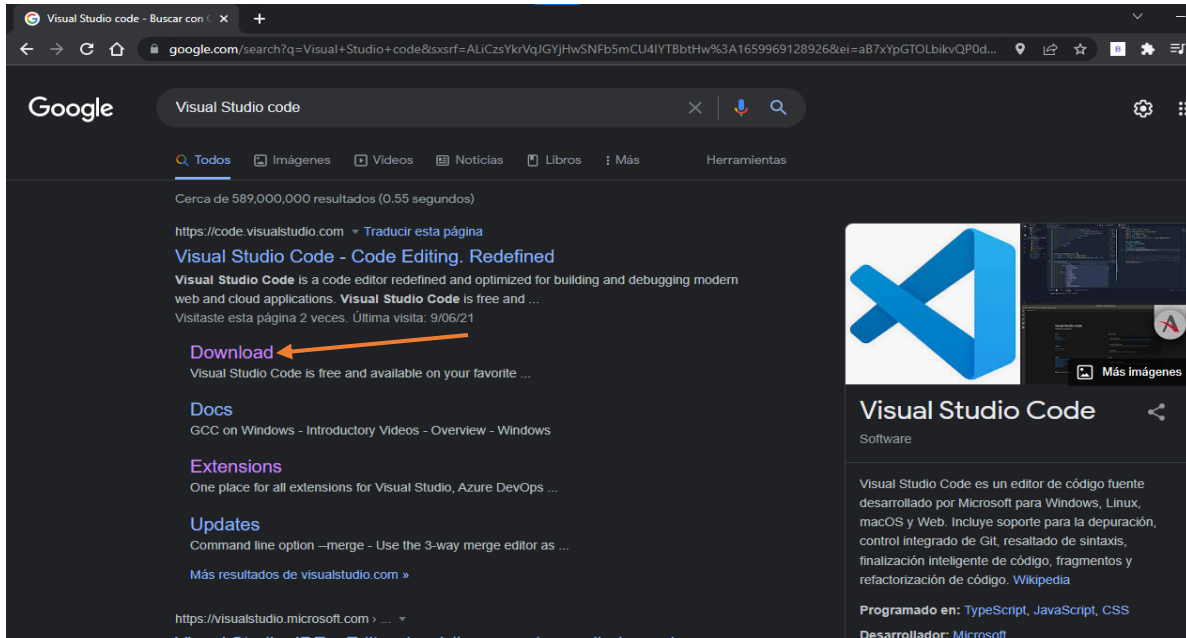
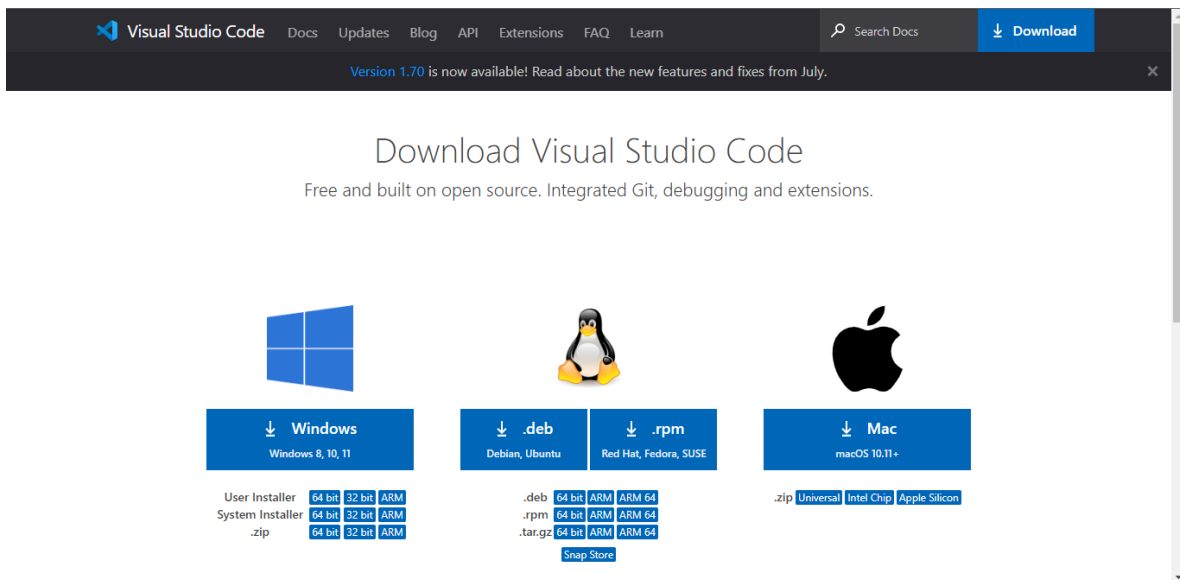


