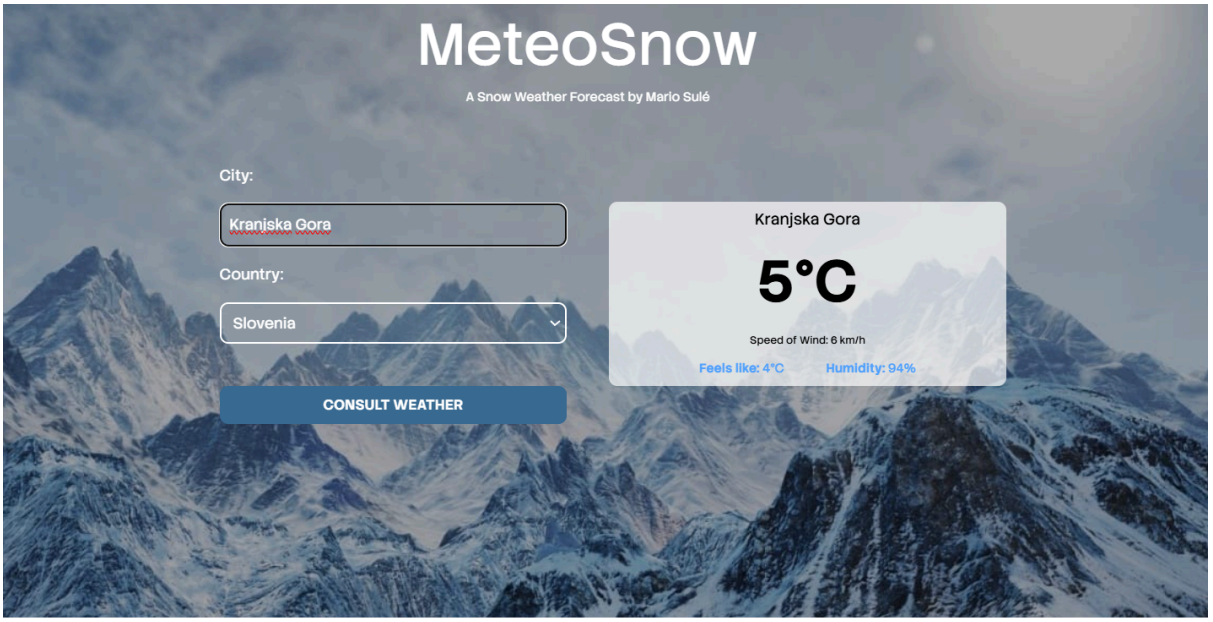
MeteoSnow

Project Documentation - Mario Sulé Domínguez

https://github.com/mariosulee/Pulse-Zustand-React-Hook-Form.git



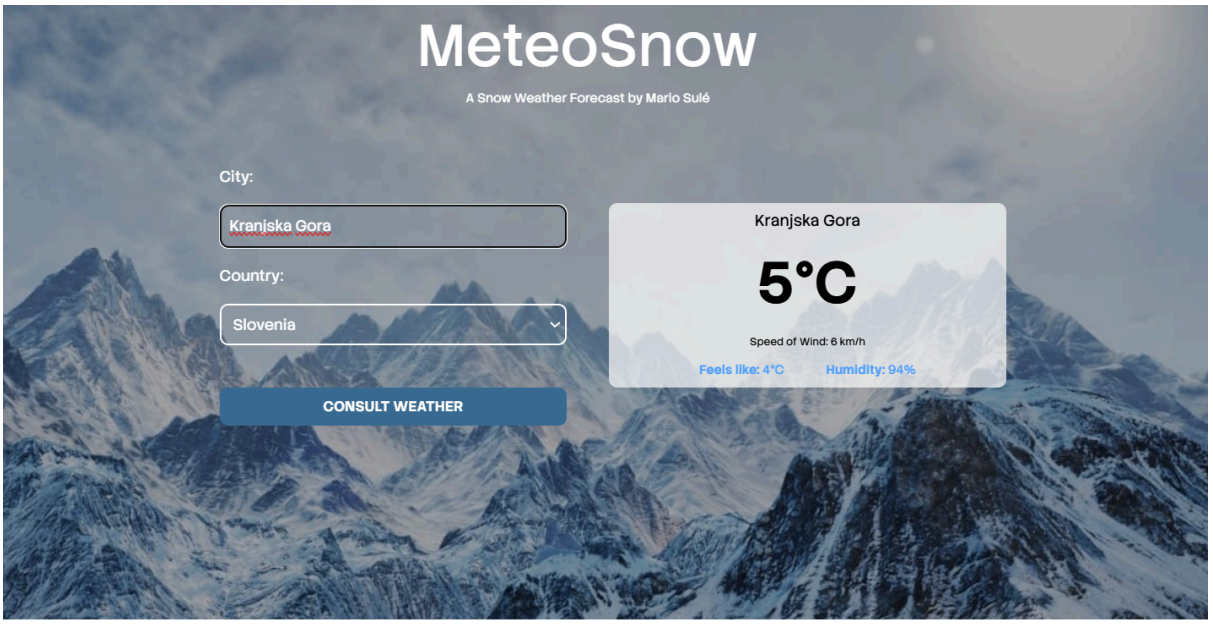
0. SUMMARY

MeteoSnow is a weather forecast web application built with React and TypeScript, using CSS Modules for styling. It is built using a modular design with custom hooks and local state management, as well as different components. The app allows users to check the current weather for a city in a selection of countries, including the USA, Mexico, Spain, Slovenia, Czech Republic, and Austria. It fetches data from the OpenWeather API, displaying temperature, feels-like, humidity, and wind speed. The app is fully responsive and designed to work smoothly on both desktop and mobile devices.

