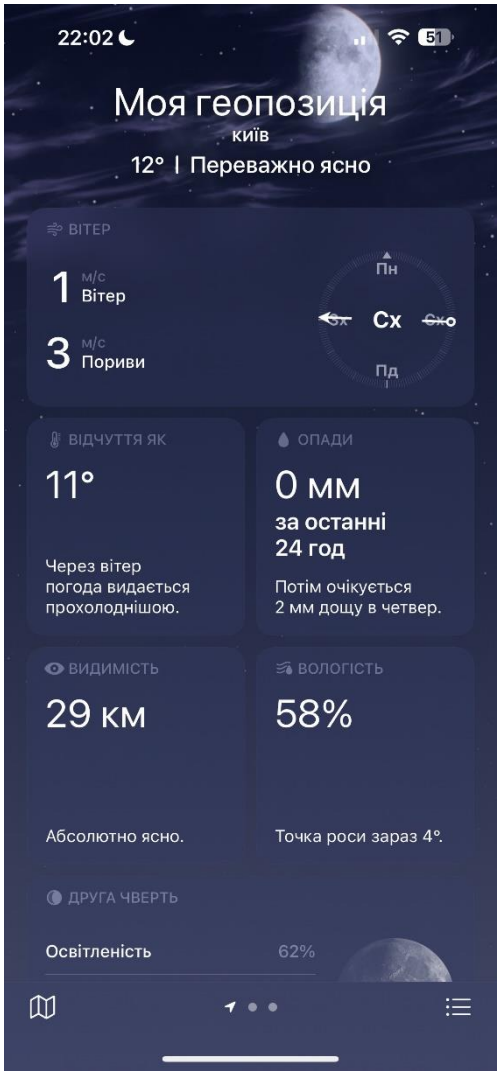


Рис. 1.2 Додаткова інформація у застосунку "Погода" на IOS/iPadOS

1.5 Застосунок «Погода» на пристроях з ОС Android

На більшості пристроїв під управлінням операційної системи Android із заводу виробника встановлено застосунок «Погода». Виробники пристроїв на цій операційній системі можуть видозмінювати зовнішній вигляд застосунку, проте функції у переважній більшості залишаються однаковими. У більшості випадках набір функціональних можливостей залишаються однаковими. Проте через різноманітність обгорток над операційною системою Android, може потерпати достовірність інформації адже кожен виробник може використовувати різні сервіси, які надають необхідну погодну інформацію.

Рис. 1.3 Приклад застосунку "Погода" на пристроях під управлінням Android

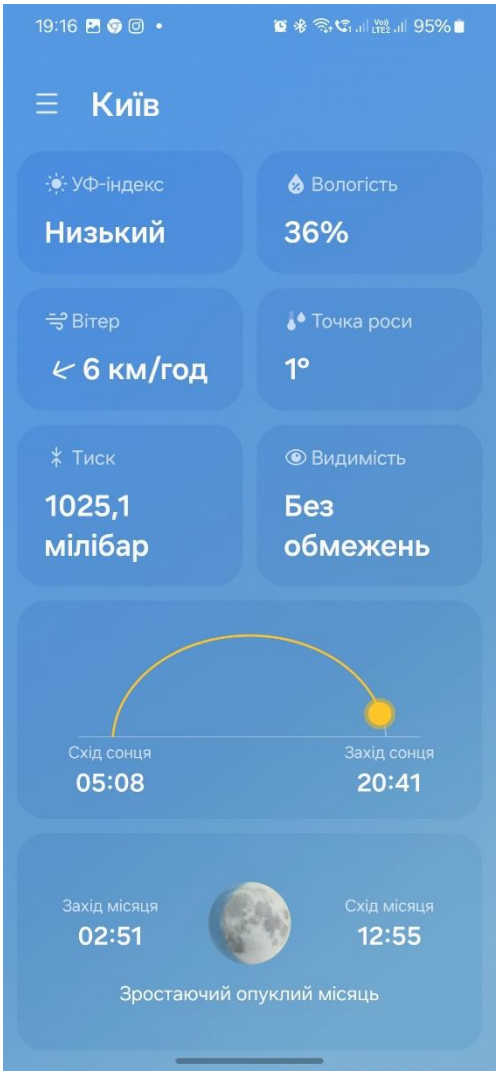


Рис. 1.4 Додаткові екранні форми застосунку "Погода" на пристроях Android

1.6 Застосунок AccuWeather

Є одним з популярних застосунків на пристроях з операційними системами IOS/iPadOS та Android. Надає більшість необхідної користувачеві інформацію і забезпечує достовірність та актуальність погодних даних завдяки використанню надійного сервісу надання погодної інформації. Але у порівнянні з іншими застосунками – аналогами у даному застосунку наявна велика кількість реклами у безкоштовній версії, що значно погіршує досвід використання.

Рис. 1.5 Застосунок AccuWeather

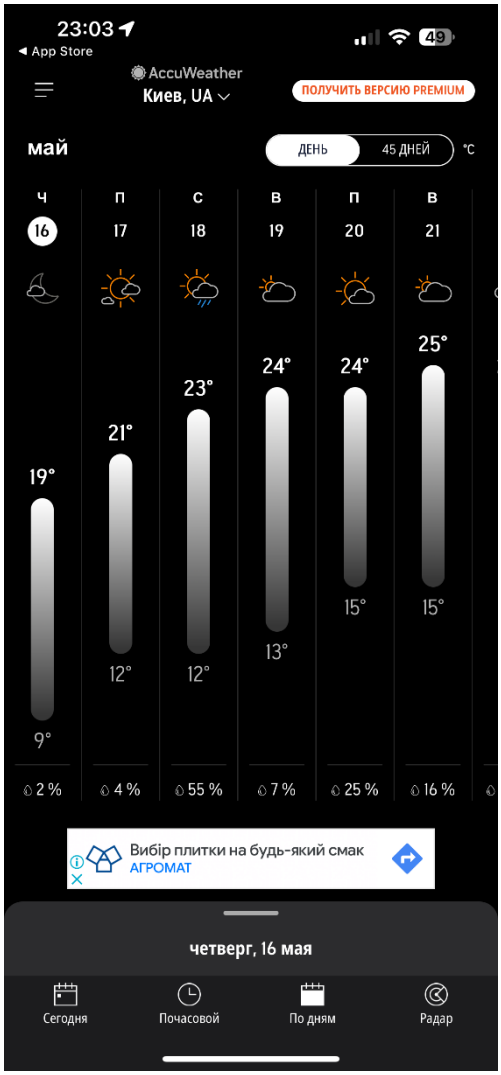


Рис. 1.6 Додаткові екранні форми у застосунку AccuWeather

1.7 Таблиця порівняння застосунків-аналогів

Таблиця 1.1

Порівняння існуючих програм-аналогів

Функціональні можливості	«Погода» на IOS/iPadOS	«Погода» на Android	AccuWeather	Convenient Weather Bot
Створення, перегляд та видалення підписок на погодну розсилку	Ні	Ні	Ні	Так

Продовження таблиці 1.1

Порівняння існуючих програм-аналогів

Функціональні можливості	«Погода» на IOS/iPadOS	«Погода» на Android	AccuWeather	Convenient Weather Bot
Детальна погодна інформація	Так	Так	Так	Так
Погодна інформація за підписками на розсилку	Ні	Ні	Ні	Так
Погода за поточним місцезнаходженням	Так	Так	Так	Так
Інтеграція з Telegram	Ні	Ні	Ні	Так
Зображення актуальних погодних умов	Так	Так	Так	Так

1.8 Огляд програмних засобів реалізації

1.8.1 Платформа .NET

.NET – це програмна платформа, розроблена компанією Microsoft, яка надає розробникам широкі можливості для створення різноманітних програм, таких як веб-застосунки, десктопні та мобільні застосунки і хмарні сервіси.

Завдяки модульності, кожний компонент оновлюється окремо через менеджер пакетів NuGet. Застосунки створені на платформі .NET можуть працювати з різними наборами модулів та не бути залежними від єдиного оновлення базової платформи.

Ключовим компонентом, який надає потужність і гнучкість платформі .NET є CLR. CLR (Common Language Runtime) – це основний компонент платформи,

який забезпечує середовище виконання для керованого коду. CLR відповідає за виконання програм, що написані на мовах програмування, підтримуваних .NET

Платформа .NET забезпечує високу продуктивність, безпеку та гнучкість, що робить її досить популярним вибором для створення сучасних програмних продуктів.

Основною мовою програмування, яка використовується для створення програм на .NET є C#. C# - строго типізована об’єктно-орієнтована мова програмування, створена Microsoft як частина платформи. Вона була створена у 2000 році та стала однією з найпопулярніших мов програмування для розробки. C# стабільно отримує нові версії, в яких додаються нові можливості та покращується синтаксис і зручність.

Рис. 1.7 Логотип платформи .NET

1.8.2 ASP.NET Core

ASP.NET Core – це кросплатформний, високопродуктивний фреймворк з відкритим кодом на основі платформи .NET для розробки сучасних хмарних та веб-застосунків. Швидкість, відкритість та широкий спектр можливостей дозволяє розробникам створювати різноманітні сервіси, а основними характеристиками та можливостями є:

1. Кросплатформеність: фреймворк підтримує розробку та виконання програм на різних операційних системах, таких як Windows, Linux та macOS.

2. Висока продуктивність: Завдяки різноманітним оптимізаціям і ефективному використанню ресурсів, ASP.NET Core забезпечує високу продуктивність і швидкість роботи.

3. Модульність: фреймворк побудований на модульній архітектурі, що дозволяє розробникам використовувати тільки необхідні компоненти, зменшуючи тим самим об’єм і складність програмного коду.

4. Розширюваність: ASP.NET Core легко розширюється за рахунок підтримки різноманітних сторонніх бібліотек.

Рис. 1.8 Приклад логотипу ASP.NET Core

1.8.3 СУБД MySQL

MySQL – це популярна система управління базами даних, яка використовується для зберігання, управління та обробки даних. Розроблена компанією TcX. MySQL є відкритим програмним забезпеченням та підтримується Oracle Corporation. Основні можливості СУБД MySQL:

1. Відкритість: MySQL є проектом з відкритим кодом.

2. Висока продуктивність: MySQL забезпечує високу швидкість роботи та ефективне розподілення ресурсів, що добре підходить для обробки великих обсягів даних.

3. Легкість у використання: MySQL має розповсюджений синтаксис SQL, що полегшує адаптацію та вивчення після інших СУБД з подібним синтаксисом.

4. Реплікація: підтримка різних видів реплікації, що забезпечує високу доступність та відновлюваність даних.

5. Інтеграція з різними мовами програмування: MySQL підтримує різні популярні мови програмування: C#, Java, PHP, Python, Ruby та інші.

Рис. 1.9 Логотип СУБД MySQL

1.8.4 Docker

Docker – це відкрите програмне забезпечення, яке дозволяє автоматизувати розгортання застосунків у контейнерах. Контейнери допомагають розробникам упаковувати створені застосунки та їх залежності у компактний вигляд. Це дозволяє спростити процес запуску застосунку та надає можливість його виконання на будь якій операційній системі, яка підтримує Docker. Коротка характеристика Docker:

1. Контейнеризація: забезпечується ізоляція застосунків та їх залежностей в окремих контейнерах, що дозволяє спростити процес запуску та уникнення конфліктів.

2. Легкість використання: Docker забезпечує простий і зрозумілий інтерфейс командного рядка та API, що полегшує створення, управління та розгортання контейнерів.

3. Масштабованість: Docker дозволяє легко масштабувати додатки, розгортаючи декілька контейнерів.

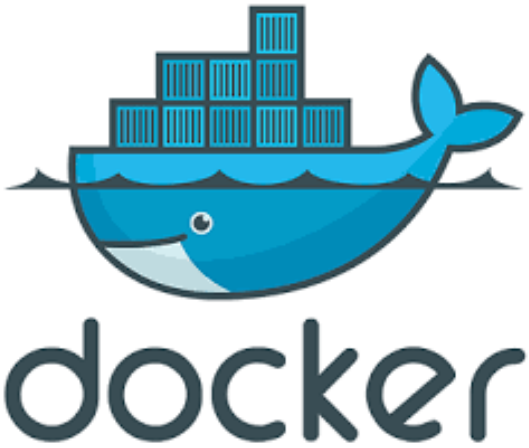


Рис. 1.10 Один із логотипів Docker

1.8.5 Git

Git – це система управління версій, що застосовується для контролю змін у коді під час розробки програмного забезпечення. Була створена у 2005 році Лінусом Торвальдсом та стала однією з найпопулярніших систем контролю версій завдяки швидкості, надійності та гнучкості. Можливості Git:

1. Розподілена архітектура: у Git кожен розробник має доступ до повної копії історії змін проекту, що дозволяє автономно працювати.

2. Гілкування та злиття змін: дозволяє створювати окремі гілки, в яких можна створювати окремих функціонал без внесення змін в основний проект, а злиття дозволяє переносити нові функції та зміни в основну та інші гілки.

3. Історія змін: Git зберігає повну історію змін у проекті, що дозволяє відслідкувати важливі зміни у проекті та місця, де вони були проведені.

Рис. 1.11 Логотип Git

1.8.6 Rider

Rider — це інтегроване середовище розробки (IDE), створене компанією JetBrains, для розробки програмного забезпечення на платформі .NET та мовами програмування, що підтримуються платформою. Rider побудований на основі платформи IntelliJ і використовує технології ReSharper для аналізу та рефакторингу коду. Основні характеристики та можливості Rider включають:

1. Кросплатформеність: Rider доступний для Windows, macOS та Linux, що забезпечує гнучкість у виборі операційної системи для розробки.

2. Інтеграція з .NET: Rider підтримує .NET, включаючи .NET Core, .NET Framework, Xamarin та Unity, що дозволяє розробляти різні типи застосунків, включаючи веб-застосунки, мобільні застосунки та ігри.

3. Інструменти для рефакторингу: Rider надає потужні інструменти для рефакторингу коду, які допомагають покращувати якість коду, виявляти та виправляти помилки, а також оптимізувати продуктивність додатків під час написання програмного коду.

4. Інтеграція з системами контролю версій: Rider підтримує інтеграцію з Git, Mercurial, Perforce та іншими системами контролю версій, що дозволяє ефективно керувати версіями проекту та співпрацювати з іншими розробниками.

Рис. 1.12 Логотип IDE Rider

1.8.7 DataGrip

DataGrip — це інтегроване середовище розробки (IDE) для роботи з базами даних, розроблене компанією JetBrains. Воно підтримує різні системи управління базами даних (СУБД) та надає потужні інструменти для управління базами даних, написання запитів та аналізу даних. Основні характеристики та можливості DataGrip включають:

1. Підтримка різних СУБД: підтримує роботу з багатьма популярними СУБД, такими як MySQL, PostgreSQL, Oracle, SQL Server, SQLite, MariaDB, DB2, H2, та іншими, що робить його універсальним інструментом для роботи з різними базами даних.

2. Навігатор по базі даних: DataGrip дозволяє переглядати структуру бази даних, включаючи таблиці, колонки, індекси, процедури, тригери та інші об'єкти.

3. Виконання та налагодження запитів: DataGrip забезпечує можливість виконання SQL-запитів і перегляду результатів у зручному інтерфейсі. Вбудований відладчик допомагає знаходити та виправляти помилки в запитах.

Рис. 1.13 Логотип IDE DataGrip

1.8.8 Додаткові засоби розробки

Telegram.Bot – це бібліотека, яка дозволяє розробникам створювати ботів для платформи Telegram за допомогою мови програмування C#. Вона надає простий і зручний набір засобів для взаємодії з Telegram Bot API, що дозволяє швидко розробляти функціональні та інтерактивні боти.

Newtonsoft.Json – це популярна бібліотека для роботи з JSON у .NET, розроблена Джеймсом Ньютоном-Кінгом. Вона дозволяє легко серіалізувати та десеріалізувати об'єкти у формат JSON, а також надає безліч корисних функцій для роботи з JSON-даними. Основні можливості та характеристики Newtonsoft.Json включають:

1. Серіалізація та десеріалізація: Newtonsoft.Json дозволяє легко перетворювати об'єкти C# у формат JSON і навпаки. Це особливо корисно для передачі даних між клієнтом і сервером у веб-застосунках.

2. Атрибути для налаштування: Ви можете використовувати атрибути, такі як [JsonProperty], [JsonIgnore], [JsonConverter] тощо, для налаштування серіалізації та десеріалізації на рівні класів і властивостей.

