МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЖИТОМИРСЬКИЙ АГРОТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ
ВІДДІЛЕННЯ «ІНЖЕНЕРНА ІНФРАСТРУКТУРА ТА
КОМП'ЮТЕРНІ НАУКИ»
ЦИКЛОВА КОМІСІЯ СПЕЦІАЛЬНОСТІ «КОМП'ЮТЕРНІ НАУКИ»

ПОЯСНЮВАЛЬНА ЗАПИСКА

до дипломної роботи
освітньо-професійний ступінь «фаховий молодший бакалавр»
на тему:

Розробка програми
«Розробка додатку для відстеження погоди»

Виконав студент (ка) _4_ курсу, групи П-41
спеціальності 122 «Комп'ютерні науки»

Сергеєв Андрій Олександрович

Керівник Устименко Людмила Миколаївна

Рецензент _____

Житомир – 2024 року

Зміст

РЕФЕРАТ

Записка: 53 стор., 27 рис., 1 додаток, 20 джерел.

Ключові слова: ДОДАТОК ДЛЯ ВІДСТЕЖЕННЯ ПОГОДИ, РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, АНАЛІЗ ПОГОДНИХ ДАНИХ, API ПОГОДИ, ГЕОЛОКАЦІЯ, МАРКЕРИ НА МАПІ, ОТРИМАННЯ ПОГОДНИХ ПРОГНОЗІВ, АЛГОРИТМИ ОБРОБКИ ДАНИХ, РЕАЛІЗАЦІЯ RESTFUL API, ЛОКАЦІЙНІ СЕРВІСИ.

Дипломна робота присвячена розробці додатка для відстеження погоди, що має велике значення у повсякденному житті сучасної людини. У роботі використовується комплексний підхід до розробки програмного забезпечення, що включає в себе аналіз потреб користувачів, вивчення існуючих технологій та методів, розробку функціональності та створення інтерфейсу користувача. Основні завдання роботи полягають у створенні зручного та ефективного інструменту для отримання актуальної інформації про погоду в будь-якому місці та часі. Досягнення цієї мети вимагає використання сучасних технологій розробки програмного забезпечення, а також уваги до деталей та потреб користувачів. Основним результатом дипломної роботи є розроблений додаток, який може бути використаний для відстеження погоди з використанням мобільних та веб-платформ.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API: Application Programming Interface - інтерфейс програмування додатків, який використовується для отримання погодних даних з відповідного джерела.

UI: User Interface - інтерфейс користувача, який включає в себе всі елементи та взаємодію з користувачем.

GPS: Global Positioning System - глобальна система позиціонування, використовується для визначення місцезнаходження користувача.

JSON: JavaScript Object Notation - формат обміну даними, часто використовується для передачі погодних даних з сервера.

HTTP: Hypertext Transfer Protocol - протокол передачі гіпертексту, використовується для взаємодії з сервером для отримання погодних даних.

UI/UX: User Interface/User Experience - дизайн інтерфейсу та користувацький досвід, важливі аспекти при розробці зручного та привабливого додатку.

SDK: Software Development Kit - комплект розробника програмного забезпечення, який може включати в себе інструменти для розробки додатків для певної платформи або сервісу.

DB: Database - база даних, де можуть зберігатися погодні дані для подальшого використання.

SSL: Secure Sockets Layer - протокол шару захищеного з'єднання, який забезпечує захищеність передачі даних між додатком та сервером.

UX Research: User Experience Research - дослідження користувацького досвіду, що включає в себе аналіз потреб користувачів та їхніх вимог до додатку.

Вступ

Люди регулярно перевіряють погоду, щоб планувати свої щоденні справи, подорожі та заходи на відкритому повітрі. Додаток, що надає точні прогнози, допомагає в прийнятті рішень і уникненні непередбачених ситуацій.

Через зміни клімату погода стає все більш непередбачуваною. Надійний додаток для відстеження погоди може допомогти людям готуватися до різких змін погодних умов, зокрема екстремальних явищ, таких як шторми, сильні дощі або спека.

Сучасні технології дозволяють розробляти додатки з високою точністю прогнозів, інтеграцією даних з різних джерел, такими як супутники та метеорологічні станції, а також використовувати штучний інтелект для покращення точності прогнозів.

Більшість людей мають смартфони, що робить мобільні додатки для відстеження погоди зручними і доступними. Користувачі можуть швидко отримувати інформацію про погоду в будь-який час і в будь-якому місці.

Для підприємств, таких як авіалінії, логістичні компанії, аграрний сектор та туризм, точні дані про погоду є критично важливими.

Додаток для відстеження погоди може попереджати про небезпечні погодні умови, такі як ожеледь, висока температура чи сильний вітер, що допомагає зберігати здоров’я і безпеку користувачів.

Деякі додатки можуть надавати інформацію про якість повітря, рівень забруднення та інші екологічні показники, що стає все більш важливим у контексті глобальних екологічних проблем.

Додатки можуть бути налаштовані відповідно до індивідуальних потреб користувачів, надаючи персоналізовані прогнози, повідомлення та рекомендації.

Враховуючи всі ці фактори, розробка додатку для відстеження погоди є актуальною та має великий потенціал для розвитку і впровадження в різних сферах життя.

Розробка додатку для відстеження погоди має кілька основних цілей, які спрямовані на задоволення потреб користувачів та підвищення ефективності використання погодної інформації. Додаток має забезпечити користувачів точними прогнозами погоди, що оновлюються в режимі реального часу. Використання даних з метеорологічних станцій, супутників та інших джерел, а також алгоритмів штучного інтелекту для аналізу та передбачення погодних умов. Створити інтуїтивно зрозумілий і зручний інтерфейс, який дозволяє швидко отримувати необхідну інформацію. Розробка простого і привабливого дизайну.

Забезпечити своєчасні попередження про небезпечні погодні явища, такі як бурі, сильні дощі, снігопади, спека або холод.

Реалізація цих цілей дозволить створити корисний, зручний та функціональний додаток для відстеження погоди, який задовольнить потреби різних категорій користувачів і буде конкурентоспроможним на ринку.

Для реалізації такого додатку для відстеження погоди, який показує температуру, вологість, швидкість та напрям вітру у місцевості, де користувач встановив маркер, нам знадобиться поєднання функціоналу карт та можливостей отримання погодних даних через API. Ось загальний підхід:

Використовується контрол Map або MapView для відображення карти на графічному інтерфейсі користувача. Ми будемо використовувати картографічні сервіси, такі як Google Maps.

Дозволимо користувачеві встановлювати маркери на карті за допомогою миші або тачскріна. Після встановлення маркера збережіть його координати (широта та довгота) для подальшого використання при запиті погодних даних.

Використовується погодний API (наприклад, OpenWeatherMap, WeatherAPI тощо), щоб отримати дані про погоду для встановленого маркера.

Надсилають запити до API, передаючи координати маркера, щоб отримати погодні дані для цього регіону.

Після отримання погодних даних від API буде відображати їх на інтерфейсі користувача. Додайте поля для відображення температури, вологості, швидкості та напряму вітру.

Потрібно звернути увагу на асинхронність запитів до API, щоб уникнути блокування інтерфейсу користувача під час очікування відповіді від сервера.

Будемо зберігати отримані дані та оновлюйте їх з заданою періодичністю, щоб користувач завжди бачив актуальну інформацію. В низу знаходиться таймер. Через певний час дані оновлюються.

Додамо можливість вибору одиниць вимірювання для температури, швидкості вітру та вологості.

Загальний алгоритм роботи полягає в тому, щоб дозволити користувачеві встановлювати маркери на карті та отримувати погодні дані для вибраних місцевостей. Це дозволить користувачеві легко відстежувати погодні умови у своєму регіоні та буде корисною функцією для планування активностей на відкритому повітрі.

Для розробки додатку для відстеження погоди використаємо мову програмування C#

Об'єктом дослідження є система для відстеження погодних умов. Це включає в себе всі компоненти та аспекти, пов'язані з моніторингом, збором, обробкою та відображенням даних про погодні умови.

Предметом дослідження є методи та технології розробки програмного забезпечення для відстеження погоди. Це включає в себе наступні аспекти:

Використання Windows Forms для створення зручного та інтуїтивно зрозумілого графічного інтерфейсу користувача (GUI).

Інтеграція картографічних сервісів для відображення місць та встановлення маркерів.

Використання зовнішніх API (наприклад, OpenWeatherMap) для отримання актуальних даних про погодні умови.

Обробка отриманих даних у форматі JSON за допомогою бібліотек, таких як Newtonsoft.Json.

Реалізація асинхронних запитів до API для забезпечення швидкого та безперебійного отримання даних без блокування основного потоку додатку.

Збереження налаштувань користувача (наприклад, обрані місця для забезпечення персоналізованого досвіду використання додатку.

Відображення температури, вологості, швидкості та напрямку вітру на графічному інтерфейсі.

Візуалізація даних у зручному та зрозумілому форматі для користувача.

Додавання підтримки різних мов та одиниць вимірювання для розширення аудиторії користувачів.

Можливість налаштування інтерфейсу під індивідуальні потреби користувача.

Дослідження предмету розробки додатку для відстеження погоди дозволяє зрозуміти кращі практики та підходи до створення ефективного і зручного інструменту для моніторингу погодних умов. Це також включає вивчення різних технологій, які можуть бути використані для покращення функціональності та продуктивності додатку.

У даному проекті для розробки додатку для відстеження погоди використовується мова програмування C#. Ця мова була обрана з кількох причин.

Платформа .NET: C# є основною мовою для розробки додатків на платформі .NET, яка забезпечує потужну інфраструктуру для створення різноманітних типів додатків, включаючи десктопні додатки.

Windows Forms: Використання Windows Forms у поєднанні з C# дозволяє легко створювати графічні інтерфейси користувача для додатків, що працюють у середовищі Windows. Windows Forms забезпечує широкий набір інструментів і компонентів для побудови зручного та інтуїтивно зрозумілого GUI.

Бібліотеки та інструменти: C# та .NET мають велику кількість готових бібліотек і інструментів, які спрощують реалізацію багатьох задач, таких як

робота з мережевими запитами (HttpClient), обробка даних у форматі JSON (Newtonsoft.Json) та інтеграція з API.

Асинхронність: C# підтримує асинхронне програмування через ключові слова async і await, що дозволяє створювати додатки, які можуть виконувати фонові операції без блокування інтерфейсу користувача, що є важливим для додатків, які часто взаємодіють з зовнішніми API.

Використання мови C# у поєднанні з платформою .NET та Windows Forms дозволило створити ефективний та зручний у використанні додаток для відстеження погоди, який відповідає всім вимогам проекту.

РОЗДІЛ 1. ОГЛЯД ТА АНАЛІЗ ТЕХНОЛОГІЙ РОЗРОБКИ

1.1. Постановка основної задачі розробки

До ключових завдань розробки додатку для відстеження погоди відноситься. Інтеграція з картографічним сервісом: Реалізувати можливість відображення карти на графічному інтерфейсі користувача та встановлення маркерів для вибору місцевості.

Отримання погодних даних через API: Підключити зовнішнє погодне API (наприклад, OpenWeatherMap) для отримання актуальної погодної інформації за вказаними координатами.

Відображення погодних даних на інтерфейсі користувача: Показати температуру, вологість, швидкість та напрям вітру для вибраної місцевості на графічному інтерфейсі.

Асинхронність та оптимізація: Забезпечити асинхронні запити до API, щоб уникнути блокування інтерфейсу користувача під час отримання погодних даних.

Персоналізація налаштувань: Додати можливість користувачам налаштовувати одиниці вимірювання (наприклад, Цельсій або Фаренгейт) та інші параметри.

Оновлення даних у реальному часі: Забезпечити автоматичне оновлення погодних даних з заданою періодичністю для підтримки актуальності інформації.

Обробка помилок та відображення повідомлень: Реалізувати обробку помилок при відсутності з'єднання з мережею або інших проблемах з отриманням даних та відображення відповідних повідомлень користувачеві.

Перегляд додаткової інформації: Додати можливість перегляду розширених погодних умов, таких як прогноз погоди на кілька днів чи показники якості повітря.

Локалізація та інтернаціоналізація: Забезпечити можливість вибору мови інтерфейсу та інших параметрів, щоб додаток був доступним для користувачів з різних країн.

