28-11-2022

Carlos Andrés Vargas Ramírez

Arquitectura de aplicaciones

Consumo de API Externa en React

OpenWeatherMap



Contenido

[Consumo de API Externa mediante Framework React 2](#_Toc120682502)

[Introducción 2](#_Toc120682503)

[Solución 2](#_Toc120682504)

[Vista Física 2](#_Toc120682505)

[Vista de Despliegue 3](#_Toc120682506)

[Desarrollo 3](#_Toc120682507)

[Instalación 3](#_Toc120682508)

[Desarrollo Inicial 5](#_Toc120682509)

[Integración de API 8](#_Toc120682510)

[Proceso de Carga de Información 12](#_Toc120682511)

[Tarjeta para Presentación de Datos 14](#_Toc120682512)

[Visualización de Datos 16](#_Toc120682513)

[Resultados 23](#_Toc120682514)

[Conclusiones 24](#_Toc120682515)

[Bibliografía 24](#_Toc120682516)

# Consumo de API Externa mediante Framework React

## Introducción

En los últimos anos la transformación digital de las empresas y organizaciones externas, han permitido a que cualquier tipo de persona u otras organizaciones tengan acceso a miles de aplicaciones e interfaces con el propósito de simplificar sus rutinas y procesos diarios incluso de forma integrada, y es aquí donde entran las APIs, ya que, existen diversos tipos de APIs, ya sean estas públicas o privadas.

Es por esto, que para el consumo de una API externa para el desarrollo de pruebas dentro de la materia de Arquitectura de Aplicaciones, he tomado la decisión de usar una API publica, llamada “OpenWeatherMap”, la cual nos da la disponibilidad de saber en tiempo real la temperatura y otras diversas informaciones acerca de una ciudad en especifica.

En el siguiente informe se hace un detallado especifico de cómo se implementó el API y cuál fue el resultado que se obtuvo.

## Solución

Para el desarrollo de la solución, se utilizó Visual Studio Code en su última versión más estable, además de Nodejs y framework React.

Se utilizó este framework, ya que es uno de los más utilizados ahora, además de crear sitios web dinámicos de una sola página, como por ejemplo los paneles de control, que es básicamente lo que se usara dentro del proyecto.

Una vez ya elegida la tecnología con la que se iba a trabajar, debía elegir una API, la cual tuviese mucha información de donde poder elegir. Es por esto que se usó OpenWeatherMap, ya que, dentro de sus APIs, además de encontrar soluciones a tiempo real como Current Weather, existe Forecast, el cual nos permite obtener una predicción de las próximas 3,6 y 9 horas.

### Vista Física

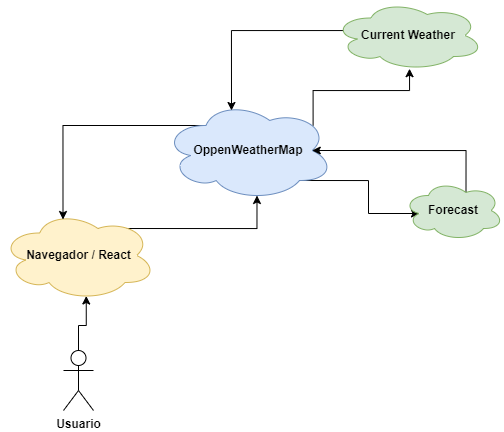


Figura 1 Comunicacion entre Usuario y la API por métodos HTTP

### Vista de Despliegue

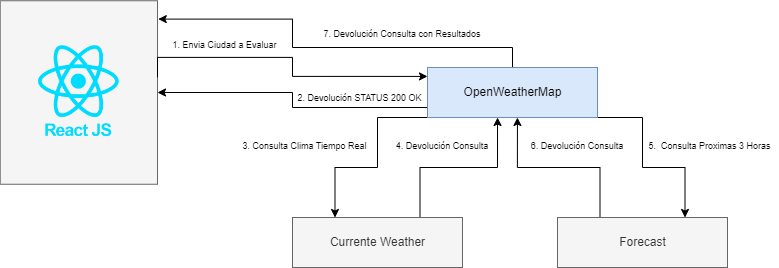
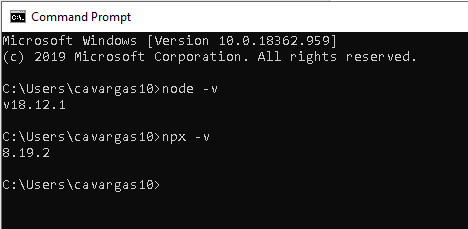


Figura 2 Vista de Despliegue y Comunicación

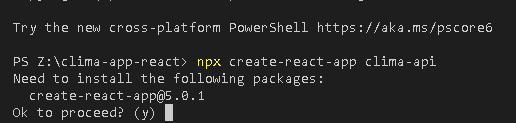
## Desarrollo

### Instalación

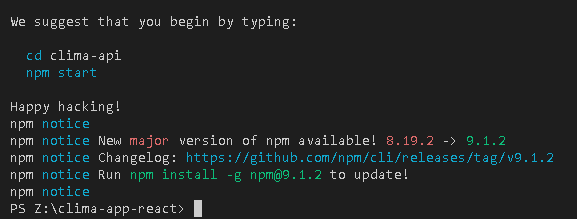
1. Lo primero que debemos hacer es instalar Visual Studio Code, en su versión mas estable. Esto lo podremos hacer desde el siguiente enlace: <https://code.visualstudio.com/>
2. Luego deberemos descargar e instalar Nodejs, ya que React, trabaja con esto por debajo. Esto se podrá realizar desde el siguiente enlace: <https://nodejs.org/en/>
3. Una vez instalado Nodejs, entraremos al cmd, para revisar que la instalación, haya sido exitosa.



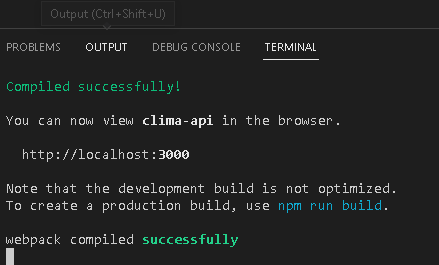
1. Seguido, abriremos VSC y ejecutaremos el terminal y colocaremos **npx create-react-app clima-app,** la cual nos creara la aplicación. Seguido deberemos confirmar la creación con **y**

