КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІМЕНІ ТАРАСА ШЕВЧЕНКА

ФАКУЛЬТЕТ РАДІОФІЗИКИ, ЕЛЕКТРОНІКИ ТА КОМП'ЮТЕРНИХ СИСТЕМ

Кафедра комп'ютерної інженерії

До захисту допущено: «На правах рукопису»
Завідувач кафедри _____ Юрій Бойко
« _ » _____ 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему:

«РОЗРОБКА ЗАСТОСУНКУ ДЛЯ ПЕРЕГЛЯДУ ПРОГНОЗУ ПОГОДИ З НАХИЛОМ НА ПОКРАЩЕННЯ ПЛАНУВАННЯ РОЗПОРЯДКУ ДНЯ НА ОСНОВІ ОТРИМАНИХ ДАНИХ»

Виконав:
студент 4-го курсу бакалаврату
денної форми навчання
спеціальності 123 Комп'ютерна інженерія
ОНП «_____»
Тарасюк Василь Олександрович _____

Науковий керівник:
кандидат фізико-математичних наук, асистент
Іваненко Дмитро Олександрович _____

Рецензент:
_____

Засвідчую, що у цій бакалаврській роботі
немає запозичень з праць інших авторів без
відповідних посилань
Студент _____

Робота допущена до захисту в ЕК рішенням кафедри _____
від «__»_____ 2023 р., протокол № __.

Завідувач кафедри _____,
кандидат фізико-математичних наук, доцент
Бойко Юрій Володимирович

(підпис)

Київ – 2023

РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра: 52 с., 21 рис., 1 табл., 10 джерел, 2 додатка.

Об’єктом роботи є процес розробки веб-застосунку для ознайомлення з даними про прогноз погоди за переліком параметрів з додатковим функціоналом створення календаря Google Calendar[7] з погодними даними однією кнопкою, детальною характеристикою даних про вітер, що допоможе планувати польоти дронів та проводити метереологічні дослідження, та публікація застосунку для надання доступу до нього користувачам.

Метою роботи було отримання робочого продукту, чий функціонал допоможе покращити користувацький досвід при взаємодії з запропонованим погодним ресурсом, а також надасть можливість гнучкого вибору даних при створенні календаря та спростить його створення у контексті його планування з погодними даними.

Етапами реалізації продукту є:

- Дизайн застосунку з метою покращення UI та UX складових, що забезпечить користувача можливість швидко та зручно орієнтуватись в застосунку,
- Написання коду застосунку з використанням зазначених технологій та інструментів,
- Публікація застосунку.

Результатом роботи очікується робоча версія застосунку, що знаходиться у вільному доступі та зручна для використання на персональних комп’ютерах та смартфонах, що збільшить потенційну аудиторію.

GOOGLE CALENDAR, SUPABASE, WEATHER, HTML, CSS, REACT, NODE.js, NPM, API, SPA, UX, UI, VSCODE, GIT.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ І ПОЗНАЧЕНЬ

HTML – HyperText Markup Language

CSS – Cascading Style Sheets

JSX – JavaScript Syntax Extension

NPM – Nutty Peanut Marshmallow

API – Application Programming Interface

SPA – Single Page Application

UX – User Experience

UI – User Interface

ВСТУП

В наш час, коли кожного дня люди мають потребу використовувати засоби для організації свого дня, є необхідність в створенні гнучких та зручних застосунків для покращення планування. IT гіганти постійно покращують та регулярно реалізовують нові додатки з цією метою. Посилаючись на цю потребу було створено веб-застосунок, який допоможе полегшити процес планування дня з урахуванням специфічних даних для пересічного користувача, а саме сайт, що надає доступ до даних прогнозу погоди та можливість створювати календар з такими даними з допомогою сервісу Google Calendar простим та швидким способом.

Розроблений застосунок надає функціонал для доступу до даних, що надаються вибраними API, та реалізує його у вигляді сучасного та динамічного інтерфейсу з дотриманням кращих практик розробки клієнтської частини застосунку. До оголошеного функціоналу входять наступні можливості: отримання та перегляд даних прогнозу погоди з використанням запропонованого сайтом інтерфейсу, отримання та перегляд прогнозу погоди з використанням сервісу Google Calendar, отримання детальної інформації про вітер з використанням графіків, що допоможе пересічному користувачу краще орієнтуватись в змінах погоди.

Для реалізації застосунку виділено наступні етапи:

- Створення дизайну застосунку з використанням програмного забезпечення Figma,

- Створення та дизайн компонентів клієнтської частини застосунку з використанням мови JavaScript, бібліотеки React, XML-подібного синтаксису JSX та сторонніх бібліотек,

- Реалізація можливості отримання даних з використанням вибраного API, що надає доступ до даних при надсиланні чітко визначених запитів користувачем,

- Створення та реалізація серверної логіки з допомогою сервісів Supabase[8] та Google Console.

Застосунок створюється з метою покращення користувацького досвіду взаємодії з вказаним API[6] шляхом створення гнучкого та зрозумілого інтерфейсу та зменшення часу, який вимагають дії, що виконуються при створенні аналогічних календарів в сервісі Google Calendar. Замість того щоб створювати календар та вручну вносити в нього необхідні дані, користувач

може зробити це лиш однією кнопкою з використанням створеного застосунку. Також існує можливість гнучкого вибору даних та періоду, за який ці дані будуть актуальними, в зручному інтерфейсі.

Спостереження за погодою має декілька важливих аспектів, які впливають на різні сфери нашого життя:

- Безпека. Спостереження за погодою допомагає попереджати про погіршення погодних умов, таких як шторми, повені, торнадо, сильні вітри, снігопади тощо, що можуть загрожувати безпеці людей.

- Здоров'я. Погода може впливати на наше фізичне та психічне здоров'я. Наприклад, температура та вологість повітря можуть впливати на відчуття комфорту людини та її здоров'я, а також можуть викликати алергічні реакції.

- Економіка. Погода може впливати на різні економічні сектори, такі як сільське господарство, транспорт, туризм, енергетика тощо. Наприклад, негода може призвести до затримок у роботі транспорту або до пошкодження посівів.

- Розваги. Багато людей планують свій відпочинок та дозвілля залежно від погодних умов. Спостереження за погодою допомагає зробити правильний вибір, коли мова йде про відвідування парків, пляжів, гірських курортів тощо.

Отже, хоча на ринку може бути багато сайтів з прогнозом погоди, спостереження за погодою є важливим інструментом для багатьох аспектів нашого життя, тому є цінним інструментом як для простих користувачів, так і для розробників.

Справедливо зазначити, що сайтів з прогнозом погоди безліч. Тому я наведу ряд підстав для актуальності мого рішення, а саме:

- Точність і детальність прогнозу. Кожен сайт використовує власні алгоритми і джерела даних для прогнозування погоди. Новий сайт може використовувати інші алгоритми та джерела даних, що можуть покращити точність та актуальність прогнозу для користувачів. В випадку з моїм сайтом детальність прогнозу забезпечує низьку похибку при відображенні даних (До 2°C), якщо брати до уваги дані, що приходять за координатами користувача. Такий висновок я зробив після спостереження за результатами протягом тижня. Відповідну таблицю з повними результатами моїх власних спостережень я навів нижче:

	22.03	23.03	24.03	25.03	26.03	27.03	28.03
Мінімальна прогнозована температура (°)	5	5	9	11	7	5	-1
Мінімальна реальна температура (°)	4	3	7	10	7	5	0
Максимальна прогнозована температура (°)	13	17	17	17	15	16	9
Максимальна реальна температура (°)	12	16	17	17	16	15	10

- Нові функції. Запропонований сайт має додаткові функції, які не доступні на інших сайтах, наприклад, можливість налаштування Google Calendar про зміну погоди, графіки зміни поведінки вітру.

- Конкуренція. Конкуренція між різними сайтами спонукає розробників до покращення якості своїх продуктів та постійного розвитку, що може покращити користувачеві досвід використання погодних сайтів.

РОЗДІЛ 1. ПОСТАНОВКА ЗАДАЧІ

Метою роботи є розробка застосунку для зручного доступу до детальних даних прогнозу погоди та швидкого створення календарів на сервісі Google Calendar. Створений застосунок має за мету спростити процедуру перегляду погоди з використанням інструменту Google Calendar шляхом створення відповідного календаря однією кнопкою.

Для реалізації застосунку необхідно виконати наступні кроки:

- Вибрати інструменти та мови розробки.
- Створити макет дизайну застосунку.
- Спроєктувати архітектуру програмного рішення.
- Реалізувати застосунок.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ЗАСТОСУНКУ

2.1 Вибір інструментів реалізації

2.1.1 FIGMA

Figma - це інноваційний інструмент для створення дизайнів та прототипування, який дозволяє створювати якісні дизайни для веб-застосунків, мобільні додатки та інтерфейси користувача з великою швидкістю та ефективністю. Його відмінність полягає у тому, що це хмарна програма, що працює в онлайн-режимі, що дозволяє дизайнерам працювати з будь-якого місця з доступом до Інтернету та спільно працювати з командою в режимі реального часу.

Figma надає безліч корисних інструментів, що допомагають з легкістю керувати ко дозволяють протестувати дизайн перед його впровадженням в реальний продукт.

Я вибрав цей інструмент через його інтуїтивну зручність, наявність необхідного функціоналу та попередній досвід роботи з ним.

2.1.2 VSCODE

Visual Studio Code - це безкоштовний редактор коду, який розроблений компанією Microsoft для роботи зі сучасними мовами програмування. Він має високу продуктивність, широкий функціонал і дуже добре підходить для розробки веб-додатків, мобільних додатків, ігор та інших програмних проектів. VSCode має дружній та простий інтерфейс, навігація та редагування коду здійснюються швидко та зручно. Він також має вбудовані можливості для версіонування коду, відлагодження програм, рефакторингу коду та автоматичної підстановки коду. Завдяки плагінам і розширенням, VSCode можна налаштувати під власні потреби, що робить його дуже гнучким та масштабованим редактором коду.

Також варто додати про підтримку Git. VSCode має вбудовану підтримку Git, що дозволяє розробникам працювати з репозиторіями без необхідності встановлювати додаткові плагіни або програми.

2.1.3 GITHUB

GitHub - це веб-платформа для спільного зберігання та управління кодом програмного забезпечення. Він дозволяє розробникам зберігати свій код в централізованому репозиторії та ділитися ним з колегами та співробітниками по всьому світу.

GitHub має безліч корисних функцій, таких як можливість редагувати та покращувати код за допомогою підтримки доступного ряду різноманітних мов програмування, створення гілок (branches) для розвитку коду окремо від головної гілки (master), створення проблем (issues) та пропозицій (pull requests) для спілкування та збору згоди з колегами, а також можливість автоматичного тестування коду перед злиттям з основною гілкою.

Крім того, GitHub має величезне співтовариство розробників, яке надає можливість взаємодіяти, навчатися та допомагати одне одному, а також розробляти відкриті проекти, що сприяють подальшому розвитку індустрії розробки та програмування.

2.1.4 NODE

Node.JS[1] – це платформа, що дає можливість JavaScript реалізуватись в якості мови загального призначення. Цю платформу називають середовищем виконання JS. Вона вміє пов'язувати код з зовнішніми бібліотеками, виконувати роль вебсервера та викликати команди з коду.

Сьогодні Node вважається однією з головних платформ для веб-розробки, а більшість вебінструментів, серверних та клієнтських, працюють саме з допомогою цього інструменту.

Основні особливості Node.js:

- Асинхронність - Node.js заснований на подієво-орієнтованому підході, який дозволяє виконувати багато операцій асинхронно, тобто без очікування завершення попередньої операції.