3. Підтримка LINQ до JSON: Newtonsoft.Json надає можливості для роботи з JSON-даними за допомогою LINQ, що дозволяє легко маніпулювати JSON-об'єктами без необхідності попередньої десеріалізації в C# об'єкти.

2 МОДЕЛЮВАННЯ ТА ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1 Технічне завдання

Розробка програмного забезпечення, що передбачає створення Telegram-боту для перегляду погодної інформації з використанням платформи .NET 8, фреймворку ASP.NET Core та мови програмування C# 12-ї версії. Створений програмний засіб повинен бути реалізований у вигляді двох мікросервісів, які здійснюють спілкування між собою за допомогою REST із використанням HTTP протоколу.

Основними вимогами до вихідного продукту є можливість перегляду актуальної погодної інформації через введення відповідного пошукового запиту щодо місця або використання інформації щодо місцезнаходження користувача у вигляді широти та довготи. Також користувач повинен мати змогу створювати, переглядати та видаляти підписки на погодну розсилку. Бот повинен бути відказостійким, обробляти помилки, що виникають під час роботи та відображати повідомлення користувачеві, якщо виникають проблеми під час використання. Постачальником погодних даних є відкрите API від Open Weather Map. Дані від API надходять у форматі JSON та десеріалізуються за допомогою бібліотеки Newtonsoft.JSON у класи C#.

Основний сервіс відповідає за обробку повідомлень та команд, які вводить користувач і надає відповідну відповідь користувачеві у зрозумілому форматі. Основний сервіс може зв’язуватись із другим сервісом для управління користувацькими підписками на погодну розсилку.

Другий сервіс побудований на ASP.NET Core та взаємодіє із СУБД MySQL для збереження та обробки даних щодо підписок.

2.2 Функціональні вимоги

Відображення актуальної погодної інформації за запитом. Користувач повинен мати змогу викликати команду для перегляду поточної погодної інформації, використовуючи пошуковий запит місця, за яким користувач бажає дізнатися інформацію.

Відображення погодної інформації за місцезнаходженням. Необхідна можливість після виклику команди для перегляду погодної інформації запропонувати користувачеві використати його поточне місцезнаходження для полегшення пошуку даних.

Створення підписки на погодну розсилку із використанням запиту. При створенні нової підписки, користувач може ввести пошуковий запит для виявлення місця, за яким необхідно створити підписку.

Створення підписки на погодну розсилку із поточним місцезнаходженням. При створенні нової підписки, користувач може обрати можливість надати поточне місцезнаходження для створення підписки на розсилки за поточною геопозицією.

Перегляд створених підписок. Повинна бути можливість переглядати усі створені підписки на погодну розсилку.

Видалення підписок. Створені підписки користувач може видалити, якщо це необхідно.

Опрацювання помилок в ході опрацювання користувацьких команд. При виникненні будь-яких виключень під час обробки користувацьких команд та повідомлень, користувачеві буде повідомлено про виникнення проблем у вигляді повідомлення від боту.

Валідація даних за погодним запитом. Повинна бути перевірка наявних погодних даних за запитом. Якщо дані відсутні, користувачеві це буде повідомлено.

Перегляд доступних операцій у боті. У користувача повинна бути можливість завдяки команді /help вивести доступні дії у боті з їх коротким описом.

2.3 Нефункціональні вимоги

Продуктивність. Бот повинен надавати відповіді на запити користувачів менш ніж за 2 секунди.

Відказостійкість. Якщо запитуємий мікросервіс знаходиться в неактивному стані, мікросервіс, що запитує продовжує роботу та повідомляє користувача про недоступність функціоналу.

Сумісність. Сервіси боту повинні підтримувати такі операційні системи:

1. Windows.
2. Linux.
3. macOS.

Системні вимоги. Для коректної роботи сервісів боту необхідне дотримання мінімальних вимог:

1. Процесор з підтримкою x64 архітектури, 4 ядра, 2.0ГГц.
2. Об’єм оперативної пам’яті у розмірі 8Гб.
3. Використання носія для збереження даних розміром 20Гб.
4. Стабільне підключення до мережі Інтернет зі швидкістю 30мбіт/с.

2.4 Модель прецедентів

На основі сформованих функціональних вимог було створено відповідну діаграму прецедентів, яку зображено на (Рис. 2.1)

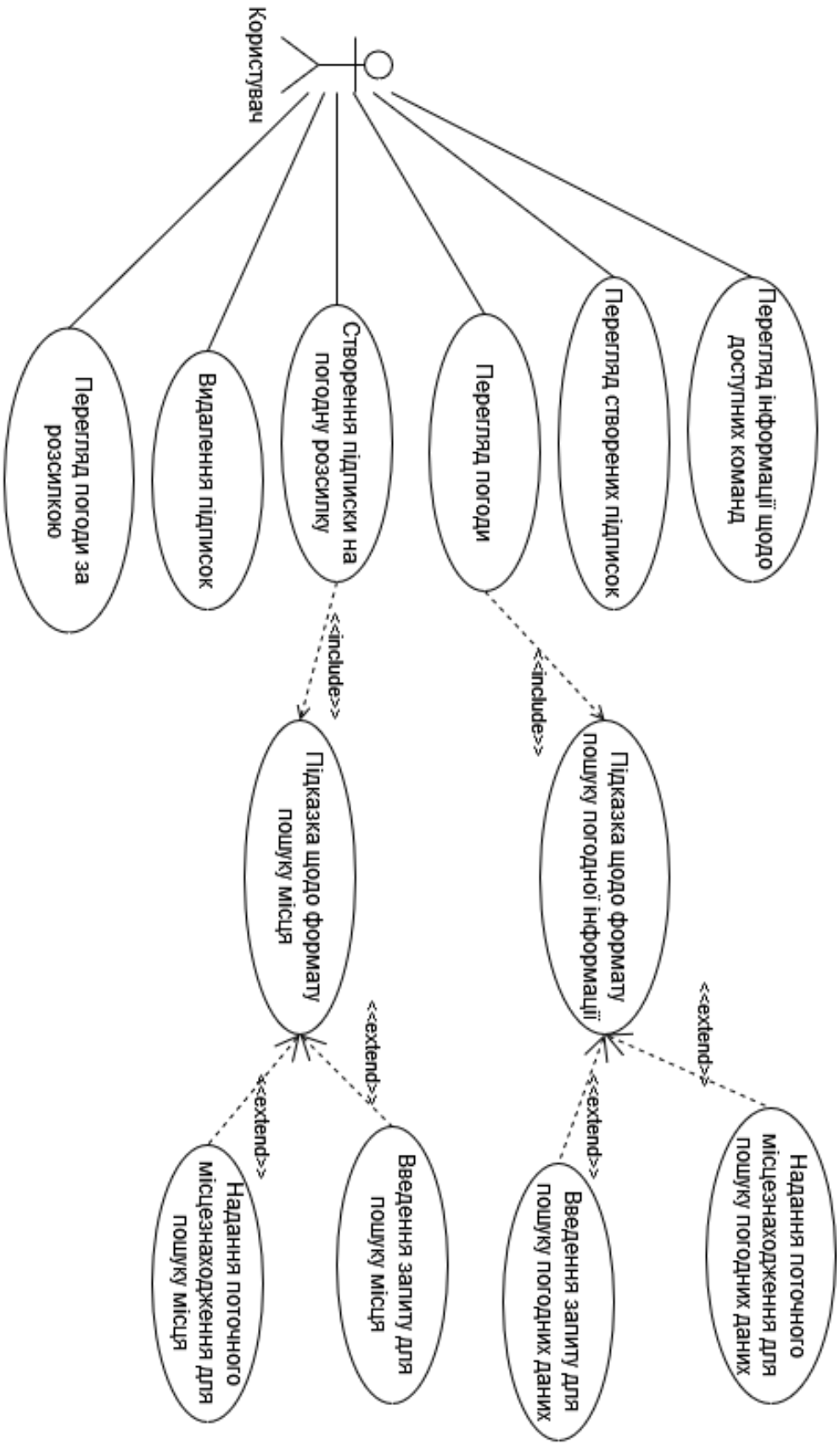


Рис. 2.1 Діаграма прецедентів

2.5 Контекстна діаграма

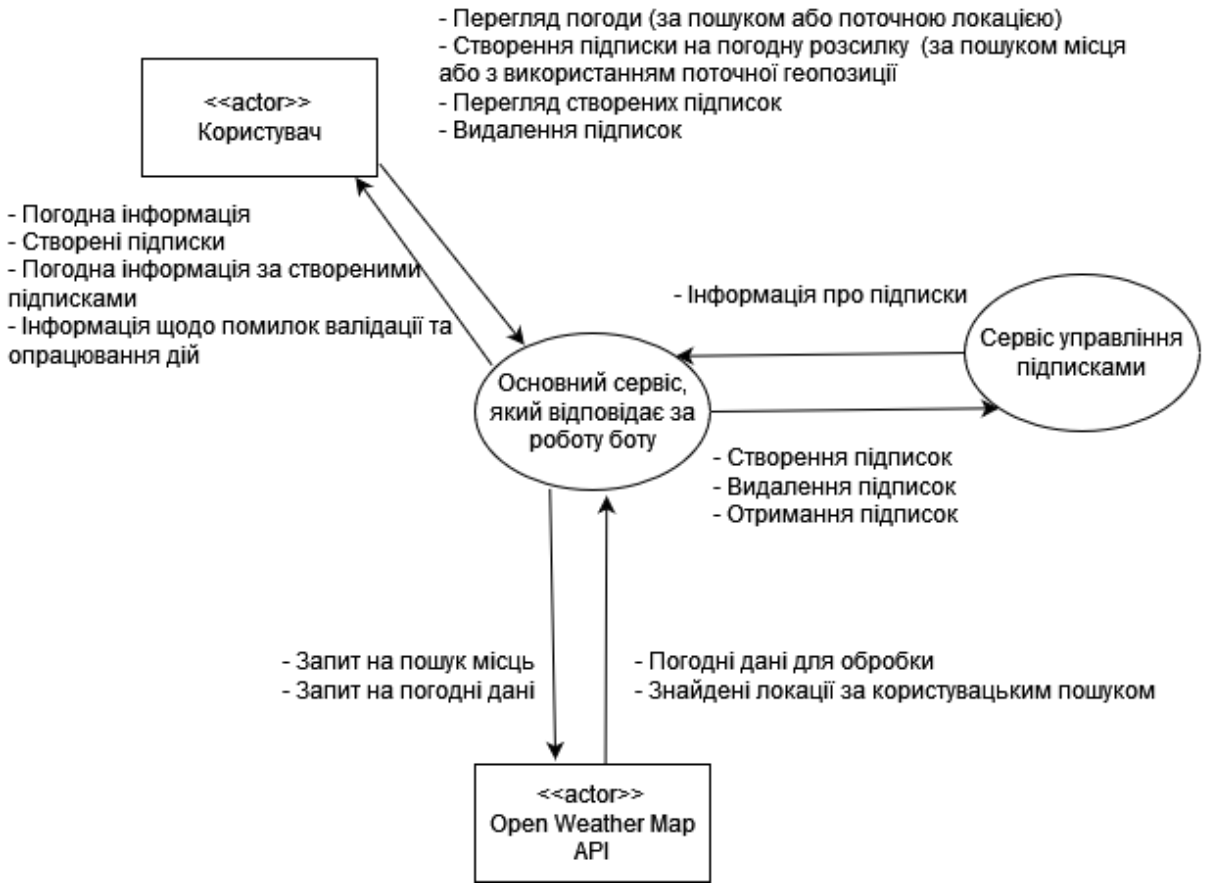


Рис. 2.2 Контекстна діаграма

2.6 Проектування діяльності

Для детального розуміння та моделювання важливого функціоналу Telegram-боту були спроектовані приклади діаграм діяльності.

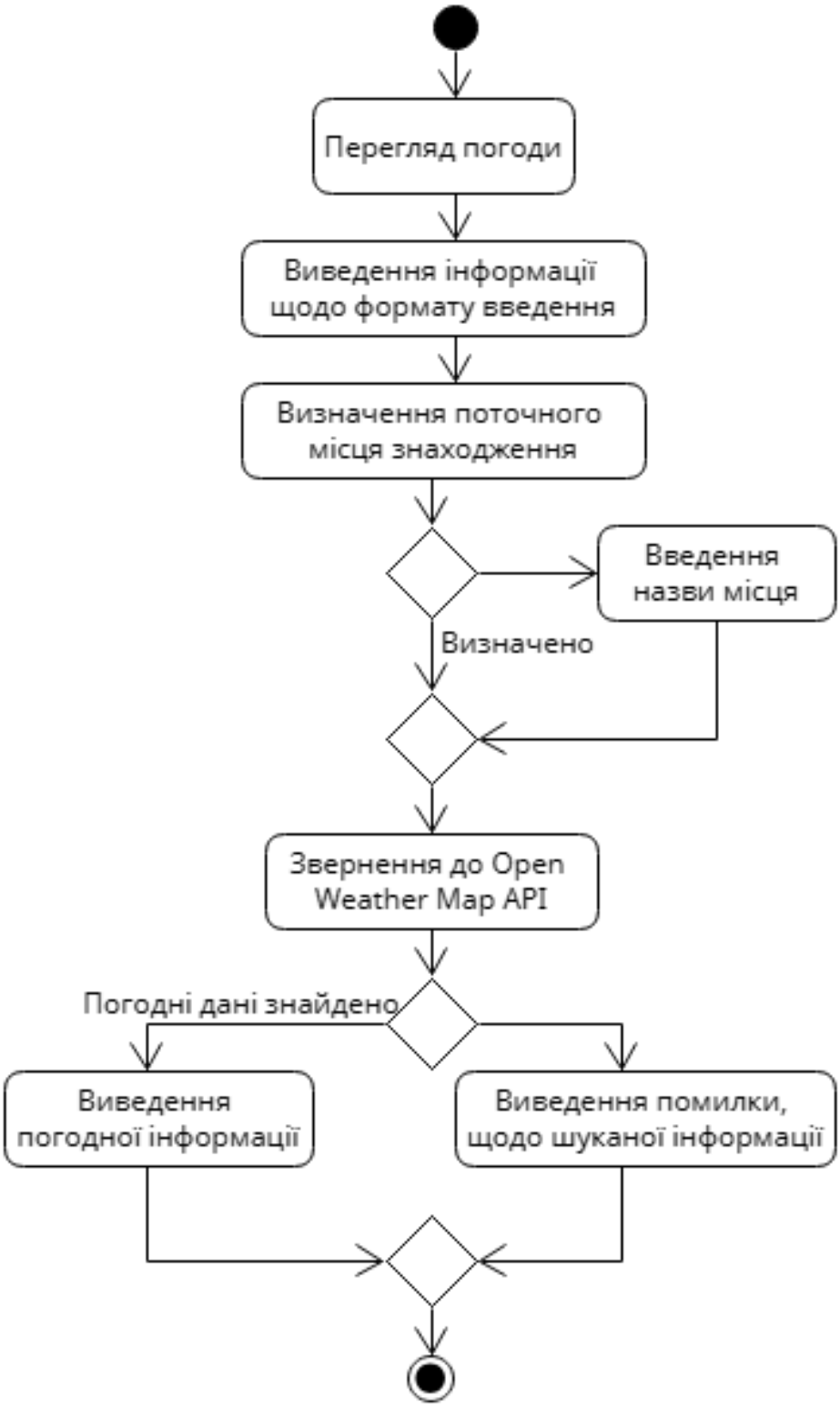


Рис. 2.3 Діаграма діяльності для перегляду актуальної погоди

На рисунку 2.4 зображено метод класу GetWeatherOperation, який є похідним від абстрактного класу Command і відповідає за продовження обробки користувацької команди, що відповідає за перегляд погодної інформації.

```
1 usage  Oleksandr Kozhemiakin
public class GetWeatherOperation(OpenWeatherService weatherService) : Command
{
    0+2 usages  Oleksandr Kozhemiakin
    public override async Task ExecuteAsync(ITelegramBotClient botClient, Update update)
    {
        CurrentWeather? weather;
        var userLocation = update.Message!.Location;

        string weatherQuery;
        if (userLocation is not null)
        {
            var lat = userLocation.Latitude.ToString(CultureInfo.InvariantCulture);
            var lon = userLocation.Longitude.ToString(CultureInfo.InvariantCulture);

            weather = await weatherService.GetCurrentWeather(lon, lat);
            weatherQuery = $"широта: {lat} \nдовгота: {lon}";
        }
        else
        {
            weatherQuery = update.Message.Text ?? string.Empty;
            var weatherLocations :List<Location>? = await weatherService.GetLocations(weatherQuery);
            if (weatherLocations is null || weatherLocations.Count < 1)
            {
                await CommandUtils.SendLocationErrorMessage(botClient, update);
                return;
            }
```

Рис. 2.4 Метод ExecuteAsync у класі GetWeatherOperation

На діаграмі, що наведено на рисунку 2.5, зображено діяльність перегляду підписок.