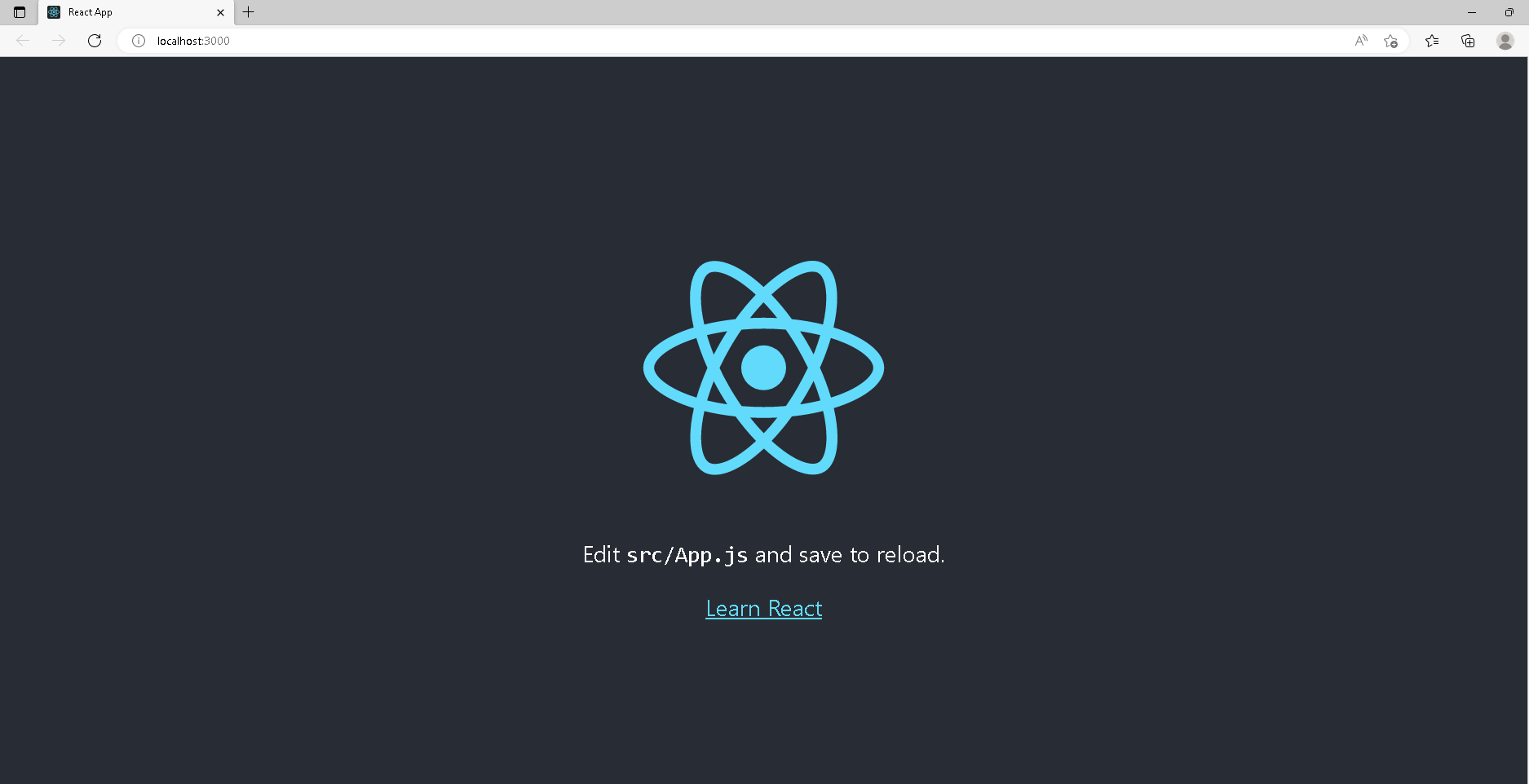


1. Una vez creado, deberemos inicializar el proyecto



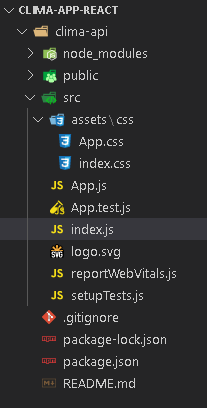
1. Una vez lanzado, nos aparecerá un mensaje de que el proyecto se ha compilado correctamente





### Desarrollo Inicial

1. Haremos un ordenamiento de clases. Esto lo hacemos, más que nada, para llevar un orden en la creación de nuestro proyecto.

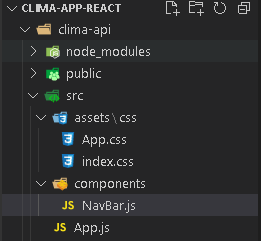


1. Debido al ordenamiento de clases deberemos re-direccionar los estilos, en las clases en las que estos, son importados. En mi caso, **App.js** y **index.js**.

import './assets/css/App.css';

import './assets/css/index.css';

1. Dentro de la carpeta **src**, creamos una carpeta llamada **Components** y dentro creamos un archivo **NavBar.js**



1. Nos dirigiremos al siguiente enlace: <https://getbootstrap.com/> y copiaremos el **cdn** para utilizar **Bootstrap**, para la creación de los estilos. Esto lo pegaremos dentro de la carpeta **public**, en **index.html**

<link

href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/css/bootstrap.min.css"

rel="stylesheet"

integrity="sha384-rbsA2VBKQhggwzxH7pPCaAqO46MgnOM80zW1RWuH61DGLwZJEdK2Kadq2F9CUG65"

crossorigin="anonymous"

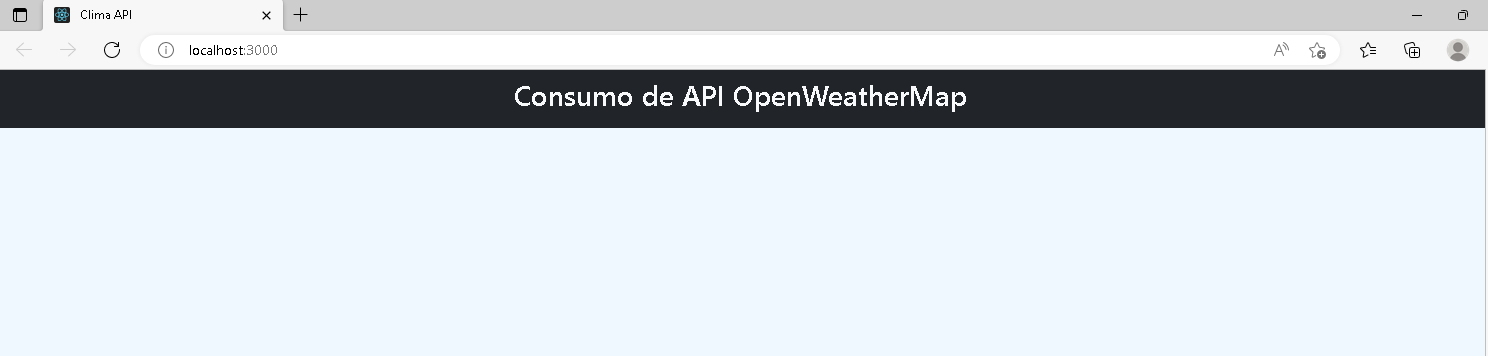
/>

1. Ahora sí, escribiremos código basado en **Bootstrap**, para la creación del **NavBar.js** y he importaremos esta clase, en **App.js**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | const NavBar = () => {  return(  <nav className="navbar bg-dark text-light mb-5">  <div className="container-fluid">  <h3 className="mx-auto">Consumo de API OpenWeatherMap</h3>  </div>  </nav>  );  }  export default NavBar; |

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | import './assets/css/App.css';  import NavBar from './components/NavBar';  function App() {  return (  <div className="App">  <NavBar />  </div>  );  }  export default App; |

1. Nos dirigiremos al navegador donde se esté ejecutando la llamada de **localhost:3000**, para verificar que la creación del **NavBar**, haya sido exitosa.

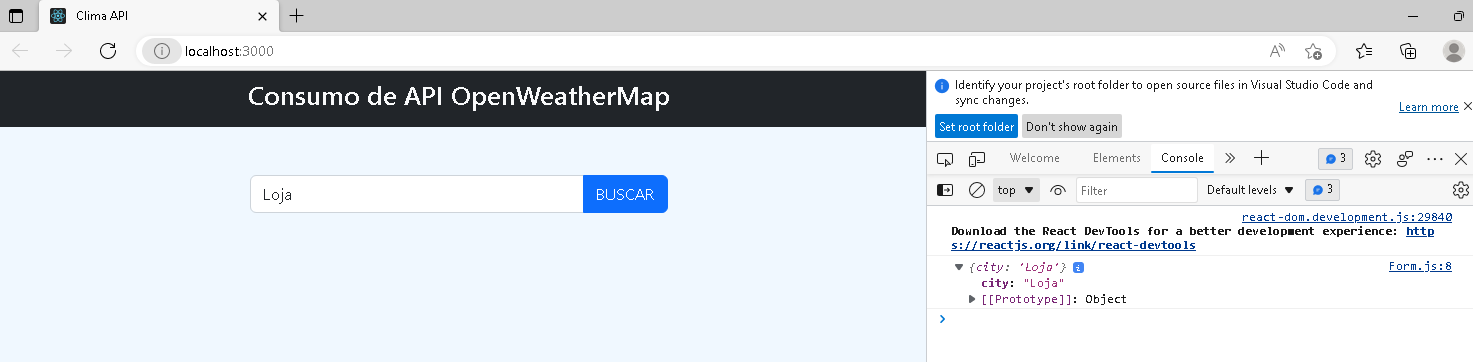


1. Dentro de la carpeta **Components**, crearemos la clase **Form.js**, y haremos su importación dentro de la clase **App.js**
2. En la línea 1, hacemos la importación de useState.
3. En la línea 3 creamos la función, que nos ayudara a establecer la ciudad que vamos a buscar.
4. Desde la línea 6, hasta la línea 12, creamos la función **onSubmit**,
5. En la línea 17 hacemos un llamado a la función **onSubmit**.
6. Desde la línea 16 hasta la línea 26 maquetamos el **input**, más el botón, con los que interactuara el usuario. Ademas, en la línea 21 creamos un evento **e**, que nos ayudará a recoger la información que ingresemos en el **input**, el cual solo se recogerá cuando pulsemos el botón.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31 | import React, {useState} from 'react';  const Form = ({newLocation}) => {  const [city, setCity] = useState("");  const onSubmit = (e) => {  e.preventDefault();  console.log({city});  if(city === "" || !city) return;  newLocation(city);  }  return(  <div className="container">  <form onSubmit={onSubmit}>  <div className="input-group mb-3 mx-auto">  <input type="text" className="form-control"  placeholder="Ciudad"  onChange={(e) =>setCity(e.target.value)}/>  <button className="btn btn-primary input-group-text"  type="submit">BUSCAR</button>  </div>  </form>  </div>  );  }  export default Form; |