- Модульність - Node.js дозволяє використовувати модулі, які можна встановлювати з бібліотек NPM[2] (Node Package Manager). Це дозволяє розробникам зручно і швидко створювати різноманітні функції і додатки.

- Підтримка різних протоколів - Node.js підтримує різні протоколи, такі як HTTP, HTTPS, TCP, UDP, DNS, і т. д. Це дозволяє створювати різноманітні додатки, такі як веб-сервери, чати, онлайн-ігри і т. д.

- Висока продуктивність - Node.js забезпечує високу продуктивність завдяки використанню асинхронного виконання коду, що дозволяє оптимізувати використання процесорних ресурсів.

- Платформонезалежність - Node.js може працювати на різних операційних системах, таких як Windows, Linux і MacOS, що дозволяє розробникам працювати з додатками на різних платформах.

- Легкість в освоєнні - Node.js використовує мову JavaScript, яка є однією з найбільш популярних мов програмування. Це дозволяє розробникам, які вже володіють JavaScript, швидко освоювати Node.js.

Я вибрав цю платформу через ефективно реалізовану асинхронність та популярність серед розробників.

2.1.5 REACT

React[3] - це бібліотека JavaScript для створення користувацьких інтерфейсів, що використовується для розробки високопродуктивних та масштабованих веб-додатків. Вона базується на ідеї компонентного підходу до розробки, де кожен елемент інтерфейсу є окремим компонентом, який може бути повторно використаний та взаємодіяти з іншими компонентами.

Завдяки використанню Virtual DOM, React забезпечує ефективне оновлення елементів інтерфейсу, що робить його дуже швидким та потужним. Більшість компонентів React можуть бути написані з використанням JSX - розширення синтаксису JavaScript, що дозволяє змішувати HTML та JavaScript в одному файлі.

Я вибрав цю бібліотеку для розробки фронтенд частини застосунку через свою ефективність та потужність. Він використовується компаніями всіх розмірів, від стартапів до великих корпорацій, і є одним з найбільш популярних інструментів у веб-розробці сьогодні.

2.1.6 JSX

JSX (JavaScript XML) - це розширення язика JavaScript, яке дозволяє описувати структуру користувацького інтерфейсу за допомогою синтаксису,

схожого на HTML або XML. JSX дозволяє комбінувати JavaScript та HTML в одному файлі, що робить його використання зручним та ефективним для розробки веб-додатків. JSX дозволяє легко створювати компоненти та переиспользовать їх в різних місцях проекту, що допомагає підвищувати швидкість розробки та зберігати код більш організованим та читабельним.

Я віддав перевагу цьому синтаксису через його лаконічність та більшу швидкість написання коду.

2.1.7 API

API, або інтерфейс програмного забезпечення, - це набір протоколів, механізмів та інструментів, що дозволяють різним програмним додаткам взаємодіяти між собою. API виступає як посередник між різними додатками, який дозволяє передавати дані, отримувати результати запитів та забезпечувати безпеку обміну інформацією. API є ключовою складовою сучасної веб-розробки та дозволяє розширювати можливості програм та створювати багатофункціональні додатки з великою кількістю інтегрованих сервісів.

Я використовую API для взаємодії клієнтської частини з серверними частинами для отримання даних про погоду та надсилання даних про календар.

2.1.8 STYLED COMPONENTS

Styled Components[4] - це бібліотека для React, що дозволяє писати CSS-стилі в декларативному стилі, використовуючи JavaScript-код. Замість того, щоб створювати окремі файли CSS, з Styled Components ви можете створювати компоненти, які містять CSS-стилі, які використовуються в додатку. Це дозволяє писати більш чистий, перевірений та організований код, який є більш легким у розумінні та підтримці для великих застосунків.

Використовуючи Styled Components, ви можете створювати динамічні стилі, які змінюються на основі пропсів компонентів, ізольовані стилі для певних компонентів, а також перевизначати стилі зовнішніх компонентів.

Я вибрав цю бібліотеку тому, що вона робить написання коду стилів інтуїтивно зручним в навігації та написанні в контексті наступного внесення правок.

2.1.9 Recharts

Recharts[5] - це бібліотека для візуалізації даних в React, яка дозволяє легко створювати привабливі графіки та діаграми. Вона має простий та інтуїтивно зрозумілий інтерфейс, що дозволяє швидко та ефективно створювати візуалізації даних для будь-якої потреби. Recharts також підтримує різні типи графіків та діаграм, такі як лінійні графіки, кругові діаграми, стовпчикові діаграми та інші, та має можливість налаштування багатьох параметрів відображення, що дозволяє створювати унікальні та привабливі візуалізації.

Я вибрав цю бібліотеку через її поширеність, що гарантує постійну її підтримку та актуальність наявного функціоналу.

2.1.10 Supabase

Supabase - це інструмент для створення і розгортання баз даних та серверних додатків з відкритим вихідним кодом. Він забезпечує легкий доступ до баз даних та платформи для розробки додатків з використанням відкритих стандартів. Supabase пропонує безкоштовний та відкритий стек з масштабованими інструментами для збереження та збору даних, що дозволяє розробникам ефективно створювати додатки різного рівня складності.

Я вибрав даний інструмент через наявний необхідний функціонал для роботи з серверною складовою і зручність та швидкість розробки, що забезпечується зрозумілим інтерфейсом та документацією.

2.2 Принципи та кроки проєктування застосунку

Планування процесу розробки сайту та розбиття його на етапи є важливими елементами ефективного та успішного процесу розробки. Планування допомагає уникнути непередбачених проблем та помилок, які можуть з'явитися в процесі розробки, таким чином зменшуючи ризики для проекту. Розбиття процесу на етапи дозволяє краще керувати часом та ресурсами, а також забезпечує більшу організованість в процесі розробки.

Весь процес створення застосунку я розділив на етапи, яких дотримувався з метою отримання якнайкращого результату.

2.2.1 Розробка макету

Створення макету (або дизайну) сайту перед початком розробки є дуже важливою складовою процесу створення веб-додатку або сайту. Макет - це візуальне представлення майбутнього веб-додатку, яке містить елементи дизайну, кольори, шрифти, логотипи, та інші деталі. Це забезпечує краще розуміння взаємодії між структурою компонентів. Для дизайну я вибрав застосунок Figma.

Основні переваги створення макету перед написанням коду наступні:

- Візуалізація - макет дозволяє візуально уявити, як буде виглядати майбутній веб-додаток або сайт. Це дає змогу побачити, як будуть розташовані елементи, які кольори і шрифти будуть використані, які будуть кнопки та інші елементи інтерфейсу.

- Оптимізація часу - макет дозволяє зменшити час розробки, оскільки перед тим як розроблювати код, дизайнер та розробник можуть спільно обговорити ідеї та побажання щодо дизайну, а також вирішити будь-які питання з погляду функціоналу.

- Економія коштів - в разі, якщо зміни дизайну потрібні на пізнішому етапі розробки, це може коштувати значно більше часу і коштів. Також, якщо макет не підходить клієнту або команді розробників, його можна виправити на етапі розробки макету без зайвих витрат.

- Якість продукту - макет дозволяє зосередитись на деталях дизайну, а також на тому, як користувачі будуть взаємодіяти з веб-додатком або сайтом. Це допомагає створити продукт високої якості, що забезпечує задоволення користувачів та розвиток бізнесу.

2.2.2 Розбиття інтерфейсу на компоненти

Компонентний підхід - це основний підхід у React, який дозволяє розбити веб-додаток на невеликі незалежні компоненти. Кожен компонент може бути

створений як окремий елемент і включає в себе HTML-код, стилі, логіку та інші компоненти.

Основні переваги компонентного підходу у React наступні:

- Перевикористання коду - кожен компонент може бути використаний в іншому місці веб-додатку, що дозволяє значно зменшити кількість дублюючого коду та збільшити швидкість розробки.

- Спрощення розробки - компонентний підхід дозволяє розбити веб-додаток на невеликі елементи, що полегшує розуміння логіки та зменшує ризик появи помилок.

- Керованість станом - кожен компонент може мати свій власний стан, який дозволяє легко відстежувати зміни та керувати ними.

- Відокремленість - кожен компонент може мати свої власні стилі та логіку, що дозволяє відокремити їх один від одного та робити проект більш модульним.

- Зручність для тестування - компоненти можуть бути протестовані окремо, що спрощує виявлення помилок та їх виправлення.

- Загалом, компонентний підхід у React дозволяє створити високоякісний веб-додаток з великою кількістю функцій, швидко та зручно розширювати його та зберігати код чистим та зрозумілим.

Виходячи з цієї інформації я прийняв рішення дотримуватись саме такого підходу. На даному етапі спершу необхідно вибрати критерії, за якими ми будемо об’єднувати елементи в компоненти та організовувати структуру проєкту. Який з підходів групування краще вибрати? Однозначної відповіді на це питання не існує, але є кілька популярних методів. Давайте їх розглянемо та виберемо той, що задовольняє конкретний проєкт.

Підхід 1: Групування за функціональністю або маршрутом

Один з найпопулярніших підходів — це розміщення файлів CSS, JS і тестів у папках, згрупованих за функціональністю або маршрутом. Даний підхід не є універсальним, тому вибір рівня деталізації залишається за мною.

Підхід 2: Групування за типом файлу

Також існує спосіб структурування проектів за схожими файлами. Існує думка, що є сенс йти ще далі і розміщувати компоненти в різні папки в

залежності від їх ролі в додатку. Яскравий приклад - методологія розробки Atomic Design.

На практиці проекти часто використовують поєднання вищезгаданих підходів. Для своєї роботи я вирішив вибрати другий підхід з додатковим створенням окремих папок під кожен компонент. З досвіду можу зазначити, що це спростить навігацію між компонентами в редакторі коду під час розробки.

Перше, що я зробив, це розділив застосунок на компоненти. Кожному компоненту дав власну назву відповідно до того, за що цей компонент відповідає. Далі потрібно вибрати які саме частини сторінки будуть окремими компонентами. Для цього я застосував принцип єдиної функції – для кожного елементу буде конкретне завдання, яке він має виконувати. Якщо в компонента з'являвся додатковий функціонал, цей компонент було розбито на кілька дрібніших, або вкладено в нього додаткові підкомпоненти.

Наступним кроком я розташую визначені компоненти в порядку підпорядкованості для збереження ієрархії відносно батьківських та дочірніх елементів.

2.2.3 Вибір типу компонентів

У React є два основних підходи до оголошення компонентів - класові та функціональні. Раніше, до введення хуків у версії React 16.8, класові компоненти використовувалися як основний підхід до роботи зі станом, а функціональні - як більш прості та зручні компоненти для відображення інформації.

Класові компоненти відмінюються від функціональних тим, що вони мають внутрішній стан, який можна оголосити за допомогою методу constructor та this.state. Класові компоненти також можуть мати методи життєвого циклу (наприклад, componentDidMount), які дозволяють виконувати певні дії після того, як компонент з'явився на сторінці. Однак, класові компоненти потребують більше коду та менш зрозумілі для початківців, а також потенційно можуть призводити до складності в майбутньому.

Функціональні компоненти, з іншого боку, є простими та зрозумілими для початківців компонентами, які дозволяють відображати інформацію на сторінці та взаємодіяти з нею. Вони не мають внутрішнього стану та методів життєвого циклу, але з'явився новий підхід до роботи зі станом - хуки. Хуки, такі як useState та useEffect, дозволяють функціональним компонентам працювати зі

станом та ефектами, що зробило їх більш зручними та гнучкими у використанні.