Тестування та відлагодження: Провести тестування функціоналу додатку для перевірки його працездатності та відповідності специфікаціям, а також виправити всі виявлені помилки.

Головною задачею розробки додатку для відстеження погоди є створення надійного та зручного інструменту, який надає користувачам актуальну інформацію про погодні умови в реальному часі для вибраних місць, використовуючи інтеграцію з картографічними та погодними сервісами.

Конкретні аспекти головної задачі включають в себе наступні аспекти:

Забезпечити отримання даних про поточні погодні умови, включаючи температуру, вологість, швидкість та напрям вітру, з достовірних джерел через API.

Розробити інтуїтивно зрозумілий графічний інтерфейс, що дозволяє користувачам легко взаємодіяти з додатком, встановлювати маркери на карті та переглядати погодні дані.

Надати користувачам можливість налаштовувати додаток відповідно до їхніх індивідуальних потреб, включаючи вибір одиниць вимірювання та мови інтерфейсу.

Забезпечити асинхронне оновлення погодних даних у реальному часі без блокування роботи додатку, щоб користувачі завжди мали доступ до свіжої інформації.

Створити архітектуру додатку, яка дозволяє легко додавати нові функції та підтримувати масштабованість для роботи з великими об'ємами даних або великою кількістю користувачів.

Досягнення головної задачі забезпечить створення функціонального та корисного додатку, який буде відповідати потребам користувачів у відстеженні погодних умов для прийняття інформованих рішень у повсякденному житті, таких як планування поїздок, заходів на відкритому повітрі та інших активностей, залежних від погоди.

Основні функції розробки додатку для відстеження погоди містять наступні аспекти:

Вбудована інтеграція з картографічними сервісами (наприклад, Google Maps або Bing Maps).

Можливість масштабування карти та переміщення по ній.

Дозволяє користувачам встановлювати маркери на карті для вибору місцевості.

Відображення координат (широта і довгота) для обраного маркера.

Інтеграція з погодними API (наприклад, OpenWeatherMap) для отримання актуальних даних.

Отримання таких параметрів, як температура, вологість, швидкість та напрям вітру.

Відображення погодних даних у зручному форматі (наприклад, у вигляді таблиць або графіків).

Показування поточних погодних умов для місцевості, де встановлено маркер.

Автоматичне оновлення погодних даних через певні інтервали часу.

Забезпечення асинхронного отримання даних без блокування інтерфейсу користувача.

Вибір одиниць вимірювання для температури (Цельсій/Фаренгейт), швидкості вітру (км/год, миль/год).

Збереження обраних місць і налаштувань користувача.

Показ історичних даних та прогнозу на кілька днів вперед.

Ці функції забезпечать створення зручного та функціонального додатку для відстеження погоди, який задовольнить потреби користувачів у наданні актуальної та точної інформації про погодні умови.

Загальною метою програмної розробки додатку для відстеження погоди є створення зручного, надійного та функціонального інструменту для отримання та відображення актуальної інформації про погодні умови в реальному часі для користувачів з різних регіонів світу. Додаток має

забезпечувати користувачів необхідними даними для планування своєї діяльності та прийняття інформованих рішень на основі погодних умов.

Забезпечити користувачів точними та своєчасними даними про погоду, включаючи температуру, вологість, швидкість і напрям вітру, атмосферний тиск та інші погодні параметри.

Розробити інтуїтивно зрозумілий інтерфейс користувача, який дозволяє легко отримувати та переглядати погодні дані.

Інтегрувати функцію картографії для вибору місцевості шляхом встановлення маркерів на карті.

Зберігати налаштування користувачів для забезпечення персоналізованого досвіду.

Реалізувати асинхронне отримання та оновлення погодних даних для забезпечення безперервного доступу до актуальної інформації без блокування інтерфейсу.

Надати розширені функції, такі як прогноз погоди на кілька днів вперед, історичні дані, оповіщення про екстремальні погодні умови та інтеграції з іншими сервісами.

Забезпечити масштабованість додатку для роботи з великими обсягами даних і підтримку великої кількості користувачів.

Захистити дані користувачів та забезпечити їх конфіденційність, використовуючи сучасні методи шифрування та безпеки даних.

Забезпечити регулярні оновлення та підтримку додатку, враховуючи зворотний зв'язок від користувачів та впроваджуючи нові функції та покращення.

Реалізація цих цілей допоможе створити високоякісний додаток для відстеження погоди, який буде корисним для користувачів у їх повсякденному житті, забезпечуючи їх необхідною інформацією для прийняття обґрунтованих рішень.

1.2. Огляд та порівняння аналогічних систем.

Розробка додатку для відстеження погоди включає в себе створення інструменту, який забезпечує користувачів актуальною та точною інформацією про погодні умови. Додаток повинен бути зручним, інтуїтивно зрозумілим та надійним, щоб задовольняти потреби широкої аудиторії.

Створення зручного графічного інтерфейсу з використанням Windows Forms.

Відображення карти з можливістю встановлення маркерів для вибору місцевості.

Інтеграція з зовнішніми погодними API для отримання актуальних даних про температуру, вологість, швидкість та напрям вітру.

Реалізація асинхронних запитів для отримання погодних даних без блокування інтерфейсу користувача.

Налаштування одиниць вимірювання та інших параметрів відповідно до потреб користувача.

Методи аналогічних систем

Для розробки додатку для відстеження погоди корисно вивчити методи, які використовуються в аналогічних системах:

API-інтеграція:

Більшість сучасних погодних додатків використовують API для отримання погодних даних. Популярні сервіси включають OpenWeatherMap, WeatherAPI, AccuWeather API та інші. Вони надають детальні дані про поточні погодні умови, прогнози та історичні дані.

Асинхронні запити та кешування:

Для покращення продуктивності та зниження навантаження на сервери, додатки використовують асинхронні запити для отримання даних та кешування результатів. Це дозволяє швидко оновлювати інформацію без затримок у роботі інтерфейсу.

Геолокація:

Багато додатків автоматично визначають місцезнаходження користувача за допомогою GPS і надають погодні дані для поточної локації. Це зручно для користувачів, які хочуть отримувати інформацію про погоду без необхідності вручну вибирати місце.

Персоналізація та налаштування:

Додатки, такі як Weather Underground і Dark Sky, дозволяють користувачам налаштовувати одиниці вимірювання, встановлювати оповіщення про зміну погоди та обирати параметри відображення інформації, що робить їх більш зручними та корисними для різних категорій користувачів.

Візуалізація даних:

Відображення даних у вигляді графіків, таблиць або інтерактивних карт є поширеною практикою. Наприклад, додатки Weather.com і Yahoo Weather використовують інтуїтивно зрозумілі графічні інтерфейси, що показують поточні умови та прогнози в зручному для сприйняття вигляді.

Розробка додатку для відстеження погоди потребує інтеграції з надійними погодними API, створення зручного та інтуїтивно зрозумілого інтерфейсу, реалізації асинхронного оновлення даних, а також можливостей персоналізації. Вивчення методів аналогічних систем дозволяє запозичити найкращі практики та уникнути поширених проблем, що сприятиме створенню якісного продукту, здатного задовольнити потреби користувачів.

1. OpenWeatherMap

OpenWeatherMap є однією з найбільш популярних систем для відстеження погоди, яка надає різноманітні погодні дані через API.

Загальні функції:

Поточні погодні дані: Температура, вологість, атмосферний тиск, швидкість і напрям вітру, хмарність, видимість.

Прогноз погоди: Погодні прогнози на кілька днів вперед, включаючи денні та нічні температури, опади, швидкість вітру.

Історичні дані: Дані про погоду за минулі дні, місяці або роки.

Погодні карти: Візуалізація погодних умов у реальному часі на карті, включаючи дощ, сніг, хмари, температуру, вітер.

Оповіщення: Попередження про екстремальні погодні умови.

2. WeatherAPI (раніше Weatherstack)

WeatherAPI надає простий у використанні API для отримання різноманітних погодних даних.

Загальні функції:

Поточні погодні дані: Температура, відчувається температура, вологість, тиск, вітер, опади.

Прогноз погоди: Денний прогноз на кілька днів вперед.

Історичні дані: Можливість отримання даних за минулі періоди.

Місцезнаходження: Автоматичне визначення місця користувача та погодних умов для цього місця.

API інтеграція: Легке інтегрування в різні додатки та вебсайти через API.

3. AccuWeather

AccuWeather відома своїми детальними прогнозами та аналізом погодних умов.

Загальні функції:

Поточні погодні дані: Температура, вологість, атмосферний тиск, швидкість і напрям вітру, відчувається температура.

Прогноз погоди: Прогноз на годину, день, тиждень з детальною інформацією.

Оповіщення про погоду: Сповіщення про небезпечні погодні умови.

Радар і супутникові зображення: Відображення поточних умов на мапі в режимі реального часу.

Індекси та показники: Ультрафіолетовий індекс, індекс якості повітря, індекс алергенів.

4. Weather Underground

Weather Underground спеціалізується на наданні гіперлокальних погодних даних з численних метеостанцій.

Загальні функції:

Поточні погодні дані: Температура, вологість, вітер, опади, видимість.

Прогноз погоди: Детальні прогнози на день і тиждень вперед.

Історичні дані: Дані про погоду за минулі періоди.

Погодні карти та радар: Інтерактивні карти з інформацією про опади, температуру, вітер.

Спільнота користувачів: Внесок даних від численних приватних метеостанцій по всьому світу.

5. Yahoo Weather

Yahoo Weather надає простий у використанні додаток з інтеграцією візуальних елементів.

Загальні функції:

Поточні погодні дані: Температура, відчувається температура, вологість, тиск, вітер.

Прогноз погоди: Прогнози на 5 та 10 днів вперед.

Фотографії місцевості: Інтеграція з Flickr для показу фотографій місцевості у відповідності до поточних погодних умов.

Оповіщення про погоду: Сповіщення про зміни в погоді.

Віджети: Віджети для швидкого перегляду погодних умов на домашньому екрані.

Кожна з вищезгаданих систем надає широкий спектр функцій для відстеження погодних умов. Вивчення їх функцій та підходів може допомогти у розробці ефективного та конкурентоспроможного додатку для відстеження погоди, який задовольняє потреби користувачів у наданні точної та актуальної інформації про погодні умови.

1.3. Вибір та обґрунтування технологій розробки

C# є популярною мовою програмування, яка активно підтримується Microsoft. Це забезпечує доступ до широкого спектру ресурсів, бібліотек та інструментів.

C# інтегрується з .NET платформою, яка забезпечує стабільність та продуктивність, необхідні для розробки сучасних додатків.

C# пропонує високу продуктивність та ефективність при роботі з асинхронними операціями, що важливо для отримання даних через API без затримок.

NET Core (частина .NET) дозволяє створювати додатки, які працюють на різних операційних системах, таких як Windows, macOS та Linux.

NET забезпечує доступ до великої кількості бібліотек для роботи з мережею, базами даних, графічним інтерфейсом та іншими важливими аспектами розробки.

Windows Forms забезпечує інтуїтивно зрозумілий інтерфейс для створення графічних додатків з використанням drag-and-drop редактора.

Windows Forms дозволяє легко налаштовувати зовнішній вигляд та поведінку елементів інтерфейсу.

Windows Forms є зрілою технологією з багаторічною історією, що забезпечує стабільність та надійність для створення настільних додатків.

Доступність та популярність: OpenWeatherMap є одним з найбільш популярних погодних API, що забезпечує доступ до широкого спектру погодних даних.

API має гарну документацію та приклади використання, що полегшує інтеграцію та використання в додатках.

OpenWeatherMap API надає доступ до поточних погодних умов, прогнозів, історичних даних та різноманітних погодних параметрів, що робить його ідеальним вибором для відстеження погоди.

HttpClient є частиною .NET і забезпечує зручний інтерфейс для виконання HTTP-запитів до погодних API. Він підтримує асинхронні операції, що дозволяє не блокувати інтерфейс користувача під час отримання даних.

Newtonsoft.Json є популярною бібліотекою для роботи з JSON у .NET. Вона забезпечує легке парсування та серіалізацію JSON-даних, що необхідно для обробки відповіді від погодного API.

Вибір технологій для розробки додатку для відстеження погоди базується на вимогах до стабільності, продуктивності, простоти використання та широких можливостях налаштування. Використання C#, .NET, Windows Forms та інтеграція з OpenWeatherMap API забезпечує створення ефективного, надійного та зручного інструменту для надання актуальної інформації про погодні умови.