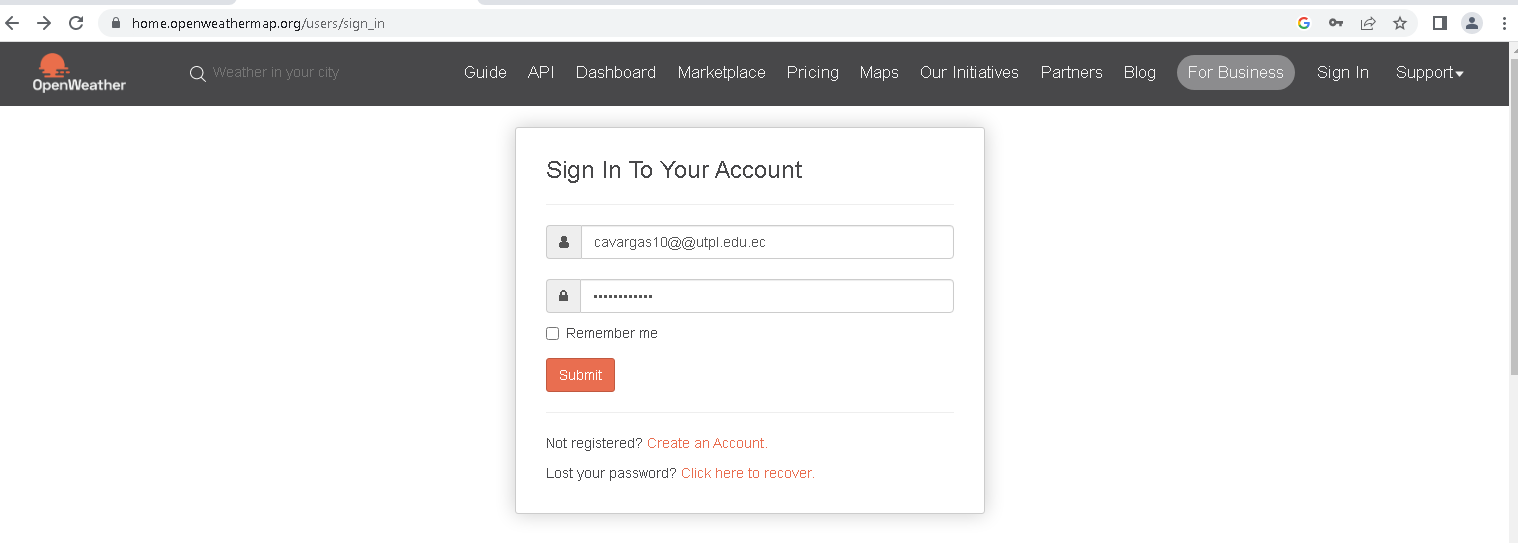
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | import './assets/css/App.css';  import NavBar from './components/NavBar';  import Form from './components/Form';  function App() {  return (  <div className="App">  <NavBar />  <Form />  </div>  );  }  export default App; |

1. Nos dirigiremos al navegador donde se esté ejecutando la llamada de **localhost:3000**, para verificar que la creación del **Form**, haya sido exitosa y este recogiendo la información.

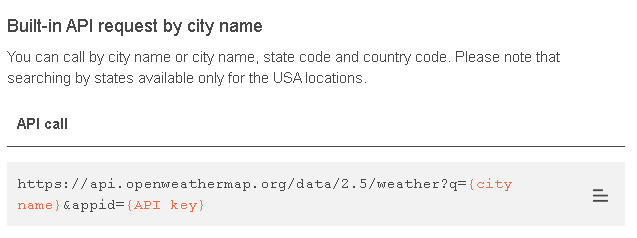


### Integración de API

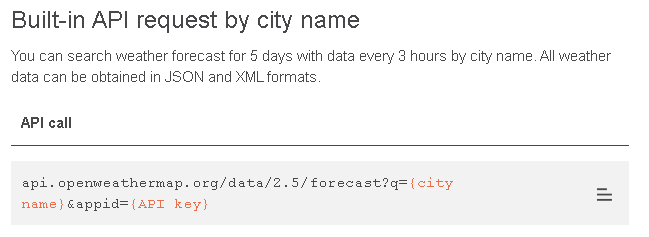
1. Es momento de utilizar el API, pero antes, deberemos dirigirnos al siguiente enlace: <https://openweathermap.org/> donde, si no tenemos cuenta, deberemos registrarnos y luego iniciar sesión.



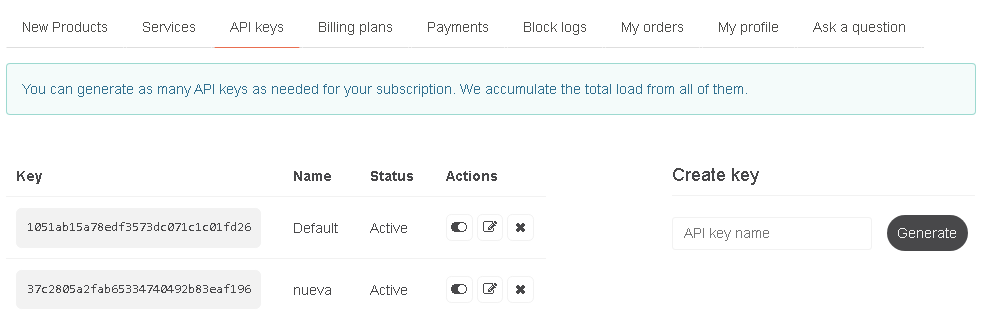
1. Nos dirigiremos al siguiente enlace: <https://openweathermap.org/current#name> , en este podremos encontrar el API Current Weather.



1. Nos dirigiremos al siguiente enlace: <https://openweathermap.org/forecast5#name5> , en este podremos encontrar el API Forecast



1. Por último, para poder hacer uso de las APIs, anteriormente nombradas, necesitaremos un API KEY, y esto nos lo ofrece la misma página, en el siguiente enlace: <https://home.openweathermap.org/api_keys>



1. Ahora, lo único que tendremos que hacer, será crear una clase, en mi caso **WeatherPanel.js**, para hacer la integración y llamada de las APIs, agregando el Api Key.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | import React, {useState} from 'react';  import Form from './Form';  const WeatherPanel = () => {  let urlWeather =  "https://api.openweathermap.org/data/2.5/weather?  appid=37c2805a2fab65334740492b83eaf196&lang=es";  let cityUrl = "&q=";  let urlForecast =  "https://api.openweathermap.org/data/2.5/forecast?  appid=37c2805a2fab65334740492b83eaf196&lang=es" |

