БАКАЛАВРСЬКА РОБОТА

БР.КІ-20.00.00.000 ПЗ

Група КІ-18-1

Марчук Олег

2022

Міністерство освіти і науки України
Івано-Франківський національний технічний університет нафти і газу
Інститут інформаційних технологій
Кафедра комп'ютерних систем і мереж

Марчук Олег Романович

УДК 004.9

БАКАЛАВРСЬКА РОБОТА

Розробка веб-додатку візуалізації зведених метеорологічних даних для планування туристичних походів з використанням OpenWeatherMap API

Комп'ютерна інженерія
(назва освітньої програми)

123 - Комп'ютерна інженерія
(шифр і назва спеціальності)

Робота містить результати власних досліджень, використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело:

Здобувач освітнього ступеня _____ *Марчук О.Р.*
(підпис, ініціали та прізвище здобувача)

Науковий керівник _____ *Слабінога М. О., к.т.н., доцент*
(підпис, прізвище, ім'я, по батькові, науковий ступінь, вчене звання керівника)

Допущено до захисту
Завідувач кафедри КСМ

д.т.н.,проф. _____ *С.І. Мельничук*
(посада) (підпис) (дата) (ініціали та прізвище)

Івано-Франківськ – 2022 рік

ЗМІСТ

ВСТУП

Актуальність роботи. Сприятливі погодні умови - це запорука вдалого туристичного походу та безпеки його учасників. Тому врахування погодних умов при розробці нитки маршруту - важливий етап планування походу. Сучасні сервіси погоди пропонують доволі точну інформацію про погодні умови, а спеціалізовані сервіси для моніторингу погодних умов в горах дають оцінку метеорологічних умов з урахуванням специфіки руху туристичної групи. Разом з тим, особливістю походів Українськими Карпатами є те, що невисокі хребти, незначні перепади висот та відносно пологі схили дозволяють будувати нитки машрутів так, що за декілька днів група може подолати перейти межі декількох районів, а то й областей. Тому при плануванні походу слід враховувати саме погодні умови на N-ний день походу в певній точці (або ближче до певного населеного пункту). Тому задача розробки веб-сервісу, що надавав би інформацію про погоду в залежності від нитки маршруту, є актуальною.

Об’єкт дослідження. Процес інформаційної підтримки планування багатоденних походів.

Предмет дослідження. Веб-сервіси, що відображають дані прогнозу погоди.

Мета дослідження: розробка веб-додатку візуалізації зведених метеорологічних даних для планування туристичних походів з використанням OpenWeatherMap API.

Досягнення мети передбачає виконання наступних **задач**:

1. Аналіз предметної галузі;
2. Вибір засобів реалізації;
3. Аналіз OpenWeatherMap API;
4. Розробка логіки програми;
5. Проектування графічного інтерфейсу користувача.

Методи дослідження, що застосовувалися: методи розробки програмного забезпечення, методи порівняльного аналізу.

Практичне значення результатів роботи полягає у створенні веб-додатку візуалізації зведених метеорологічних даних для планування туристичних походів, що дає змогу отримати метеорологічні дані по днях походу в певних населених пунктах, залежно від нитки маршруту.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Проблема прогнозування погодних умов при плануванні туристичних маршрутів

Безаварійність – найважливіша вимога для кожної туристичної подорожі. Для недосвідчених туристів навіть найпростіший вихід за місто може бути тісно переплетеним з факторами ризику. Для того, щоб туризм був джерелом здоров’я та фізичного розвитку, кожен турист повинен суворо дотримуватися правил поведінки та безпеки за будь-яких умов.

Одним із важливих моментів підготовки до походу є вибір та розробка маршруту. Процес складання маршруту можна розділити на два етапи: складання маршруту в цілому, його затвердження та детальне вивчення об’єктів, через які проходить нитка маршруту.

Після того як маршрут був сформований, його потрібно умовно розбити на маленькі сегменти, у відповідності до днів походу, чи окремими ділянками між опорними точками. На цьому етапі одним з ключових факторів є погодні умови.

Сильний супротивний вітер, незважаючи на значне нервово-м'язове напруження, знижує швидкість руху людини на 20-25%, заважає диханню, порушує його нормальний ритм і збільшує навантаження на дихальну систему. Крім того, тривалий сильний вітер впливає на нервову систему, викликаючи збудження і роздратування, що виникає при швидкості вітру 6-7 м/с.

Поєднання сильного низьких температур та вітру надзвичайно небезпечне. Таким чином, починаючи зі швидкості вітру 15 м/с, подальше збільшення швидкості на 1 м/с відповідає зниженню температури повітря приблизно на 5 °С. Якщо врахувати, що на кожні 100 м підйому температура знижується в середньому на 0,62 °С, то можна бути впевненим, що навіть влітку в середньогір’ї можна застудитися.

Опади є надзвичайно важливим кліматичним фактором. Коли температура повітря знижується з висотою, це призводить до підвищення відносної вологості повітря. Приміром, повітря з абсолютною вологістю 2,5мм рт. ст. за температури + 15°С має відносну вологість 19,5%. А з тією ж абсолютною вологістю, але за

температурі -5 °С відносна вологість становитиме 79%, що є близьким до стану насиченості. Однак коли значення відносної вологості повітря перевищує 100% – отримуємо опади у вигляді снігу чи дощу.

Крім туману, дощу, снігу, в горах часто трапляється град. Випадає він з грозових хмар, за температури наземного повітря більше +10°С. Навіть звичайний град (до 3 см в діаметрі) може поранити або пошкодити намет. Град – явище характерне лише для теплих періодів і помірних широт. Зазвичай град припадає на другу половину дня, від обіду і до вечора (12-18 години). Це явище є короткочасним, зазвичай не перевищує 20-30 хвилин.

В зимовий період та період міжсезоння велику небезпеку становлять снігові лавини, лавини. Найбільш реальною загрозою під час подорожей по горах України поза сезоном є лавини. Лавину вже зафіксували при ухилі 15°. Були випадки сходження лавин і в Криму, не кажучи вже про Карпати. Лавини не характерні для гірського Криму. Але в окремі сніжні зими вони знаходяться на головному хребті.

Сезонні коливання смертності свідчать про несприятливий вплив низьких температур на здоров’я людини. Взимку у Великобританії, США, Франції, Швеції та ін. країни мають більше госпіталізацій і смертей, ніж влітку. Існує лінійна залежність між зимовою смертністю та зовнішньою температурою. Однак лише невелика кількість зимових смертей безпосередньо пов’язана з низькою температурою тіла (переохолодженням). Найчастішими причинами смерті є серцево-судинні та респіраторні захворювання. Серцево-судинна недостатність посилюється при тривалих застудах. Кількість серцевих захворювань збільшується через 1-2 дні холодів, інфарктів - через 3-4 дні, пневмонії та бронхіту - через 7 днів холодів.

Влітку сонце і тепло становлять небезпеку. Вплив високих температур може призвести до перегрівання організму. При цих станах відбувається розширення судин, зниження тонусу серця і артеріального тиску, почастішання пульсу, підвищення температури шкіри. Значний перегрів може призвести до серйозних патологічних явищ з боку серцево-судинної системи і загального стану організму (теплового удару). Сонячний удар виникає раптово і нерідко призводить до

серйозних ускладнень. Найхарактернішим симптомом є дуже висока температура тіла.

З огляду на ці фактори, важливою задачею при плануванні походу є розробка маршруту з огляду на метеорологічні умови. Цей процес ускладнюється при плануванні тривалих багатоденних лінійних походів, де перехід інколи проходить двома чи більше областями, а крайні точки маршруту знаходяться по різні боки декількох ниток гірських хребтів. Саме тому актуальною задачею є агрегація прогнозу погоди в контрольних точках, які буде проходити група, з огляду на час проходження даних точок.

1.2 Сервіси отримання інформації про погодні умови в горах.

Sinoptik - сервіс для перегляду прогнозу погоди. Перейшовши на головну сторінку сервісу сайту користувач отримує доступ до основного функціоналу(рисунок 1.1).

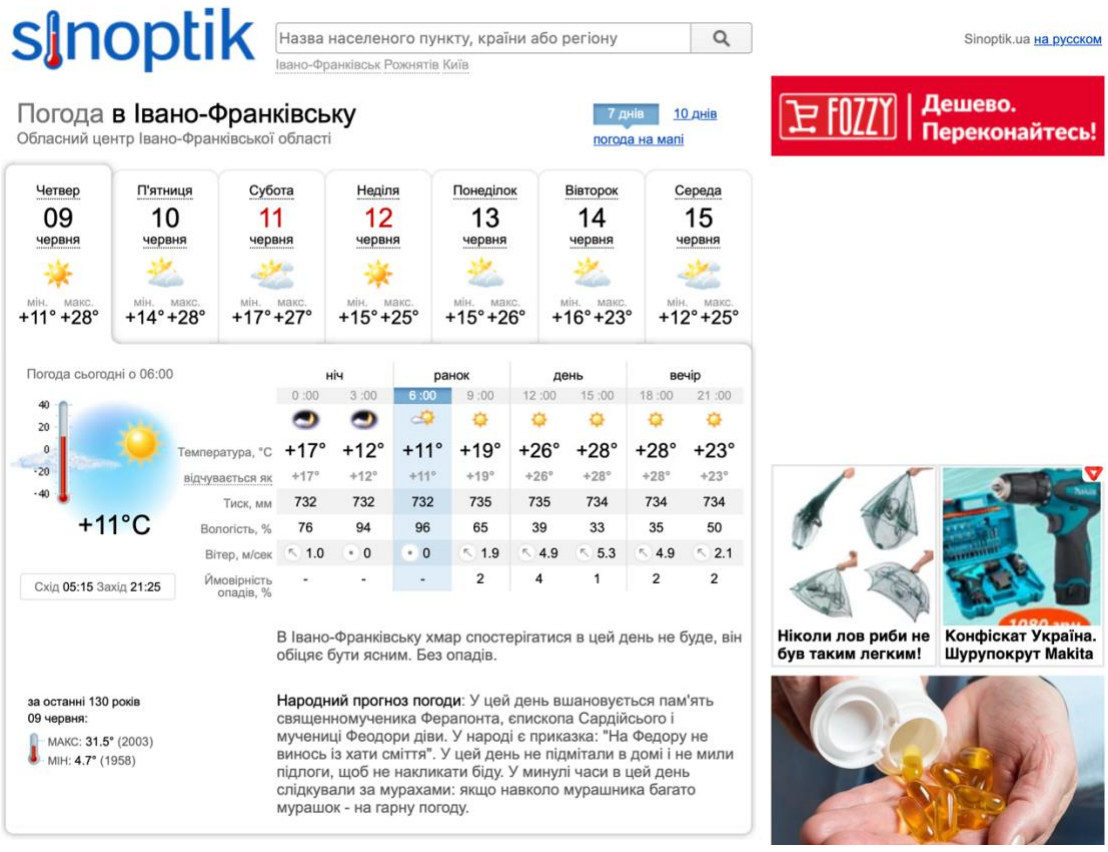

Рисунок 1.1 – Головна сторінка сайту Sinoptik

Глянувши на сторінку можна одразу виділити такі ключові його частини: хедер(так звана «шапка» сайту), основний контент, та футер(так званий «підвал» сайту).

До хедеру належать такі елементи:

1. логотип сайту;
2. поле пошуку потрібного населеного пункту. Під ним, як підказки, вказані раніше шукані населені пункти;
3. посилання на російськомовну версію сайту;
4. назва населеного пункту для якого відображається погода;
5. посилання-перемикач на версію сайту з погодою на 10 днів.

Основна частина сайту зображена у вигляді вкладок, які дозволяють відкрити панель з детальною інформацією для конкретного дня. Вкладка містить коротку інформацію, а саме:

1. день тижня, число та місяць;
2. іконка, що умовно зображує погоду(ясно, мінлива хмарність, дощ і тд.);
3. мінімальна та максимальна температура на добу.

Відкрита панель з погодою складається з таких елементів:

- блок з зображенням, яке умовно ілюструє погоду на найближчу наступну годину з прогнозу. Зображення доступне тільки для сьогоднішнього дня, у всіх наступних воно заміняються на календарну дату, на подобу тої, що є у вкладці;
- час сходу і заходу сонця;
- історичні дані про рекордні значення мінімальної та максимальної температури за останні 130 років;
- таблиця з полями метеорологічних даних: температура, як вона відчувається, тиск, вологість, вітер, ймовірність опадів. Та значеннями цих параметрів для ночі, ранку, дня, вечора. Для перших двох днів значення подаються з інтервалом в три години, тобто по два значення на частину доби, і з інтервалом у шість годин для всіх наступних днів;

- короткий текст опису погоди на протязі дня;
- народний прогноз погоди, де розповідають про народні повір’я та релігійні свята, що стосуються погоди чи повсякденного життя.

Також, Sinoptik дає можливість переглядати погоду населених пунктів на карті. Як видно на рисунку 1.2 на карті біля назви населеного пункту є іконка погоди, як на головній сторінці, а також температура повітря.

