Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

Навчально-науковий інститут деревообробних та
комп'ютерних технологій і дизайну
(повне найменування інституту, назва факультету(відділення))

Кафедра інформаційних систем і комп'ютерного моделювання
(повна назва кафедри (предметної циклової комісії))

Пояснювальна записка

до дипломної роботи

перший (бакалаврський)

(рівень вищої освіти)

на тему: «Розроблення інформаційної системи для сповіщень про погодні умови засобами Angular»

Виконав студент 4 курсу, групи ІСТ-41
спеціальності: 126
„Інформаційні системи та технології"
(шифр і назва напрямку підготовки спеціальності)

Грищук Назар Олександрович
(прізвище, ініціали)

Керівник: ст.викл. Бекас Б.О.,
доц. Олянишин Т.В.
(прізвище, ініціали)

Рецензент: Олянишин С.І.
(прізвище, ініціали)

Львів-2023

Національний лісотехнічний університет України

(повне найменування вищого навчального закладу)

ННІ деревооброблювальних та комп'ютерних технологій і дизайну

Кафедра Інформаційних систем і комп'ютерного моделювання

Рівень вищої освіти перший (бакаравський)

Спеціальність 126 *„Інформаційні системи та технології"*

ЗАТВЕРДЖУЮ:

В.о завідувача кафедри ІСКМ

_____Сторожук О.Л.

„21" 11 2022.

ЗАВДАННЯ

НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Грищук Назар Олександрович

(прізвище, ім'я, по батькові)

1.Тема магістерської роботи: Розроблення інформаційної системи для спощень про погодні умови засобами Angular.

керівник роботи: ст.викл. Бекас Б.О., доц. Олянишин Т.В.,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від "21" 11.2022 року, №С-521

2.Термін подання студентом проекту(роботи) 10 червня 2023р

3. Вихідні дані до проекту (роботи) Розробити програмне забезпечення для сповіщення погодніх умов відносно геолокації. Для розробки використати фрейсворк Angular, Bootstrap та APIXU API.

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

Стан проблемної області

Інформаційне забезпечення

Програмне забезпечення

Висновки

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

Підготовка матеріалу до доповіді .

6. Консультанти розділів проекту (роботи)

7. Дата видачі завдання 23 листопада 2022р.

КАЛЕНДАРНИЙ ПЛАН

№, з/п	Етапи бакалаврської роботи	Термін виконання етапів роботи	Примітка
1.	Огляд літератури згідно досліджуваної теми. Збір необхідних матеріалів.	23.11-20.12	Виконано
2.	Постановка задачі і її формалізація	20.12-13.01	Виконано
3.	Виконання вхідного етапу технології	13.01-14.04	Виконано
4.	Реалізація головних класів проекту	14.04-20.05	Виконано
5.	Виконання етапу відлагодження проєкту	20.05.-25.05.	Виконано
6.	Виконання етапу впровадження та випуску бета-версії.	25.05.-02.06.	Виконано
7.	Оформлення записки до дипломного проекту.	02.06.-10.06.	Виконано

Студент _____ Грищук Н.О.
(підпис) (прізвище та ініціали)

Керівник роботи
_____ ст. викл. Бекас Б.О.,
_____ доц. Олянишин Т.В.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Дипломна робота містить *42* сторінок пояснювальної записки, *10* рисунків, *1* додаток, *15* джерел.

Дипломна робота присвячена розробці програмного забезпечення для відображення погоди, використовуючи Angular, Bootstrap та APIXU API. Також, на функціоналі, який після визначення місцезнаходження у формі пошуку, застосунок буде відображати поточні погодні дані для введеного місця. Для реалізації використати версію Angular 7.2.0 та версію Bootstrap 4.2.1. Використовуючи APIXU, користувач може отримати оновлену інформацію про погоду, а також прогнози на майбутні дні для будь-якого місця на планеті.

Ключові слова: Angular, Bootstrap, API, програмне забезпечення.

ABSTRACT

Thesis contains *42* pages of explanatory note, *10* drawings, *1* appendix, *15* sources.

The thesis is devoted to the development of weather display software using Angular, Bootstrap and APIXU API. Also, on the functionality that after determining the location in the search form, the application will display the current weather data for the entered location. For implementation, use version Angular 7.2.0 and version Bootstrap 4.2.1. Using APIXU, the user can get updated weather information as well as forecasts for the coming days for any place on the planet.

Keywords: Angular, Bootstrap, API, software.

ТЕХНІЧНЕ ЗАВДАННЯ

Розробити програмне забезпечення для відображення погоди, використовуючи Angular, Bootstrap та APIXU API. Створити функціонал, який після визначення місцезнаходження у формі пошуку, застосунок буде відображати поточні погодні дані для введеного місця. Для реалізації використати версію Angular 7.2.0 та версію Bootstrap 4.2.1. Використовуючи APIXU, користувач може отримати оновлену інформацію про погоду, а також прогнози на майбутні дні для будь-якого місця на планеті.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

API – Application Programming Interface, інтерфейс програмування застосунків для взаємодії між різними програмними компонентами.

APIXU – сервіс погодного API для отримання метеорологічних даних через мережу Інтернет.

CLI – Command Line Interface, інтерфейс командного рядка для взаємодії користувача з програмним забезпеченням.

CSS – Cascading Style Sheets, мова стилів для оформлення веб-сторінок.

HTML – HyperText Markup Language, мова розмітки гіпертекстових документів для створення веб-сторінок.

JSON – JavaScript Object Notation, текстовий формат обміну даними між клієнтом та сервером.

MVC – Model–View–Controller, архітектурний шаблон програмування, який розділяє логіку додатку на модель, представлення та контролер.

MVVM – Model–View–ViewModel, архітектурний шаблон побудови інтерфейсів користувача.

NOAA – National Oceanic and Atmospheric Administration, національне управління океанічних і атмосферних досліджень США.

GFS – Global Forecast System, глобальна система прогнозування погоди, яка використовується для створення метеорологічних прогнозів.

SPA – Single Page Application, односторінковий веб-додаток, у якому сторінка оновлюється без повного перезавантаження.

PWA – Progressive Web Application, веб-додаток, що поєднує можливості веб-сайтів і мобільних застосунків.

UI – User Interface, інтерфейс користувача.

ВСТУП

Більшість з нас отримують хоча б деяку інформацію про погоду з додатків. Вони зручні, прості у використанні та оновлюються набагато частіше, ніж очікування п'ятигодинних новин. Існує багато різних погодних програм. Деякі з них можуть мати яскравіші користувальницькі інтерфейси, а деякі пропонують спеціальні інструменти або сповіщення, але який найточніший додаток погоди? Ця стаття допоможе вам знайти найнадійніше джерело погоди та надасть вам більше контексту про точність програми погоди.

Об’єктом дослідження використання Angular для створення погодного застосунку.

Метою роботи створення активного сервісу для повідомлень клієнтів про погодні умови.

Предметом дослідження є використання фреймворку Angular у інформаційних застосунках.

Практичне значення гнучкий інструмент інформування в мережі інтернет

РОЗДІЛ 1. СТАН ПРОБЛЕМНОЇ ОБЛАСТІ

1.1. Огляд проблемної області.

Хоча кожен може подумати про той час-два (а може й більше), що метеоролог дуже помилився, реальність така, що в 2023 році прогнози погоди в середньому надзвичайно хороші. NOAA каже, що в середньому п'ятиденний прогноз є правильним приблизно в 90% випадків, тоді як семиденний прогноз може точно передбачити погоду в 80% випадків. До десяти днів прогнози точні лише приблизно в 50% випадків.