Рис. 2.5 Діаграма діяльності перегляду створених користувачем підписок на погодну розсилку

На рисунку 2.6 зображено клас, що відповідає за опрацювання користувацької команди щодо перегляду створених ним підписок.

Рис. 2.6 Клас GetSubscriptionCommand

Клас GetSubscriptionCommand також звертається до класу - посередника, що забезпечує спілкування з сервісом управління користувацькими підписками.

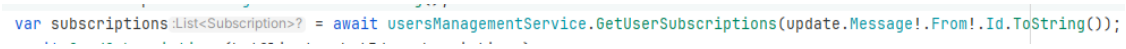


Рис. 2.7 Звернення до класу посередника для отримання підписок користувача

2.7 Проектування класів

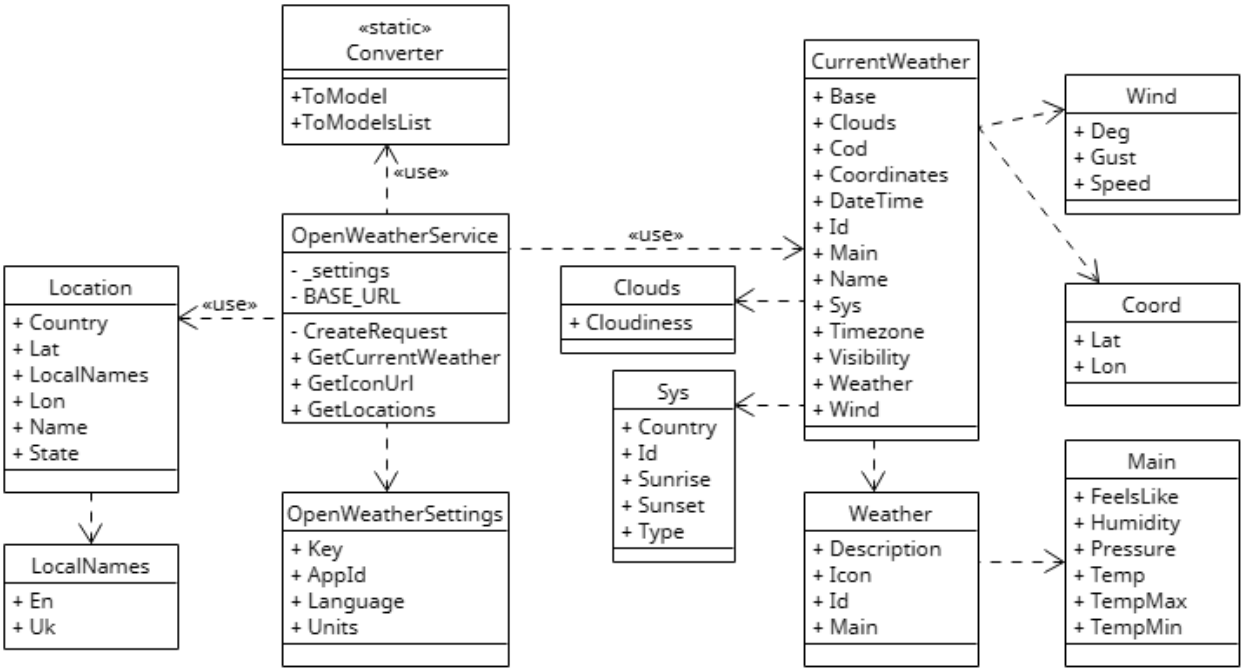


Рис. 2.8 Діаграма класів бібліотеки класів OpenWeather

Для коректної десеріалізації даних, що отримуємо при зверненні до Open Weather Map було спроектовано відповідні класи (рисунок 2.8). Клас OpenWeatherService містить необхідні методи, які потрібні для отримання погодних даних від Open Weather (рисунок 2.9).

```
6 usages  Oleksandr Kozhemiakin
public class OpenWeatherService(RequestSender sender, IOptions<OpenWeatherSettings> options)
{
    private readonly OpenWeatherSettings _settings = options.Value;
    private const string BASE_URL = "https://api.openweathermap.org";

    3 usages  Oleksandr Kozhemiakin
    public Task<CurrentWeather?> GetCurrentWeather(string? lon, string? lat)
    {
        var parameters = new List<Parameter>
        {
            new("lat", lat),
            new("lon", lon),
            new("units", _settings.Units),
            new("lang", _settings.Language)
        };


        var response = sender.SendRequest(CreateRequest(parameters, method: "/data/2.5/weather")).Result.ToJToken();
        return Task.FromResult(response?.ToModel<CurrentWeather>());
    }

    2 usages  Oleksandr Kozhemiakin
    public Task<List<Location>?> GetLocations(string? query)
    {
        var parameter = new Parameter("q", query);
        var response = sender.SendRequest(CreateRequest(parameters: new List<Parameter>{parameter}, method: "/geo/1.0/direct")).Result.ToJToken();
        if (response is JArray array)
        {
```

Рис. 2.9 Клас OpenWeatherService

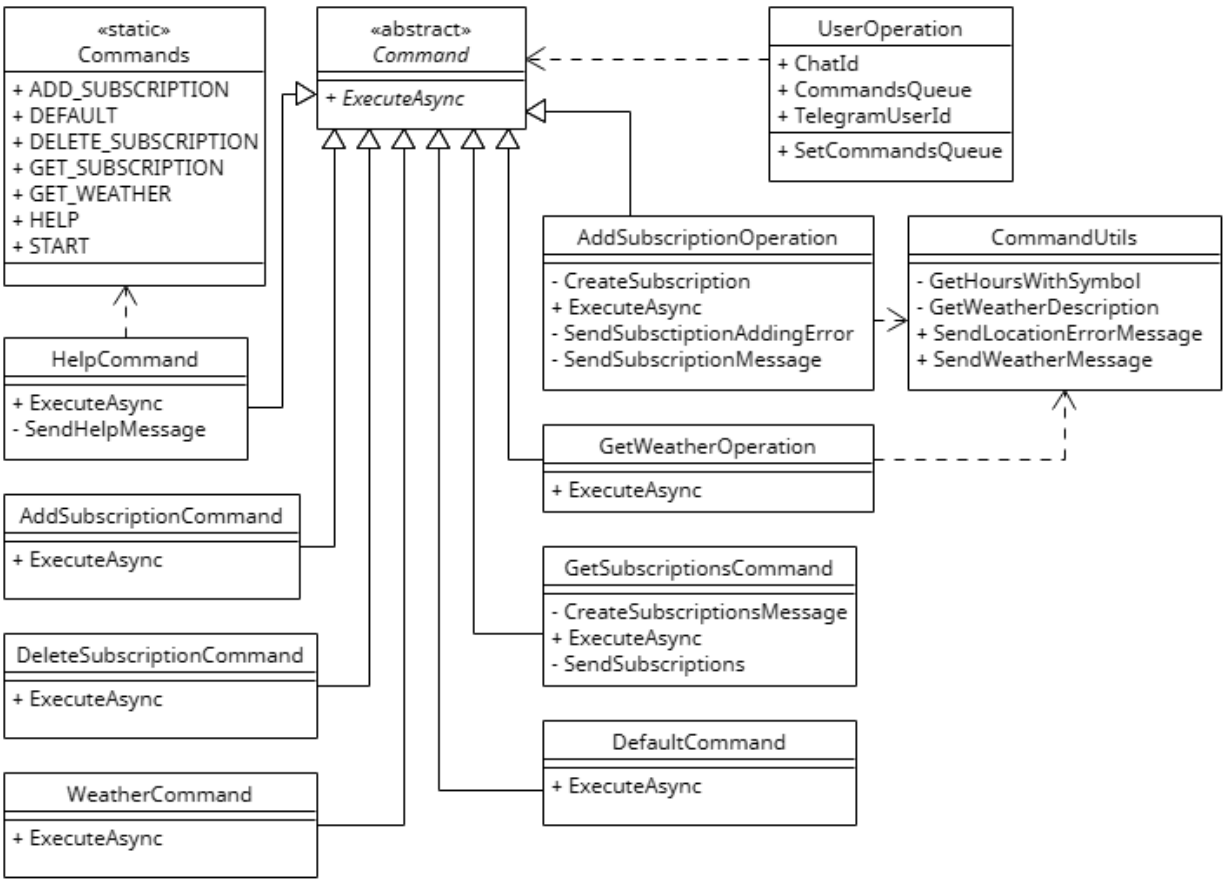


Рис. 2.10 Діаграма класів команд та операцій у TelegramWeatherBot.Server

На рисунку 2.10 зображено діаграму класів, спроектовані для обробки користувацьких команд, які надходять до боту. Кожна команда наслідує абстрактний клас Command, який містить абстрактний метод для виконання певної логіки. Кожен клас – нащадок має надати реалізацію абстрактного методу (рисунок 2.11).

```
2 usages  Oleksandr Kozhemiakin
public class HelpCommand : Command
{
    0+2 usages  Oleksandr Kozhemiakin
    public override async Task ExecuteAsync(ITelegramBotClient botClient, Update update)
    {
        await SendHelpMessage(botClient, update);
    }

    1 usage  Oleksandr Kozhemiakin
    private static async Task SendHelpMessage(ITelegramBotClient botClient, Update update)
    {
        var updateMessage = update.Message;
        var botName :string = botClient.GetMyNameAsync().Result.Name;
```

Рис. 2.11 Приклад класу – нащадку

2.8 Модель проектування

2.8.1 Структура програмного застосунку

Результатом розробки є Telegram – бот для перегляду прогнозу погоди. Він складається з двох мікросервісів, кожен з яких виконує свою роль. Основним сервісом є TelegramWeatherBot.Server, другим мікросервісом є UsersManagement. Основний сервіс обробляє користувацькі дії, команди, запити та відправляє повідомлення користувачу. Другий сервіс використовується для взаємодії із СУБД щоб виконувати керування користувацькими підписками щодо погодних розсилок. Для забезпечення необхідного коректного функціонування та уникнення ситуацій з дублюванням програмного коду, створені допоміжні бібліотеки класів, які містять допоміжні методи.

В таблиці 2.8.1 наведено структуру програмного продукту з коротким описом.

Таблиця 2.1

Структура програмного рішення

Назва елементу	Опис
TelegramWeatherBot.Server	Відповідає за обробку інформації, що надходить від боту
UsersManamement	Сервіс, що відповідає за управління користувацькими підписками
UsersManagement.BAL	Другий рівень UsersManagement, який відповідає за бізнес – логіку, яка стосується підписок
UsersManagement.DAL	Третій рівень UsersManagement, який взаємодіє з базою даних для виконання відповідних запитів
HttpHelpers	Бібліотека класів, яка містить набір методів для полегшення виконання HTTP запитів
OpenWeather	Бібліотека класів, що використовується для спілкування з API Open Weather. Містить необхідні методи та класи - моделі
Tests	Налічує модульні тести для перевірки окремих частин

2.8.2 Проектування розгортання системи

Розроблюване програмне рішення складається з декількох мікросервісів, кожен з яких виконує свою роль у забезпеченні роботи Telegram-боту.

Кожен мікросервіс є окремим застосунком, який не обов’язково повинен бути запущений на одному комп’ютері чи сервері. Для забезпечення надійної роботи Telegram-боту було спроектовано діаграму розгортання (рисунок 2.12), на якій зображено варіант розгортання систем застосунку.

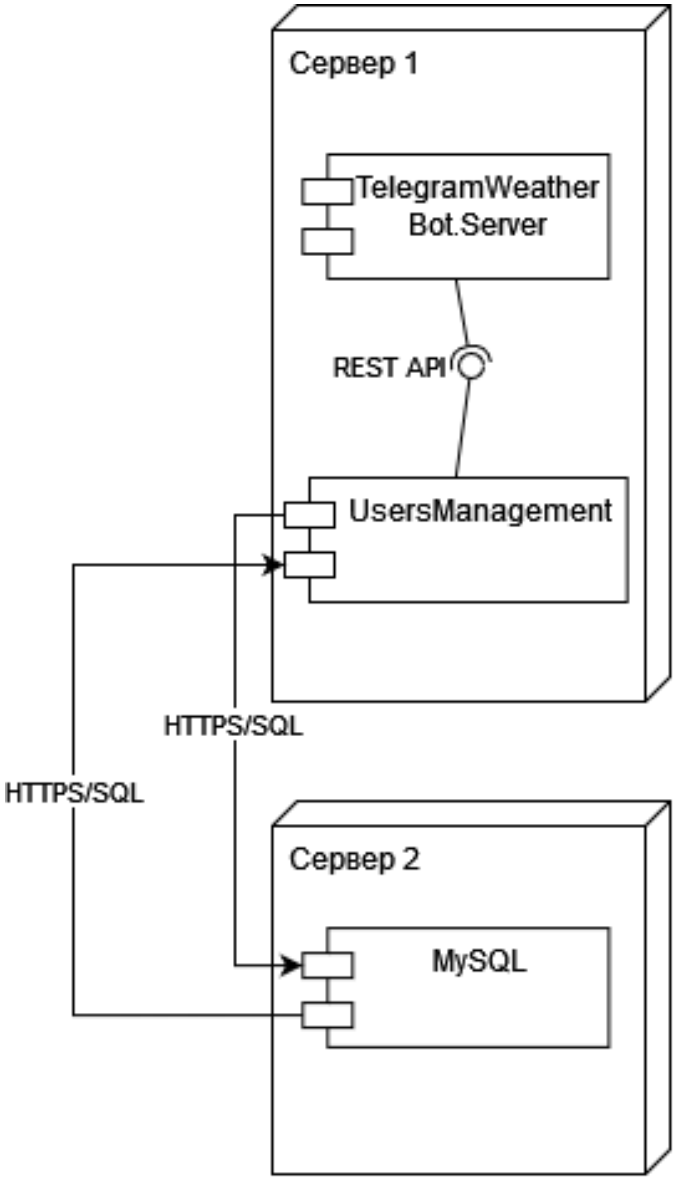


Рис. 2.12 Діаграма розгортання

Мікросервіси можна розгорнути у вигляді контейнерів у Docker. Файл конфігурації розгортання сервісу зображено на рисунку 2.13

```
FROM mcr.microsoft.com/dotnet/aspnet:8.0 AS base
USER $APP_UID
WORKDIR /app
EXPOSE 8080
EXPOSE 8081

FROM mcr.microsoft.com/dotnet/sdk:8.0 AS build
ARG BUILD_CONFIGURATION=Release
WORKDIR /src
COPY ["UsersManagement/UsersManagement.csproj", "UsersManagement/"]
RUN dotnet restore "UsersManagement/UsersManagement.csproj"
COPY . .
WORKDIR "/src/UsersManagement"
RUN dotnet build "UsersManagement.csproj" -c $BUILD_CONFIGURATION -o /app/build

FROM build AS publish
ARG BUILD_CONFIGURATION=Release
RUN dotnet publish "UsersManagement.csproj" -c $BUILD_CONFIGURATION -o /app/publish /p:UseAppHost=false

FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .
ENTRYPOINT ["dotnet", "UsersManagement.dll"]
```

Рис. 2.13 Приклад Dockerfile для розгортання застосунку у контейнері

2.8.3 Проектування архітектури

Розробка програмного застосунку передбачає використання певної архітектури, необхідної для забезпечення структурованості вихідного програмного забезпечення.

Розроблюваний Telegram-бот є комплексним рішенням, який задовольняє визначенні вимоги. Для візуального представлення взаємозв’язків різних компонентів застосунку, було спроектовано діаграми пакетів (рисунки 2.14-2.15), для двох мікросервісів, які забезпечують роботу боту.