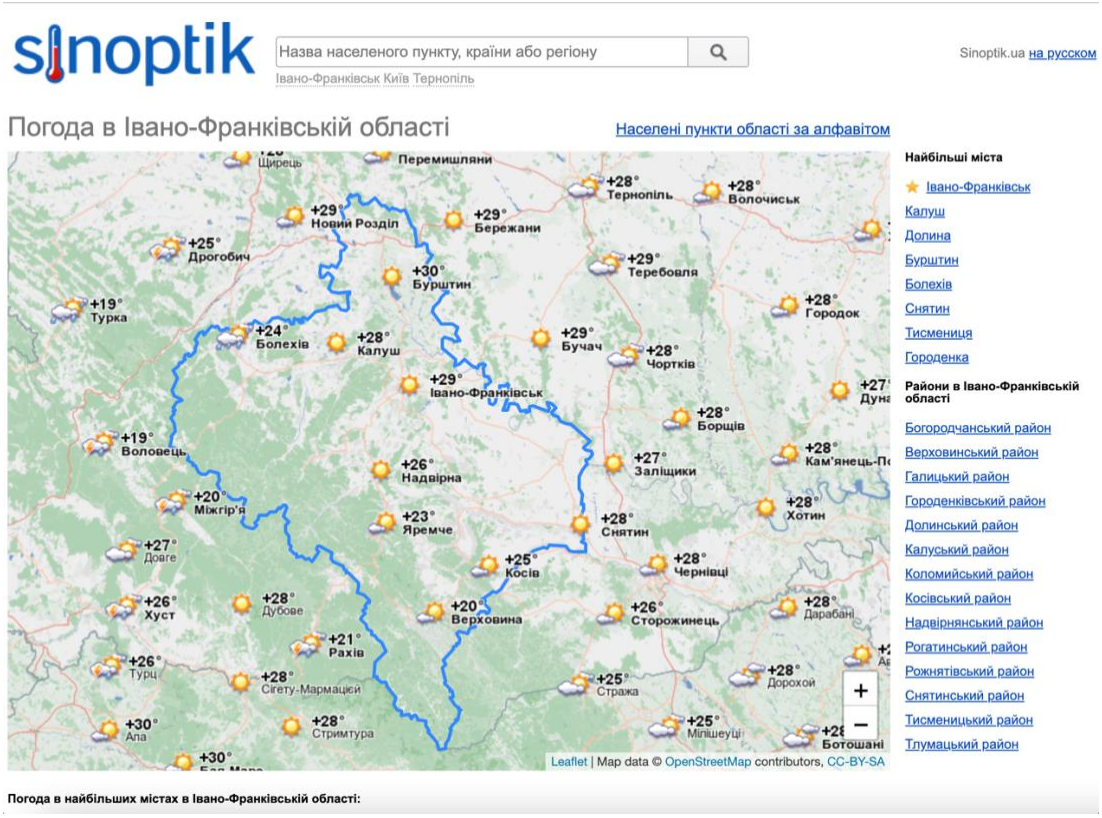

Рисунок 1.2 – Сторінка сайту для перегляду погоди на карті

Щоб дізнатися, як розвивався Sinoptik, я скористався веб архівом Wayback Machine. Згідно результат пошуку на рисунку 1.3. можна побачити, що перший запис сайту Sinoptik був зроблений 21 серпня 2010 року. Інтерфейс сайту практично не змінився з тих часів. Основні відмінності – відсутність полів «відчувається як», «ймовірність опадів», а також присутнє поле з інформацією про радіаційний фон. Також можна помітити посилання «Погода в цей день рік тому» для перегляду історичних даних, яке зараз відсутнє.

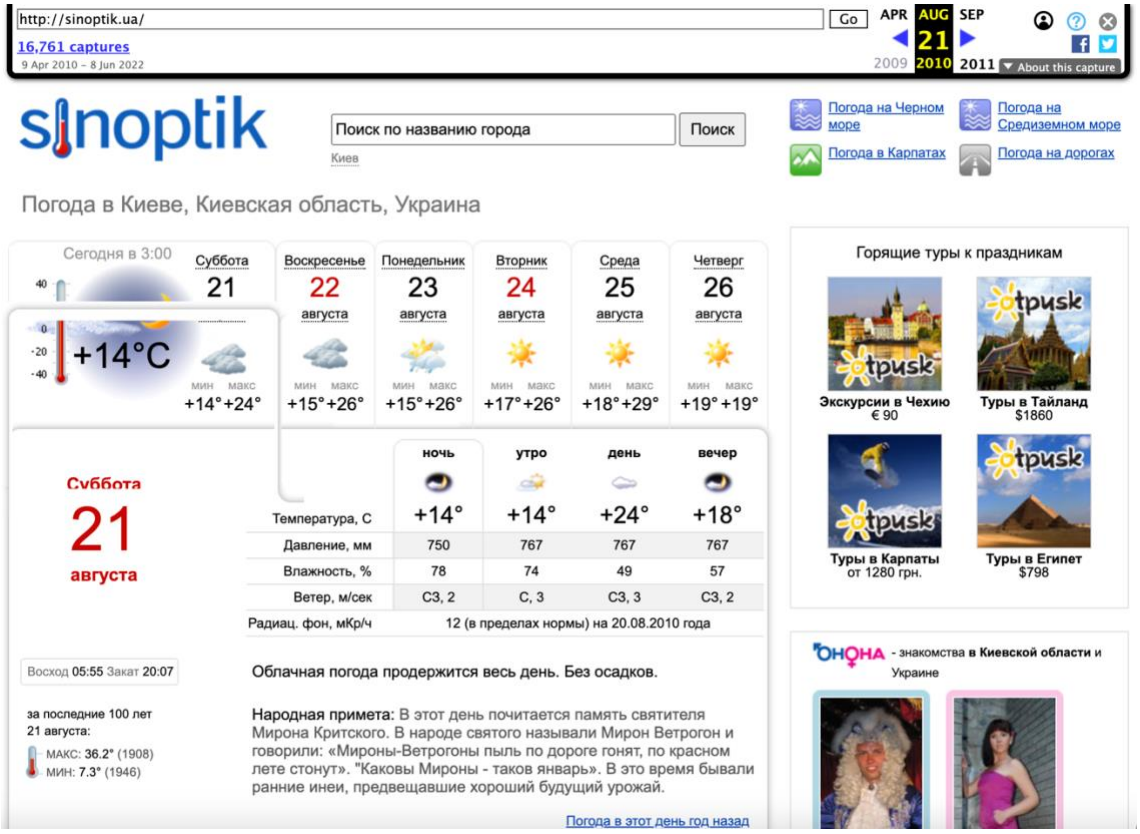

Рисунок 1.3 – Перший запис сайту Sinoptik у сервісі Wayback Machine

Сервіс Sinoptik видається першим результатом в пошуку, що свідчить про те, що ним активно користуються. Незважаючи на те, що його інтерфейс трохи старомодний, і не сильно змінився за ці роки, але це дає свої переваги – краща підтримка старих пристроїв. Також це плюс для старших людей, які скептично ставляться до всього нового.

Gismeteo – сервіс для перегляду прогнозу погоди, який видається другим результатом пошуку гугл за словом «погода». Як видно на рисунку 1.4 інтерфейс користувача має досить сучасний вигляд.

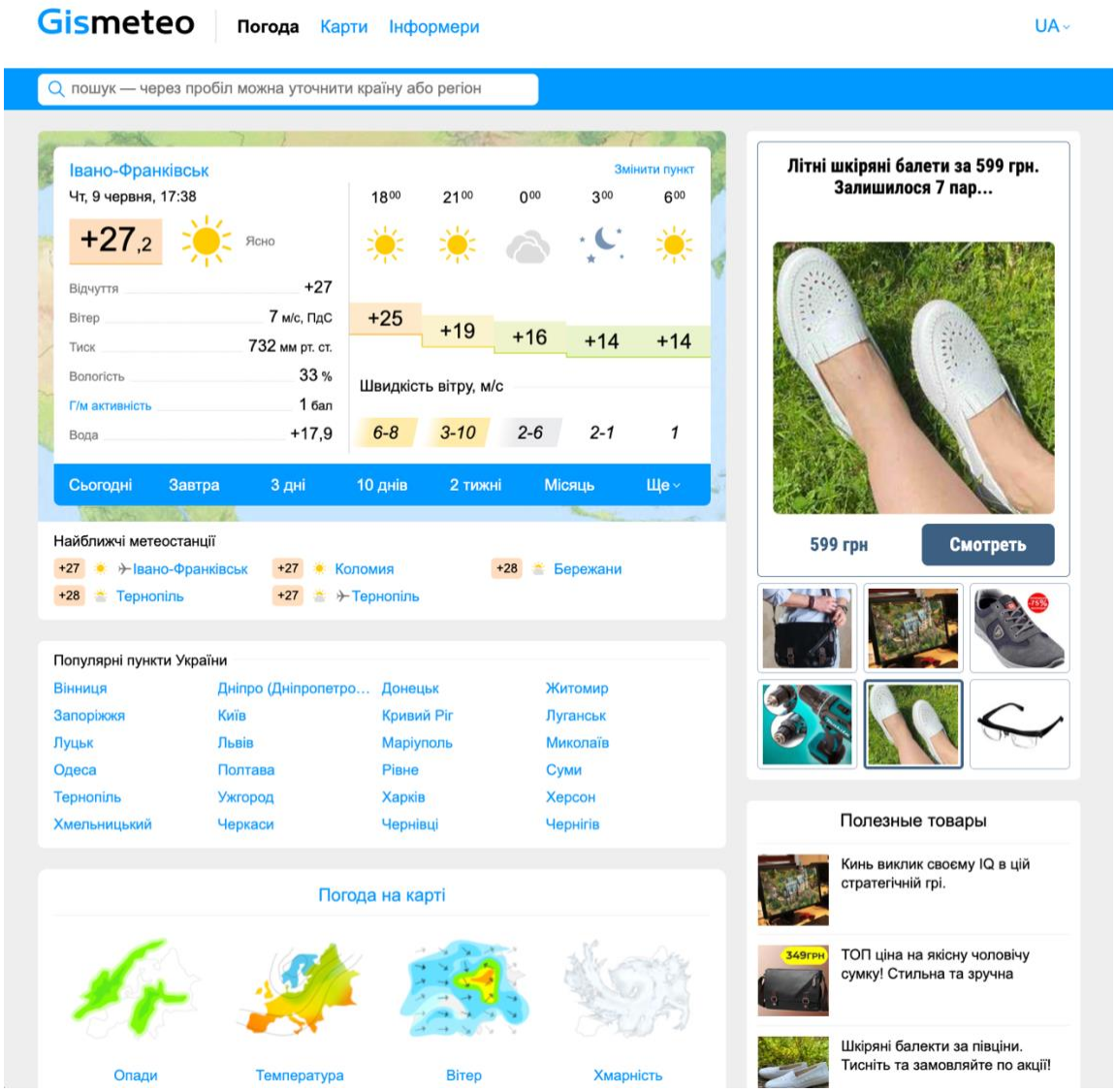

Рисунок 1.4 – Головна сторінка сайту Gismeteo

Основний вміст сторінки, складається з трьох блоків:

- блок погоди, з основною інформацією метеорологічною інформацією міста з локацією IP-адреси. Внизу карти з погодою є посилання для перегляду детальної інформації для: сьогодні, завтра, 3 дні, 10 днів, 3 тижні, місяць і тд. Ще нижче знаходяться посилання на погоду для сьогоднішнього дня в найближчих населених пунктах з метеостанцією;
- блок з посиланнями на детальну інформацію для переглянутих раніше міст та популярних населених пунктів України;
- блок для з посиланнями на карти з наступними даними: опади, температура, вітер та хмарність.

При кліку на одне з вищезгаданих посилань, чи посилання в хедері відкривається сторінка з карти. Як можна побачити на рисунку 1.5 сторінка карти складається з того ж таки хедеру, до якого додались випадаючий список вибору регіону карти: Європа, Сибір, Далекий Схід, та посилання для перемикання типів карти. Нижче вказаний обраний тип карти та регіон. Далі розміщена карта, внизу якої зображений градієнт кольорів та значення величини, яку він позначає. В блоці карти є елементи плеєра, які дозволяють відтворювати анімацію. Справа від карти є випадаючий список вибору часового поясу, і список дати та часу, для перегляду стану карти станом на конкретний момент найближчих трьох днів з інтервалом у три години.