Висновки до розділу 1

Розробка додатку для відстеження погоди є комплексним процесом, що вимагає ретельного вибору технологій та підходів для забезпечення точності, надійності та зручності користувача. На основі аналізу аналогічних систем та обґрунтування вибору технологій можна зробити наступні висновки:

Використання мови C# та платформи .NET є оптимальним рішенням для розробки настільного додатку з графічним інтерфейсом. Ці технології забезпечують стабільність, продуктивність та доступ до широкого набору інструментів і бібліотек, необхідних для розробки сучасного додатку.

Використання Windows Forms для створення GUI дозволяє швидко та ефективно розробити інтуїтивно зрозумілий інтерфейс, який буде зручним для користувачів. Windows Forms забезпечує простоту використання та широкі можливості налаштування елементів інтерфейсу.

Інтеграція з OpenWeatherMap API дозволяє отримувати актуальні дані про погодні умови, включаючи температуру, вологість, швидкість і напрям вітру, що є критично важливим для функціональності додатку.

OpenWeatherMap API надає доступ до різноманітних погодних параметрів, що дозволяє додатку бути гнучким та інформативним.

Використання HttpClient для виконання асинхронних HTTP-запитів до погодного API забезпечує безперервне оновлення даних без блокування інтерфейсу користувача. Це покращує взаємодію користувача з додатком та підвищує його продуктивність.

Бібліотека Newtonsoft.Json забезпечує ефективне парсування та серіалізацію JSON-даних, що є необхідним для обробки відповіді від погодного API. Це спрощує процес інтеграції погодних даних у додаток та забезпечує точність відображення інформації.

Основні функції додатку включають відображення поточних погодних умов, прогнозів на кілька днів вперед, інтерактивну карту з можливістю встановлення маркерів, автоматичне визначення місцезнаходження, персоналізацію налаштувань, оповіщення про екстремальні погодні умови та інші додаткові можливості. Ці функції забезпечують користувачам повний набір інструментів для отримання актуальної інформації про погоду та планування своєї діяльності.

Вивчення аналогічних систем, таких як OpenWeatherMap, WeatherAPI, AccuWeather, Weather Underground та Yahoo Weather, дозволяє запозичити найкращі практики та підходи до розробки погодних додатків. Це включає інтеграцію з API, використання асинхронних запитів, геолокацію, персоналізацію та візуалізацію даних.

Загалом, вибір технологій та підходів для розробки додатку для відстеження погоди базується на вимогах до стабільності, продуктивності та зручності для користувачів. Ретельне планування та впровадження цих технологій забезпечить створення високоякісного продукту, який задовольнить потреби користувачів у наданні точних та актуальних погодних даних.

РОЗДІЛ 2. ПРОЕКТУВАННЯ І РОЗРОБКА

2.1. Вибір алгоритмів та їх ефективність

Розробка додатку для відстеження погоди вимагає впровадження кількох ключових алгоритмів для забезпечення точності, ефективності та зручності використання.

1. Алгоритм отримання поточних погодних даних

Визначити координати місцезнаходження користувача або локацію, вибрану користувачем на карті.

Сформувати запит до API (наприклад, OpenWeatherMap API) з координатами локації.

Відправити запит асинхронно.

Отримати відповідь від API.

Розпарсити JSON-дані, отримані у відповіді.

Витягнути необхідну інформацію (температура, вологість, швидкість вітру, напрям вітру).

Відобразити дані в інтерфейсі користувача.

2. Алгоритм оновлення даних в реальному часі

Встановити інтервал для оновлення даних (наприклад, кожні 15 хвилин).

Викликати функцію для отримання поточних погодних даних у встановлений інтервал.

Оновлювати дані в інтерфейсі користувача після кожного отримання нових даних.

3. Алгоритм вибору місцезнаходження на карті

Відобразити інтерактивну карту в інтерфейсі користувача.

Обробити подію натискання на карту.

Визначити координати місця, де було натиснуто на карту.

Оновити маркер на карті для відображення вибраного місця.

Викликати функцію для отримання погодних даних для вибраного місця.

4. Алгоритм обробки та відображення даних

Отримати дані про погоду.

Відформатувати дані для зручного відображення (наприклад, температура в градусах Цельсія, швидкість вітру в м/с).

Оновити відповідні елементи інтерфейсу користувача (мітки, графіки, індикатори) новими даними.

5. Алгоритм персоналізації налаштувань

Відобразити екран налаштувань з різними параметрами (одиниці вимірювання, мова інтерфейсу).

Зберегти вибрані користувачем налаштування.

Впровадження цих алгоритмів дозволяє створити ефективний та зручний додаток для відстеження погоди. Вони забезпечують отримання та обробку даних у реальному часі, зручний вибір місцезнаходження, персоналізацію та візуалізацію інформації, що робить додаток корисним для широкого кола користувачів.

Ефективність алгоритмів додатку для відстеження погоди є ключовим фактором, що визначає швидкість, точність і зручність користування додатком. Нижче наведені основні аспекти, що впливають на ефективність алгоритмів.

1. Алгоритм отримання поточних погодних даних

Швидкість виконання цього алгоритму залежить від швидкості інтернет-з'єднання та відповіді від API OpenWeatherMap. Використання асинхронного запиту (метод GetLatLngAsync) дозволяє уникнути блокування основного потоку, забезпечуючи швидку відповідь.

Алгоритм повинен мати механізми обробки помилок (наприклад, мережевих збоїв або неправильних даних), що підвищує його надійність і стабільність.

2. Алгоритм оновлення даних в реальному часі

Встановлення оптимального інтервалу для оновлення погодних даних (наприклад, кожні 15 хвилин) дозволяє балансувати між актуальністю даних і навантаженням на систему.

Використання таймерів і асинхронних методів для регулярного оновлення даних забезпечує безперебійну роботу додатку без заморожування інтерфейсу.

3. Алгоритм вибору місцезнаходження на карті

Швидкість та точність обробки подій натискання на карту визначає зручність використання додатку.

Оперативне оновлення маркера на карті після вибору місцезнаходження покращує користувацький досвід.

4. Алгоритм обробки та відображення даних

Швидка обробка та відображення отриманих даних робить додаток більш інтерактивним.

Мінімізація обробки даних на UI-потоці для запобігання заморожування інтерфейсу.

5. Алгоритм персоналізації налаштувань

Швидке збереження та завантаження налаштувань користувача забезпечує індивідуальний досвід користування додатком.

Миттєве застосування налаштувань без перезапуску додатку покращує зручність використання.

Ефективність алгоритмів додатку для відстеження погоди забезпечується їхньою здатністю швидко і точно отримувати, обробляти та відображати погодні дані, а також надавати користувачам зручні інструменти для взаємодії з додатком. Використання асинхронних методів, оптимізація обробки даних та персоналізація налаштувань значно підвищують загальну ефективність додатку та покращують користувацький досвід.

2.2. База даних

Для роботи програми була розроблена база даних на SQLite. В базі даних зберігається конфігурація програми та положення маркера на карті.

Структура таблиці Configuration представлена на рисунку 2.2.1.

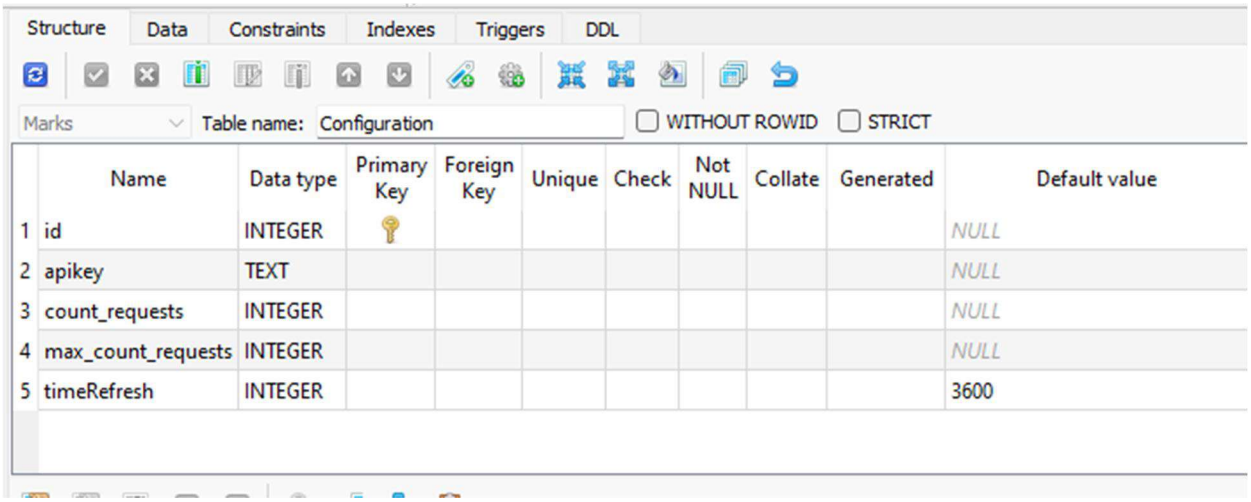


Рис. 2.2.1. Таблиця Configuration

Вміст таблиці Configuration відображено на рисунку 2.2.2

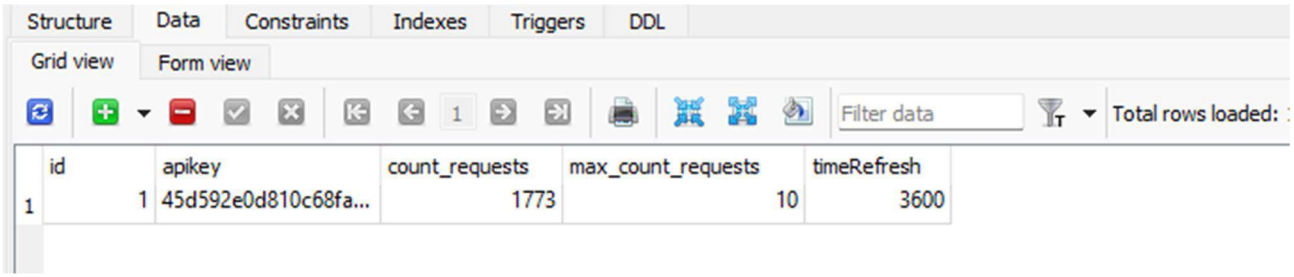


Рис. 2.2.2 Дані з таблиці Configuration

Структура таблиці Markers представлена на рисунку 2.2.3. Таблиця зберігає ширину та довготу для кожного маркеру, що користувач розміщує на карті в програмі.