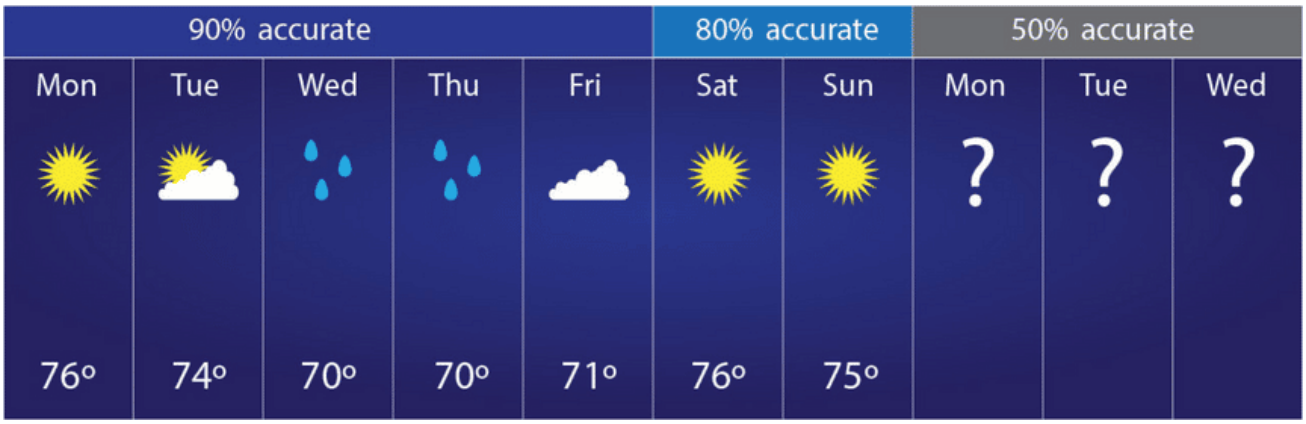


Рис.1.1. П'ятиденний і семиденний прогноз досить точний, але вони менш надійні за межами цього діапазону. (Зображення надано: NOAA)

Чому так відбувається? Відповідь криється в центральній для метеорології ідеї – теорії хаосу. Якщо ви чули якусь версію приказки «метелик махає крилами в Бразилії і змушує торнадо приземлитися в Техасі», це ідея теорії хаосу. Крихітне порушення зараз може мати величезні і несподівані наслідки пізніше, а це означає, що навіть через десять днів прогнози погоди стають майже марними.

Ми не можемо пояснити кожне крихітне порушення. Уявіть, що ви намагаєтеся відстежити кожного метелика і всі збої в повітрі, які ви можете створити в процесі. У якийсь момент ці невеликі порушення переважать будь-який сигнал прогнозу, на який ми покладалися.

1.2. Як працюють програми погоди і чому вони іноді помиляються

Як програми погоди все одно дають вам прогноз? Вони покладаються на комп'ютерні моделі атмосфери, які беруть інформацію про те, що погода робить зараз, і використовують її для прогнозування того, що може статися в майбутньому.

Оскільки комп'ютери настільки великі, погодні моделі повинні розділити світ на маленькі коробочки, щоб впоратися з проблемою. Залежно від моделі, ці коробки знаходяться від двох до шести миль з кожного боку. Це важливо, тому що модель однаково ставиться до всього, що знаходиться в цих коробках. Часто, насправді, це не так! Невелика літня гроза над Атлантою може бути лише милю завширшки, не займаючи цілої коробки і повністю пролітаючи «під радаром» з точки зору метеорологічного додатка.

Аналогічним чином, якщо ваша коробка включає 25 квадратних миль землі і 5 квадратних миль океану, середня температура буде більше схожа на внутрішню спеку, ніж на прохолодний морський бриз, який ви можете відчути на пляжі. Те ж саме справедливо і для районів гірської місцевості. Розмір цих полів і той факт, що всі програми погоди покладаються на певну їх версію, є великою причиною, чому ви можете не побачити ідеальний прогноз від свого додатка.

Незважаючи на дико мінливий рельєф на фотографії вище, погодні моделі розглядали б більшу частину цієї області як єдине поле, передбачаючи середні умови, які можуть не представляти ні гірської бази, ні вершини.

Один із способів, яким метеопрограми часто вирішують цю проблему, особливо щодо опадів, - це надання вам відсоткової ймовірності дощу або снігу. Зазвичай ці відсотки обчислюються для всієї коробки, навіть якщо реальна загроза може бути більш сконцентрована, наприклад, на вершині гори. Більш детально про ймовірність опадів, про те, як вона розраховується, і деякі поширені помилки, читайте в нашій статті про процентне значення дощу.

Інша важлива причина пов'язана з тим, як моделі погоди роблять припущення про те, що погода зараз може означати для погоди пізніше. Різні моделі (і додатки) роблять дещо різні припущення, що призводить до кардинально різних прогнозів. Більшість із цих припущень здебільшого точні, тому програми погоди, як правило, не зведуть вас занадто далеко на манівці.

Однак припущення зазвичай мають недоліки, що призводять до неточних прогнозів. Наприклад, флагманська модель погоди NOAA, GFS (Global Forecast System), яка забезпечує роботу більшості сучасних метеорологічних додатків, останнім часом має проблеми з припущенням про те, як тепло і волога розподіляються через найнижчий шар атмосфери в дуже спекотні дні в семіарідному кліматі. Більшість людей цього не помітять, але якщо ваш додаток раптом закликає до високої температури 120 ° F у Денвері, це цілком може бути пов'язано з таким помилковим припущенням!

Рис.1.2. Метеоролог Центру прогнозування погоди Ендрю Оррісон використовує дані погодних моделей. (Зображення надано: NOAA)

Оскільки сучасні метеорологічні додатки, як правило, не мають людей-метеорологів, які стежать за прогнозом у режимі реального часу, вони схильні до цих помилок припущень. Деякі сучасні програми погоди, такі як The Weather

Channel, мають рівень людського внеску, який може допомогти виявити ці проблеми, перш ніж вони знайдуть шлях до вашого телефону. Інші додатки використовують сучасну статистику або машинне навчання для виявлення прогнозних значень, які здаються недоречними, і зменшення їх кінцівок. Недоліком є те, що іноді трапляються екстремальні погодні умови! У будь-якому випадку, боротьба з деякими упередженнями, що виникають через помилкові припущення погодних моделей, є однією з значних проблем, з якими стикаються погодні програми, і однією з головних причин, чому вони можуть помилятися.

1.3. Міфи про точність моделей погоди

У кожного є своя улюблена теорія про те, чому погодні програми можуть бути неправильними. Ось деякі теорії, які ви часто чуєте або, можливо, навіть думали про себе:

- **"Вони просто не розуміють нашої унікальної погоди тут, у <вставте стан>!"**

- **«Ми занадто далеко від найближчого аеропорту!»**
- **«Спостереження за погодою тут проводяться недостатньо часто!»**

Все це варіюється від напівправди до відвертих міфів. Давайте розберемося, чому.

Чи є програми погоди кращими чи гіршими в певних районах? Загалом, ні. За даними Forecast Advisor, ваш прогноз в Далласі (~83% точний) буде приблизно таким же хорошим, як в Чикаго (~83% точний) або Нью-Йорку (~82% точний). Ви можете стверджувати, що програми погоди не розуміють горезвісних вітряних штормів Blue Norther в Техасі, і я б повернувся, сказавши, що вони так само погано справляються з озерним бризом біля озера Мічиган або морським бризом біля Атлантичного океану.

Може бути один виправданий виняток. Гори завдають шкоди моделям погоди, тому що коробки занадто великі, щоб «побачити» всі хребти, долини та вершини, які так важливі для погоди будь-якого місця. Bozeman, MT, наприклад, має лише середні ~77% точних прогнозів. Однак це падіння точності прогнозу на п'ять процентних пунктів (порівняно з місцями, згаданими вище) є досить незначним у грандіозній схемі. Навіть сумно відомі «непередбачувані» високі степи Монтани, оточені з усіх боків високими горами, все ще користуються точними прогнозами більше трьох чвертей часу.

Як щодо вашої відстані від найближчого аеропорту? Ця теорія часто наводиться як причина неточних прогнозів, оскільки спостереження за погодою зазвичай проводяться в аеропортах для підтримки безпечних польотів.

Якщо ви живете за багато кілометрів від найближчого аеропорту, як моделі погоди дізнаються, що відбувається у вашому місті? Відповідь - супутники. Переважна більшість (>80%) вхідних даних в погодні моделі надходять із супутників, що обертаються 24/7, які спостерігають за всім світом, навіть його частинами далеко від аеропортів. Спостереження за поверхнею важливі, особливо для метеорологів, які намагаються ще раз перевірити, чи добре

працюють моделі. Однак прогноз додатка базується переважно на супутникових даних, і на його точність не сильно вплине ваша близькість до аеропорту чи інших станцій спостереження за погодою.