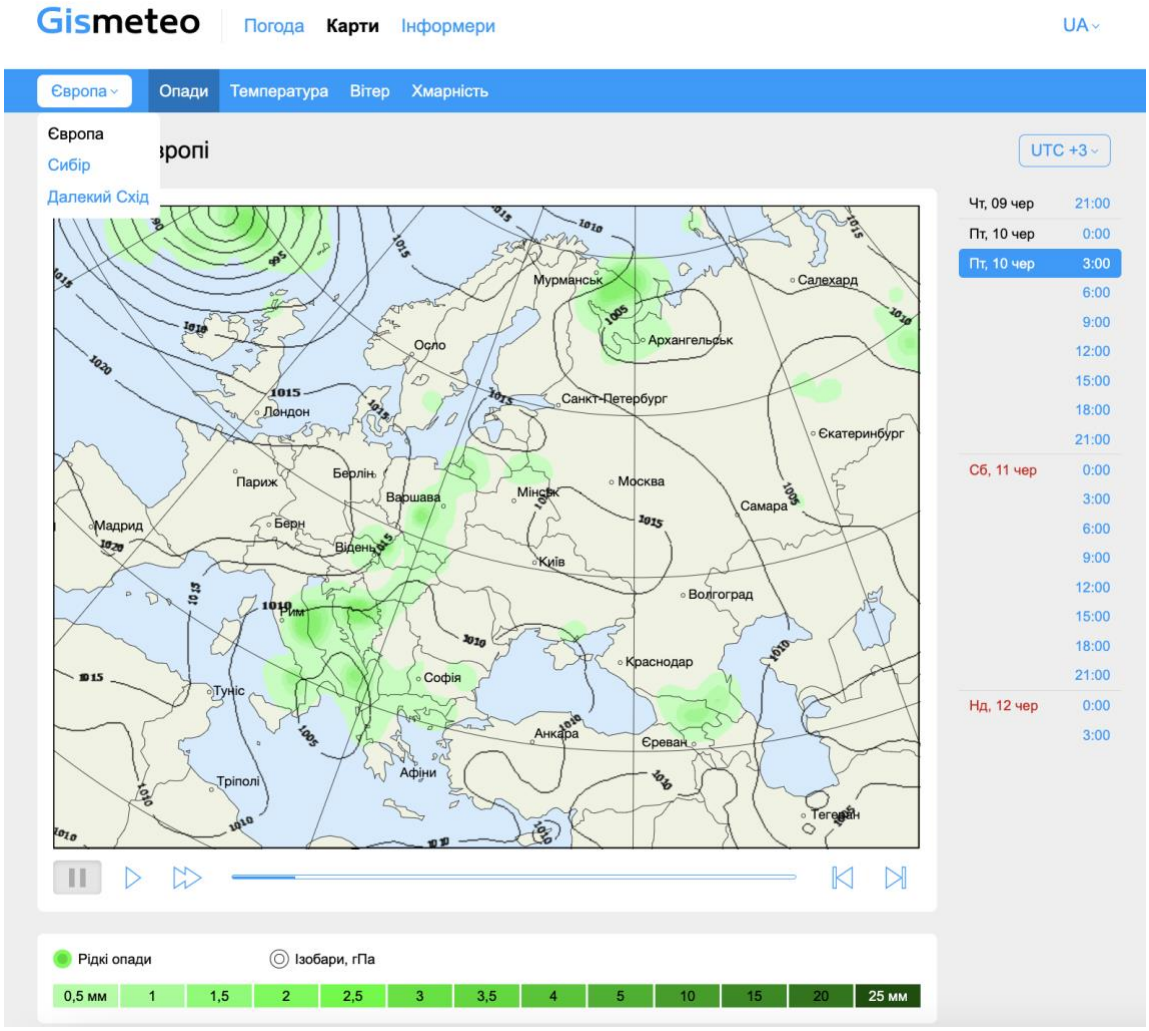

Рисунок 1.5 – Сторінка «Карти» сайту Gismeteo

При кліку на посилання назви міста чи час погоди, відкривається сторінка з детальною інформацією(рисунок 1.6). В хедері з’явились додаткові посилання на посилання для упрощення навігації по відповідних блоках сторінки: Погода, Вітер, Пилок, Дороги, Тиск, Вологість, Сонце і Місяць, Геомагнітна активність. Блок з погодою при містить три вкладки для навігації між днями, коли відкрито показ погоди на один день, та звичайні кнопки навігації, коли обрано показ. Блок погода відображає такі параметри: температура, швидкість вітру, сума опадів, а також іконки з умовним позначенням стану погоди. При перегляді погоди на один день ці параметри відображаються з інтервалом в три годин, при відображенні для трьох днів з інтервалом в 6 годин, та при відображенні від тижня і більше – одне значення на день. Інші вкладки відображають інформацію в такому ж форматі. Відрізняється тільки «Сонце і Місяць» там вказано тривалість дня, ночі, та час сходу, заходу сонця і місяця.

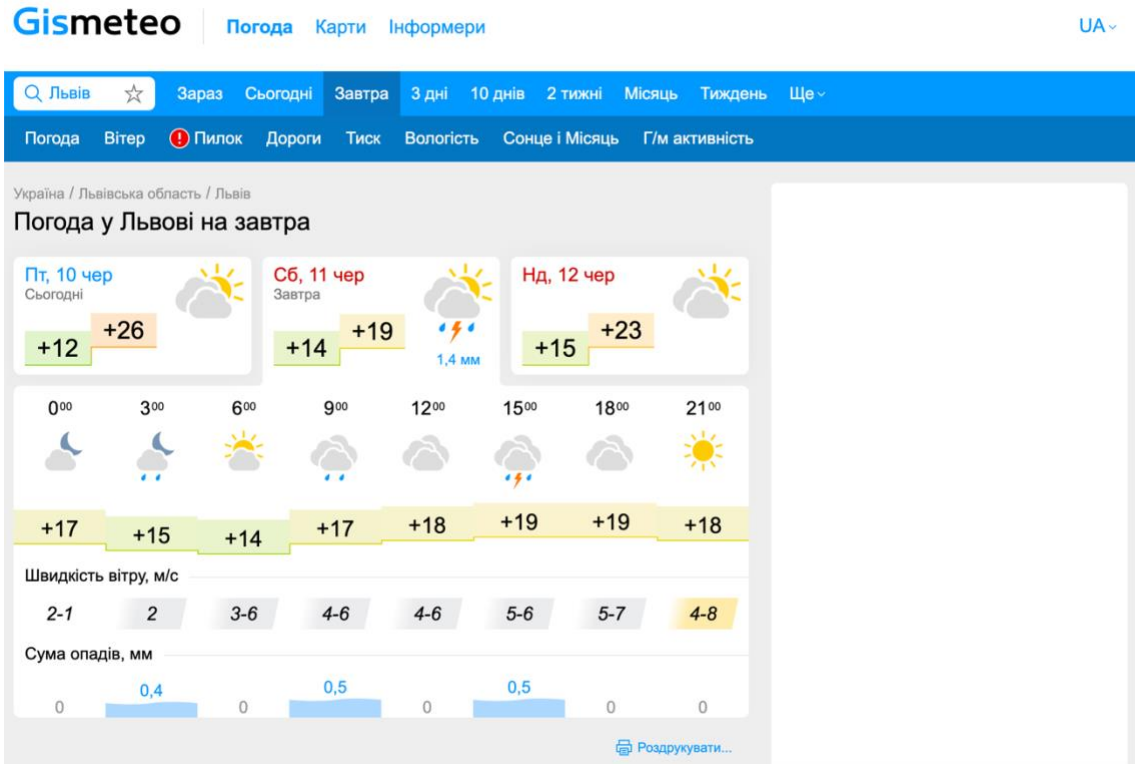

Рисунок 1.6 – Сторінка з детальною інформацією сайту Gismeteo

Gismeteo, на відміну від Sinoptik, має сучасний інтерфейс. Попри те, що замість простішого «ймовірність опадів» в Sinoptik, Gismeteo має суму опадів, а

параметр «Відчувається як» взагалі відсутній, цей сервіс надає набагато більше різних прогнозів як і в звичайному форматі, так і на карті.

Mountain-Forecast – один з найпопулярніших сайтів, які надають спеціальні прогнози погоди для планування походів в гори. Як видно з рисунку 1.7, головна сторінка сайту має два ключові елементи для вибору вершини: карту, та форму, з полями: хребет, під хребет, гора. Крім них головна сторінка має таблицю висоти найближчих вершин, посилання на магазин речей з символікою сервісу, посилання карти: опадів, хмар, температури, вітру, і блок з останніми нотатками з походів.

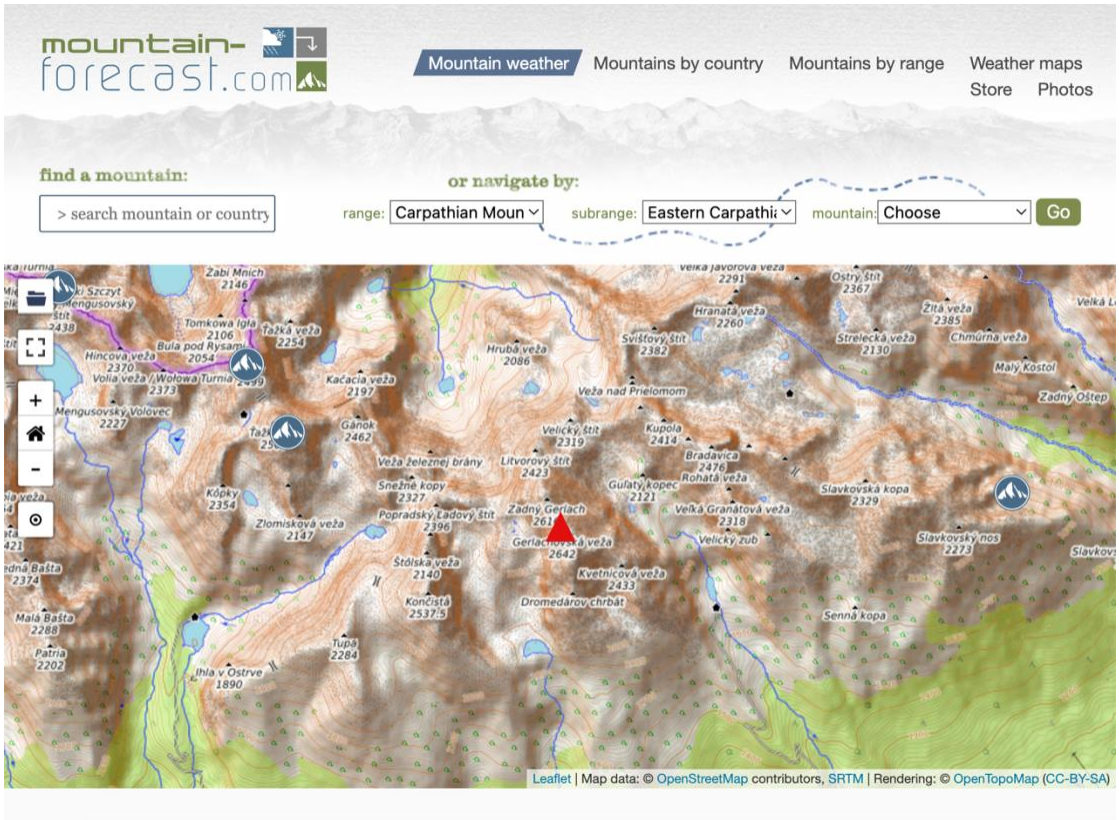

Рисунок 1.7 – Головна сторінка сайту Mountain-Forecast

При виборі гори, відкривається сторінка з детальною інформацією. Окрім карти, з'являється банер з фотографією, назвою, та координатами вершини. Нижче карти та банеру, що можна побачити на рисунку 1.8, знаходяться вкладки-посилання на: прогноз погоди, карту погоди, інформацію про вершину, нотатки пов'язані з цією вершиною, фото.

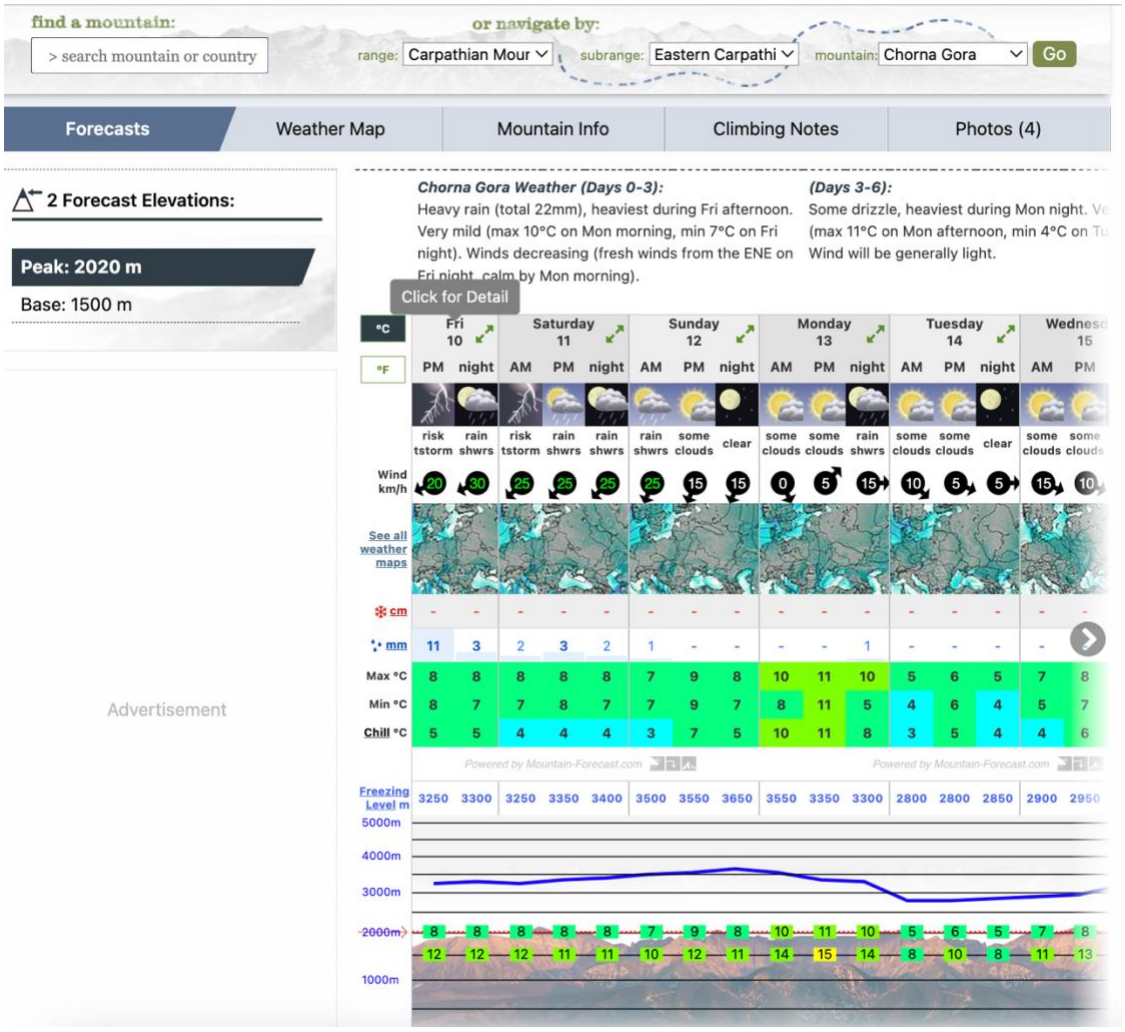

Рисунок 1.8 – Головна сторінка сайту Mountain-Forecast

На вкладці «Прогноз погоди» розміщено інформацію для найближчих дванадцяти днів. Вона складається з блоку з коротким текстовим описом погоди на кожні три дня з цього діапазону, та таблиці. В таблиці за замовчуванням для кожного дня вказано по три значення всіх величин та наступні поля:

1. іконка, з схематичним зображенням погоди, та текстом;
2. швидкість та напрям та міні-карту вітру;
3. сумарна кількість снігу, що випав за вказаний період часу;
4. сумарна кількість дощу, що випав за вказаний період часу;
5. максимальна, мінімальна температура, та температура комфорту, за вказаний період часу;
6. графік висоти замерзання за вказаний період часу;
7. час сходу та заходу сонця.

