

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. Сікорського

Кафедра
інформатики та програмної інженерії
(повна назва кафедри, циклової комісії)

КУРСОВА РОБОТА

з дисципліни Компоненти програмної інженерії
(назва дисципліни)

на тему: веб-сервіс прогнозування погоди, шляхом агрегації даних онлайн
гідромер

Студента 3 курсу, групи ІІІ-13
Карам'яна Варта́на Су́реновича

Спеціальності 121 «Інженерія програмного забезпечення»

Керівник асистент Ахаладзе І.Е.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Кількість балів: _____
Національна оцінка _____

Члени комісії

_____	к.т.н., доцент Лісовиченко О. І.
(підпис)	(вчене звання, науковий ступінь, прізвище та ініціали)
_____	асистент Ахаладзе А. Е.
(підпис)	(вчене звання, науковий ступінь, прізвище та ініціали)

Київ- 2024 рік

**Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”**

Факультет (інститут) Інформатики та обчислювальної техніки
(повна назва)

Кафедра інформатики та програмної інженерії
(повна назва)

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – *121 Інженерія програмного забезпечення*

Освітньо-професійна програма – *Програмне забезпечення інформаційних
управляючих систем та технологій*

ЗАТВЕРДЖУЮ

Ілля АХАЛАДЗЕ
(підпис)

“ ” _____ 2024 р.

**ЗАВДАННЯ
НА КУРСОВУ РОБОТУ СТУДЕНТУ**

Карам'яна Варта́на Су́реновича
(прізвище, ім'я, по батькові)

1. Тема роботи *«Веб-сервіс прогнозування погоди, шляхом агрегації
даних онлайн гідрометцентрів»*

керівник роботи Ахаладзе Ілля Елдарійович
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

2. Термін подання студентом роботи *«30» грудня 2023 року*

3. Вихідні дані до роботи

Технічне завдання

4. Зміст пояснювальної записки

*1) Аналіз вимог до програмного забезпечення: основні визначення та терміни,
опис предметного середовища, огляд існуючих технічних рішень та відомих
програмних продуктів, розробка функціональних та нефункціональних вимог*

*2) Моделювання та конструювання програмного забезпечення: опис BPNM-діаграми,
розробка діаграми класів, опис програмного забезпечення, засобів розробки,
технічні рішення, архітектури програмного забезпечення*

3) Аналіз якості та тестування програмного забезпечення: метрики оцінки якості, опис процесів тестування функціональних і нефункціональних вимог, опис контрольного прикладу

4) Впровадження та супровід програмного забезпечення: опис процесу білдингу застосунку, опис підтримки оновлень програмного забезпечення

5. Перелік графічного матеріалу

1) Модель вимог у загальному вигляді

2) Схема структурна варіантів використання

3) Схема структурна послідовностей

4) Діаграма контексту та контейнерів

5) Схема бази даних

6) Схема структурна класів програмного забезпечення

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання «12» жовтня 2023 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання курсової роботи	Термін виконання етапів роботи	Примітка
1.	Вивчення рекомендованої літератури	до 10.10.2023	
2.	Аналіз існуючих методів розв'язання задачі	10.10.2023 - 12.10.2023	
3.	Постановка та формалізація задачі	13.10.2023 - 15.10.2023	
4.	Аналіз вимог до програмного забезпечення	16.10.2023- 20.10.2023	
5.	Алгоритмізація задачі	21.10.2023- 25.10.2023	
6.	Моделювання програмного забезпечення	26.10.2023- 31.10.2023	
7.	Обґрунтування використовуваних технічних засобів	01.11.2023- 05.11.2023	
8.	Розробка архітектури програмного забезпечення	06.11.2023- 13.11.2023	
9.	Розробка програмного забезпечення	14.11.2023- 07.12.2023	
10.	Налагодження програми	08.12.2023- 14.12.2023	
11.	Виконання графічних документів	15.12.2023-	

		17.12.2023	
12.	Оформлення пояснювальної записки	18.12.2023- 30.12.2023	
13.	Подання КР на перевірку	30.12.2023	
14.	Захист КР	06.01.2024	

Студент _____ Вартан КАРАМЯН
(підпис)

Керівник _____ Ілля АХАЛАДЗЕ
(підпис)

АНОТАЦІЯ

Структура та обсяг роботи. Пояснювальна записка курсової роботи складається з 4 розділів, містить 39 рисунків, 55 таблиць, 3 додатки, 15 джерел.

Мета. Метою розробки є підвищення точності отриманої інформації про поточну та майбутню погоду для полегшення прийняття рішень у бізнесі чи повсякденному житті.

У розділі аналізу вимог були поставлені вимоги для програмного забезпечення.

У розділі моделювання програмного забезпечення було описано архітектуру програмного забезпечення та алгоритми вирішення прикладних задач.

У розділі аналіз якості були описані основні тест кейси, та стани системи після проведення тестування.

У розділі впровадження та супровід було описано процеси автоматизованого розгортання програмного забезпечення на виділеному сервері.

КЛЮЧОВІ СЛОВА: ВЕБ-СЕРВІС, API, ГІДРОМЕТЦЕНТР, ДАНІ, ВЕБ-ЗАСТОСУНОК, ЗАПИТ, HTTP, JSON

Пояснювальна записка до курсової роботи

на тему: веб-сервіс прогнозування погоди, шляхом агрегації даних онлайн
гідрометцетрів

КП.ІП-1314.045440.02.81

Київ – 2024

ЗМІСТ

<u>ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ</u>	10
<u>ВСТУП</u>	11
<u>1 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ</u>	12
<u>1.1 Загальні положення</u>	12
<u>1.2 Змістовний опис і аналіз предметної області</u>	14
<u>1.3 Аналіз існуючих технологій та успішних ІТ-проектів</u>	15
<u>1.3.1 Аналіз відомих алгоритмічних та технічних рішень</u>	16
<u>1.3.2 Аналіз допоміжних програмних засобів та засобів розробки</u>	16
<u>1.3.3 Аналіз відомих програмних продуктів</u>	18
<u>1.4 Аналіз вимог до програмного забезпечення</u>	21
<u>1.4.1 Розроблення функціональних вимог</u>	26
<u>1.4.2 Розроблення нефункціональних вимог</u>	30
<u>1.5 Постановка задачі</u>	31
<u>Висновки до розділу</u>	32
<u>2 МОДЕЛЮВАННЯ ТА КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ</u>	34
<u>2.1 Моделювання та аналіз програмного забезпечення</u>	34
<u>2.2 Архітектура програмного забезпечення</u>	36
<u>2.3 Конструювання програмного забезпечення</u>	38
<u>2.4 Аналіз безпеки даних</u>	42
<u>Висновки до розділу</u>	43
<u>3 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ</u>	44
<u>3.1 Аналіз якості ПЗ</u>	44
<u>3.2 Опис процесів тестування</u>	45
<u>3.3 Опис контрольного прикладу</u>	49
<u>Висновки до розділу</u>	56
<u>4 ВПРОВАДЖЕННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ</u>	57
<u>4.1 Розгортання програмного забезпечення</u>	57
<u>4.2 Підтримка програмного забезпечення</u>	59

<u>Висновки до розділу</u>	59
<u>ВИСНОВКИ</u>	61
<u>СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ</u>	63

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

IDE	– Integrated Development Environment (інтегроване середовище розробки)
API	– Application Programming Interface (прикладний програмний інтерфейс)
БД	– База даних
ПЗ	– Програмне забезпечення
BPMN	– Business Process Model and Notation (методологія моделювання бізнес-процесів)
C4	– Context, Containers, Components, Code (контекст, контейнери, компоненти, код)
ER	– Entity-Relationship (сутність-зв'язок)
GDPR	– General Data Protection Regulation (загальний регламент з питань захисту даних)

ВСТУП

Актуальність сучасних технологій та їх вплив на різні аспекти нашого життя невинно росте, і однією з ключових сфер, яка відчуває цей вплив, є прогнозування погоди. З урахуванням змін клімату та зростання інтересу до точних та швидких прогнозів, розробка нових методів та інструментів для передбачення погодніх умов стає надзвичайно важливою задачею.

За останні роки спостерігається збільшення інтересу громадськості до використання інтернет-технологій для отримання інформації про погоду. Ця тенденція створює потребу у нових та ефективних методах прогнозування, які враховують глобальні зміни та відповідають очікуванням сучасного суспільства.

Світові тенденції в розв'язанні схожих завдань вказують на інтенсивний розвиток технологій штучного інтелекту, великих даних та інтернету речей у сфері метеорології. Провідні науковці та фахівці активно впроваджують інновації для поліпшення точності та доступності прогнозів

Розроблений веб-сервіс може знайти широке застосування у різних галузях (туризм, сільське господарство, транспорт, енергетика, бізнес, безпека, споживчі додатки), надаючи користувачам цінну інформацію для прийняття обґрунтованих рішень.

1 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Загальні положення

В епоху інформаційних технологій програмне забезпечення відіграє визначальну роль у наданні актуальної та точної інформації користувачам. Аналіз вимог до програмного забезпечення є першим етапом у процесі розробки, оскільки він визначає основні потреби та очікування щодо функціональності та ефективності веб-сервісу.

Аналіз вимог є критично важливим етапом, на якому визначаються як функціональні, так і нефункціональні вимоги до розроблюваного програмного продукту. У випадку сервісу агрегації прогнозів погоди, це включає визначення необхідності точних та різноманітних даних, їхньої актуальності та можливості користувача отримати консолідовану інформацію з різних джерел.

Відправною точкою у розробці буде вірний та детальний аналіз вимог, оскільки від його якісної реалізації залежить ефективність та успішність всього проекту. Аналіз вимог визначить вектор подальших кроків у розробці веб-сервісу, забезпечуючи оптимальне задоволення потреб користувачів у прогнозуванні погоди. У даному розділі описано загальні визначення, ключові аспекти та напрямки розробки в цій області.

Гідрометцентри є ключовими установами, які займаються збором, аналізом та прогнозуванням метеорологічних та гідрологічних явищ.. Гідрометцентри включають в себе мережу метеорологічних станцій для збору даних про погодні умови. Ці станції обладнані різноманітними приладами для вимірювання відповідних параметрів. Гідрометцентри використовують зібрані дані для розробки метеорологічних прогнозів. Це включає в себе використання різних моделей та алгоритмів для передбачення поведінки умов на різні терміни.

Отримання даних з онлайн гідрометцентрів включає в себе процес звернення до відповідних веб-ресурсів або використання API, якщо такий надається.

У створенні подібного програмного забезпечення існують кілька ключових проблем, які можуть вплинути на якість та ефективність сервісу:

а) Надійність та достовірність даних

Агрегація даних з різних джерел може призвести до виникнення різниць у якості та достовірності інформації. Проблеми із синхронізацією, відмінностями у методах вимірювання, або неправильним обробленням даних можуть призвести до неточностей у прогнозах.

б) Стандартизація та об'єднання даних

Різні джерела можуть використовувати різні формати та структури даних. Проблеми стандартизації та їх об'єднання можуть ускладнювати процес агрегації та зробити його менш ефективним;

в) Актуальність та частота оновлення даних

Для точних прогнозів важливо мати актуальні дані. Затримки у процесі агрегації та оновлення інформації можуть вплинути на достовірність прогнозів та надійність сервісу;

г) Обробка великого об'єму даних

Погодні дані можуть бути об'ємними, і обробка їх у режимі реального часу може становити велике завдання. Важливо оптимізувати процес агрегації та обробки, щоб забезпечити ефективність та швидкість реакції сервісу;

д) Змінність погодних умов

Погодні умови можуть швидко змінюватися, і невизначеність в їх прогнозуванні є великим викликом. Важливо розробляти алгоритми, які враховують динаміку змін погоди та надають користувачам актуальну інформацію.

Вирішення цих проблем вимагатиме не лише вдосконалення технічних аспектів агрегації даних, але й глибокого розуміння особливостей метеорологічних процесів та розробки адаптивних стратегій управління даними для забезпечення найвищої якості прогнозів погоди через веб-сервіс.

В контексті прогнозування погоди, метрики грають ключову роль у вимірюванні та оцінці стану погоди. Вони дозволяють користувачам на їх основі зробити свої висновки для прийняття подальших рішень. Основні метрики, які використовуються в області прогнозування погоди, можуть бути наступними [1]:

- температура повітря: один із ключових параметрів погоди, який має значущий вплив на життя, комфорт та безпеку людей і природи загалом.
- відчуття температури повітря: враховує вплив вітру та вологості на сприйняття температури людським організмом.
- атмосферний тиск: міра ваги стовпа повітря над певною точкою. важливий показник для змін погодніх умов та впливу на здоров'я людей.
- вологість повітря: відсоткове співвідношення кількості водяної пари в повітрі до максимально можливої кількості при даній температурі.
- швидкість вітру: впливає на теплообмін та метеорологічні умови.
- хмарність: Визначає частку неба, покриту хмарами.
- ймовірність опадів: Інформація про ймовірність випадіння опадів у певний час.
- тип опадів: Якісна характеристика опадів (дощ, сніг тощо).

1.2 Змістовний опис і аналіз предметної області

Знання з метеорології широко використовуються для створення моделей, аналізу гідрометеорологічних даних та прогнозування погоди. На сучасному етапі розвитку ІТ-технологій використання знань з метеорології у програмному забезпеченні має вирішальне значення для надання точних та актуальних

прогнозів погоди. Процес імплементації предметної області у сфері ІТ, зокрема в галузі прогнозування погоди, активно розвивається, проте наявні певні недоліки та виклики:

- короткострокові прогнози залишаються менш точними через складність передбачення миттєвих змін погоди;
- нерівномірність гідрометеорологічних спостережень у деяких регіонах може обмежувати точність прогнозів;
- інтеграція та оптимальне використання супутникових даних може бути несуттєвою.

Можливі шляхи покращення:

- впровадження технологій штучного інтелекту для покращення аналізу та передбачення погодних умов;
- покращення систем для автоматичного збору та обробки гідрометеорологічних даних;
- зміцнення співпраці між національними та міжнародними гідрометцентрами для обміну даними.

У рамках курсової роботи визначено, що основний акцент буде розміщено на використанні декількох гідрометцентрів для підвищення точності короткострокових погодних прогнозів. Впровадження такого підходу передбачає використання різноманітності погодних моделей та методів прогнозування для отримання більш точних та надійних результатів.

1.3 Аналіз існуючих технологій та успішних ІТ-проєктів

Проаналізуємо відоме на сьогодні алгоритмічне забезпечення у даній області та технічні рішення, що допоможуть у реалізації веб-сервісу прогнозування погоди. Далі будуть розглянуті допоміжні програмні засоби, засоби розробки та готові програмні рішення.

1.3.1 Аналіз відомих алгоритмічних та технічних рішень

Розроблюваний веб-сервіс буде надійним ресурсом для користувачів, адже він забезпечить агрегацію високоякісних метеорологічних даних, зокрема від OpenWeatherMap[2], VisualCrossing[3] та sinoptik.ua[4], забезпечуючи точні прогнози погоди для різних регіонів та користувацьких потреб. Програмне забезпечення вбере в себе найкращі аспекти уже відомих технічних рішень. Порівняти їх сильні та слабкі сторони можна за допомогою наведеної нижче таблиці 1.1.

Таблиця 1.1 – Порівняння технічних рішень

Функціонал	OpenWeatherMap	VisualCrossing	sinoptik.ua
Тип сервісу	Глобальний	Глобальний	Регіональний та Національний
Тип доступу	Безкоштовний та Платний	Безкоштовний та Платний	Безкоштовний
API Доступ	Так	Так	Ні
Географічне покриття	Глобальне	Глобальне	Регіональне
Штучний інтелект	Ні	Та	Ні
Поточні умови	Так	Так	Та
Прогнози	Так	Так	Так
Мапи	Так	Так	Так

1.3.2 Аналіз допоміжних програмних засобів та засобів розробки

Вибір мови програмування, фреймворків та допоміжних засобів для розробки веб-сервісу є стратегічним рішенням і залежить від ряду факторів, таких як продуктивність, зручність розробки, підтримка спільноти, можливості для розширення та інші. Ось перелік обраних технологій:

а) Мова програмування Python[5]

Python відомий своєю лаконічністю та читабельністю коду, що полегшує розробку та підтримку проекту. Також важливим фактором стала величезна екосистема бібліотек та фреймворків Python, що сприяє швидкій та ефективній розробці різноманітних функцій.

б) Веб-фреймворк Flask[6]

Flask відомий своєю простотою та гнучкістю, що ідеально підходить для розробки веб-сервісів різного рівня складності. Flask має велику та активну спільноту, яка надає підтримку та ресурси для розробників.

в) Бібліотеки (asyncio[7], bs4[8], numpy[9], pandas[10])

Бібліотеки обрано відповідно до задач, які виникнуть під час розробки:

- асинхронність: asyncio дозволяє ефективно виконувати асинхронні запити, що може бути важливо для оптимізації веб-сервісу, який обробляє багато запитів одночасно.
- web scraping: BeautifulSoup допомагає витягувати дані з веб-сторінок, що корисно для агрегації інформації з різних джерел.
- обробка даних: numpy та pandas є потужними інструментами для обробки та аналізу даних, що є важливим для роботи з прогнозами погоди та агрегації інформації.

г) IDE Pycharm[11]

Pycharm є потужним інтегрованим середовищем розробки для Python, яке пропонує багато корисних функцій, що полегшують роботу розробників.

д) React JS[12] (веб-застосунок):

Для розробки веб-застосунку, що демонструватиме роботу веб-сервісу, основними критеріями вибору технологій були сучасність та швидкодія.

Ідеально підходить мова програмування JavaScript[13] з фреймворком React, що надає можливість створення ефективних застосунків.

Отже, усі технології були обрані з огляду на їхню ефективність, зручність використання та широкий спектр можливостей для успішної реалізації вашого веб-сервісу прогнозування погоди.

1.3.3 Аналіз відомих програмних продуктів

У даному пункті вивчається і оцінюється ряд вже існуючих програмних рішень у сфері прогнозування погоди та агрегації метеорологічних даних. Цей аналіз спрямований на виявлення сильних та слабких сторін конкурентів, визначення особливостей їхніх функціональних можливостей та визначення тенденцій у розвитку галузі.

Метою даного аналізу є поглиблене вивчення того, як існуючі програмні рішення впорядковують та представляють метеорологічні дані, їхня ефективність у прогнозуванні погоди, а також можливості для подальшого вдосконалення. Цей розділ виявить ключові особливості кожного програмного продукту та створить базу для подальшого порівняльного аналізу та розробки унікального підходу до власного веб-сервісу прогнозування погоди.

Серед відомих програмних продуктів, що розглядаються в даному розділі, зазначаються такі світові гравці, як, MeteoMatics[14] та The Weather Channel[15]. Аналіз їхніх характеристик, функціональності та інноваційних рішень надасть можливість отримати об'єктивну оцінку сучасного ринку метеорологічних сервісів.

The Weather Channel є одним з найбільших та найвідоміших веб-сервісів прогнозу погоди. Застосунок є довіреним джерелом метеорологічної інформації, яке надає точні та оновлювані прогнози, використовуючи широкий спектр даних та інтерактивних засобів для задоволення різноманітних потреб користувачів у вивченні та розумінні погодних умов. Розглянемо основні переваги даного сервісу:

- а) Географічне покриття: The Weather Channel надає прогнози погоди для різних частин світу. Вони пропонують інформацію як для глобальних, так і для регіональних масштабів.
- б) Прогноз та поточні умови: сервіс надає користувачам не тільки прогнози на різний термін, але й інформацію про поточні погодні умови, включаючи температуру повітря, вологість, швидкість вітру, атмосферний тиск тощо.
- в) Інтерактивні карти: володіє рядом інтерактивних карт, що дозволяють користувачам візуалізувати різні погодні параметри, такі як опади, температура, хмарність та інші.
- г) Відеоматеріали: відомий своїми відеоматеріалами, The Weather Channel часто надає новини та аналіз погоди через відеоформат.
- д) Інші функції: The Weather Channel може також включати інші функції, такі як радар, алерти про надзвичайні ситуації та інші сервіси, які підвищують загальний досвід користувачів.

MeteoMatics – це компанія, яка спеціалізується на аналізі та прогнозуванні погодних умов. Їхні послуги охоплюють широкий спектр метеорологічних даних та аналітики, включаючи дані від датчиків, інформацію від метеорологічних станцій та моделі прогнозування. Основні переваги:

- а) Штучний інтелект (ШІ): MeteoMatics використовує технології штучного інтелекту для поліпшення точності прогнозування та аналізу великих обсягів метеорологічних даних.
- б) Дані від датчиків: спростовуючи дані від різних датчиків, MeteoMatics може надавати більш точні та актуальні інформацію про погоду.
- в) Метеорологічна аналітика: крім простого прогнозування, MeteoMatics може надавати аналітичні звіти та висновки на основі зібраних даних, що може бути корисним для підприємств, що залежать від точних прогнозів.

г) Платформа для розробників: MeteoMatics надає API або інші інструменти для розробників, які дозволяють інтегрувати їхні метеорологічні дані у власні додатки чи сервіси.

Основними перевагами власної розробки над конкурентами, які повинні привабити пересічного користувача:

- а) Можливість отримання основних метеорологічних даних, без потреби розбиратись у складних API, які надають широкий спектр різних властивостей та метрик.
- б) Локалізація одиниць виміру у зручний формат, саме для користувачів нашого регіону.
- в) Цілковита безкоштовність сервісу.

Для порівняння курсової роботи з аналогом можна скористатись таблицею 1.2.

Таблиця 1.2 – Порівняння з аналогом

Функціонал	Власна розробка	The Weather Channel	MeteoMatics
Тип сервісу	Метеорологічний	Метеорологічний, аналітичний	Метеорологічний, аналітичний
Тип доступу	Безкоштовний	Безкоштовний Платний	Платний
API Доступ	Так	Ні	Так
Географічне покриття	Глобальне	Глобальне	Глобальне
Локалізація одиниць	По стандарту	Заданням додаткових параметрів	Заданням додаткових параметрів
Поточні умови	Так	Так	Ні
Прогнози	Так	Так	Так
Детальні дані	Ні	Так	Так

1.4 Аналіз вимог до програмного забезпечення

Програмне забезпечення буде складатися з двох частин: безпосередньо веб-сервіс головною функцією якого є надання погодних даних та веб-застосунок, що отримуватиме ці дані та візуалізуватиме для користувачів

Основні функції веб-сервісу можна побачити на рисунку 1.1:

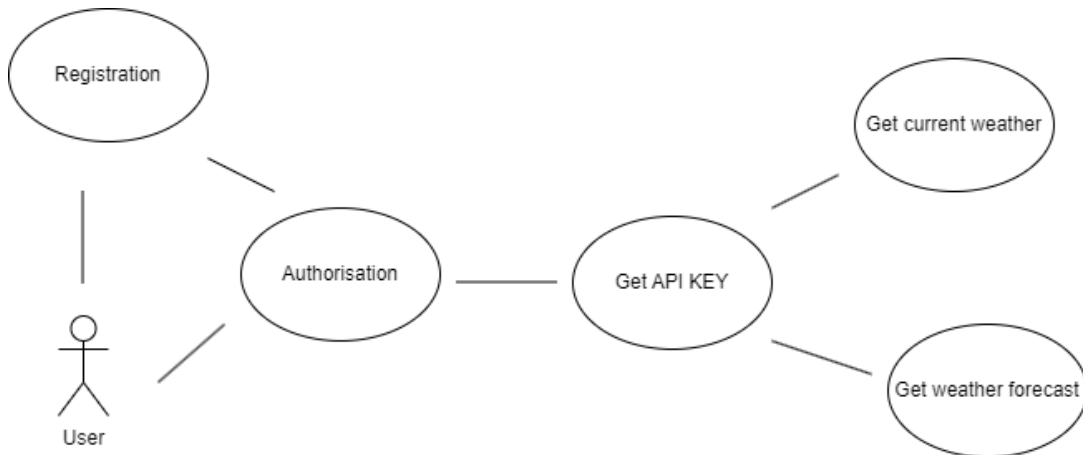


Рисунок 1.1 – Діаграма варіантів використання веб-сервісу

В таблицях 1.3 - 1.7 наведені варіанти веб-сервісу.