У підсумку, і класові компоненти та функціональні компоненти є важливими для роботи з React, проте в останніх версіях React рекомендовано використовувати функціональні компоненти на базі хуків. Функціональні компоненти мають більш просту структуру, меншу кількість коду та можуть бути більш оптимізованими для роботи з пам'яттю. Крім того, функціональні компоненти дозволяють використовувати хуки, що значно спрощує роботу зі станом та дозволяє виконувати ефектні дії на основі життєвого циклу компонента. Однак, класові компоненти все ще корисні для роботи з React і можуть бути використані у випадках, коли потрібні деякі функції, які не підтримуються функціональними компонентами на базі хуків.

Для своєї роботи я вирішив використовувати функціональні компоненти з урахуванням описаних вище відмінностей.

2.2.4 Побудова інтерактивної версії в React

На цьому етапі я мав перелік ієрархічно визначених компонентів і міг зібрати з них функціонуючий застосунок. Було створено рішення, яке використовує визначену модель даних і рендерить інтерфейс відповідно до неї.

Для реалізації інтерактивної версії додатку я створив компоненти, що мають різні рівні вкладеності інших компонентів. Для передачі даних між ними я використав пропси. Пропси – це реалізація передачі даних від батьківського елементу до дочірнього (тільки в одному напрямку). Також було реалізовано стан, що дало можливість зберігати дані на рівні конкретного компоненту та ререндерити інтерфейс відповідно до них за допомогою спеціального хуку. З урахуванням розмірів застосунку я прийняв рішення, що використовувати додаткові рументи для керування стану додатку немає потреби.

Оскільки мій застосунок не передбачає надмірного переповнення компонентами, я почав розробку з компонентів, що знаходяться вище за ієрархію.

2.2.5 Визначення відображення стану інтерфейсу

У React стан (state) використовується для зберігання даних, які можуть змінюватися під час взаємодії з користувачем або залежати від зміни даних з

сервера. В React на хуках для роботи зі станом використовуються спеціальні хуки useState і useEffect.

Хук useState дозволяє оголосити стан компонента та зберігати його значення. Після оголошення можна звертатися до стану за допомогою змінної, що повертається хуком. Якщо стан змінюється, React автоматично рендерить компонент з оновленим значенням стану. При цьому можна передати нове значення стану через функцію, яка повертається разом зі значенням стану.

Хук useEffect дозволяє виконувати ефекти (дії, що повинні відбуватися при зміні стану) після рендерингу компонента. Цей хук дозволяє використовувати змінні стану та пропсів в ефектах. Якщо ефект повинен виконуватися тільки під час першого рендерингу компонента, то треба передати пустий масив залежностей як другий аргумент хука.

Робота зі станом на хуках дозволяє писати чистий та читабельний код. Це дозволяє розбити логіку на менші компоненти, зберігаючи при цьому стан всередині компонента. Оскільки хуки забезпечують роботу зі станом на рівні компонента, вони дозволяють більш гнучко керувати змінами відображення та реагувати на взаємодію з користувачем.

Щоб правильно спроєктувати застосунок, необхідно врахувати мінімальну кількість станів, що знадобляться додатку. Я дотримувався принципу DRY (Don’t Repeat Yourself) . Визначивши мінімальну кількість стану, який потрібен застосунку, я розподілив його між компонентами відповідно до того, в який компонентах та підкомпонентах він використовується.

Даліварто визначити компоненти, в яких знаходитимуться стани. Враховуючи те, що в React односторонній потік даних, а ці дані необхідні для розрахунку в спільному для всі компонентів, що їх використовують, компоненті, стан було розміщено в їхньому першому спільному батьківському компоненті для всіх випадків.

2.2.6 Створення зворотнього потоку даних

Зворотній потік даних (англ. two-way data binding) в React означає, що зміна значення елемента форми (наприклад, введення тексту в текстове поле) призводить до оновлення відповідного значення в стані компонента, а також, коли змінюється стан компонента, відповідний елемент форми також оновлюється.

У React зворотній потік даних можна реалізувати різними способами. Один з них - використання контрольованих компонентів (controlled components).

Контрольованими називаються ті компоненти, які отримують свої значення та обробники подій з зовнішнього джерела, наприклад, зі стану батьківського компонента. Таким чином, зміна значення в контрольованому компоненті призводить до відповідної зміни стану батьківського компонента, а зміна стану батьківського компонента приводить до оновлення відповідного значення в контрольованому компоненті.

Інший спосіб реалізації зворотного потоку даних - за допомогою передачі функції з батьківського компонента в дочірній компонент. Цей підхід дозволяє дочірньому компоненту змінювати стан батьківського компонента, коли стан дочірнього компонента змінюється. Зазвичай, такі функції передаються як властивості компонентів (props).

Зворотній потік даних є важливим поняттям в React, оскільки він дозволяє ефективно та просто оновлювати стан компонентів на основі введення користувача та інших подій. При правильному використанні цього підходу можна створити більш динамічні та інтерактивні компоненти.

В своїй роботі я використав другий підхід тому, що він на мою особисту думку є зручнішим для наступної підтримки коду.

2.2.7 Створення функціоналу для спілкування з сервером

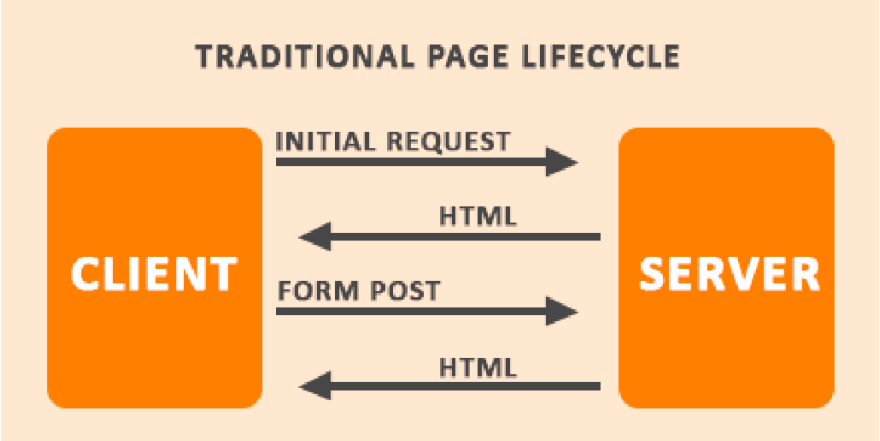
Мати інтерфейс достатньо для того, щоб відображати дані, але відображувати буде нічого, якщо даних немає. Для отримання даних необхідно написати функції, які створюватимуть запити на сервер. У відповідь на такі запити браузер отримуватиме відповідь з необхідними даними в текстовому форматі, які будуть відмальовуватись в інтерфейсі.

2.3 Вибір підходу в контексті кількості сторінок

Web-застосунки розділяються на односторінкові (SPA) та багатосторінкові (MPA).

MPA - це традиційна модель веб-додатку, в якій кожен запит на сервер повертає нову сторінку з сервера, і кожна сторінка зазвичай містить в собі повну структуру HTML, CSS та JavaScript коду. Це означає, що при кожному переході між сторінками веб-додатку, весь код повинен бути

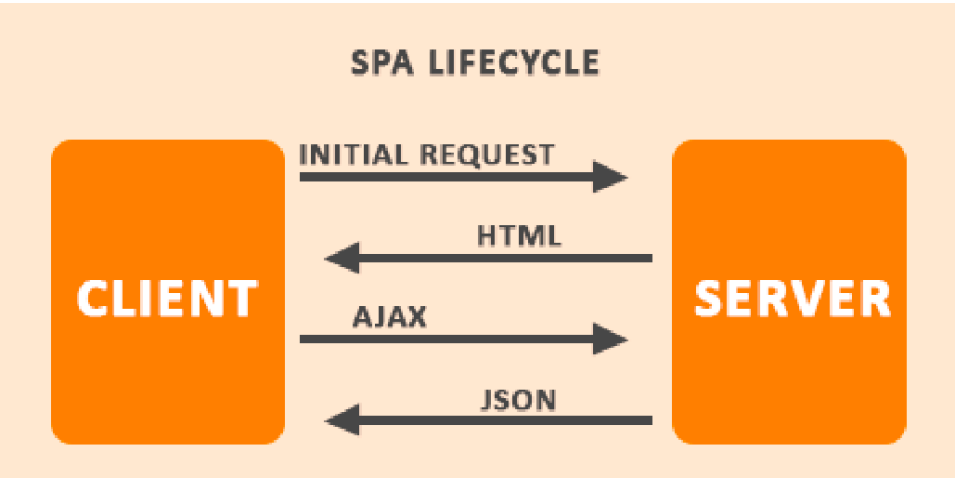
перезавантажений, що може призводити до більш тривалих часів завантаження сторінок.



Ілюстрація 2.1. Модель роботи не SPA

SPA - це модель веб-додатку, в якій всі сторінки генеруються в одному HTML-файлі, а зміна вмісту сторінки відбувається динамічно за допомогою JavaScript. При цьому, замість перезавантаження сторінки при кожному запиті, тільки необхідний вміст оновлюється, що дозволяє значно зменшити час завантаження сторінок і збільшити швидкість відгуку веб-додатку.

Для розробки SPA зазвичай використовують фреймворки, такі як React, Angular або Vue, які надають інструменти для ефективної організації та роботи зі структурою сторінок в одному файлі. У MPA зазвичай використовують традиційні технології веб-розробки, такі як HTML, CSS та JavaScript, без необхідності використовувати фреймворки.



Ілюстрація 2.2. Модель роботи SPA

SPA – це фактично одна сторінка, чиї компоненти взаємодіють з користувачем, проводячи автоматичний рендер сторінки. За такого підходу немає потреби завантажувати нові сторінки з сервера. Серед прикладів, що доводять ефективність даного підходу, є: Twitter, Facebook, Gmail, Trello.

В моїй роботі всі компоненти завантажуються під час першого запиту. Також зручно реалізована навігація. Дії користувачів фіксуються для зручної навігації. Це дає можливість продувжити роботу на тому моменті, на якому було скопійовано посилання на сторінку. Для реалізації маршрутів я використаю <BrowserRouter>.

Під час підвантаження нових компонентів у односторінкових сайтів контент буде оновлюватись тільки в тих місцях, де він був змінений. Такий механізм вибраної бібліотеки допоможе зменшити кількість даних, що передаються від сервера до браузера, та зменшити час очікування відповіді.

SPA (Single Page Application) завантажує весь необхідний код у форматі однієї сторінки. Це економить час на повторне завантаження елементів інших підсторінок, а тому я вибрав саме цей підхід під час написання коду для своєї роботи.

2.4 Зберігання даних

Збереження даних локально в браузері може бути корисним для збереження налаштувань, які були зроблені користувачем, та для збереження інформації, яку користувач може повторно використати на сайті.

Для заощадження часу користувача під час визначення локації частину даних я вирішив зберігати для повторного використання в самому браузері. Для реалізації такого функціоналу я розглянув та вирішив використовувати один з можливих способів збереження даних.

Для розгляду способі зберігання даних в браузерній частині застосунку я відібрав: LocalStorage[9] (Локальне сховище), SessionStorage[10] (Сесійне сховище) та Cookie.

2.4.1 LocalStorage

Local Storage [9] - це механізм збереження даних в браузері, що дозволяє зберігати дані на стороні клієнта, тобто на комп'ютері користувача. Local Storage зберігає дані у вигляді пар ключ-значення і може бути використаний для збереження різноманітної інформації, такої як налаштування користувача, обрані елементи або локальні дані додатка.

Local Storage підтримується всіма сучасними браузерами, включаючи Google Chrome, Mozilla Firefox, Safari та Internet Explorer. Для роботи з Local Storage використовується JavaScript API, що дозволяє зчитувати та записувати дані в локальному сховищі.

Основні переваги Local Storage:

- Легкість використання: Local Storage має простий інтерфейс, що дозволяє легко зберігати та отримувати дані.

- Зручність для користувача: Local Storage дозволяє зберігати налаштування та інші дані без необхідності входу в систему або реєстрації.