Використовуючи кнопки з зеленими стрілками біля числа місяця та дня тижня, можна отримати ці ж дані з іншим інтервалом: для перших двох днів – щогодинний прогноз, для перших семи днів – з інтервалом у три години.

При кліку на посилання карти погоди, відкривається сторінка з картою погоди для цілого регіону(рисунок 1.9). Є можливість перемикатися між різними типами карт: температура, хмари, погоди, вітру, опадів. Також є можливість відтворювати анімацію зміни карти.

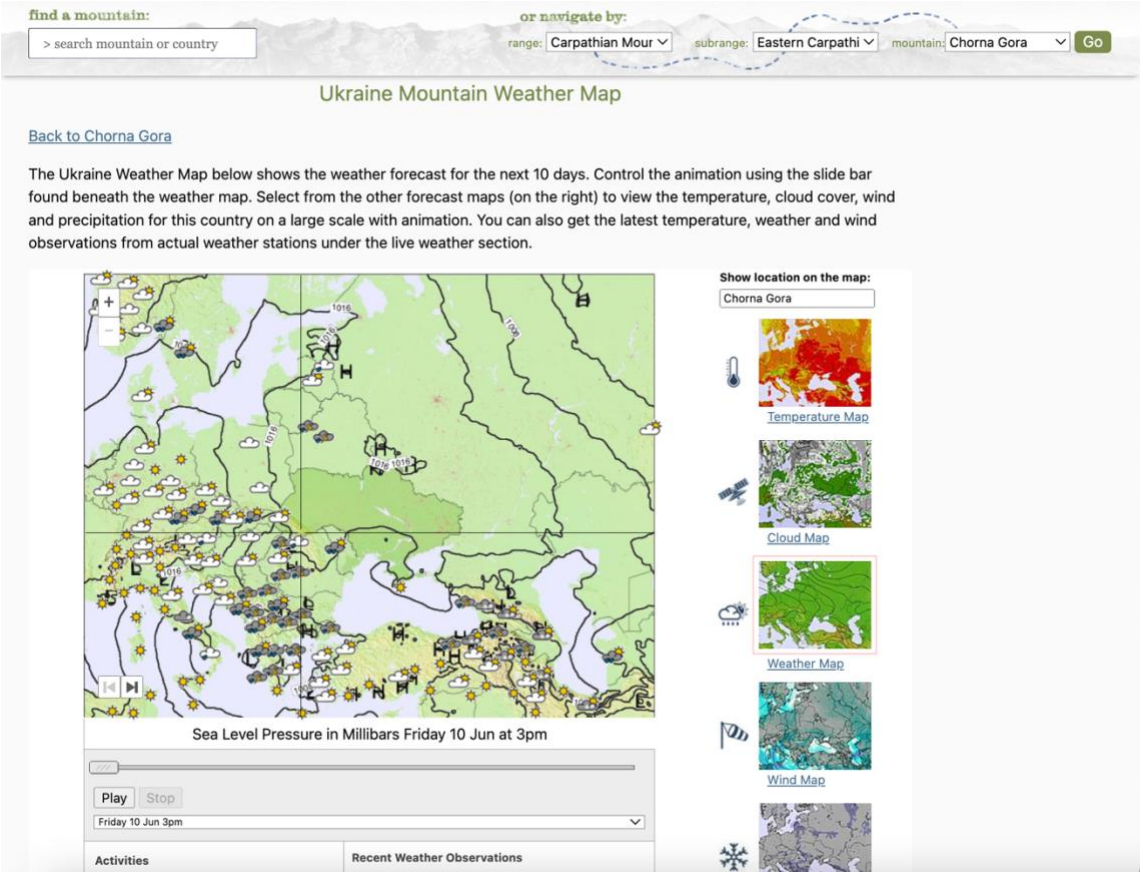

Рисунок 1.9– сторінка карти погоди сайту Mountain-Forecast

Mountain-Forecast – спеціалізований сервіс прогнозу погоди, це проявляється як і в маленьких деталях, наприклад, опади розділені на дощ і сніг, бо це важливо для оцінки лавинонебезпеки так і у великих – він має елементи форуму. Його інтерфейс також як і в Sinoptik малу змінився за останні 10 років, але це не проблема, враховуючи його специфіку.

1.3 Постановка задачі

Виходячи з проведеного аналізу, видно, що сучасні сервіси погоди пропонують доволі точну інформацію про погодні умови, а спеціалізовані сервіси для моніторингу погодних умов в горах дають оцінку метеорологічних умов з урахуванням специфіки руху туристичної групи. Разом з тим, особливістю походів Українськими Карпатами є те, що невисокі хребти, незначні перепади висот та відносно пологі схили дозволяють будувати нитки машрутів так, що за декілька днів група може подолати перейти межі декількох районів, а то й областей. Тому було прийнято рішення розробити веб-додаток, який би відображав погоду для нитки маршруту багатоденного походу щодня, залежно від запланованого пункту перебування групи.

Висновок до розділу 1

В першому розділі було здійснено аналіз предмету дослідження, здійснено аналіз схожих сервісів та сформовано задачу бакалаврської роботи.

2 РОЗРОБКА JAVASCRIPT ДЛЯ ЗАВАНТАЖЕННЯ ТА ВІДОБРАЖЕННЯ ДАНИХ ПРО ПОГОДНІ УМОВИ

2.1 Вибір засобів реалізації

Сучасні веб технології пройшли не малий шлях, щоб стати такими, які ми знаємо зараз. Спочатку, коли клієнтська сторона застосунку почала переймати на себе частину функціоналу серверної, був поширений підхід AJAX (Asynchronous JavaScript and XML), він передбачав часткове отримування «чистих» даних та оновлення частин додатку без перезавантаження сторінки. Зараз цих підходів стало більше:

- SPA (Single Page Application) – продовження концепції Ajax, що передбачає відчутність перезавантажень сторінки взагалі, та повне керування застосунком через JavaScript.
- PWA (Progressive Web Application) – передбачає наближення до нативних додатків – сторінка після завантаження зберігається в пам’яті пристрою і працює ніби окремий додаток.
- SSR (Server Side Rendering) – рендеринг інтерфейсу на стороні сайту.

Реалізація додатку не залежно від підходу, в тій чи іншій мірі передбачає використання HTML, CSS та JavaScript. HTML (Hypertext Markup Language) – мова розмітки потрібна для створення основного скелету сторінки. CSS (Cascading Style Sheets) – мова для організації стилістичного оформлення сторінки. JavaScript – динамічна та об’єктоорієнтована мова програмування веб додатків. Найбільшого використання дана мова програмування отримала у програмуванні клієнтської частини, оскільки відповідає за функціонування оформленого веб додатку. Це незмінна основа, однак існує велика кількість бібліотек та фреймворків, що спрощують розробку.

Для розробки застосунку було обрано JavaScript бібліотеку React та її екосистему. React – є одним із трьох найпопулярніших фронтенд фреймворків на рівні з Angular та Vue. Всі в певній мірі підтримують вищезгадані підходи та мають багато спільного, проте вони різні. Angular є повноцінним фреймворком який хоч і

має великі можливості, та сильно нав’язує свою екосистему, є досить складним і «громіздким» для невеликих проектів. Vue хоч і найменш популярний але досить збалансований, проте має в собі багато специфічного синтаксису. React гарно підходить для проектів різного рівня, він є тільки графічною бібліотекою, що дає гнучкість у виборі інших засобів, а також простий і зрозумілий.

Оскільки React – лише графічна бібліотека, було використано додаткові засоби розробки:

Redux – контейнер стану програми, він є дуже розповсюдженим, і має велику екосистему, ось його основні переваги:

- підвищує передбачуваність стану застосунку;
- легко підтримується, структура будь-якої редакс програми зрозуміла;
- запобігає небажане оновлення компонентів графічної бібліотеки;
- упрощує відлагодження застосунку;

Axios – зручний HTTP клієнт, що може працювати як і в браузері так і в node.js, з наступними основними перевагами:

- дозволяє створювати власні екземпляри з потрібною конфігурацією;
- дозволяє легко вказувати таймаут запиту;
- автоматична трансформація JSON;
- дозволяє легко перехоплювати, та трасформовувати запити;
- дозволяє лагко відслітковувати прогрес завантаження;

Material UI – шаблонні компоненти для упрощеної побудови інтерфейсів, його основні переваги:

- добре документований;
- регулярно оновлюється;
- легко модифікується за допомогою компоненту теми;

2.2 OpenWeather API

OpenWeatherMap – онлайн сервіс, що налижить OpenWeather Ltd. Він надає надає API для доступу до глобальних метеорологічні прогнози, історичні дані про погоду для будь-якого геграфічного локації. Модель згортового машинного навчання

використовується для обробки метеорологічних даних з метеорологічних станцій аеропортів, наземних радарів, супутників, супутників дистанційного зонування та автоматизованих метеостанцій.

Різноманітність погодних API, що надаються OpenWeatherMap, знайшла значну популярність серед розробників програмного забезпечення. API підтримують багато мов, декілька одиниць вимірювання, та стандартів формату даних, такі JSON і XML.

OpenWeather надає дані для управління ризиками погодних умов на основі індивідуальних угод для таких галузей:

- страхування – використання історичних даних для верифікації претензій страхового випадку;
- роздрібна торгівля – моделі прогнозування продажів для галузей, чутливих до погодних умов (їжа, напої, розваги). Комплексна аналітика впливу погоди на конкретний бізнес чи проект;
- енергетика – точне прогнозування на основі даних попиту та пропозиції на енергію. Поради щодо торгівлі на основі прогнозованих даних прогнозу погоди;
- сільське господарство – швидкий та зручний моніторинг сьогочасної погоди та шкідливих метеорологічних умов на полях (мороз, гроза тощо) для фермерів, щоб мінімізувати ризики втрат та ефективніше вести господарство;
- логістика - короткочасні прогнози для попереджень про сувору погоду, щоб ефективніше планувати маршрути та мінімізувати ризики втрати, затримки чи пошкодження вантажу.

Для створення власних проектів, OpenWeatherMap надає ряд API за різним доступом, деякі з них умовно безкоштовні, деякі надаються за підпискою, а деякі за особистим запитом через email. Типи підписок перелічені в таблиці 2.1. Студенти та викладачі можуть отримати піврічний безкоштовний доступ до підписки «Developer» та «Medium».

Таблиця 2.1 – Перелік підписок , що надає OpenWeatherMap

Назва підписки	**Ліміт трафіку, ГБ/міс.**	**Ліміт запитів у хвилину, тис.**	**Ліміт запитів у місяць, млн.**	**Перелік додаткових доступних API**
Free	0.0012	0.06	1	Current Weather, 3-hour Forecast 5 days, Basic weather maps, Weather DashBoard, Air Pollution API, Geocoding API, Weather widgets
Startup	30	0.6	10	Daily Forecast 16 days
Developer	140	3	100	Forecast 1 hour, Hourly Forecast 4 days, National Weather Alerts, Historical weather 5 days, Climatic Forecast 30 days, Advanced weather maps, Historical maps
Professional	370	30	1000	Bulk Download, Global Precipitation Map – Historical data
Enterprise	1500	200	5000	Road Risk API

Також крім загальних підписок, API за особистим запитом є спеціальні підписки на історичні дані.

Для розробки додатку було обрано «One Call API» для отримання прогнозу за координатами, та «Geocoding API», для отримання координат за пошуком. Ці API доступні безкоштовно з обмеженнями без спеціальної підписки.