1. Definimos las diferentes variables de estado para **useState**
   1. Línea 11, respuesta del tiempo actual, con la función del **setWeather**.
   2. Línea 12, respuesta a predicción de las siguientes horas, con la función de **setForecast**.
   3. Línea 13, respuesta del spinner para visualizar que se carga la información (esto se creara después, pero lo declaramos de una vez), con la función **setLoading**, con un valor **booleano** de **false**.
   4. Línea 14, respuesta de la **tarjeta** de visualización (esta, también se creará después) con la función **setShow**, con un valor **booleano** de **false**.
   5. Línea 15, respuesta de comunicación con el formulario, con la función setLocation, con un valor de **texto** vacío.
   6. Línea 17, función asíncrona que recogerá la localización de la ciudad.
      1. Dentro tenemos **setLoading**, el cual se volverá **true** cuando hagamos la llamada, esto quiere decir, que el **spinner** se hará visible.
      2. Modificaremos el parámetro vacío de **setLocation**, y lo cambiaremos por la ciudad ingresada.

|  |  |
| --- | --- |
| 11  12  13  14  15 | const [weather, setWeather] = useState([]);  const [forecast, setForecast] = useState([]);  const [loading, setLoading] = useState(false);  const [show, setShow] = useState(false);  const [location, setLocation] = useState(""); |

1. Debemos completar la **urlWeather**, concatenando **cityUrl** y la locación (**loc**). Lo siguiente, será usar el **API Fetch**. Y la verificación de errores con **throw** y **cath**

|  |  |
| --- | --- |
| 23  24  25  26  27  28  29  30  31  32  33  34  35 | urlWeather = urlWeather + cityUrl + loc;  await fetch(urlWeather).then((response) =>{  if(!response.ok) throw {response}  return response.json();  }).then((weatherData) =>{  console.log(weatherData);  setWeather(weatherData);  }).catch(error =>{  console.log(error);  setLoading(false);  setShow(false);  }); |

1. Debemos completar la **urlForecast**, concatenando **cityUrl** y la locación (**loc**). Lo siguiente, será usar el **API Fetch**. Y la verificación de errores con **throw** y **cath**

|  |  |
| --- | --- |
| 39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55 | urlForecast = urlForecast + cityUrl + loc;  await fetch(urlForecast).then((response) =>{  if(!response.ok) throw {response}  return response.json();  }).then((forecastData) =>{  console.log(forecastData);  setForecast(forecastData);  setLoading(false);  setShow(true);  }).catch(error =>{  console.log(error);  setLoading(false);  setShow(false);  }); |

1. Primero antes de seguir cabe recalcar, que hubo un cambio de planes y toco cambiar la importación y visualización del **From.js** en **App.js** y ahora importaremos el **Form.js** en **weatherPanel.js**.
2. Dentro del **return**, creamos un **Reac.Fracment** y a continuación incluimos la llamada **Form** y dentro incluimos un **prop**, que se llamara **newLocation**, el cual obtendrá la ciudad.

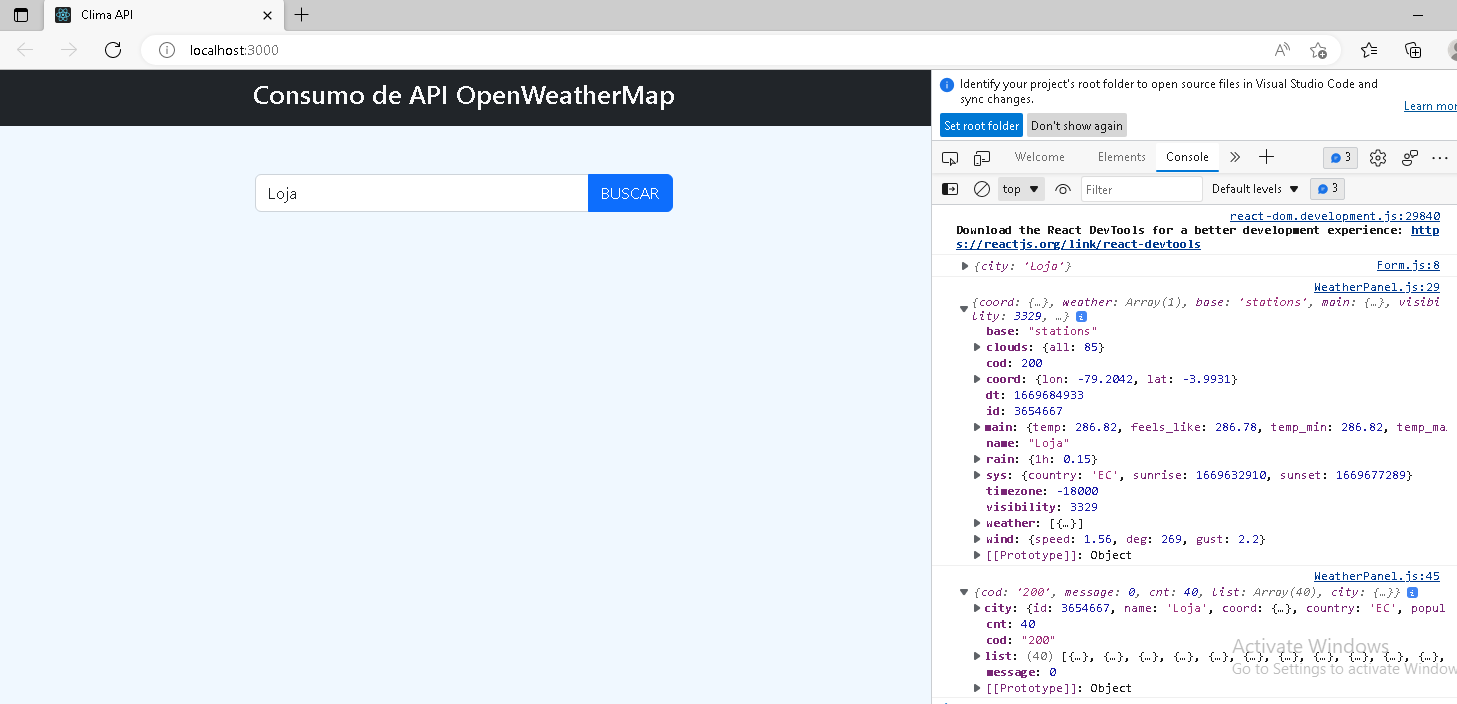
|  |  |
| --- | --- |
| 61  62  63  64  65  66  67  68  69  70  71 | return(  <React.Fragment>    <Form  newLocation = {getLocation}  />  </React.Fragment>  ); |

1. Entonces el **prop**, creado anteriormente lo pasaremos en **Form.js**. Y cuando vamos a hacer uso del prop, pues en la línea 11 decimos que en caso de que tengamos la ciudad, le pasamos **city** a **newLocation**. Y ya por último, importaremos y presentaremos en **App.js** el **weatherPanel.js.**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | import React, {useState} from 'react';  const Form = ({newLocation}) => {  const [city, setCity] = useState("");  const onSubmit = (e) => {  e.preventDefault();  console.log({city});  if(city === "" || !city) return;  newLocation(city);  } |

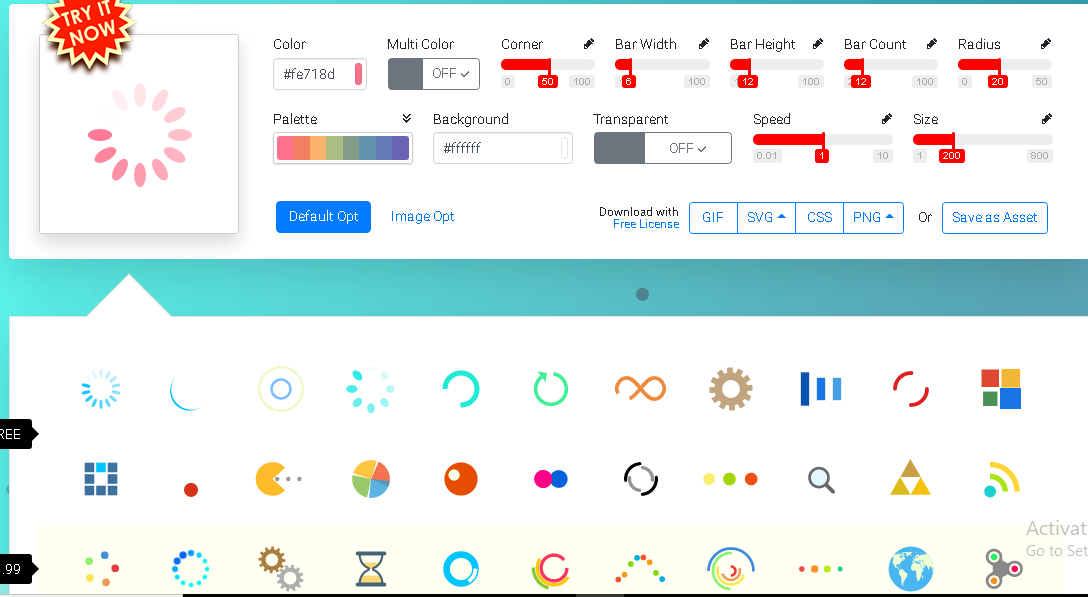
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | import './assets/css/App.css';  import NavBar from './components/NavBar';  import Form from './components/Form';  function App() {  return (  <div className="App">  <NavBar />  <Form />  </div>  );  }  export default App; |

1. Nos dirigiremos al navegador donde se esté ejecutando la llamada de **localhost:3000**, para verificar que haya sido exitoso el llamamiento de los datos.