Таблиця 1.3 - Варіант використання UC-01

Use case name	Реєстрація користувача
Use case ID	UC-01
Goals	Реєстрація нового користувача в системі
Actors	Гість (неzareєстрований користувач)
Trigger	Користувач бажає zareєструватися
Pre-conditions	-
Flow of Events	Користувач переходить на сторінку реєстрації. В поля для реєстрації вводяться відповідні дані: пошта користувача, пароль в системі, та його повтор для підтвердження. Після заповнення даних користувача натискає кнопку реєстрації. Після цього з'являється повідомлення про успішну реєстрацію, і користувач перенаправляється на сторінку входу.
Extension	-

Post-Condition	Створення сторінки користувача, перехід на сторінку входу
----------------	---

Таблиця 1.4 - Варіант використання UC-02

Use case name	Авторизація користувача
Use case ID	UC-02
Goals	Успішний вхід зареєстрованого користувача в систему
Actors	Користувач
Trigger	Користувач бажає увійти в систему.
Pre-conditions	Користувач повинен бути зареєстрований в системі
Flow of Events	Користувач переходить на сторінку входу в систему. Він вводить свою електронну адресу та пароль. Користувач натискає кнопку "Увійти". Система перевіряє правильність введених даних. Якщо дані введено вірно, система входить користувача в систему та перенаправляє його на особистий кабінет. Якщо дані введено невірно, система виводить повідомлення про помилку та пропонує повторити вхід.
Extension	Якщо користувач введе невірну електронну адресу або пароль, система виводить повідомлення про помилку та пропонує повторити вхід.
Post-Condition	Успішний вхід користувача в систему та перехід на його особистий кабінет.

Таблиця 1.5 - Варіант використання UC-03

Use case name	Отримання API KEY
Use case ID	UC-03
Goals	Отримання API KEY для подальшого використання його в запитах
Actors	Користувач

Trigger	Користувач бажає отримати API KEY.
Pre-conditions	Користувач повинен бути зареєстрований та увійшов в систему.
Flow of Events	Користувач, увійшовши в систему, переходить на свій особистий кабінет. На сторінці особистого кабінету з'являється кнопка "Get API KEY ". Користувач натискає на кнопку для отримання ключа. Система генерує API KEY та повідомляє користувача про успішне отримання. API KEY відображається на екрані та зберігається в особистому кабінеті користувача.
Extension	Якщо при генерації API KEY виникає помилка, система повідомляє користувача про це та пропонує спробувати ще раз.
Post-Condition	Успішне отримання API KEY.

Таблиця 1.6 - Варіант використання UC-04

Use case name	Отримання поточної погоди
Use case ID	UC-04
Goals	Отримати актуальні погодні дані для вказаного місця.
Actors	Користувач
Trigger	Користувач бажає дізнатися поточні погодні умови.
Pre-conditions	Користувач повинен бути зареєстрований та мати API KEY.
Flow of Events	Користувач здійснює запит за ендпоінтом /weather, передаючи необхідні параметри (apikey, lat, lon). Система обробляє запит, використовуючи API KEY для автентифікації та визначає поточні погодні умови за вказаними координатами (lat, lon). Система повертає користувачеві дані про поточну погоду у форматі JSON.
Extension	Якщо надано невірний API KEY або параметри запиту, система повідомляє про помилку та надає відповідне повідомлення..
Post-Condition	Користувач поточні погодні умови.

Таблиця 1.7 - Варіант використання UC-05

Use case name	Отримання прогнозу погоди
Use case ID	UC-05
Goals	Отримати прогноз погоди на вказаний період для вказаного місця.
Actors	Користувач
Trigger	Користувач бажає дізнатися прогноз погоди на певний період.
Pre-conditions	Користувач повинен бути зареєстрований та мати API KEY.
Flow of Events	Користувач здійснює запит за ендпоінтом /forecast, передаючи необхідні параметри (apikey, lat, lon , days - опціонально). Система обробляє запит, використовуючи API KEY для автентифікації та визначає поточні погодні умови за вказаними координатами (lat, lon) на вказану кількість днів (за замовчуванням 7 днів). Система повертає користувачеві дані про поточну погоду у форматі JSON.
Extension	Якщо надано невірний API KEY або параметри запиту, система повідомляє про помилку та надає відповідне повідомлення..
Post-Condition	Користувач отримує прогноз погоди на визначений період.

Основні функції веб- застосунку можна побачити на рисунку 1.2:

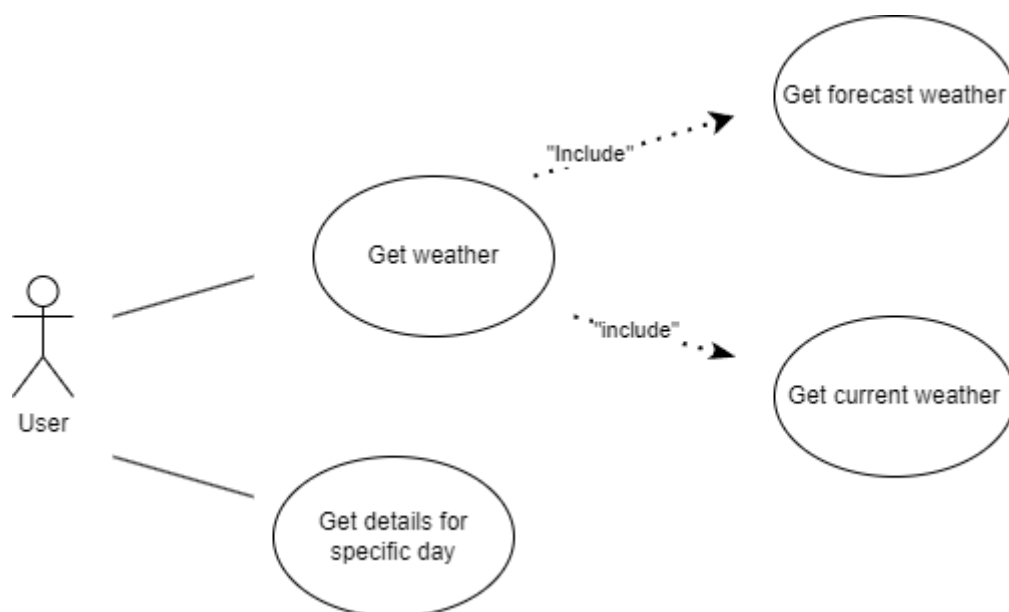


Рисунок 1.2 – Діаграма варіантів використання веб-сервісу

В таблицях 1.8 - 1.9 наведені варіанти використання веб-застосунку.

Таблиця 1.8 - Варіант використання UC-01

Use case name	Отримання погоди
Use case ID	UC-01
Goals	Отримання погоди у заданому місті
Actors	Гість (незареєстрований користувач)
Trigger	Користувач бажає отримати погоду у конкретному місті
Pre-conditions	-
Flow of Events	Користувач натискає на поле для пошуку та починає вводити назву міста. Користувач вибирає у випадяючому списку потрібне місто. На сторінці з'являється інформація про поточну погоду та прогноз погоди на 7 днів.
Extension	-
Post-Condition	На сторінці відображено інформацію

Таблиця 1.9 - Варіант використання UC-07

Use case name	Отримання детальної інформації
Use case ID	UC-02
Goals	Отримання детальної інформації про погоду у конкретний день
Actors	Гість (незареєстрований користувач)
Trigger	Користувач бажає отримати детальнішу інформацію
Pre-conditions	Користувач отримав погоду за містом
Flow of Events	Користувач на сторінці з погодою натискає на потрібний день та отримує детальнішу інформацію.
Extension	-
Post-Condition	Вкладку вибраного дня розгорнуто.

1.4.1 Розроблення функціональних вимог

Програмне забезпечення розділене на модулі. Кожен модуль має свій певний набір функцій. На рисунку 1.4 наведено загальну модель вимог веб-сервісу, а в таблицях 1.10 – 1.22 наведений опис функціональних вимог.

Опис	Назва	Приоритет	Ризик
1. Реєстрація користувача	FR-1	2	Високий
1.1. Перевірка логіну	FR-2		
1.2. Перевірка паролю	FR-3		
2. Авторизація користувача	FR-4	2	Високий
2.1. Перевірка логіну	FR-2		
2.2. Перевірка паролю	FR-3		
3. Доступ за API ключем	FR-5	2	Високий
4. Повернення поточної погоди	FR-6	1	Високий
4.1. Перевірка координат	FR-7		
4.2. Отримання даних з трьох джерел	FR-8		
4.3. Агрегація даних	FR-9		
4.4. Повернення даних у форматі JSON	FR-10		
5. Отримання прогнозу погоди	FR-11	3	Середній
5.1. Перевірка координат	FR-7		
5.2. Перевірка кількості днів	FR-12		
4.2. Отримання даних з трьох джерел	FR-8		
4.3. Агрегація даних	FR-9		
4.4. Повернення даних у форматі JSON	FR-10		

Рисунок 1.4 – Модель вимог веб-сервісу у загальному вигляді

Таблиця 1.10 – Функціональна вимога FR-1

Назва	Реєстрація користувача
Опис	Система повинна надавати можливість реєстрації користувачеві шляхом введення логіну, паролю, підтвердження паролю.

Таблиця 1.11 – Функціональна вимога FR-2

Назва	Перевірка логіну
Опис	Система повинна надавати можливість перевірки на коректність введеного користувачем логіну.

Таблиця 1.12 – Функціональна вимога FR-3

Назва	Перевірка паролю
-------	------------------

Опис	Система повинна надавати можливість перевірки на коректність введеного користувачем паролю.
------	---

Таблиця 1.13 – Функціональна вимога FR-4

Назва	Авторизація користувача
Опис	Система повинна надавати можливість авторизації зареєстрованому користувачеві шляхом введення логіну та паролю.

Таблиця 1.14 – Функціональна вимога FR-5

Назва	Доступ за API KEY
Опис	Система повинна надавати можливість отримання API ключа авторизованому користувачеві та його використання для запитів.

Таблиця 1.15 – Функціональна вимога FR-6

Назва	Отримання поточної погоди
Опис	Система повинна надавати можливість отримання поточної погоди.

Таблиця 1.16 – Функціональна вимога FR-7

Назва	Перевірка координат
Опис	Система повинна надавати можливість перевірку на коректність введених користувачем координат.

Таблиця 1.17 – Функціональна вимога FR-8

Назва	Отримання даних з трьох джерел
Опис	Система повинна отримувати дані про погоду від трьох різних джерел.

Таблиця 1.18 – Функціональна вимога FR-9

Назва	Агрегація даних
Опис	Система повинна мати механізми агрегації отриманих даних для підвищення точності прогнозу.

Таблиця 1.19 – Функціональна вимога FR-10

Назва	Повернення даних у форматі JSON
Опис	Система повинна надавати можливість отримання погоди у форматі JSON.

Таблиця 1.21 – Функціональна вимога FR-11

Назва	Отримання прогнозу погоди
Опис	Система повинна надавати можливість отримання прогнозу погоди.

Таблиця 1.22 – Функціональна вимога FR-12

Назва	Перевірка кількості днів
Опис	Система повинна перевіряти на коректність введену користувачем кількість днів

Таблиця 1.23 – Матриця трасування вимог веб-сервісу

	FR-1	FR-2	FR-3	FR-4	FR-5	FR-6	FR-7	FR-8	FR-9	FR-10	FR-11	FR-12
UC-1	+	+	+									
UC-2		+	+	+								
UC-3					+							
UC-4						+	+	+	+	+		
UC-5							+	+	+	+	+	+

На рисунку 1.5 наведено загальну модель вимог веб-застосунку, а в таблицях 1.24 – 1.30 наведений опис функціональних вимог, що забезпечать користувачам зручний та інформативний доступ до поточної погоди та прогнозу за допомогою веб-застосунку.

Опис	Назва	Приоритет	Ризик
1. Отримання погоди	FR-1	1	Високий
1.1. Пошуку за містом	FR-2		
1.1.1. Автозаповнення міста	FR-3		
1.2. Отримання поточної погоди	FR-4		
1.3. Отримання прогнозу погоди	FR-5		
2. Перегляд детальної інформації	FR-6	2	Середній
2.1 Вибір конкретного дня	FR-7		

Рисунок 1.5 – Модель вимог веб-застосунку у загальному вигляді

Таблиця 1.24 – Функціональна вимога FR-1

Назва	Отримання погоди
Опис	Система повинна надавати можливість користувачеві отримання погоди.

Таблиця 1.25 – Функціональна вимога FR-2

Назва	Пошук за містом
Опис	Система повинна мати механізми для пошуку погоди за містом.

Таблиця 1.26 – Функціональна вимога FR-3

Назва	Автозаповнення міста
Опис	Система повинна надавати можливість автозаповнення назви міста, щоб спростити та прискорити процес введення.

Таблиця 1.27 – Функціональна вимога FR-4

Назва	Отримання поточної погоди
Опис	Користувач повинен мати можливість отримати актуальну

	інформацію про поточні погодні умови для введеного міста.
--	---

Таблиця 1.28 – Функціональна вимога FR-5

Назва	Отримання прогнозу погоди
Опис	Користувач повинен мати можливість отримати прогноз погоди на найближчі 7 днів для введеного міста.

Таблиця 1.29 – Функціональна вимога FR-6

Назва	Перегляд детальної інформації
Опис	Користувач повинен мати можливість отримати детальну інформацію про погоду.

Таблиця 1.30 – Функціональна вимога FR-7

Назва	Вибір конкретного дня
Опис	Користувач повинен мати можливість отримати детальну інформацію для конкретного обраного дня.

Таблиця 1.31 – Матриця трасування вимог веб-застосунку

	FR-1	FR-2	FR-3	FR-4	FR-5	FR-6	FR-7
UC-1	+	+	+	+	+		
UC-2						+	+

1.4.2 Розроблення нефункціональних вимог

Таблиця 1.32 – Нефункціональна вимога NFR-1

Назва	Точність даних
-------	----------------

Опис	Система повинна забезпечувати високий рівень точності та надійності отриманих даних.
------	--

Таблиця 1.33 – Нефункціональна вимога NFR-2

Назва	Безпека даних
Опис	Система повинна забезпечувати безпеку та конфіденційність обміну даними, включаючи API KEY та особисті дані користувачів.

Таблиця 1.34 – Нефункціональна вимога NFR-3

Назва	Швидкодія
Опис	Система повинна забезпечувати швидку відповідь на запити.

Таблиця 1.35 – Нефункціональна вимога NFR-4

Назва	Сумісність
Опис	Система повинна бути сумісною з різними браузерами та пристроями.

Таблиця 1.36 – Нефункціональна вимога NFR-5

Назва	Надійність
Опис	Система повинна бути стійкою до великого обсягу запитів та забезпечувати надійну роботу.

1.5 Постановка задачі

У рамках курсової роботи буде розроблено веб-сервіс прогнозування погоди, шляхом агрегації даних онлайн гідрометцентрів. Програмне забезпечення надаватиме доступ користувачам до поточного стану погоди та прогнозу на найближчі дні з використанням трьох різних джерел даних. Також для демонстрації роботи сервісу буде реалізовано веб-застосунок, що використовуватиме його дані.

Метою розробки є підвищення точності отриманої інформації про поточну та майбутню погоду для полегшення прийняття рішень у бізнесі чи повсякденному житті.

Серед задач, що підлягають розв'язанню у результаті розробки програмного забезпечення, можна виділити основні:

- а) створення та розвиток веб-сервісу, який дозволить користувачам швидко та зручно отримувати актуальні погодні дані;
- б) інтеграція зовнішніх джерел (OpenWeatherMap, VisualCrossing, sinoptik.ua) для покращення точності та повноти інформації;
- в) реалізація агрегації отриманих даних та приведення їх до єдиного стандарту;
- г) створення API для надання доступу до погодних даних, зокрема отримання поточної погоди та прогнозів на визначений період;
- д) оптимізація швидкості роботи сервісу та використання пам'яті;
- е) розробка та впровадження заходів безпеки для збереження конфіденційності особистих даних користувачів та API ключів;
- ж) забезпечення ефективної роботи веб-сервісу навіть при великому обсязі запитів, забезпечення швидкої відповіді на запити користувачів;

Веб-застосунок передбачатиме створення зручного веб-інтерфейсу, що дозволяє користувачам легко отримувати доступ до інформації. Також буде реалізовано механізм пошуку та автозаповнення для зручного та швидкого введення назв міст.

Висновки до розділу

У результаті аналізу вимог до програмного забезпечення було виявлено ключові аспекти та напрямки, які визначатимуть успішний розвиток та функціональність веб-сервісу прогнозування погоди.

Ми ознайомилися з предметною областю та проблемами, які можуть виникнути під час розробки даного програмного забезпечення. Проаналізували існуючі технології та наявні успішні ІТ-проекти, за допомогою таблиць

порівнянь. Визначилися з допоміжними програмними засобами, які будуть використані для розробки, а саме мови програмування Python, JavaScript та відповідно фреймворки Flask, React, також деякі допоміжні бібліотеки та середовище розробки Pycharm. Далі основні функції веб-сервісу та застосунку було візуалізовано за допомогою діаграм варіантів використання та детально описано у таблицях. На основі цього було синтезовано модель вимог та відповідно до неї сформовано функціональні вимоги, серед яких забезпечення реєстрації та авторизації користувача, отримання інформації про поточну погоду та прогноз через використання API ключа. Для узагальнення представлено матрицю трасування вимог. Для веб-застосунку було окремо побудовано модель, матрицю та безпосередньо самі вимоги. Також визначено основні нефункціональні вимоги. У останньому підрозділі було чітко визначено мету та задачі розробки.

Отже, аналіз вимог дозволяє визначити шлях розвитку та імплементації веб-сервісу, спрямованого на забезпечення користувачів актуальною та достовірною інформацією.

2 МОДЕЛЮВАННЯ ТА КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Моделювання та аналіз програмного забезпечення

У даному розділі проведено детальний огляд методології моделювання бізнес-процесів BPMN (Business Process Model and Notation) та аналіз його застосування для розробки веб-сервісу прогнозування погоди. BPMN стане ключовим інструментом для визначення та оптимізації процесів, що відбуваються на кожному етапі від збору даних до надання користувачам актуальної погодної інформації. Аналіз використання BPMN у програмному забезпеченні дозволить з'ясувати його переваги та визначити ефективність в контексті розробки та управління процесами, пов'язаними з прогнозуванням погоди.

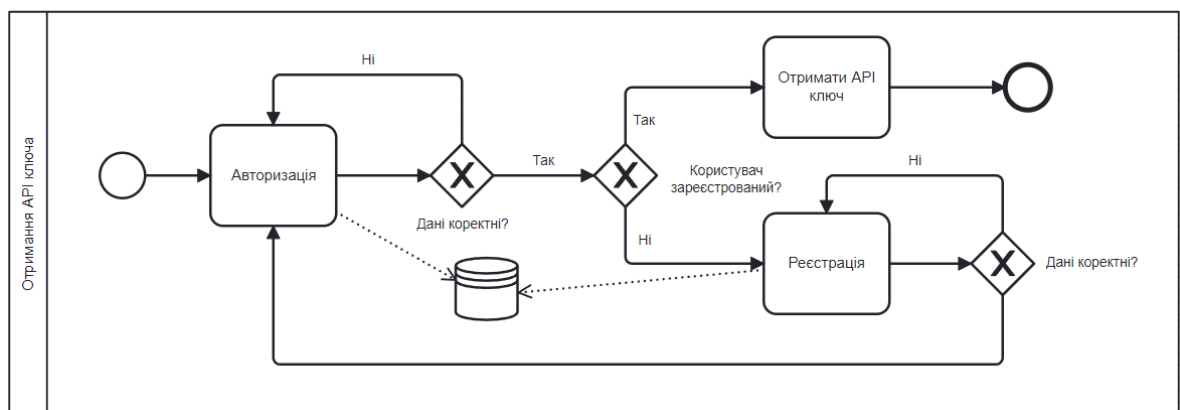


Рисунок 2.1 – BPMN модель отримання API ключа

Опис послідовності отримання API ключа користувачем:

- а) користувач переходить на сторінку авторизації та вводить свій логін та пароль;
- б) якщо введено некоректні дані, користувач відповідні поля підсвічуються помилкою;
- в) якщо дані коректні, то перевіряється чи зареєстрований користувач;
- г) зареєстрований користувач потрапляє у особистий кабінет, де може отримати API ключ.

д) якщо користувач не зареєстрований, то він перенаправляється на сторінку реєстрації;

е) якщо введено коректні дані, користувач повертається до пункту а);

На рисунку 2.2 показано процес отримання поточної погоди. На діаграмі можна побачити взаємодію користувача з веб-сервісом.

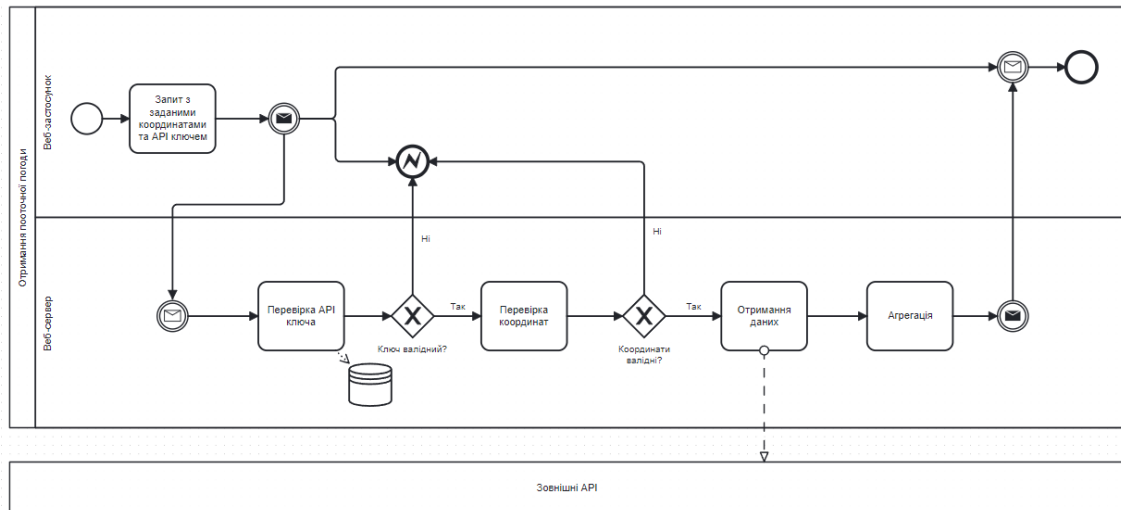


Рисунок 2.2 – BPMN модель отримання поточної погоди

Опис послідовності отримання користувачем поточної погоди:

а) користувач, у даному випадку веб-застосунок, надсилає запит до веб-сервісу;

б) при передачі некоректного API ключа або координат сервер повертає помилку;

в) сервіс звертається до зовнішніх API гідрометцентів, щоб отримати дані;

г) далі відбувається агрегація та приведення до єдиного формату отриманих даних;

д) сервер формує відповідь у вигляді JSON файлу та надсилає користувачеві.

2.2 Архітектура програмного забезпечення

C4 Model (Context, Containers, Components, Code) - це візуальна модель, яка дозволяє представити архітектуру програмного забезпечення на різних рівнях деталізації. Розглянемо перший рівень на рисунку 2.3 для веб-сервісу та застосунку прогнозування погоди:

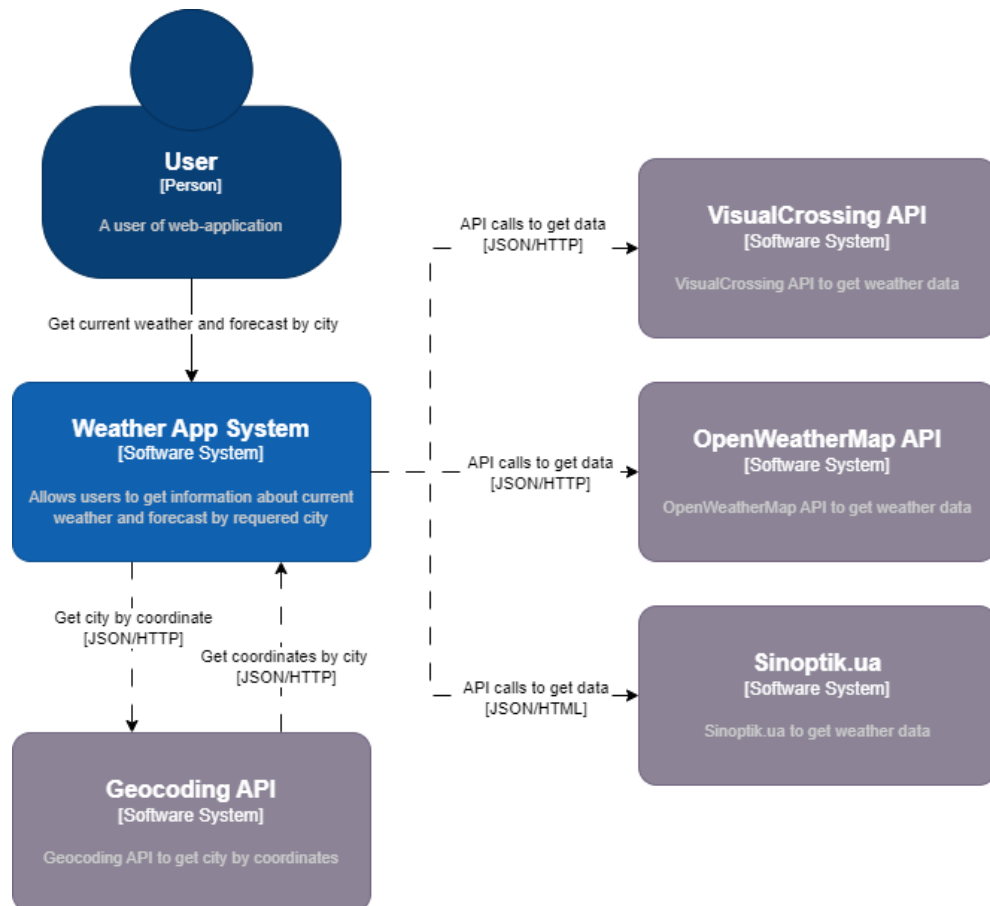


Рисунок 2.3 – Перший рівень C4 Model

Перший рівень в C4 Model - це діаграма контексту. Цей рівень надає високорівневий огляд системи та її взаємодії з зовнішніми сутностями. На цьому рівні визначається, як система взаємодіє з оточуючим світом. Основна мета - показати систему та її основні сутності без деталей їх внутрішньої будови. Було виділено такі основні сутності:

- Система: веб-сервіс та веб-застосунок для прогнозу погоди;
- Користувачі: кінцеві користувачі, які використовують ваш веб-сервіс та застосунок для отримання погодної інформації;

- в) Зовнішні API для отримання даних (OpenWeatherMap, VisualCrossing, sinoptik.ua): сутності, які надають погодні дані через свої API;
- г) Зовнішнє API для роботи з координатами (Geocoding API).