- Швидкість доступу: Дані, збережені в Local Storage, можуть бути доступні негайно, що забезпечує високу продуктивність.

- Безпека: Дані, збережені в Local Storage, доступні лише на стороні клієнта та не передаються на сервер, що робить їх більш безпечними для користувача.

Однак, слід пам'ятати, що Local Storage має обмеження на обсяг збережуваних даних та не може зберігати складні об'єкти, такі як функції та класи. Крім того, дані, збережені в Local Storage, можуть бути витерті користувачем або видалені браузером при очищенні кешу.

Зразок запису даних в LocalStorage на прикладі сніпету коду з репозиторію виконаного проєкту:

```
localStorage.setItem("latFromNoirWeather", userLocation.lat);

localStorage.setItem("lonFromNoirWeather", userLocation.lon);
```

Зразок отримання даних з LocalStorage на прикладі сніпету коду з репозиторію виконаного проєкту:

```
localStorage.getItem("latFromNoirWeather");

localStorage.getItem("lonFromNoirWeather");
```

2.4.2 SessionStorage

Session Storage [10] - це механізм збереження даних в браузері, що дозволяє зберігати дані на стороні клієнта протягом однієї сесії, тобто поки користувач залишається на сайті або до закриття вікна браузера. При закритті вікна браузера або перезавантаженні сторінки, дані збережені в Session Storage втрачаються.

Session Storage зберігає дані у вигляді пар ключ-значення, так само, як і Local Storage, і може бути використаний для збереження різноманітної інформації, такої як налаштування користувача, обрані елементи або локальні дані додатка.

Session Storage підтримується всіма сучасними браузерами, включаючи Google Chrome, Mozilla Firefox, Safari та Internet Explorer. Для роботи з Session Storage використовується JavaScript API, що дозволяє зчитувати та записувати дані в сховищі сесії.

Основні переваги Session Storage:

- Легкість використання: Session Storage має простий інтерфейс, що дозволяє легко зберігати та отримувати дані.

- Зручність для користувача: Session Storage дозволяє зберігати налаштування та інші дані без необхідності входу в систему або реєстрації.

- Швидкість доступу: Дані, збережені в Session Storage, можуть бути доступні негайно, що забезпечує високу продуктивність.

- Безпека: Дані, збережені в Session Storage, доступні лише на стороні клієнта та не передаються на сервер, що робить їх більш безпечними для користувача.

Як і у випадку з Local Storage, слід пам'ятати, що Session Storage має обмеження на обсяг збережуваних даних та не може зберігати складні об'єкти, такі як функції та класи.

Зразок запису даних в SessionStorage:

```
sessionStorage.setItem("key", "value");
```

Зразок отримання даних з SessionStorage:

```
sessionStorage.getItem("key");
```

2.4.3 Cookie

Cookie - це невеликий текстовий файл, який зберігається на комп'ютері користувача через веб-браузер. Він містить інформацію про відвідування сайту, на якому був створений cookie, і зазвичай використовується для зберігання налаштувань, інформації про користувача та іншої інформації, яка може бути корисною для подальшої роботи з веб-сайтом.

Основні характеристики cookie:

- Термін зберігання: Cookie може мати термін зберігання від кількох годин до кількох років, що дозволяє зберігати дані про користувача на тривалий час.

- Доступність: Cookie доступні з будь-якої сторінки сайту, що дозволяє зберігати інформацію про користувача на протязі всього його перебування на сайті.

- Розмір: Розмір cookie обмежений і зазвичай складає до 4 Кб, що забезпечує швидку обробку і передачу даних між браузером і сервером.

- Безпека: Cookie не містять особистої інформації про користувача, але можуть бути використані для стеження за його поведінкою на сайті. Це робить їх досить безпечними, якщо вони використовуються правильно.

Існують два типи cookie: сесійні та постійні.

- Сесійні cookie зберігаються на короткий термін, до закриття веб-браузера або до завершення сесії. Вони використовуються для збереження тимчасової інформації про користувача, такої як обраний товар у кошику або налаштування пошуку.

- Постійні cookie зберігаються на довгий термін і мають термін зберігання від кількох годин до кількох років.Вони використовуються для збереження інформації про користувача, такої як логін і пароль, щоб користувач міг автоматично увійти на сайт при наступному відвідуванні. Також вони можуть використовуватися для збереження налаштувань сайту, які були зроблені користувачем.

Враховуючи характеристики кожного способу збереження даних я вирішив зберігати дані в локальному сховищі. Такий спосіб, на мою думку, найбільш оптимальний та гарантує збереження цих даних на конкретному пристрої під час різних сесій.

2.5 Спілкування з сервером

HTTP (Hypertext Transfer Protocol) - це протокол, що використовується для передачі даних з сервера на клієнт (браузер) та навпаки. Браузер і сервер спілкуються між собою, обмінюючись HTTP запитами та відповідями.

HTTP запити можуть бути різних типів. Найбільш поширеними є GET та POST запити. GET запит використовується для запиту даних з сервера, наприклад, отримання вмісту сторінки або інформації з бази даних. POST запит використовується для відправлення даних на сервер, наприклад, при відправленні форми на сторінці.

Після того, як браузер відправляє HTTP запит на сервер, сервер обробляє запит та відправляє відповідь назад до браузера. Відповідь може містити HTML-код сторінки, JavaScript-файли, CSS-стилі та інші ресурси.

Коли браузер отримує відповідь від сервера, він може відобразити ці дані на сторінці. Відповідь може бути в різних форматах, таких як HTML, JSON, XML тощо. Браузер також може виконувати додаткові дії з отриманими даними, наприклад, зберігати їх у локальному сховищі, оновлювати сторінку або виконувати динамічні ефекти.

Одним з головних переваг HTTP запитів є те, що вони дозволяють динамічно оновлювати сторінки без необхідності перезавантаження сторінки повністю. Це дозволяє створювати більшість веб-додатків, таких як соціальні мережі, електронні магазини та інші, які працюють у режимі реального часу.

В SPA (Single Page Application) більшість логіки обробки даних відбувається на стороні клієнта (браузера), що дозволяє зменшити навантаження на сервер та зменшити час відповіді на запит. Однак, SPA мають певні обмеження, такі як відсутність підтримки SEO та проблеми зі складністю розробки.

Є декілька способів надсилання запитів на сервер у JavaScript. Найпоширенішими з них є:

XMLHttpRequest: Це стандартний метод надсилання запитів на сервер, який використовується в більшості браузерів. Він дозволяє надсилати запити на сервер і отримувати відповідь у вигляді XML або JSON. Щоб використовувати XMLHttpRequest, треба створити новий об'єкт XMLHttpRequest, налаштувати його та надіслати запит.

Fetch API: Це новітній метод, який був доданий до JavaScript для надсилання запитів на сервер. Fetch API дозволяє надсилати запити та отримувати відповідь у вигляді JSON або тексту. Він працює з промісами, що дозволяє зручно обробляти результат запиту.

Axios: Це стороння бібліотека, яка дозволяє зручно надсилати запити на сервер з JavaScript. Axios підтримує обіцянки (promises) та дозволяє легко обробляти помилки, які можуть виникати під час взаємодії з сервером.

jQuery AJAX: Це метод надсилання запитів на сервер, який використовує jQuery бібліотеку. Він дозволяє надсилати запити на сервер та отримувати відповідь у вигляді JSON або HTML. jQuery AJAX також дозволяє зручно обробляти помилки та взаємодіяти з сервером без перезавантаження сторінки.

Ці способи дозволяють взаємодіяти з сервером та отримувати від нього дані, які можуть бути використані для динамічного оновлення сторінки без перезавантаження. Кожен з них має свої переваги та недоліки, тому вибір залежить від потреб проекту та ваших уподобань.

В роботі я вирішив використовувати метод fetch() без сторонніх бібліотек, бо такий синтаксис особисто мені більше подобається.

2.6 Автентифікація користувача

Supabase підтримує OAuth 2.0 для автентифікації користувачів за допомогою різних провайдерів. OAuth 2.0 - це стандарт протоколу автентифікації, який дозволяє користувачам надавати доступ до своїх облікових записів в інших сервісах, не надаючи їм своїх логінів та паролів. Замість цього користувачі надають дозвіл на доступ до своїх даних через спеціальний токен.

Supabase підтримує наступні провайдери OAuth 2.0:

- Google - дозволяє користувачам авторизуватися за допомогою своїх облікових записів Google.

- Facebook - дозволяє користувачам авторизуватися за допомогою своїх облікових записів Facebook.

- GitHub - дозволяє користувачам авторизуватися за допомогою своїх облікових записів GitHub.

- GitLab - дозволяє користувачам авторизуватися за допомогою своїх облікових записів GitLab.

- Bitbucket - дозволяє користувачам авторизуватися за допомогою своїх облікових записів Bitbucket.

- Apple - дозволяє користувачам авторизуватися за допомогою своїх облікових записів Apple.

При використанні провайдерів OAuth 2.0, користувачі можуть авторизуватися на сайті, використовуючи свої облікові записи в інших сервісах, без необхідності створювати новий обліковий запис. Після успішної автентифікації користувача Supabase поверне токен доступу, який можна використовувати для доступу до інших функцій Supabase API, таких як доступ до бази даних та інші.

У загальному, використання провайдерів OAuth 2.0 дозволяє спростити авторизацію користувачів на вашому сайті та зберігання облікових записів. В маємо випадку необхідна авторизація через сервіс Google. Це цілком логічно тому, що я використовую інший сервіс цієї компанії.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ЗАСТОСУНКУ

3.1 Створення дизайну застосунку

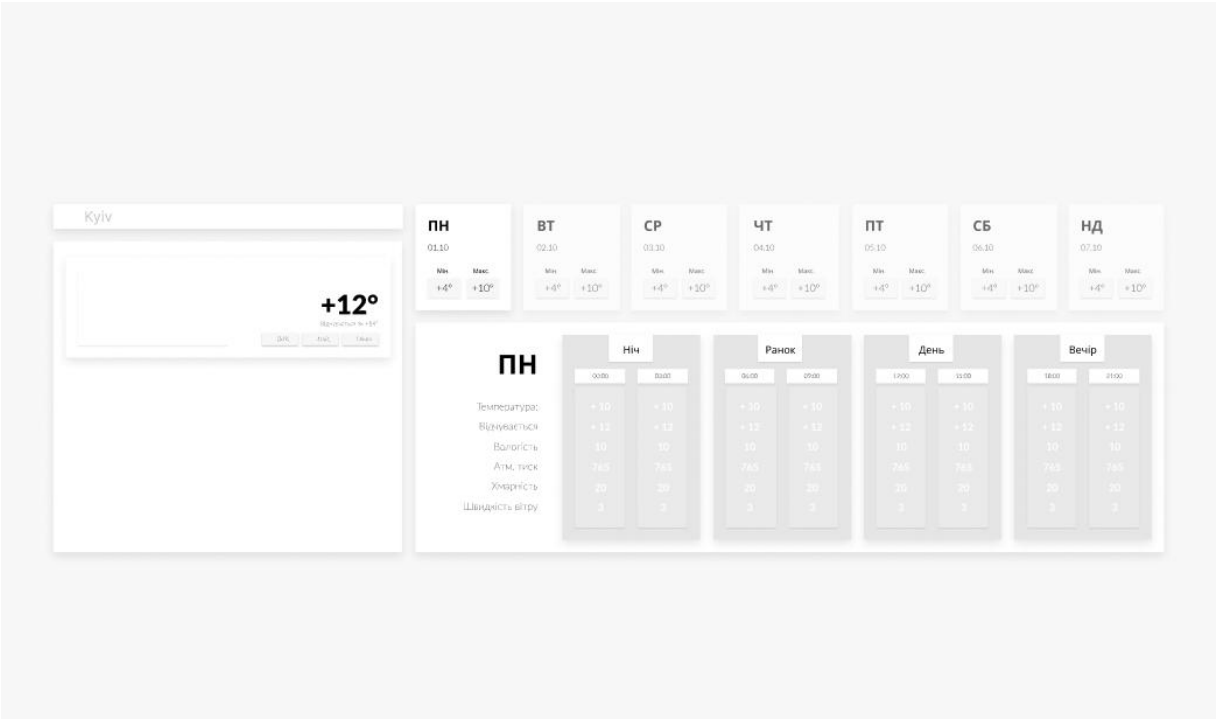
Процес створення дизайну сайту в Figma може бути розділений на запропоновані далі етапи:

Перший етап - це створення wireframe, який відображає структуру та зміст сайту без будь-яких дизайнерських елементів. В Figma це можна зробити за допомогою вбудованих інструментів або завантаживши вже готові елементи з бібліотек.