## Proceso instalación Visual Studio Code

**Paso 1:** Escribimos en Google Visual Studio Code y seleccionamos donde dice “Download”.



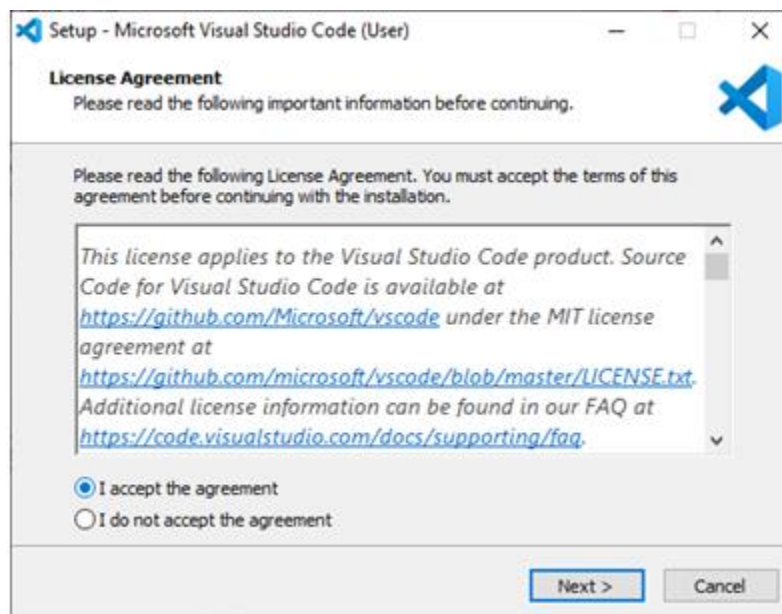
**Paso 2:** Seleccionamos el sistema operativo que tenemos y lo descargamos.



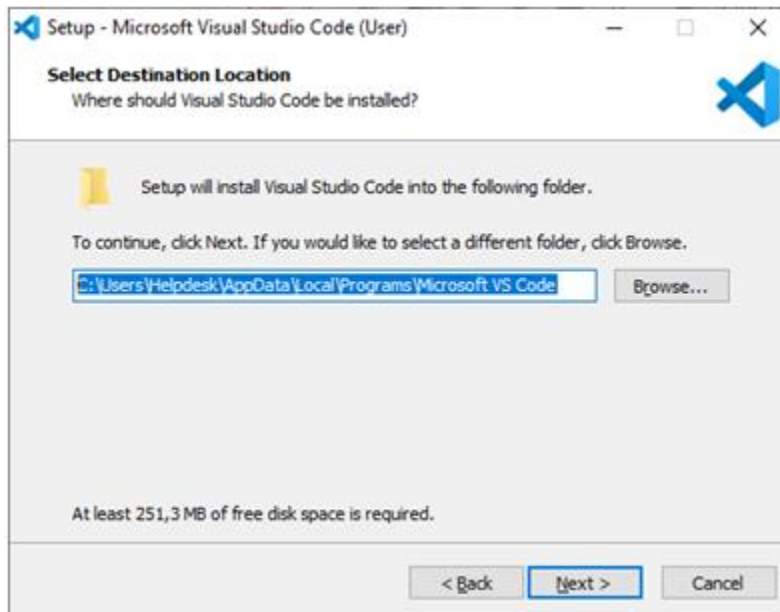
**Paso 3:** Al darle clic nos descargará un .exe, al cual le daremos clic encima.