MeteoSnow

Documentación del proyecto - Mario Sulé Domínguez

https://github.com/mariosulee/Pulse-Zustand-React-Hook-Form.git



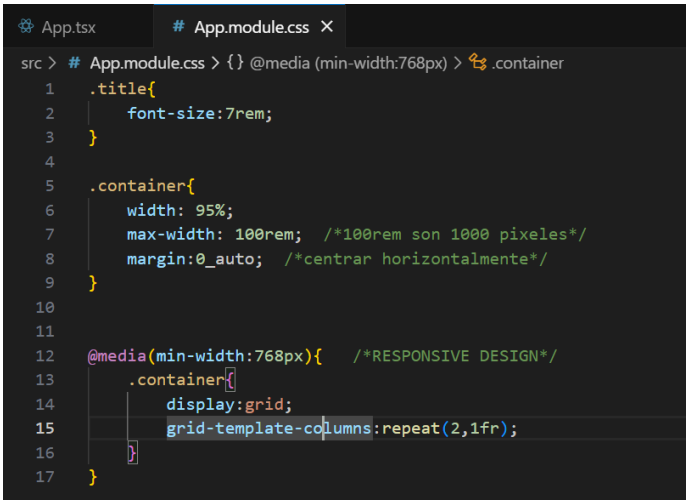
0. RESUMEN DE LA APLICACIÓN WEB DESARROLLADA

MeteoSnow es una aplicación web de previsión meteorológica construida con React y TypeScript, usando CSS Modules para el estilo. Está diseñada con un enfoque modular, utilizando custom hooks y gestión de estado local, además de distintos componentes. La app permite a los usuarios consultar el clima actual de una ciudad en una selección de países, incluyendo EE. UU., México, España, Eslovenia, República Checa y Austria. Obtiene los datos de la API de OpenWeather, mostrando temperatura, sensación térmica, humedad y velocidad del viento. La aplicación es totalmente responsive, funcionando correctamente tanto en escritorio como en dispositivos móviles.

1. INTRO A CSS MODULES Y CREACION DE FORMULARIO (Form.tsx)

Se debe crear una API Key en OpenWeather. Para importar una fuente de google fonts, la busco en su web y hago un copia pega del <link> que se inserta en mi index.html.

En este proyecto se utilizan módulos de CSS para el diseño en vez de TailWindCSS. Estos son una forma de aplicar CSS a un solo componente. Si por ejemplo quiero aplicarlo a mi App.tsx creo un nuevo archivo App.module.css en el cual escribiré reglas de CSS puro, de la siguiente forma (ver imagen). El punto que precede a los nombres significa que es una clase que en App.tsx se especifica de la siguiente forma, habiendo hecho el import styles from " …" previamente:




```
<h1 className={styles.title}>MeteoSnow</h1>
```

Seguidamente, se crea el formulario en el nuevo componente **Form.tsx** para buscar el clima y se le da diseño con su correspondiente módulo. Luego se crea un objeto state para guardar la ciudad y el país de la siguiente forma:

```
export default function Form(){

    // ESTADO GLOBAL
    const[search, setSearch]=useState<SearchType>({
        city: '',
        country:''
    });
```

En el input de city y del select del form lo pongo de la siguiente forma en el parámetro value

```
<input value={search.city}        y        <select value={search.country}
```

También se cambia el parámetro **onChange** de estos campos y se llama a la nueva función handleChange.

Para validar el formulario creo la función **handleSubmit** y el nuevo estado **alert**, junto con su componente para mostrarse que toma un children:

```
//VALIDACION DEL FORMULARIO
const[alert,setAlert]=useState('')

const handleSubmit=(e:React.FormEvent<HTMLFormElement>) => {
    e.preventDefault() //para manejar el formulario como quiero

    if(search.city==='' || search.country===''){
        setAlert("All fields are mandatory")
        return
    }

}


return(
    <>
        <form className={styles.form} onSubmit={handleSubmit}>

            {alert && <Alert>{alert}</Alert>}
```

2. CREACIÓN DE UN CUSTOM HOOK PARA MANEJAR LA API (useWeather.ts)

En este custom hook **useWeather.ts** creado en la nueva carpeta hooks, se crea la función fetchWeather que se pasa al componente principal usando el hook, y del componente principal al Form se pasa vía props. Esta función es llamada posteriormente a que se valide el formulario.

Para hacer la primera llamada a la API, instalo **Axios**, que es una librería de JS que permite hacer peticiones HTTP a APIs o servidores. La instalo escribiendo en terminal npm i axios y la importo en mi custom hook.