На другому рівні (діаграма контейнерів) моделюється внутрішня будова системи та з'ясовуються взаємозв'язки між її основними компонентами. Отже, розкриємо структуру та взаємодії між групами компонентів побудувавши другий рівень, рисунок 2.4.

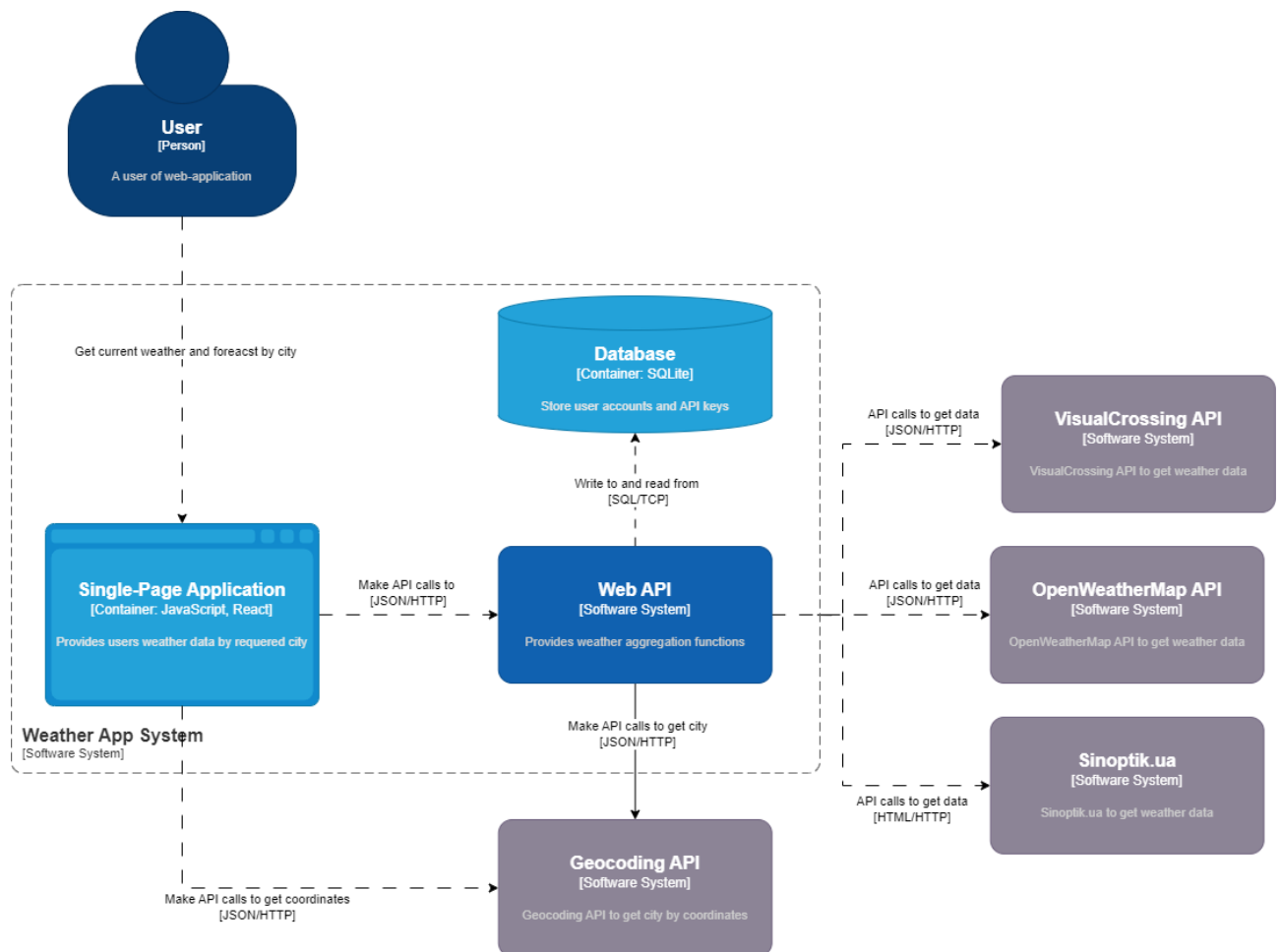


Рисунок 2.4 – Другий рівень C4 Model

Бачимо, що роботи нашої системи складається з веб-сервісу (Weather API), у якому виконується агрегація даних та основна логіка програмного забезпечення. Веб-сервіс взаємодіє з базою даних для запису та читання даних користувачів. Користувач отримує інформацію про погоду, використовуючи інтерфейс веб-застосунку, як проміжну ланку між ним та веб-сервісом.

2.3 Конструювання програмного забезпечення

Робота веб-сервісу побудована на отриманні даних із зовнішніх джерел за допомогою HTTP запитів. У випадку агрегації даних для прогнозу на 7 днів, кількість запитів доходить до десяти, що суттєво впливає на швидкість роботи програмного забезпечення. Бібліотека requests виконує запити, по черзі, один за одним, і це пояснює такі затрати у часі. Через таку специфіку, довелося розробляти рішення, яке надсилатиме запити більш оптимізовано.

Таким рішенням стали бібліотеки asyncio та aiohttp, оскільки вони разом надають потужний інструментарій для ефективного оброблення асинхронних HTTP-запитів. asyncio вбудовує механізми для створення асинхронних функцій та управління потоками виконання. Завдяки цьому, коли одна частина коду чекає відповіді на запит, інші частини можуть продовжувати виконання, що дозволяє рівномірно розподілити навантаження та підвищити продуктивність.

Такий підхід дозволяє вам забезпечити високу продуктивність веб-сервісу, надаючи користувачам швидкий доступ до актуальної інформації про погоду та знижуючи час відповіді на їхні запити.

В якості системи управління базами даних використовується MySQL. База даних серверу призначена для зберігання користувачів, а також даних про їх API ключі. Модель бази даних наведена на рисунку 2.5. Опис таблиць бази даних наведено у таблицях 2.6 - 2.7.

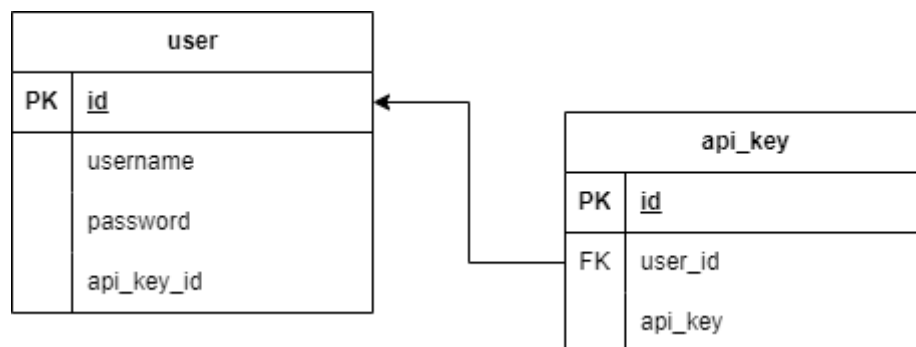


Рисунок 2.5 – Модель бази даних

Таблиця 2.2 – Опис таблиці user

Таблиця	Назва поля	Тип даних	Опис
---------	------------	-----------	------

user	id	INT	ідентифікаційний номер користувача
	username	VARCHAR(256)	ім'я користувача або електронна пошта користувача
	password	VARCHAR(128)	пароль користувача

Таблиця 2.3 – Опис таблиці api_key

Таблиця	Назва поля	Тип даних	Опис
api_keys	id	INT	ідентифікаційний номер ключа
	user_id	INT	ідентифікаційний номер користувача
	api_key	VARCHAR(64)	API ключ користувача

Програма написана з використанням методології ООП, що передбачає об'єднання даних та функцій, що з ними пов'язані, в єдиний об'єкт або клас. Об'єктно-орієнтоване програмування базується на концепції "об'єкта", який є екземпляром класу і об'єднує дані (змінні) та методи (функції), які опрацьовують ці дані.

ООП дозволяє структурувати код таким чином, що він відображає реальний світ і спрощує розробку та обслуговування програм. Програми, написані з використанням ООП, зазвичай є більш зрозумілими, легшими для розширення та модифікації.

Вибір ООП має численні переваги, такі як підвищення читабельності коду, зручність управління складними системами, можливість використання готових класів і бібліотек, а також полегшення розподілу роботи між

командами розробників. Внутрішню структуру проекту можна побачити, поглянувши на діаграму класів рисунок 2.6.

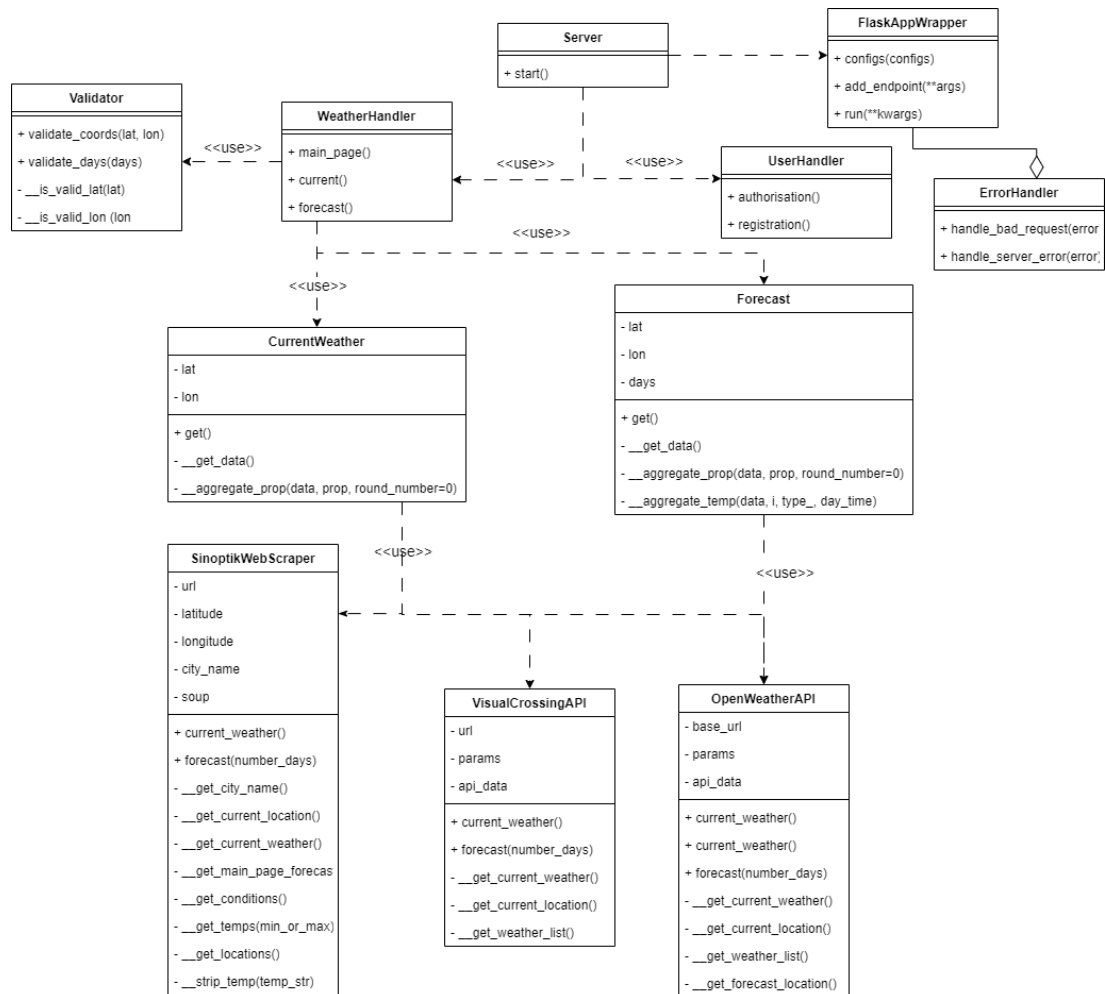


Рисунок 2.6 – Діаграма класів

Як бачимо, отримання даних з джерел реалізовано відповідними класами (OpenWeatherAPI, VisualCrossingAPI, SinoptikWebScraper). У класі CurrentWeather відбувається агрегація даних про поточну погоду, у Forecast – прогнозів. Запит користувача направляється до WeatherHandler, де відбувається перевірка заданих параметрів за допомогою Validator, що містить відповідні методи валідації. Далі, якщо дані введені правильно, WeatherHandler звертається до агрегаторів. Також присутній UserHandler, який займається реєстрацією, авторизацією та видачею API ключів. ErrorHandler обробляє помилки, що можуть виникнути у процесі роботи сервісу.

Опис утиліт та іншого стороннього програмного забезпечення, що використовується у розробці наведено в таблиці 2.4.

Таблиця 2.4 – Опис утиліт

№ п/п	Назва утиліти	Опис застосування
1	PyCharm	Головне середовище розробки програмного забезпечення серверної частини курсової роботи.
2	VS Code	Середовище розробки програмного забезпечення клієнтської частини курсової роботи.
2	Postman	Програмне забезпечення необхідне для тестування REST запитів. Використовувалось для тестування API інтерфейсів, та клієнтських запитів.
3	DataGrip	Програмне забезпечення, яке надає легкий графічний інтерфейс для доступу до бази даних.

Таблиця 2.4 – Опис бібліотек

№ п/п	Назва утиліти	Опис застосування
1	Flask	Веб-фреймворк для побудови веб-додатків на Python.
2	asyncio	Бібліотека для асинхронного програмування в Python.
2	aiohttp	Клієнт та сервер для асинхронного HTTP-зв'язку.
4	Beautiful Soup (bs4)	Бібліотека для парсингу HTML і XML-документів.
3	pandas	Бібліотека для обробки та аналізу даних в Python.

4	numpy	Бібліотека для роботи з масивами та математичними функціями.
5	React	JavaScript-бібліотека з відкритим вихідним кодом для розробки інтерфейсів користувача.

2.4 Аналіз безпеки даних

Аналіз вразливостей програмного забезпечення є критичним аспектом для забезпечення стабільної роботи системи. Система повинна враховувати різні види загроз та реалізовувати захист від них:

- а) Забезпечення коректної обробки введених даних та параметризованих запитів для уникнення SQL-ін'єкцій.
- б) Екранування та очищення виводу, що виводиться на сторінках, для запобігання XSS-атакам.
- в) Налаштування обмежень доступу через HTTP-заголовки, щоб уникнути неправомірного доступу до ресурсів.
- г) Забезпечення надійного механізму аутентифікації та авторизації для контролю доступу до функціональності.

У контексті розробки веб-сервісу, що надає дані, їх безпека посідає чи не найперше місце. Ефективними методами захисту можуть стати:

- а) Використання протоколу HTTPS для захищеного обміну даними між клієнтом та сервером.
- б) Використання безпечних методів для передачі та збереження конфіденційної інформації.
- в) Встановлення систем моніторингу для виявлення та реагування на спроби несанкціонованого доступу або атак.
- г) Регулярне створення резервних копій даних для запобігання втрати інформації внаслідок випадкових або зловмисних подій.

Висновки до розділу

В результаті моделювання та конструювання програмного забезпечення було виявлено, що використання моделі BPMN стало ефективним інструментом для аналізу та моделювання бізнес-процесів. Використання такої стандартизованої мови сприяло кращому розумінню та візуалізації взаємодії між компонентами системи, що полегшило подальший аналіз та оптимізацію бізнес-процесів.

У побудові архітектури програмного забезпечення використовувалася методологія C4. Створення дворівневої діаграми C4 дозволило чітко визначити контекст системи, ідентифікувати контейнери, компоненти та взаємодії між ними. Це сприяло високому рівню абстракції та зрозумілості архітектурних рішень.

Процес конструювання програмного забезпечення включав побудову ER-діаграми та діаграми класів. Використання ER-діаграми дозволило чітко визначити сутності та їхні взаємозв'язки в базі даних. Діаграма класів була ефективним інструментом для представлення структури системи, опису взаємодій об'єктів та їхніх атрибутів.

Окремий акцент у розділі був зроблений на питання безпеки даних. Врахування принципів та впровадження відповідних заходів забезпечення дозволили створити систему, яка відповідає стандартам безпеки, таким як GDPR. Захист персональних даних та надійність системи стали пріоритетними завданнями під час проектування та конструювання програмного забезпечення.

3 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Аналіз якості ПЗ

Аналіз коду було проведено за допомогою статичного аналізатору коду Qodana від компанії JetBrains:

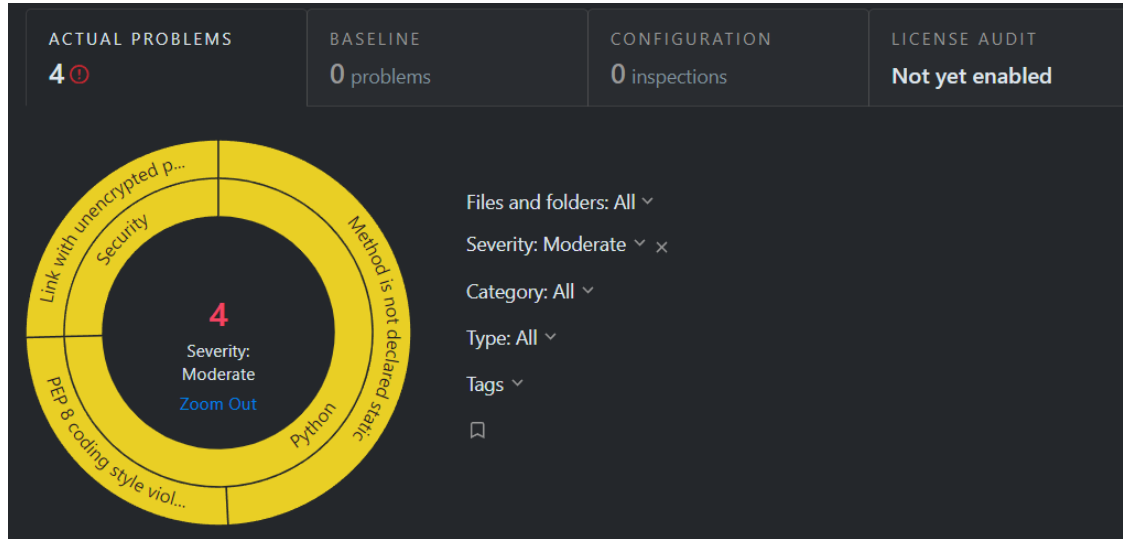


Рисунок 3.1 – Звіт якості коду від Qodana

В результаті аналізу виявлено чотири помилки з помірним рівнем важливості. Перші дві помилки пов'язані з можливістю визначення методів як статичних, що може впливати на коректність їх виклику та взаємодії з іншими елементами системи. Рекомендацією є уточнення призначення методів та їх виділення як статичних або нестатичних.

Також виявлено порушення стилю коду PEP 8, яке стосується використання некоректного форматування чи іменування елементів коду. Рекомендацією є адаптація кодової бази до встановлених стандартів оформлення.

Додатково, виявлено попередження про безпеку, пов'язане із невикористанням безпечних зв'язків для HTTP-посилань. Рекомендацією є переведення HTTP-посилань на протокол HTTPS для забезпечення безпеки передачі даних.

В цілому, результати аналізу не виявили якихось серйозних помилок з рівнем важливості вище середнього, що вказує на задовільну якість коду.

Рекомендації з усунення виявлених проблем допоможуть забезпечити стабільність, ефективність та безпеку розроблюваного програмного продукту.

3.2 Опис процесів тестування

Було виконане мануальне тестування програмного забезпечення, опис відповідних тестів наведено у таблицях 3.3 – 3.30.

Таблиця 3.1 – Тест 1

Тест	Реєстрація користувача
Модуль	Реєстрація користувача
Номер тесту	1.1
Початковий стан системи	Користувач знаходиться на сторінці реєстрації
Вхідні данні	Електронна пошта, пароль, підтвердження паролю
Опис проведення тесту	У відповідні поля вводяться: коректна електронна пошта, яка до цього не була зареєстрована в системі, пароль від 10 до 64 символів, який містить хоча б з одну англійську літеру, одне число і один спеціальний символ, , підтвердження паролю, яке співпадає з раніше введеним паролем. Після цього натискаємо кнопку “Зареєструватися”.
Очікуваний результат	Реєстрація проходить успішно, користувач додається у систему і перенаправляється на сторінку авторизації.
Фактичний результат	Реєстрація проходить успішно, користувач додається у систему і перенаправляється на сторінку авторизації.

Таблиця 3.2 – Тест 2

Тест	Авторизація користувача
------	-------------------------

Модуль	Авторизація користувача
Номер тесту	2
Початковий стан системи	Користувач знаходиться на сторінці авторизації
Вхідні данні	Електронна пошта, пароль
Опис проведення тесту	У відповідні поля вводяться: коректна зареєстрована електронна пошта. Після цього натискання на кнопку "Увійти".
Очікуваний результат	Успішна авторизація, користувач перенаправляється на головну сторінку.
Фактичний результат	Успішна авторизація, користувач перенаправляється на головну сторінку.

Таблиця 3.3 – Тест 3

Тест	Отримання API ключа
Модуль	Отримання API ключа
Номер тесту	3
Початковий стан системи	Користувач перебуває на головній сторінці веб-сервісу.
Вхідні данні	-
Опис проведення тесту	Користувач натискає на кнопку "Отримати API ключ". На екрані з'являється унікальний ключ, який прив'язано до облікового запису.
Очікуваний результат	Користувач успішно отримує унікальний API ключ.
Фактичний результат	Користувач успішно отримує унікальний API ключ.

Таблиця 3.4 – Тест 4

Тест	Отримання Поточної погоди
Модуль	Поточна погода
Номер тесту	4
Початковий стан системи	Користувач отримав API ключ.
Вхідні данні	Широта, довгота, API ключ
Опис проведення тесту	Користувач виконує запит на отримання поточної погоди за відповідним адресою /weather. Користувач додає у параметри координати місця, для якого він хоче отримати погоду (широта та довгота) та дійсний API ключ у відповідному полі запиту. Користувач відправляє запит до серверу.
Очікуваний результат	Користувач успішно отримує JSON-відповідь з даними.
Фактичний результат	Користувач успішно отримує JSON-відповідь з даними.

Таблиця 3.5 – Тест 5

Тест	Отримання Прогнозу погоди
Модуль	Прогноз погоди
Номер тесту	5
Початковий стан системи	Користувач отримав API ключ.
Вхідні данні	Широта, довгота, API ключ
Опис проведення тесту	Користувач виконує запит на отримання поточної погоди за відповідним адресою /forecast. Користувач додає у параметри координати місця, для якого він хоче

	отримати погоду (широта та довгота) та дійсний API ключ у відповідному полі запити. Користувач відправляє запит до серверу.
Очікуваний результат	Користувач успішно отримує JSON-відповідь з даними.
Фактичний результат	Користувач успішно отримує JSON-відповідь з даними.