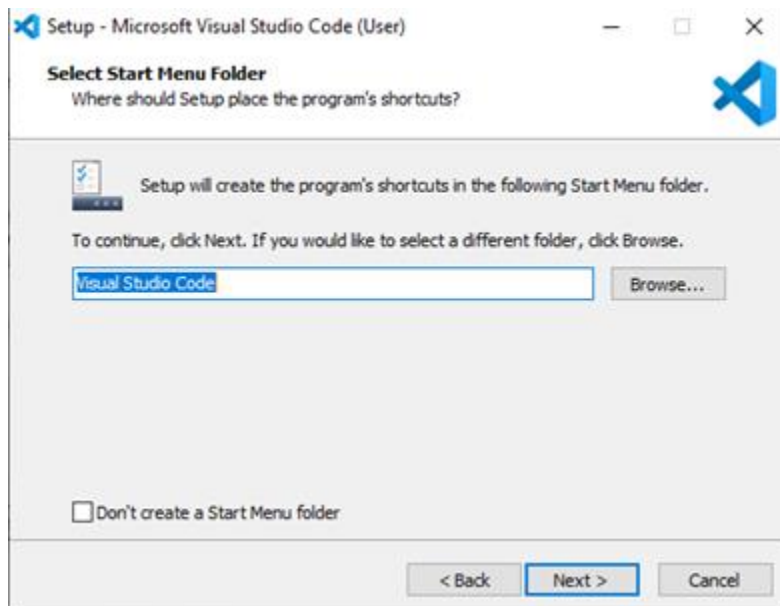
**Paso 4:** Lee y acepta el acuerdo de licencia. Haz clic en Next para continuar.



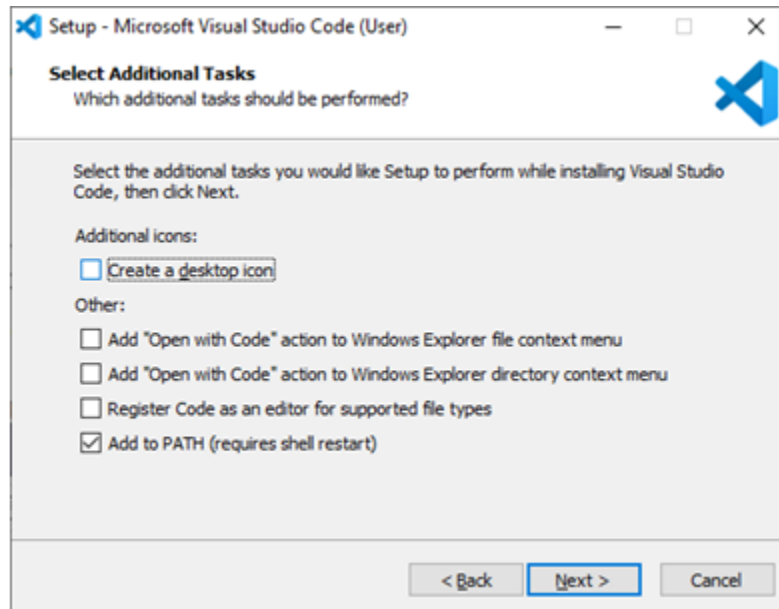
**Paso 5:** Puedes cambiar la ubicación de la carpeta de instalación o mantener la configuración predeterminada. Haz clic en Next para continuar.