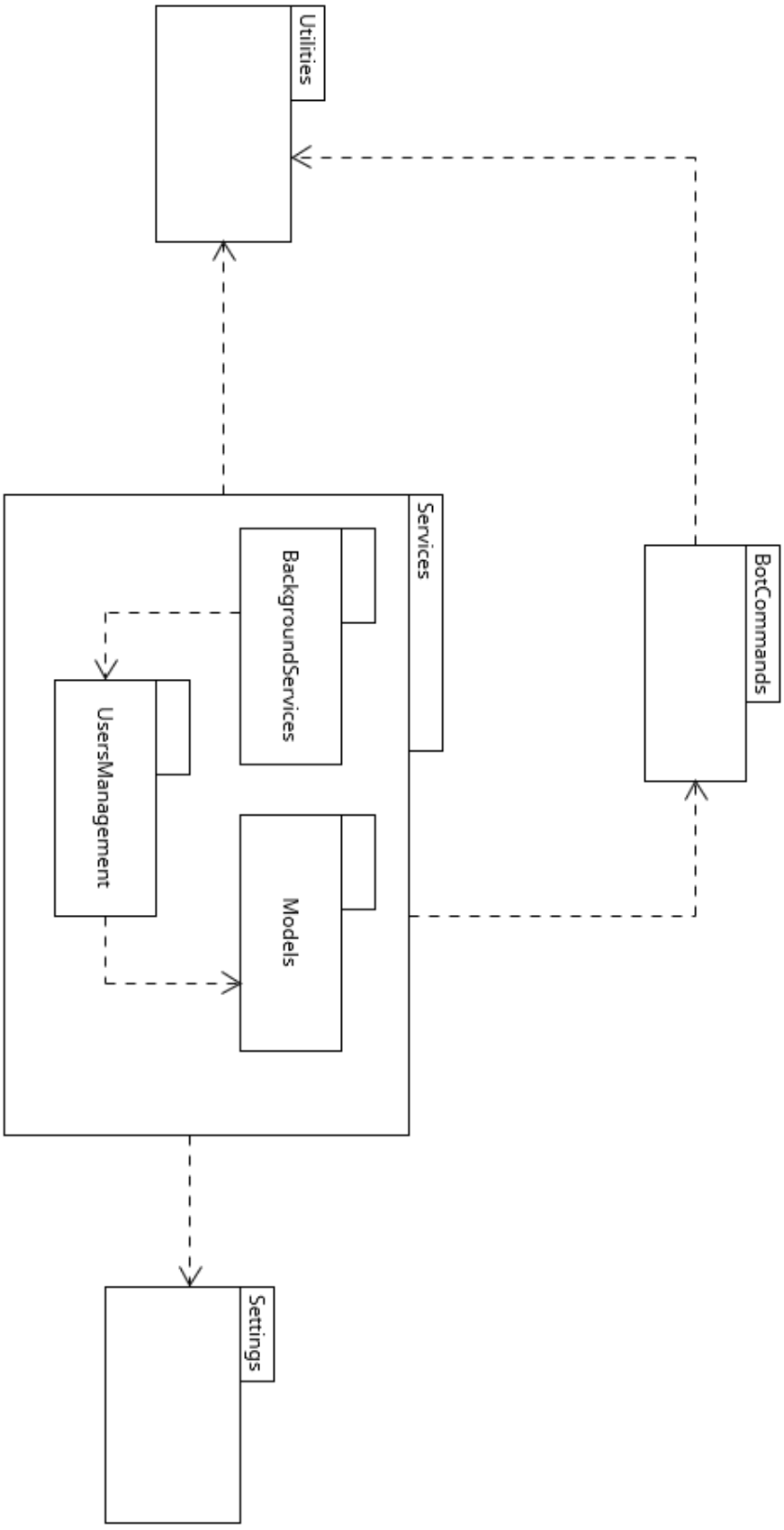


Рис. 2.14 Діаграма пакетів мікросервісу TelegramWeatherBot.Server

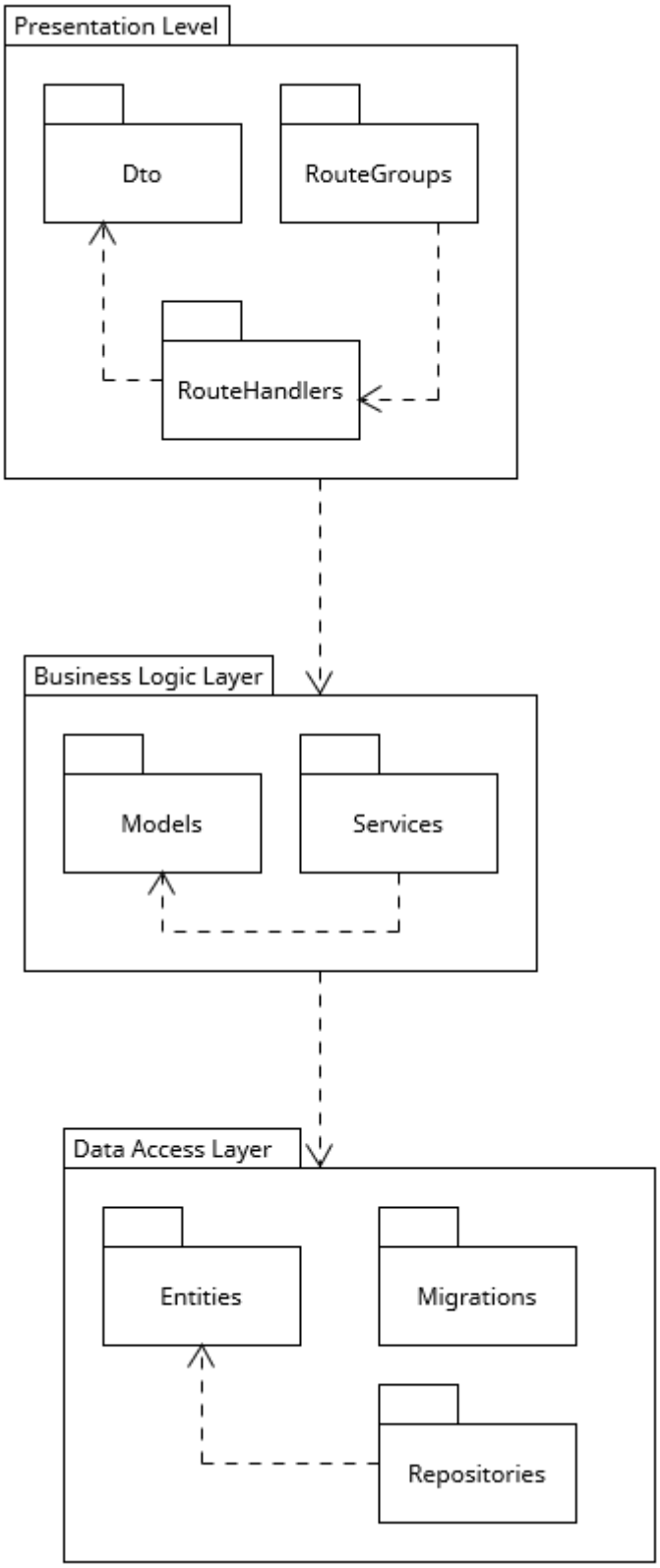


Рис. 2.15 Діаграма пакетів мікросервісу UsersManagement

2.9 Проектування бази даних

Для забезпечення збереження та обробки даних щодо користувацьких підписок було використано СУБД MySQL в якій створено базу даних та відповідні таблиці:

1. Subscriptions. Містить дані щодо користувача, якому належить створена підписка для полегшення ідентифікування належності. Також наявна інформація щодо самої підписки. Назва місця, яке надає API Open Weather, та координати для отримання погодних даних.
2. __EFMigrationsHistory. Системна таблиця, яка генерується автоматично. Ця таблиця необхідна для зберігання створених міграцій і визначення змін у структурі бази даних.

На рисунку 2.16 зображено діаграму бази даних зі структурою створених таблиць

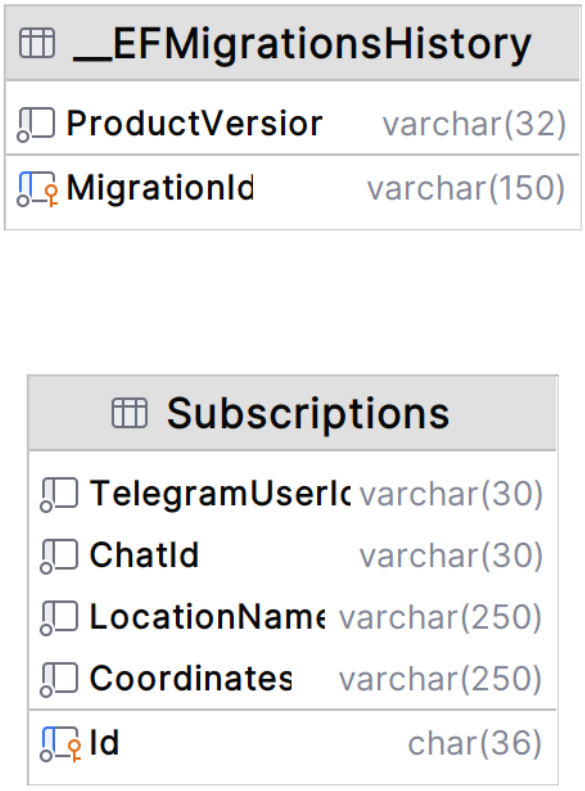


Рис. 2.16 Діаграма бази даних

3 РЕАЛІЗАЦІЯ TELEGRAM – БОТУ

3.1 Використання шаблону Options

Шаблон Options (Налаштування) забезпечує спосіб централізованого управління конфігурацією додатка, дозволяючи розробникам легко отримувати налаштування з різних джерел конфігурації.

Для впровадження цього шаблону в застосунках на платформі .NET використовується бібліотека Microsoft.Extensions.Options.

Насамперед цей шаблон зручно використовувати для швидкої зміни необхідних налаштувань, не змінюючи при цьому константи або властивості у самих класах.

```
{
  "commonSettings": {
    "botToken": "botToken",
    "usersManagementServiceHost": "https://localhost:44386/users-management/subscriptions",
    "SubscriptionsSyncTimeoutInMinutes": 150
  },
  "openWeather": {
    "AppId": "appid",
    "Language": "ua",
    "Units": "metric"
  }
}
```

Рис. 3.1 Файл конфігурації у форматі JSON для налаштування Telegram – боту

На рисунку 3.1 наведено приклад конфігураційного файлу у форматі JSON, який використовується для реалізації шаблону Options. У даному випадку файл конфігурації містить два об’єкти, які мають певний набір значень.

CommonSettings містить значення, що стосуються загальних параметрів:

1. botToken – унікальне значення, яке надається під час створення боту у месенджері Telegram за допомогою BotFather (рисунок 3.2)

2. usersManagementServiceHost – адреса сервісу, що відповідає за управління підписками користувачів.

3. subscriptionSyncTimeoutInMinutes – проміжок часу у хвилинах, через який відбувається відправлення погодних даних за створеними підписками.

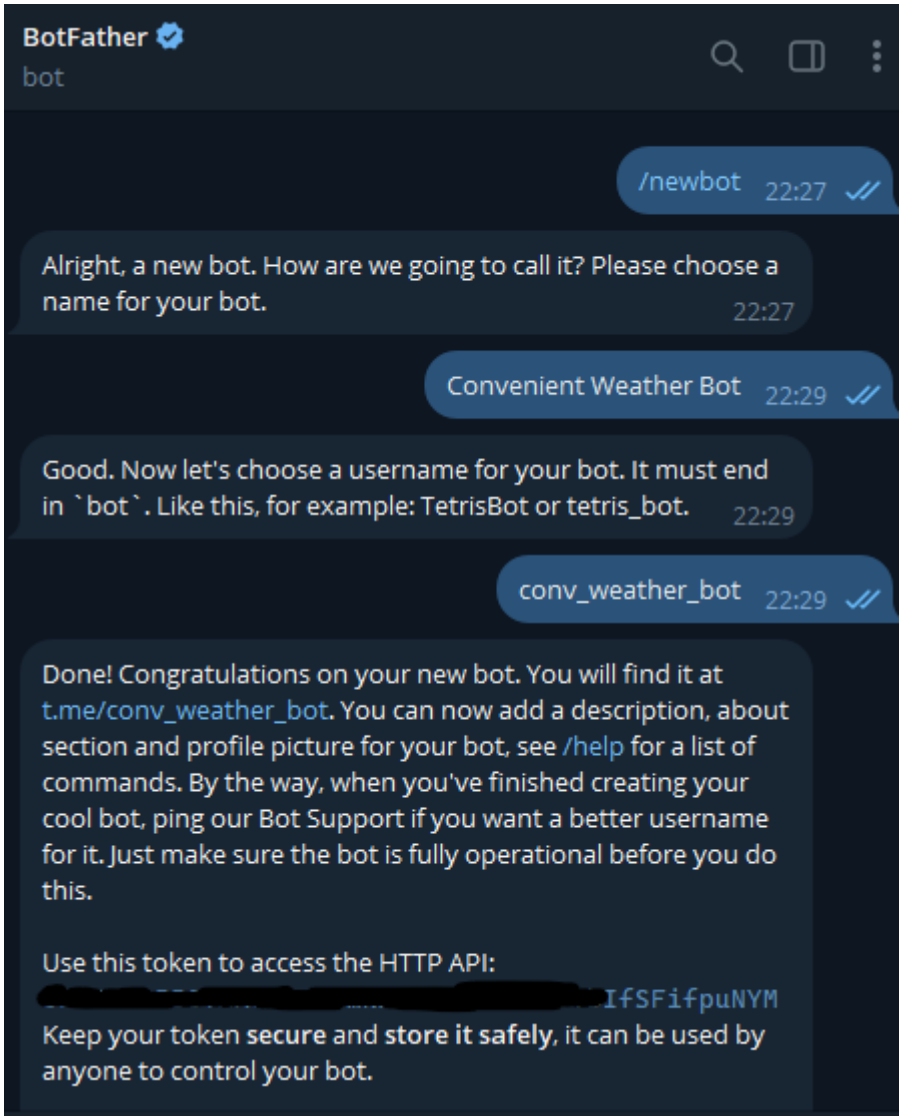


Рис. 3.2 Створення боту та отримання токену

OpenWeather зберігає значення, які необхідні для коректної обробки запитів отримання погодної інформації від Open Weather API

Бібліотека надає всі необхідні засоби для впровадження шаблону. Для коректного зчитування параметрів з файлу конфігурації необхідно створити клас, який буде відповідати за свій набір властивостей. Клас, що використовується для збереження CommonSettings зображено на рисунку 3.3

```
8 usages  Oleksandr Kozhemiakin
public class CommonSettings
{
    public const string KEY = "commonSettings";
    4 usages
    public string UsersManagementServiceHost { get; set; } = string.Empty;
    1 usage
    public string BotToken { get; set; } = string.Empty;
    1 usage
    public int SubscriptionsSyncTimeoutInMinutes { get; set; } = 10;
}
```

Рис. 3.3 Клас CommonSettings, який зберігає загальні параметри

Для уникнення виключень, використовуються властивості зі стандартними значеннями, які будуть використані у разі відсутності відповідних записів у файлі конфігурацій.

```
const string configPath = "config.json";
var configuration = new ConfigurationBuilder()
    .AddJsonFile(configPath, optional: false) // IConfigurationBuilder
    .Build();
```

Рис. 3.4 Запис файлу конфігурацій у пам'ять

На рисунку 3.4 зображено приклад коду, який використовується для збереження даних файлу конфігурації у пам’ять. На рисунках 3.5 – 3.6 зображено створення екземплярів класів, на основі даних з файлу конфігурації та використання впровадженого шаблону Options у інших класах відповідно.

```
1 usage  Oleksandr Kozhemiakin
public static IServiceCollection AddSettings(this IServiceCollection collection, IConfiguration configuration)
{
    collection.Configure<CommonSettings>(configuration.GetSection(CommonSettings.KEY));
    collection.Configure<OpenWeatherSettings>(configuration.GetSection(OpenWeatherSettings.Key));

    return collection;
}
```

Рис. 3.5 Створення екземплярів класів з даними із файлу конфігурацій

```
6 usages  Oleksandr Kozhemiakin
public class UsersManagementService(RequestSender sender, IOptions<CommonSettings> settings)
{
    private readonly CommonSettings _settings = settings.Value;
```

Рис. 3.6 Використання впровадженого шаблону Options

3.2 Отримання погодної інформації

Під час розробки програмного засобу було використано API від Open Weather Map для отримання та обробки погодних даних. Open Weather надає різноманітні API для різних сфер використання. Головною перевагою використання API від Open Weather є можливість безкоштовного використання для отримання актуальних погодних даних.