Перегляньте це зображення, що показує всі спостереження, що надходять в модель Глобальної системи прогнозування (GFS) NOAA для одного запуску в грудні 2014 року (єдиною зміною з тих пір було більше супутникових даних - набагато більше супутникових даних):

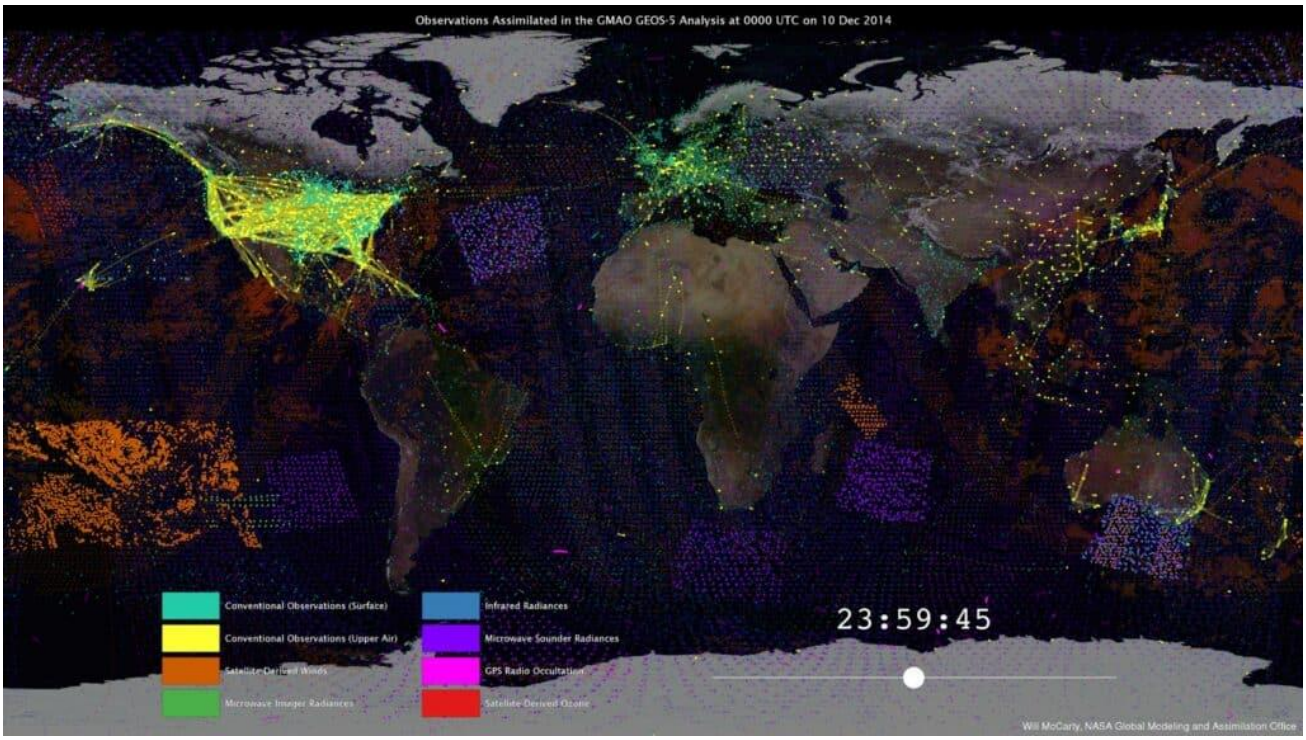
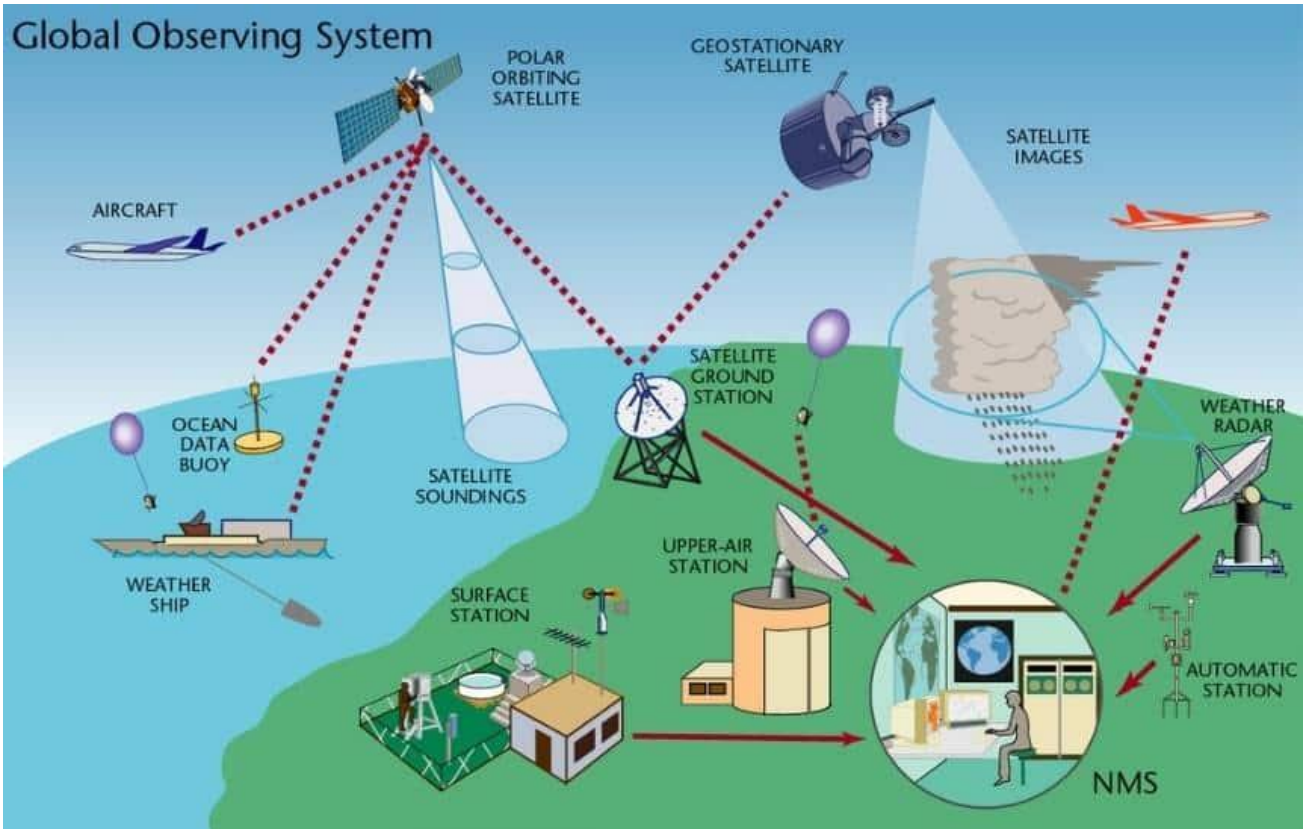


Рис. 1.3. Зображення, що показує спостереження за погодою, засвоєні однією з моделей погоди NOAA в грудні 2014 року джерелом спостережень

Всі чиркові точки є наземними станціями (як в аеропортах). Їх не так багато! Більшість точок (помаранчева, червона, синя і фіолетова) позначають дані з різних супутникових джерел, причому здорова меншість (жовта) надходить зі спостережень за літаками і метеорологічними кулями. Підсумок? Супутники (і літаки) керують днем з точки зору подачі погодних моделей (і, таким чином, додатків) інформації про поточну погоду.

Супутники також є причиною того, що частота спостереження не має значення для вашого метеорологічного додатка. Спостереження в аеропортах зазвичай робляться щогодини приблизно о 50-55 хвилині після години (це для того, щоб вони могли бути відправлені через телеграму на радіостанції ще вдень для оновленого погляду на погоду радіоведучим у верхній частині кожної години). Чи означає це, що ваш додаток точніший о 10:59, ніж о 10:49? Ні, супутники містять більшу частину вхідних даних для моделей погоди, і вони оновлюються кожні 5-15 хвилин. Самі моделі погоди повторно запускаються лише кожні 6-12 годин, тому загалом ваш прогноз змінюється лише один раз вдень і один раз рано вранці. Невеликі зміни в короткостроковому прогнозі (на сьогодні і завтра) можуть відбуватися частіше, але прогноз на 3-10 днів оновлюється лише кілька разів на день.



Глобальна система спостережень Всесвітньої метеорологічної організації (Зображення: ВМО)

Тим не менш, деякі погодні додатки відображатимуть певну форму «поточних умов», що відображають спостереження найближчого аеропорту. Якщо у вашому додатку сказано, що зараз сонячно, але насправді йде дощ, виною всьому може бути застаріле або далеке спостереження. Але вам не потрібен був додаток, щоб розповісти вам, що ви бачите зовні!