«One Call API» дозволяє отримати:

- Дані про сьогочасну погоду;
- Щохвилинний прогноз погоди на найближчу годину;
- Щогодинний прогноз погоди на найближчі 48 годин;
- Денний прогноз погоди на 8 днів;
- Національні попередження.

– Історичні дані (з 1 січня 1979 р.)

Дані отримуються шляхом виконання HTTP GET запита, за адресою: https://api.openweathermap.org/data/3.0/onecall, з додаванням наступних параметрів:

– lat, lon (обов'язкові) – Географічні координати;
– appid (обов'язковий) – Власний API-ключ (доступний на сторінці акаунту);
– exlude – параметр дозволяє виключити з відповіді API деякі частини погодних даних;
– units – одиниці вимірювання: standart, metric, imperial
– lang – дозволяє отримати відповідь на обраній мові

Тіло відповіді буде мати наступну структуру:

– lat - географічні координати місця розташування (широта);
– lon - географічні координати місця (довгота);
– timezone - назва часового поясу для запитуваного місцезнаходження;
– timezone_offset – нсув в секундах від UTC;
– current – сьогочасні погодні дані;
    – current.dt – UTC, Unix час сьогочасної погоди;
    – current.sunrise – UTC, UNIX час сходу сонця;
    – current.sunset – UTC, UNIX час заходу сонця;
    – current.temp - Температура. Одиниці вимірювання – за замовчуванням: кельвін, метричні: цельсій, імперські: фаренгейт;
    – current.feels_like – параметр визначає сприйняття людиною температури;
    – current.pressure – атмосферний тиск на рівні моря, гПа;
    – current.humidity – відносна вологість повітря;
    – current.dew_point – точка роси;
    – current.clouds – хмарність, %;
    – current.uvi – поточний індекс ультрафіолету
    – current.visibility – середня видимість, максимальне значення – 10 км;

- current.wind_speed - одиниці вимірювання – за замовчуванням: метр/сек. , метрична система: метр/сек, імперська система: милі/година;
- current.wind_gust – пориви вітру, одиниці ті ж самі;
- current.wind_deg – напрям вітру, градуси;
- current.rain.1h – обсяг опадів за останню годину, мм;
- current.snow.1h – обсяг снігу за останню годину, мм;
- current.weather – погодні умови;
  - current.weather.id – OpenWeatherMap ідентифікатор погодних умов;
  - current.weather.main – група погодних умов параметрів (Дощ сніг і тд.);
  - current.weather.description – погодні умови;
  - current.weather.icon – ідентифікатор зображення погоди;

- minutely – щохвилинний прогноз погоди;
  - minutely.dt – unix, UTC час прогнозу;
  - minutely.precipitation – обсяш опадів, мм;
- hourly – щогодинний прогноз погоди;
  - hourly.dt - unix, UTC час прогнозу;
  - hourly.temp - температура. Одиниці вимірювання – за замовчуванням: кельвін, метричні: цельсій, імперські: фаренгейт;
  - hourly.feels_like – параметр визначає сприйняття людиною температури;
  - hourly.pressure – атмосферний тиск на рівні моря, гПа;
  - hourly.humidity – відносна вологість повітря;
  - hourly.dew_point – точка роси;
  - hourly.clouds – хмарність, %;
  - hourly.uvi – поточний індекс ультрафіолету

- hourly.visibility – середня видимість, максимальне значення – 10 км;
- hourly.wind_speed - одиниці вимірювання – за замовчуванням: метр/сек. , метрична система: метр/сек, імперська система: милі/година;
- hourly.wind_gust – пориви вітру, одиниці ті ж самі;
- hourly.wind_deg – напрям вітру, градуси;
- hourly.rain.1h – обсяг опадів за останню годину, мм;
- hourly.snow.1h – обсяг снігу за останню годину, мм;
- hourly.weather – погодні умови;
    - hourly.weather.id – OpenWeatherMap ідентифікатор погодних умов;
    - hourly.weather.main – група погодних умов параметрів (Дощ сніг і тд.);
    - hourly.weather.description – погодні умови;
    - hourly.weather.icon – ідентифікатор зображення погоди;

- daily – щоденний прогноз погоди;
    - daily.dt – UTC, Unix час прогнозу;
    - daily.sunrise – UTC, UNIX час сходу сонця;
    - daily.sunset – UTC, UNIX час заходу сонця;
    - daily.moonrise – UTC, UNIX час сходу місяця;
    - daily.moonset – UTC, UNIX час заходу місяця;
    - daily.moon_phase – Фаза місяця: 0 і 1 – молодий, 0.25 – перша чверть місяця, 0.5 – повний, 0.75 – остання чверть місяця;
    - daily.temp - Температура. Одиниці вимірювання – за замовчуванням: кельвін, метричні: цельсій, імперські: фаренгейт;
        - daily.temp.morn – ранкова температура;
        - daily.temp.day – денна температура;
        - daily.temp.eve – вечірня температура;

    - daily.temp.night – нічна температура;
    - daily.temp.min – мінімальна добова температура;
    - daily.temp.max – максимальна добова температура;
- daily.feels_like – параметр визначає сприйняття людиною температури;
    - daily.feels_like.morn – ранкова температура;
    - daily.feels_like.day – денна температура;
    - daily.feels_like.eve – вечірня температура;
    - daily.feels_like.night – нічна температура;
- daily.pressure – атмосферний тиск на рівні моря, гПа;
- daily.humidity – відносна вологість повітря;
- daily.dew_point – точка роси;
- daily.clouds – хмарність, %;
- daily.uvi – поточний індекс ультрафіолету
- daily.visibility – середня видимість, максимальне значення – 10 км;
- daily.wind_speed - одиниці вимірювання – за замовчуванням: метр/сек. , метрична система: метр/сек, імперська система: милі/година;
- daily.wind_gust – пориви вітру, одиниці ті ж самі;
- daily.wind_deg – напрям вітру, градуси;
- daily.pop – ймовірність опадів, %;
- daily.rain.1h – обсяг опадів за останню годину, мм;
- daily.snow.1h – обсяг снігу за останню годину, мм;
- daily.weather – погодні умови;
    - daily.weather.id – OpenWeatherMap ідентифікатор погодних умов;
    - daily.weather.main – група погодних умов параметрів (Дощ сніг і тд.);
    - daily.weather.description – погодні умови;

- daily.weather.icon – ідентифікатор зображення погоди;

- alerts – Національні попереджувальні дані про погоду від основних національних систем попередженння про погоду;
  - alerts.sender_name – назва джерела сповіщення;
  - alerts.event – назва події сповіщення;
  - alerts.start – дата і час початку сповіщення, unix, UTC;
  - alerts.end – дата і час закінчення сповіщення, unix, UTC;
  - alerts.description – опис сповіщення;
  - alerts.tags – тип погоди сповіщення.

«Geocoding API» - простий інструмент, який полегшує пошук місць під час роботи з географічними назвами та координатами. API геокодування OpenWeather підтримує як прямі, так і зворотні методи, працюючи з назвами міст, районів і районів, країн і штатів:

- прямий геокодинг перетворює вказану назву локації або поштовий індекс у точні географічні координати;
- Зворотній геокодинг перетворює географічні координати в назви найближчих місць.

Щоб отримати географічні координати(широта, довгота) потрібно здійснити виклик API прямого геокодингу. Для цього потрібно виконати HTTP GET запит, за адресою: http://api.openweathermap.org/geo/1.0/direct з додаванням наступних параметрів:

- q (обов’язковий) – назва міста;
- appid (обов’язковий) – API-ключ;
- limit – кількість результатів пошуку.

Відповідь матиме вигляд масиву структури:

- name – назва знайденого місця;
- local_names – локальні назви локації;

- local_names.[код мови] - назва знайденого місця різними мовами. Список імен може відрізнятися для різних місць;
- lat – широта знайденого місця;
- lon – довгота знайденого місця;
- country – країна розташування;
- state – штат знайденого місця.

Щоб отримати іконки погоди, які є частиною погодних даних потрібно використати посилання, підставивши туди id потрібної іконки: http://openweathermap.org/img/wn/{id}@2x.png.

2.3 Розробка програмного забезпечення для взаємодії

Розробка невеликого React веб-додатку починається зі створення проекту за допомогою інструменту командного рядка «Create React App»: запустивши його за допомогою npm – пакетного менеджера node.js, створюється проект за шаблоном, готовий до розробки. Цей інструменти також є однією із залежностей проекту, допомагає керувати додатком, та містить в собі додаткові модулі для розробки.

Наступним кроком є створення та підключення redux store – «єдиного джерела істини», в якому будь зберігатися дані пов’язані з бізнес-логікою веб-застосунку. Щоб надати доступ до стору react-компонентам в головному файлі index.tsx потрібно огорнути головний компонент додатку в компонент «Provider», імпортується з пакету «react-redux», та передати йому парамтер «store» - store застосунку. Фрагмент коду з додатку А, який описує під’єднання стору:

```
const store = setupStore();
ReactDOM.render(
  <Provider store={store}>
    <App />
  </Provider>,
  document.getElementById('root')
);
```

Для створення стору у файлі src/index.tsx, використано власноруч написану функцію, з файлу src/store/store.ts, що використовує функцію «configureStore» яка

приймає якості параметру об’єкт з переданим в його поле «reducer». Фрагмент коду з додатку Б:

```
export const setupStore = () =>
  configureStore({
    reducer: rootReducer,
  });
```

«Робочий потік» даних в redux є доволі простим і складається з таких ключових елементів:

- сам «store» єдине джерело істини, яке не можна змінювати;
- «action creator» - функція що повертає об’єкт дії;
- об'єкт дії, що містить в собі поле «type», для ідентифікації дії та «payload», що дозволяє передавати потрібні параметри в reducer;
- «dispatch» - функція, що дозволяє надсилати дії;
- «reducer» - функція що отримує дії та відповідно до її типу замінює store на новий з оновленими значеннями;
- «middleware» - функція що дозволяє перехоплювати кожну надіслану дію, щоб вносити до неї зміни, чи скасовувати. В основному використовується для роботи з API;

Оскільки для створення стору потрібно вказати глобальний редюсер, то створюєм його передавши у функцію combineReducer() створені редюсери. Фрагмент коду з додатку Б:

```
const rootReducer = combineReducers({
  weatherReducer,
  searchReducer,
});
```

Було створено два редюсери: перший – для зберігання погодних даних, другий – для здійснення пошуку місць.

Фрагмент коду з додатку В, що описує стан редюсера пошуку searchReducer:

```
interface SearchState {
  showSearch: boolean;
  destinationDay: number;
```

```
  citySearchResult: IGeocodingIndexed[];
  isLoading: boolean;
  error: string;
}

const initialState: SearchState = {
  showSearch: false,
  destinationDay: 0,
  citySearchResult: [],
  isLoading: false,
  error: '',
};
```

Редюсер пошуку searchReducer містить поля: showSearch - поле булевого типу, що визначає чи буде відображено компонент пошуку, destinationDay – поле що містить індекс дня подорожі, для якого здійснюється пошук, citySearchResult – поле що містить масив результатів пошуку, isLoading – булеве поле яке в подальшому дозволить показувати індикатори завантаження даних, error – текстове поле, що для збереження повідомлення помилки.

Щоб компонент пошуку мав змогу змінювати дані та робити запити, необхідно створити функції - action creator, потрібні для генерації об’єкта дії, виконання сторонніх дій за допомогою middleware, та функції-приймачі що будуть на основі об’єктів дій оновлювати стан додатку. Пакет redux toolkit надає зручний синтаксис для створення всіх цих сутностей за допомогою єдиної функції – createSlice(), але для сторонніх дій потрібне застосування middleware redux-thunk, тому щоб уникнути плутанини в коді, всі функції action-creator були реалізовані за допомогою функції createAsyncThunk(), для простого прикладу нижче наведений фрагмент коду додатка Г для створення асинхронного thunk для очистки результату пошуку:

```
export const clearSearchResult = createAsyncThunk(
  'search/clearSearchResult',
  async (payload, thunkAPI) => payload
);
```

createAsyncThunk() приймає два параметра перший - рядок щоб буде використаний для генерації поля type об’єкту дії створеної thunk, другий – callback-функція, що буде виконана при передачі thunk в dispatch() функцію. Callback-функція теж приймає два параметри: payload – стандартне поле для корисних даних об’єкту дії, та thunkAPI – об’єкт, що дозволяє виконувати функцію dispatch, зчитувати стан, відхіляти, чи успішно завершувати виконання thunk-у. Thunk «clearSearchResult» нічого особливого не робить, просто генерує об’єкт дії успішного виконання, для подальшої обробки в reducer-і. Те саме можна сказати і про thunk-и «showSearch», «hideSearch», що потрібні для реалізації можливості вмикати та вимикати відображення компоненту пошуку. Фрагмент коду з додатку Г:

```
export const showSearch = createAsyncThunk(
  'search/showSearch',
  async (payload: number, thunkAPI) => payload
);
export const hideSearch = createAsyncThunk(
  'search/hideSearch',
  async (payload, thunkAPI) => payload
);
```

Для здійснення пошуку було створено thunk «fetchSearchCity», ось фрагмент коду з додатку Г, що його описує:

```
export const fetchSearchCity = createAsyncThunk(
  'search/fetchSearchCity',
  async (cityName: string, thunkAPI) => {
    const searchResult = await OwmService.searchCity(cityName).catch(
      (e: AxiosError) => thunkAPI.rejectWithValue(e.message)
    );
    return searchResult;
  }
);
```

асинхронна callback-функція першим параметром «cityName» приймає назву міста, за якою буде здійснено пошук. Для виконання API використовується метод searchCity з спеціально розробленим для роботи з API сервісу OwmService. Якщо

виклик API не поверне помилок і результат виклику буде просто повернутий з callback-функції, thunk створить об’єкт дії з типом «fullfield», для подальшої обробки редюсером, якщо помилка виникне, то вона буде перехоплена блоком .catch() в якому за допомогою методу rejectWithValue виконання thunk відхилиться, що створить об’єкт з типом «rejected».