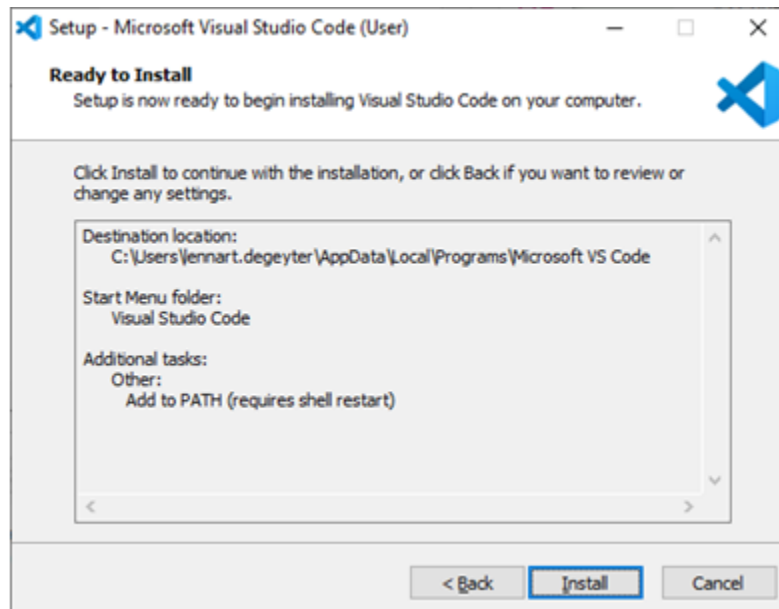
**Paso 6:** Elige si deseas cambiar el nombre de la carpeta de accesos directos en el menú Inicio o si no deseas instalar accesos directos en absoluto. Haz clic en Next.



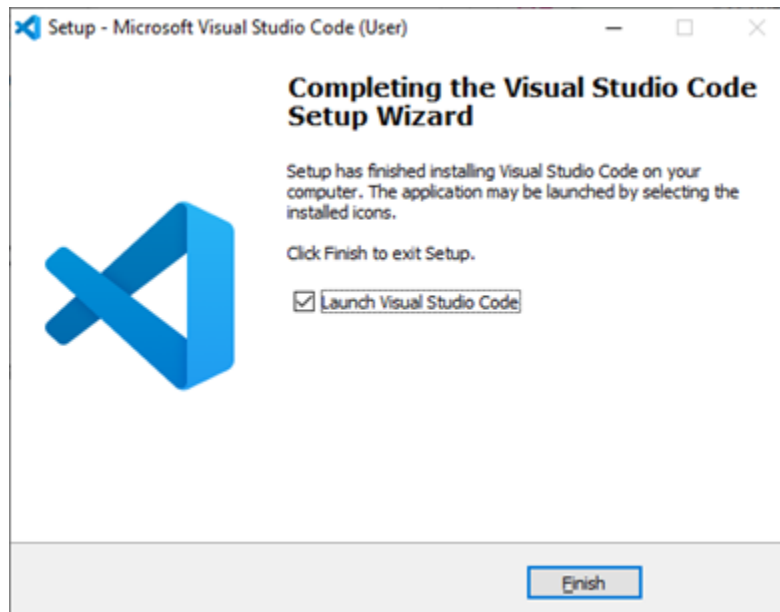
**Paso 7:** Selecciona las tareas adicionales, por ej. crear un icono en el escritorio o añadir opciones al menú contextual de Windows Explorer. Haz clic en Next.



**Paso 8:** Haz clic en Install para iniciar la instalación.

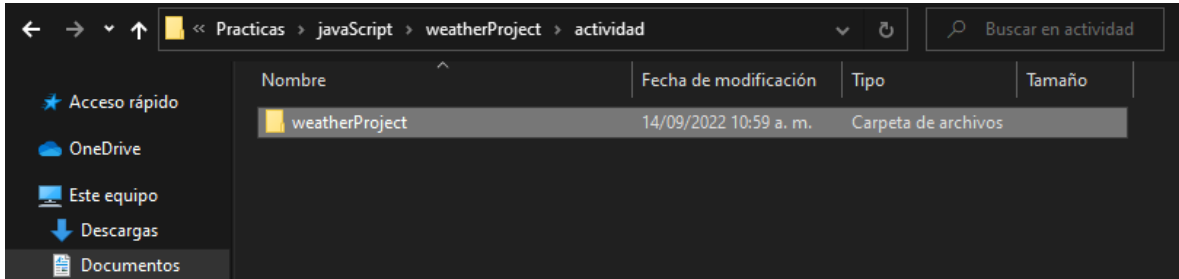


**Paso 9:** El programa está instalado y listo para usar. Haz clic en Finish para finalizar la instalación y lanzar el programa.