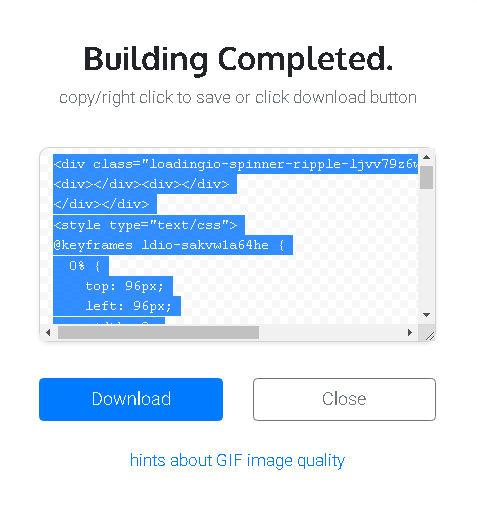


### Proceso de Carga de Información

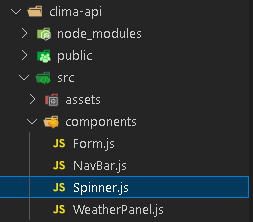
1. Nos dirigiremos al siguiente enlace: <https://loading.io/> Dentro de esta página, existen modelos hechos de procesos de carga, solo nos deberemos registrar, elegir uno a nuestra conveniencia.

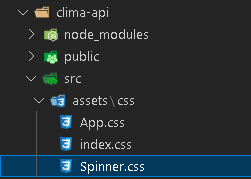


1. Una vez hayamos escogido uno a nuestro gusto, la página nos arrojara su construcción, solo deberemos copiar y pegar el código.

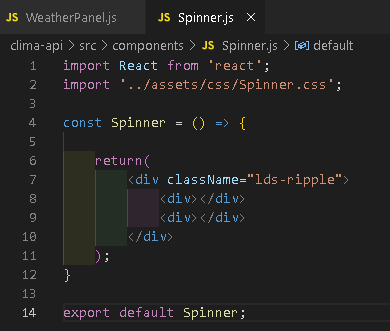


1. Para hacer uso del spinner de carga que hemos descargado, antes tendremos que crear un nuevo componente dentro de la carpeta **src**, y dentro del módulo de **components**, a esta la llamaremos **Spinner.js.** Y dentro de **assets/css** crearemos una hoja de estilos para nuestro componente, a este le llamaremos **Spinner.css**, y pegaremos los estilos.





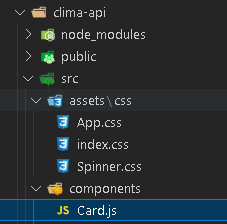
1. Dentro de **Spinner.js**, importaremos el css que ya habíamos creado, y dentro del return, pegaremos el código html que nos proporcionó la página.



1. Para poder visualizar como quedo nuestro Spinner, primero deberemos crear una tarjeta que se utilizara para la presentación de los datos.

### Tarjeta para Presentación de Datos

1. Dentro de **components**, crearemos **Card.js**, e importaremos el **Spinner.js**



1. Luego, dentro de **weatherPanel**, importaremos **Card.js** y en el **return**, le pasaremos algunos props, dentro de la llamada que le hacemos a **Card**
   1. Línea 71, showData para visualizar información.
   2. Línea 72, loadingData para controlar el spinner
   3. Línea 73, weather para visualizar datos en tiempo real
   4. Línea 74, forecast para visualizar la predicción de las siguientes horas

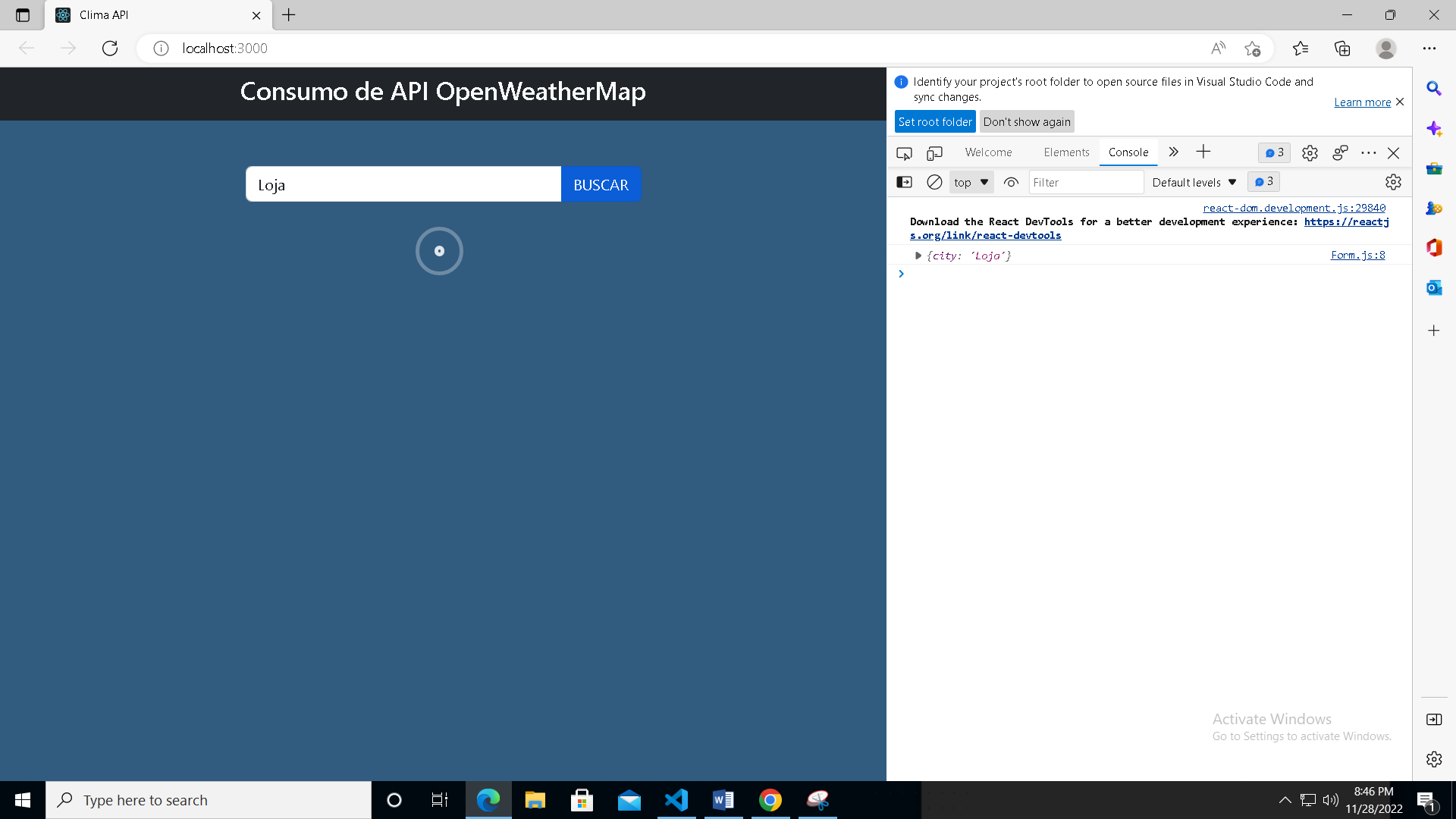
|  |  |
| --- | --- |
| 1  2  3 | import React, {useState} from 'react';  import Form from './Form';  import Card from './Card'; |

|  |  |
| --- | --- |
| 62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80 | return(  <React.Fragment>    <Form  newLocation = {getLocation}  />  <Card  showData = {show}  loadingData = {loading}  weather = {weather}  forecast = {forecast}  />  </React.Fragment>  ); |