La función **fetchWeather** va a ser async, esto quiere decir que es una función que tardará tiempo en ejecutarse, y que por ello no debe bloquear el resto del código. Esta función toma como parámetro una búsqueda de tipo SearchType y usa await para esperar que la promesa de axios se resuelva antes de seguir. La API key se debe ocultar en una variable de entorno ya que es información sensible. Esta variable se guarda en el archivo **.env.local** La función queda así inicialmente:

```
const fetchWeather= async (search:SearchType)=> {

    const APIkey= import.meta.env.VITE_API_KEY // mi clave para acceder a la API, se debe ocultar
    try{
        const geoURL= `http://api.openweathermap.org/geo/1.0/direct?q=${search.city},${search.country}&appid=${APIkey}`

        const {data}= await axios(geoURL)   //hace una peticion HTTP GET a la URL. el destructuring de data es para extraer la prop
        console.log(data)
    }catch(error){
        console.log("Error al consultar la API: " , error)
    }
}
```

Ahora se debe hacer la segunda llamada a la API, en la que se consulta el clima. La primera daba parámetros como la comunidad autónoma del lugar. Se requiere latitud y longitud para obtener el clima actual.

```
//segunda llamada a la api
const lat=data[0].lat
const lon=data[0].lon

const weatherURL = `https://api.openweathermap.org/data/2.5/weather?lat=${lat}&lon=${lon}&appid=${APIkey}&units=metric`;
const {data: data2}=await axios(weatherURL)
console.log(data2)
```

Follow link (ctrl + click)

Hecho esta segunda llamada que ya incluye información sobre el clima, se tipa este resultado ya que da un array con muchos elementos que no se necesitan:

```
export type Weather={
    name:string,
    main: {
        temp:number,
        feels_like: number,
        humidity:number
    }

    wind: {
        speed: number
    }

}
```

Seguidamente se coloca en un nuevo state **weather** la respuesta que se obtiene (esto dentro del custom hook):

```
    const[weather,setWeather]=useState<Weather>({
        name:'',
        main:{
            temp:0,
            feels_like:0,
            humidity:0
        },
        wind:{
            speed:0
        }
    })
```

En el fetchWeather llamo a la función del state setWeather y luego creo un nuevo componente que se renderizará a la derecha del formulario. Este componente es **WeatherDetail.tsx**

3. CREACIÓN DEL COMPONENTE QUE MUESTRE EL TIEMPO (WeatherDetail.tsx)

Para hacer que si el estado de weather está vacío, no se muestre nada, se debe crear en el custom hook la función hasWeather que usa useMemo para saber si el estado de weather tiene algo, y uso esta función en el componente App.

```
const hasWeather= useMemo(() => weather.name, [weather])
```

Hecho eso, vuelvo al componente WeatherDetail y se escriben los parámetros de sensación térmica, humedad, velocidad del viento…etc.

Para mostrar un spinner de carga en lo que se muestra la información, vuelvo al custom hook y creo un state **loading** que se establezca a true una vez comienza la función fetchWeather, es decir, cuando se ha pulsado el botón submit. Para ello se pone también a false una vez muestre algo, con el try / catch / finally. Paso entonces el estado al componente principal y creo un nuevo componente que se llama en App.tsx cuando loading es true. Este nuevo componente es **Spinner.tsx** y en el .tsx de ese componente pongo el html de lo que dice esta página https://tobiasahlin.com/spinkit/ y se le adopta su código css.

Para que se muestre un mensaje indicando que no se ha encontrado una ciudad que no existe, se debe crear el nuevo estado local del custom hook llamado **notFound**, y antes de la latitud y longitud se comprueba si el array de data en la posición cero esta vacío. Si lo está, se cambia a true este estado.

```
try{
    const geoURL= `http://api.openweathermap.org/geo/1.0/direct?q=${search.cit

    const {data}= await axios(geoURL)   //hace una peticion HTTP GET a la URL

    //comprobar si existe la ciudad
    if(!data[0]){
        setNotFound(true)
        return //para que no se ejecute lo demas del codigo
    }

    //segunda llamada a la api
    const lat=data[0].lat
    const lon=data[0].lon

    const weatherURL = `https://api.openweathermap.org/data/2.5/weather?lat=${
```

Se exporta este estado al componente principal y si está a true se muestra el componente Alert que lo reutilizo ya que estaba creado de antes.

```
{/* SI EL ESTADO LOCAL LOADING ESTA A TRUE */}
{loading && <SpinnerLoading/>}

{/* SI EL ESTADO LOCAL NOTFOUND ESTA A TRUE */}
{notFound && <Alert>City not found</Alert>}


{hasWeather && (
  <WeatherDetail
    weather={weather}/>
)}
```

4. DEPLOYMENT (es distinto al tener variables de entorno)

Al incluir el archivo .env.local (el cual es ignorado en GitHub) con el API KEY en el proyecto se debe hacer primero npm run build como siempre, que genera la carpeta dist. Posteriormente, al subirlo a netlify añado una environment variable con el nombre VITE_API_KEY como en ese archivo), e introducir su valor. De esta forma ya funcionará la aplicación.