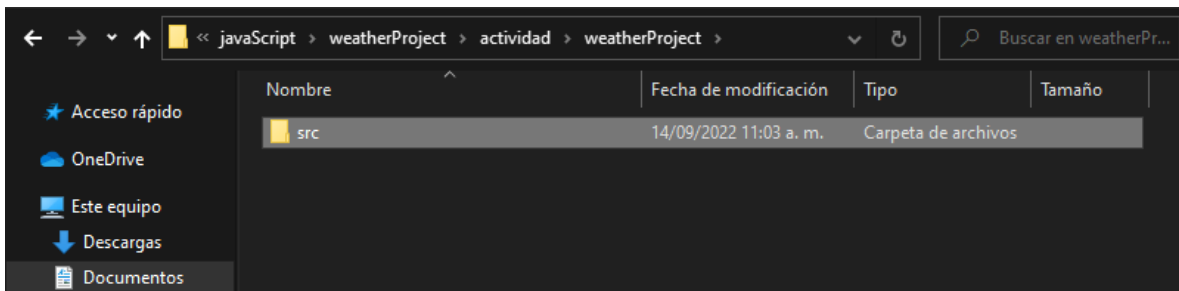


## Creación de archivos y carpetas

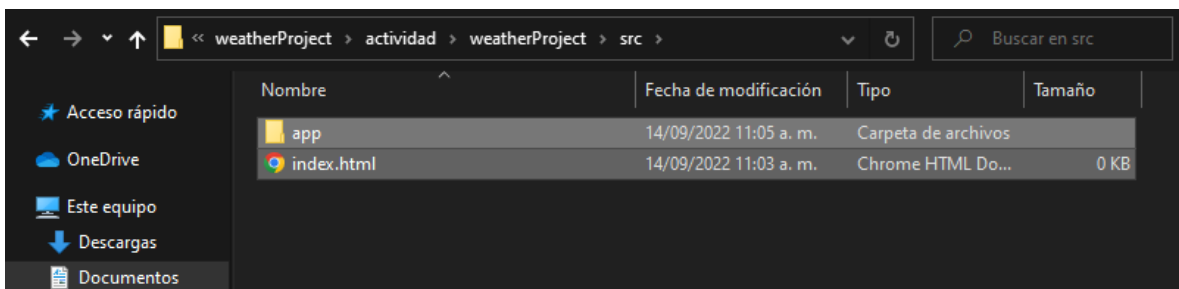
Creamos una carpeta raíz. En nuestro caso la llamaremos “weatherProject”.



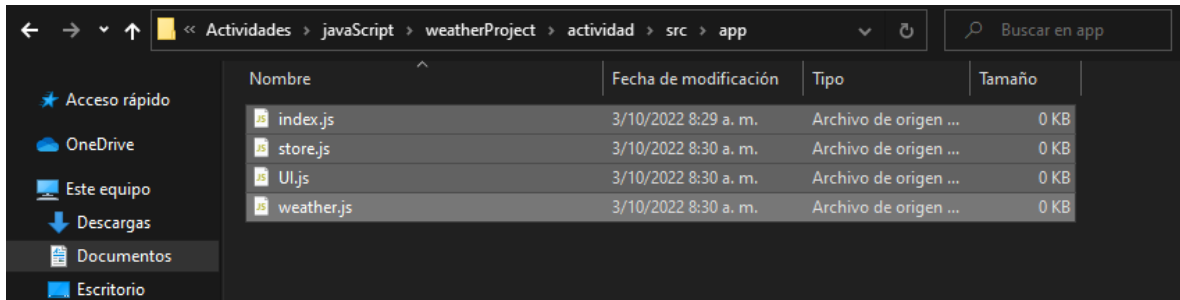
Dentro de esta, crearemos otra carpeta llamada “src”.



Dentro de esta carpeta, crearemos otra carpeta llamada “app”: donde estará toda la lógica del aplicativo y el diseño; y un archivo llamado “index.html”: que es donde estará toda la estructura del aplicativo.