Я вирішив не користуватись елементами з бібліотек, а створити свій власний wireframe.

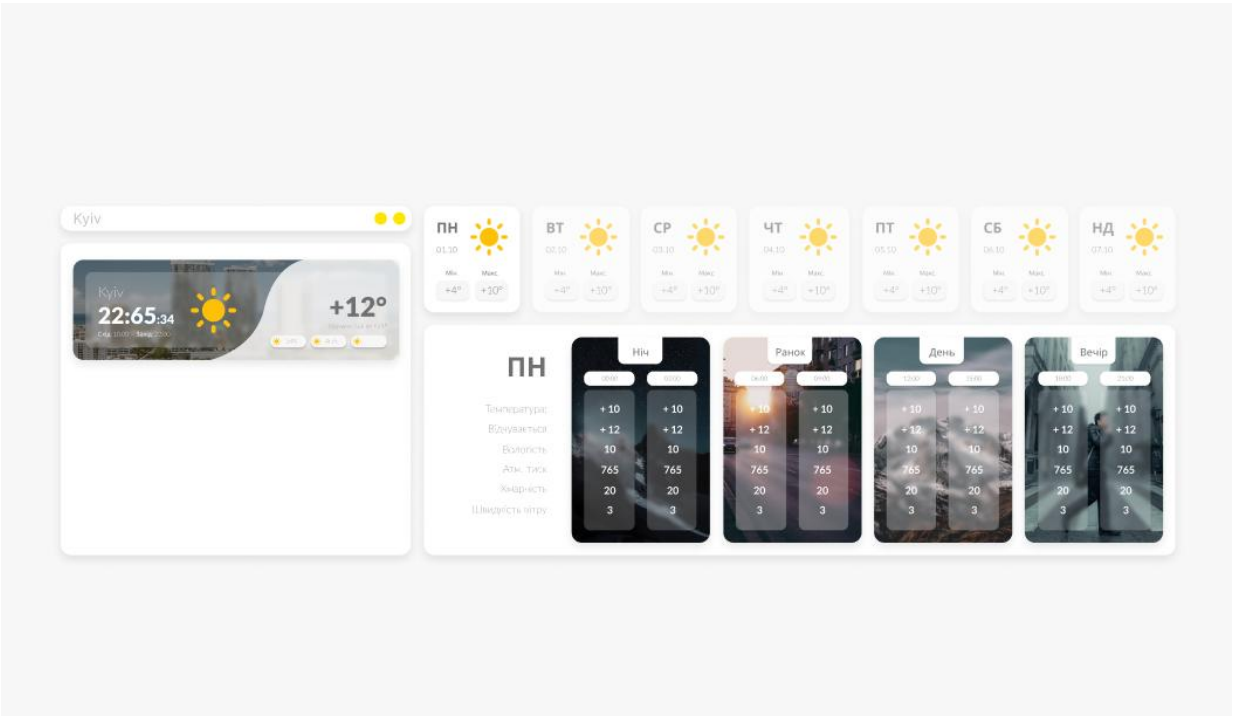
Ілюстрація 3.1. Створений wireframe

Наступним етапом є створення макету, де розміщуються всі елементи та компоненти сайту. В Figma ви можете використовувати шари, векторні форми, текст та інші інструменти для створення макету.



Ілюстрація 3.2. Створений макет

Після створення макету можна додати дизайнерські елементи, такі як кольори, фони, шрифти та інші деталі. В Figma можна використовувати бібліотеки компонентів, щоб швидко додавати стандартні елементи дизайну.



Ілюстрація 3.3. Створений макет з початковим дизайном

3.2 Створення структури проєкту

Перед початком наповнення сайту контентом потрібно створити цей проєкт. Для місця збереження роботи я вибрав сервіс GitHub, який я буду використовувати для контролю версій застосунку.

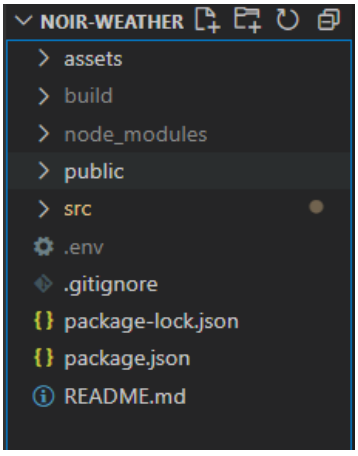
Після створення проєкту на вказаному сервісі, я клоную репозиторій на локальний комп'ютер та починаю створення проєкту в застосунку VSCode. Спершу варто вибрати середовище розробки. Мій вибір зупинився на наборі інструментів create-react-app, який є чудовим способом реалізувати проєкт таких масштабів, як у мене. Для встановлення використовую наступну команду:

```
npx create-react-app my-app
```

Використана утиліта пропонує свою власну структуру репозиторію, яку я планую використати. Спершу я очищую репозиторій від непотрібних запропонованих файлів та організовую ієрархію папок відповідно до вибраного підходу.

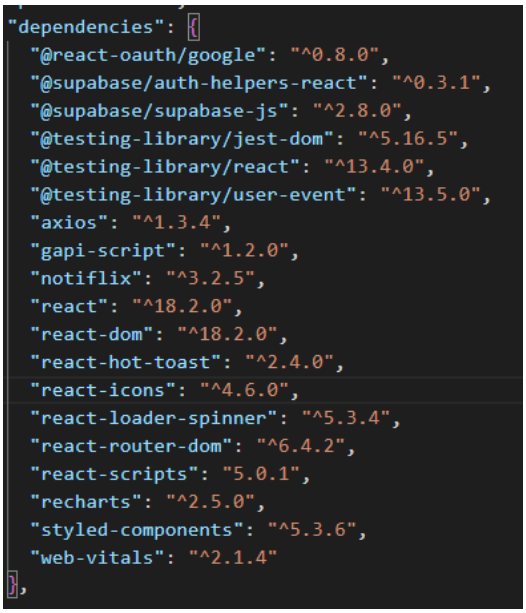
В кореневій папці проєкту є перелік папок з контентом сайту та правилами, що описують те, як цей контент необхідно відображати на сторінці браузера користувача.

До важливих для процесу розробок безпосередньо компонентів сторінки входять наступні папки: public, src, env, package.json.



Ілюстрація 3.4. Початкова структура репозиторію

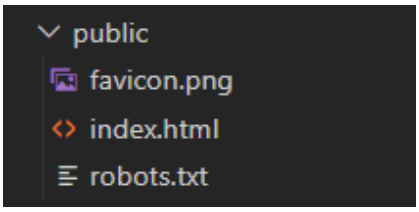
В файлі ./package.json я зберігаю дані про налаштування проєкту та його залежності. До використаних на даному проєкті залежностей входять наступні бібліотеки та інші програмні засоби:



Ілюстрація 3.5. Використані в проєкті залежності

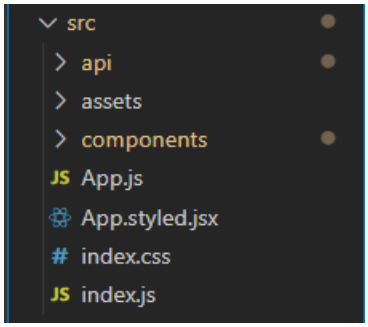
В файлі ./env я зберігаю змінні оточення, які використано в застосунку для налаштування середовища, в якому вони працюють.

В папці ./public знаходяться файл index.html, в який будуть рендеритись всі компоненти для відображення на сторінці, фавікон сайту та набір правил для пошукових роботів robots.txt.



Ілюстрація 3.6. Початкова структура папки ./public

В папці ./src знаходиться весь контент сайту, а саме всі компоненти та логіка їх роботи.



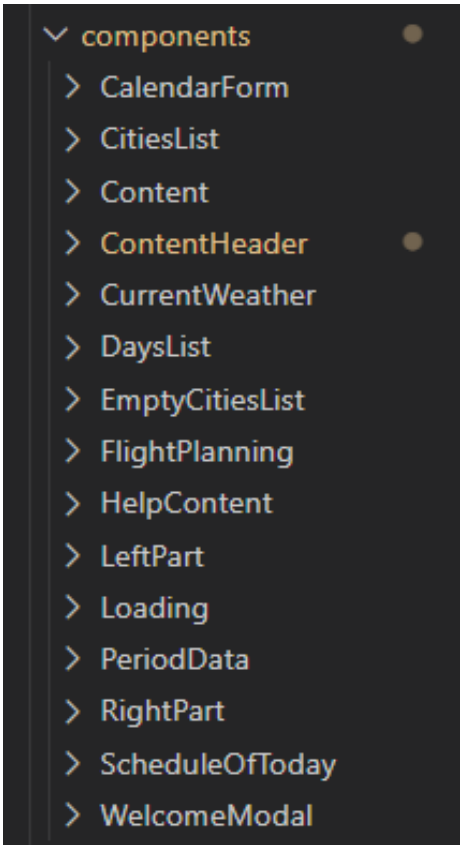
Ілюстрація 3.7. Початкова структура папки ./public

Тепер, коли в проєкту є базова структура та налаштування, можна почати створювати наповнення для сторінок. Для цього було створено index.js, контент якого буде рендеритись у вищезгаданий index.html та відображатись на сторінці.

Сніпет коду з index.js який рендерить код компоненту App у відповідний кореневий елемент сторінки:

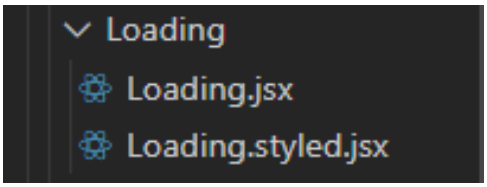
```
const root =
ReactDOM.createRoot(document.getElementById("root"));
root.render(
  <React.StrictMode>
        <App />
  </React.StrictMode>
);
```

Наступним етапом я розбиваю всю область проєкту на окремі компоненти. Важливість такого підходу було детально описано в попередньому розділі. Для зберігання компонентів використовую папаку ./src/components/. В даній папці відображено структуру каталогу з наявними компонентами:



Ілюстрація 3.8. Структура папки ./components

Відповідно кожен компонент в структурі папок міститиме два файли: файл з доком компоненту та файл з кодом стилізації компоненту.



Ілюстрація 3.9. Зразок структури папки випадкового компоненту

3.3 Написання коду розмітки компонентів застосунку

На даному етапі реалізовано основну структуру проєкту. Надалі я створюю наповнення для заданої структури. Основними будівельними блоками сторінки є компоненти. Кожен компонент описує кодом певну частину сайту, а саме набір елементів компоненту, розташування елементів, особливості відображення елементів.