РОЗДІЛ 2. ІНФОРМАЦІЙНЕ ТА МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

2.1. Angular

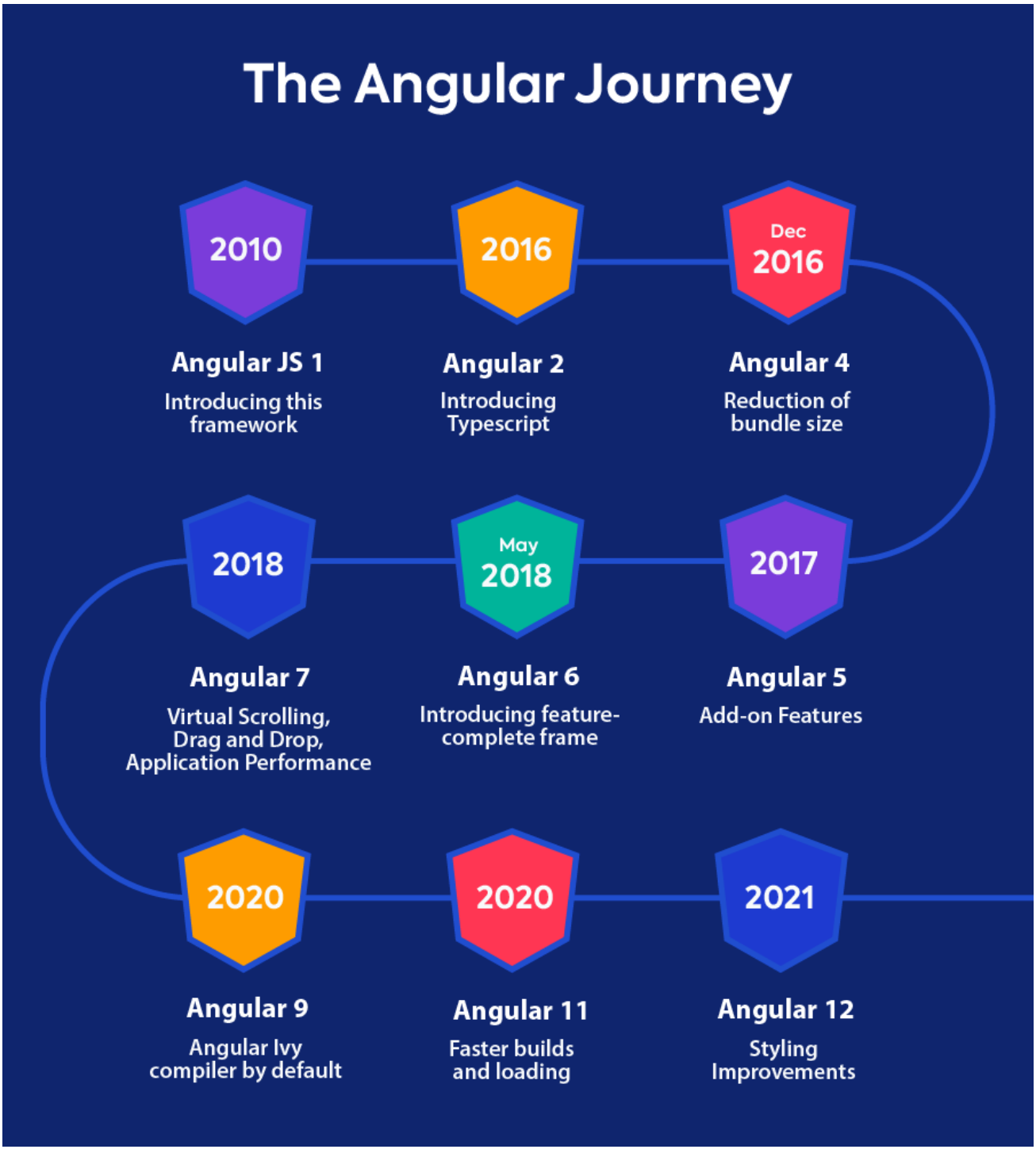


Рис. 2.1. Еволюція фрейморку

Angular - це потужний фреймворк розробки, який має багато переваг і може стати відмінним вибором для вашого проекту. Ось 8 перевірених причин, чому варто використовувати Angular:

Підтримується Google: Angular підтримується Google, що забезпечує його стабільність і подальше розвиток. Ви можете розраховувати на довгострокову підтримку від Google та навчатися у сертифікованих професіоналів Angular.

TypeScript: Angular використовує мову TypeScript, яка надає більшу безпеку та допомагає виявляти помилки на ранніх стадіях розробки. TypeScript є верхнім індексом для JavaScript і має багато корисних функцій для покращення процесу написання коду.

Декларативний UI: Angular використовує HTML для визначення інтерфейсу користувача, що робить його декларативним та легким для розуміння. Ви можете швидко визначити, що вам потрібно, і Angular позаботиться про решту.

POJO: Всі об'єкти в Angular є POJO (Plain Old JavaScript Object), що спрощує маніпуляцію ними. Ви можете легко додавати або видаляти властивості об'єктів та зациклювати їх за потреби.

PWA і SPA: Angular підтримує Progressive Web Applications (PWA), які забезпечують ефективну роботу веб-сайтів як мобільних додатків. Він також дозволяє розробляти односторінкові додатки (SPA), що покращує рейтинг SEO та забезпечує швидку загрузку сторінок.

Спрощений шаблон MVC: Angular використовує спрощений шаблон архітектури MVC, що робить розробку простішою і зменшує кількість непотрібного коду. Він також дозволяє кер, 8 messages hidden

Швидше, він лише просить розділити додаток і піклується про все інше. Отже, структури дизайну Angular і MVVM (Model-View-View-Model) досить схожі.

Angular забезпечує легку розробку, оскільки усуває потребу в непотрібному коді. Він має спрощену архітектуру MVC, що робить непотрібним написання геттерів і сеттерів. Директивами може керувати інша команда, оскільки вони не є частиною коду програми. Загалом, розробникам обіцяють менше кодування, а також легші та швидші програми.

Модулярна архітектура

В Angular код організовується за допомогою модулів, які включають компоненти, директиви, пайпи та сервіси. Ці модулі допомагають структурувати функціональні можливості програми, розділяючи її на логічні блоки, які можна використовувати повторно. Крім того, модулі підтримують відкладене завантаження, що дозволяє завантажувати функції програми у фоновому режимі або за запитом.

Angular сприяє спільній роботі різних учасників команди, забезпечуючи організований код. Якщо ви добре розумієте модулі, ви можете максимально ефективно їх використовувати. Розробники також можуть підвищити продуктивність, використовуючи відповідні модулі.

Код відповідає стандартам і легко тестується

Кожен проект потребує послідовного кодування. Нестримане кодування може призвести до затримок у запуску або збільшених витрат. Узгоджений код дозволяє полегшити використання вашого веб-сайту та використання шаблонів або попередньо визначених фрагментів коду.

Angular базується на компонентах, які дотримуються одного стилю. Кожен компонент описується у вигляді класу компонента та метаданих, що визначаються за допомогою декоратора @Component. Ці компоненти є малими елементами інтерфейсу, незалежними один від одного, і надають кілька переваг:

Повторне використання: компонентна структура Angular сприяє повторному використанню компонентів у вашому додатку, дозволяючи створювати переносні частини інтерфейсу, що спрощує розробку.

Спрощене модульне тестування: через незалежність компонентів їх легко тестувати окремо.

Покращена читабельність: послідовність в кодуванні допомагає новим розробникам швидко орієнтуватись у проекті, що підвищує продуктивність і ефективність роботи.

Легке обслуговування: окремі компоненти можна легко замінювати кращими реалізаціями, що сприяє ефективному обслуговуванню та оновленню коду.

Крім того, тестування в Angular є дуже простим. Модулі Angular містять окремі частини програми, які легко керувати. Завдяки розбиттю на модулі можна завантажувати потрібні сервіси, одночасно ефективно автоматизувати тестування. Навіть порядок завантаження модулів не потрібно запам'ятовувати, якщо дотримуватись принципу "один файл - один модуль".

РОЗДІЛ 3. ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1. Встановлення Angular та створення застосунку.

Для реалізації програми нам необхідно мати наступне:

- Node.js та npm
- Ключ API APIXU
- JSON

Відкриваємо термінал та виконуємо наступну команду, щоб встановити глобально Angular CLI на комп'ютері:

```
npm install -g @angular/cli
```

Angular CLI є командним рядковим інтерфейсом (Command Line Interface) для Angular. Він є головним інструментом для створення нового проекту Angular та створення різних компонентів, які складають проект Angular. Встановлення його глобально здійснюється за допомогою аргументу -g.

Після успішної установки ми спостерігаємо такий результат:

```
...
+ @angular/cli@7.2.2
...
```

Для створення додатку ми використовуємо команду "ng" в терміналі.

Для виконання будь-якої дії з Angular з терміналу команда "ng" є обов'язковою передумовою. Для створення нового додатку використовується команда "ng new". Команда "ng new" створює новий додаток Angular, автоматично імпортує необхідні бібліотеки та генерує весь необхідний код, що складає основу нашого додатку.

Починаємо зі створення нового додатку, він буде називатися "weather-app":

```
ng new weather-app
```

Після виконання команди "ng new" нам буде запропоновано надати додаткову інформацію про функціональні можливості, які ми бажаємо включити до нашого додатку.

```
Output
Would you like to add Angular routing? (y/N)
```

Механізм маршрутизації в Angular дозволяє нам створювати односторінкові додатки з різними видами шляхів, використовуючи маршрути та компоненти. Вводимо "y", щоб прийняти значення за замовчуванням.

```
Output
Which stylesheet format would you like to use? (Use arrow keys)
```

Для прийняття значення за замовчуванням для CSS натискаємо клавішу ENTER. Переходио в папку "weather-app".