API від Open Weather (рисунок 3.7) побудовано з використанням підходу REST, а тип даних, що використовується для обміну даними є JSON.

Call current weather data

How to make an API call

API call

```
https://api.openweathermap.org/data/2.5/weather?lat={lat}
&lon={lon}&appid={API key}
```

Parameters

`lat`	required	Latitude. If you need the geocoder to automatic convert city names and zip-codes to geo coordinates and the other way around, please use our Geocoding API
`lon`	required	Longitude. If you need the geocoder to automatic convert city names and zip-codes to geo coordinates and the other way around, please use our Geocoding API
`appid`	required	Your unique API key (you can always find it on your account page under the "API key" tab)
`mode`	optional	Response format. Possible values are `xml` and `html`. If you don't use the `mode` parameter format is JSON by default. Learn more
`units`	optional	Units of measurement. `standard`, `metric` and `imperial` units are available. If you do not use the `units` parameter, `standard` units will be applied by default. Learn more
`lang`	optional	You can use this parameter to get the output in your language. Learn more

Рис. 3.7 Приклад документації щодо використання API від Open Weather Map

Щоб реалізувати отримання та обробку погодних даних було спроектовано та розроблено бібліотеку класів OpenWeather, яка містить необхідні класи та методи для опрацювання.

Для того щоб отримати погодні дані від API необхідно відповідно до документації сформувати та надіслати HTTP запит, у відповідь на який буде надано

JSON із даними. Код для створення та відправки запиту на отримання погодної інформації зображено на рисунку 3.8

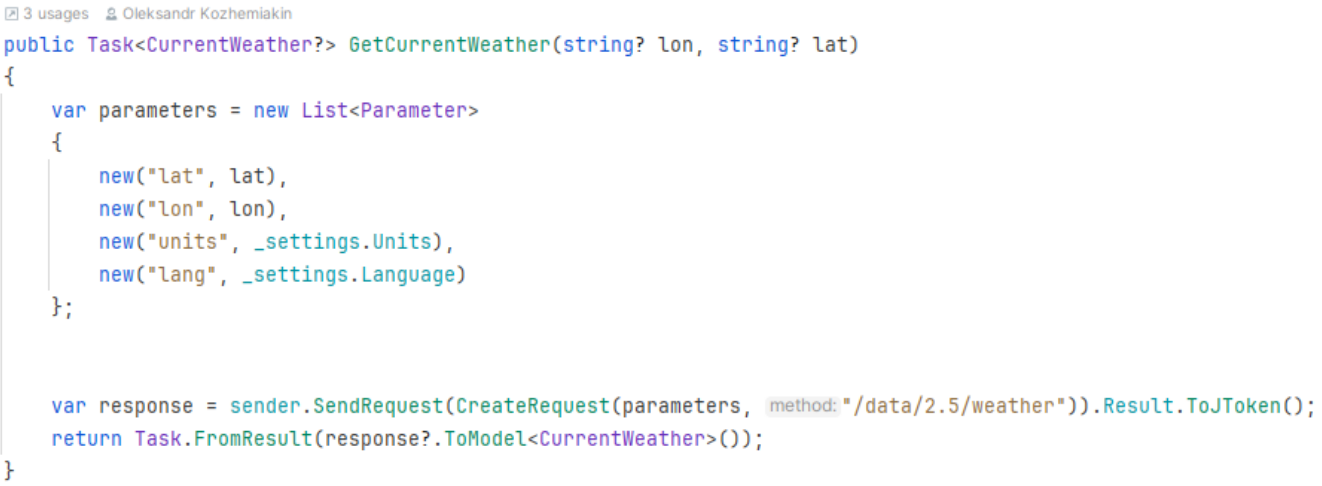


Рис. 3.8 Метод отримання погодних даних від Open Weather Map

Конвертація у класи C# відбувається за допомогою десеріалізації. Необхідні методи надає бібліотека Newtonsoft.JSON. Приклад використання бібліотеки для десеріалізування JSON з результату на запит продемонстровано на рисунку 3.9

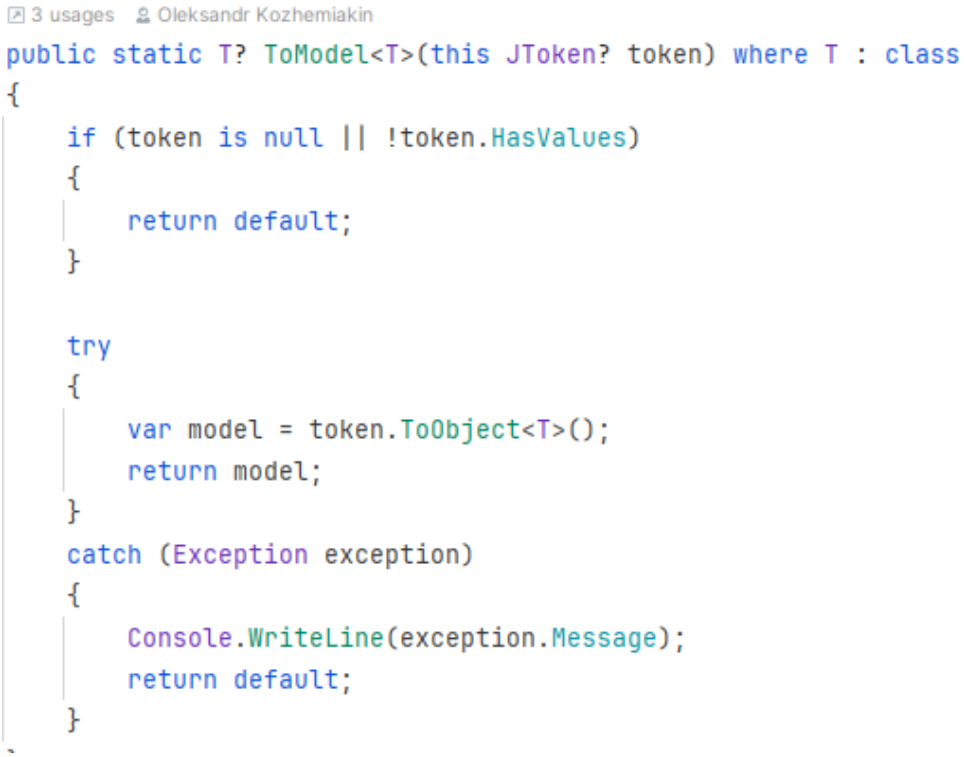


Рис. 3.9 Використання бібліотеки Newtonsoft.JSON

3.3 Обробка користувацьких команд

Для взаємодії з ботом використовуються команди, які реалізують функції, описані у вимогах. Користувачеві доступні такі команди:

1. /help – перегляд допомоги щодо функцій боту.
2. /start – стартує використання боту для користувача та відображає повідомлення щодо функцій боту.
3. /weather – перегляд погодних даних з використанням пошукового запиту або поточного місцезнаходження.
4. /addSubscription – створення підписки на погодну розсилку, використовуючи запит на локацію або поточне місцезнаходження користувача.
5. /subscription – перегляд створених підписок на погодну розсилку.
6. /removeSubscription – видаляє створені підписки, якщо користувач раніше їх створював

Обробка команд користувачем відбувається за рахунок методів класу CommandsCoordinator (рисунок 3.10)

```
3 usages  Oleksandr Kozhemiakin
public class CommandsCoordinator(CommandsService commandsService)
{
    private readonly Dictionary<string, List<Command>> _commands = commandsService.GetCommands();
    private readonly List<UserOperation> _usersOperations = [];

    1 usage  Oleksandr Kozhemiakin
    public Task StartCommandsExecution(ITelegramBotClient botClient, Update update)
    {
        var message = update.Message;
        if (message is null)
        {
            return Task.CompletedTask;
        }

        var userId = message.From?.Id.ToString();
        if (string.IsNullOrEmpty(userId))
        {
            return Task.CompletedTask;
        }

        var chatId = message.Chat.Id.ToString();
```

Рис. 3.10 Клас CommandsCoordinator

Такі команди як /weather передбачає введення даних від користувача у вигляді пошукового запиту або поточного місцезнаходження (рисунок 3.11 – 3.12)

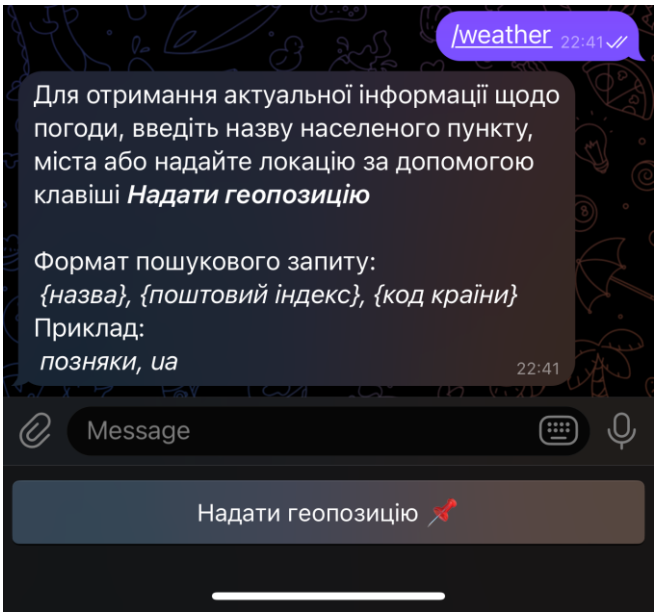


Рис. 3.11 Виклик команди /weather

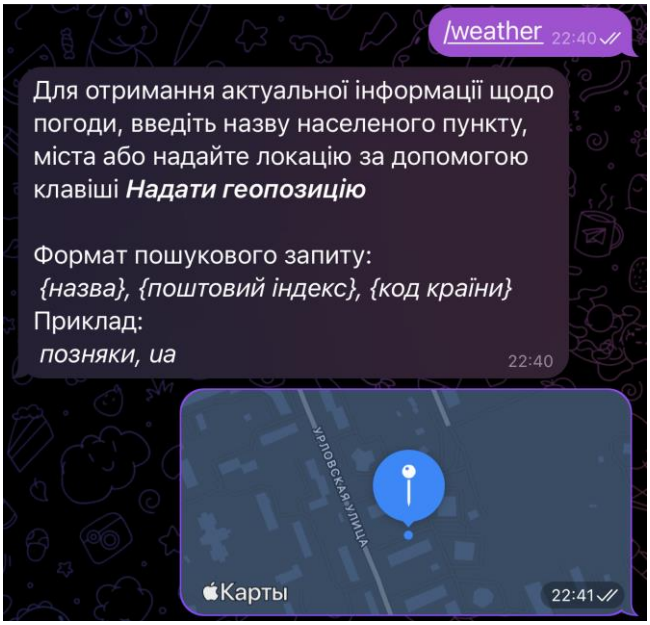


Рис. 3.12 Надання поточного місцезнаходження

Для послідовної обробки дій користувача було реалізовано механізм, який зберігає останню команду, який вводив певний користувач. Відбувається перевірка

чи бажає користувач продовжити виконання минулої дії або виконати нову команду. Перевірка зображена у коді на рисунку 3.13

```
var chatId = message.Chat.Id.ToString();
AddUserOperationIfNotExists(userId, chatId);

if (message.Entities?.FirstOrDefault()?.Type is not MessageEntityType.BotCommand)
{
    return ExecuteCommand(botClient, update, userId);
}

var commandName :string = message.EntityValues!.First();
return ExecuteCommand(botClient, update, userId, commandsKey: commandName);
```

Рис. 3.13 Перевірка на бажання користувача продовжити виконання попередньої команди

Щоб правильно обробити послідовне виконання команди, використаний механізм черги. Завдяки цьому механізму з’являється можливість покрокового виклику методів, які знаходяться у черзі. Якщо у черзі не залишилось додаткових операцій, то команда повністю опрацьована. Таким чином можна створювати складі команди з багатьма кроками, які потребують введення різних даних.

На рисунку 3.14 зображено метод, який використовує чергу для опрацювання складних команд.

```
var userOperation = GetUserOperation(userId);
if (userOperation is null)
{
    return;
}

if (userOperation.CommandsQueue?.TryDequeue(out var currentCommand) is null or false)
{
    return;
}

await currentCommand.ExecuteAsync(botClient, update);
```

Рис. 3.14 Використання черги

3.4 Використання успадкування

Спадкування (inheritance) є основним принципом об'єктно-орієнтованого програмування (ООР), який дозволяє створювати нові класи на основі існуючих. Це допомагає зменшити дублювання коду та спрощує його підтримку. Класи, які було створені на основі іншого класу називають класом – нащадком.

Цей фундаментальний принцип було використано для реалізації підстановки. Використовуючи підстановку, з’являється можливість використовувати тип базового класу при передачі параметрів або виклику певних методів.

Таким чином було створено базовий абстрактний клас Command, що містить в собі абстрактний метод ExecuteAsync, який обов’язково повинен бути реалізований у класі – нащадку. Кожен нащадок має власну реалізацію, тому при виклику відповідного методу у класі – нащадку, відпрацює потрібна логіка

```
1 usage  Oleksandr Kozhemiakin
public class DeleteSubscriptionCommand(UsersManagementService usersManagementService) : Command
{
    0+2 usages  Oleksandr Kozhemiakin
    public override async Task ExecuteAsync(ITelegramBotClient botClient, Update update)
    {
        var telegramUserId = update.Message!.From!.Id.ToString();
        var chatId :long = update.Message!.Chat.Id;
        var result :bool = await usersManagementService.RemoveSubscriptions(telegramUserId);

        var text :string = result ? "Успішно видалено створені розсилки" : "Розсилок ще не було додано";
        await botClient.SendTextMessageAsync( chatId, text);
    }
}
```

Рис. 3.15 Нащадок абстрактного класу Command з власною реалізацією ExecuteAsync

```
var currentCommand = userOperation.CommandsQueue!.Dequeue();
await currentCommand.ExecuteAsync(botClient, update);
```

Рис. 3.16 Застосування підстановки

3.5 Дотримання принципів REST

Щоб організувати взаємодію з СУБД MySQL для збереження користувацьких підписок було розроблено окремий мікросервіс, який створений на основі фреймворку ASP.NET Core та являє собою REST API. Цей мікросервіс є окремою одиницею та може бути розгорнутий окремо незалежно від основного сервісу, який відповідає за роботу боту. Мікросервіс спроектовано із використанням трирівневої архітектури (рисунок 3.17), що продемонстровано на рисунку 3.18