Зразок коду компоненту, який рендерить елемент “loader”:

```
import { LoaderStyled, Loader } from "./Loading.styled";
import loader from "../../assets/images-webp/logo-no-bg.webp";

const Loading = () => {
  return (
    <LoaderStyled>
      <Loader src={loader} alt="loader" />
    </LoaderStyled>
  );
};

export default Loading;
```

Запропонований компонент містить елемент-обгортку та зображення, яке знаходить на першому рівні вкладеності свого батьківського елементу. Для реалізації розмітки я використав синтаксис JSX.

Аналогічний код використано для опису всіх компонентів. Оскільки проєкт містить 4016 рядків коду, в даній роботі я можу відобразити лиш їх частину.

3.4 Написання коду для стилізації компонентів

Створений елемент існує та відображається, але тепер нам потрібно його стилізувати. Для стилізації я використав бібліотеку styled-components. Спершу потрібно імпортувати раніше встановлену залежність в файл стилів. Для цього використаю наступний код:

```
import styled from "styled-components";
```

Тепер став доступний синтаксис для опису стилів, який є, на мою особисту думку, зручнішим за стандартний підхід. Зразок коду стилізації випадкового елементу вибраного компоненту:

```
export const LoaderStyled = styled.div`
  position: absolute;
  width: 100%;
  height: 100%;
  display: flex;
```

```
  flex-direction: column;
  justify-content: center;
  align-items: center;
  z-index: 1000;
`;
```

Вкінці потрібно імпортувати описані стилі в файл компоненту. Для цього використаємо наступний код:

```
import { LoaderStyled, Loader } from "./Loading.styled";
```

3.5 Написання коду логіки роботи компонентів

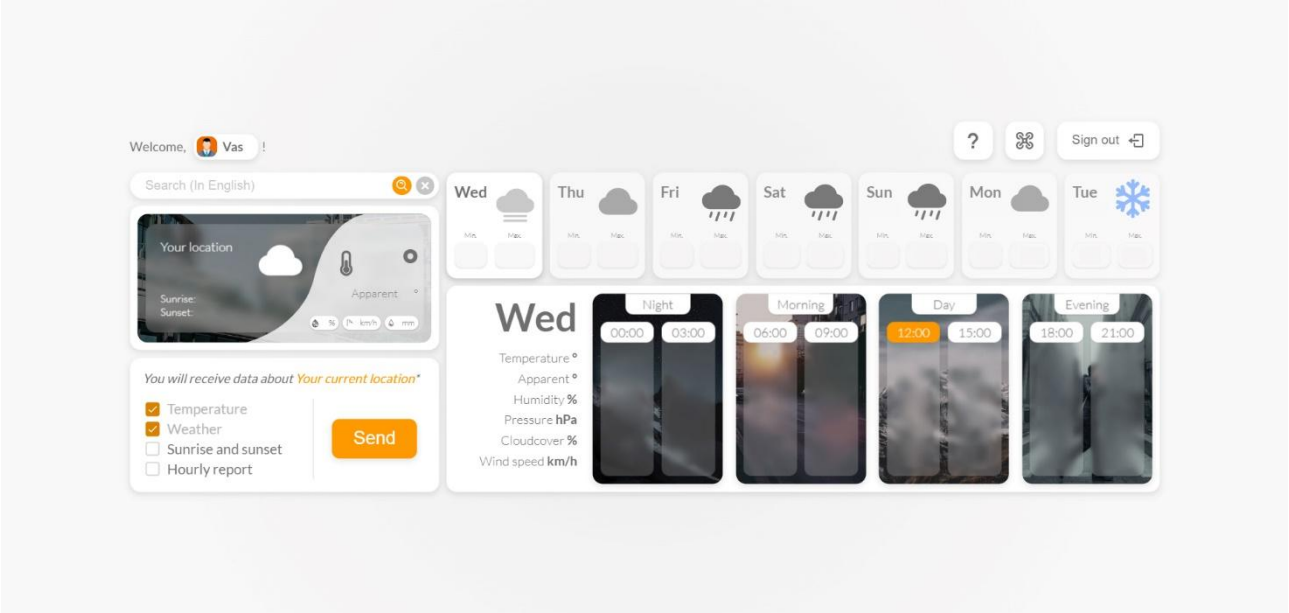
Більшість компонентів вимагають використання логіки роботи у вигляді функцій. Для прикладу наведу реалізовану функцію, що викликається тільки при першому відвідуванні сторінки:

```
useEffect(() => {
    setisLoadingPage(true);
    const handleData = async () => {
      try {
        const fetchedData = await getData();
        setChosenCity("Your location");
        if (fetchedData) {
          setWeatherData(fetchedData);
        }
      } catch (e) {
        alert("На даний момент сервер не працює");
      } finally {
        setisLoadingPage(false);
      }
    };

    handleData();
  }, [isFirstTime]);
```

З коду, що подібний до продемонстрованого, складається вся логіка роботи застосунку. Використаний синтаксис є репрезентативним для всіх компонентів даного проєкту.

В результаті реалізації компонентів я отримав наступний інтерфейс:

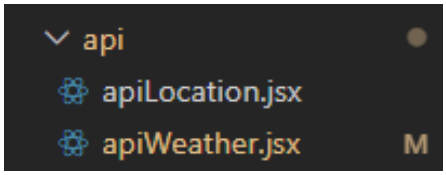


Ілюстрація 3.10. Вигляд інтерфейсу, що складається зі створених компонентів

3.6 Створення запитів на погодний API

Основним функціоналом застосунку є взаємодія з погодним API. Ця взаємодія реалізується шляхом відправлення запитів на сервер при чітко визначених діях користувача.

Всі функції, що реалізують взаємодію з API, винесено в окрему папку ./src/api/.



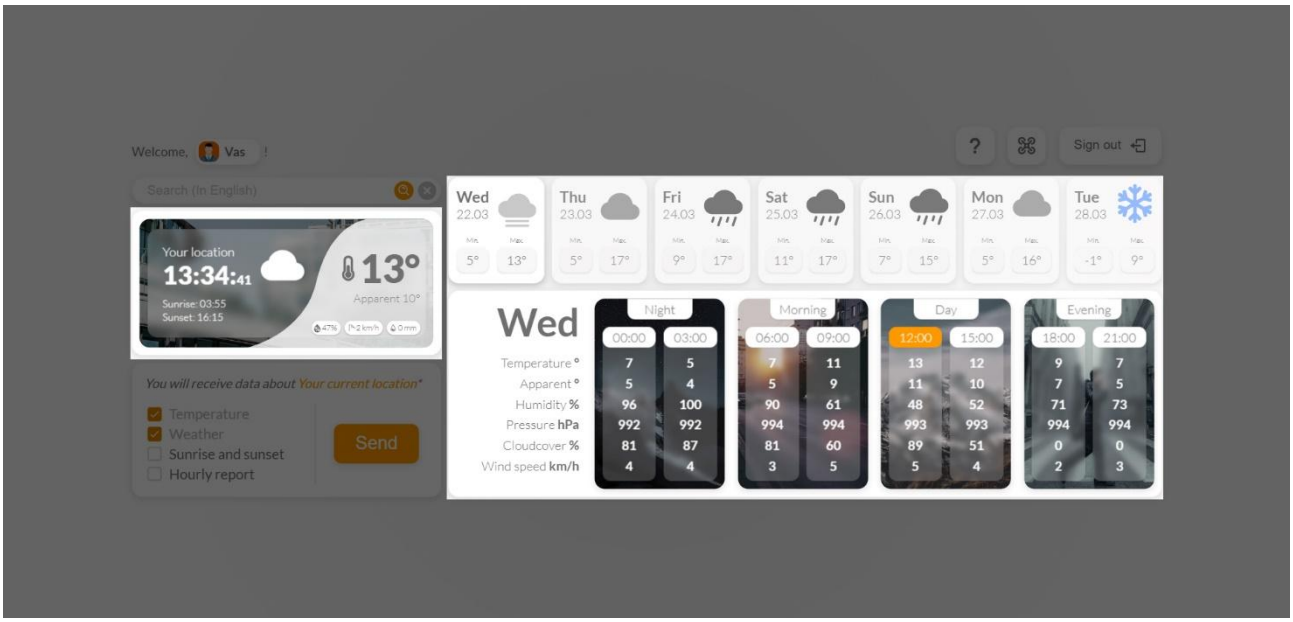
Ілюстрація 3.11. Структура папки ./src/api

Розглянемо кілька функції, що надсилають запити. Функція, що повертає погодні дані при першому відкритті сторінки:

```
export const getData = async () => {
  try {
    const data = await fetch(
      `https://api.open-
meteo.com/v1/forecast?latitude=${parseFloat(
        getLocation().lat
      )}&longitude=${parseFloat(
        getLocation().lon

)}&hourly=temperature_2m,relativehumidity_2m,apparent_temperat
ure,precipitation,rain,showers,snowfall,weathercode,surface_pr
essure,cloudcover,windspeed_10m,winddirection_10m&daily=weathe
rcode,temperature_2m_max,temperature_2m_min,sunrise,sunset&cur
rent_weather=true&timezone=GMT`
    );
    const parsedData = await data.json();
    return parsedData;
  } catch (e) {
    console.log(e);
  }
};
```

Написана функція є асинхронною, що є обов'язковою умовою при створенні подібних запитів. В посилання запиту підставляються координати користувача, а в результат ми отримуємо дані у JSON форматі. З отриманих даних я утворюю об'єкт, значення властивостей якого підставляю у вищеописаний код компонентів та отримую наступний результат:



Ілюстрація 3.12. Відображення отриманих даних на реалізованій сторінці

Для отримання координат користувача виконується наступна функція:

```
const getLocation = () => {
  if (
    localStorage.getItem("latFromNoirWeather") !== null &&
    localStorage.getItem("lonFromNoirWeather") !== null
  ) {
    try {
      userLocation.lat =
localStorage.getItem("latFromNoirWeather");
      userLocation.lon =
localStorage.getItem("lonFromNoirWeather");
    } catch (e) {
      console.log(e);
    }
  }

  sessionStorage.setItem("key", "value");

  sessionStorage.getItem("key");

  const setLatLon = (position) => {
    userLocation.lat = position.coords.latitude;
    userLocation.lon = position.coords.longitude;
    localStorage.setItem("latFromNoirWeather",
userLocation.lat);
    localStorage.setItem("lonFromNoirWeather",
userLocation.lon);
  };

  navigator.geolocation.getCurrentPosition(setLatLon);
  return userLocation;
};
```

Ми отримуємо доступ до об’єкту локації та записуємо значення повернутих функцією координат у створені змінні. Дані зберігаються в локальному сховищі для уникнення необхідності уточнювати їх при кожному наступному відкритті сторінки.