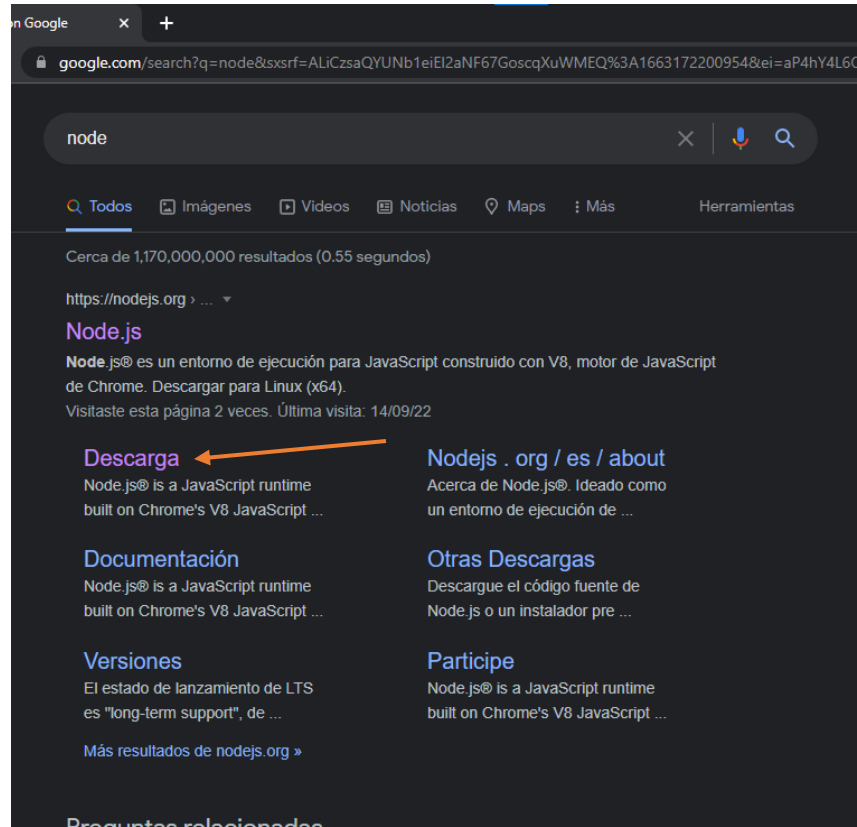


Ahora, entraremos a la carpeta app, allí crearemos algunos archivos: index.js: que será el archivo que arrancará todo el aplicativo, store.js: será para poder guardar los datos en el localStorage, UI.js: donde tendremos la interacción con el dom e interactuar con la vista, weather.js: será quien pedirá los datos del clima a la api (Es importante que tengan exactamente esa extensión).




## Proceso instalación node.js

Entramos a Google y escribimos “node” en el buscador, y seleccionamos donde dice “Descarga”.







Dentro de la página, descargo el instalador del sistema operativo que se esté usando en el momento, en nuestro caso Windows.






[INICIO](#) | [ACERCA](#) | [DESCARGAS](#) | [DOCUMENTACIÓN](#) | [PARTIcipe](#) | [SEGURIDAD](#) | [NOTICIAS](#) | [CERTIFICATION](#)

## Descargas

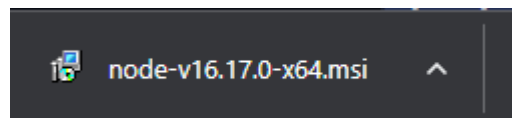
Versión actual: **16.17.0** (includes npm 8.15.0)

Descargue el código fuente de Node.js o un instalador pre-compilado para su plataforma, y comience a desarrollar hoy.