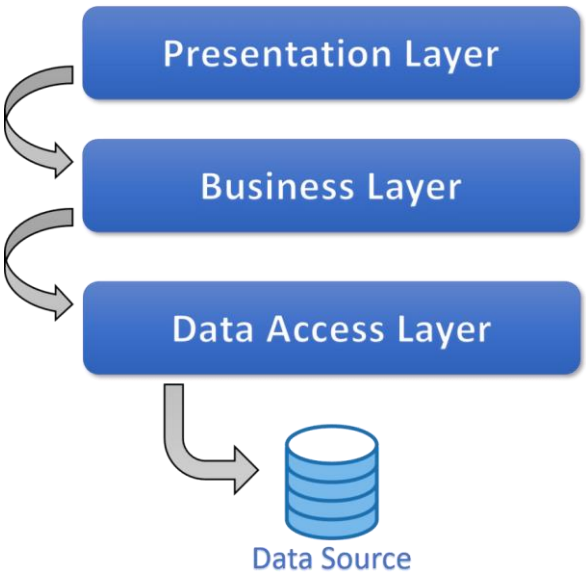


Рис. 3.17 Трирівнева архітектура

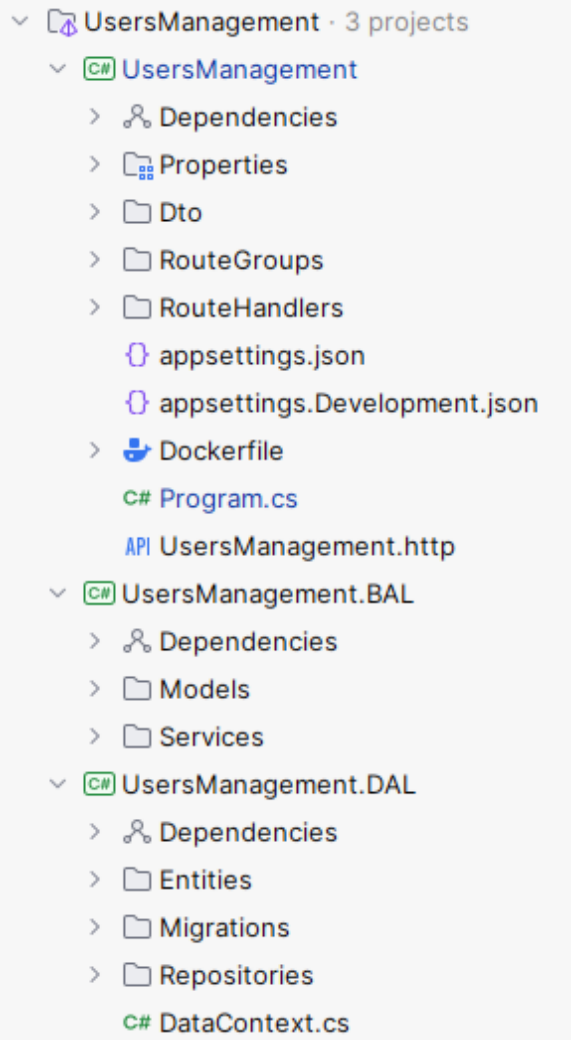


Рис. 3.18 Трирівнева архітектура у мікросервісі UsersManagement

Мікросервіс надає необхідні HTTP методи для роботи з підписками (рисунок 3.19)

```
1 usage  Oleksandr Kozhemiakin  More...
public static RouteGroupBuilder MapSubscriptionsApi(this RouteGroupBuilder group)
{
    group.MapPost( pattern: "/", SubscriptionsHandlers.AddSubscription);
    group.MapDelete( pattern: "/{telegramUserId}", SubscriptionsHandlers.RemoveSubscriptions);
    group.MapGet( pattern: "/", SubscriptionsHandlers.GetSubscriptions);
    group.MapGet( pattern: "/{telegramUserId}", SubscriptionsHandlers.GetUserSubscriptions);
    return group;
}
```

Рис. 3.19 HTTP методи для роботи з підписками

Створені методи використовують сервіси, що створені у другому рівні, який відповідає за необхідну бізнесову логіку (рисунок 3.20).

```
1 usage  Oleksandr Kozhemiakin
public static async Task<IResult> AddSubscription(
    [FromBody, Required] SubscriptionDto subscriptionDto,
    SubscriptionsService service)
{
    var result :bool = await service.CreateSubscription(subscriptionDto.ToBal());
    return result ? TypedResults.Ok() : TypedResults.BadRequest();
}
```

Рис. 3.20 Виклик методів, що знаходяться на другому рівні

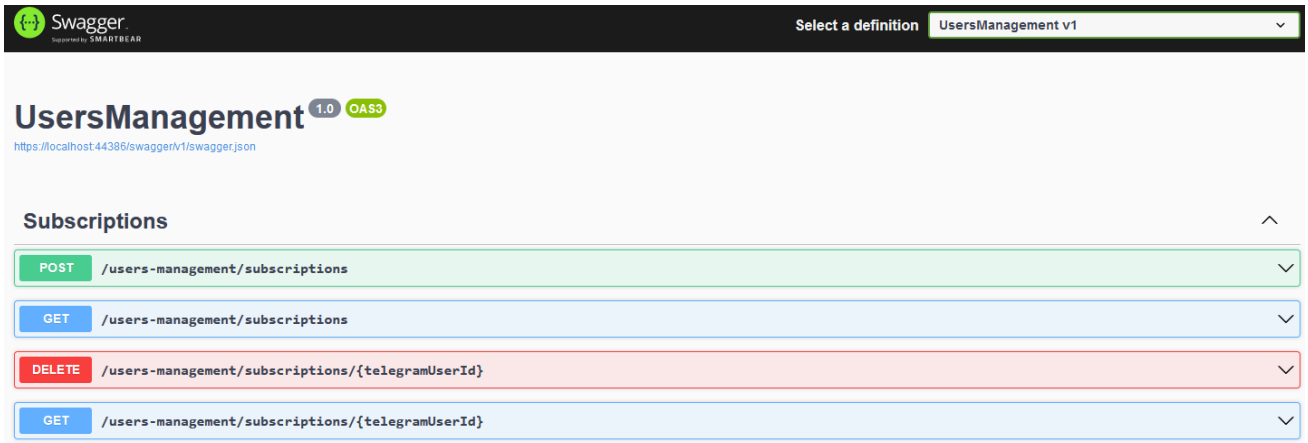


Рис. 3.21 Екранна форма мікросервісу з управління підписками

4 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАСТОСУНКУ

4.1 Модульне тестування

Модульне тестування є критично важливим аспектом розробки програмного забезпечення з кількох причин. Воно забезпечує якість, надійність і стабільність коду, що в кінцевому результаті зменшує витрати на підтримку та розвиток системи.

Модульне тестування проводиться на етапі розробки для виявлення критичних проблем та їх швидкого вирішення. Такий вид тестування забезпечує гнучкість та ізольованість. Завдяки цьому можна відокремити певну частину системи та протестувати її окремі функції.

Для створення модульних тестів при розробці програмого продукту було використано бібліотеку xUnit. Приклад розроблених тестів продемонстровано на рисунку 4.1

```
[Fact]
Oleksandr Kozhemiakin
public void ToModel_ValidToken_ReturnsModel()
{
    // Arrange
    var json = "{ \"Name\": \"John Doe\", \"Age\": 30 }";
    var token = JToken.Parse(json);
    // Act
    var result = token.ToModel<Person>();
    // Assert
    Assert.NotNull(result);
    Assert.Equal(expected: "John Doe", actual: result.Name);
    Assert.Equal(expected: 30, actual: result.Age);
}

[Fact]
Oleksandr Kozhemiakin
public void ToModel_NullToken_ReturnsNull()
{
    // Arrange
    JToken? token = null;
    // Act
    var result = token.ToModel<Person>();
    // Assert
    Assert.Null(result);
}
```

Рис. 4.1 Модульні тести для класу Converter бібліотеки OpenWeather

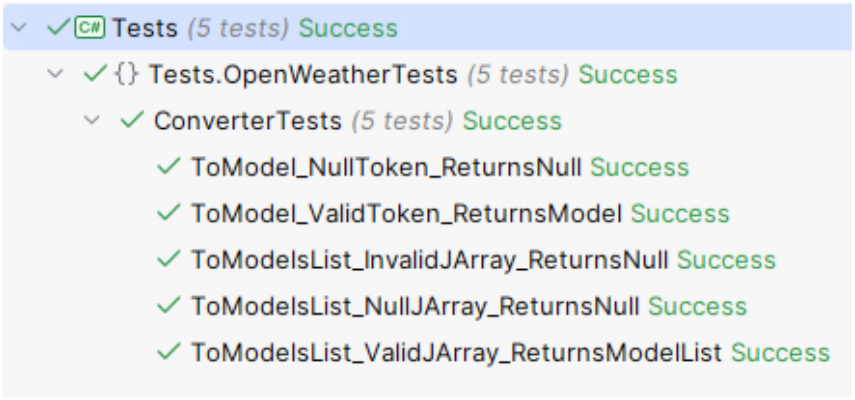


Рис. 4.2 Успішне виконання тестів

4.2 Мануальне тестування

Мануальне тестування є важливим аспектом забезпечення якості програмного забезпечення (QA). Воно включає в себе ручне виконання тестів без

автоматизації, щоб перевірити, чи відповідає програма очікуваним результатам. Мануальне тестування передбачає перевірку функцій застосунку.

Отже, під час мануального тестування було протестовано функції та сценарії використання Telegram – боту. У таблиці 4.1 наведено мануальні тести.

Таблиця 4.1

Мануальні тести Telegram - боту

Опис тесту	Очікуваний результат	Фактичний результат	Статус
Перевірка повідомлення щодо допомоги	Виведення повідомлення з інформацією щодо доступних користувачеві команд	Відображення повідомлення з усіма доступними командами та описом	Успіх
Перевірка перегляду поточної погоди за пошуком	Виведення детальної погодної інформації	Відображення повідомлення з погодною інформацією	Успіх
Перевірка створення підписки на погодну розсилку	Виведення повідомлення, яке вказує на успішне додавання нової підписки	Відображення повідомлення щодо створення нової підписки	Успіх

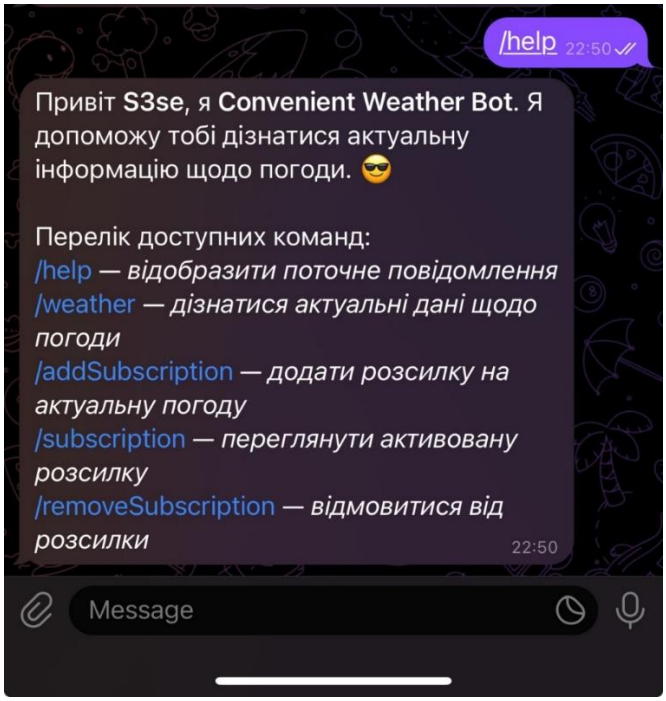


Рис. 4.3 Виклик команди /help

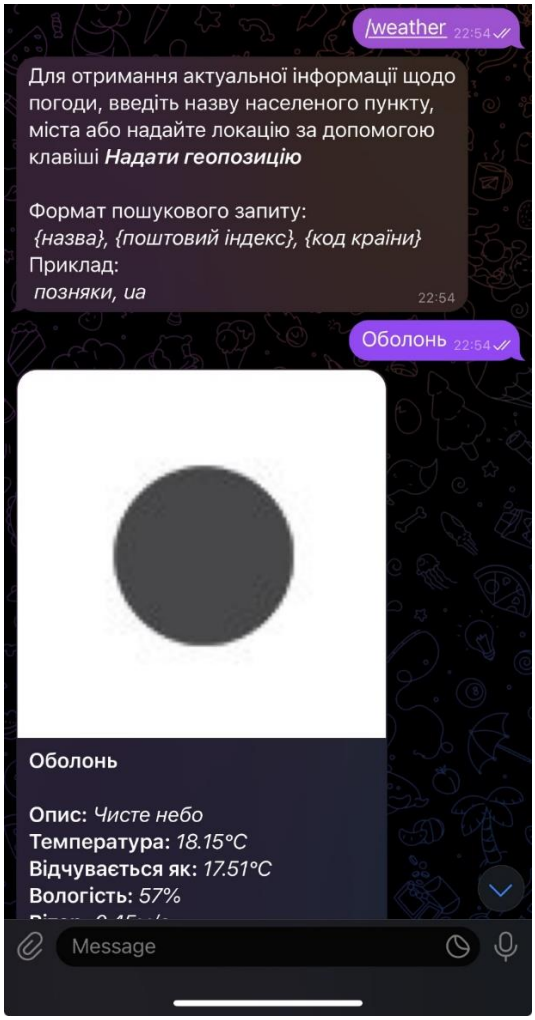


Рис. 4.4 Виклик команди /weather

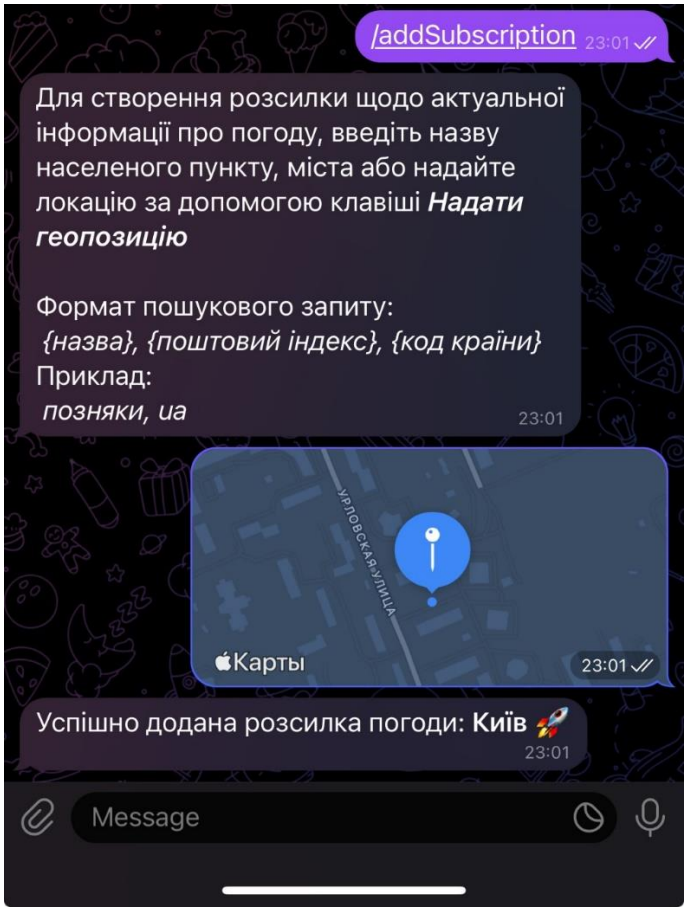


Рис. 4.5 Виклик команди /addSubscription

Перевірка наявності створеної підписки виконанням запиту до СУБД (рисунок 4.6)

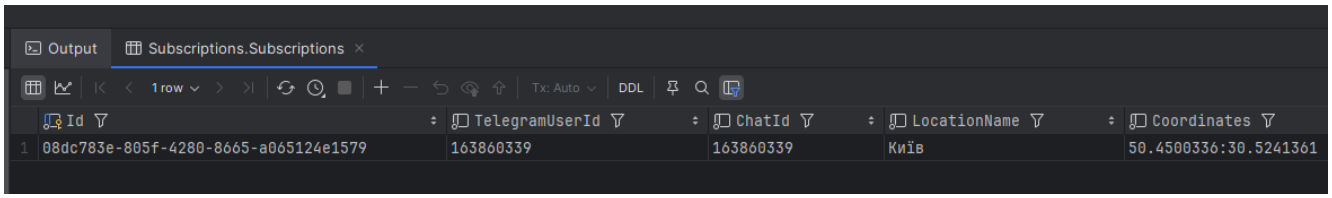


Рис. 4.6 Створена підписка у СУБД

ВИСНОВКИ

У результаті виконання дипломної роботи було спроектовано та розроблено Telegram – бот для перегляду прогнозу погоди. Розробка є актуальною, адже проведений аналіз предметної області підтверджує факт важливості погодної інформації.

1. Проведено аналіз предметної галузі. В результаті аналіз було виявлені важливі аспекти притаманні галузі. Визначені ключові функціональні можливості, які необхідні користувачеві.

2. Здійснено дослідження існуючих аналогічних програмних засобів. Визначено необхідний функціонал, який треба розробити.

3. Визначені ключові функціональні та нефункціональні вимоги.

4. За результатами аналізу було обрано відповідні програмні засоби для реалізації Telegram боту.

5. Визначено місце і роль Telegram-боту в контексті інших акторів.

6. Спроектовано діяльність щодо перегляду погоди та розсилок.

7. Спроектовано розгортання застосунку на обчислювальних вузлах.

8. Спроектовано інтерфейс користувача.

9. Спроектовано та впроваджено механізм отримання погодних даних. Використано API від Open Weather Map, яке створено за принципом REST та надає дані у форматі JSON. Для легкої та швидкої взаємодії з API створено бібліотеку класів та інші допоміжні засоби, що відповідають за взаємодію з API для отримання погодних даних.

10. Розроблено Telegram – бот відповідно до визначених функціональних вимог. У результаті було створено 2 мікросервіси, що відповідають за необхідний функціонал.

11. Тестування програмного застосунку здійснювалося за допомогою модульного тестування та мануального тестування. За результатами модульного та мануального тестування отримано успішне виконання всіх тестів.