1. Ahora nos dirigiremos a **Card.js** y le pasaremos todos los **props** que creamos. Y, además, le crearemos una condición de que, si es verdad **loadingData**, nos presente el componente **Spinner.** Para esto en el **return** colocamos una etiquete h1, simplemente para que no nos de error.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | import React from "react";  import Spinner from "./Spinner";  const Card = ({ loadingData, showData, weather, forecast }) => {    if (loadingData) {  return <Spinner />;  }    return (  <h1> </h1>  );  };  export default Card; |

1. Nos dirigiremos al navegador donde se esté ejecutando la llamada de **localhost:3000**, para verificar que haya sido exitoso el llamamiento del **Spinner**. Esto lo haremos ingresando el nombre de una ciudad y dando clic en **BUSCAR**.

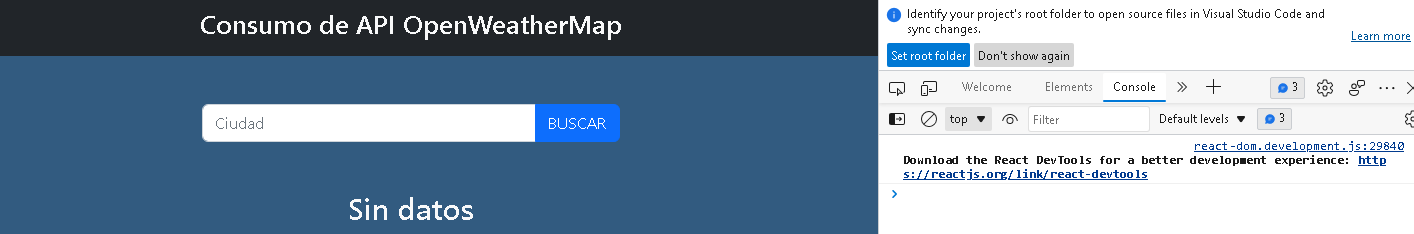


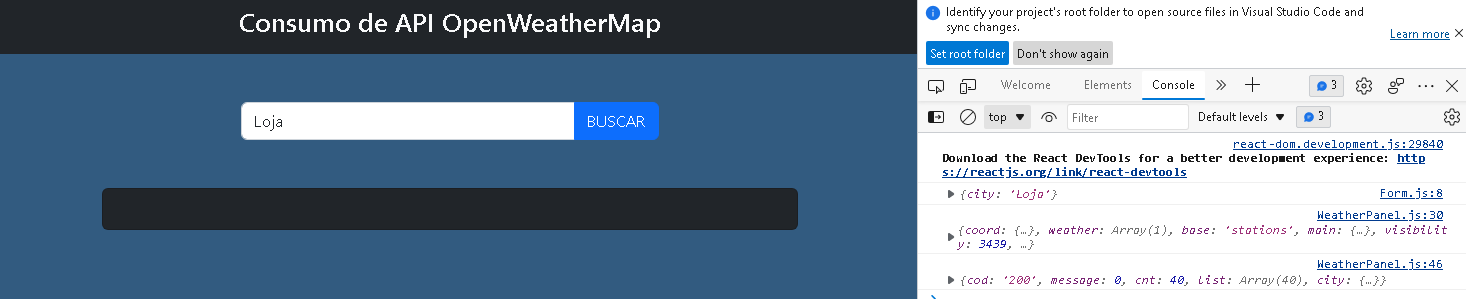
1. Ahora, simplemente debemos crear una estructura donde se muestra la tarjeta cuando encuentre datos, y si no los encuentra que nos muestre un mensaje. Esto lo haremos gracias a showData en la línea 11, en el cual, si es true, muestra la tarjeta, caso contrario en la línea 27, muestra un mensaje “Sin datos”.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | import React from "react";  import Spinner from "./Spinner";  const Card = ({ loadingData, showData, weather, forecast }) => {  if (loadingData) {  return <Spinner />;  }  return (  <div className="mt-5">  {showData === true ? (  <div className="container">  <div className="card mb-3 mx-auto bg-dark text-light">  <div className="row g-0">  <div className="col-md-4">    </div> |

|  |  |
| --- | --- |
| 18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33 | <div className="col-md-8">  <div className="card-body text-start mt-2">    </div>  </div>  </div>  </div>  </div>  ) : (  <h2 className="text-light">Sin datos</h2>  )}  </div>  );  };  export default Card; |

1. Nos dirigiremos al navegador donde se esté ejecutando la llamada de **localhost:3000**, para verificar que haya sido exitoso. En la primera grafica podemos observar que aún no se le ha dado al botón de **BUSCAR**, y ya nos muestra el mensaje que configuramos antes, y en la segunda imagen, ya hemos colocado una ciudad y hemos dado clic en **BUSCAR**, es por eso que nos muestra la **Card** que estamos creando.



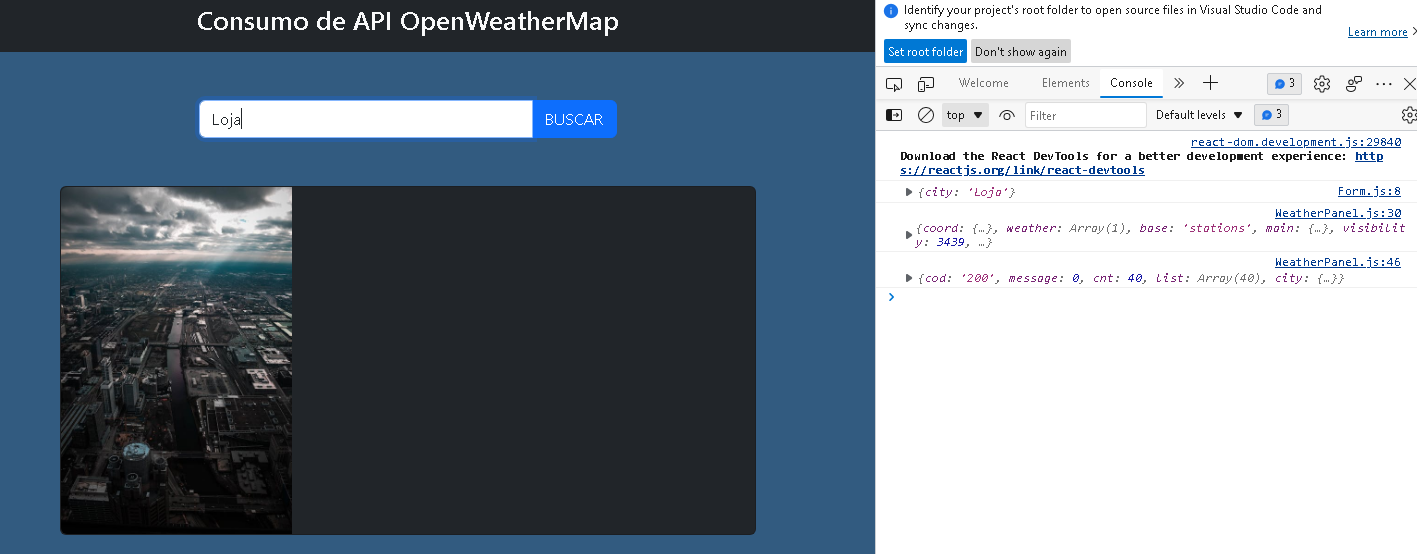


### Visualización de Datos

1. Para ya ir culminando con el proyecto, vamos a mejor el diseño de nuestra **Card**, para eso, nos dirigiremos al siguiente enlace: <https://www.pexels.com/> Esta es una página web, donde podremos descargar imágenes gratis. Elegiremos una que nos agrade y la integraremos a nuestro código.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | <div className="container">  <div className="card mb-3 mx-auto bg-dark text-light">  <div className="row g-0">  <div className="col-md-4">  <img  src="https://images.pexels.com/photos/1707820/  pexels-photo-1707820.jpeg?auto=compress&  cs=tinysrgb&w=1260&h=750&dpr=1"  className="img-fluid rounded-start"  alt="..."  />  </div> |

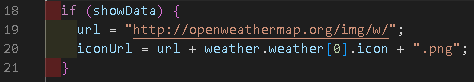
1. Nos dirigiremos al navegador donde se esté ejecutando la llamada de **localhost:3000**, para verificar que haya sido exitoso.