Для створення редюсеру пошуку було використано функцію createSlice(), ось фрагмент коду додатку В що викликає її:

```
export const searchSlice = createSlice({
  name: 'search',
  initialState,
  reducers: {},
  extraReducers: (builder) => {
    builder
      .addCase(fetchSearchCity.fulfilled, (state, action) => {
        state.isLoading = false;
        state.error = '';
        state.citySearchResult = action.payload;
      })
      .addCase(fetchSearchCity.rejected, (state, action) => {
        state.isLoading = false;
        state.citySearchResult = [];
        typeof action.payload === 'string' && (state.error =
action.payload);
      })
      .addCase(fetchSearchCity.pending, (state) => {
        state.isLoading = true;
      })
      .addCase(clearSearchResult.fulfilled, (state) => {
        state.citySearchResult = [];
      })
      .addCase(showSearch.fulfilled, (state, action) => {
        state.showSearch = true;
        state.destinationDay = action.payload;
      })
      .addCase(hideSearch.fulfilled, (state) => {
        state.showSearch = false;
```

```
        state.citySearchResult = [];
        state.error = '';
      });
  },
});
```

функція createSlice приймає єдиним аргументом об’єкт, з наступними полями:

- name – ім’я створюваного редюсера, в цьому випадку це «search»;
- initialState – початковий стан редюсера;
- reducers – об’єкт що містить методи редюсера для зміни стан. На основі цих функцій автоматично генеруються функції action creator. Тут використана бібліотека immer що дозволяє писати код мутуючий стан редюсера;
- extraReducers – поле для опису функцій-редюсері об’єктів-дій, що були описані ззовні.

Для опису функцій-редюсерів використовується callback-функція, що приймає єдиним аргументом об’єкт «builder». Для додавання функції-редюсера використовується метод addCase об’єкта builder. Першим аргументом він приймає тип дії (type) яку він має обробити тип, в цей параметр передаються автоматично згенеровані типи thunk-ом, кожен створює три такі:

- «pending» для дії яка автоматично надсилається при його виклиці;
- «fullfilled»для успішно виконання thunk-у;
- «rejected» для вдіхиленого thunk-у;

Другим аргументом – callback-функцію, яка, власне, і буде обробляти створені раніше дії. Callback-функція має два аргументи: перший – state, стан редюсера, а другий - action, об’єкт дії що обробляється.

У функції-редюсері для дії «clearSearchResult.fullfilled» , щоб очистити результати пошуку полю стану citySearchResult присвоюється значення порожнього масиву.

У функції-редюсері для дії «showSearch.fullfilled», щоб ввімкнути відображення компоненту пошуку:

- полю стану showSearch присвоюється значення true;

– полю destinationDay – значення передане в дії.

У функції-редюсері для дії «hideSearch.fullfilled», щоб приховати пошук та обнулити його стан:

– полю стану showSearch присвоюється значення false;

– полю destinationDay – значення передане в дії;

– полю error – значення порожнього рядка.

У функції-редюсері для дії «clearSearchResult.fullfilled» полю стану citySearchResult присвоюється значення порожнього масиву, щоб очистити результати пошуку.

У функції-редюсері для дії «featchSearchCity.pending» полю стану isLoading присвоюється значення true, щоб відобразити стан завантаження.

У функції-редюсері для дії «featchSearchCity.fullfilled», щоб зберегти дані пошуку:

– полю стану isLoading присвоюється значення false;

– полю стану error присвоюється значення пустого рядка;

– полю стану citySearchResults присвоюється значення передані в дії.

У функції-редюсері для дії «featchSearchCity.fullfilled», щоб зберегти результат невдалого пошуку:

– полю стану isLoading присвоюється значення false;

– полю стану error присвоюється значення повідомлення помилки з об’єкту дії;

– полю стану citySearchResults присвоюється значення порожнього масиву.

Фрагмент коду додатку Д, що описує стан редюсера погоди weatherReducer:

```
export interface WeatherState {
  cities: { [id: IGeocodingIndexed['id']]: IGeocodingIndexed };
  tripDays: ITripDay[];
  isLoading: boolean;
  error: string;
}
```

```
const initialState: WeatherState = {
  cities: {},
  tripDays: [],
  isLoading: false,
  error: '',
};
```

Редюсер погоди weatherReducer містить поля:

– cities – об’єкт, для збереження даних про місця та їх погоду;

– tripDays – масив днів подорожі, що містять в собі масив локацій;

– isLoading – булеве поле яке в подальшому дозволить показувати індикатор завантаження даних;

– error – текстове поле, що для збереження повідомлення помилки.

Створено прості thunk-и, для автоматичної генерації функцій action creator, і з додатковою логікою:

Для додавання нового дня подорожі - addDay. Фрагмент коду додатку Е:

```
export const addDay = createAsyncThunk(
  'weather/addDay',
  async (payload) => payload
);
```

Для видалення дня подорожі - deleteDay, що приймає аргументом індекс дня до видалення. Фрагмент коду додатку Е:

```
export const deleteDay = createAsyncThunk(
  'weather/deleteDay',
  async (dayIndex: number) => dayIndex
);
```

Для видалення локації з масиву дня подорожі – deleteLocatin, що приймає аргументом об’єкт з id місця та індексом необхідного дня. Фрагмент коду додатку Е:

```
export const deleteLocation = createAsyncThunk(
  'weather/deleteLocation',
  async (params: IDeleteLocation) => params
);
```

Для додавання локації в масив дня подорожі – addLocatin, що приймає аргументом об’єкт з індексом необхідного дня подорожі та об’єкт місця, що повертається у запиті пошуку. Фрагмент коду додатку Е:

```
export const addLocation = createAsyncThunk(
 'weather/addLocation',
 async ({ dayIndex, location }: IAddLocation, thunkAPI) => {
   const { lat, lon } = location;
   return OwmService.getForecast(lat, lon)
     .then((locationWeather) => {
       const locationWithWeather: IAddLocation['location'] = {
         ...location,
         weather: locationWeather,
       };
       const addLocationParams: IAddLocation = {
         dayIndex,
         location: locationWithWeather,
       };
       return addLocationParams;
     })
     .catch((e: AxiosError) => thunkAPI.rejectWithValue(e.message));
 }
);
```

код виконує запит для отримання прогнозу погоди за допомогою метода getForecast сервісу OmwService, передаючи в якості аргументів координати, при успішному отримання даних погоди, створюється новий об’єкт місця з полем «weather», що містить погодні дані. Функція поверта об’єкт, що містить в собі об’єкт місця та індекс дня подорожі. Якщо виникає помилка, то вона перехоплюється в стандартний спосіб.

Для видалення об’єкту міста з стану – deleteCity, що приймає аргумент cityId – рядок-id місця, що повинне бути видалене. Фрагмент коду додатку Е:

```
export const deleteCity = createAsyncThunk(
  'weather/deleteCity',
  async (cityId: IGeocodingIndexed['id'], { getState }) => {
```

```
        const { weatherReducer } = getState() as { weatherReducer:
WeatherState };
        const tripDayUsingCity = weatherReducer.tripDays.find((tripDay) =>
          tripDay.cityIds.includes(cityId)
        );
        if (!tripDayUsingCity) {
          return cityId;
        }
      }
    );
```

Цей thunk отримує з стану поле tripDays та перевірє, чи не використовується ця локація в якомусь з днів подорожі, і якщо вона непотрібна, то id місця передається далі.

Для оновлення погодних даних – updateCityWeather, що приймає аргументнт – об’єкт місця, для оновлення погодних даних. Фрагмент коду додатку Е:

```
    export const updateCityWeather = createAsyncThunk(
      'weather/updateCityWeather',
      async (cityToUpdate: IGeocodingIndexed, thunkAPI) => {
        const { lat, lon } = cityToUpdate;
        return OwmService.getForecast(lat, lon)
          .then((cityWeather) => {
            const updatedCity: IGeocodingIndexed = {
              ...cityToUpdate,
              weather: cityWeather,
            };
            return updatedCity;
          })
          .catch((e: AxiosError) => thunkAPI.rejectWithValue(e.message));
      }
    );
```

Callback функція thunk використовуючи кординати з об’єкту місця та метод getForecast сервсісу OwmService робить запит до api і в разі успішного його виконання поверне новий об’єкт місця з оновленими погодніми даними.

Для створення редюсеру погоди було використано вже знайому функцію createSlice(), ось фрагмент коду додатку Д що викликає її:

```
export const weatherSlice = createSlice({
  name: 'weather',
  initialState,
  reducers: {},
  extraReducers: (builder) => {
    builder
      .addCase(Actions.addLocation.pending, (state) => {
        state.isLoading = true;
      })
      .addCase(Actions.addLocation.fulfilled, (state, action) => {
        const { dayIndex, location } = action.payload;
        state.isLoading = false;
        state.error = '';

state.tripDays[dayIndex].cityIds.push(action.payload.location.id);
        state.cities[location.id] = location;
      })
      .addCase(Actions.addLocation.rejected, (state, action) => {
        state.isLoading = false;
        state.error = action.payload as string;
      })
      .addCase(Actions.updateCityWeather.pending, (state) => {
        state.isLoading = true;
      })
      .addCase(Actions.updateCityWeather.fulfilled, (state, action) => {
        state.isLoading = false;
        state.error = '';
        state.cities[action.payload.id] = action.payload;
      })
      .addCase(Actions.updateCityWeather.rejected, (state, action) => {
        state.isLoading = false;
        state.error = action.payload as string;
      })
      .addCase(Actions.deleteLocation.fulfilled, (state, action) => {
        state.tripDays[action.payload.dayIndex].cityIds =
state.tripDays[
          action.payload.dayIndex
        ].cityIds.filter((cityId) => cityId !== action.payload.cityId);
```

```
      })
      .addCase(Actions.addDay.fulfilled, (state) => {
        state.tripDays.push({ cityIds: [] });
      })
      .addCase(Actions.deleteDay.fulfilled, (state, action) => {
        state.tripDays.splice(action.payload, 1);
      })
      .addCase(Actions.deleteCity.fulfilled, (state, action) => {
        typeof action.payload === 'string' &&
          delete state.cities[action.payload];
      });
  },
});
```

У функції-редюсері для дії «addLocation.pending» полю стану isLoading присвоюється значення true, щоб відобразити стан завантаження.

У функції-редюсері для дії «addLocation.fulfilled» , щоб додати локацію:

- полю стану isLoading присвоюється значення false;
- полю стану error присвоюється значення порожнього рядка;
- в масив tripDays дня пошуку з індексом переданим в дії, додається значення id нової локаці;
- в об’єкті cities створюється поле з ключем – id локації, яке містить в собі об’єкт локації.

У функції-редюсері для дії «addLocation.rejected», щоб обробити помилку запиту:

- полю стану isLoading присвоюється значення false;
- полю стану error присвоюється рядок повідомлення помилки, що передається в дії.

У функції-редюсері для дії «updateCityWeather.pending» полю стану isLoading присвоюється значення true, щоб відобразити стан завантаження.

У функції-редюсері для дії «updateCityWeather.fulfilled» , щоб оновити погодні дані для локації:

- полю стану isLoading присвоюється значення false;

– полю стану error присвоюється значення порожнього рядка;
– в об’єкті cities оновлюється поле з ключем – id локації, значенням переданим у функції.

У функції-редюсері для дії «updateCityWeather.rejected», щоб обробити помилку запиту:

– полю стану isLoading присвоюється значення false;
– полю стану error присвоюється рядок повідомлення помилки, що передається в дії.

У функції-редюсері для дії «deleteLocation.fulfilled» , щоб оновити погодні дані для локації:

– полю стану isLoading присвоюється значення false;
– полю стану error присвоюється значення порожнього рядка;
– в об’єкті cities оновлюється поле з ключем – id локації, значенням переданим у функції.

У функції-редюсері для дії «addDay.fulfilled», щоб створити день походу, додається новий елемент до масиву tripDays.

У функції-редюсері для дії «deleteDay.fulfilled», щоб видалити день походу, з масиву tripDays вирізається елемент за індексом, що передається в дії.

У функції-редюсері для дії «deleteCity.fulfilled», видаляється поле об’єкту cities за ключем, що передається в дії.