Функція, що повертає погодні дані при запиті за вказаним користувачем містом:

```
export const getDataFromSearch = async (lat, lon) => {
  try {
    const data = await fetch(
      `https://api.open-
meteo.com/v1/forecast?latitude=${parseFloat(
        lat
      )}&longitude=${parseFloat(
        lon

)}&hourly=temperature_2m,relativehumidity_2m,apparent_temperat
ure,precipitation,rain,showers,snowfall,weathercode,surface_pr
essure,cloudcover,windspeed_10m,winddirection_10m&daily=weathe
rcode,temperature_2m_max,temperature_2m_min,sunrise,sunset&cur
rent_weather=true&timezone=GMT`
    );
    const parsedData = await data.json();
    return parsedData;
  } catch (e) {
    console.log(e);
  }
};
```

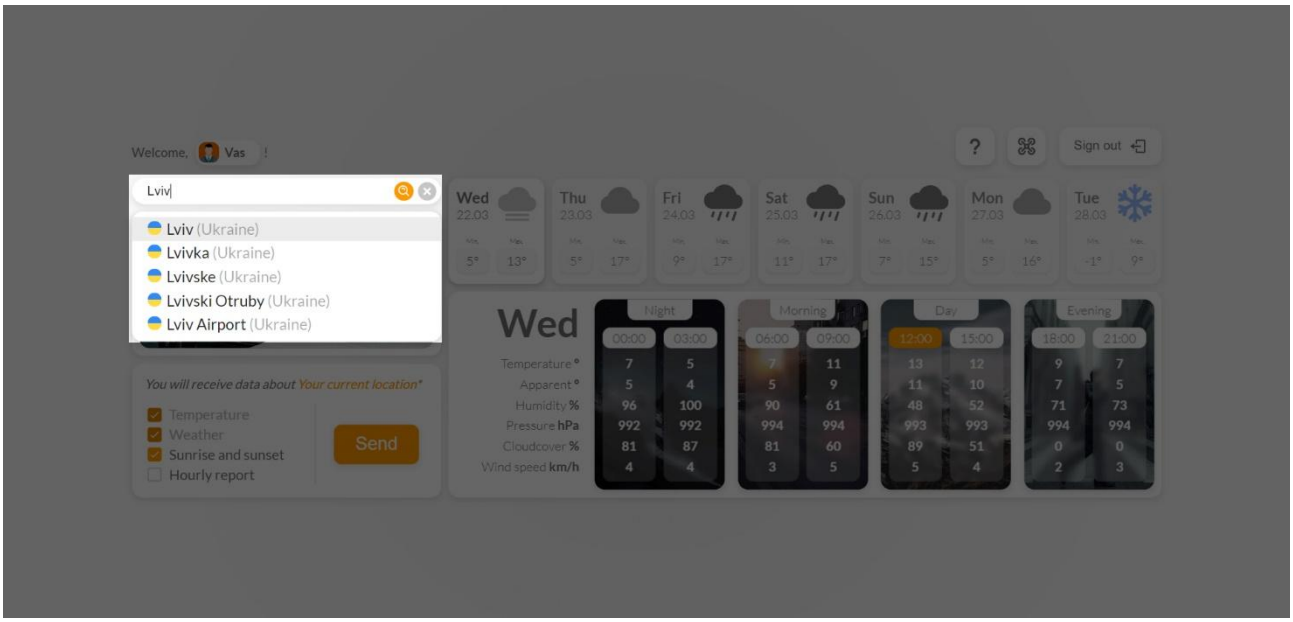
Створена функція приймає два аргументи: значення довготи та значення широти, та підставляє їх в посилання запиту.

Для визначення координатів локації за вибраним користувачем містом використовується наступна функція:

```
export const getLocation = async (cityName) => {
  try {
    const data = await fetch(
      `https://geocoding-api.open-
meteo.com/v1/search?name=${cityName}`
    );
    const parsedData = await data.json();
    return parsedData;
  } catch (e) {
    console.log(e);
  }
```

```
};
```

Створена функція приймає місто аргументом та повертає координати, за якими воно розташоване.

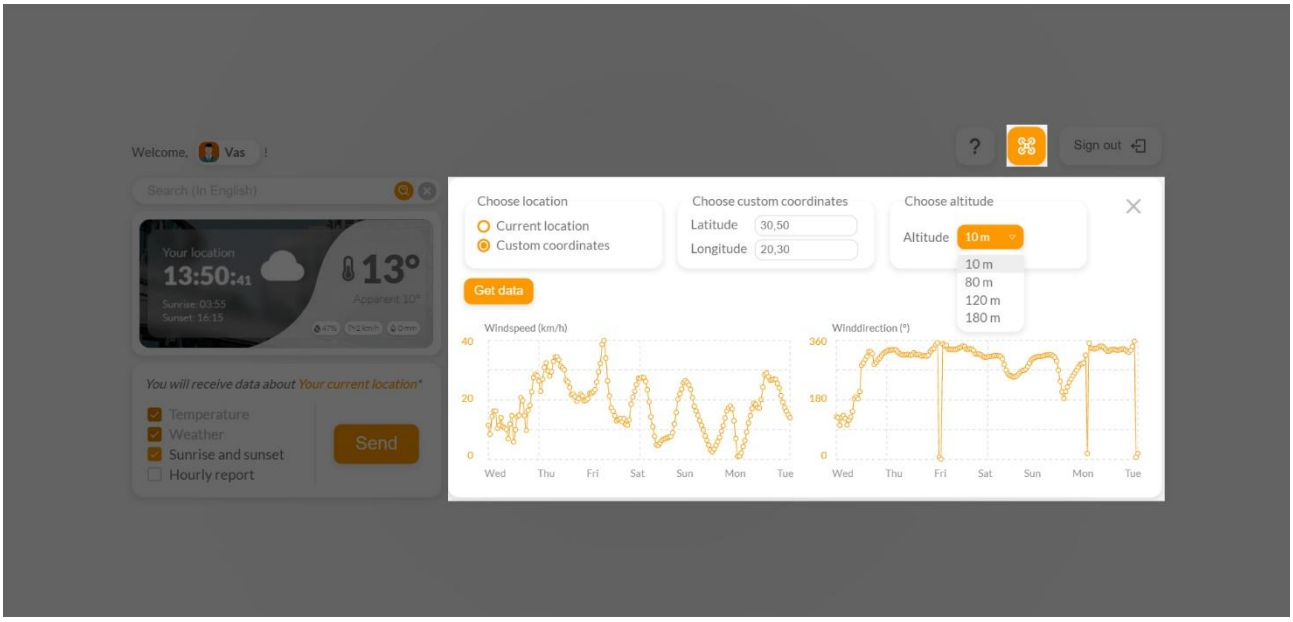


Ілюстрація 3.13. Частина інтерфейсу для вибору користувачем міста

Для функціоналу, який реалізує можливість отримати детальні дані про поведінку вітру, створено наступну функцію:

```
export const getWindDataFromSearch = async (lat, lon, alt) =>
{
  try {
    const data = await fetch(
      `https://api.open-
meteo.com/v1/forecast?latitude=${lat}&longitude=${lon}&hourly=
windspeed_${alt}m,winddirection_${alt}m`
    );
    const parsedData = await data.json();
    return parsedData;
  } catch (e) {
    console.log(e);
  }
};
```

Запропонована функція приймає три аргументи: довготу, широту, висоту та повертає детальні дані.



Ілюстрація 3.14. Відображення отриманих даних на реалізованій сторінці

3.7 Налаштування авторизації

Для отримання можливості реалізувати основний функціонал надсилання даних на Google Calendar користувача необхідно спершу додати можливість авторизації. Це необхідно для того, щоб Google Calendar API знав з яким користувачем взаємодіяти. Для реалізації даного функціоналу я вибрав використання supabase.

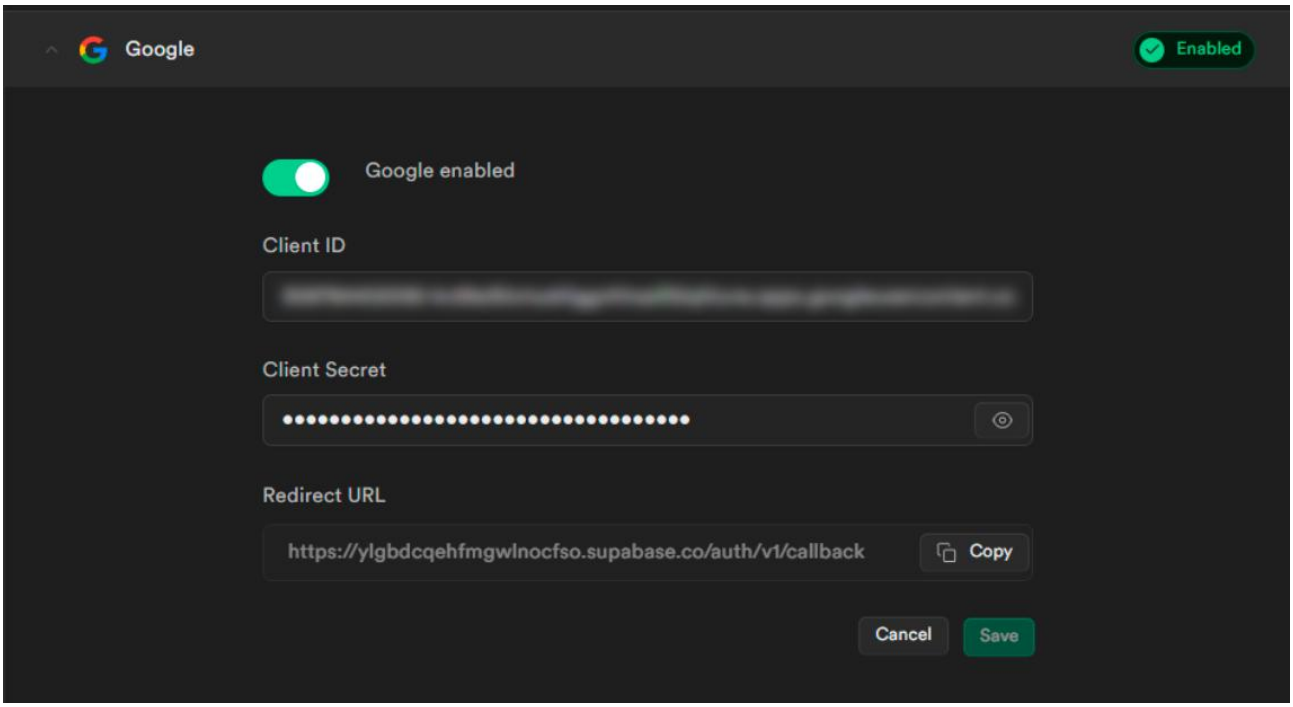
Supabase - це відкрите програмне забезпечення, яке дозволяє швидко створювати повноцінні back-end системи для веб-додатків. Вона пропонує безкоштовний план, який може забезпечити авторизацію через різні соціальні мережі, такі як Google, Facebook та GitHub.

Я виконав наступні кроки під час налаштування авторизації через Google з використанням Supabase:

- Створив Supabase проєкт.

Ілюстрація 3.15. Демонстрація створеного проєкту

- Додав провайдер та вибрав Google в запропонованому меню.



Ілюстрація 3.16. Демонстрація доданого провайдеру

- Для реалізації функціоналу надсилання даних на Google Calendar я використовую Google Console. Спершу створив проєкт та додав до нього сервіс для взаємодії з календарем – Google Calendar API. Відкрив вікно налаштувань авторизації Google. Ввів назву проєкту, його доменне ім'я та додав URL перенаправлення від Google.

- Отримав клієнтський ідентифікатор та секретний ключ Google. Скопіював їх для наступних кроків.

- Спершу в коді обгорнув компонент App у відповідний провайдер:

```
    <SessionContextProvider supabaseClient={supabase}>
        <App />
    </SessionContextProvider>
```

- Зловив властивість, яку передано провайдером. Вона необхідна для ідентифікації конкретного користувача.

```
const supabase = useSupabaseClient();
```

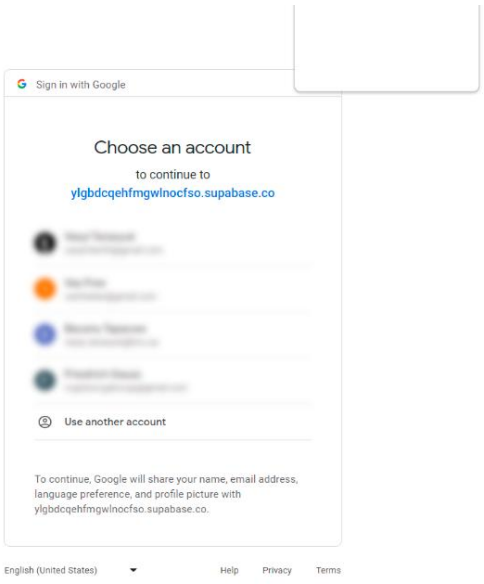
- Для здійснення авторизації через Google створив функції, яка буде викликати Supabase API для авторизації користувачів та при припинені з'єднання:

```
const googleSignIn = async () => {
  const { error } = await supabase.auth.signInWithOAuth({
    provider: "google",
    options: {
      scopes: "https://www.googleapis.com/auth/calendar",
    },
  });

  if (error) {
    toast.error("Login failured");
  }
};

const googleSignOut = async () => {
  await supabase.auth.signOut();
};
```

Переконуємось в працездатності створеного функціоналу шляхом виклику відповідної функції:



Ілюстрація 3.17. Демонстрація авторизації на сайті

3.8 Робота з Google Calendar API

Для створення нового календаря потрібно спершу переконатись в тому, що аналогічний календар відсутній. А, якщо він є, його спершу необхідно вилучити. Це варто зробити тому, що в іншому випадку будуть створюватись кілька календарів при повторних запитах.