При розгляді структури нашого каталогу ми відзначимо наявність кількох різних папок та файлів:

- Файл **package.json** розташований у кореневій папці "weather-app" і виконує функції, подібні до будь-якого іншого додатку Node.js. Він містить всі необхідні бібліотеки, що використовуються в нашому додатку, назву додатку, команди для тестування та інші налаштування. Основними даними, які містяться у цьому файлі, є деталі про залежності зовнішніх бібліотек, необхідних для належної роботи нашого додатку.
- Файл **app.module.ts** розташовується у папці "app" всередині директорії "src" у проекті "weather-app". Цей файл вказує Angular, як зібрати додаток, і містить деталі про компоненти, модулі та провайдери, що використовуються в додатку. У масиві imports вже імпортований модуль BrowserModule. Модуль BrowserModule надає основні сервіси та

директиви для додатку і завжди має бути першим імпортованим модулем у масиві imports.

- Файл **angular.json** розташований у кореневій папці "weather-app". Це файл конфігурації для Angular CLI, який містить внутрішні налаштування, необхідні для запуску Angular-додатку. В цьому файлі задаються значення за замовчуванням для всього додатку, а також визначаються опції, такі як використання конфігураційних файлів під час тестування, використання глобальних стилів у додатку та визначення папки для виведення зібраних файлів.

3.2. Встановлення Bootstrap

Для належної роботи Bootstrap потрібно встановити дві залежності: jQuery і popper.js. jQuery є JavaScript-бібліотекою, спрямованою на клієнтський скриптинг, а popper.js - бібліотекою позиціонування, яка основною метою керує підказками та виринаючими вікнами.

В терміналі ми навігуємо до кореневої папки "weather-app":

```
cd weather-app
```

Після цього виконуємо наступну команду для встановлення всіх залежностей та збереження посилань на них у файлі package.json:

```
npm install --save jquery popper.js bootstrap
```

Опція --save автоматично імпортує посилання у файл package.json, тому не потрібно вручну додавати їх після встановлення.

Крім того, ми повинні включити ці бібліотеки до додатку.

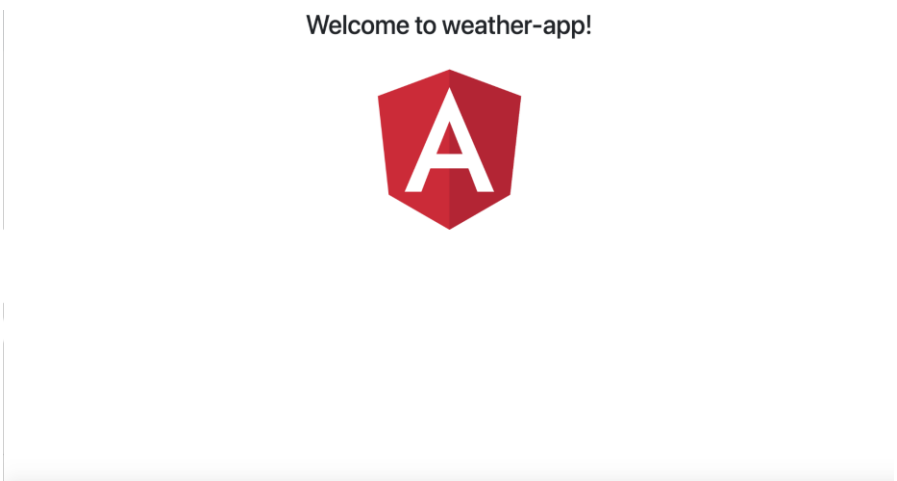
Для **popper.js**, файл, який ми повинні включити, розташований за шляхом *node_modules/popper.js/dist/umd/popper.js*. Щодо **jQuery**, нам необхідний файл

node_modules/jquery/dist/jquery.slim.js. Нарешті, для **Bootstrap** нам потрібно включити два файли - як JavaScript, так і CSS. Вони знаходяться відповідно за шляхами *node_modules/bootstrap/dist/js/bootstrap.js* і *node_modules/bootstrap/dist/css/bootstrap.css*.

Тепер, коли у нас є всі необхідні шляхи до файлів, відкриваємо файл angular.json. Масив styles використовується для додавання посилань на CSS-файли, а масив scripts - для посилань на всі скрипти.

```
"options:"                                                    {
...
"styles":                                                     [
   "node_modules/bootstrap/dist/css/bootstrap.css",
    "src/styles.css"
],
"scripts":                                                    [
   "node_modules/jquery/dist/jquery.slim.js",
   "node_modules/popper.js/dist/umd/popper.js",
   "node_modules/bootstrap/dist/js/bootstrap.js"
]},
```

Тепер ми успішно імпортували основні .js- та .css-файли, необхідні для коректної роботи Bootstrap. Ми вказали відносні шляхи до цих файлів у файлі angular.json, додавши .css-файли до масиву styles та .js-файли до масиву scripts.



3.3. Створення та доступ до компоненту "Weather"

Створюємо компонент "Weather" і автоматично імпортуємо його до файлу app.module.ts.

```
ng generate component weather
```

Ми побачимо вивід подібний до такого:

```
CREATE src/app/weather/weather.component.css (0 bytes)
CREATE src/app/weather/weather.component.html (21 bytes)
CREATE src/app/weather/weather.component.spec.ts (614 bytes)
CREATE src/app/weather/weather.component.ts (265 bytes)
UPDATE src/app/app.module.ts (405 bytes)
```

Цей результат демонструє, що Angular автоматично створив чотири необхідних файли для компонента:

- **weather.component.css** - файл стилів компонента
- **weather.component.html** - файл шаблону компонента
- **weather.component.spec.ts** - файл юніт-тестів для компонента
- **weather.component.ts** - файл з логікою компонента

Крім того, файл **app.module.ts** було оновлено для імпорту компонента "Weather".

Створюємо файл routes.ts і зберігаємо його у директорії src/app.

```
import { Routes } from '@angular/router'
```

Визначаємо шлях URL і компонент у файлі src/app/routes.ts. Наш додаток буде налаштований таким чином, що при відкритті домашньої сторінки (http://localhost:4200), ми будемо мати доступ до новоствореного компонента

```
import { Routes } from '@angular/router';
```

```
import { WeatherComponent } from './weather/weather.component';

export const routes: Routes = [
  { path: '', component: WeatherComponent }
];
```

Ми імпортували компонент WeatherComponent і створили змінну allAppRoutes, яка представляє масив типу Routes. У масиві allAppRoutes ми визначили об'єкти маршрутів, кожен з яких має URL-шлях та компонент, пов'язаний з ним. Зазначили, що при переході на кореневий URL (''), маршрутизатор повинен перенаправляти до компонента WeatherComponent.

Додаємо ці маршрути до головного файлу app.module.ts. Для цього передаємо створений масив allAppRoutes в модуль з назвою RouterModule. RouterModule ініціалізує та налаштовує Router, який відповідає за навігацію в програмі, і передає йому дані маршрутизації з allAppRoutes:

```
import {WeatherComponent} from './weather/weather.component';
import {RouterModule} from '@angular/router';
import {allAppRoutes} from './routes';
...
@NgModule({
   declarations:[
     ...
   ],
   imports: [
      BrowserModule,
      RouterModule.forRoot(allAppRoutes)
   ]
   ...
```

```
})
```

У цьому файлі ми імпортували RouterModule та масив allAppRoutes, що містить об'єкти маршрутів. Потім передали масив allAppRoutes в RouterModule, щоб Router знав, куди маршрутизувати URL-адреси.

Останнім кроком є ввімкнення маршрутизації. Для цього відкриваємо файл app.component.ts. У цьому файлі є властивість templateUrl, яка вказує на шлях до HTML-коду для цього компонента: ./app.component.html. Ми відкриваємо цей файл, src/app/app.component.html, і бачимо, що він містить весь HTML-код для нашоє сторінки localhost:4200.

Видаляємо увесь HTML-код, що міститься в файлі app.component.html, і замінюємо його на наступний код:

```
<router-outlet></router-outlet>
```

3.4. Визначення інтерфейсу програми та налаштування форми

Ми оберали Bootstrap як основу для відображення інтерфейсу нашого додатка.