Для здійснення HTTP-запитів в застосунку використовується пакет axios. Axios надає можливість створювати екземпляр з потрібною конфігурацією для упрощення роботи. Нижче наведений конфігураційний фрагмент коду, що створює новий екземпляр – OpwApiCore. В конфігураційному об’єкті вказана url-адреса яка буде основою для здійснення запитів до api. Фрагмент коду додатку Ж:

```
const OpwApiCore = axios.create({
  baseURL: OpwBaseUrl,
});
```

Також дуже зручною є можливість перехоплювати запити, нижче наведений фрагмент коду додатку Ж, що додає перехоплювач запиту до нового екземпляру:

```
OpwApiCore.interceptors.request.use((config) => {
  config.params.appid = OpwApiKey;
  return config;
});
```

Він являє собою callback-функцію, що єдиний параметром приймає конфігураційний об’єкт, змінює та повертає його. В цьому фрагменті функція додає до параметрів запиту поле appid, з рядковим значенням API-ключа необхідного для доступу до цих API.

На основі цього екземпляру axios було побудовано сервіс для взаємодії з API – OwmService. Фрагмент коду додатку И що описує сервіс:

```
class OwmService {
  private oneCallPath = '/data/2.5/onecall';
  private geocodingPath = '/geo/1.0/direct';
  private iconPath = 'http://openweathermap.org/img/wn/';
  async getForecast(
    lat: number,
    lon: number,
    config: IGetForecastConfig = { units: 'metric', lang: 'uk' }
  ) {
    return OpwApiCore.get<IOneCallForecast>(this.oneCallPath, {
      params: {
        lat,
        lon,
        ...config,
      },
    }).then((res) => res.data);
  }
  async searchCity(cityName: string, resultsLimit = 5) {
    return OpwApiCore.get<IGeocoding[]>(this.geocodingPath, {
      params: {
        q: cityName,
        limit: resultsLimit,
      },
```

```
      })
        .then((res) => {
          const resultWithId: IGeocodingIndexed[] = res.data.map((city) =>
({
            ...city,
            id: `${city.lat}${city.lon}`,
          }));
          return resultWithId;
        })
        .catch((e: AxiosError) => {
          throw e;
        });
    }
    getIconUrl(icon: string) {
      return `${this.iconPath}/${icon}@2x.png`;
    }
  }
```

Клас сервісу містить в собі три поля: oneCallPath – шлях по якому здійснюється запит для отримання погодних даних, geocodingPath – шлях для здійснення запитів для пошуку, та iconPath – url-адреса для отримання іконок погодних умов.

Метод getForecast потрібен для отримання погодних даних, приймає три параметри: lat – широта, lon – довгота, config – параметр зі значеннями за замовчуваннями, об’єкт містить поля «units» та «lang» що дозволяють вказувати одиниці вимірювання певних параметрів прогнозу погоди та мову результату. Метод повертає promise від виконная методу get з використанням екземпляру axios.

Метод searchCity для пошуку місця приймає два параметри: перший – назва місця для здійснення пошуку, другий – ліміт результатів, за замовчуванням вказано максимальний – 5. Запит виконується таким же чином, що і в методі getForecast, за вийнятком того що до кожного місця з результатів пошуку додається поле id, згенероване шляхом конкатинації строкового значення широти і довготи.

Метод getIconUrl, потрібний для отримання посилання на іконки, він приймає єдиний параметр – id іконки, яке просто підставляється в шаблонний рядок.

Висновок до розділу 2

В цьому розділі було вибрано засоби реалізації веб-додатку. Розроблено і розібрано функціонал застосунку, який відповідає за внутрішньо його роботу, та роботу з API.

3 РЕАЛІЗАЦІЯ ПРОЕКТУ

3.1 Структура проекту

Для реалізації проекту було обрано графічну javascript-бібліотеку React. Її особливістю є відсутність нав’язування строгої структури проекту, що дає свободу розробнику у його діях. На рисунку 3.1 зображена файлова структура проекту.

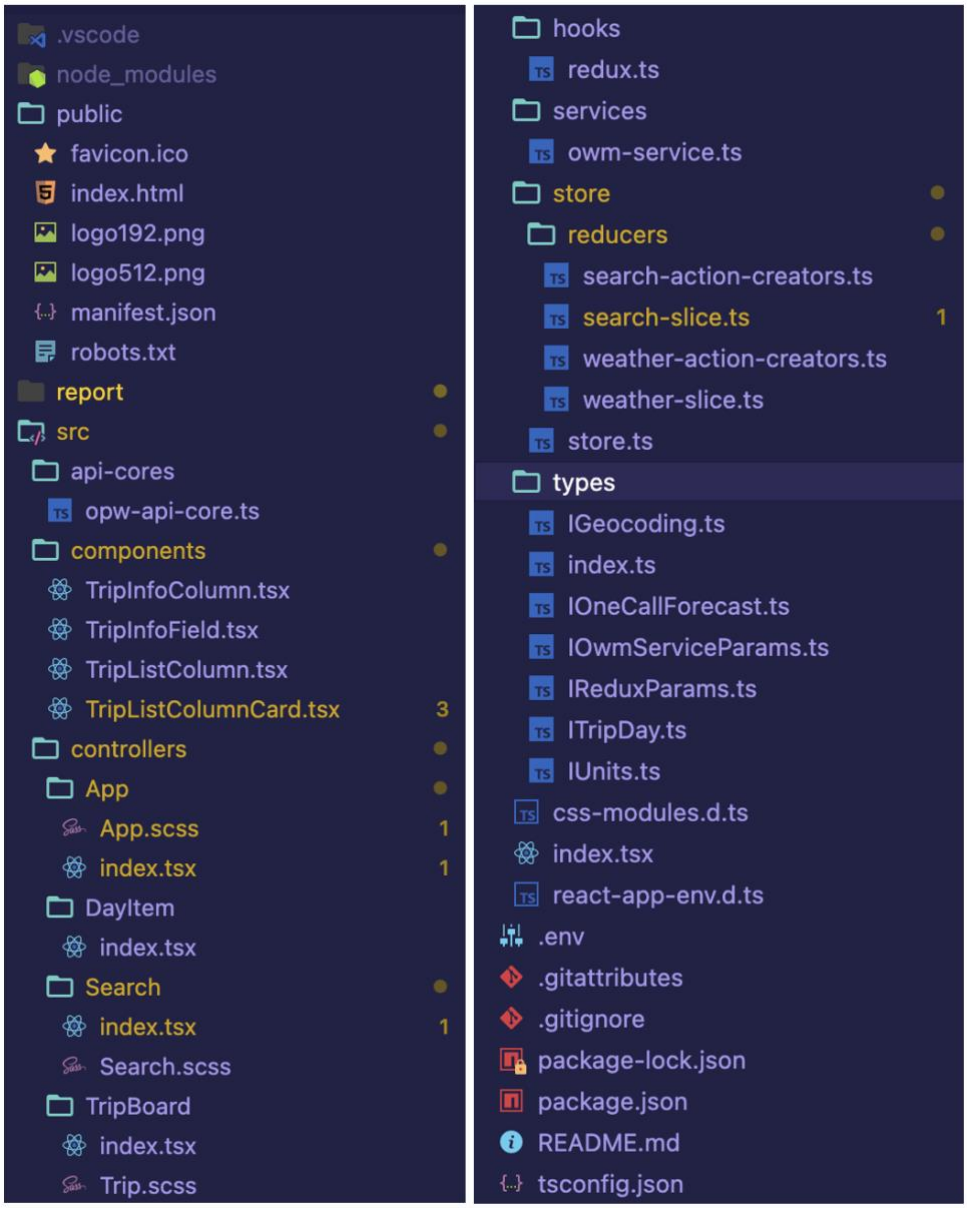

Рисунок 3.1 – структура файлів та папок веб-застосунку

В корені директорії проекту знаходяться папки і файли, що схожі для більшості додатків побудованих з використанням javascript та npm – пакетного менеджеру node.js:

- .vscode – прихована папка редактора коду VS Code, що зберігає локальні налаштування для робочого простору, і тд;
- node_modules – папка в яку завантажуються всі встановлені пакети;
- public – папка, що зберігає деякі статичні файли, в певній мірі, потрібні для роботи проекту;
- .env файл – потрібен для запису змінних середовища, записані змінні доступні в любому місці коду.
- .gitattributes – прихований файл, що містить певні атрибути системи контролю версій git;
- .gitignore – файл потрібний, щоб вказати папки, чи файли, які не мають потрапити в git(node_modules і тд.);
- package.json – файл, що містить в собі інформацію про проект, його залежності, скрипти, і тд.;
- README.md – файл у форматі markdown, у якому, зазвчиай, описуються деталі роботи з додатком;
- tsconfig.json – конфігураційний файл TypeScript.

В папці src міститься основний код додатку:

- в папці api-cores знаходить файл opw-api-core. Що містить в собі екземпляр axios налаштований для роботи з OpenWeatherMap API;
- в папці component знаходяться файли react-компонентів, які можна перевикористовувати, а саме: TripInfoColumn.tsx, TripInfoField.tsx, TripListColumn.tsx, TripListColumnCard.tsx;
- в папці controllers містяться папки react-компонентів, які містять логіку, представляють елементи сторінки, а саме: App, DayItem, Search, TripBoard. Кожна папка містить index.tsx файл, та за потреби scss файл для опису стилів компоненту;

- в папці hooks знаходиться файл redux.ts, що експортує типізовані хуки useAppDispatch та useAppSelector, для запуску дій, та отримання снату застосунку в компонентах;
- в папці services знаходиться файл що експортує OwmService, методи якого виконують запити до api;
- Папка store, з якої експортовується функція створення нового store-у, та деякі потрібні типи. Також там знаходиться папка reducers, в якій знаходяться файли аснихронних thunk та редюсерів;
- Папка types містить в собі файли інтрефейсів даних для упрощення обробки даних з API;
- файли опису модулів typescript (.d.ts);
- файл index.tsx – основний компонент, який підключається до блоку з id «root», та рендерить компонент головний App.

3.2 Реалізація візуальної частини фронт-енд додатку

Будь-який веб-застосунок, побудований з використанням react – відкритої графічної JavaScript-бібліотеки, починається з головного компоненту App, що містить в собі основні значущі компоненти.

Якщо розглядати тільки графічні компоненти, то можна помітити, що вони умовно поділяються на два типи: одні з них легкі, створені з думкою про їхнє перевикористання, та інші, які напряму позначають якісь елементи сторінки, мають в собі частину логіки застосунку.

Як вже було помітно з попереднього розділу, у розроблюваному веб-застосунку будь дві основні сутності: компонент пошуку - Search, та компонент з метеорологічними даними - TripBoard.

Оскільки було використано пакет material ui з деякими готовими компонентами, верстка детально розглядатися не буде

При відкритті сторінки нас зустрічає статична частина компоненту TripBoard - компонент TripListHeader (рисунок. 3.2)

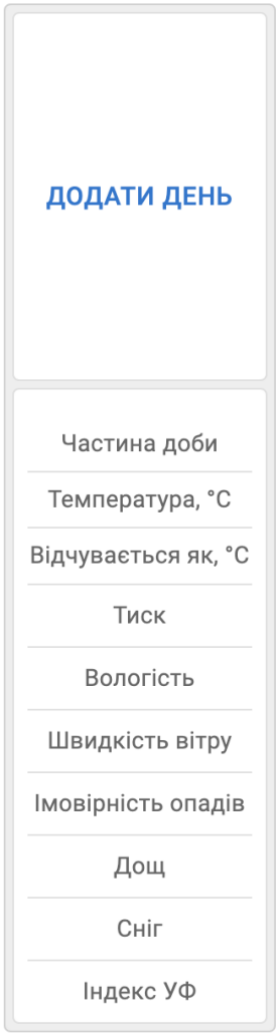

Рисунок 3.2 – Компонент TripListHeader

Як свідчить назва, компонент є свого роду заголовком таблиці, другий його блок містить список параметрів метеорологічних даних, які будуть відображатися при після того як будуть створені дні подорожі а перший – кнопку створення нового дня подорожі. Кнопка «додати день» при натисканні виконує функцію, що передається компоненту в пропсах. Ось фрагмент коду додатку К, що описує компонент TripListHeader:

```
      const TripListHeader = ({ addDayClickHandler }: TripListHeaderProps) =>
{
        return (
          <TripListColumn>
            <Stack spacing={0.5}>
              <Paper
                variant="outlined"
```

```
        sx={{
          display: 'flex',
          justifyContent: 'center',
          alignItems: 'center',
          p: 1,
          minHeight: '200px',
          backgroundColor: (theme) => theme.palette.common.white,
        }}
      >
        <Button
          size={'large'}
          sx={{ margin: 'auto' }}
          onClick={addDayClickHandler}
        >
          Додати день
        </Button>
      </Paper>

      <TripListColumnCard maxHeight>
        <Stack spacing={0.85} divider={<Divider sx={{ width: '100%' }}
/>}>
          <Stack
            spacing={0.75}
            component={'ul'}
            divider={<Divider sx={{ width: '100%' }} />}
            sx={{
              width: '100%',
              p: 0,
              m: 0,
              alignItems: 'center',
              marginTop: '0.86rem',
            }}
          >
            <TripInfoField>
              <Typography
                variant={'body2'}
                sx={{ color: (theme) => theme.palette.text.secondary
}}
```

```
                >
                  Частина доби
                </Typography>
              </TripInfoField>
              <TripInfoField>
                <Typography
                  variant={'body2'}
                  sx={{ color: (theme) => theme.palette.text.secondary
}}
                >
                  Температура, &deg;C
                </Typography>
              </TripInfoField>
              <TripInfoField>
                <Typography
                  variant={'body2'}
                  sx={{ color: (theme) => theme.palette.text.secondary
}}
                >
                  {' '}
                  Відчувається як, &deg;C{' '}
                </Typography>
              </TripInfoField>
            </Stack>
            <Stack
              component={'ul'}
              spacing={1}
              divider={<Divider sx={{ width: '100%' }} />}
              sx={{
                width: '100%',
                p: 0,
                m: 0,
                alignItems: 'center',
                paddingTop: '0.07rem',
              }}
            >
              <TripInfoField>
                <Typography
```

```
                        variant={'body2'}
                        sx={{ color: (theme) => theme.palette.text.secondary
}}
                      >
                        {' '}
                        Тиск
                      </Typography>
                      ...
                    </TripInfoField>
                  </Stack>
                </Stack>
              </TripListColumnCard>
            </Stack>
          </TripListColumn>
        )};
```