Таблиця 3.6 – Тест 6

Тест	Введення некоректних координат
Модуль	Поточна погода
Номер тесту	6
Початковий стан системи	Користувач отримав API ключ.
Вхідні данні	Широта, довгота, API ключ
Опис проведення тесту	Користувач виконує запит на отримання поточної погоди за відповідним адресою /weather. Користувач додає у параметри неможливі координати місця, для якого він хоче отримати погоду (широта та довгота) та дійсний API ключ у відповідному полі запити. Користувач відправляє запит до серверу.
Очікуваний результат	Користувач отримує JSON-відповідь з інформацією про невалідні координати.
Фактичний результат	Користувач отримує JSON-відповідь з інформацією про невалідні координати.

Таблиця 3.7 – Тест 7

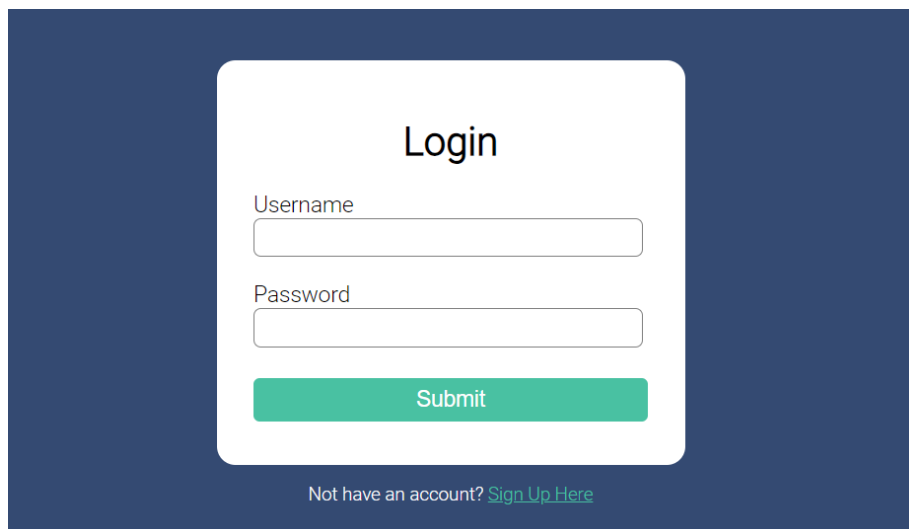
Тест	Введення неіснуючого API ключа
------	--------------------------------

Модуль	Поточна погода
Номер тесту	7
Початковий стан системи	-
Вхідні данні	Широта, довгота, API ключ
Опис проведення тесту	Користувач виконує запит на отримання поточної погоди за відповідним адресою /weather. Користувач додає у парметри координати місця, для якого він хоче отримати погоду (широта та довгота) та неіснуючий API ключ у відповідному полі запиту. Користувач натискає відправляє запит до серверу.
Очікуваний результат	Користувач отримує JSON-відповідь з інформацією про невалідний API ключ.
Фактичний результат	Користувач отримує JSON-відповідь з інформацією про невалідний API ключ.

3.3 Опис контрольного прикладу

Протестуємо увесь цикл роботи програмного забезпечення. Спочатку ми потрапляємо на сторінку авторизації:

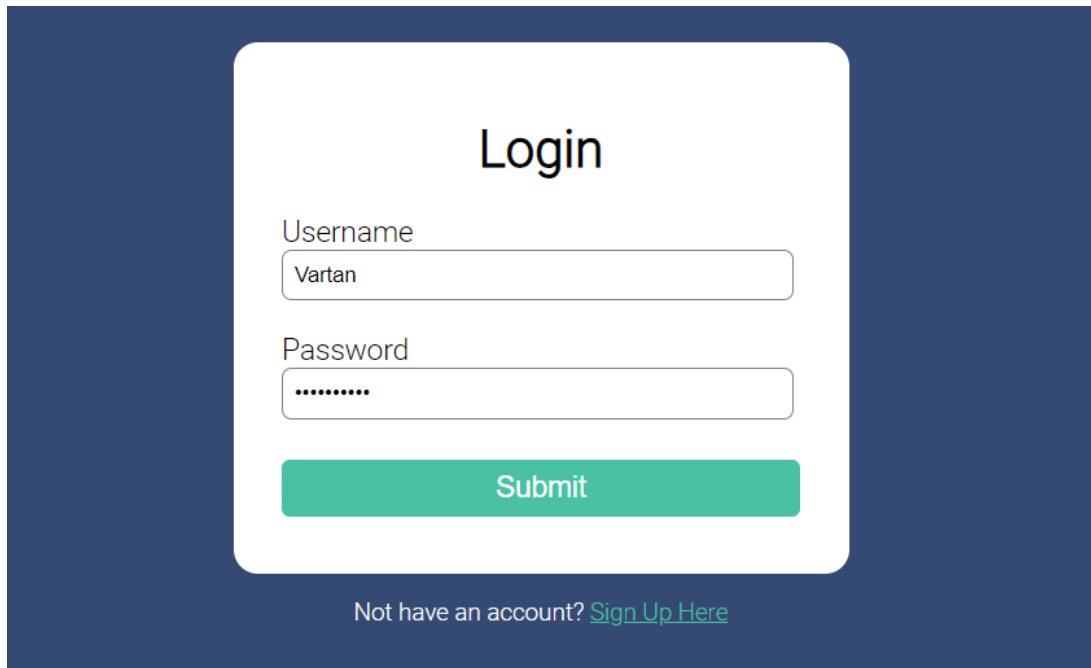
а) Авторизація



The image shows a login interface. It has a dark blue background. In the center is a white rounded rectangle containing the text 'Login'. Below this are two input fields: the first is labeled 'Username' and the second is labeled 'Password'. Below the input fields is a green button with the text 'Submit'. At the bottom of the white rectangle, there is a link that says 'Not have an account? Sign Up Here'.

Рисунок 3.2 – Авторизація

Введемо необхідні дані та відправимо форму:

A screenshot of a login form titled "Login" centered on a dark blue background. The form is a white rounded rectangle containing two input fields: "Username" with the text "Vartan" and "Password" with masked characters "*****". Below the fields is a green "Submit" button. At the bottom of the form, there is a link: "Not have an account? [Sign Up Here](#)".

Username

Vartan

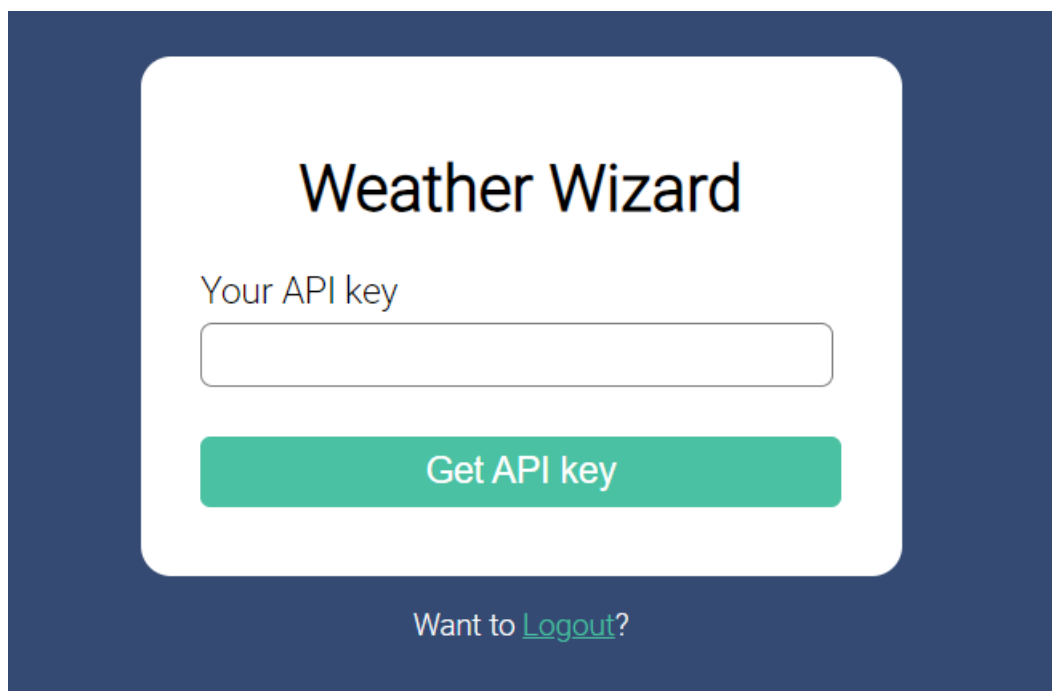
Password

Submit

Not have an account? [Sign Up Here](#)

Рисунок 3.3 – Спроба авторизації

б) Отримання API ключа

A screenshot of a form titled "Weather Wizard" centered on a dark blue background. The form is a white rounded rectangle containing a single input field labeled "Your API key". Below the field is a green "Get API key" button. At the bottom of the form, there is a link: "Want to [Logout?](#)".

Weather Wizard

Your API key

Get API key

Want to [Logout?](#)

Рисунок 3.4 – Сторінка отримання ключа

Натиснемо на кнопку “Отримати API ключ”:

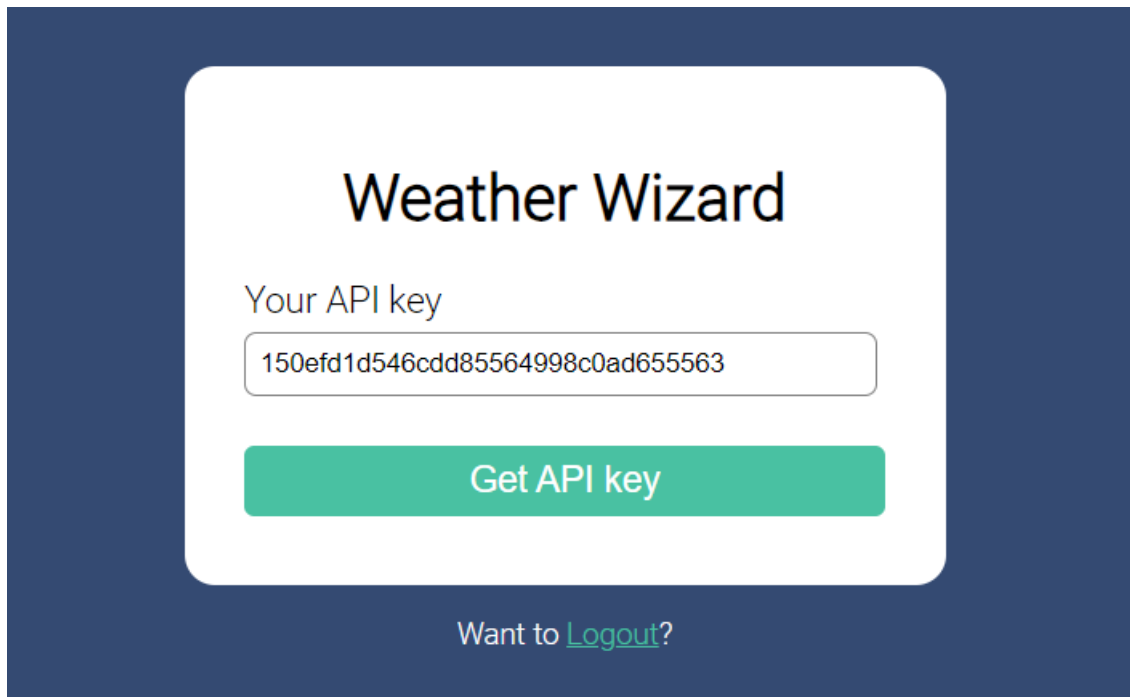


Рисунок 3.5 –Отримання ключа

в) Отримання поточної погоди

Тепер у нас є ключ і можна перейти до безпосередньо використання сервісу. Проведемо тестування за допомогою Postman – інструмента для розробки та тестування API. Він надає зручний інтерфейс для взаємодії з API, надсилаючи HTTP-запити та отримуючи HTTP-відповіді. Тестування поточної погоди:

а) Коректні вхідні дані

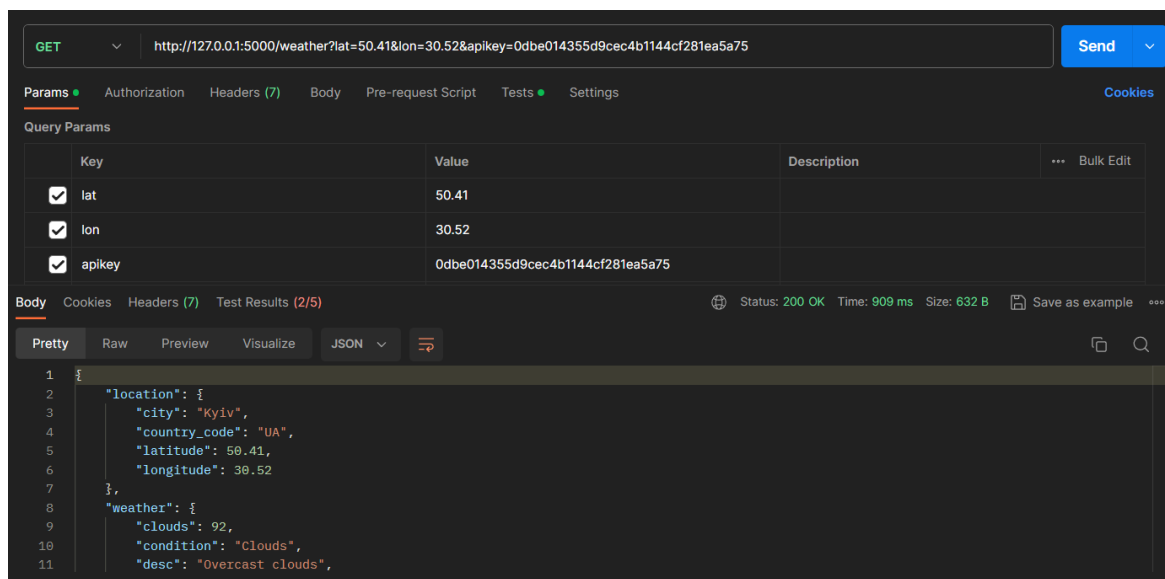


Рисунок 3.2 – Коректні вхідні дані

б) Некоректні координати

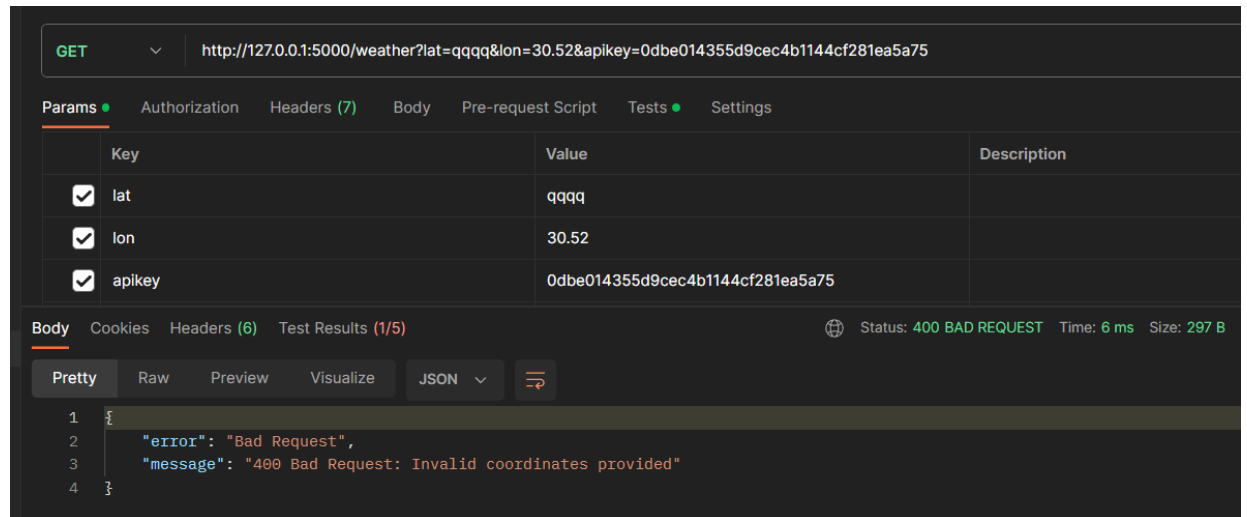


Рисунок 3.3 – Некоректні координати

в) Неіснуючі координати

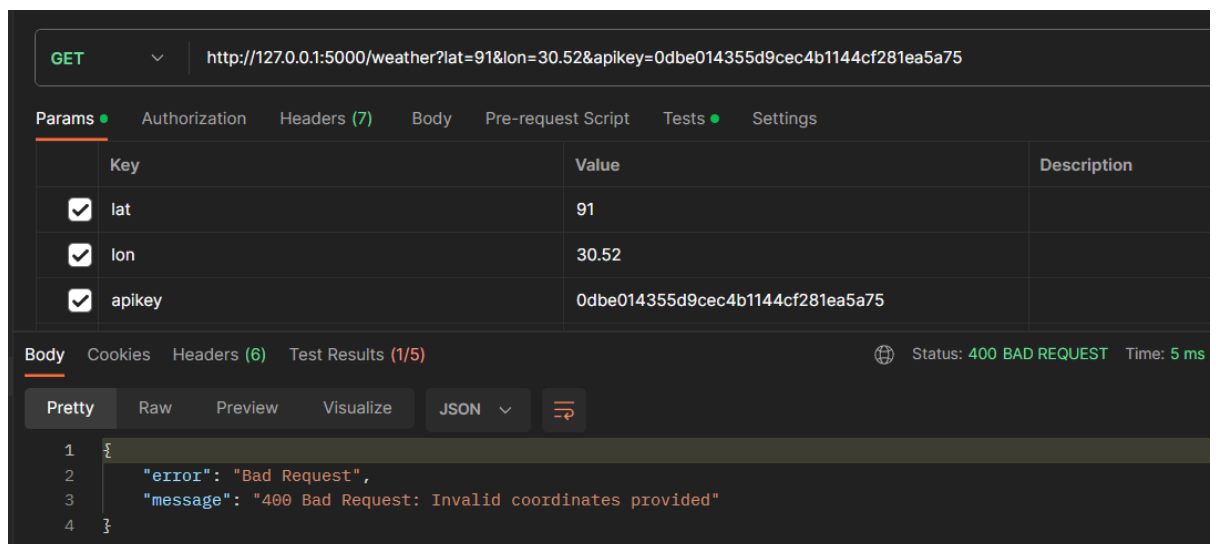


Рисунок 3.4 – Неіснуючі координати

г) Граничні значення координат

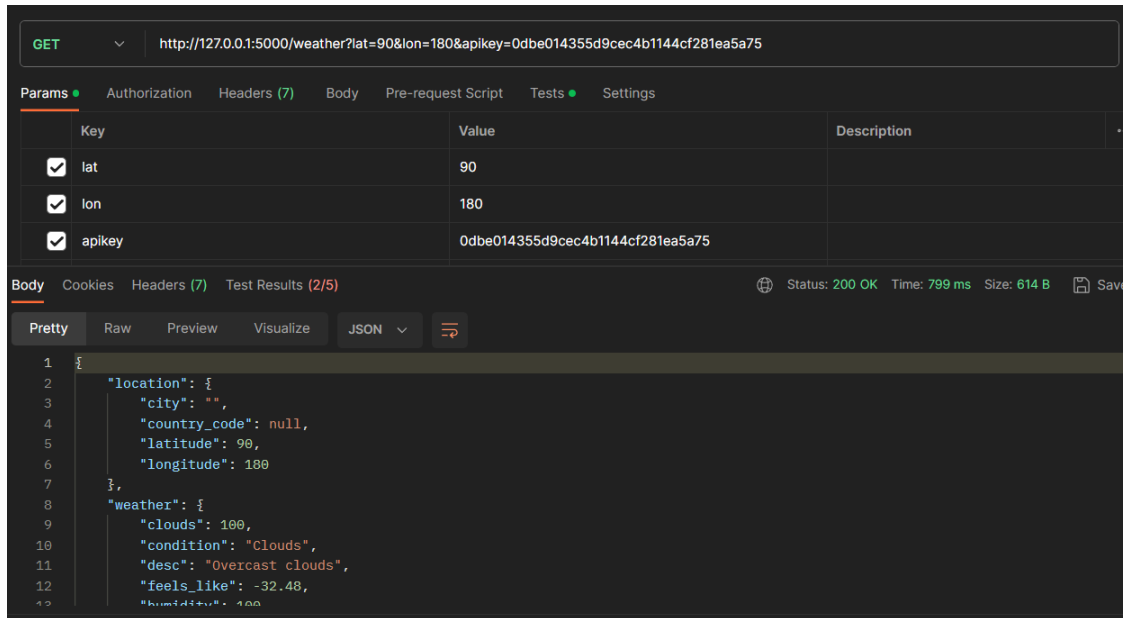


Рисунок 3.5 – Граничні значення координат

д) Неіснуючий API ключ

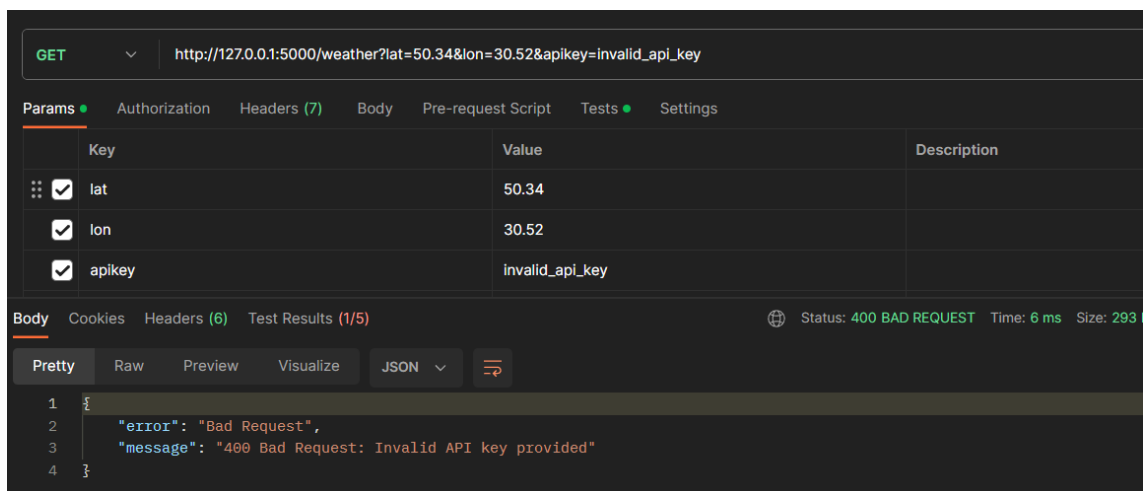


Рисунок 3.6 – Неіснуючий API ключ

г) Тестування отримання прогнозу погоди:

а) Прогноз погоди на вказану кількість днів

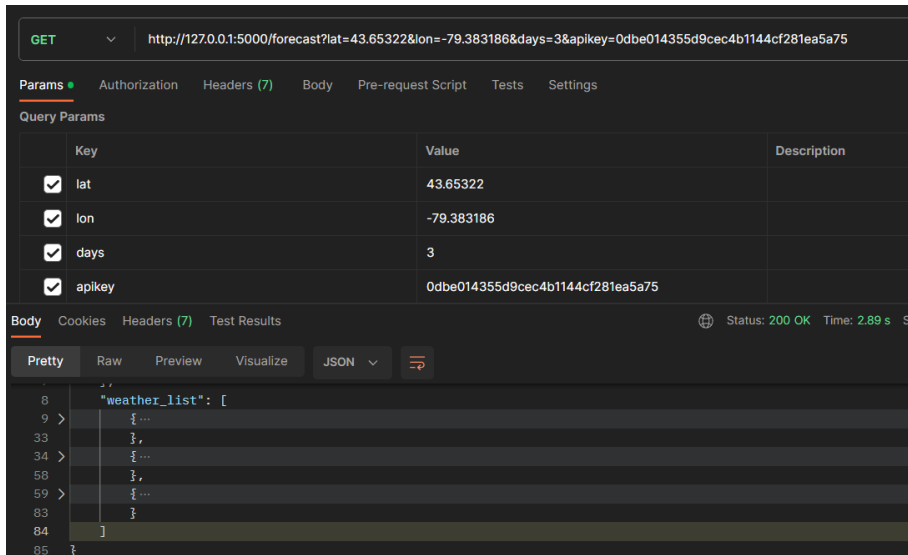


Рисунок 3.7 – Отримання прогнозу на 3 дні

б) Некоректна кількість днів

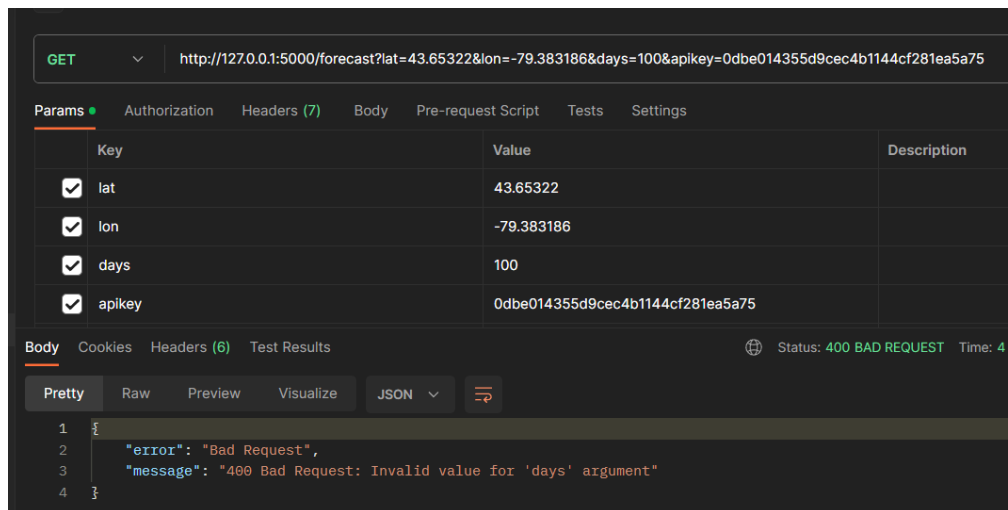


Рисунок 3.7 – Некоректна кількість днів

Тестування веб-застосунку:

а) Пошук за містом

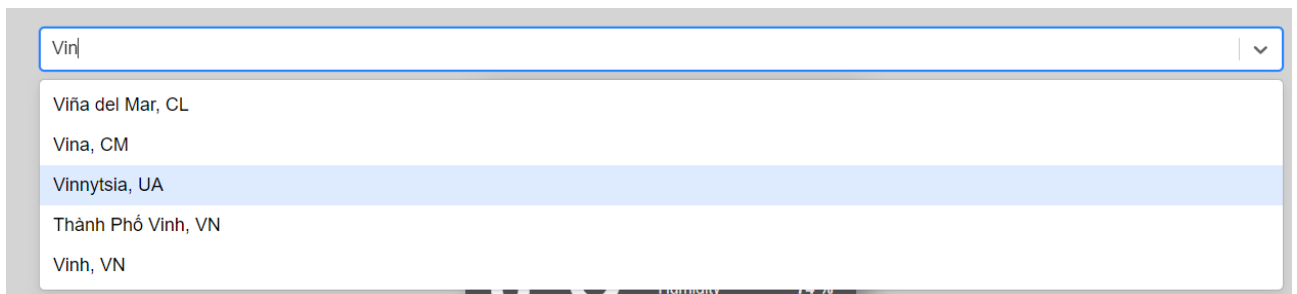


Рисунок 3.7 – Пошук за містом

б) Отримання погоди

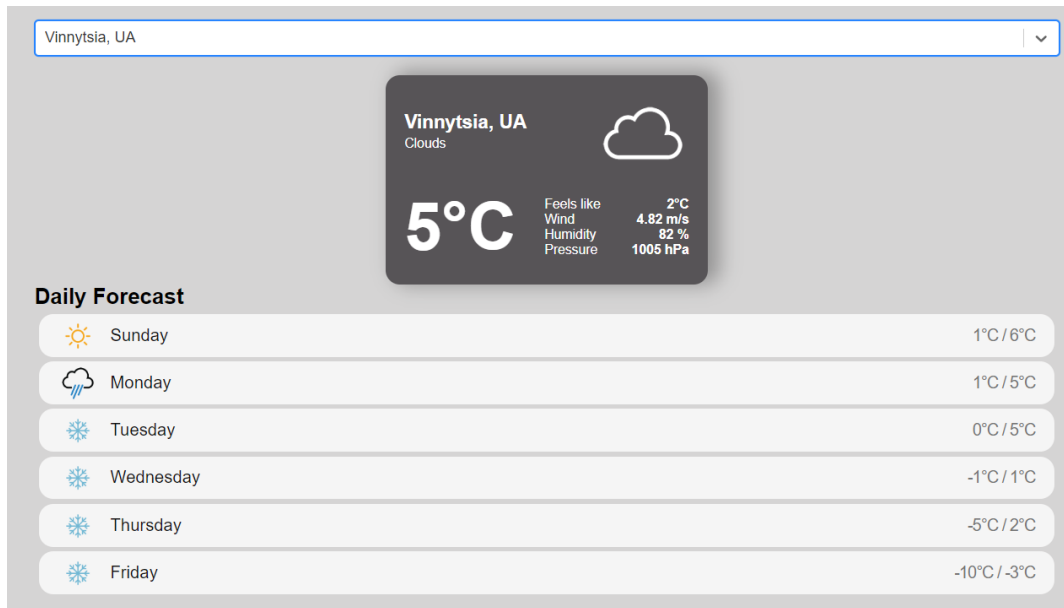


Рисунок 3.7 – Отримання погоди

в) Натиснувши на 1 з днів можна отримати детальнішу інформацію.

Подивимося деталі для вівторка:

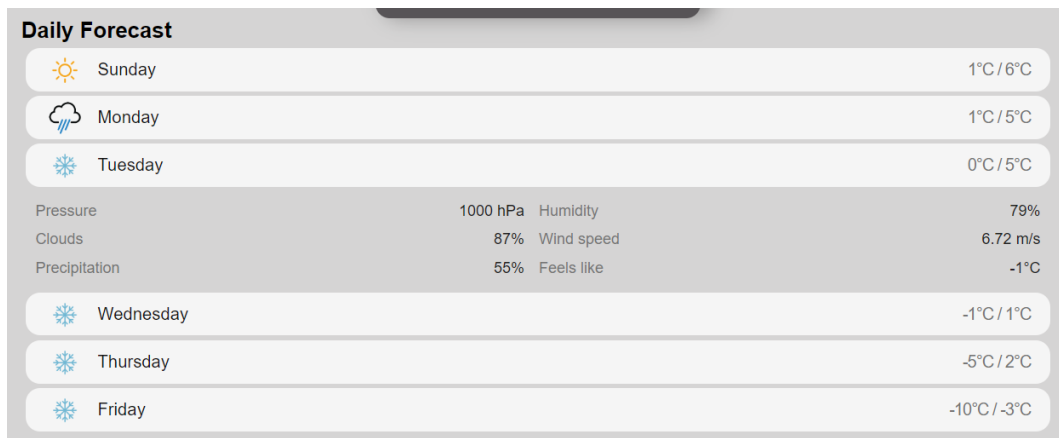


Рисунок 3.7 – Отримання деталей

Висновки до розділу

У даному розділі було проведено комплексний аналіз якості програмного забезпечення з використанням інструменту Qordana та розроблено низку тест-кейсів для оцінки різноманітних випадків використання системи.

Використання Qordana дозволило виявити ряд помилок та потенційних проблем у вихідному коді, серед яких зокрема виявлення методів, які можуть бути оголошені як 'static', а також порушення стилю коду згідно із PEP 8 та безпеки через використання незахищених HTTP посилань.

Тест-кейси були розроблені для різних сценаріїв використання, що дозволило визначити правильність функціональності системи та її відповідність вимогам та очікуванням користувачів. Описаний контрольний приклад, демонструючи роботу системи при реєстрації користувача, надав можливість перевірити коректність обробки даних та взаємодії з користувачем.

Таким чином, проведений аналіз та розроблені тест-кейси становлять важливий крок у забезпеченні якості програмного продукту та виявленні можливих вад та недоліків для їх подальшого виправлення та вдосконалення.

4 ВПРОВАДЖЕННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Розгортання програмного забезпечення

Розгортання сервера на Flask:

а) Підготовка середовища

Переконайтеся, що на сервері встановлено Python та інші залежності для Flask, такі як virtualenv.

б) Встановлення зовнішніх залежностей Flask

Створіть віртуальне середовище та встановіть Flask за допомогою наведеного нижче коду:

```
python -m venv venv  
venv/Script/activate  
pip install flask
```

в) Завантаження вихідного коду Flask

Склонуйте репозиторій або завантажте вихідний код Flask-додатку.

г) Запустіть сервер Flask:

```
python app.py
```

Розгортання застосунку React:

а) Підготовка середовища

Переконайтеся, що на сервері встановлено Node.js та npm.

б) Встановлення зовнішніх залежностей React

Перейдіть у каталог з додатком React та встановіть залежності, код наведено нижче:

```
cd react_app
```


`npm install`

в) Завантаження вихідного коду React

Склонуйте репозиторій або завантажте вихідний код застосунку React.

г) Налаштування конфігурації React

Налаштуйте файли конфігурації, такі як `.env`, згідно з вимогами вашого застосунку.

д) Запуск застосунку React

`npm start`

Налаштування MySQL:

а) Встановлення MySQL

Переконайтеся, що MySQL встановлено на вашому сервері або локальному середовищі розробки. Якщо його немає, завантажте та встановіть його з офіційного веб-сайту MySQL або використовуйте менеджер пакетів вашої операційної системи.

б) Запуск MySQL

Перейдіть у командний рядок Windows та виконайте наступну команду, якщо ви змінили шлях завантаження MySQL Server за замовчування, то вкажіть свій шлях:

```
C:\> "C:\Program Files\MySQL\MySQL Server 8.0\bin\mysqld"
```

в) Створення бази даних

Запустіть скрипт `create_db.sql`, який можна знайти у папці `sql_scripts` у репозиторії проекту

4.2 Підтримка програмного забезпечення

Оновлення програмного забезпечення (ПЗ) включає в себе кілька етапів. Ось загальна інструкція для оновлення:

а) Збережіть резервні копії

Перед оновленням важливо зробити резервні копії всіх важливих даних і конфігурацій системи. Це дозволяє вам відновити попередню версію, якщо виникнуть проблеми під час оновлення.

б) Перевірте вимоги до оновлення

Переконайтеся, що ви ознайомилися з вимогами до нової версії ПЗ і зрозуміли, які зміни будуть внесені.

в) Завантажте новий код з репозиторію

```
git pull origin master
```

г) Встановіть залежності, якщо вони змінилися

```
pip install -r requirements.txt
```

д) Застосуйте міграції бази даних

```
flask db upgrade
```

е) Зміни у конфігурації

Якщо нова версія вимагає змін у конфігурації, внесіть необхідні зміни у файли конфігурації.

Висновки до розділу

У першому підрозділі пов'язаним з розгортанням було детально розглянуто процес введення в експлуатацію розробленого програмного

забезпечення. В ході цього етапу були визначені кроки, необхідні для успішного розгортання системи.

У підрозділі “Підтримка програмного забезпечення” було розглянуто питання щодо подальшого супроводу та підтримки розробленого програмного продукту. Визначені стратегії регулярного моніторингу, аналізу та оновлення для забезпечення найвищої ефективності та безпеки системи.

Здійснення поетапного впровадження та уважне планування супроводу дозволять забезпечити плавний та успішний життєвий цикл програмного забезпечення, а систематичне тестування та моніторинг дозволять швидко реагувати на можливі проблеми та підтримувати високу якість продукту.

ВИСНОВКИ

У результаті виконання курсової роботи було спроектовано та реалізовано веб-сервіс прогнозування погоди, що забезпечує користувачів актуальною та достовірною інформацією. В якості середовища розробки обрано PyCharm, що дозволило зручно та ефективно вести розробку на мовах програмування Python та JavaScript. У якості системи управління базою даних використано MySQL, що надає надійне зберігання та обробку інформації для ефективної роботи веб-сервісу.

У процесі виконання роботи я застосував на практиці усі знання, отримані на дисципліні “Компоненти програмної інженерії”. Для вирішення поставлених цілей довелося познайомитися з різними цікавими технологіями, більшість з яких стали для мене новими. Я отримав досвід розробки веб-застосунку, роботи з різними API, застосування асинхронності для оптимізації, використання різних бібліотек Python.

Розробка даного застосунку виявилася трудомісткою та комплексною задачею, що включала в себе аналіз вимог, моделювання, конструювання, аналіз якості та впровадження програмного забезпечення. У результаті проведених досліджень та заходів було досягнуто визначених цілей та створено функціональний та надійний продукт.

Аналіз вимог до програмного забезпечення визначив ключові аспекти та напрямки розвитку веб-сервісу. Використані технології та інструменти були обрані на основі ретельного аналізу та порівнянь, що дало змогу сформулювати чіткі функціональні вимоги та необхідність захисту персональних даних.

Моделювання та конструювання програмного забезпечення відобразилися в ефективному використанні мови BPMN для аналізу бізнес-процесів та методології C4 для побудови архітектурних діаграм. Впроваджені безпечні практики та стандарти дозволили створити стабільну та надійну систему.

Аналіз якості та тестування програмного забезпечення був важливим етапом у забезпеченні функціональності та надійності системи. Виявлені

помилки та порушення стилів були виправлені, а розроблені тест-кейси дозволили покрити основні можливості та виявити можливі недоліки.

Впровадження та супровід програмного забезпечення були успішно здійснені, забезпечуючи стійку та ефективну роботу системи під час її життєвого циклу. Планове моніторинг та оновлення гарантують актуальність та безпеку продукту у подальшому.

Усі ці етапи спільно внесли свій внесок у створення високоякісного та функціонального веб-сервісу прогнозування погоди, що задовольняє вимоги користувачів та відповідає сучасним стандартам розробки та безпеки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

- 1) How do we measure the weather and climate? [Електронний ресурс] – <https://climate.ncsu.edu/learn/how-do-we-measure-the-weather-and-climate/>
- 2) Open Weather Map [Електронний ресурс] – <https://openweathermap.org/api>
- 3) Visual Crossing API [Електронний ресурс] – <https://www.visualcrossing.com/weather-api>
- 4) Sinoptik [Електронний ресурс] – <https://ua.sinoptik.ua/>
- 5) Python [Електронний ресурс] – <https://www.python.org/>
- 6) Flask [Електронний ресурс] – <https://flask.palletsprojects.com/en/3.0.x/>
- 7) Asyncio [Електронний ресурс] – <https://docs.python.org/uk/3/library/asyncio.html>
- 8) Beautiful soup [Електронний ресурс] – <https://beautiful-soup-4.readthedocs.io/en/latest/>
- 9) Numpy [Електронний ресурс] – <https://numpy.org/>
- 10) Pandas [Електронний ресурс] – <https://pandas.pydata.org/>
- 11) Pycharm [Електронний ресурс] – <https://www.jetbrains.com/pycharm/>
- 12) JavaScript [Електронний ресурс] – <https://developer.mozilla.org/en-US/docs/Web/JavaScript>
- 13) React [Електронний ресурс] – <https://react.dev/>
- 14) Meteomatics [Електронний ресурс] – <https://www.meteomatics.com/>
- 15) The Weather Channel [Електронний ресурс] – <https://weather.com/uk-UA/weather/today/1/50.43,30.52>

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

ДОДАТОК А

“ЗАТВЕРДЖЕНО”

Керівник роботи

_____ Ілля АХАЛАДЗЕ

“ ____ ” _____ 2024 р.

**ВЕБ-СЕРВІС ПРОГНОЗУВАННЯ ПОГОДИ, ШЛЯХОМ АГРЕГАЦІЇ ДАНИХ
ОНЛАЙН ГІДРОМЕТЦЕНТРІВ**

Технічне завдання

КПІ.ІП-1314.045440.01.91

“ПОГОДЖЕНО”

Керівник роботи:

Ілля АХАЛАДЗЕ

Виконавець:

Вартан КАРАМЯН

Київ – 2024

1	3
2	4
3	5
4	6
4.1	6
4.1.1	6
4.1.2	6
4.1.3	7
4.1.4	7
4.2	8
4.3	8
4.3.1	8
4.3.2	8
4.4	8
4.5	8
4.5.1	8
4.5.2	9
4.5.3	9
4.5.4	9
4.5.5	9
4.6	9
4.7	9
4.8	9
5	10
5.1	10
5.2	10
6	11
7	12

1 НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: Веб-сервіс прогнозування погоди, шляхом агрегації даних онлайн гідрометцентрів.

Галузь застосування: Сервіси прогнозування погоди.

Наведене технічне завдання поширюється на розробку веб-сервісу прогнозування погоди WeatherWizard, який використовується для отримання актуальних погодніх даних інтегруючись з онлайн гідрометцентрами та призначений для таких галузей застосування як туризм, сільське господарство, транспорт, енергетика, бізнес, безпека, споживчі додатки тощо.

2 ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки веб-сервісу прогнозування погоди є підвищення ефективності отримання даних гідрометцентрів.

3 ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для полегшення отримання актуальних погодних даних.

Метою розробки є підвищення точності отриманої інформації про поточну та майбутню погоду для полегшення прийняття рішень у бізнесі чи повсякденному житті.

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вимоги до функціональних характеристик

Програмне забезпечення повинно забезпечувати виконання наступних основних функцій:

4.1.1 Для користувача веб-сервісу:

- Можливість отримання поточної погоди;
- Можливість отримання прогнозу погоди до 7 днів;
- Можливість отримання погоди у форматі JSON;
- Можливість задання потрібних властивостей погоди та кількості днів за допомогою параметрів;
- Можливість отримання даних, що є агрегацією даних з трьох різних джерел;

4.1.2 Користувацького інтерфейсу веб-застосунку

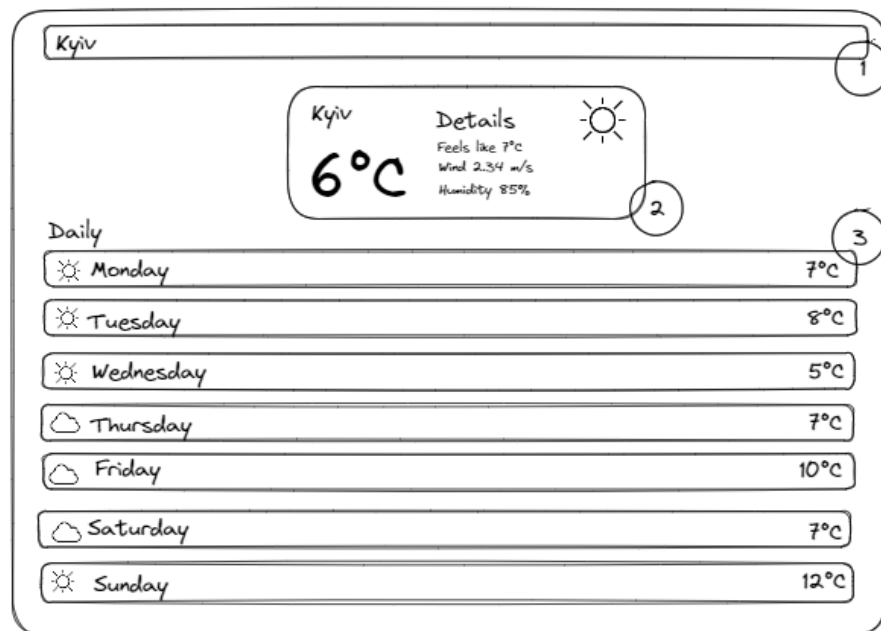


Рисунок 4.1 – Прототип сторінки

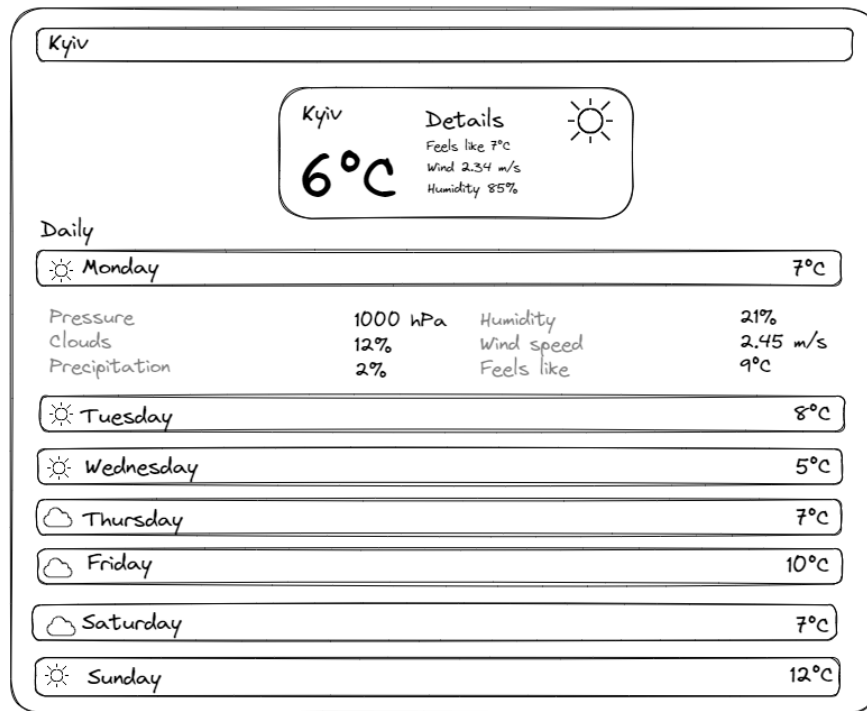


Рисунок 4.2 – Детальна інформація

4.1.3 Для користувача веб-застосунку:

- Можливість здійснювати пошук за містом (Рисунок 4.1 Елемент 1);
- Можливість отримання поточної погоди (Рисунок 4.1 Елемент 2);
- Можливість перегляду швидкої інформації про стан погоди у наступні 7 днів, а саме загальний стан погоди (сонячно, хмарно, дощ тощо) та температури повітря (Рисунок 4.1 Елемент 1);
- Можливість перегляду детальної інформації: атмосферний тиск, хмарність, вологість, швидкість вітру, ймовірність опадів та відчуття температури повітря (Рисунок 4.2).

4.1.4 Додаткові вимоги:

- Використати як джерело даних API гідрометцентру;
- Використати як джерело даних результат скрапінгу веб-сторінки гідрометцентру;
- Автозаповнення міст у полі для пошуку (Рисунок 4.1 Елемент 1);
- забезпечення сумісності з наступними браузерами: Google Chrome, Opera, Mozilla Firefox.

4.2 Вимоги до надійності

Передбачити контроль введення інформації та захист від некоректних дій користувача.

4.3 Умови експлуатації

Умови експлуатації згідно СанПін 2.2.2.542 – 96.

4.3.1 Вид обслуговування

Вимоги до виду обслуговування не висуваються.

4.3.2 Обслуговуючий персонал

Вимоги до обслуговуючого персоналу не висуваються.

4.4 Вимоги до складу і параметрів технічних засобів

Мінімальна конфігурація технічних засобів:

- тип процесору: Intel Core i5;
- об'єм ОЗП: 4 Гб;
- підключення до мережі Інтернет зі швидкістю від 20 мегабіт;

Рекомендована конфігурація технічних засобів < (та на якій виконувалась розробка) >:

- тип процесору: Intel Core i5;
- об'єм ОЗП: 16 Гб;
- підключення до мережі Інтернет зі швидкістю від 80 мегабіт;

4.5 Вимоги до інформаційної та програмної сумісності

Програмне забезпечення повинно працювати під управлінням операційних систем сімейства WIN32 (Windows'XP, Windows NT і т.д.) або Unix.

4.5.1 Вимоги до вхідних даних

Вимоги до вхідних даних не висуваються.

4.5.2 Вимоги до вихідних даних

Результати повинні бути представлені в наступному форматі: JSON.

4.5.3 Вимоги до мови розробки

- Розробку веб-сервісу виконати на мові програмування Python (Flask);
- Розробку веб-застосунку виконати на мові програмування JavaScript (React).

4.5.4 Вимоги до середовища розробки

Розробку виконати на платформах PyCharm Professional, Visual Studio Code.

4.5.5 Вимоги до представлення вихідних кодів

Вихідний код програми має бути представлений у вигляді завантаженого проекту на GitHub.

4.6 Вимоги до маркування та пакування

Вимоги до маркування та пакування не висуваються.

4.7 Вимоги до транспортування та зберігання

Вимоги до транспортування та зберігання не висуваються.

4.8 Спеціальні вимоги

Спеціальні вимоги до продукту не висуваються.

5 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

5.1 Попередній склад програмної документації

У склад супроводжувальної документації повинні входити наступні документи на аркушах формату А4:

- пояснювальна записка;
- технічне завдання;
- керівництво користувача;
- програма та методика тестування;
- текст програми.

Графічна частина повинна бути виконана на аркушах формату А3 та містити наступні документи:

- схема структурна варіантів використання;
- схема структурна компонент;
- схема структурна класів програмного забезпечення;
- креслення вигляду екранних форм.

5.2 Спеціальні вимоги до програмної документації

Програмні модулі, котрі розробляються, повинні бути задокументовані, тобто тексти програм повинні містити всі необхідні коментарі.