Ми розділимо сторінку на дві секції, розмістивши їх у формі шести колонок. У лівій колонці буде розташована форма пошуку, а в правій колонці будуть відображені деталі погоди. Для здійснення цього, відкриваємо файл src/app/weather/weather.component.html, щоб отримати доступ до HTML-коду компонента WeatherComponent.

```
<div class="container">
  <div class="row">
    <div class="col-md-6"><h3 class="text-center">Пошук погоди:</h3></div>
```

```
   <div class="col-md-6"><h3 class="text-center">Деталі погоди:</h3></div>
  </div>
</div>
```

Створення розмітки HTML для веб-сторінки з двома колонками

Після цього кроку у створенні форми ми перейдемо до роботи з першою колонкою col-md-6. Тут ми додамо кнопку, яка буде відправляти введені дані з форми до APIXU та отримувати запитані деталі погоди відповідно.

```
<div class="col-md-6">
  <h3 class="text-center">Пошук погоди:</h3>
  <form>
    <div class="form-group">
      <input
        class="form-control"
        type="text"
        id="weatherLocation"
        aria-describedby="weatherLocation"
        placeholder="Введіть вашу локацію"
      />
    </div>
    <div class="text-center">
      <button type="submit" class="btn btn-success btn-md">
        Шукати погоду</button>
    </div>
  </form>
</div>
```

Рис. 3.1. Створення форми для пошуку погоди

Рис.3.2. Сторінка погодного додатку у браузері матиме такий вигляд

Надамо сторінці додатковий CSS для покращення її вигляду та створення більшого простору.

У файлі weather.component.html додаємо клас .my-4 з CSS-стилями Bootstrap до кожного тегу <h3>. Клас "my" встановлює зовнішній відступ для елемента, при цьому "m" визначає зовнішній відступ як зверху, так і знизу елемента, а число 4 визначає величину цього зовнішнього відступу.

```
<div class="col-md-6">
  <h3 class="text-center my-4">Пошук погоди:</h3>
  <form>
    <div class="form-group">
      <input
        class="form-control"
        type="text"
        id="weatherLocation"
        aria-describedby="weatherLocation"
        placeholder="Введіть вашу локацію"
      />
    </div>
```

```
    <div class="text-center">
      <button type="submit" class="btn btn-success btn-md">
        Шукати погоду
      </button>
    </div>
  </form>
</div>
<div class="col-md-6">
  <h3 class="text-center my-4">Деталі погоди:</h3>
</div>
```

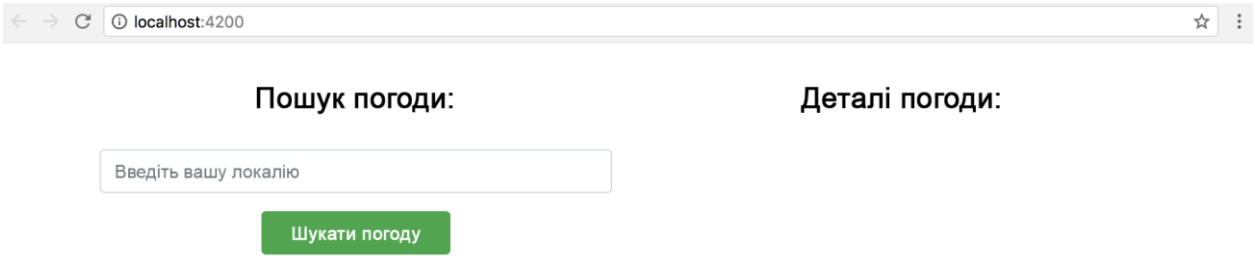


Рис.3.3. Сторінка погодного додатку у браузері матиме такий вигляд

Була створена форма, а також секція, де буде відображатися інформація, отримана з використанням APIXU API. Наступним кроком буде налаштування форми для правильного вводу місцезнаходження.

Щоб використовувати реактивні форми, потрібно відкрити файл app.module.ts. Далі, оголошуємо імпорт ReactiveFormsModule, розташувавши його на початку файлу.

```
import { ReactiveFormsModule } from '@angular/forms';
```

```
@NgModule({
   ...
})
```

На завершення, включимо ReactiveFormsModule до списку імпортів.

```
@NgModule({
   ...
   imports: [
      BrowserModule,
      RouterModule.forRoot(allAppRoutes),
      ReactiveFormsModule
   ]
   ...
})
```

Відкриваємо файл weather.component.ts та додаємо імпорт класів FormBuilder і FormGroup.

```
import { Component, OnInit } from '@angular/core';
import { FormBuilder, FormGroup } from '@angular/forms';
```

Створимо змінну у файлі weather.component.ts, яка буде посилатися на FormGroup:

```
export class WeatherComponent implements OnInit {
  public weatherSearchForm: FormGroup;
  constructor() { }
```

Додамо імпорт FormBuilder у конструктор, щоб мати доступ до нього у компоненті:

```
public weatherSearchForm: FormGroup;
constructor(private formBuilder: FormBuilder) {}
```

Додавши formBuilder до конструктора, створюється екземпляр класу FormBuilder, що дозволяє використовувати його всередині компонента.

Перед завершенням процесу зв'язування з HTML, необхідно створити форму. У методі ngOnInit компонента WeatherComponent ми ініціалізуємо форму:

```
constructor(private formBuilder: FormBuilder) {}
ngOnInit() {
   this.weatherSearchForm = this.formBuilder.group({
     location: ['']
   });
  }
```

У файлі weather.component.ts було визначено компоненти форми відповідно до стилю реактивних форм. Була створена група композитних елементів форми, де наразі присутній лише один елемент - location. Масив [''] використовується для вказання додаткових параметрів для вхідних елементів форми, таких як попереднє заповнення значеннями та використання валідаторів для перевірки введених даних.

Відкриваємо файл weather.component.html і додаємо атрибут [formGroup] до форми. Цей атрибут буде відповідати змінній, яка була оголошена у файлі weather.component.ts - weatherSearchForm. Далі, прив'язуємо елемент location (також оголошений у файлі weather.component.ts) до HTML.

```
<form
  [formGroup]="weatherSearchForm" >
  <div class="form-group">
```

```
    <input
      class="form-control"
      type="text"
      id="weatherLocation"
      aria-describedby="weatherLocation"
      placeholder="Введіть вашу локацію"
    />formControlName="location" />
  </div>
  <div class="text-center">
    <button type="submit" class="btn btn-success btn-md">
      Пошук погоди
    </button>
  </div>
</form>
```

3.5. З’єднання кнопки та Виклик APIXU API

APIXU API отримує дані про розташування, шукає деталі поточної погоди для цього розташування і повертає їх клієнту. Для виконання HTTP-запитів в Angular потрібно імпортувати модуль

```
HttpClientModule:
import { ReactiveFormsModule } from '@angular/forms';
import { HttpClientModule } from '@angular/common/http';
@NgModule({
  ...
  imports: [
    BrowserModule,
```

```
    RouterModule.forRoot(allAppRoutes),
    ReactiveFormsModule,
    HttpClientModule
  ]
  ...
})
```

Імпорт двох модулів ReactiveFormsModule та HttpClientModule та реєстрація їх в NgModule

Наступним кроком буде здійснення HTTP-виклику до APIXU API, для цього запускаємо команду в терміналі:

```
ng g service apixu
```

Після виконання цієї команди було створено файл сервісу (**apixu.service.ts**) та файл тестів (**apixu.service.spec.ts**).

Для використання цього сервісу в додатку потрібно додати його як провайдера до файлу **app.module.ts**.

```
import { HttpClientModule } "@angular/common/http";
import { ApixuService } from "./apixu.service";
```

Також потрібно додати імпортований ApixuService як провайдера у блок провайдерів:

```
@NgModule({
    ...
    providers: [ApixuService],
    ...
})
```

Відкриваємо файл src/app/apixu.service.ts. У цьому файлі бачимо початковий код для створення сервісу. Починається він з імпорту інтерфейсу

Injectable з Angular, потім вказується, що сервіс повинен бути доступний через кореневий інжектор (для всього додатку), і на кінці використовується декоратор @Injectable для позначення сервісу.

import { Injectable } from '@angular/core';

```
@Injectable({
  providedIn: 'root'
})
export class ApixuService {
  constructor() { }
}
```

Додатково, у файлі apixu.service.ts потрібно імпортувати HttpClient з бібліотеки Angular для здійснення HTTP-запитів до APIXU API безпосередньо з цього файлу. Відкриваємо apixu.service.ts і пишемо:

```
import { HttpClient } from '@angular/common/http';
```

Тепер ми переходимо до написання методу getWeather(), який має один параметр - location. Цей метод виконує запит до APIXU API і повертає дані про погоду для вказаного місцезнаходження.

Для виконання цієї операції потрібно мати API-ключ, який було отримано при реєстрації в APIXU API.