ПЕРЕЛІК ПОСИЛАНЬ

1. Кожем’якін О.В., Гаманюк І.М. Аналіз засобів розробки Телеграм-боту для перегляду прогнозу погоди: Матеріали Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в ІКТ». Збірник тез. 24.04.2024, ДУІКТ, м. Київ. К.: ДУІКТ – 173 с.

2. Кожем’якін О.В., Гаманюк І.М. Використання RESTful WEB API для отримання погодної інформації щодо Телеграм-боту для перегляду прогнозу погоди: Матеріали Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в ІКТ». Збірник тез. 24.04.2024, ДУІКТ, м. Київ. К.: ДУІКТ – 175 с.

3. Overview of ASP.NET Core. Microsoft Learn: Build skills that open doors in your career. URL: https://learn.microsoft.com/uk-ua/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-8.0

4. Welcome to .NET - .NET. Microsoft Learn: Build skills that open doors in your career. URL: https://learn.microsoft.com/uk-ua/dotnet/welcome

5. Introduction to ASP.NET Core Minimal APIs | The .NET Tools Blog. The JetBrains Blog. URL: https://blog.jetbrains.com/dotnet/2023/04/25/introduction-to-asp-net-core-minimal-apis/

6. Weather API - OpenWeatherMap. Current weather and forecast - OpenWeatherMap. URL: https://openweathermap.org/api

7. JetBrains. Rider: The Cross-Platform .NET IDE from JetBrains. JetBrains. URL: https://www.jetbrains.com/rider/

8. JetBrains. DataGrip: The Cross-Platform IDE for Databases & SQL by JetBrains. JetBrains. URL: https://www.jetbrains.com/datagrip/?var=light

9. GitHub - TelegramBots/Telegram.Bot: .NET Client for Telegram Bot API. GitHub. URL: https://github.com/TelegramBots/Telegram.Bot

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

РОЗРОБКА TELEGRAM-БОТУ МОВОЮ ПРОГРАМУВАННЯ C# ДЛЯ ПЕРЕГЛЯДУ ПРОГНОЗУ ПОГОДИ

Виконав студент 4 курсу
групи ПД-41
Кожем'якін Олександр Вікторович
Керівник роботи
д.т.н., професор, Іьїн Олег Юрійович

Київ – 2024

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

- **Мета роботи** – полегшення отримання погодної інформації за допомогою впровадження Telegram-боту.
- **Об'єкт дослідження** – процес отримання погодної інформації.
- **Предмет дослідження** – програмне забезпечення для отримання погодної інформації

2

ЗАДАЧІ ДИПЛОМНОЇ РОБОТИ

1. Здійснити аналіз предметної області.
2. Дослідити існуючі програмні засоби, виявити ключові можливості.
3. Визначити функціональні та нефункціональні вимоги застосунку.
4. Проаналізувати доступні інструменти для розробки програмного продукту.
5. Розробити механізм отримання та обробки погодних даних.
6. Спроектувати та розробити Telegram-бот відповідно до визначених потреб та вимог.
7. Виконати тестування розробленого Telegram-боту.

3

АНАЛІЗ АНАЛОГІВ

Функціональні можливості	«Погода» на IOS та iPadOS	«Погода» на Android	AccuWeather	Convenient Weather Bot
Створення, перегляд та видалення підписок на погодну розсилку	Ні	Ні	Ні	Так
Детальна погодна інформація	Так	Так	Так	Так
Погодна інформація за підписками на розсилку	Ні	Ні	Ні	Так
Погода за поточним місцезнаходженням	Так	Так	Так	Так
Інтеграція з Telegram	Ні	Ні	Ні	Так
Зображення актуальних погодних умов	Так	Так	Так	Так

4

ФУНКЦІОНАЛЬНІ ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1. Відображення актуальної погодної інформації за запитом.
2. Відображення погодної інформації за місцезнаходженням.
3. Створення підписки на погодну розсилку із використанням запиту.
4. Створення підписки на погодну розсилку із поточним місцезнаходженням.
5. Перегляд створених підписок.
6. Видалення підписок
7. Обробка помилок в ході опрацювання користувацьких команд.
8. Валідація даних за погодним запитом.
9. Перегляд доступних операцій у боті.

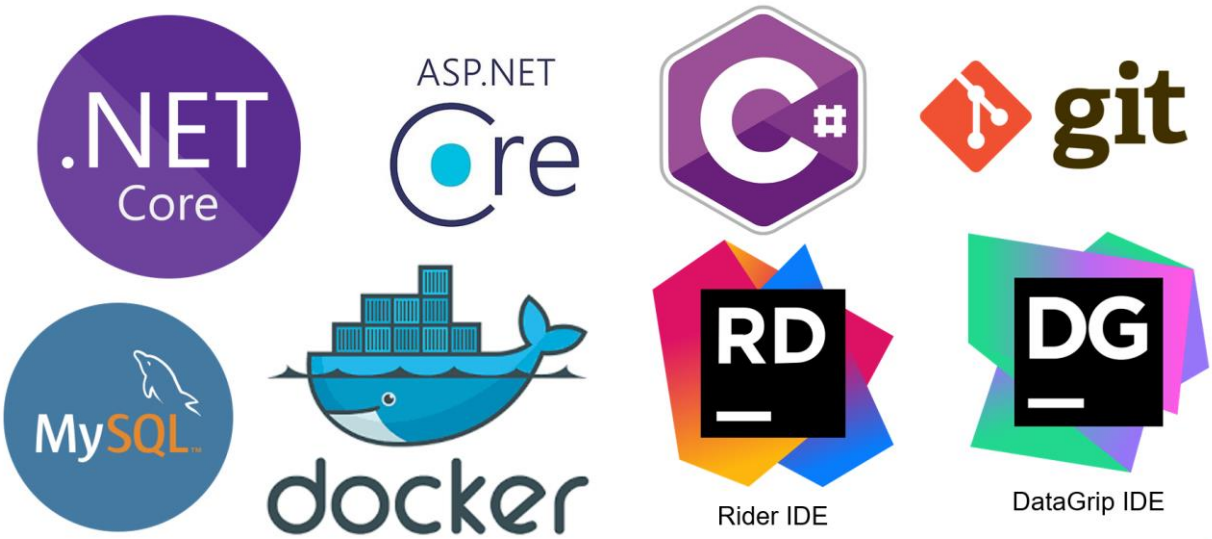
5

НЕФУНКЦІОНАЛЬНІ ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1. Продуктивність. Бот повинен надавати відповіді на запити користувачів менш ніж за 2 секунди.
2. Відказостійкість. Якщо запитуємий мікросервіс знаходиться в неактивному стані, мікросервіс, що запитує продовжує роботу та повідомляє користувача про недоступність функціоналу.
3. Сумісність. Сервіси боту повинні підтримувати такі операційні системи:
    1. Windows
    2. Linux
    3. maxOS
4. Системні вимоги. Для коректної роботи сервісів боту необхідне дотримання мінімальних вимог:
    1. Процесор з підтримкою x64 архітектури, 4 ядра, 2.0ГГц.
    2. Об'єм оперативної пам'яті у розмірі 8Гб.
    3. Використання носія для збереження даних розміром 20Гб.
    4. Стабільне підключення до мережі Інтернет зі швидкістю 30мбіт/с