В свою чергу компонент TripBoard містить в собі функцію обробки нажаття, котра передається пропсом в компонент TripListHeader, і при кліку відправляє thunk додавання дню. Використовуючи хук useAppSelector компонент отримує зі стану застосунку масив днів подорожей – tripDays, який за допомогою ітератора map перетворює на масив компонентів DayItem. Фрагмент коду додатку Л:

```
      const TripBoard = (props: TripBoardProps) => {
        const dispatch = useAppDispatch();
        const { tripDays } = useAppSelector((state) => state.weatherReducer);
        const addDayClickHandler: React.MouseEventHandler<HTMLButtonElement> =
(
          e
        ) => {
          dispatch(addDay());
        };
        return (
          <>
            <Stack direction="row" spacing={0.5}>
              <TripListHeader addDayClickHandler={addDayClickHandler} />
              {tripDays.map((tripDay, tripDayIndex) => (
                <DayItem tripDayIndex={tripDayIndex} key={tripDayIndex} />
              ))}
```

```
      </Stack>
    </>
  );};
```

При Натисканні на кнопку «Додати день», в стані програми додається новий елемент масиву tripDay, і на екрані з’являється компонент DayItem(рисунок 3.3)

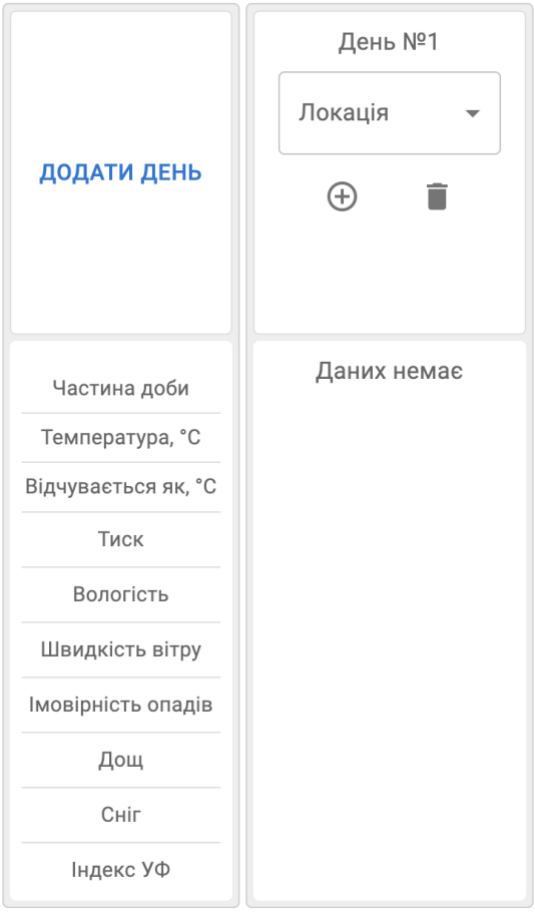

Рисунок 3.3 – Порожній компонент DayIndex

Компонент DayIndex, лістинг його коду наведено в додатку М також складається з двох блоків. В нижньому, поки не обрано локацію для відображення погоди, відображається напис «немає даних про погоду». У верхньому відображається поле вибору локації, кнопка додавання локації, та видалення. Коли список доступних локацій порожній, кнопка видаляє день подорожі.

При натисканні компонент за допомогою хука dispatch відправляє thunk, який вмикає відображення поля пошуку(рисунок 3.4)

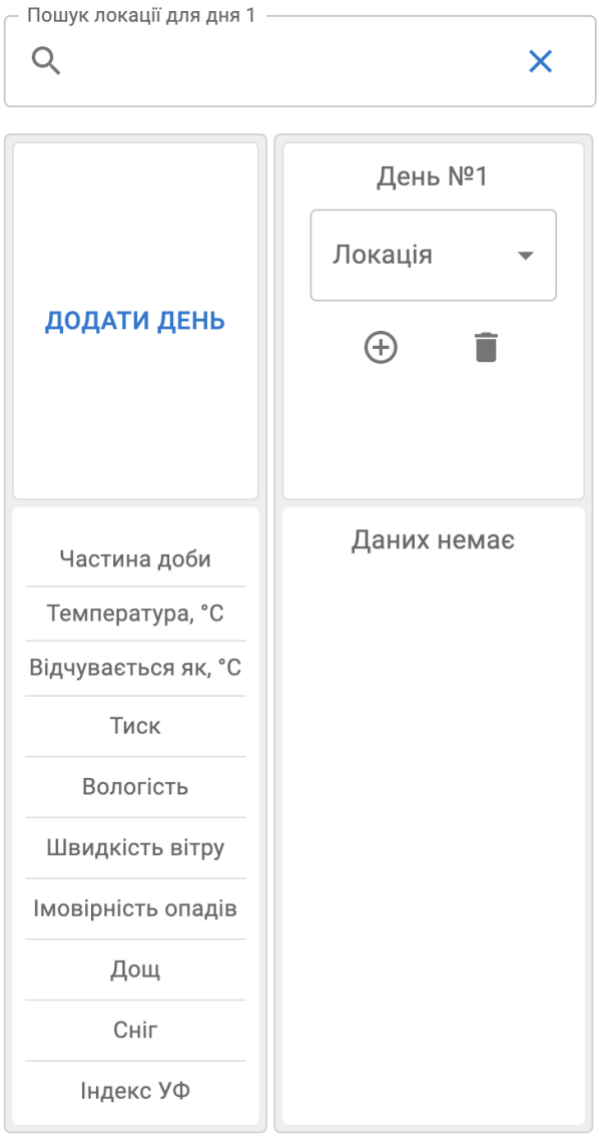

Рисунок 3.4 – Поява компоненту

Поле пошуку має надпис який вказує, для якого дня подорожі здійснюється пошук, а також конпоку для закриття пошуку. При введені тексту в поле пошуку з’являється список результатів(рисунок 3.5).

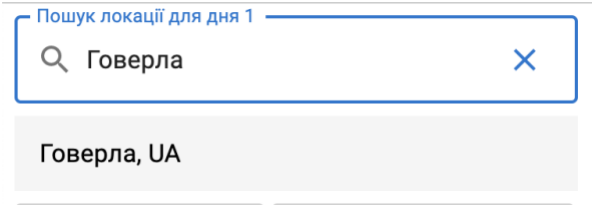

Рисунок 3.5– Результат пошуку

При кліку на один з ним він заноситься зберігається для вказаного дня подорожі. Ось фрагмент коду додатку Н, описує компонент пошуку:

```
const Search = (props: SearchProps) => {
  const [searchInput, setSearchInput] = useState('');
  const dispatch = useAppDispatch();
  const { citySearchResult, isLoading, error, destinationDay, showSearch
} =
    useAppSelector((state) => state.searchReducer);
  const onChange: React.ChangeEventHandler<HTMLInputElement> = (e) => {
    const value = e.currentTarget.value;
    setSearchInput(value);
    const isEmpty = value.trim() === '';
    if (!isEmpty) {
      dispatch(fetchSearchCity(value));
    } else {
      dispatch(clearSearchResult());
    }
  };
  const onClickClose: React.MouseEventHandler<HTMLButtonElement> = (e)
=> {
    dispatch(hideSearch());
    setSearchInput('');
  };
  const onClickLocation = async (
    e: React.MouseEvent<HTMLDivElement>,
    location: IGeocodingIndexed
  ) => {
    await dispatch(addLocation({ dayIndex: destinationDay, location }));
    dispatch(hideSearch());
  };
  if (!showSearch) {
    return <></>;
  }
  return (
    <Box sx={{ width: '100%', maxWidth: 360 }}>
      <TextField
        sx={{ width: '100%' }}
        label={`Пошук локації для дня ${destinationDay + 1}`}
        variant="outlined"
        InputProps={{
```

```
            startAdornment: (
              <InputAdornment position="start">
                <SearchIcon />
              </InputAdornment>
            ),
            endAdornment: (
              <InputAdornment position="end">
                <IconButton
                  color="primary"
                  aria-label="close"
                  onClick={onClickClose}
                >
                  <CloseIcon />
                </IconButton>
              </InputAdornment>
            ),
          }}
          value={searchInput}
          onChange={onChange}
        />
        <nav aria-label="secondary mailbox folders">
          <List>
            {!error &&
              citySearchResult.map((city) => (
                <ListItem disablePadding key={city.id}>
                  <ListItemButton onClick={(e) => onClickLocation(e,
city)}>
                    <ListItemText
                      primary={`${city?.local_names?.uk || city.name}, ${
                        city.country
                      }`}
                    />
                  </ListItemButton>
                </ListItem>
              ))}
          </List>
        </nav>
      </Box>
```

```
    );
};
```

Компонент пошуку за допомогою хуку отримує всі поля редюсера пошуку. Функція, що опрацьовує нажаття кнопки закриття пошуку використовує thunk «hideSearch». Функція, що опрацьовує нажаття на результат пошуку використовує thunk-и «addLocation» для додавання локаціїї та «hideSearch» для закриття пошуку. Поле пошуку є звичайним керованим полем, коли для збереження його значення в компоненті піднімається стан,/ а функція що опрацьовує зміни – просто оновлює його.

Після додавання локації компонент DayItem починає відображати погодні дані(рисунок 3.6), у верхньому блоці з'являється іконка стану погодних умов та мінімальна, максимальна температура завказану добу.

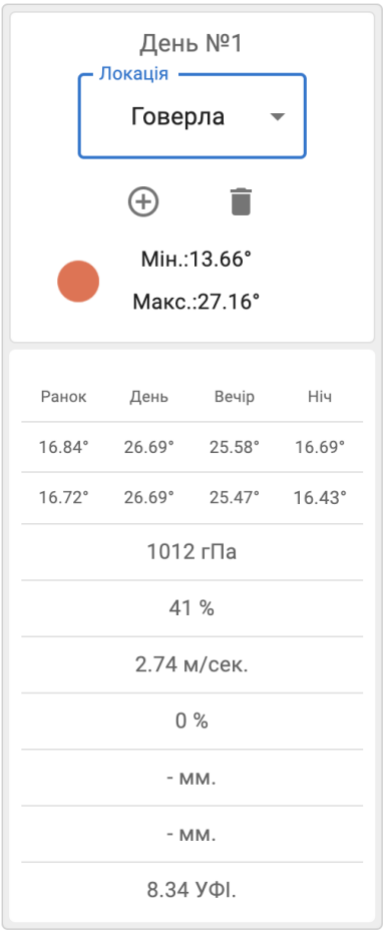

Рисунок 3.6– Компонент DayItem з доданою локацією

Компонент DayItem отримує пропсами індекс дня подорожі. Використовуючи хук, він отримує зі стану об'єкт дня подорожі, та cities - об'єкт зі всіма містами.

Поле вибору локації також є керованим полем, що дозволяє легко отримати доступ до потрібної погоди.

Таким чином веб-додаток буде мати вигляд зображений на рисунку 3.7

Рисунок 3.7 – Сторінка веб-додатку

Висновок до розділу 3

В розділі 3 було реалізовано веб-додаток вузуалізації зведених метеорологічних даних для планування туристичних походів з використанням OpenWeatherMap API, описано файлову структуру проекту та роботу графічної частини застосунку.

ВИСНОВКИ

В даній бакалаврській роботі було розроблено веб-додаток візуалізації зведених метеорологічних даних для планування туристичних походів з використанням OpenWeatherMap API.

В першому розділі було здійснено аналіз предмету дослідження, здійснено аналіз схожих сервісів та сформовано задачу бакалаврської роботи.

В другому розділі було В цьому розділі було вибрано засоби реалізації веб-додатку. Розроблено і розібрано функціонал застосунку, який відповідає за внутрішньо його роботу, та роботу з API.

В третьому розділі було реалізовано веб-додаток вузуалізації зведених метеорологічних даних для планування туристичних походів з використанням OpenWeatherMap API, описано файлову структуру проекту та роботу графічної частини застосунку.

В четвертому розділі було проаналізовано вимоги охорони праці щодо роботи з ПК під час розробки веб-додатку візуалізації зведених метеорологічних даних для планування туристичних походів з використанням OpenWeatherMap API в офісі IT-компанії та розроблено заходи, щодо забезпечення відповідних умов праці. Проведно розрахунок надлишків тепла. За його результатами в приміщенні рекомендовано встановити кондиціонер, що здатний забезпечити повітрообмін з 575 $м^3$/год.