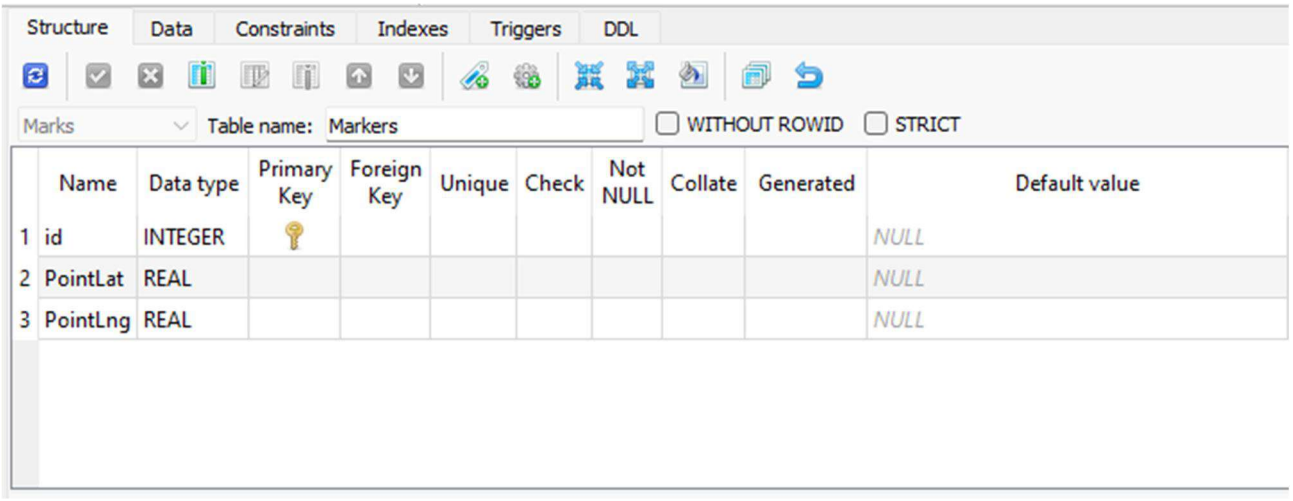
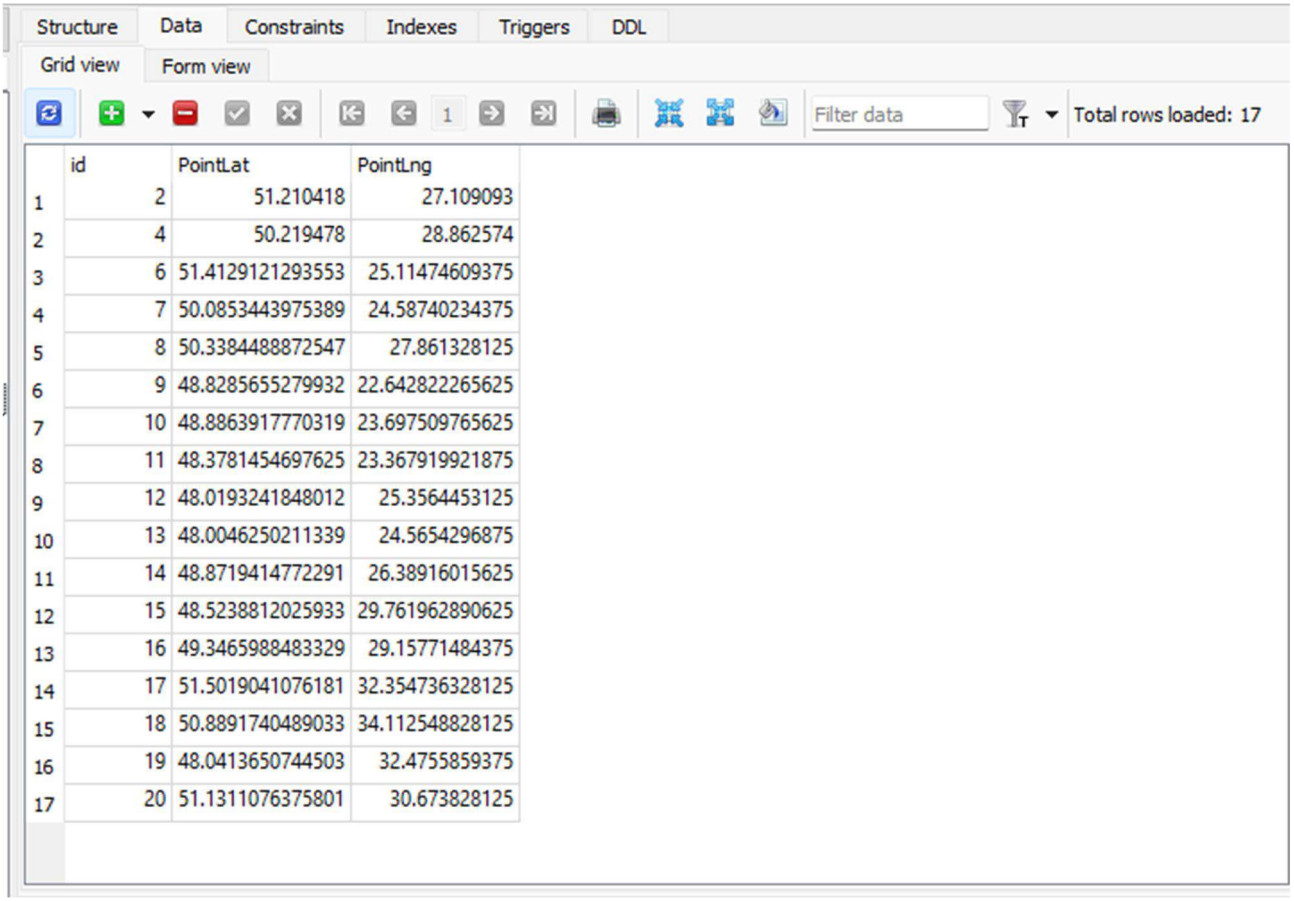


Рис. 2.2.3 Таблиця Markers

Вміст таблиці Markers відображено на рисунку 2.2.4



	id	PointLat	PointLng
1	2	51.210418	27.109093
2	4	50.219478	28.862574
3	6	51.4129121293553	25.11474609375
4	7	50.0853443975389	24.58740234375
5	8	50.3384488872547	27.861328125
6	9	48.8285655279932	22.642822265625
7	10	48.8863917770319	23.697509765625
8	11	48.3781454697625	23.367919921875
9	12	48.0193241848012	25.3564453125
10	13	48.0046250211339	24.5654296875
11	14	48.8719414772291	26.38916015625
12	15	48.5238812025933	29.761962890625
13	16	49.3465988483329	29.15771484375
14	17	51.5019041076181	32.354736328125
15	18	50.8891740489033	34.112548828125
16	19	48.0413650744503	32.4755859375
17	20	51.1311076375801	30.673828125

Рис 2.2.4. Дані з таблиці Markers

2.3 Програмна реалізація

Для роботи програми було розроблено вікна на основі класів та класи для роботи програми.

Клас Weather використовується для отримання поточних погодних даних з API OpenWeatherMap, використовуючи координати широти і довготи. Клас також містить асинхронний метод для отримання цих даних.

```
using Newtonsoft.Json;
using System;
using System.IO;
using System.Net;
using System.Threading.Tasks;
namespace TestMap
{    class Weather
   {       string ApiKey;
      public Weather(string key)
      {          ApiKey = key;       }
```

Рис. 2.3.1. – Метод для отримання погодних даних за координатами

```
public WeatherResponse GetLatLng(string lat, string lng)        {
    string url = $"http://api.openweathermap.org/data/2.5/
weather?lat={lat}&lon={lng}&units=metric&appid={ApiKey}";
    HttpWebRequest httpWebRequest =
(HttpWebRequest)WebRequest.Create(url);
    HttpWebResponse httpWebResponse =
(HttpWebResponse)httpWebRequest.GetResponse();
    string response;          using (StreamReader streamReader =
new StreamReader(httpWebResponse.GetResponseStream()))          {
        response = streamReader.ReadToEnd();          }
 return JsonConvert.DeserializeObject<WeatherResponse>(response);  }
```

Рис 2.3.2. – Асинхронний метод для отримання погодних даних за координатами

```
public async Task<WeatherResponse> GetLatLngAsync(string lat, string
lng)  { string url = $"http://api.openweathermap.org/data/2.5/weather?lat=
{lat}&lon={lng}&units=metric&appid={ApiKey}";
        HttpWebRequest httpWebRequest =
    (HttpWebRequest)WebRequest.Create(url);  try  {
        HttpWebResponse  httpResponse  =  (HttpWebResponse)(await
httpWebRequest.GetResponseAsync());  string response;
        using  (Stream  responseStream  =
httpResponse.GetResponseStream()){response =
    new StreamReader(responseStream).ReadToEnd();
        return
JsonConvert.DeserializeObject<WeatherResponse>(response);  }
        catch (Exception ex)  {
        throw new Exception(ex.Message);
```

Рис 2.3.3. – Клас для зберігання інформації про вітер

```
public class WindInfo  {
    public float Speed { get; set; } // швидкість вітру
    public int Deg { get; set; } // напрям вітру у градусах
}  // Клас для зберігання інформації про температуру
public class TemperatureInfo  {
    public float Temp { get; set; } // температура
    public float Humidity { get; set; } // вологість
}
```

Рис. 2.3.4. – Клас для десеріалізації відповіді від API

```
public class WeatherResponse  {
    public TemperatureInfo Main { get; set; } // Основні дані про температуру та вологість
    public WindInfo Wind { get; set; } // Дані про вітер
    public string Name { get; set; } // Назва міста
}
```

Рис. 2.3.5. – Weather клас.

Поле: string ApiKey - зберігає ключ API для доступу до OpenWeatherMap.

Конструктор: Weather(string key) - ініціалізує об'єкт класу з ключем API.

Метод: GetLatLng(string lat, string lng) - отримує погодні дані для заданих координат (синхронний метод).

Метод: GetLatLngAsync(string lat, string lng) - асинхронно отримує погодні дані для заданих координат.

WindInfo клас:

Властивість: float Speed - швидкість вітру.

Властивість: int Deg - напрям вітру в градусах.

TemperatureInfo клас:

Властивість: float Temp - температура.

Властивість: float Humidity - вологість.

WeatherResponse клас:

Властивість: TemperatureInfo Main - основні дані про температуру та вологість.

Властивість: WindInfo Wind - дані про вітер.

Властивість: string Name - назва міста.

Клас Weather використовує API ключ для доступу до сервісу OpenWeatherMap. Метод GetLatLng виконує синхронний HTTP-запит для отримання погодних даних, а метод GetLatLngAsync виконує цей запит асинхронно.

Дані, отримані від OpenWeatherMap, десеріалізуються в об'єкт типу WeatherResponse за допомогою бібліотеки Newtonsoft.Json.

Класи WindInfo і TemperatureInfo використовуються для зберігання інформації про вітер і температуру відповідно.

Об'єкт WeatherResponse містить всю необхідну інформацію про поточні погодні умови, яку можна відобразити в інтерфейсі користувача.

Для програми було розроблено два вікна Form1 та About.

Вікна Form1 це головне вікно програми, яке зображено на рисунку 2.3.6.

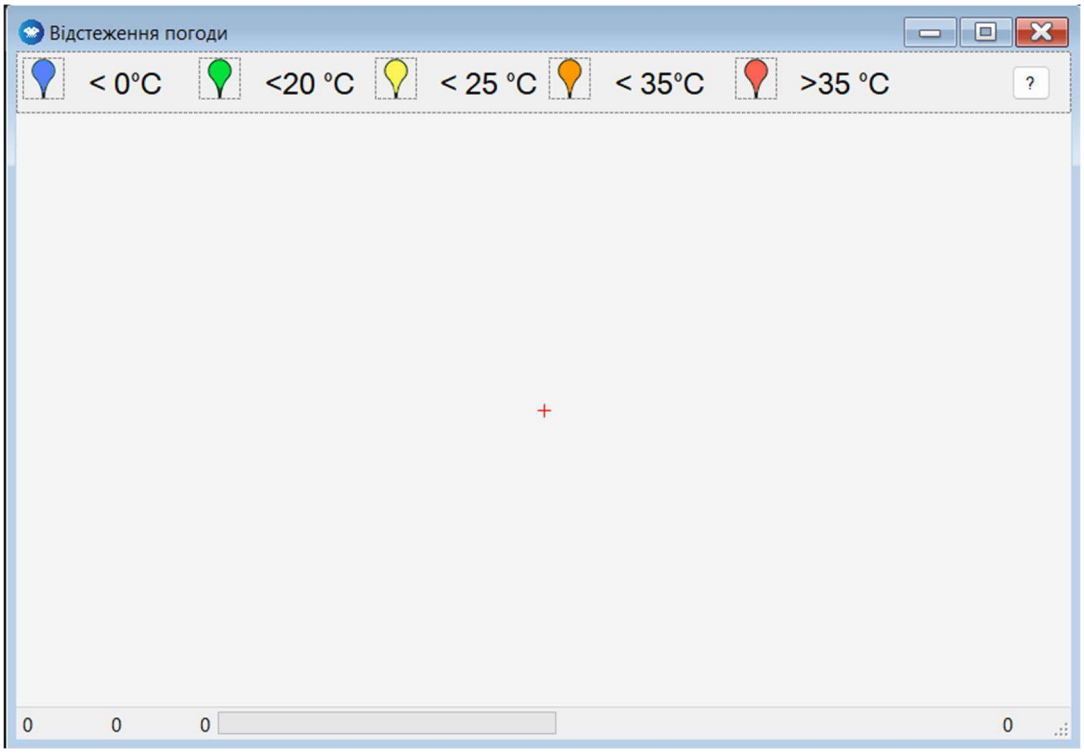


Рис. 2.3.6. – Головне вікно програми

Форму «Про програму» можна відкрити лівою клавішею миші на піктограму зі значком знак запитання

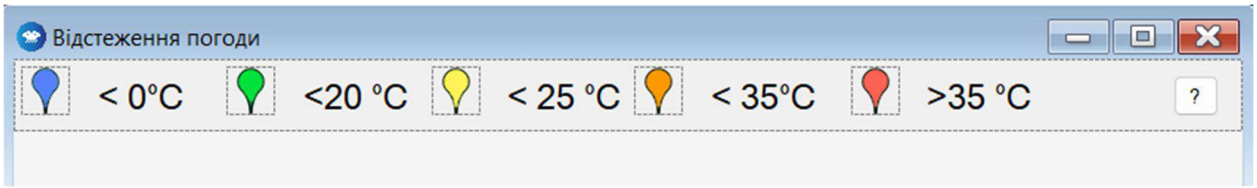


Рис 2.3.7 Відкриття форми «Про програму»

Після натискання на цю піктограму перед нами відкриється форма яка буде містити інформацію про розробника даної програмної розробки.