Для того, щоб видалити календар, спершу потрібно отримати список наявних календарів. Для отримання відповідного списку я реалізував наступну функцію:

```
  const getCalendarsList = async () => {
    await fetch(

`https://www.googleapis.com/calendar/v3/users/me/calendarList`
,
      {
        method: "GET",
        headers: {
          Authorization: `OAuth ${session.provider_token}`,
        },
      }
    )
      .then((data) => {
        return data.json();
      })
      .then((data) => {
        const resultListIds = data.items.map((item) => {
          let currentId = null;

          if (item.summary === "NoirWeather") {
            currentId = item.id;
          }
          return currentId;
        });

        for (let id of resultListIds) {
          if (id !== null) {
            deleteCalendar(id);
          }
        }
        createCalendar();
```

```
      });
  };
```

Вкінці цієї функції є умова. В залежності від цієї умови буде викликано функцію для видалення поточного календаря. У випадку наявності поточного календаря буде викликано наступну функцію:

```
  const deleteCalendar = async (id) => {
    await
fetch(`https://www.googleapis.com/calendar/v3/calendars/${id}`
, {
      method: "DELETE",
      headers: {
        Authorization: `OAuth ${session.provider_token}`,
      },
    });
  };
```

Після цього буде викликано функцію для створення нового календаря:

```
  const createCalendar = async () => {
    await
fetch(`https://www.googleapis.com/calendar/v3/calendars`, {
      method: "POST",
      headers: {
        Authorization: `OAuth ${session.provider_token}`,
      },
      body: JSON.stringify({
        summary: "NoirWeather",
      }),
    })
      .then((data) => {
        if (data.status === 403) {
          toast.error("Calendar usage limits exceeded. Try
again later");
          setCreatingCalandar(false);
        } else {
          toast.success("Calendar created");
          return data.json();
        }
      })
      .then((data) => {
        sendEvents(data.id);
```

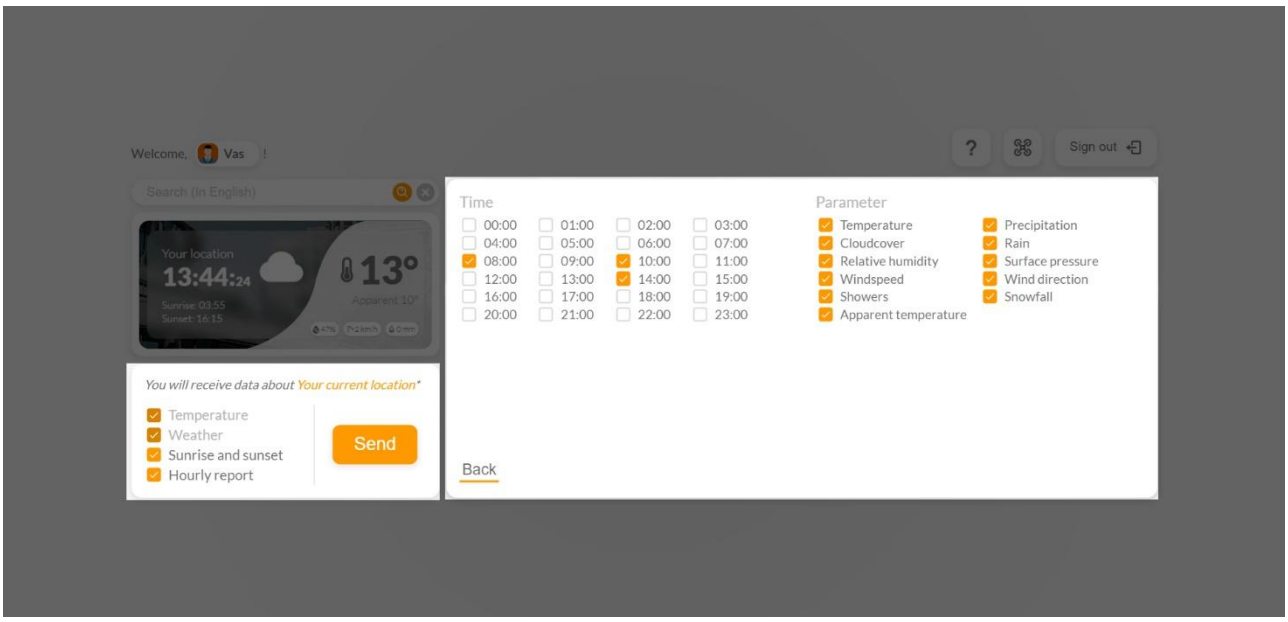
```
      });
  };
```

Тепер, коли новий календар створено, необхідно додати в нього події з даними про погоду. Для створення подій я написав наступну функцію:

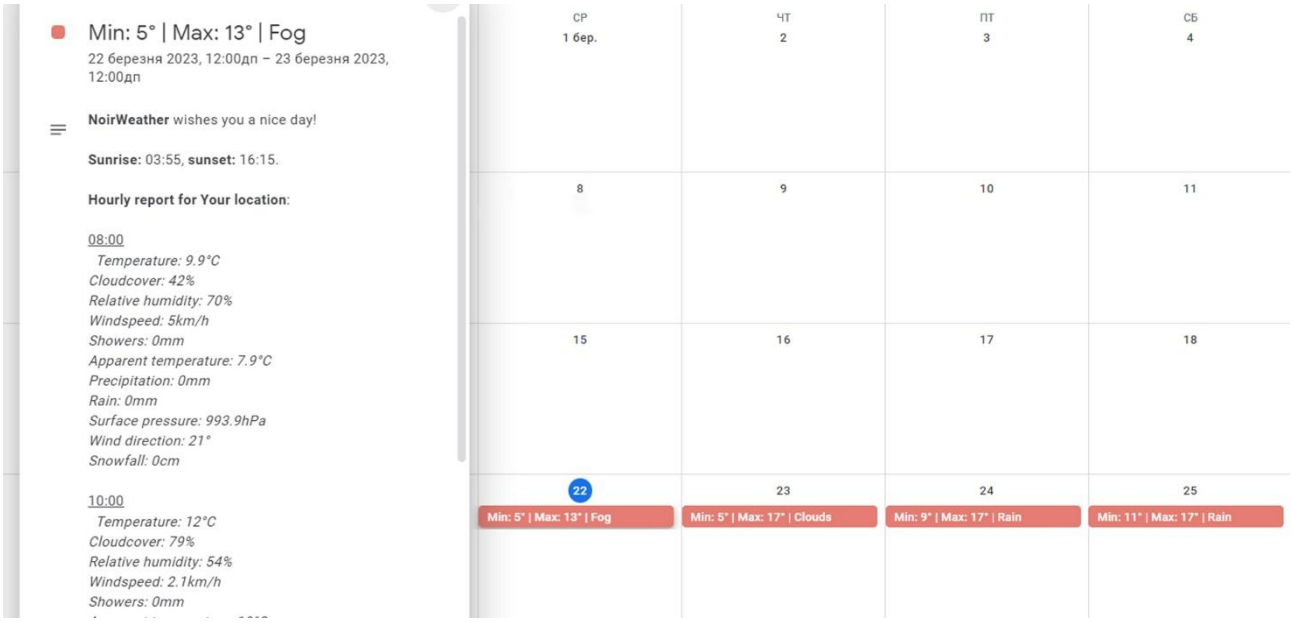
```
const sendEvents = async (calendarId) => {
    await fetch(

`https://www.googleapis.com/calendar/v3/calendars/${calendarId
}/events`,
      {
        method: "POST",
        headers: {
          Authorization: `OAuth ${session.provider_token}`,
        },
        body: JSON.stringify(event),
      }
    ).then((data) => {
      setCreatingCalandar(false);
      return data.json();
    });
  }
};
```

Цього функціоналу цілком достатньо для реалізації надсилання даних на Google Calendar. В дієздатності результату можна переконатись після перевірки шляхом взаємодії з інтерфейсом:



Ілюстрація 3.18. Демонстрація меню для надсилання даних на календар



Ілюстрація 3.19. Демонстрація Google Calendar з отриманими даними

ВИСНОВКИ

Протягом виконання роботи було написано 4016 рядків коду та в результаті реалізовано застосунок, що надає зручний та динамічний інтерфейс для взаємодії з погодними даними та реалізує можливість взаємодіяти з вказаними даними з використанням сервісу Google Calendar.

Можливість отримувати доступ до сервісу Google Calendar з використанням акаунту користувача отримано шляхом налаштування авторизації з використанням сервісу Supabase. З урахуванням поточних об’ємів виконаного проєкту даний підхід цілком раціональний.

Для реалізації інтерфейсу було використано найдоцільніший на думку розробника інструментарій та набір підходів. Запропоновані мови та бібліотеки розробки є наразі найактуальнішими на ринку, що забезпечує велику тривалість їхньої підтримки та активну спільноту.

Також протягом реалізації проєкту були покращені вміння використовувати вищезгадані інструменти розробником. Це дасть можливість розвивати проєкт і надалі та робити його зручнішим та коректнішим з кожним наступним оновленням. Зручну майбутню роботу з оновлення застосунку також забезпечують зручність та популярність вибраних інструментів, чиї версії постійно оновлюються та дають можливість покращувати взаємодію розробника з кодом.

В зв’язку з наявною потребою в постійній оптимізації повсякденних планувань потенціал запропонованого застосунку є актуальним та буде таким завжди. Даний застосунок може використовуватись і використовується для ознайомлення з прогнозом погоди та швидкого планування.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Документація NODE [Електронний ресурс], Доступ до ресурсу: https://nodejs.org/en/docs/

2. Документація пакетного менеджеру NPM [Електронний ресурс], Доступ до ресурсу: https://docs.npmjs.com/

3. Документація бібліотеки REACT [Електронний ресурс], Доступ до ресурсу: https://ru.reactjs.org/docs/getting-started.html

4. Документація бібліотеки STYLED COMPONENTS [Електронний ресурс], Доступ до ресурсу: https://styled-components.com/

5. Документація бібліотеки Recharts [Електронний ресурс], Доступ до ресурсу: https://recharts.org/en-US/api

6. Документація погодного API [Електронний ресурс], Доступ до ресурсу: https://open-meteo.com/en/docs

7. Документація Google Calendar API [Електронний ресурс], Доступ до ресурсу: https://developers.google.com/calendar/api/v3/reference

8. Документація Supabase [Електронний ресурс], Доступ до ресурсу: https://supabase.com/docs

9. Документація використання локального сховища [Електронний ресурс], Доступ до ресурсу: https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage

10. Документація використання сесійного сховища [Електронний ресурс], Доступ до ресурсу: https://developer.mozilla.org/en-US/docs/Web/API/Window/sessionStorage

ДОДАТОК А

Код створеного застосунку: https://github.com/ElVent0/noir-weather

ДОДАТОК Б

Опублікований застосунок: https://noir-weather.netlify.app/