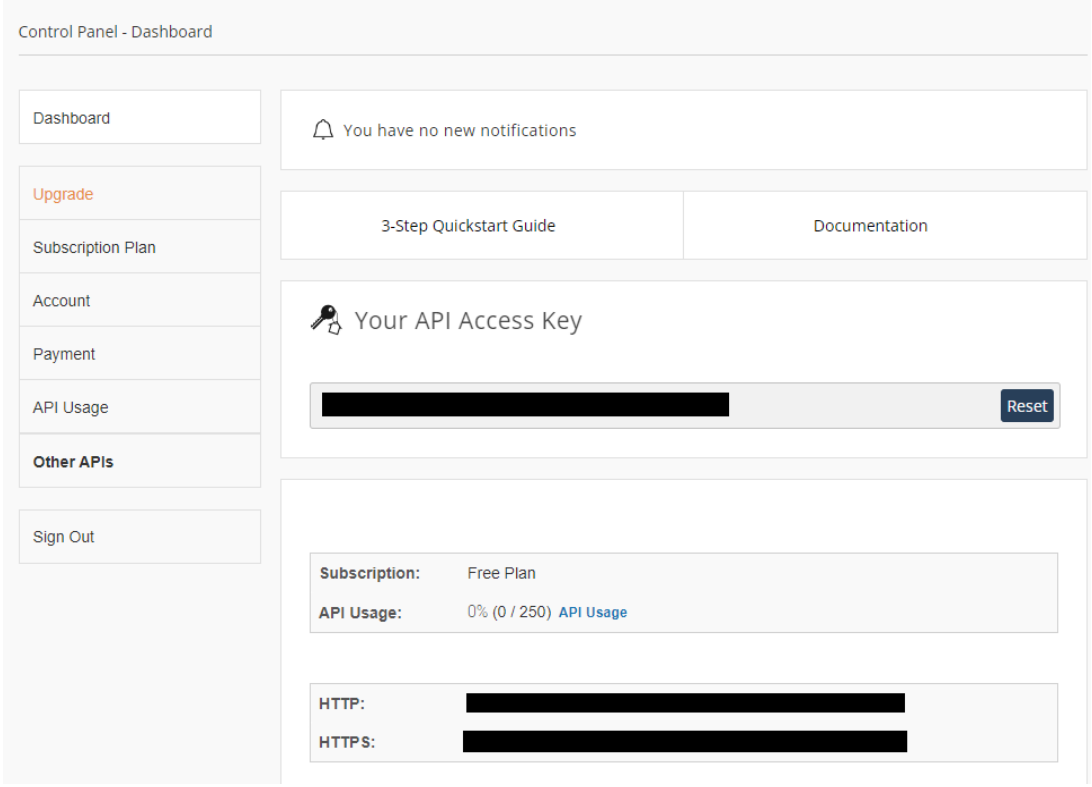


Рис.3.4. API-ключ

Бачимо API-ключ та нижче посилання на API URL з використанням ключа. Це посилання вже заповнене для отримання даних про поточну погоду та прогнозу. Копіюємо HTTPS-посилання для отримання деталей про поточну погоду, ось такого типу:

https://api.apixu.com/v1/current.json?key=PASTE_API_KEY&q=Lviv

Місцезнаходження буде передаватись з форми у параметрі **&q=**.

```
export class ApixuService {
  constructor(private http: HttpClient) {}
  getWeather(location){
     return this.http.get(
        'https://api.apixu.com/v1/current.json?key=YOUR_API_KEY&q=' +
location
     );
```

```
 }}
```

Ми додали залежність HttpClient до конструктора, щоб мати можливість використовувати його. Також ми створили метод getWeather(), який отримує параметр location і виконує GET-запит за вказаним URL. В даному випадку, ми залишаємо параметр &q= порожнім, оскільки місцезнаходження буде передаватись безпосередньо з параметру location у методі.

Тепер погодний сервіс готовий. Також потрібно імпортувати службу в WeatherComponent, внести її в конструктор для використання та оновити метод sendToAPIXU(), щоб передати місцезнаходження в новостворену службу. Відкриваємо файл weather.component.ts, та допрацьовуємо його:

```
import { FormBuilder, FormGroup } from "@angular/forms";
import { ApixuService } from "../apixu.service";
...
constructor(
   private formBuilder: FormBuilder,
   private apixuService: ApixuService
 ) {}
...
ngOnInit(){...}
sendToAPIXU(formValues){
   this.apixuService
     .getWeather(formValues.location)
     .subscribe(data => console.log(data));
}
```

Ми видалили попередню конструкцію console.log з методу sendToAPIXU() і оновили його вміст. Тепер дані місцезнаходження передаються з форми у

метод sendToAPIXU(), який був створений раніше. Потім ці дані передаються в метод getWeather() класу ApixuService, який виконує HTTP-запит до API з використанням цього місцезнаходження. Після цього ми підписуємось на отриману відповідь і записуємо ці дані в консоль. При роботі з HTTP-запитами завжди необхідно викликати метод subscribe, оскільки запит не почнеться, доки ми не отримаємо відповідь Observable. Ми не зможемо отримати дані з Observable, поки підписник не підпишеся на нього, оскільки він не виконається до цього моменту.

Відкриваємо консоль в браузері. Вводимо "Lviv, Ukraine" і натискаємо кнопку "Шукати погоду". Якщо натиснемо на стрілки вкладок, побачимо список деталей погоди в консолі.

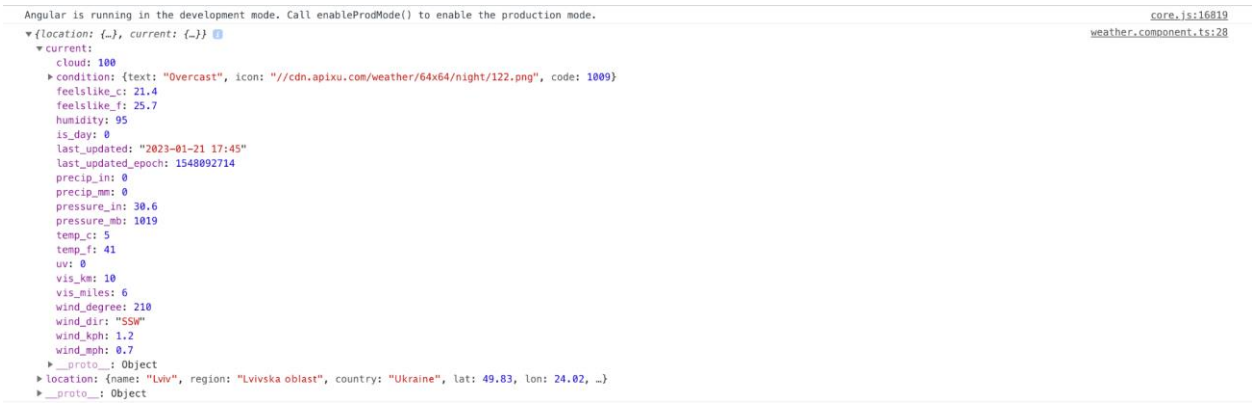


Рис.3.5. JSON-об'єкти

Вивід демонструє JSON-об'єкти, що містять повну інформацію про погоду. Отримано два об'єкти: "поточний" (current) і "місцезнаходження" (location). Перший об'єкт містить необхідні деталі про погоду, а другий - інформацію про місцезнаходження.

3.6. Відображення погодних даних в додатку

Спочатку ми створюємо змінну для зберігання отриманих погодних даних. Потім ми використовуємо інтерполяцію в HTML коді для відображення цих даних.

У файлі weather.component.ts ініціалізуємо змінну weatherData, в яку зберігаємо отримані JSON-дані з API, та змінюємо код, який раніше був у виразі .subscribe():

```
export class WeatherComponent implements OnInit {
public weatherSearchForm: FormGroup;
public weatherData: any;
...
sendToAPIXU(formValues){
   this.apixuService
   .getWeather(formValues.location)
   .subscribe(data => this.weatherData = data)
     console.log(this.weatherData);
   }
}
```

У файлі weather.component.html ми розміщуємо підзаголовки для даних, які будуть відображені.

```
<div class="col-md-6">
  <h3 class="text-center my-4">Деталі погоди:</h3>
  <p class="text-center">Поточні погодні умови:</p>
  <p class="text-center">Температура за Цельсієм:</p>
  <p class="text-center">Температура за Фаренгейтом:</p>
  <p class="text-center">Відчувається як (за Цельсієм:)</p>
```

```
  <p class="text-center">Відчувається як (за Фаренгейтом:)</p>
  <p class="text-center">Місце пошуку:</p>
</div>
```

Додаємо дані, які отримали з JSON-об'єкта, до HTML коду:

```
<h3 class="text-center my-4 "Деталі погоди:</h3>
<p class="text-center">
  Поточні погодні умови: {{this.weatherData?.current.condition.text}}
</p>
<p class="text-center">
  Температура за Цельсієм: {{this.weatherData?.current.temp_c}}
</p>
<p class="text-center">
  Температура за Фаренгейтом: {{this.weatherData?.current.temp_f}}
</p>
<p class="text-center">
  Відчувається як (за Цельсієм:) {{this.weatherData?.current.feelslike_c}}
</p>
<p class="text-center">
  Відчувається як (за Фаренгейтом:)
  {{this.weatherData?.current.feelslike_f}}
</p>
<p class="text-center">
  Місце пошуку: {{this.weatherData?.location.name}},
  {{this.weatherData?.location.country}}
</p>
```

У HTML коді використовувався оператор Ельвіса ("?") для отримання даних зі змінної weatherData.