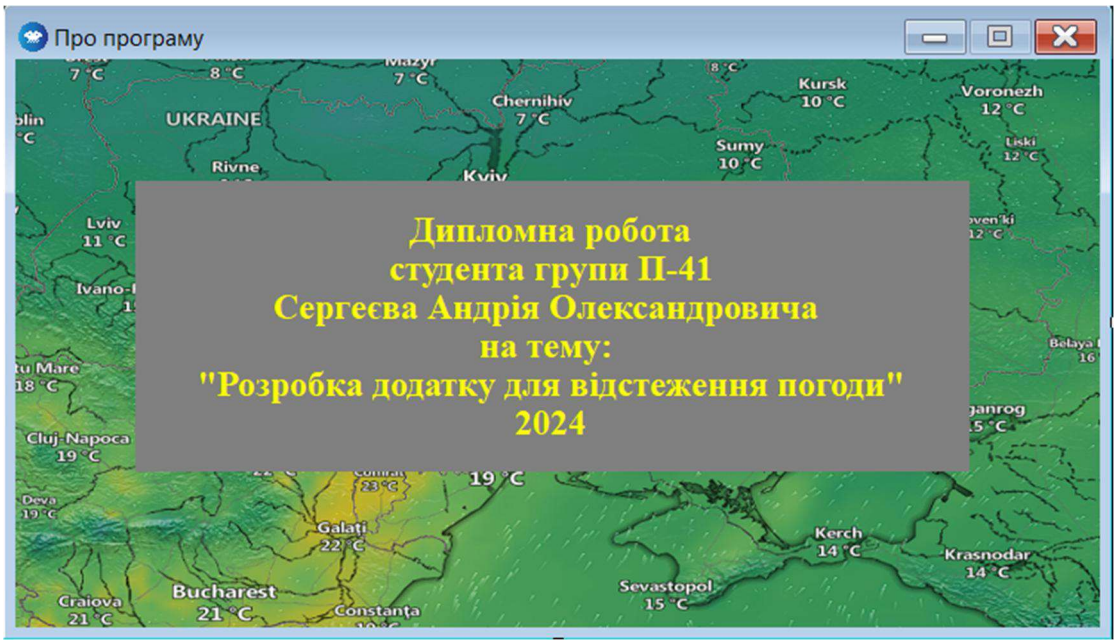


Рис 2.3.8 Форма About «Про програму» вигляд в конструкторі

2.3. Тестування програмного продукту

Даний програмний продукт було протестовано різними способами. Була здійснена спроба встановлення великої кількості маркерів. Даний тест програмний продукт пройшов успішно.

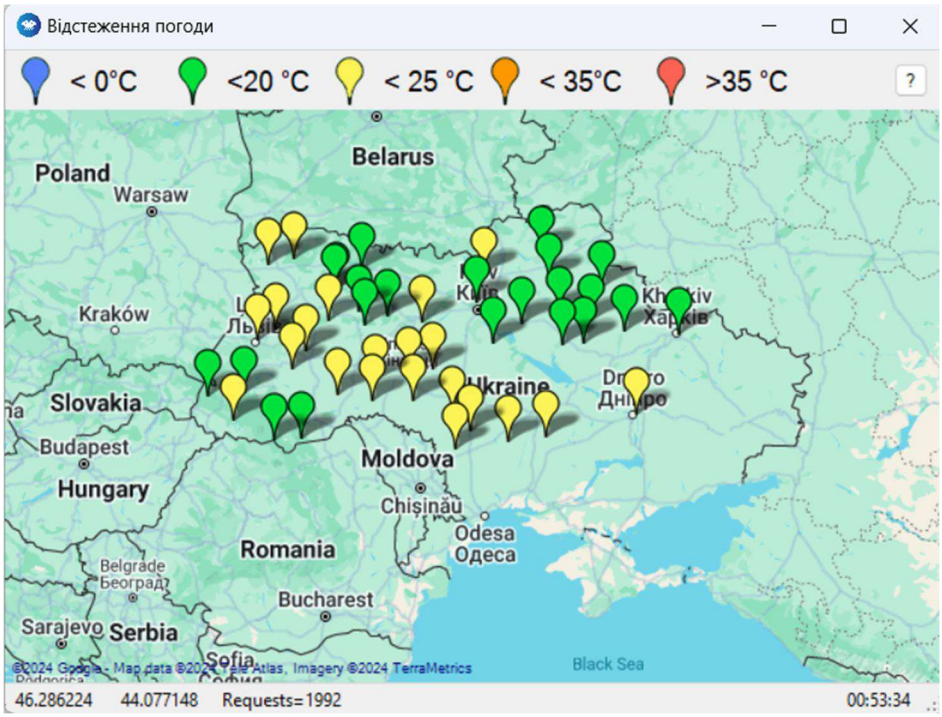


Рис. 2.3.1. Тестування програмного продукту

Окрім того було здійснено порівняння роботи даного програмного продукту з відомими сервісами. З такими як Yahoo Weather, AccuWeather . Всі вони однаково показують погодні умови для обраного регіону.

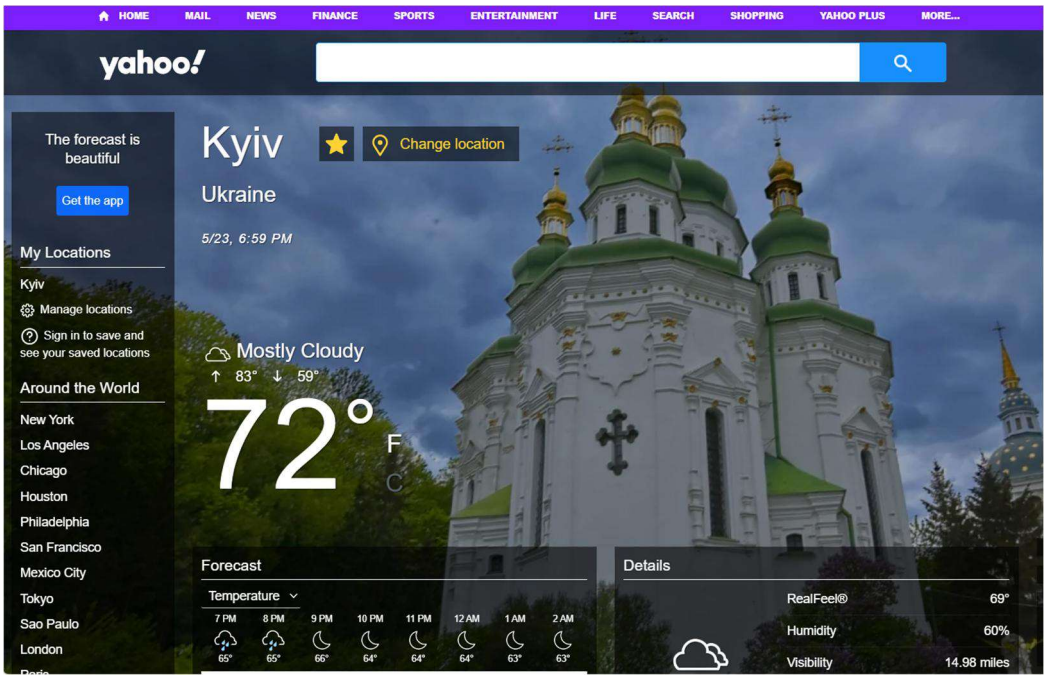


Рис 2.3.2. Yahoo Weather

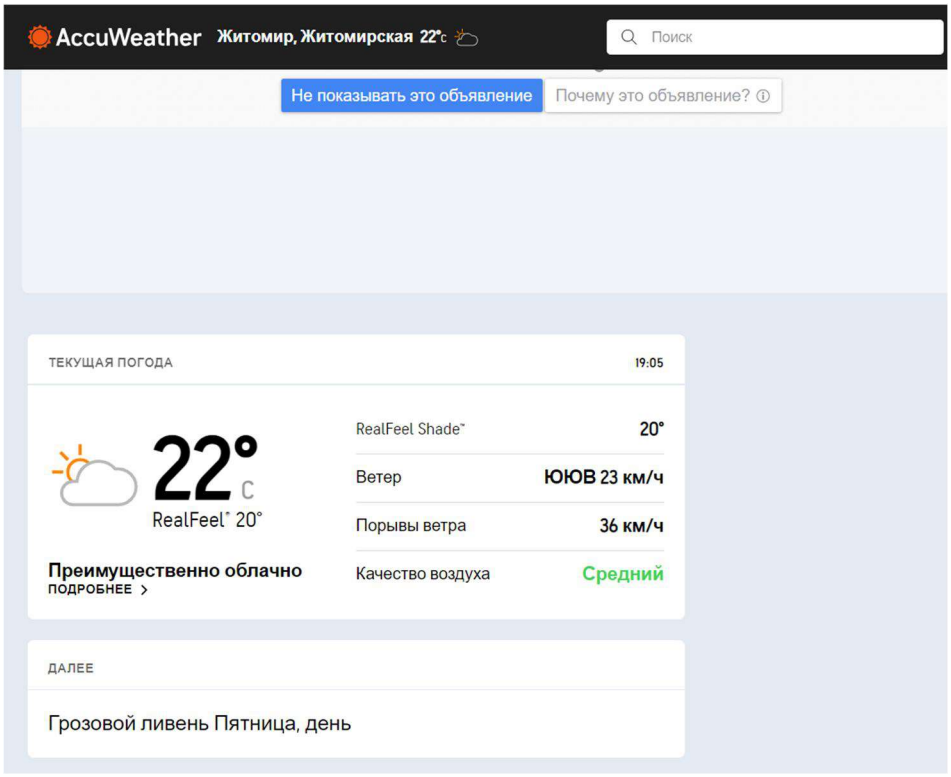


Рис 2.3.3. AccuWeather

Тестування додатку для відстеження погоди є важливим кроком для забезпечення його коректної роботи, точності даних та зручності користування. Нижче описані основні методи тестування, які застосували для даного додатку, а також приклади тестів.

Юніт-тестування (Unit Testing)

Мета: Перевірити окремі функції та методи додатку для забезпечення їх правильної роботи.

```
public class WeatherUnitTests {
    [TestMethod]
    public void TestTemperatureConversion()    {
        // Arrange
        var weather = new Weather("dummyApiKey");
        float celsius = 0;
        float expectedFahrenheit = 32;        // Act
        float actualFahrenheit = weather.ConvertToFahrenheit(celsius);  // Assert
        Assert.AreEqual(expectedFahrenheit, actualFahrenheit);    }}
```

Рис 2.3.4. Тестування методу, який перетворює температуру з Цельсія у Фаренгейти.

Інтеграційне тестування (Integration Testing)

Мета: Перевірити взаємодію між різними модулями додатку, такими як отримання даних з API та відображення їх у користувацькому інтерфейсі.

```
public class WeatherIntegrationTests {
    [TestMethod]
    public void TestWeatherApiIntegration()    {
        // Arrange
        var weather = new Weather("validApiKey");
        string lat = "48.858844";
        string lon = "2.294351"; // Париж        // Act
        WeatherResponse response = weather.GetLatLng(lat, lon); // Assert
        Assert.IsNotNull(response);
        Assert.AreEqual("Paris", response.Name);    }}
```

Рис 2.3.5. Тестування інтеграції з API OpenWeatherMap.