6 СТАДІЇ І ЕТАПИ РОЗРОБКИ

7

№	Назва етапу	Строк	Звітність
1.	Вивчення літератури за тематикою роботи	12.10	
2.	Розробка технічного завдання	19.10	Технічне завдання
3.	Аналіз вимог та уточнення специфікацій	26.10	Специфікації програмного забезпечення
4.	Проектування структури програмного забезпечення, проектування компонентів	02.11	Схема структурна програмного забезпечення та специфікація компонентів (діаграма класів, схема алгоритму)
5.	Програмна реалізація програмного забезпечення	09.11	Тексти програмного забезпечення
6.	Тестування програмного забезпечення	09.12	Тести, результати тестування
7.	Розробка матеріалів текстової частини роботи	23.12	Пояснювальна записка
8.	Розробка матеріалів графічної частини роботи	25.12	Графічний матеріал проекту
9.	Оформлення технічної документації роботи	30.12	Технічна документація

8 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

Тестування розробленого програмного продукту виконується відповідно до “Програми та методики тестування”.

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

ДОДАТОК Б

“ЗАТВЕРДЖЕНО”

Керівник роботи

_____ Максим ГОЛОВЧЕНКО

“ ” _____ 2023 р.

**ВЕБ-СЕРВІС ПРОГНОЗУВАННЯ ПОГОДИ, ШЛЯХОМ АГРЕГАЦІЇ
ДАНИХ ОНЛАЙН ГІДРОМЕТЦЕНТРІВ**

Текст програми

КПІ. ІП-1314.045440.03.12

“ПОГОДЖЕНО”

Керівник роботи:

_____ Максим ГОЛОВЧЕНКО

Консультант:

_____ Ілля АХАЛАДЗЕ

Виконавець:

_____ Вартан КАРАМЯН

Київ – 2023

app.py

```
from flask import Flask, render_template, request, redirect, url_for, session
from flask_cors import CORS
from flask_mysqldb import MySQL
from flask_bcrypt import Bcrypt
from flask_wrapper import FlaskAppWrapper
from views.weather.weather_handler import WeatherHandler
from views.login.login_handler import LoginHandler
from views.api_key.api_key_handler import ApiKeyHandler
from config import APP_SECRET_KEY
```

```
# Flask app
```

```
app_wrapper = FlaskAppWrapper(Flask(__name__))
```

```
# Add configs
```

```
app_wrapper.configs(MYSQL_HOST='localhost',
                    MYSQL_USER='Vartan',
                    MYSQL_PASSWORD='vartan2004',
                    MYSQL_DB='weatherwizard',
                    SECRET_KEY=APP_SECRET_KEY)
```

```
# Database
```

```
mysql = MySQL(app_wrapper.app)
```

```
# Password hashing
```

```
bcrypt = Bcrypt(app_wrapper.app)
```

```
# Allow cors
```

```
CORS(app_wrapper.app)
```

```
# Authorisation
```

```
app_wrapper.add_endpoint('/', 'index', LoginHandler.index, methods=['GET'])
app_wrapper.add_endpoint('/login', 'login', LoginHandler.login, methods=['GET',
'POST'])
app_wrapper.add_endpoint('/logout', 'logout', LoginHandler.logout,
methods=['GET', 'POST'])
app_wrapper.add_endpoint('/register', 'register', LoginHandler.register,
methods=['GET', 'POST'])
```

```
# Getting API key
```

```
app_wrapper.add_endpoint('/apikey', 'apikey', ApiKeyHandler.api_key,
methods=['GET', 'POST'])
```

```
# Getting Weather
```

```
app_wrapper.add_endpoint('/weather', 'weather', WeatherHandler.current,
methods=['GET'])
app_wrapper.add_endpoint('/forecast', 'forecast', WeatherHandler.forecast,
methods=['GET'])
```

```
if __name__ == '__main__':
    app_wrapper.run(debug=True)
```

flask_wrapper.py

```
from handlers.error_handler import ErrorHandler
```

```
class FlaskAppWrapper:
```

```
    def __init__(self, app, **configs):
        self.app = app
        self.configs(**configs)
```

```

self.error_handler = ErrorHandler(app)

def configs(self, **configs):
    for config, value in configs.items():
        self.app.config[config.upper()] = value

def add_endpoint(self, endpoint=None, endpoint_name=None, handler=None,
methods=None, *args, **kwargs):
    self.app.add_url_rule(endpoint, endpoint_name, handler, methods=methods,
*args, **kwargs)

def run(self, **kwargs):
    self.app.run(**kwargs)

```

error_handler.py

```

from flask import jsonify

```

```

class ErrorHandler:
    def __init__(self, app):
        self.app = app

    # Register error handlers
    self.app.register_error_handler(400, self.handle_bad_request)
    self.app.register_error_handler(500, self.handle_server_error)

    def handle_bad_request(self, error):
        response = jsonify({'error': 'Bad Request', 'message': str(error)})
        response.status_code = 400
        return response

```

```

def handle_server_error(self, error):
    response = jsonify({'error': 'Server Error', 'message': str(error)})
    response.status_code = 500
    return response

```

login_handler.py

```

import re
from flask import render_template, url_for, redirect, request
import MySQLdb.cursors

```

```

class LoginHandler:

```

```

    @staticmethod

```

```

    def index():

```

```

        return redirect(url_for('login'))

```

```

    @staticmethod

```

```

    def login():

```

```

        msg = "

```

```

        print(request.form)

```

```

        if request.method == 'POST' and 'username' in request.form and 'password' in
request.form:

```

```

            from app import mysql, bcrypt, session

```

```

            username = request.form['username']

```

```

            password = request.form['password']

```

```

            cursor = mysql.connection.cursor(MySQLdb.cursors.DictCursor)

```

```

        cursor.execute('SELECT * FROM users WHERE username = %s',
[username])

        account = cursor.fetchone()

        if account:

            if bcrypt.check_password_hash(account['password'], password):

                session['logged'] = True
                session['id'] = account['id']
                session['username'] = account['username']
                print(session)
                return redirect(url_for('apikey'))
            else:
                msg = 'Invalid password'

        else:
            msg = 'Invalid username'

    return render_template('login.html', msg=msg)

@staticmethod
def logout():
    from app import mysql, bcrypt, session
    session.pop('logged', None)
    session.pop('id', None)
    session.pop('username', None)
    print(session)
    return redirect(url_for('login'))

```



```

@staticmethod
def register():
    msg = "
    print(request.form)

    if request.method == 'POST' and 'username' in request.form and 'password' in
request.form:

        from app import mysql, bcrypt, session
        username = request.form['username']
        password = request.form['password']
        confirm_password = request.form['confirm_password']

        cursor = mysql.connection.cursor(MySQLdb.cursors.DictCursor)
        cursor.execute('SELECT * FROM users WHERE username = %s',
(username,))
        account = cursor.fetchone()
        if account:
            msg = 'Account already exists!'
        elif not username or not password or not confirm_password:
            msg = 'Please fill out the form !'
        elif not re.match(r'[A-Za-z0-9]+', username):
            msg = 'Username must contain only characters and numbers!'
        elif len(password) < 8:
            msg = 'Password must be at least 8 characters long!'
        elif not re.search("[a-z]", password) or not re.search("[A-Z]", password):
            msg = 'Password must contain both lowercase and uppercase letters!'
        elif not re.search("[0-9]", password):
            msg = 'Password must contain at least one digit!'
        elif password != confirm_password:
            msg = 'Passwords do not match!'
        else:

```

```

        hashed_password = bcrypt.generate_password_hash(password)
        cursor.execute('INSERT INTO users VALUES (NULL, %s, %s)',
(username, hashed_password))
        mysql.connection.commit()
        msg = 'You have successfully registered !'

        return redirect(url_for('login'))

elif request.method == 'POST':
    msg = 'Please fill out the form!'

    return render_template('register.html', msg=msg)

```

api_key_handler.py

```

import MySQLdb.cursors
import secrets
from flask import render_template, url_for, redirect

class ApiKeyHandler:
    @classmethod
    def api_key(cls):
        from app import session, mysql, request

        print(session)
        if session.get('logged'):
            print("User logged")
            if request.method == 'POST':

```

```

        cursor = mysql.connection.cursor(MySQLdb.cursors.DictCursor)
        user_id = session['id']

        cursor.execute('SELECT api_key FROM api_keys WHERE user_id =
%s', (user_id,))
        existing_api_key_data = cursor.fetchone()

        # get api key from db
        if existing_api_key_data:
            api_key = existing_api_key_data['api_key']

        # generate new api key
        else:
            api_key = cls.generate_api_key()

        cursor.execute('INSERT INTO api_keys (user_id, api_key) VALUES
(%s, %s)', (user_id, api_key))
        mysql.connection.commit()

        return render_template('api_key.html', api_key=api_key)
    else:
        return render_template('api_key.html')

    else:
        print("User didnt log")
        return redirect(url_for('login'))

    @staticmethod
    def generate_api_key(length=32):
        return secrets.token_hex(length // 2)

```

weather_handler.py

```
from flask import jsonify, request, abort
from data_aggregation.current_weather import CurrentWeather
from data_aggregation.forecast import Forecast
from handlers.validation import Validator
```

```
class WeatherHandler:
```

```
    @staticmethod
```

```
    def hello_world():
```

```
        return "<h1>Weather Wizard API</h1>"
```

```
    @staticmethod
```

```
    def current():
```

```
        try:
```

```
            lat, lon = Validator.validate_coords(
                request.args.get('lat'),
                request.args.get('lon'))
```

```
        except ValueError as error:
```

```
            abort(400, error)
```

```
        current_weather = CurrentWeather(lat, lon)
```

```
        data = current_weather.get()
```

```
        if data is not None:
```

```
            response = jsonify(data)
```

```
            response.headers.add('Content-Type', 'application/json')
```

```
            response.headers.add('Access-Control-Allow-Origin', '*')
```

```
            return response
```

```

else:
    abort(500, f"Cannot find weather by given coordinates: lat={lat},
lon={lon}")

@staticmethod
def forecast():
    try:
        lat, lon = Validator.validate_coords(
            request.args.get('lat'),
            request.args.get('lon'))

        days = Validator.validate_days(
            request.args.get('days'))
    except ValueError as error:
        abort(400, error)

    forecast_handler = Forecast(lat, lon, days)
    data = forecast_handler.get()

    if data is not None:
        response = jsonify(data)
        response.headers.add('Content-Type', 'application/json')
        response.headers.add('Access-Control-Allow-Origin', '*')
        return response
    else:
        abort(500, f"Cannot find weather by given coordinates: lat={lat},
lon={lon}")

```

validator.py

```
import re
```

```
from config import MIN_DAYS_FORECAST, MAX_DAYS_FORECAST
```

```
class Validator:
```

```
    @classmethod
```

```
    def validate_coords(cls, lat, lon):
```

```
        if lat is None or lon is None:
```

```
            raise ValueError("Please enter 'lat' and 'lon' arguments")
```

```
        if cls.__is_valid_lat(lat) and cls.__is_valid_lon(lon):
```

```
            return float(lat), float(lon)
```

```
        else:
```

```
            raise ValueError("Invalid coordinates provided")
```

```
    @staticmethod
```

```
    def __is_valid_lat(lat):
```

```
        pattern = r'^(\+|-)?(?:90(?:\.\d{1,6})?)|(?:[0-9][1-8][0-9])(?:\.\d{0-9}{1,6})?)$'
```

```
        return bool(re.match(pattern, lat))
```

```
    @staticmethod
```

```
    def __is_valid_lon(lon):
```

```
        pattern = r'^(\+|-)?(?:180(?:\.\d{1,6})?)|(?:[0-9][1-9][0-9]1[0-7][0-9])(?:\.\d{0-9}{1,6})?)$'
```

```
        return bool(re.match(pattern, lon))
```

```
    @classmethod
```

```
    def validate_days(cls, days):
```

```
        if days is None:
```

```
            return MAX_DAYS_FORECAST
```

```

else:
    try:
        number = int(days)
        if MIN_DAYS_FORECAST <= number <= MAX_DAYS_FORECAST:
            return number
        else:
            raise ValueError
    except ValueError:
        raise ValueError("Invalid value for 'days' argument")

```

current_weather.py

```

import asyncio
import numpy as np
from data_sources.open_weather_api.open_weather import OpenWeatherAPI
from data_sources.visual_crossing.visual_crossing import VisualCrossingAPI
from data_sources.sinoptik.sinoptik_web_scraper import SinoptikWebScraper
import time
import json
from config import city_coordinates

```

```

class CurrentWeather:
    def __init__(self, lat, lon):
        self.lat = lat
        self.lon = lon

    def get(self):
        data = asyncio.run(self.__get_data())

        # remove None sources from list

```

```

if None in data:
    data = [source for source in data if source is not None]

# return None if no weather
if len(data) == 0:
    return None

weather = {
    "time": f"/".join([source['weather']['time'] for source in data]),
    "condition": data[0]["weather"].get("condition"),
    "desc": data[0]["weather"].get("desc"),
    "icon": data[0]["weather"].get("icon"),

    "temp": self.__aggregate_prop(data, "temp", 2), # temp in Celsius
    "feels_like": self.__aggregate_prop(data, "feels_like", 2), # feels_like temp
in Celsius

    "pressure": self.__aggregate_prop(data, "pressure"), # pressure in hPa
    "humidity": self.__aggregate_prop(data, "humidity"), # humidity in %
    "wind_speed": self.__aggregate_prop(data, "wind_speed", 2), # wind
speed in m/s

    "clouds": self.__aggregate_prop(data, "clouds"), # clouds in %
    "precip_prob": self.__aggregate_prop(data, "precip_prob", ), #
precipitation probability in %
}

aggregated_weather_data = {
    "location": data[0]["location"],
    "weather": weather
}

```



```

with open("data/aggregated_data.json", "w") as file:
    json.dump(aggregated_weather_data, file, indent=4, ensure_ascii=False)

return aggregated_weather_data

async def __get_data(self):
    weather1 = OpenWeatherAPI(self.lat, self.lon)
    weather2 = VisualCrossingAPI(self.lat, self.lon)
    weather3 = SinoptikWebScraper(self.lat, self.lon)

    tasks = [asyncio.create_task(weather1.current_weather()),
              asyncio.create_task(weather2.current_weather()),
              asyncio.create_task(weather3.current_weather())]

    results = await asyncio.gather(*tasks)

    return results

    @staticmethod
    def __aggregate_prop(data, prop, round_number=0):
        agg_data = [source["weather"].get(prop) for source in data if source is not
None]

        if any(agg_data):
            aggregated_prop = np.mean([data for data in agg_data if data is not None])
            return round(aggregated_prop, round_number) if round_number else
round(aggregated_prop)
forecast.py
import asyncio

```

```

import json
import time
import numpy as np

from config import city_coordinates
from data_sources.open_weather_api.open_weather import OpenWeatherAPI
from data_sources.sinoptik.sinoptik_web_scraper import SinoptikWebScraper
from data_sources.visual_crossing.visual_crossing import VisualCrossingAPI

class Forecast:

    def __init__(self, lat, lon, days):
        self.lat = lat
        self.lon = lon
        self.days = days

    def get(self):
        data = asyncio.run(self.__get_data())

        # remove None sources from list
        if None in data:
            data = [source for source in data if source is not None]

        # return None if no weather
        if len(data) == 0:
            return None

        weather_list = []
        temp_keys = list(data[0]["weather_list"][0]["temp"].keys())
        feels_like_keys = list(data[0]["weather_list"][0]["feels_like"].keys())

```

```

for i in range(self.days):
    temp = {
        day_time: self.__aggregate_temp(data, i, "temp", day_time) for day_time
in temp_keys
    }
    feels_like = {
        day_time: self.__aggregate_temp(data, i, "feels_like", day_time) for
day_time in feels_like_keys
    }

    weather = {
        "date": f"/".join([source['weather_list'][i]['date'] for source in data]),
        "condition": data[0]["weather_list"][i].get("condition"),
        "desc": data[0]["weather_list"][i].get("desc"),
        "icon": data[0]["weather_list"][i].get("icon"),
        "temp": temp,
        "feels_like": feels_like,
        "pressure": self.__aggregate_prop(data, i, "pressure"),
        "humidity": self.__aggregate_prop(data, i, "humidity"),
        "wind_speed": self.__aggregate_prop(data, i, "wind_speed", 2),
        "clouds": self.__aggregate_prop(data, i, "clouds"),
        "precip_prob": self.__aggregate_prop(data, i, "precip_prob")
    }

    weather_list.append(weather)

aggregated_weather_data = {
    "location": data[0]["location"],
    "weather_list": weather_list

```

```
}
```

```
with open("data/aggregated_forecast.json", "w") as file:
```

```
    json.dump(aggregated_weather_data, file, indent=4, ensure_ascii=False)
```

```
return aggregated_weather_data
```

```
async def __get_data(self):
```

```
    weather1 = OpenWeatherAPI(self.lat, self.lon)
```

```
    weather2 = VisualCrossingAPI(self.lat, self.lon)
```

```
    weather3 = SinoptikWebScraper(self.lat, self.lon)
```

```
    tasks = [asyncio.create_task(weather1.forecast(self.days)),
```

```
              asyncio.create_task(weather2.forecast(self.days)),
```

```
              asyncio.create_task(weather3.forecast(self.days))]
```

```
    results = await asyncio.gather(*tasks)
```

```
    return results
```

```
@staticmethod
```

```
def __aggregate_temp(data, i, type_, day_time):
```

```
    agg_data = [source["weather_list"][i][type_][day_time] for source in data if  
source is not None]
```

```
    if any(agg_data):
```

```
        agg_prop = np.mean(agg_data)
```

```
        return round(agg_prop, 2)
```

```
@staticmethod
```

```
def __aggregate_prop(data, i, prop, round_number=0):
```

```
agg_data = [source["weather_list"][i].get(prop) for source in data if source is not None]
```

```
if any(agg_data):
```

```
    agg_prop = np.mean([data for data in agg_data if data is not None])
```

```
    return round(agg_prop, round_number) if round_number else
```

```
round(agg_prop)
```

open_weather.py

```
import asyncio
```

```
import time
```

```
import json
```

```
from config import OPEN_WEATHER_API_KEY, OPEN_WEATHER_URL, city_coordinates
```

```
from datetime import datetime
```

```
from aiohttp import ClientSession
```

```
class OpenWeatherAPI:
```

```
    def __init__(self, latitude, longitude):
```

```
        self.base_url = OPEN_WEATHER_URL
```

```
        self.params = {
```

```
            "lat": latitude,
```

```
            "lon": longitude,
```

```
            "appid": OPEN_WEATHER_API_KEY,
```

```
            "units": "metric"
```

```
        }
```

```
        self.api_data = None
```

```
    async def current_weather(self):
```

```
        async with ClientSession() as session:
```

```

url = f"{self.base_url}/weather"

async with session.get(url, params=self.params) as response:
    try:
        if response.status == 200:
            self.api_data = await response.json()

            current_weather = {
                "location": self.__get_current_location(),
                "weather": self.__get_current_weather()
            }

            self.write_to_json(current_weather, "ow_cur")

            return current_weather

        else:
            raise Exception(f"bad request with status: {response.status}")

    except Exception as error:
        print(f"Open Weather API weather data fetching error: {error}")

def __get_current_weather(self):
    weather = {
        # weather time
        "time": str(datetime.fromtimestamp(self.api_data["dt"]).time()),
        # weather description
        "condition": self.api_data['weather'][0]['main'],
        "desc": self.api_data['weather'][0]['description'].capitalize(),
        "icon": self.api_data['weather'][0]['icon'],
    }

```

```

        # temperature in Celsius
        "temp": self.api_data['main']['temp'],
        "feels_like": self.api_data['main']['feels_like'],

        # other properties
        "pressure": self.api_data['main']['pressure'],
        "humidity": self.api_data['main']['humidity'],
        "clouds": self.api_data['clouds']['all'],
        "wind_speed": self.api_data['wind']['speed']
    }

    return weather

def __get_current_location(self):
    return {
        "city": self.api_data.get('name'),
        "country_code": self.api_data['sys'].get('country'),
        "latitude": self.api_data["coord"]["lat"],
        "longitude": self.api_data["coord"]["lon"]
    }

async def forecast(self, number_days):
    async with ClientSession() as session:
        url = f"{self.base_url}/forecast/daily"
        self.params["cnt"] = number_days

    async with session.get(url, params=self.params) as response:
        try:
            if response.status == 200:

```

```

        self.api_data = await response.json()

        forecast = {
            "location": self.__get_forecast_location(),
            "weather_list": self.__get_forecast_weather_list()
        }

        self.write_to_json(forecast, "ow_forecast")

        return forecast

    else:

        raise Exception(f"bad request with status: {response.status}")

except Exception as error:

    print(f"Open Weather API forecast fetching error: {error}")

def __get_forecast_weather_list(self):
    weather_list = []
    for day in self.api_data["list"]:
        weather = {
            "date": str(datetime.fromtimestamp(day['dt']).date()),
            'condition': day['weather'][0]['main'],
            'desc': day['weather'][0]['description'].capitalize(),
            'icon': day['weather'][0]['icon'],
            'temp': day['temp'],
            'feels_like': day['feels_like'],
            'pressure': day['pressure'],
            'humidity': day['humidity'],
            'clouds': day['clouds'],
            'wind_speed': day['speed'], # m/s

```



```

        'precip_prob': float(day['pop']) * 100 # percent
    }

    weather_list.append(weather)

return weather_list

def __get_forecast_location(self):
    return {
        "city": self.api_data['city'].get('name'),
        "country_code": self.api_data['city'].get('country'),
        "latitude": self.api_data['city']["coord"]["lat"],
        "longitude": self.api_data['city']["coord"]["lon"]
    }

    @staticmethod
    def write_to_json(data, file_name):
        with open(f"data/{file_name}.json", "w") as file:
            json.dump(data, file, indent=4, ensure_ascii=False)

    async def main():
        start_time = time.time()

        weather_handler = OpenWeatherAPI(1, 1)
        # data = visual_crossing.forecast(7)

        task = asyncio.create_task(weather_handler.current_weather())

        await task

```

```
print(f"Time: {(time.time() - start_time)}")
```

```
if __name__ == "__main__":  
    asyncio.run(main())
```

visual_crossing.py

```
import asyncio  
import json  
import time  
from aiohttp import ClientSession  
from datetime import datetime, timedelta  
from config import VISUAL_CROSSING_API_KEY,  
VISUAL_CROSSING_URL  
  
class VisualCrossingAPI:  
    def __init__(self, latitude, longitude):  
        self.url = f"{VISUAL_CROSSING_URL}/{latitude},{longitude}"  
        self.params = {  
            'key': VISUAL_CROSSING_API_KEY,  
            'unitGroup': 'metric',  
        }  
        self.api_data = None  
  
    async def current_weather(self):  
        async with ClientSession() as session:  
            current_datetime = datetime.now()  
            date = current_datetime.strftime("%Y-%m-%dT%H:%M:%S")
```

```

self.params['include'] = "current"

    async with session.get(f"{self.url}/{date}", params=self.params) as
response:

    try:
        if response.status == 200:
            self.api_data = await response.json()

            current_weather = {
                "location": self.__get_current_location(),
                "weather": self.__get_current_weather()
            }

            self.write_to_json(current_weather, "vc_cur")

            return current_weather
        else:
            raise Exception(f"bad request with status code {response.status}")

    except Exception as error:
        print(f"Visual Crossing API weather data fetching error: {error}")

def __get_current_location(self):
    return {
        'timezone_city': self.api_data['timezone'],
        'latitude': self.api_data['latitude'],
        'longitude': self.api_data['longitude'],
    }

```

```

def __get_current_weather(self):
    return {
        'time':
            f'{self.api_data['currentConditions']['datetime']}',
        'desc': self.api_data['currentConditions']['conditions'],
        'temp': self.api_data['currentConditions']['temp'],
        'feels_like': self.api_data['currentConditions']['feelslike'],
        'pressure': self.api_data['currentConditions']['pressure'],
        'humidity': self.api_data['currentConditions']['humidity'],
        'clouds': self.api_data['currentConditions']['cloudcover'],
        'wind_speed': round(self.api_data['currentConditions']['windspeed'] / 3.6,
2), # m/s
        'precip_prob': self.api_data['currentConditions']['precipprob'] # Probability
of precipitation
    }

async def forecast(self, number_days):
    async with ClientSession() as session:
        current_datetime = datetime.now()
        forecast_datetime = current_datetime + timedelta(days=number_days - 1)

        date_1 = current_datetime.strftime("%Y-%m-%dT%H:%M:%S")
        date_2 = forecast_datetime.strftime("%Y-%m-%dT%H:%M:%S")

        async with session.get(f"{self.url}/{date_1}/{date_2}",
params=self.params) as response:

            try:
                if response.status == 200:

```

```

        self.api_data = await response.json()

        forecast = {
            "location": self.__get_current_location(),
            "weather_list": self.__get_weather_list()
        }
        self.write_to_json(forecast, 'vc_forecast')

    return forecast

    else:

        raise Exception(f"bad request with status code {response.status}")

    except Exception as error:

        print(f"Visual Crossing API forecast data fetching error: {error}")

def __get_weather_list(self):
    weather_list = []
    for day in self.api_data["days"]:
        weather = {
            'date': day['datetime'],
            'condition': day['conditions'],
            "temp": {
                "night": day['hours'][2]['temp'],
                "morn": day['hours'][8]['temp'],
                "day": day['hours'][14]['temp'],
                "eve": day['hours'][20]['temp'],
                "min": day['tempmin'],
                "max": day['tempmax']
            },

```

```

        "feels_like": {
            "night": day['hours'][2]['feelslike'],
            "morn": day['hours'][8]['feelslike'],
            "day": day['hours'][14]['feelslike'],
            "eve": day['hours'][20]['feelslike'],
        },
        'pressure': day['pressure'],
        'humidity': day['humidity'],
        'clouds': day['cloudcover'],
        'wind_speed': round(day['windspeed'] / 3.6, 2),
        'precip_prob': day['precipprob']
    }

    weather_list.append(weather)

return weather_list

@staticmethod
def write_to_json(data, file_name):
    with open(f"data/{file_name}.json", "w") as file:
        json.dump(data, file, indent=4, ensure_ascii=False)

async def main():
    start_time = time.time()

    lat, lon = city_coordinates['Vlad']

    visual_crossing = VisualCrossingAPI(lat, lon)
    # data = visual_crossing.forecast(7)

```

```

task = asyncio.create_task(visual_crossing.forecast(7))

await task

# current_datetime = datetime.now()
# date = current_datetime.strftime("%Y-%m-%dT%H:%M:%S")
# print(current_datetime + timedelta(days=2))
print(f"Time: {(time.time() - start_time)}")

if __name__ == '__main__':
    from config import city_coordinates

    asyncio.run(main())

```

sinoptik_web_scraper.py

```

import asyncio
import json
import re
import time
import requests
import numpy as np
import pandas as pd
from datetime import timedelta, date
from io import StringIO
from aiohttp import ClientSession
from bs4 import BeautifulSoup

```

```
from config import OPEN_WEATHER_API_KEY, USER_AGENT,
SINOPTIK_URL, city_coordinates
```

```
class SinoptikWebScraper:
```

```
    def __init__(self, latitude, longitude):
```

```
        self.url = SINOPTIK_URL
```

```
        self.latitude = latitude
```

```
        self.longitude = longitude
```

```
        self.city_name = self.__get_city_name()
```

```
        self.soup = None
```

```
    def __get_city_name(self):
```

```
        geocode_url = "http://api.openweathermap.org/geo/1.0/reverse"
```

```
        params = {
```

```
            "lat": self.latitude,
```

```
            "lon": self.longitude,
```

```
            "appid": OPEN_WEATHER_API_KEY,
```

```
            "limit": 1
```

```
        }
```

```
        response = requests.get(geocode_url, params=params)
```

```
        try:
```

```
            if response.status_code == 200:
```

```
                data = response.json()
```

```
                ru_city = data[0]["local_names"]["ru"].strip().lower().replace(' ', '-')
```

```
                print(f"ru: <{ru_city}>")
```

```
                return ru_city
```



```

else:
    raise Exception(f"bad request with status code {response.status_code}")

except Exception as error:
    print(f"Geocoding API error! Response will be without Sinoptik.ua data:
{error}")

async def current_weather(self):
    async with ClientSession() as session:
        if not self.city_name:
            return None

        url = f"{self.url}/погода-{self.city_name}"
        headers = {
            'User-Agent': USER_AGENT}

        async with session.get(url, headers=headers) as response:
            try:
                if response.status == 200:
                    data = await response.text()

                    self.soup = BeautifulSoup(data, "html.parser")

                    current_weather = {
                        "location": self.__get_current_location(),
                        "weather": self.__get_current_weather(),
                    }

                    self.write_to_json(current_weather, 'sin_cur')

```

```

        return current_weather

    else:
        raise Exception(f"bad request with status code {response.status}")

    except Exception as error:
        print(f"Sinoptik weather data fetching error: {error}")

def __get_current_location(self):
    location = self.soup.find(class_="cityName cityNameShort")

    return {
        "city": location.find("h1").text.strip(),
        "region": location.find(class_="currentRegion").text.strip()
    }

def __get_current_weather(self) -> dict:
    current_elements = self.soup.find_all(class_="cur")
    precip_prob = current_elements[7].text

    return {
        "time": current_elements[0].text.replace(' ', ''),
        "condition": current_elements[1].find('div').get("title"),
        "temp": int(self.__strip_temp(current_elements[2].text)),
        "feels_like": int(self.__strip_temp(current_elements[3].text)),
        "pressure": round(int(current_elements[4].text) * 1.333, 2), # to hPa
        (millibars)
        "humidity": int(current_elements[5].text),
        "wind_speed": float(current_elements[6].text),
        "precip_prob": 0 if precip_prob == '-' else int(precip_prob)
    }

```

```

    }

    async def forecast(self, number_days: int):
        if not self.city_name:
            return None

        try:
            tasks = []
            for i in range(number_days):
                weather_date = (date.today() + timedelta(days=i)).strftime("%Y-%m-%d")

            tasks.append(asyncio.create_task(self.get_forecat_weather(weather_date)))

            tasks.append(asyncio.create_task(self.__get_main_page_forecast_data()))

            # get async results
            results = await asyncio.gather(*tasks)

            # get location and min/max temps
            main_page_data = results.pop()

            # add min/max temps to result
            for i in range(number_days):
                results[i]["temp"]["min"] = main_page_data["temp"]["min"][i]
                results[i]["temp"]["max"] = main_page_data["temp"]["max"][i]
                results[i]["condition"] = main_page_data["conditions"][i]

            forecast = {
                "location": main_page_data["location"],
                "weather_list": results
            }

```

```

    }

    self.write_to_json(forecast, "sin_forecast")
    return forecast

except Exception as error:
    print(f"Sinoptik forecast data fetching error: {error}")

async def get_forecat_weather(self, weather_date: str) -> dict:
    async with ClientSession() as session:
        url = f"{self.url}/погода-{self.city_name}/{weather_date}"
        headers = {
            'User-Agent': USER_AGENT}

        async with session.get(url, headers=headers) as response:
            if response.status == 200:

                html_source = await response.text()
                data = pd.read_html(StringIO(html_source))[0]

                weather = {
                    "date": weather_date,
                    "temp": {
                        "night": self.__strip_temp(data["ночь"][2]),
                        "morn": self.__strip_temp(data["утро"][2]),
                        "day": self.__strip_temp(data["день"][2]),
                        "eve": self.__strip_temp(data["вечер"][2]),
                    },
                    "feels_like": {
                        "night": self.__strip_temp(data["ночь"][3]),

```

```

        "morn": self.__strip_temp(data["утро"][3]),
        "day": self.__strip_temp(data["день"][3]),
        "eve": self.__strip_temp(data["вечер"][3]),
    },
    "pressure": round(data.loc[4].astype(float).mean() * 1.333, 2),
    "humidity": round(data.loc[5].astype(float).mean(), 2),
    "wind_speed": round(data.loc[6].astype(float).mean(), 2),
    "precip_prob": round(np.mean([int(precip) if precip != '-' else 0 for
precip in data.loc[7]]))
    }

    return weather
else:
    raise Exception(f"bad request to {url} with status code {response}")

async def __get_main_page_forecast_data(self):
    async with ClientSession() as session:
        url = f"{self.url}/погода-{self.city_name}"
        headers = {
            'User-Agent': USER_AGENT}

        async with session.get(url, headers=headers) as response:

            if response.status == 200:

                html_source = await response.text()
                self.soup = BeautifulSoup(html_source, "html.parser")

                return {
                    "conditions": self.__get_conditions(),

```

```

        "temp": {
            'min': self.__get_temps("min"),
            'max': self.__get_temps("max"),
        },
        "location": self.__get_locations()
    }

    else:

        raise Exception(f"bad request to {url} with status code {response}")

def __get_conditions(self):
    tabs = self.soup.find("div", class_="tabs").find_all("div",
class_="weatherIco")
    conditions = [tab.get("title") for tab in tabs]
    return conditions

def __get_temps(self, min_or_max):
    divs = self.soup.find_all("div", class_=min_or_max)
    temps = [self.__strip_temp(div.find("span").text) for div in divs]
    return temps

def __get_locations(self):
    location = self.soup.find(class_="cityName cityNameShort")

    return {
        "city": location.find("h1").text.strip(),
        "region": location.find(class_="currentRegion").text.strip()
    }

@staticmethod

```

```
def __strip_temp(temp_str: str):
    matches = re.findall(r'[-+]?\\d+°', temp_str)
    return int(matches[0][:1]) if matches else None
```

api_key.html

```
<!DOCTYPE html>
<html lang="en">
<head>
<title>Login</title>
<link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}" />
<link
href="https://fonts.googleapis.com/css2?family=Roboto:wght@300&display=swa
p"
rel="stylesheet"
/>
</head>
<body>
<div class="container">
<div class="form-box">
<h1>Weather Wizard</h1>
<form method="post">
<label for="api-key" class="apikey-label">Your API key</label>
<input type="text" id="api-key" name="api-key" value="{{ api_key
}}" />

<button class="submit-button">Get API key</button>

<p class="message">{{ msg }}</p>
```

```

        </form>
    </div>
    <div class="alt-form">
        <p>Want to <a href="{ { url_for('logout') }}"
class="btn">Logout</a>?</p>
    </div>

```

```

</div>
</body>
</html>

```

login.html

```

<!DOCTYPE html>
<html lang="en">
<head>
    <title>Login</title>
    <link rel="stylesheet" href="{ { url_for('static', filename='style.css') }}" />
    <link

```

```

href="https://fonts.googleapis.com/css2?family=Roboto:wght@300&display=swa
p"

```

```

    rel="stylesheet"
    />
</head>
<body>
    <div class="container">
        <div class="form-box">
            <h1>Login</h1>
            <form action="{ { url_for('login') }}" method="post">

```



```

        <label for="username">Username</label>
        <input type="text" id="username" name="username" placeholder=""
/>

        <label for="password">Password</label>
        <input type="password" id="password" name="password"
placeholder="" />

        <button class="submit-button">Submit</button>

        <p class="message">{{ msg }}</p>
    </form>
</div>
<div class="alt-form">
    <p>Not have an account? <a href="{{ url_for('register') }}">Sign Up
Here</a></p>
</div>

</div>
</body>
</html>

```

register.html

```

<!DOCTYPE html>
<html lang="en">
<head>
    <title>Sign Up | By Code Info</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}" />
    <link

```

```

href="https://fonts.googleapis.com/css2?family=Roboto:wght@300&display=swa
p"
    rel="stylesheet"
/>
</head>
<body>
    <div class="container">
        <div class="form-box">
            <h1>Sign Up</h1>
            <form action="{ { url_for('register') } }" method="post">
                <label for="username">Username</label>
                <input id="username" name="username" type="text" placeholder="" />

                <label for="password">Password</label>
                <input id="password" name="password" type="password" placeholder=""
/>

                <label for="confirm_password">Confirm Password</label>
                <input id="confirm_password" name="confirm_password"
type="password" placeholder="" />

                <button class="submit-button">Submit</button>

                <p class="message">{ { msg } }</p>

            </form>
        </div>
        <div class="alt-form">

```

```
<p>Already have an account? <a href="{ { url_for('login') }}">Login
here</a></p>
</div>
</div>
</body>
</html>
```

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

ДОДАТОК В

“ЗАТВЕРДЖЕНО”

Керівник роботи

Ілля АХАЛДЗЕ

“ ” _____ 2024 р.

**ВЕБ-СЕРВІС ПРОГНОЗУВАННЯ ПОГОДИ, ШЛЯХОМ АГРЕГАЦІЇ
ДАНИХ ОНЛАЙН ГІДРОМЕТЦЕНТРІВ**

Програма та методика тестування

КПІ. ІП-1314.045440.04.51

“ПОГОДЖЕНО”

Керівник роботи:

_____ Ілля АХАЛАДЗЕ

Виконавець:

_____ Вартан КАРАМЯН

Київ – 2023

ЗМІСТ

<u>1</u>	<u>ОБ’ЄКТ ВИПРОБУВАНЬ</u>	3
<u>2</u>	<u>МЕТА ТЕСТУВАННЯ</u>	4
<u>3</u>	<u>МЕТОДИ ТЕСТУВАННЯ</u>	5
<u>4</u>	<u>ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ</u>	6

5 ОБ'ЄКТ ВИПРОБУВАНЬ

Об'єктом випробування є веб-сервіс прогнозування погоди, шляхом агрегації даних онлайн гідрометцентрів.

6 МЕТА ТЕСТУВАННЯ

Мета тестування веб-сервісу полягає в забезпеченні високої якості та надійності програмного продукту перед впровадженням його в експлуатацію.

7 МЕТОДИ ТЕСТУВАННЯ

Для тестування програмного забезпечення використовуються такі методи:

- мануальне тестування – тестування без використання автоматизації, тест-кейси пише особа, що тестує програмне забезпечення;

8 ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Тестування виконується мануально з використанням наскрізного тестування та написанням unit-тестів, з метою знаходження помилок та недоліків як у функціональній частині програмного забезпечення так і в зручності користування.

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

ДОДАТОК Г

“ЗАТВЕРДЖЕНО”

Керівник роботи

_____ Ілля АХАЛАДЗЕ

“ ” _____ 2024 р.

**ВЕБ-СЕРВІС ПРОГНОЗУВАННЯ ПОГОДИ, ШЛЯХОМ АГРЕГАЦІЇ
ДАНИХ ОНЛАЙН ГІДРОМЕТЦЕНТРІВ**

Керівництво користувача

КП.ІП-1314.045440.05.34

“ПОГОДЖЕНО”

Керівник роботи:

_____ Ілля АХАЛАДЗЕ

Виконавець:

_____ Вартан КАРАМЯН

Київ – 2024

ЗМІСТ

<u>1</u>	<u>ПРИЗНАЧЕННЯ ПРОГРАМИ</u>	3
<u>2</u>	<u>ПІДГОТОВКА ДО РОБОТИ З ПРОГРАМНИМ ЗАБЕЗПЕЧЕННЯМ</u>	4
<u>2.1</u>	<u>Системні вимоги для коректної роботи</u>	4
<u>2.2</u>	<u>Завантаження застосунку</u>	4
<u>2.3</u>	<u>Перевірка коректної роботи</u>	6
<u>3</u>	<u>ВИКОНАННЯ ПРОГРАМИ</u>	7

1 ПРИЗНАЧЕННЯ ПРОГРАМИ

Веб-сервісом для прогнозування погоди, який надає користувачам актуальну та точну інформацію про погодні умови. Включає у себе реєстрацію та авторизацію користувачів, можливість отримання поточної погоди та прогнозу за вказаним періодом. Реалізовано з використанням технологій Python (Flask) для бекенду та JavaScript (React) для фронтенду.

Кінцева збірка програмного забезпечення включає в себе всі необхідні файли та компоненти для успішного встановлення та запуску веб-сервісу. Зокрема, вона містить виконуваний файл для запуску серверної частини, а також усі ресурси та залежності, необхідні для роботи фронтенду. Кінцева збірка готова до розгортання на сервері або локальному середовищі.

Репозиторій містить вихідний код програмного забезпечення, ресурси для його розробки та тестування. Він включає в себе папки для серверної та клієнтської частин, конфігураційні файли, тести, документацію та інші необхідні елементи. Репозиторій може бути структурованим згідно з кращими практиками організації коду та може використовувати систему контролю версій, таку як Git.

2 ПІДГОТОВКА ДО РОБОТИ З ПРОГРАМНИМ ЗАБЕЗПЕЧЕННЯМ

2.1 Системні вимоги для коректної роботи

Для успішної роботи даного застосунку необхідне виконання наступних вимог:

- ОС: Windows 10 (64 Bit);
- об'єм ОЗП: 4 Гб;
- процесор: Intel Core i3;
- для встановлення гри на персональному комп'ютері повинно бути не менше 200 МБ вільної пам'яті.

2.2 Завантаження застосунку

Розгортання сервера на Flask:

д) Підготовка середовища

Переконайтеся, що на сервері встановлено Python та інші залежності для Flask, такі як virtualenv.

е) Встановлення зовнішніх залежностей Flask

Створіть віртуальне середовище та встановіть Flask за допомогою наведеного нижче коду:

```
python -m venv venv  
venv/Script/activate  
pip install flask
```

ж) Завантаження вихідного коду Flask

Склонуйте репозиторій або завантажте вихідний код Flask-додатку.

з) Запустіть сервер Flask:

```
python app.py
```

Розгортання застосунку React:

е) Підготовка середовища

Переконайтеся, що на сервері встановлено Node.js та npm.

ж) Встановлення зовнішніх залежностей React

Перейдіть у каталог з додатком React та встановіть залежності, код наведено нижче:

```
cd react_app
```

```
npm install
```

з) Завантаження вихідного коду React

Склонуйте репозиторій або завантажте вихідний код застосунку React.

и) Налаштування конфігурації React

Налаштуйте файли конфігурації, такі як .env, згідно з вимогами вашого застосунку.

к) Запуск застосунку React

```
npm start
```

Налаштування MySQL:

а) Встановлення MySQL

Переконайтеся, що MySQL встановлено на вашому сервері або локальному середовищі розробки. Якщо його немає, завантажте та встановіть його з офіційного веб-сайту MySQL або використовуйте менеджер пакетів вашої операційної системи.

б) Запуск MySQL

Перейдіть у командний рядок Windows та виконайте наступну команду, якщо ви змінили шлях завантаження MySQL Server за замовчування, то вкажіть свій шлях:

```
C:\> "C:\Program Files\MySQL\MySQL Server 8.0\bin\mysqld"
```

в) Створення бази даних

Запустіть скрипт `create_db.sql`, який можна знайти у папці `sql_scripts` у репозиторії проекту

2.3 Перевірка коректної роботи

Для забезпечення коректної роботи потрібно виконати процес установки на чистому середовищі для перевірки, чи він виконується успішно без помилок.

Перевірити, чи всі компоненти програмного забезпечення встановлено правильно.

Запустити програмне забезпечення після встановлення та перевірити, чи воно коректно працює без помилок.

Переконайтеся, що усі функціональності доступні та виконуються належним чином.

перевірити, чи всі конфігураційні файли правильно встановлено та застосовано під час установки. Переконайтеся, що налаштування, такі як з'єднання з базою даних або інші параметри, правильно налаштовані.

Переконайтеся, що всі залежності, необхідні для роботи програмного забезпечення, встановлені та сумісні.

3 ВИКОНАННЯ ПРОГРАМИ

Виконання веб-сервісу:

Відкриваючи веб-сервіс потрапляємо на сторінку авторизації:

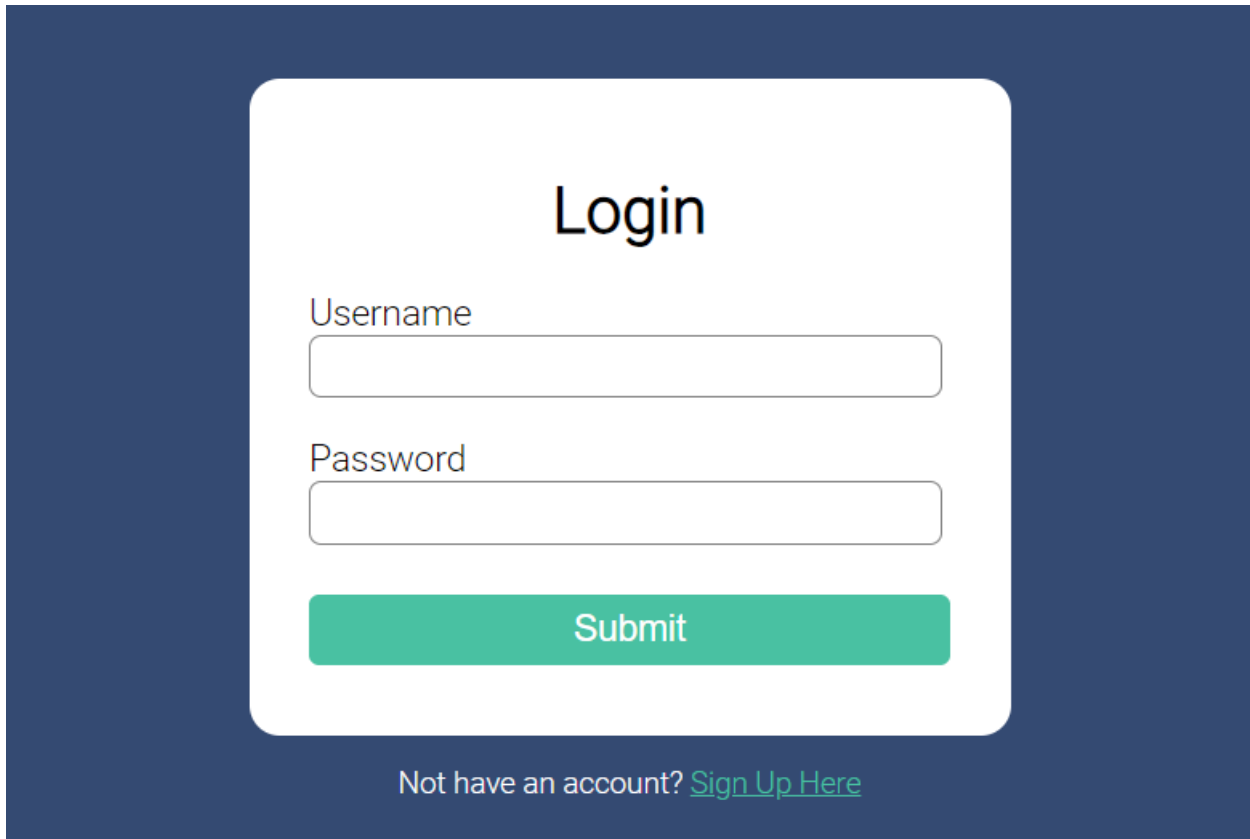
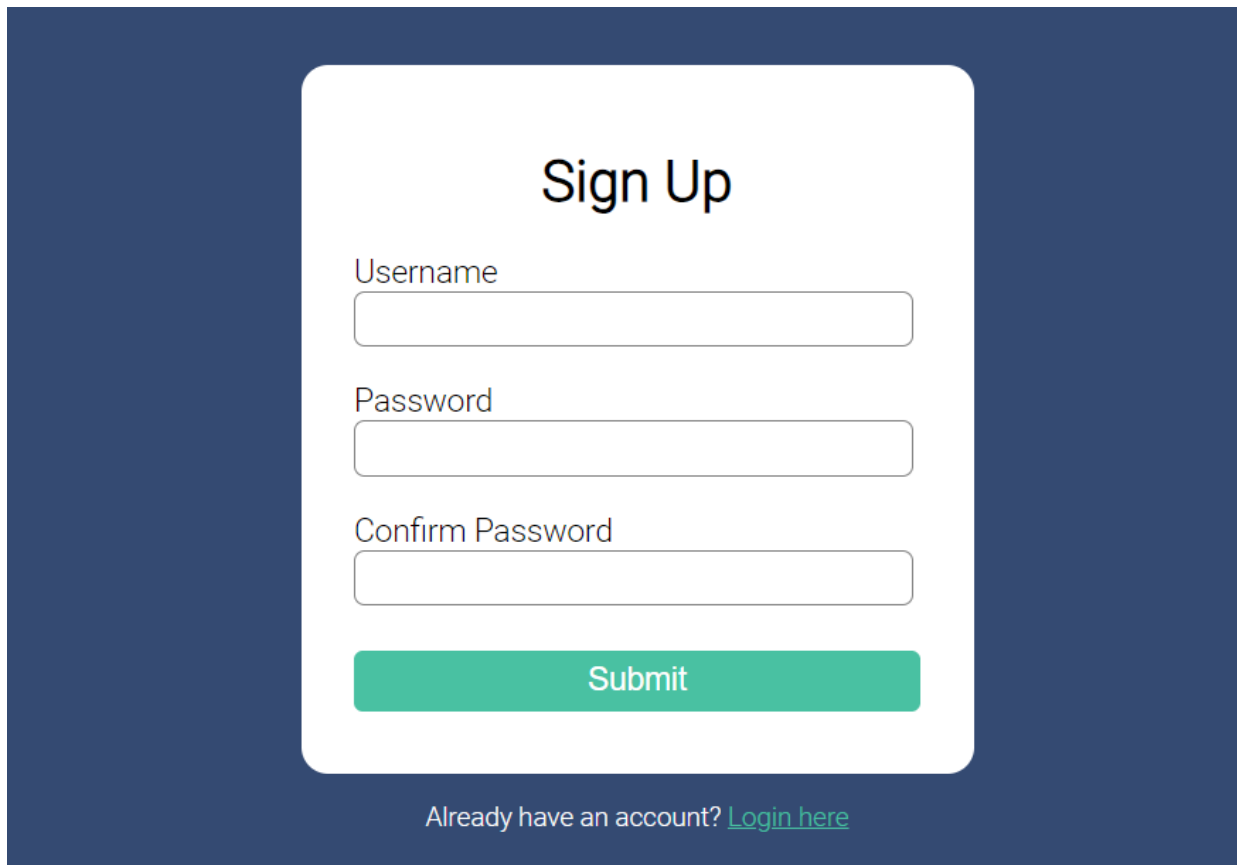
A screenshot of a web application's login page. The page has a dark blue background. In the center, there is a white rounded rectangle containing the login form. The word "Login" is displayed in a large, black, sans-serif font at the top of the white box. Below it, there are two input fields: the first is labeled "Username" and the second is labeled "Password". Both labels are in a smaller, black, sans-serif font. Each label is positioned to the left of its corresponding input field, which is a white rectangle with a thin grey border. Below the input fields is a green rectangular button with the word "Submit" in white, sans-serif font. At the bottom of the white box, there is a line of text: "Not have an account? [Sign Up Here](#)". The text "Not have an account?" is in a small, black, sans-serif font, and the link "Sign Up Here" is in a green, sans-serif font.