1. Una de las informaciones importantes para el usuario, es saber la fecha actual y para establecerla, crearemos variables.
   1. Línea 6, variable para obtener el día, con **getDate**
   2. Línea 7, variable para obtener el mes, con **getMonth**
   3. Línea 8, variable para obtener el año, con **getFullYear**
   4. Línea 9, concatenamos las variables para su presentación
   5. Línea 11 y 12, variables **url** y **iconUrl** de cadena vacia para obtener el icono directamente de la API

|  |  |
| --- | --- |
| 4  5  6  7  8  9  10  11  12 | const Card = ({ loadingData, showData, weather, forecast }) => {  var today = new Date();  var day = today.getDate();  var month = today.getMonth() + 1;  var year = today.getFullYear();  var date = day + "/" + month + "/" + year;  var url = "";  var iconUrl = ""; |

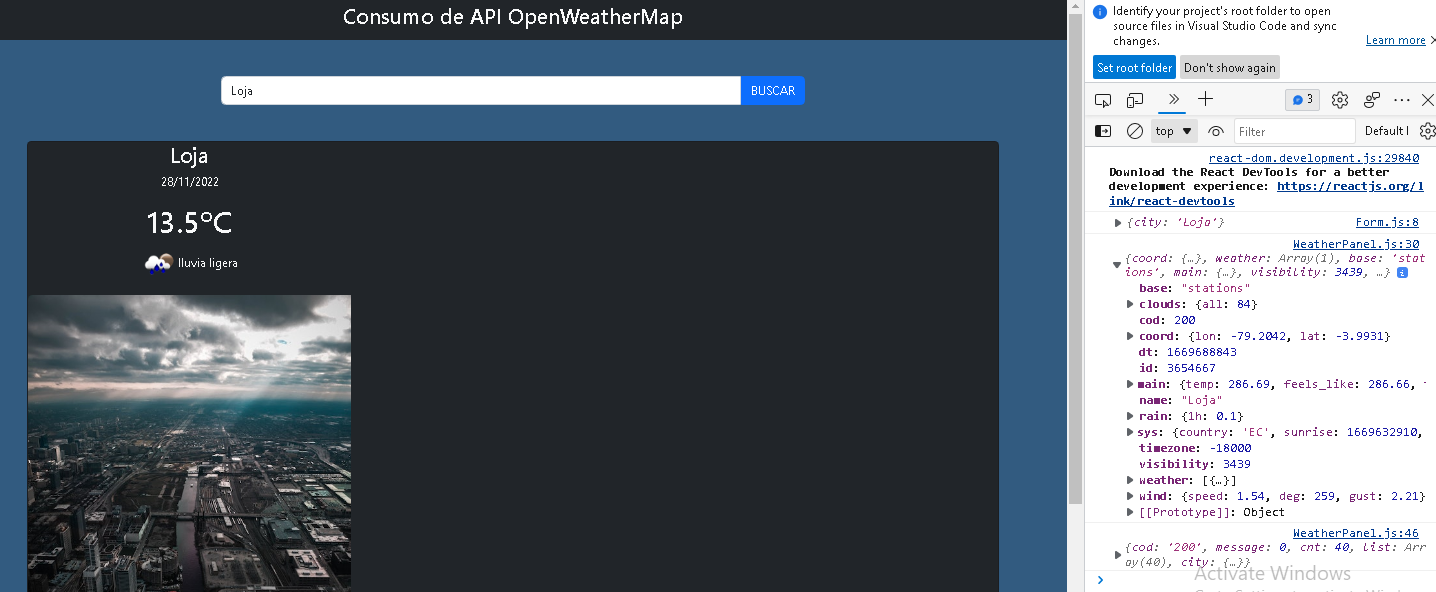
1. Creamos una condición, para saber cuándo se deberían mostrar los datos. Usando **showData**



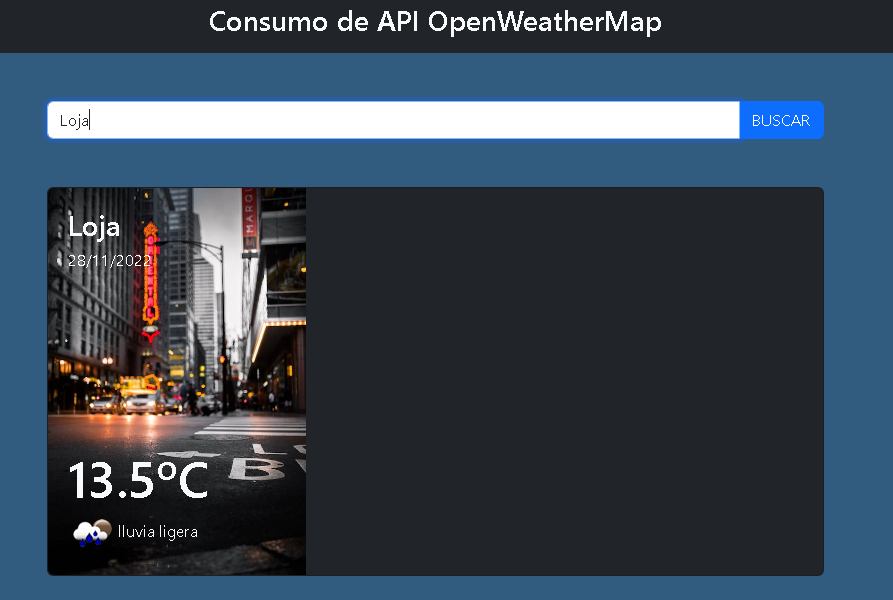
1. Estableceremos datos que se visualizaran dentro de la imagen que habíamos escogido antes.
   1. Línea 30, usamos **weather.name** para asignar el nombre de la ciudad.
   2. Línea 31, usamos **date** para asignar la fecha actual.
   3. Línea 33, usamos **weather.main.temp** para asignar la temperatura, pero como la api nos asigna la temperatura en grados Kelvin, debemos hacer una transformación a grados Celsius, con un solo decimal.
   4. Línea 36, usamos **iconUrl**, para traer el icono que nos da la API.
   5. Línea 37, usamos **weather.weather[0].description**, para obtener una descripcion del clima

|  |  |
| --- | --- |
| 23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46 | return (  <div className="mt-5">  {showData === true ? (  <div className="container">  <div className="card mb-3 mx-auto bg-dark text-light">  <div className="row g-0">  <div className="col-md-4">  <h3 className="card-title">{weather.name}</h3>  <p className="card-date">{date}</p>  <h1 className="card-temp">  {(weather.main.temp - 273.15).toFixed(1)}°C  </h1>  <p className="card-desc">  <img src={iconUrl} alt="icon" />  {weather.weather[0].description}  </p>  <img  src="https://images.pexels.com/photos/1707820/  pexels-photo-1707820.jpeg?auto=compress&  cs=tinysrgb&w=1260&h=750&dpr=1"  className="img-fluid rounded-start"  alt="..."  />  </div> |

1. Nos dirigiremos al navegador donde se esté ejecutando la llamada de **localhost:3000**, para verificar que haya sido exitoso.



1. Le agregamos un poco de estilos, cambiamos la imagen de referencia para que no se solapen los colores y nos dirigiremos al navegador donde se esté ejecutando la llamada de **localhost:3000**, para verificar que haya sido exitoso.