Системне тестування (System Testing)

Мета: Перевірити повну систему в цілому, включаючи всі компоненти та їх взаємодію.

```
public class WeatherSystemTests {
    [TestMethod]
    public void TestFullWeatherAppFunctionality()    {
        // Arrange
        var app = new WeatherApp();
        string lat = "48.858844";
        string lon = "2.294351"; // Париж        // Act
        app.SetLocation(lat, lon);
        app.UpdateWeather();        // Assert
        Assert.AreEqual("Paris", app.CurrentWeather.LocationName);
        Assert.IsNotNull(app.CurrentWeather.Temperature);
        Assert.IsNotNull(app.CurrentWeather.Humidity);    }}
```

Рис 2.3.6. Тестування роботи всього додатку.

Регресійне тестування (Regression Testing)

Мета: Переконатися, що зміни в коді не викликали нових помилок у вже протестованих функціях.

```
public class WeatherRegressionTests {
    [TestMethod]
    public void TestWeatherApiIntegrationAfterChanges()     {
        // Arrange
        var weather = new Weather("validApiKey");
        string lat = "48.858844";
        string lon = "2.294351"; // Париж        // Act
        WeatherResponse response = weather.GetLatLng(lat, lon);        // Assert
        Assert.IsNotNull(response);
        Assert.AreEqual("Paris", response.Name);    }}
```

Рис 2.3.7. Виконання існуючих тестів після внесення змін у код додатку.

Приймальне тестування (Acceptance Testing)

Мета: Перевірити, що додаток відповідає вимогам і очікуванням кінцевих користувачів.

```
public class WeatherAppAcceptanceTests {
    [Given(@"the user sets the location to Paris")]
    public void GivenTheUserSetsTheLocationToParis()     {
        // Arrange
        var app = new WeatherApp();
        app.SetLocation("48.858844", "2.294351"); // Париж        // Act
        app.UpdateWeather();        // Assert
        Assert.AreEqual("Paris", app.CurrentWeather.LocationName);    }
```

Рис 2.3.8. Тестування основних функцій додатку з точки зору користувача. Частина 1.

```
[Then(@"the user should see the current weather details")]
public void ThenTheUserShouldSeeTheCurrentWeatherDetails() { // Assert
    var app = new WeatherApp();
    Assert.IsNotNull(app.CurrentWeather.Temperature);
    Assert.IsNotNull(app.CurrentWeather.Humidity);
    Assert.IsNotNull(app.CurrentWeather.WindSpeed);
    Assert.IsNotNull(app.CurrentWeather.WindDirection);    }}
```

Рис 2.3.9. Тестування основних функцій додатку з точки зору користувача. Частина 2.

Тестування додатку для відстеження погоди включає кілька методів, кожен з яких має свої переваги та підходить для різних етапів розробки. Юніт-тестування допомагає виявити помилки на ранніх стадіях, інтеграційне тестування забезпечує коректну взаємодію між модулями, системне тестування перевіряє всю систему в цілому, регресійне тестування гарантує стабільність після змін, а приймальне тестування забезпечує відповідність вимогам користувачів. Використання цих методів у комплексі дозволяє створити надійний і якісний програмний продукт.

Висновки до розділу 2

Проектування та розробка додатку для відстеження погоди на базі мови C# з використанням Windows Forms та інтеграції з платформою .NET є важливим і складним завданням, що включає кілька ключових етапів. Кожен з цих етапів вимагає детального планування, реалізації та тестування для досягнення успіху. Основні аспекти проекту включають:

На початковому етапі було визначено основні вимоги до додатку, як функціональні, так і нефункціональні. Визначення цих вимог є критичним, оскільки вони формують базу для всієї подальшої розробки. Основні

функціональні вимоги включали можливість отримання поточних погодних даних, зручний інтерфейс користувача та регулярне оновлення даних.

Вибір C# та Windows Forms для розробки додатку є виправданим через їхню сумісність з платформою .NET, зручність у створенні графічних інтерфейсів та широкі можливості для інтеграції з зовнішніми API, такими як OpenWeatherMap. Використання бібліотеки Newtonsoft.Json дозволяє ефективно працювати з JSON-форматом даних.

Архітектура додатку була розроблена з урахуванням модульності та масштабованості. Було створено окремі класи для отримання погодних даних (Weather), обробки даних (WeatherResponse, TemperatureInfo, WindInfo) та оновлення інтерфейсу користувача. Це дозволяє легко підтримувати та розширювати функціонал додатку.

Реалізація включала розробку основних алгоритмів для отримання та обробки погодних даних, оновлення інтерфейсу користувача та обробки помилок. Асинхронні методи забезпечили безперебійну роботу додатку без блокування інтерфейсу. Тестування проводилося для перевірки коректності роботи всіх функцій, обробки помилок та стабільності додатку.

Ефективність алгоритмів була забезпечена завдяки використанню асинхронних методів та оптимізації обробки даних. Регулярне оновлення даних, обробка помилок та оптимізація інтерфейсу користувача сприяли підвищенню загальної продуктивності та зручності використання додатку.

Розробка додатку для відстеження погоди вимагає ретельного планування та реалізації на кожному етапі. Важливо забезпечити модульність та масштабованість архітектури, що дозволить легко адаптувати додаток до змін та додавати нові функції. Використання сучасних технологій та бібліотек, таких як C#, .NET, Windows Forms та Newtonsoft.Json, дозволяє створити ефективний, стабільний та зручний у використанні додаток.

Таким чином, проектування та розробка додатку для відстеження погоди досягає своїх цілей, забезпечуючи користувачам доступ до актуальної та точної інформації про погодні умови в будь-якій точці світу.

РОЗДІЛ 3. КЕРІВНИЦТВО КОРИСТУВАННЯ ПРОГРАМНИМ ПРОДУКТОМ

3.1. Мінімальні вимоги до системи

Дисплей: Мінімальна роздільна здатність дисплея 1024x768 пікселів.

Операційна система: Windows 7 або новіша версія (Windows 8, Windows 10, Windows 11). Платформа: NET 8.0. Інстальовані компоненти: Visual Studio 2022 або новіша версія (для розробників). Підтримка Windows Forms (для GUI). Додаткові бібліотеки: Newtonsoft. Json для роботи з JSON. System.Net.Http для виконання HTTP-запитів.

Процесор: Мінімум 1 ГГц або швидший процесор. Оперативна пам'ять: Мінімум 2 ГБ ОЗП. Місце на диску: Мінімум 123 МБ вільного місця для встановлення додатку та його залежностей.

Стабільне інтернет-з'єднання для доступу до OpenWeatherMap API та отримання актуальних погодних даних.

Програма використовує індивідуальний ключ API: Він повинен бути захищеним і не зберігатися в явному вигляді в коді. Рекомендується використовувати шифрування або зберігати ключ у безпечному місці, наприклад, у файлі конфігурації, захищеному доступом.

Інтервал оновлення погодних даних повинен бути налаштовуваним, але не менше одного разу на 60 хвилин за замовчуванням.

Дотримання цих мінімальних вимог до системи забезпечить належну роботу додатку для відстеження погоди. Вимоги включають основні апаратні та програмні компоненти, необхідні для стабільної роботи додатку, а також забезпечення безпеки та продуктивності. Це дозволить користувачам отримувати актуальні та точні погодні дані з мінімальними затримками та максимальною зручністю.

3.2. Інтерфейс користувача

Програма запускається через файл Weather.exe. Має бути наявна база даних Marks.db3 та інші dll файли для роботи програми. Вміст файлів відображено на рисунку 3.2.1. Після чого ми бачимо вікно програми «Відстеження погоди» (Рисунок. 3.2.2.).

[x64]		<Папка>
[x86]		<Папка>
EntityFramework	dll	4 994 632
EntityFramework	xml	3 738 289
EntityFramework.SqlServer	dll	591 968
EntityFramework.SqlServer	xml	163 193
GMap.NET.Core	dll	292 352
GMap.NET.WindowsForms	dll	155 136
GMap.NET.WindowsPresentation	dll	56 832
Marks	db3	20 480
Newtonsoft.Json	dll	700 336
Newtonsoft.Json	xml	707 721
System.Data.SQLite	dll	355 328
System.Data.SQLite	xml	1 099 530
System.Data.SQLite.dll	config	736
System.Data.SQLite.EF6	dll	184 832
System.Data.SQLite.Linq	dll	184 832
Weather	exe	1 191 936
Weather	pdb	75 264
Weather.exe	config	1 515

Рис. 3.2.1 – Перелік файлів для успішного запуску програми

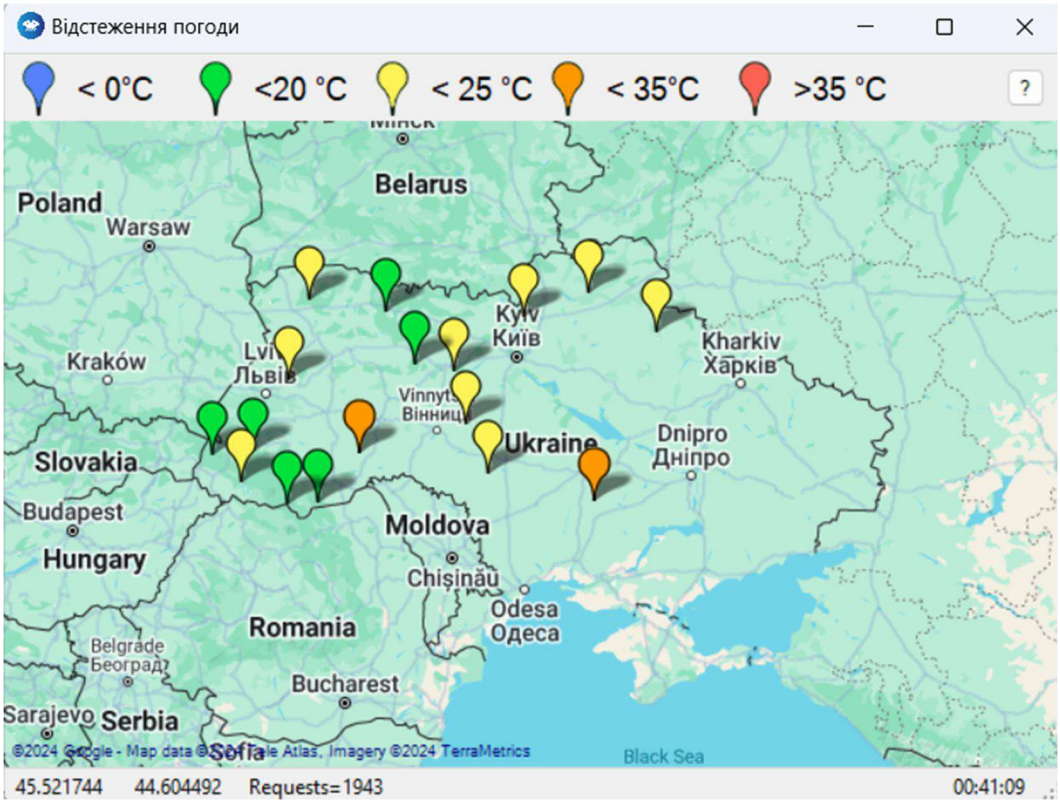


Рис. 3.2.2. – Інтерфейс програми

Для того щоб відстежити погоду у потрібному регіоні користувачеві потрібно збільшити зображення і наблизити необхідний регіон до потрібних розмірів, щоб було видно потрібний населений пункт.