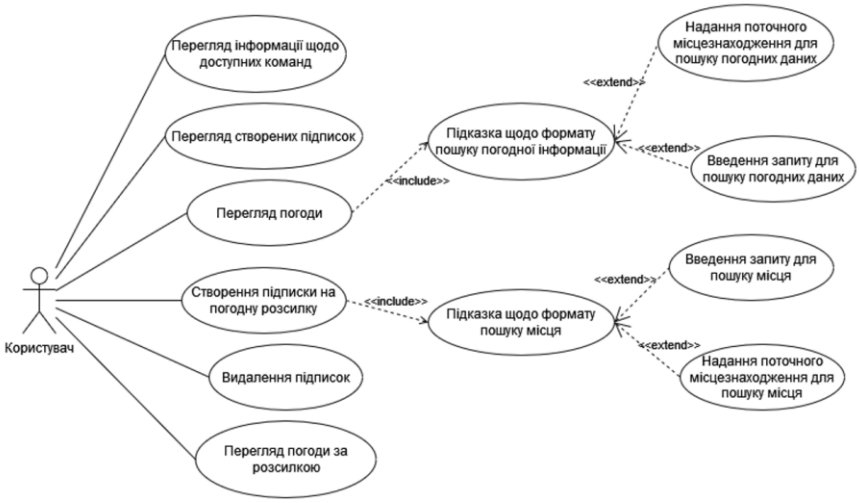
6

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ



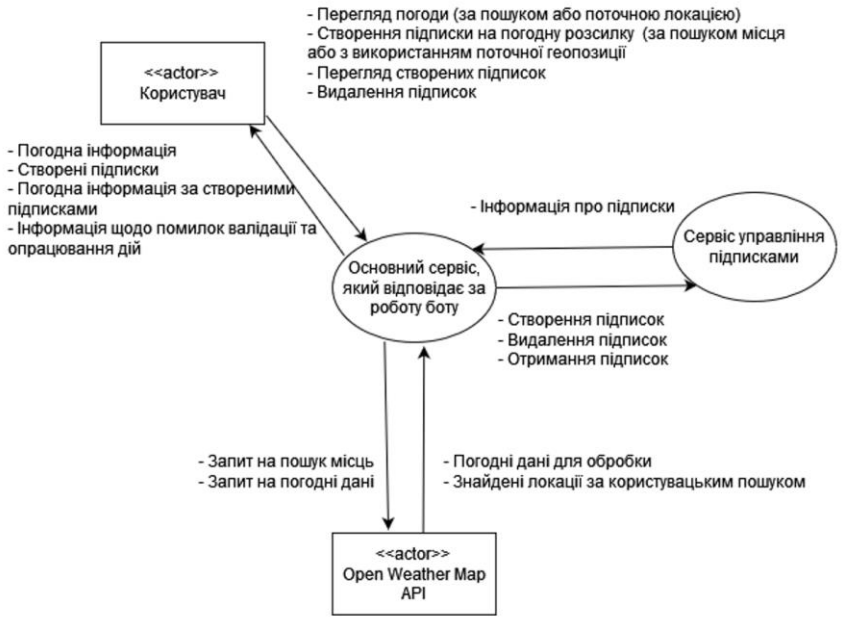
7

ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ



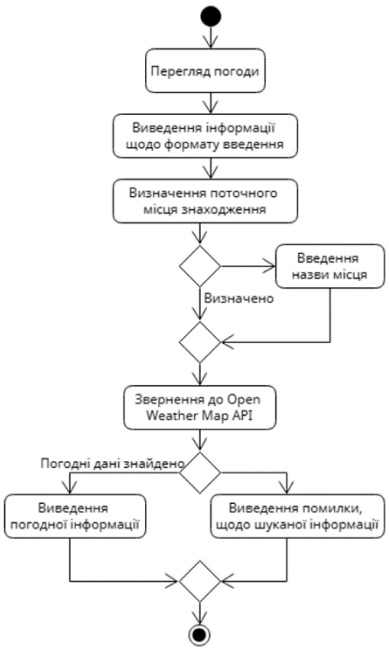
8

КОНТЕКСТНА ДІАГРАМА



9

ДІАГРАМА ДІЯЛЬНОСТІ ПЕРЕГЛЯДУ ПОГОДИ

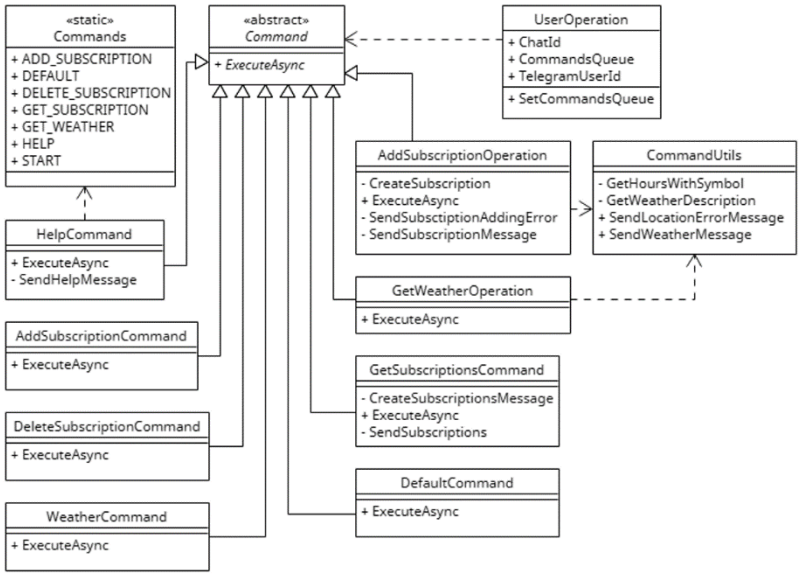


10

ДІАГРАМА ДІЯЛЬНОСТІ ПЕРЕГЛЯДУ СТВОРЕНИХ РОЗСИЛОК

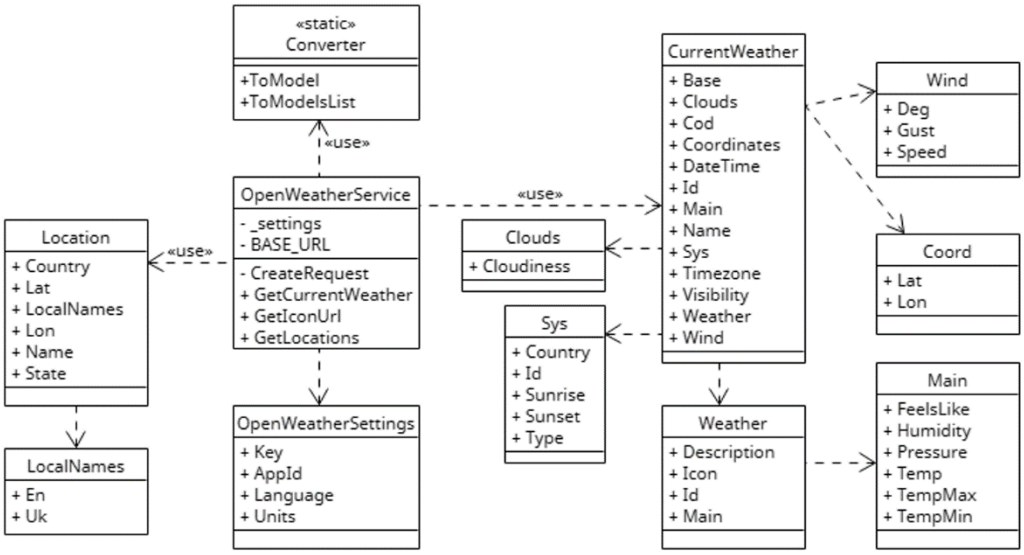
11

ДІАГРАМА КЛАСІВ КОМАНД ТА ОПЕРАЦІЙ У TELEGRAMWEATHERBOT.SERVER



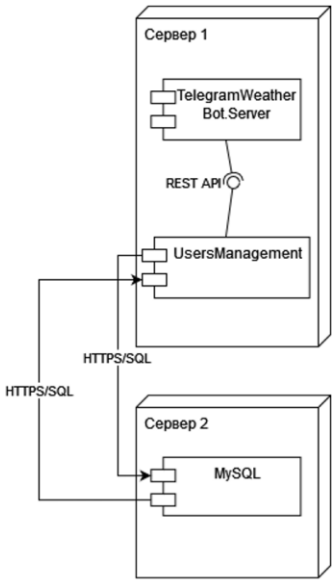
12

ДІАГРАМА КЛАСІВ OPENWEATHER



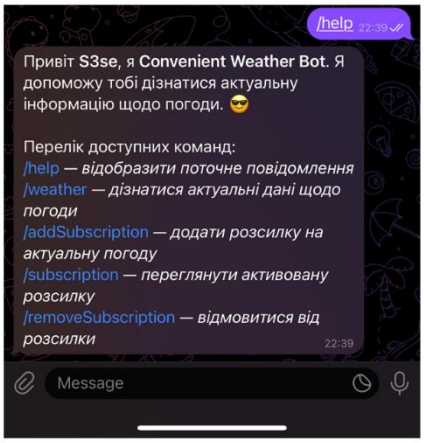
13

ДІАГРАМА РОЗГОРТАННЯ

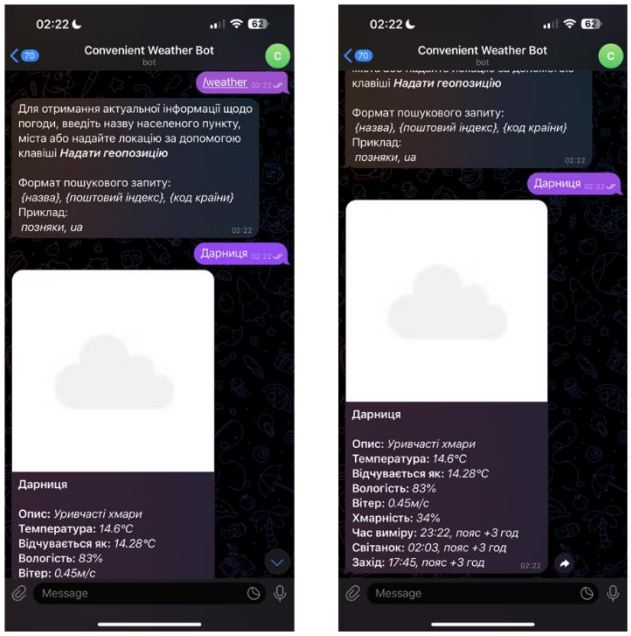


14

ЕКРАННІ ФОРМИ

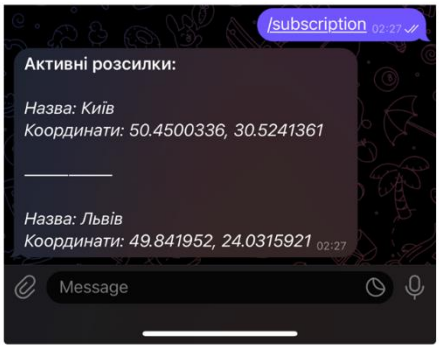


Відображення повідомлення з описом команд

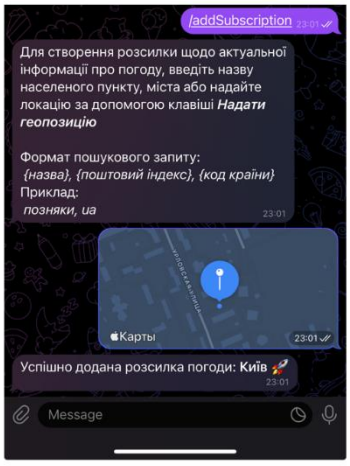


Погодна інформація за пошуковим запитом

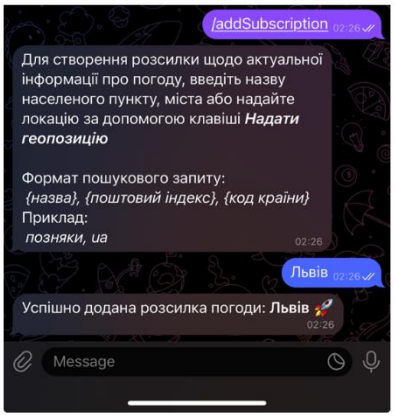
ЕКРАННІ ФОРМИ



Перегляд створених підписок на розсилки

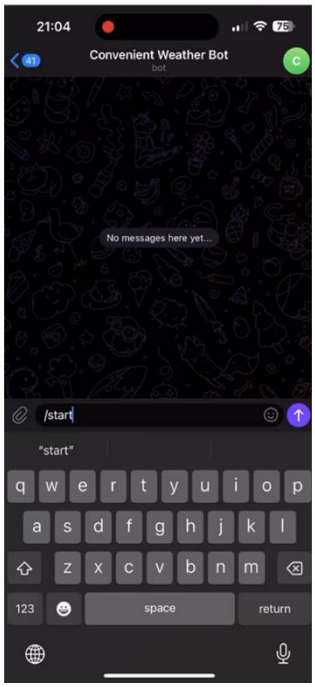


Створення підписки за пошуком



Створення підписки за поточним місцезнаходженням

ВІДЕО РОБОТИ ЗАСТОСУНКУ



17

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Кожем’якін О.В., Гаманюк І.М. Аналіз засобів розробки Телеграм-боту для перегляду прогнозу погоди: Матеріали Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в ІКТ». Збірник тез. 24.04.2024, ДУІКТ, м. Київ. К.: ДУІКТ – 173 с.
2. Кожем’якін О.В., Гаманюк І.М. Використання RESTful WEB API для отримання погодної інформації щодо Телеграм-боту для перегляду прогнозу погоди: Матеріали Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в ІКТ». Збірник тез. 24.04.2024, ДУІКТ, м. Київ. К.: ДУІКТ – 175 с.

18

ВИСНОВКИ

1. Проведено аналіз предметної галузі. В результаті аналізу було виявлені важливі аспекти притаманні галузі. Визначені ключові функціональні можливості, які необхідні користувачеві.
2. Здійснено дослідження існуючих аналогічних програмних засобів. Визначено необхідний функціонал, який треба розробити.
3. Визначені ключові функціональні та нефункціональні вимоги.
4. За результатами аналізу було обрано відповідні програмні засоби для реалізації Telegram боту.
5. Визначено місце і роль Telegram-боту в контексті інших акторів.
6. Спроектовано діяльність щодо перегляду погоди та розсилок.
7. Спроектовано розгортання застосунку на обчислювальних вузлах.
8. Спроектовано інтерфейс користувача.
9. Спроектовано та впроваджено механізм отримання погодних даних. Використано API від Open Weather Map, яке створено за принципом REST та надає дані у форматі JSON. Для легкої та швидкої взаємодії з API створено бібліотеку класів та інші допоміжні засоби, що відповідають за взаємодію з API для отримання погодних даних.
10. Розроблено Telegram – бот відповідно до визначених функціональних вимог. У результаті було створено 2 мікросервіси, що відповідають за необхідний функціонал.
11. Тестування програмного застосунку здійснювалося за допомогою модульного тестування та мануального тестування. За результатами модульного та мануального тестування отримано успішне виконання всіх тестів.

ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ

```
1 usage  Oleksandr Kozhemiakin
public class SubscriptionsRepository(DataContext context) : ISubscriptionsRepository
{
    0+1 usages  Oleksandr Kozhemiakin
    public async Task<bool> RemoveSubscriptions(string telegramUserId)
    {
        var activeSubscriptions :List<Subscription> = await context.Subscriptions // DbSet<Subscription>
            .AsNoTracking() // IQueryable<Subscription>
            .Where(x :Subscription => x.TelegramUserId == telegramUserId).ToListAsync(); // Task<List<...>>

        if (activeSubscriptions.Count < 1)
        {
            return false;
        }

        context.Subscriptions.RemoveRange(activeSubscriptions);
        var result :int = await context.SaveChangesAsync();
        return result > 0;
    }

    0+1 usages  Oleksandr Kozhemiakin
    public async Task<List<Subscription>> GetSubscriptions(string telegramUserId)
    {
        return await context.Subscriptions // DbSet<Subscription>
            .AsNoTracking()
            .Where(x :Subscription => x.TelegramUserId == telegramUserId) // IQueryable<Subscription>
            .ToListAsync(); // Task<List<...>>
    }

    0+1 usages  Oleksandr Kozhemiakin
    public async Task<List<Subscription>> GetSubscriptions()
    {
        return await context.Subscriptions.AsNoTracking().ToListAsync();
    }

    0+1 usages  Oleksandr Kozhemiakin
    public async Task<bool> AddSubscription(Subscription subscription)
    {
        await context.Subscriptions.AddAsync(subscription);
        var result :int = await context.SaveChangesAsync();
        return result > 0;
    }
}
```

Рис. 1 Репозиторій для роботи з підписками SubscriptionsRepository

```
8 usages  Oleksandr Kozhemiakin  3 exposing APIs
public class RequestBuilder
{
    private readonly HttpRequestMessage _httpRequestMessage;
    private readonly List<Parameter> _parameters = [];

    5 usages  Oleksandr Kozhemiakin
    public RequestBuilder(HttpMethod method, string baseAddress)
    {
        _httpRequestMessage = new HttpRequestMessage(method, new Uri(baseAddress, UriKind.RelativeOrAbsolute));
    }

    1 usage  Oleksandr Kozhemiakin
    public RequestBuilder AddParameter(Parameter parameter)
    {
        _parameters.Add(parameter);
        return this;
    }

    1 usage  Oleksandr Kozhemiakin
    public RequestBuilder AddParameters(IEnumerable<Parameter> parameters)
    {
        _parameters.AddRange(parameters);
        return this;
    }

    1 usage  Oleksandr Kozhemiakin
    public RequestBuilder SetJsonBody(object body)
    {
        _httpRequestMessage.Content = new StringContent(body.ToString() ?? string.Empty);
        _httpRequestMessage.Content.Headers.ContentType = new MediaTypeHeaderValue("application/json");
        return this;
    }

    5 usages  Oleksandr Kozhemiakin
    public HttpRequestMessage Build()
    {
        if (_parameters.Count <= 0)
        {
            return _httpRequestMessage;
        }

        var uriBuilder = new UriBuilder(_httpRequestMessage.RequestUri!)
        {
            Query = _parameters.ToQueryString()
        };

        _httpRequestMessage.RequestUri = uriBuilder.Uri;
        return _httpRequestMessage;
    }
}
```

Рис. 3 Клас для створення HTTP запитів RequestBuilder

```
public static class CommandUtils
    3 usages  Oleksandr Kozhemiakin
    public static async Task SendLocationErrorMessage(ITelegramBotClient botClient, Update update)
    {
        var text = "Інформації щодо локації не було знайдено 😢\\. Спробуйте ще раз";
        await botClient.SendTextMessageAsync(update.Message!.Chat.Id, text);
    }

    2 usages  Oleksandr Kozhemiakin
    public static async Task SendWeatherMessage(
        ITelegramBotClient botClient,
        string chatId,
        CurrentWeather weather,
        string? weatherQuery)
    {
        var weatherInfo = weather.Weather?.FirstOrDefault();
        var weatherDescription :string = GetWeatherDescription(weatherInfo);
        var timeZoneHours :double = CommonUtils.ToHours(weather.Timezone);
        var iconUrl :string = OpenWeatherService.GetIconUrl(weatherInfo?.Icon);

        var text :string = $"*{CommonUtils.EscapeString(weatherQuery)}* \n\n" +
                   $"*Опис:* _{weatherDescription}_ \n" +
                   $"*Температура:* _{CommonUtils.EscapeString(weather.Main?.Temp)}\u00b0C_ \n" +
                   $"*Відчувається як:* _{CommonUtils.EscapeString(weather.Main?.FeelsLike)}\u00b0C_ \n" +
                   $"*Вологість:* _{weather.Main?.Humidity}%_ \n" +
                   $"*Вітер:* _{CommonUtils.EscapeString(weather.Wind?.Speed)}м/с_ \n" +
                   $"*Хмарність:* _{weather.Clouds?.Cloudiness}%_ \n" +
                   $"*Час виміру:* _{CommonUtils.ToDateTime(weather.DateTime):HH:mm}, пояс {GetHoursWithSymbol(timeZoneHours)} год_ \n" +
                   $"*Світанок:* _{CommonUtils.ToDateTime(weather.Sys?.Sunrise):HH:mm}, пояс {GetHoursWithSymbol(timeZoneHours)} год_ \n" +
                   $"*Захід:* _{CommonUtils.ToDateTime(weather.Sys?.Sunset):HH:mm}, пояс {GetHoursWithSymbol(timeZoneHours)} год_ ";

        await botClient.SendPhotoAsync(
            chatId: chatId,
            photo: InputFile.FromUri(iconUrl),
            caption: text,
            parseMode: ParseMode.MarkdownV2,
            replyMarkup: new ReplyKeyboardRemove()); // Task<Message>
    }

    1 usage  Oleksandr Kozhemiakin
    private static string GetWeatherDescription(Weather? weatherInfo)
    {
        return weatherInfo?.Description is null
            ? string.Empty
            : $"{char.ToUpper(weatherInfo.Description[0])}{weatherInfo.Description[1..]}";
    }

    3 usages  Oleksandr Kozhemiakin
    private static string GetHoursWithSymbol(double hours)
    {
        return CommonUtils.EscapeString(
            hours > 0
                ? $"+{hours.ToString(CultureInfo.InvariantCulture)}"
                : hours.ToString(CultureInfo.InvariantCulture));
    }
```

Рис. 4 Допоміжний клас CommandUtils

```
public class SubscriptionWeather(
    ITelegramBotClient botClient,
    UsersManagementService usersManagementService,
    OpenWeatherService weatherService,
    IOptions<CommonSettings> options) : BackgroundService
{
    private readonly CommonSettings _commonSettings = options.Value;

    Oleksandr Kozhemiakin
    protected override async Task ExecuteAsync(CancellationToken stoppingToken)
    {
        using PeriodicTimer timer = new( period: TimeSpan.FromMinutes(_commonSettings.SubscriptionsSyncTimeoutInMinutes));
        while (await timer.WaitForNextTickAsync(stoppingToken))
        {
            Console.WriteLine("Started tick for Subscription weather background service");
            await SendSubscriptionWeather();
            Console.WriteLine("Ended tick for Subscription weather background service");
        }

        Console.WriteLine("Subscription weather background service stopped");
    }

    1 usage  Oleksandr Kozhemiakin
    private async Task SendSubscriptionWeather()
    {
        var subscriptions :List<Subscription>? = await usersManagementService.GetSubscriptions();
        if (subscriptions is null || subscriptions.Count < 1)
        {
            return;
        }

        var groupedSubscriptions :IEnumerable<IGrouping<...>> = subscriptions.GroupBy(x :Subscription => x.TelegramUserId);
        foreach (var subscriptionsGroup :IGrouping<string?,Subscription> in groupedSubscriptions)
        {
            foreach (var subscription in subscriptionsGroup)
            {
                if (string.IsNullOrEmpty(subscription.TelegramUserId) || string.IsNullOrEmpty(subscription.ChatId))
                {
                    continue;
                }

                try
                {
                    var coordsSplit :string[] = subscription.Coordinates?.Split( separator: ':') ?? [];
                    if (coordsSplit.Length < 2)
                    {
                        await SendUnsuccessfulSubscriptionWeather(subscription.LocationName, subscription.ChatId);
                        continue;
                    }

                    var weatherInfo = await weatherService.GetCurrentWeather( lon: coordsSplit[1], lat: coordsSplit[0]);
                    if (weatherInfo?.Main is null)
                    {
                        await SendUnsuccessfulSubscriptionWeather(subscription.LocationName, subscription.ChatId);
                        continue;
                    }
```

Рис. 5 Клас сервісу, що відповідає за погодні розсилки

```
Oleksandr Kozhemiakin
public static class CoreDependencies
{
    1 usage  Oleksandr Kozhemiakin
    public static IServiceCollection AddSettings(this IServiceCollection collection, IConfiguration configuration)
    {
        collection.Configure<CommonSettings>(configuration.GetSection(CommonSettings.KEY));
        collection.Configure<OpenWeatherSettings>(configuration.GetSection(OpenWeatherSettings.Key));

        return collection;
    }

    1 usage  Oleksandr Kozhemiakin
    public static IServiceCollection AddWeatherServices(this IServiceCollection collection)
    {
        collection.AddTransient<OpenWeatherService>();
        return collection;
    }

    1 usage  Oleksandr Kozhemiakin
    public static IServiceCollection AddOtherServices(this IServiceCollection collection)
    {
        collection.AddTransient<UsersManagementService>();
        collection.AddTransient<RequestSender>();
        return collection;
    }

    1 usage  Oleksandr Kozhemiakin
    public static IServiceCollection AddBotServices(this IServiceCollection collection, IConfiguration configuration)
    {
        var settings = configuration.GetSection(CommonSettings.KEY).Get<CommonSettings>();
        TelegramBotClient botClient = new(settings!.BotToken);

        collection.AddSingleton<ITelegramBotClient>(botClient);
        collection.AddTransient<CommandsService>();
        collection.AddSingleton<CommandsCoordinator>();
        return collection;
    }

    1 usage  Oleksandr Kozhemiakin
    public static IServiceCollection AddHostedServices(this IServiceCollection collection)
    {
        collection.AddHostedService<TelegramBotUpdate>();
        collection.AddHostedService<SubscriptionWeather>();
        return collection;
    }
}
```

Рис. 6 Клас з методами розширення для додавання залежностей через механізм Dependency Injection