При виконанні HTTP-запиту в асинхронному режимі отримуємо дані з певною затримкою, оскільки це не миттєва відповідь. Однак Angular продовжить заповнювати HTML даними, які були вказані у змінній weatherData. Якщо до моменту, коли Angular починає заповнювати абзаци, дані ще не були отримані, виникне помилка, оскільки Angular не зможе знайти ці дані. Наприклад, .current або .location будуть відображатися як undefined.

Оператор Ельвіса є безпечним навігатором, який запобігає такій проблемі. Він каже Angular, щоб спочатку перевіряв, чи змінна weatherData має значення, перш ніж продовжувати відображати дані у HTML. Коли всі дані будуть доступні у змінній weatherData, Angular оновить прив'язки та відображатиме дані як зазвичай.

Ми використовували шаблон поверненого об'єкта погоди у форматі JSON для отримання потрібних даних. Коли ми вводимо "Lviv, Ukraine" у браузері, ми спостерігаємо, як погодні дані з'являються на правій стороні.

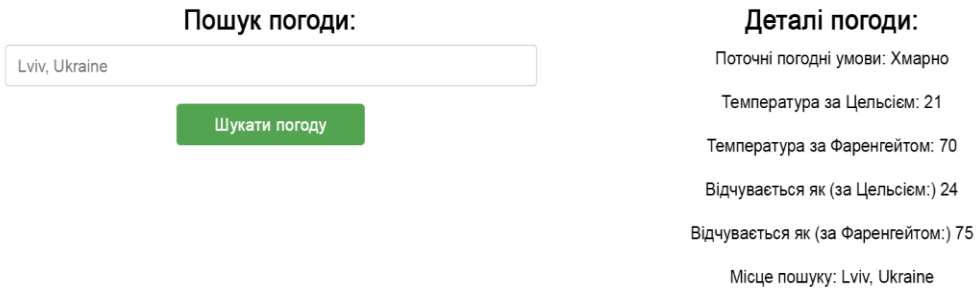


Рис.3.6. Результат

ВИСНОВКИ

Розроблено програмне забезпечення для відображення погоди, використовуючи Angular, Bootstrap та APIXU API. Створино функціонал, який після визначення місцезнаходження у формі пошуку, застосунок буде відображати поточні погодні дані для введеного місця. Для реалізації використано версію Angular 7.2.0 та версію Bootstrap 4.2.1. Використовуючи APIXU, користувач може отримати оновлену інформацію про погоду, а також прогнози на майбутні дні для будь-якого місця на планеті.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Adam Freeman: Pro Angular 2nd ed. Edition— NY, February 3, 2017.
2. Aristeidis Bampakos: Svelte 3 Up and Running: Learning Angular: A no-nonsense beginner's guide to building web applications with Angular 10 and TypeScript, 3rd Edition– Packt Publishing Ltd, September 7, 2020
3. Brad Dayley, Brendan Dayley, Caleb Dayley: Learning Angular, 2nd Edition – Addison-Wesley Professional, October 2017.
4. Freeman A. Pro Angular: Build Powerful and Dynamic Web Apps. – Apress, 2022, 820 ст.
5. Banks A., Porcello E. Learning React: Modern Patterns for Developing React Apps. – 2nd ed. – O'Reilly Media, 2020, 310 ст.
6. Meier J. D. Angular Projects. – 2nd ed. – Packt Publishing, 2020, 426 ст.
7. Flanagan D. JavaScript: The Definitive Guide. – 7th ed. – O'Reilly Media, 2020, 704 ст.
8. Osmani A. Learning JavaScript Design Patterns. – O'Reilly Media, 2020, 254 ст.
9. Nixon R. Learning PHP, MySQL & JavaScript: With jQuery, CSS & HTML5. – 6th ed. – O'Reilly Media, 2021, 832 ст.
10. Steyer R. Building Web Applications with Vue.js. – Springer Vieweg, 2022, 350 ст.
11. Hempel B., Lubin J., Chugh R. Sketch-n-Sketch: Output-Directed Programming for SVG, 2019.
12. Bylinskii Z., Herman L., Hertzmann A. Towards Better User Studies in Computer Graphics and Vision, 2022.
13. Bae S. JavaScript Data Structures and Algorithms. – Apress, 2019.
14. Russell S., Norvig P. Artificial Intelligence: A Modern Approach. – 4th ed. – Pearson, 2020, 1136 ст.
15. Abelson H., Sussman G., Sussman J. Structure and Interpretation of Computer Programs (JavaScript Edition). – MIT Press, 2022, 657 ст.

ДОДАТКИ

```
import { Component, OnInit } from "@angular/core";
import { FormBuilder, FormGroup } from "@angular/forms";
import { ApixuService } from "../apixu.service";

@Component({
  selector: "app-weather",
  templateUrl: "./weather.component.html",
  styleUrls: ["./weather.component.css"]
})
export class WeatherComponent implements OnInit {
  public weatherSearchForm: FormGroup;
  public weatherData: any;

  constructor(
    private formBuilder: FormBuilder,
    private apixuService: ApixuService
  ) {}

  ngOnInit() {
    this.weatherSearchForm = this.formBuilder.group({
      location: [""]
    });
  }

  sendToAPIXU(formValues) {
    this.apixuService.getWeather(formValues.location).subscribe(data => {
```

```
    this.weatherData = data;
    console.log(this.weatherData);
   });
  }
}
```

```
<div class="container">
 <div class="row">
  <div class="col-md-6">
   <h3 class="text-center my-4">Search for Weather:</h3>
   <form
    [formGroup]="weatherSearchForm"
    (ngSubmit)="sendToAPIXU(weatherSearchForm.value)"
   >
    <div class="form-group">
     <input
      class="form-control"
      type="text"
      id="weatherLocation"
      aria-describedby="weatherLocation"
      placeholder="Please input a Location"
      formControlName="location"
     />
    </div>
    <div class="text-center">
```

```
      <button type="submit" class="btn btn-success btn-md">
        Search for the weather
      </button>
    </div>
  </form>
</div>
<div class="col-md-6">
  <h3 class="text-center my-4">Weather Details:</h3>
  <p class="text-center">
    Current weather conditions: {{ this.weatherData?.current.condition.text
    }}.
  </p>
  <p class="text-center">
    Temperature in Degrees Celsius: {{ this.weatherData?.current.temp_c }}
  </p>
  <p class="text-center">
    Temperature in Degrees Farenheit: {{ this.weatherData?.current.temp_f }}
  </p>
  <p class="text-center">
    Feels like in Degrees Celsius: {{ this.weatherData?.current.feelslike_c
    }}
  </p>
  <p class="text-center">
    Feels like in Degrees Farenheit: {{
    this.weatherData?.current.feelslike_f }}
  </p>
  <p class="text-center">
    Location Searched: {{ this.weatherData?.location.name }}, {{
```

```
        this.weatherData?.location.country }}.
      </p>
    </div>
  </div>
</div>
```

```
import { TestBed, async } from '@angular/core/testing';
import { AppComponent } from './app.component';

describe('AppComponent', () => {
  beforeEach(async(() => {
    TestBed.configureTestingModule({
      declarations: [
        AppComponent
      ],
    }).compileComponents();
  }));

  it('should create the app', () => {
    const fixture = TestBed.createComponent(AppComponent);
    const app = fixture.debugElement.componentInstance;
    expect(app).toBeTruthy();
  });

  it(`should have as title 'weather-app'`, () => {
    const fixture = TestBed.createComponent(AppComponent);
    const app = fixture.debugElement.componentInstance;
```

```
    expect(app.title).toEqual('weather-app');
  });

  it('should render title in a h1 tag', () => {
    const fixture = TestBed.createComponent(AppComponent);
    fixture.detectChanges();
    const compiled = fixture.debugElement.nativeElement;
    expect(compiled.querySelector('h1').textContent).toContain('Welcome to
weather-app!');
  });
});
```