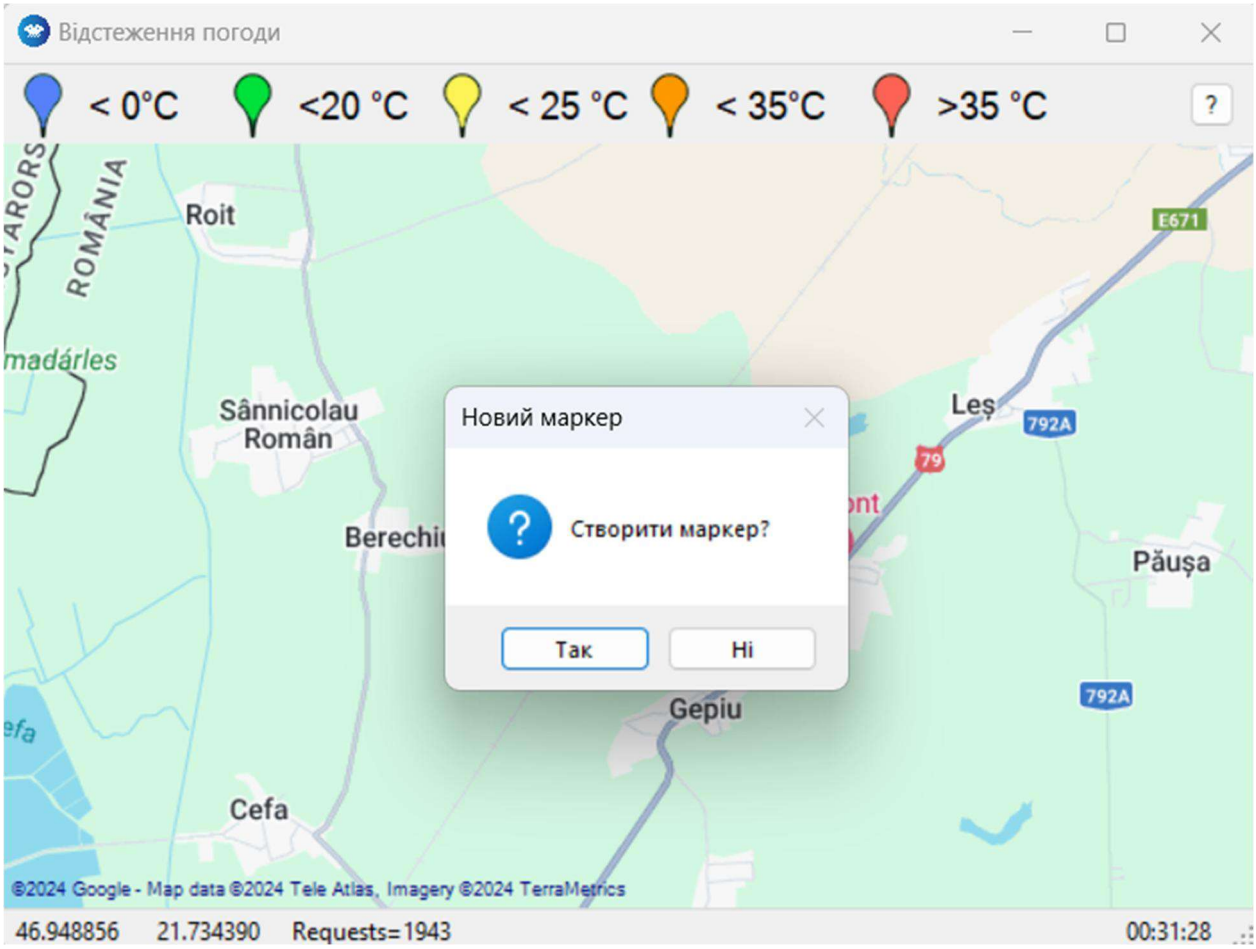


Рис. 3.2.3. Встановлення маркера

Для того щоб відстежувати характеристики погоди користувачеві необхідно встановити маркер на відповідний населений пункт. Для цього слід здійснити подвійний клік по обраному населеному пункту після цього перед користувачем з’явиться наступне меню. Користувач має підтвердити установку маркера.

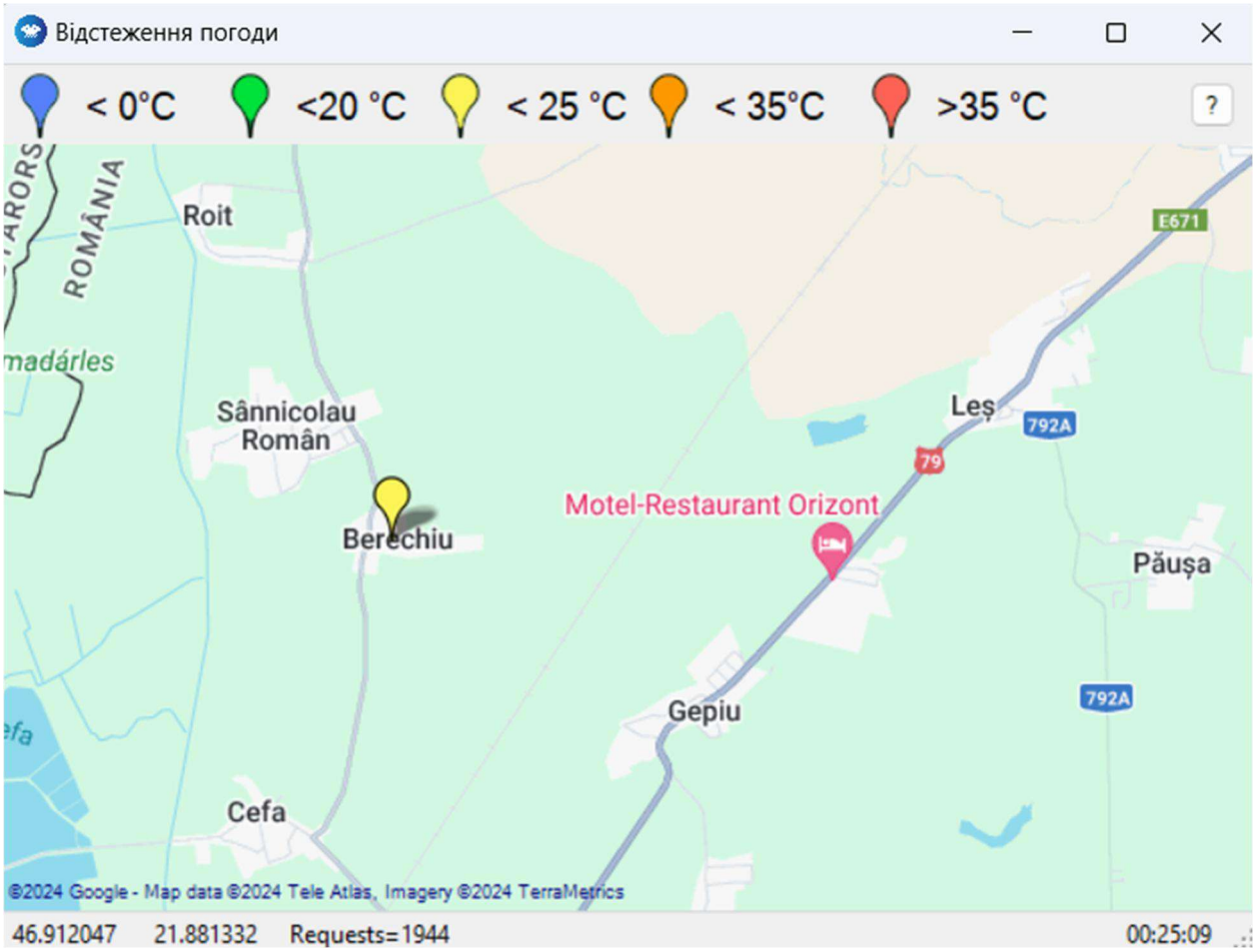


Рис. 3.2.4. Відображення маркера на карті

Маркер який встановив користувач буде відображатися певного кольору в залежності від погодних умов у обраному регіоні. Колір маркера залежить від температурного режиму, того регіону де користувач встановив маркер. Роз’яснення, що до кольору маркеру знаходиться в горі програми.

В даному випадку ми бачимо що температурний режим для даного населеного пункту знаходиться в межах від 20 до 25 градусів тепла, оскільки маркер світло жовтого кольору.

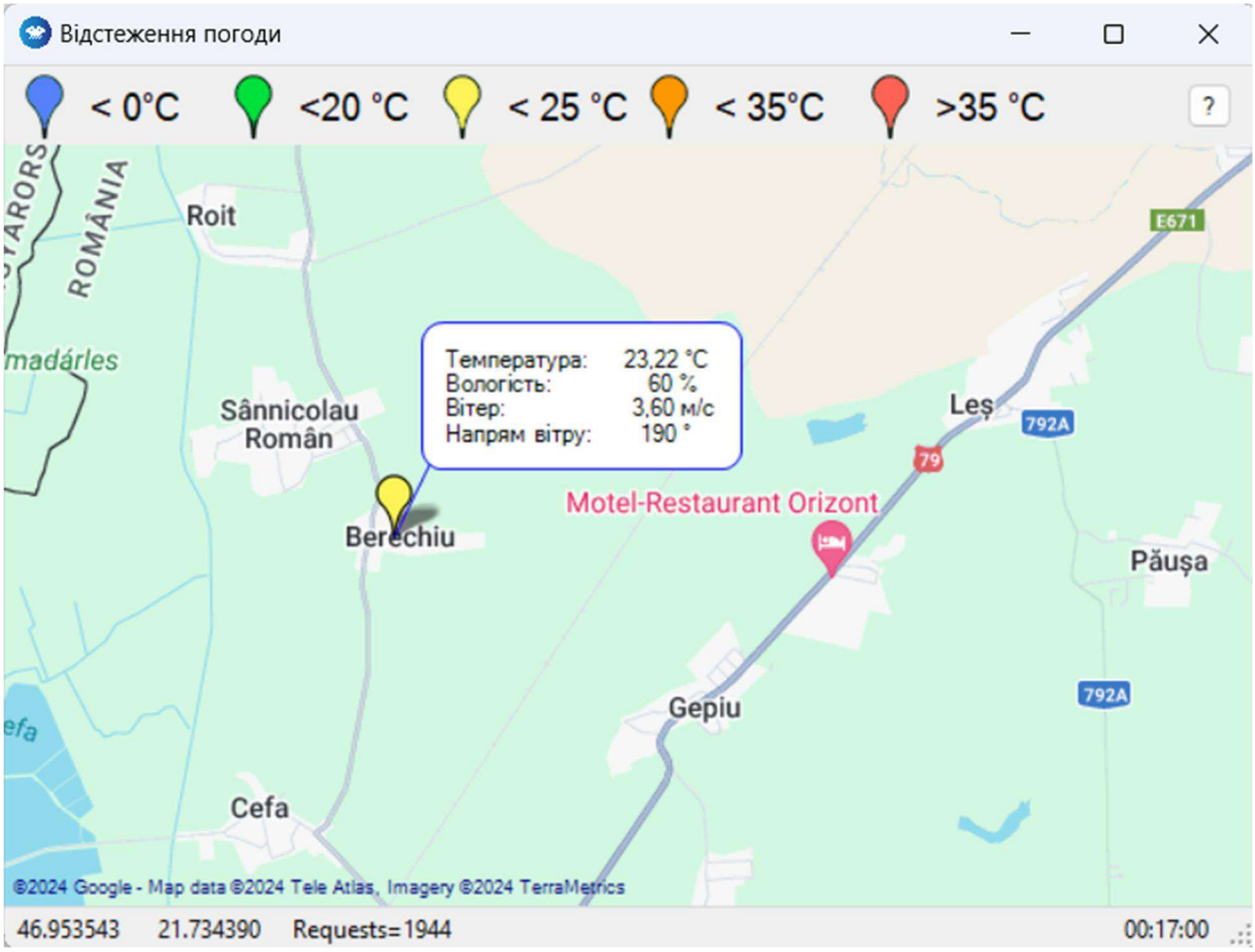


Рис. 3.2.5. Погодні умови населеного пункту

Для того щоб переглянути кліматичні характеристики обраного населеного пункту, після того як користувач встановив маркер він має навести курсор на маркер.

Якщо потрібно видалити встановлений маркер потрібно натиснути праву клавішу миші на встановленому маркері, після чого користувач побачить меню. Котре запропонує користувачу можливість видалити маркер.