Рисунок 3.1 – Авторизація

Вводимо ім'я користувача та пароль та натискаємо кнопку Submit, якщо ви уже зареєстрований користувач. Якщо ви новий користувач, то натискаємо внизу форми на посилання Sign Up Here та потрапляємо на сторінку реєстрації:

A sign-up form titled "Sign Up" is centered on a dark blue background. The form is a white rounded rectangle containing three input fields: "Username", "Password", and "Confirm Password". Below these fields is a green "Submit" button. At the bottom of the form, there is a link that says "Already have an account? [Login here](#)".

Sign Up

Username

Password

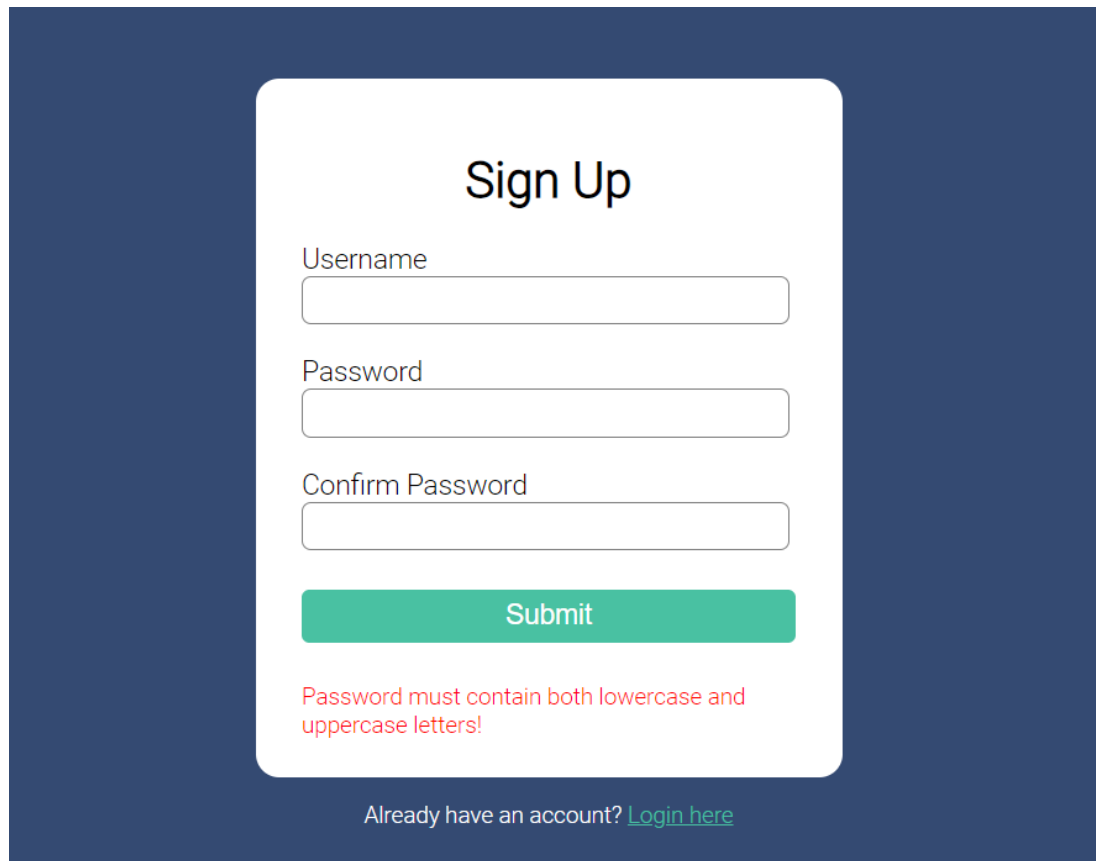
Confirm Password

Submit

Already have an account? [Login here](#)

Рисунок 3.2 – Реєстрація

Придумуємо ім'я користувача, що може складатися з латинських букв та цифр. Далі вводимо пароль, довжиною хоча б 8 символів та включає, як мінімум одну велику літеру, одну маленьку та цифру. При введенні невалідного паролю, з'явиться відповідна підказка:

A screenshot of a 'Sign Up' form on a dark blue background. The form is a white rounded rectangle containing three input fields: 'Username', 'Password', and 'Confirm Password'. Below the fields is a green 'Submit' button. A red error message, 'Password must contain both lowercase and uppercase letters!', is displayed below the 'Confirm Password' field. At the bottom, a link 'Login here' is provided for users who already have an account.

Sign Up

Username

Password

Confirm Password

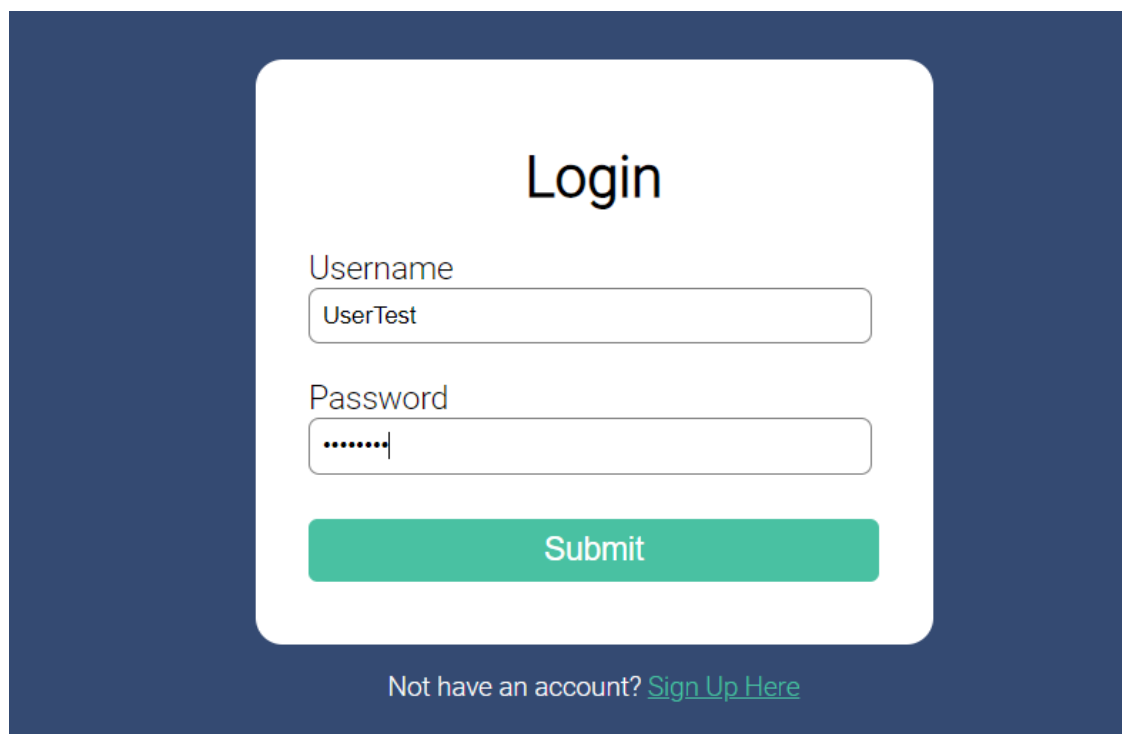
Submit

Password must contain both lowercase and uppercase letters!

Already have an account? [Login here](#)

Рисунок 3.3 – Введення валідного паролю

При успішній реєстрації, потрапляємо знову на сторінку логіну, де вводимо дані щойно створеного облікового запису:

A screenshot of a 'Login' form on a dark blue background. The form is a white rounded rectangle containing two input fields: 'Username' (with the text 'UserTest') and 'Password' (with masked characters '.....'). Below the fields is a green 'Submit' button. At the bottom, a link 'Sign Up Here' is provided for users who do not have an account.

Login

Username

UserTest

Password

.....

Submit

Not have an account? [Sign Up Here](#)

Рисунок 3.4 – Авторизація

Натиснувши кнопку “Submit”, ви потрапляєте на головну сторінку веб-сервісу, де можете отримати API ключ для подальшого використання веб-сервісу:

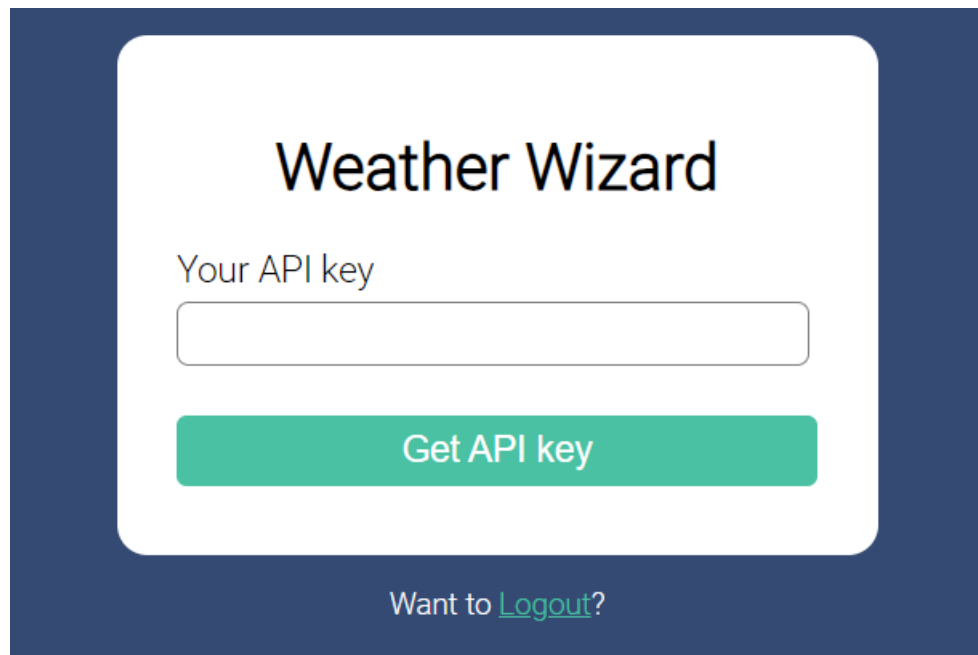


Рисунок 3.5 – Головна сторінка

Отримуємо API ключ за допомогою кнопки “Get API key”. Якщо ви новий користувач, то буде згенеровано ваш особистий унікальний API ключ. Якщо ви уже отримували, ключ то буде надано ваш уже створений раніше ключ. Також можна вийти з облікового запису натиснувши Logout.

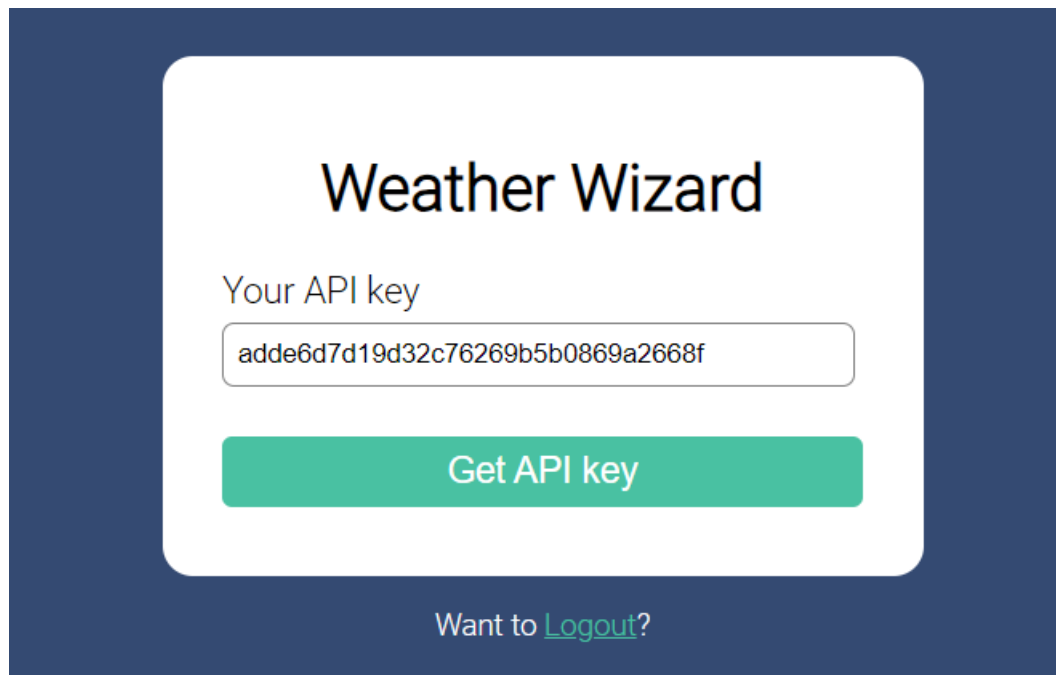


Рисунок 3.6 – Отримання ключа

Отримавши API ключ, ви можете користуватися послугами веб-сервісу. Для отримання поточної погоди потрібно звернутися за адресою /weather та задати наступні параметри lat та lon, що є широтою та довготою місця, та ввести свій API ключ в параметр apikey:

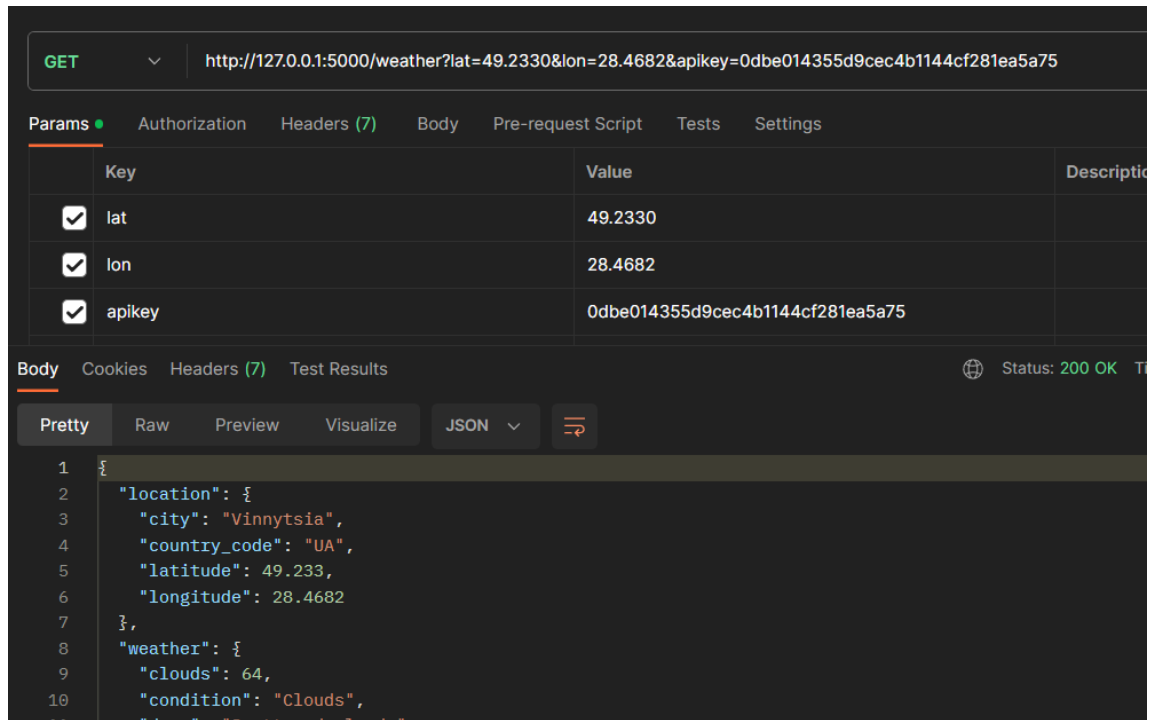


Рисунок 3.7 – Отримання поточної погоди

Виконання веб-застосунку:

На головній сторінці натиснути на поле для пошуку:

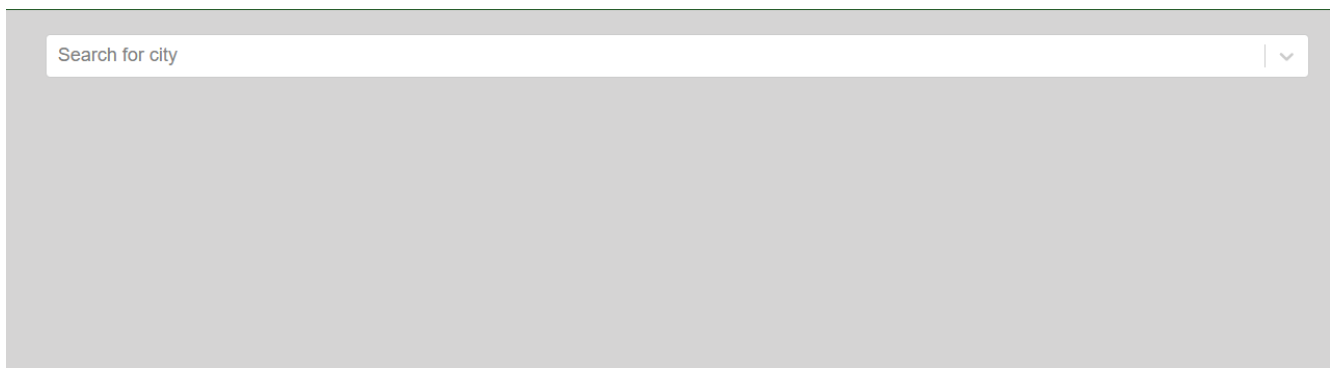


Рисунок 3.8– Поле для пошуку

Почати вводити назву місту та обрати серед запропонованих:

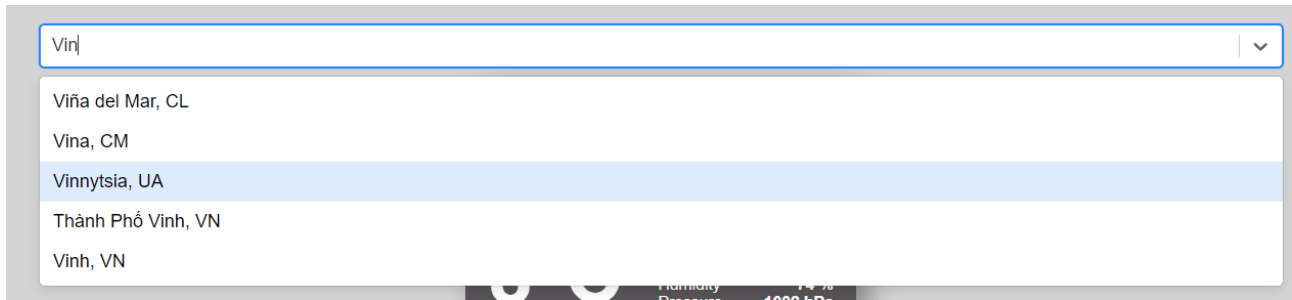


Рисунок 3.9 – Вибір міста

Далі отримуємо поточну та прогноз погоди в обраному місті:

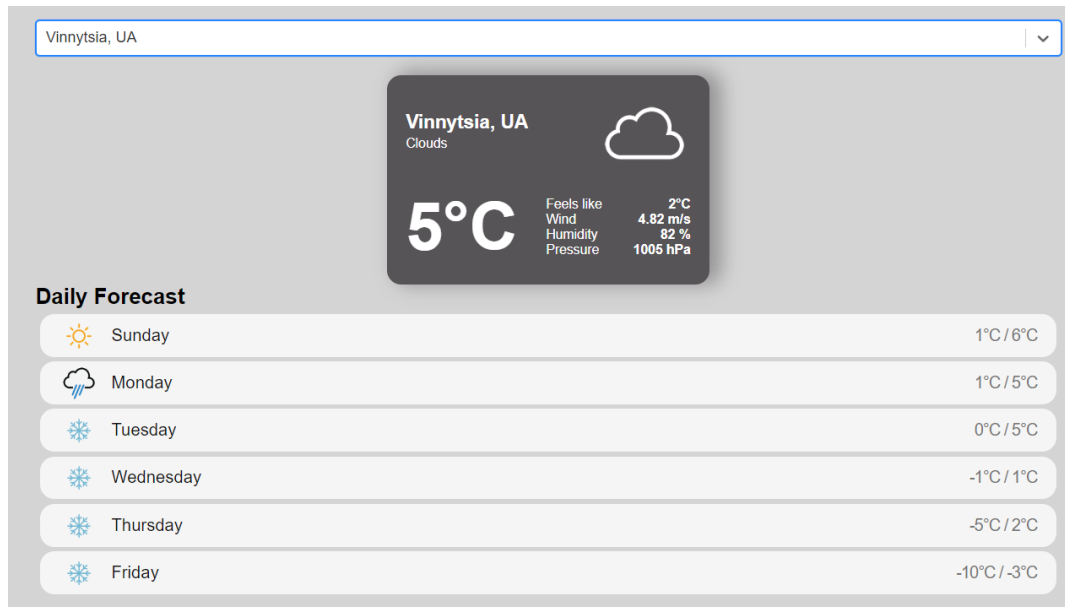


Рисунок 3.10 – Отримання погоди

Натиснувши на один з днів можна отримати детальнішу інформацію.

Подивимося деталі для вівторка:

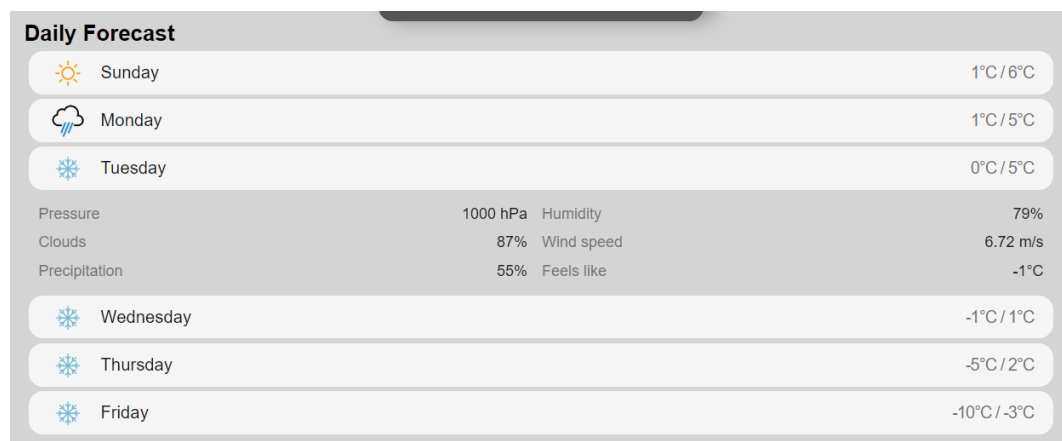


Рисунок 3.11 – Отримання деталей

Зм.	Арк.	ПІБ	Підп.	Дата							
Розробн.		Карамян В.С.			Відомість курсової роботи	Лім.		Лист		Листів	
Керівн.		Ахаладзе І.Е.						1		1	
Консульт						КПІ ім.Ігоря Сікорського кафедра ІПІ гр. ІП-13					
Н/контр.											
В.о.зав.каф											