1. Ahora debemos trabajar, con los demás datos que vamos a mostrar en la tarjeta.
   1. Línea 48 y 49, para obtener la temperatura máxima usaremos **weather.main.temp\_max**, y transformamos a grados Celsius igual que hace un momento.
   2. Línea 52 y 53, para obtener la temperatura mínima usaremos **weather.main.temp\_min**, y transformamos a grados Celsius igual que hace un momento.
   3. Línea 56 y 57, para obtener la sensación térmica usaremos **weather.main.feels\_like**, y transformamos a grados Celsius igual que hace un momento.
   4. Línea 60, para obtener la humedad usaremos **weather.main.humidity**
   5. Línea 63, para obtener la velocidad del viento usaremos **weather.wind.speed**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | <div className="card-body text-start mt-2">  <h5 className="card-text">  TEMPERATURA MAXIMA:{" "}  {(weather.main.temp\_max - 273.15).toFixed(1)}ºC  </h5>  <h5 className="card-text">  TEMPERATURA MINIMA:{" "}  {(weather.main.temp\_min - 273.15).toFixed(1)}ºC  </h5>  <h5 className="card-text">  SENSACION TERMICA:{" "}  {(weather.main.feels\_like - 273.15).toFixed(1)}ºC  </h5>  <h5 className="card-text">  HUMEDAD: {weather.main.humidity}%  </h5>  <h5 className="card-text">  VELOCIDAD DEL VIENTO: {weather.wind.speed}m/s  </h5>  </div>  <hr /> |

1. Nos dirigiremos al navegador donde se esté ejecutando la llamada de **localhost:3000**, para verificar que haya sido exitoso.



1. Ahora vamos a trabajar, con la predicción para las siguientes horas: 3, 6 y 9 horas.
   1. Línea 14, 15 y 16, estableceremos las variables para los nuevos iconos de 3, 6 y 9 respectivamente.
   2. Línea 18, 19 y 20, estableceremos las variables para almacenar la fecha y horas de las predicciones correctamente

|  |  |
| --- | --- |
| 14  15  16  17  18  19  20 | var iconUrl3 = "";  var iconUrl6 = "";  var iconUrl9 = "";  var forecastDate3 = "";  var forecastDate6 = "";  var forecastDate9 = ""; |

1. Usamos la condicional de **showData**, para saber cuándo se deberían mostrar los datos de las predicciones de las siguientes horas.

|  |  |
| --- | --- |
| 26  27  28  29  30  31  32 | if (showData) {  url = "http://openweathermap.org/img/w/";  iconUrl = url + weather.weather[0].icon + ".png";  iconUrl3 = url + forecast.list[1].weather[0].icon + ".png";  iconUrl6 = url + forecast.list[2].weather[0].icon + ".png";  iconUrl9 = url + forecast.list[3].weather[0].icon + ".png"; |

1. Dentro de **showData**, concatenamos las fecas y horas para las predicciones de 3, 6 y 9 horas

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | forecastDate3 =  forecast.list[1].dt\_txt.substring(8, 10) +  "/" +  forecast.list[1].dt\_txt.substring(5, 7) +  "/" +  forecast.list[1].dt\_txt.substring(0, 4) +  " " +  forecast.list[1].dt\_txt.substring(11, 13);  forecastDate6 =  forecast.list[2].dt\_txt.substring(8, 10) +  "/" +  forecast.list[2].dt\_txt.substring(5, 7) +  "/" +  forecast.list[2].dt\_txt.substring(0, 4) +  " " +  forecast.list[2].dt\_txt.substring(11, 13);  forecastDate9 =  forecast.list[3].dt\_txt.substring(8, 10) +  "/" +  forecast.list[3].dt\_txt.substring(5, 7) +  "/" +  forecast.list[3].dt\_txt.substring(0, 4) +  " " +  forecast.list[3].dt\_txt.substring(11, 13); |

1. Establecemos la estuctura, para mostrar la fecha y hora, icono, descripción y temperatura de cada una de las predicciones

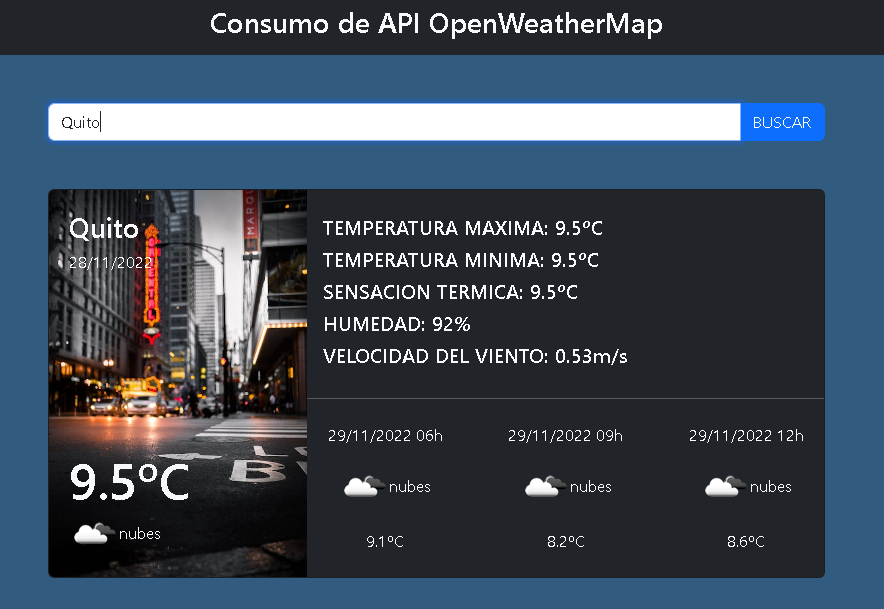
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31 | <div className="row mt-4">  <div className="col">  <p>{forecastDate3}h</p>  <p className="description">  <img src={iconUrl3} alt="icon" />  {forecast.list[1].weather[0].description}  </p>  <p className="temp">  {(forecast.list[1].main.temp - 273.15).toFixed(1)}ºC  </p>  </div>  <div className="col">  <p>{forecastDate6}h</p>  <p className="description">  <img src={iconUrl6} alt="icon" />  {forecast.list[2].weather[0].description}  </p>  <p className="temp">  {(forecast.list[2].main.temp - 273.15).toFixed(1)}ºC  </p>  </div>  <div className="col">  <p>{forecastDate9}h</p>  <p className="description">  <img src={iconUrl9} alt="icon" />  {forecast.list[3].weather[0].description}  </p>  <p className="temp">  {(forecast.list[3].main.temp - 273.15).toFixed(1)}ºC  </p>  </div> |

### Resultados

1. Pruebas ciudad “Loja”



1. Pruebas ciudad “Quito”



## Conclusiones

1. React es una librería completa, adecuada en muchos tipos de proyectos distintos. Nos permite un desarrollo ágil, ordenado y con una arquitectura mantenerle, focalizada en componentes y que nos ofrece un gran performance.
2. Se pudo observar que el uso de APIs, algunos momentos tiene cosas más complicadas que otras, pero a medida que se toma ritmo y se lee la documentación, se vuelve muy sencillo optar por tecnologías así.
3. Otro aspecto que realmente me sorprendió, fue que realmente la lógica del negocio no estuvo en ninguna parte, es decir no hubo una programación tediosa o cansada, para encontrar la solución a algún problema.
4. Otro punto de los cuales me gustaría rescatar, es que el uso de prácticas, taller o proyectos de esta índole, hace que el estudiante se sienta mucho más involucrado con su carrera y pueda obtener muchos más conocimientos del que podría obtener.

## Bibliografía

[1] Olawanle, J. (2022, 10 noviembre). *How to Consume REST APIs in React – a Beginner's Guide*. freeCodeCamp.org. <https://www.freecodecamp.org/news/how-to-consume-rest-apis-in-react/>

[2] OpenWeatherMap.org. (s. f.). *Current weather data - OpenWeatherMap*. <https://openweathermap.org/current>

[3] OpenWeatherMap.org. (s. f.-a). *5 day weather forecast - OpenWeatherMap*. <https://openweathermap.org/forecast5>

[4] Herrera, F. N. (2018, 16 agosto). *¿Como conectar y consumir un API con ReactJs? Parte 2 y ejemplo practico.* Medium. <https://medium.com/@kakuherrera/como-conectar-y-consumir-un-api-con-reactjs-parte-2-c4aa3fa244b4>