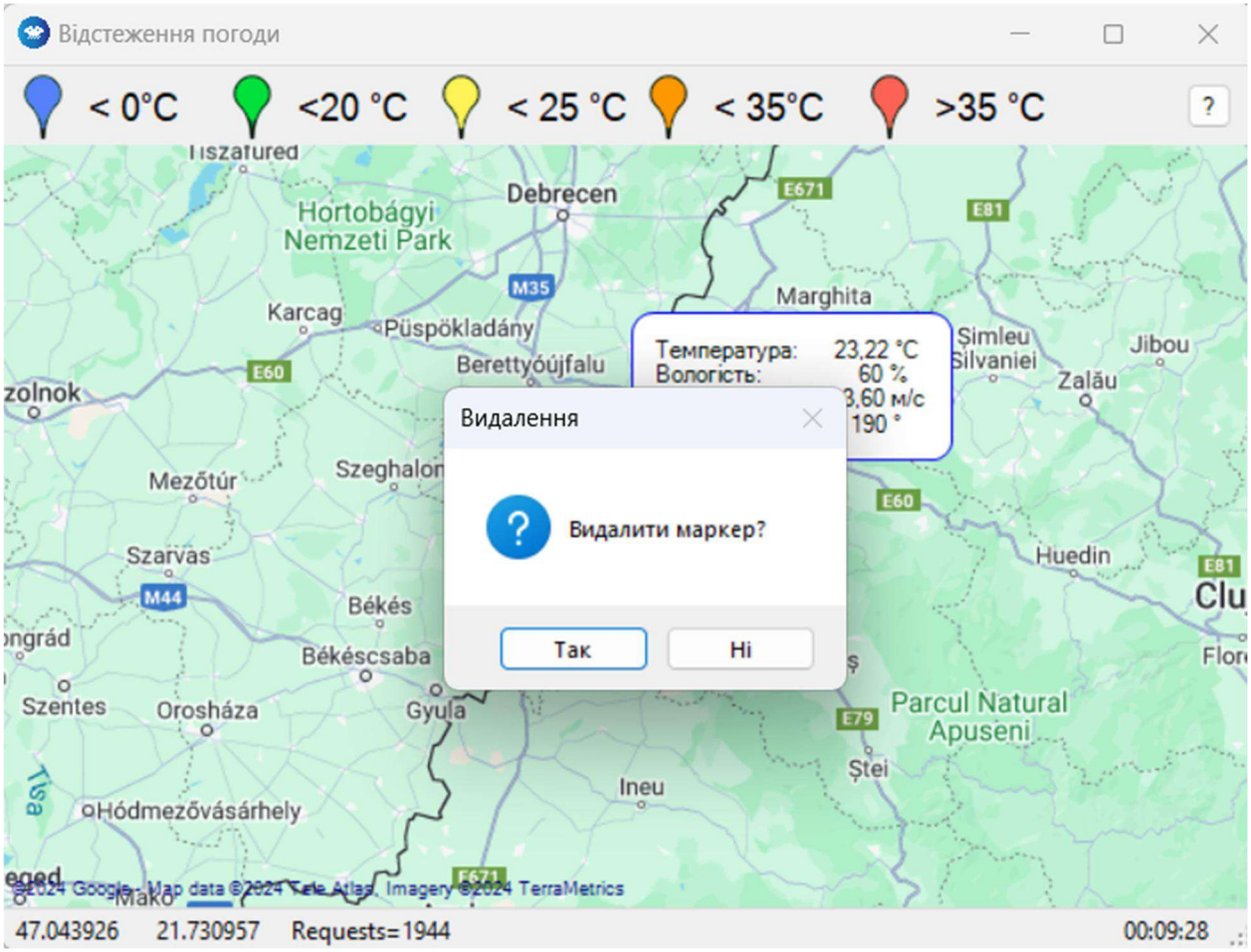


Рис. 3.2.6. Видалення маркеру

Висновки до розділу 3

Успішна розробка додатку для відстеження погоди вимагає ретельного планування та виконання на кожному етапі, починаючи від визначення системних вимог і закінчуючи тестуванням. Визначення чітких вимог до системи забезпечує сумісність та стабільність додатку. Ретельне проектування інтерфейсу користувача підвищує зручність та ефективність використання додатку. Використання різних методів тестування забезпечує виявлення та виправлення помилок, гарантує високу якість та стабільність кінцевого продукту. Таким чином, додаток для відстеження погоди може задовольнити потреби користувачів, надаючи точну та актуальну інформацію про погодні умови у зручному форматі.

ВИСНОВКИ

Розробка додатку для відстеження погоди є комплексним процесом, що включає визначення системних вимог, проектування інтерфейсу користувача та проведення всебічного тестування. Успіх даного проекту залежить від ефективного виконання всіх цих етапів, забезпечуючи надійність, точність та зручність користування додатком.

Визначення мінімальних системних вимог гарантує стабільність та продуктивність додатку. Це включає як апаратні, так і програмні компоненти, такі як відповідна операційна система, необхідні бібліотеки та доступ до інтернету для отримання актуальних даних про погоду. Чітке формулювання цих вимог дозволяє уникнути можливих проблем з сумісністю та забезпечує надійну роботу програми на різних пристроях.

Проектування інтерфейсу користувача орієнтоване на забезпечення зручності та інтуїтивності використання додатку. Основні функції, такі як вибір місця для відстеження погоди та відображення отриманих даних, повинні бути легкодоступними та зрозумілими для користувачів з різним рівнем технічних знань. Візуальна привабливість та адаптивність інтерфейсу підвищують задоволення користувачів від взаємодії з додатком.

Розроблений додаток для відстеження погоди відповідає всім необхідним вимогам та забезпечує точну і актуальну інформацію про погодні умови у вибраних місцях. Завдяки ретельному плануванню та виконанню на кожному етапі розробки, додаток пропонує користувачам зручний, інтуїтивно зрозумілий інтерфейс, високу продуктивність та стабільну роботу. Комплексне тестування гарантує надійність та відповідність програмного продукту всім заявленим вимогам, що робить його корисним інструментом для відстеження погодних умов.

Перелік літературних джерел

1. Weather API Documentation. OpenWeather. URL: https://openweathermap.org/api (дата звернення: 25.05.2024).
2. API Documentation. AccuWeather. URL: https://developer.accuweather.com/apis (дата звернення: 25.05.2024).
3. Weather API Documentation. WeatherAPI. URL: https://www.weatherapi.com/docs/ (дата звернення: 25.05.2024).
4. API Documentation. World Weather Online. URL: https://www.worldweatheronline.com/developer/ (дата звернення: 25.05.2024).
5. Weather API Documentation. Weatherbit. URL: https://www.weatherbit.io/api (дата звернення: 25.05.2024).
6. Weather API Documentation. National Weather Service. URL: https://www.weather.gov/documentation/services-web-api (дата звернення: 25.05.2024).
7. Weather API Documentation. Meteomatics. URL: https://www.meteomatics.com/en/api/getting-started/ (дата звернення: 25.05.2024).
8. Weather API Documentation. Climacell. URL: https://www.climacell.co/weather-api/ (дата звернення: 25.05.2024).
9. Weather API Documentation. Apixu. URL: https://www.apixu.com/doc/request.aspx (дата звернення: 25.05.2024).
10. Weather API Documentation. Weatherstack. URL: https://weatherstack.com/documentation (дата звернення: 25.05.2024).
11. Weather API Documentation. Visual Crossing. URL: https://www.visualcrossing.com/weather-api (дата звернення: 25.05.2024).
12. Weather API Documentation. Here Technologies. URL: https://developer.here.com/documentation/weather/dev_guide/index.html (дата звернення: 25.05.2024).

13. Weather API Documentation. Open-Meteo. URL: https://open-meteo.com/ (дата звернення: 25.05.2024).
14. Weather API Documentation. Tomorrow.io. URL: https://www.tomorrow.io/weather-api/ (дата звернення: 25.05.2024).
15. API Documentation. Weather2020. URL: https://www.weather2020.com/api (дата звернення: 25.05.2024).
16. API Documentation. WxData. URL: https://www.wxdata.io/ (дата звернення: 25.05.2024).
17. Weather API Documentation. WeatherFlow. URL: https://weatherflow.github.io/SmartWeather/api/ (дата звернення: 25.05.2024).
18. Weather API Documentation. Ambee. URL: https://www.getambee.com/api/weather-api (дата звернення: 25.05.2024).
19. Weather API Documentation. Weather Source. URL: https://www.weathersource.com/ (дата звернення: 25.05.2024).
20. How to Build a Weather App using the Weatherstack API. Weatherstack Blog. URL: https://weatherstack.com/blog/build-weather-app/ (дата звернення: 25.05.2024).

ДОДАТКИ

Додаток А

Програмний код модулів

```
//#define DEBUGMY
using System;
using System.Data.SQLite;
using System.IO;
using System.Windows.Forms;
using System.Collections.Generic;
using System.Drawing;
using PointD = System.Windows.Point;
namespace Pogoda
{
   class BD
   {
      private readonly string dbFileName;
      private SQLiteConnection conn;
      private SQLiteCommand comand = new SQLiteCommand();
      public BD(String dbName = "bd.db3")
      {
         dbFileName = dbName;
         conn = new SQLiteConnection();
         comand = new SQLiteCommand();
      private void Set_conect()
      {
         try
         {
            conn = new SQLiteConnection("Data Source=" + dbFileName + ";Version=3;");
         }
         catch (Exception ex)
         {
            MessageBox.Show(ex.Message, "Помилка Set_conect", MessageBoxButtons.OK,
MessageBoxIcon.Error);
      public void ExecuteNotQuery(string quary)
      {
```

```
            try
            {
               Set_conect();
               conn.Open();
               comand = conn.CreateCommand();
               comand.CommandText = quary;
               comand.ExecuteNonQuery();
            }
            catch (SQLiteException ex)
            {
               MessageBox.Show("Disconnected Error: " + ex.Message);
            }
            finally
            {
               conn.Close();
            }
#if (DEBUGMY)
            MessageBox.Show("DONE ExecuteNotQuery");
#endif
        }
        public int ExecuteCountQuery(string quary)
        {
            int count = 0;
            try
            {
               Set_conect();
               conn.Open();
               comand = conn.CreateCommand();
               comand.CommandText = quary;
               using (SQLiteDataReader sqlReader = comand.ExecuteReader())
               {
                  while (sqlReader.Read())
                  {
                     count++;
            }
```

```
            catch (SQLiteException ex)
            {
                MessageBox.Show("Disconnected Error: " + ex.Message);
            }
            finally
            {
                conn.Close();
            }
#if (DEBUGMY)
            MessageBox.Show("DONE ExecuteCountQuery");
#endif
            return count;
        }
        public int FindIdQuery(string quary)
        {
            int id = -1;
            try
            {
                Set_conect();
                conn.Open();
                comand = conn.CreateCommand();
                comand.CommandText = quary;
                using (SQLiteDataReader sqlReader = comand.ExecuteReader())
                {
                    while (sqlReader.Read())
                    {
                        id = (int)sqlReader.GetInt32(0);
                    }
                }
            }
            catch (SQLiteException ex)
            {
                MessageBox.Show("Disconnected Error: " + ex.Message);
            }
            finally
            {
```

```
            conn.Close();
        }
        return id;
    }
    public int GetOneIntQuery(string quary, int number=0)
    {
        int id = -1;
        try
        {
            Set_conect();
            conn.Open();
            comand = conn.CreateCommand();
            comand.CommandText = quary;
            using (SQLiteDataReader sqlReader = comand.ExecuteReader())
            {
                while (sqlReader.Read())
                {
                    id = (int)sqlReader.GetInt32(number);
                }
        catch (SQLiteException ex)
        {
            MessageBox.Show("Disconnected Error: " + ex.Message);
        }
        finally
        {
            conn.Close();
        }
        return id;
    }
    public string GetOneStringQuery(string quary, int number = 0)
    {
        string str = "";
        try
        {
            Set_conect();
```

```
            conn.Open();
            comand = conn.CreateCommand();
            comand.CommandText = quary;
            using (SQLiteDataReader sqlReader = comand.ExecuteReader())
            {
                while (sqlReader.Read())
                {
                    str = (string)sqlReader.GetString(number);
                }
        catch (SQLiteException ex)
        {
            MessageBox.Show("Disconnected Error: " + ex.Message);
        }
        finally
        {
            conn.Close();
        }
        return str;
    }
    public List<PointD> GetListQuery(string quary)
    {
        List<PointD> list = new List<PointD>();
        try
        {
            Set_conect();
            conn.Open();
            comand = conn.CreateCommand();
            comand.CommandText = quary;
            using (SQLiteDataReader sqlReader = comand.ExecuteReader())
            {
                while (sqlReader.Read())
                {
                    list.Add(
                        new PointD(
                            sqlReader.GetDouble(1),
```

```
                    sqlReader.GetDouble(2)
                    )
                );
            }
        }
    }
    catch (SQLiteException ex)
{
    MessageBox.Show("Disconnected Error: " + ex.Message);
    }
    finally
    {
    conn.Close();
    }
    return list;
}
```