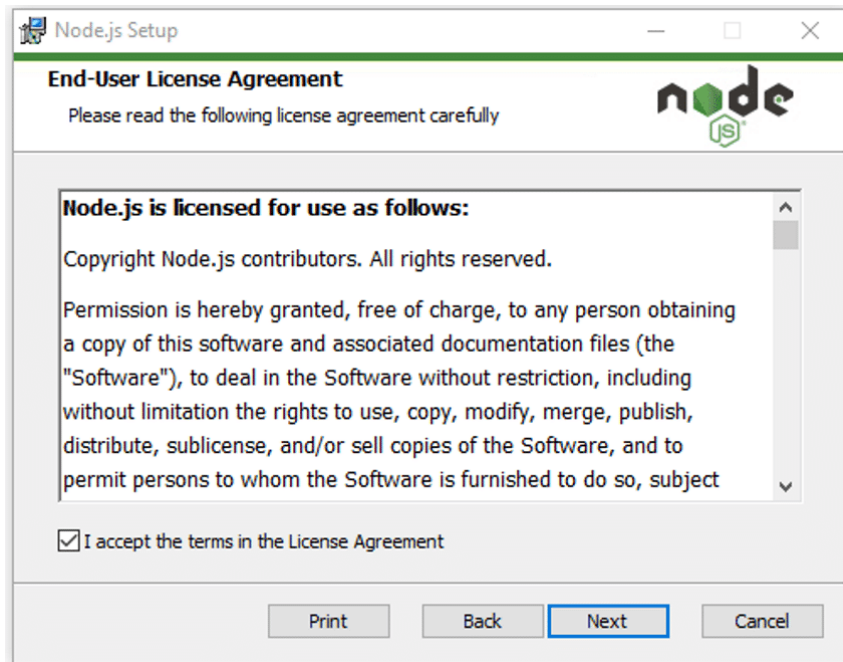
LTS Recomendado para la mayoría	Actual Últimas características	
 <b>Instalador Windows</b> <small>node-v16.17.0-x64.msi</small>	 <b>Instalador macOS</b> <small>node-v16.17.0.pkg</small>	 <b>Código Fuente</b> <small>node-v16.17.0.tar.gz</small>
<b>Instalador Windows (.msi)</b>	32-bit	64-bit
<b>Binario Windows (.zip)</b>	32-bit	64-bit
<b>Linux (.deb)</b>	64-bit / ARM64	

<https://nodesource.com/dist/v16.17.0/node-v16.17.0-x64.msi>

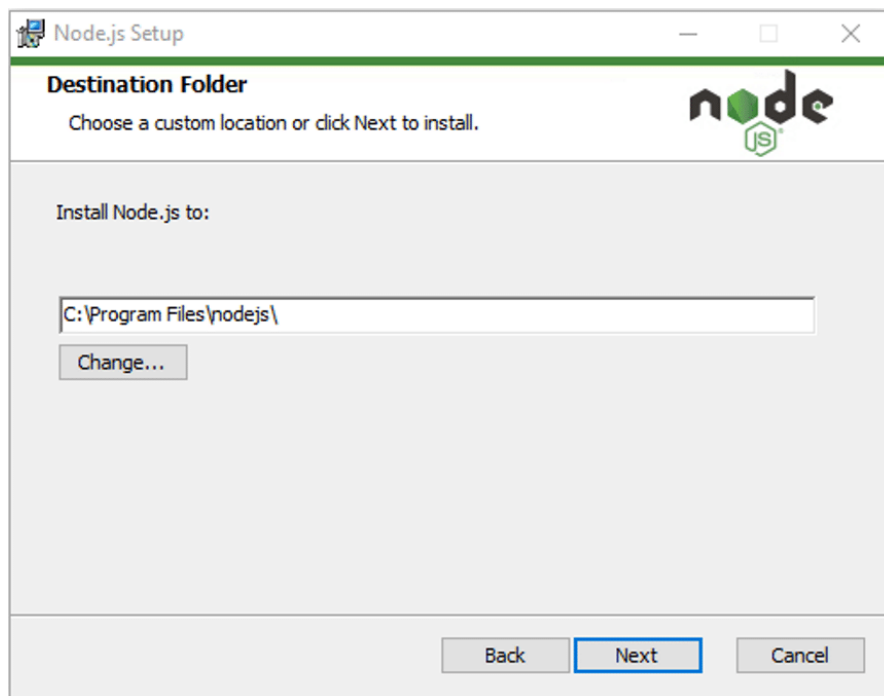
Eso nos descarga el instalador, cuando termine la descarga, le damos clic para ejecutarlo.



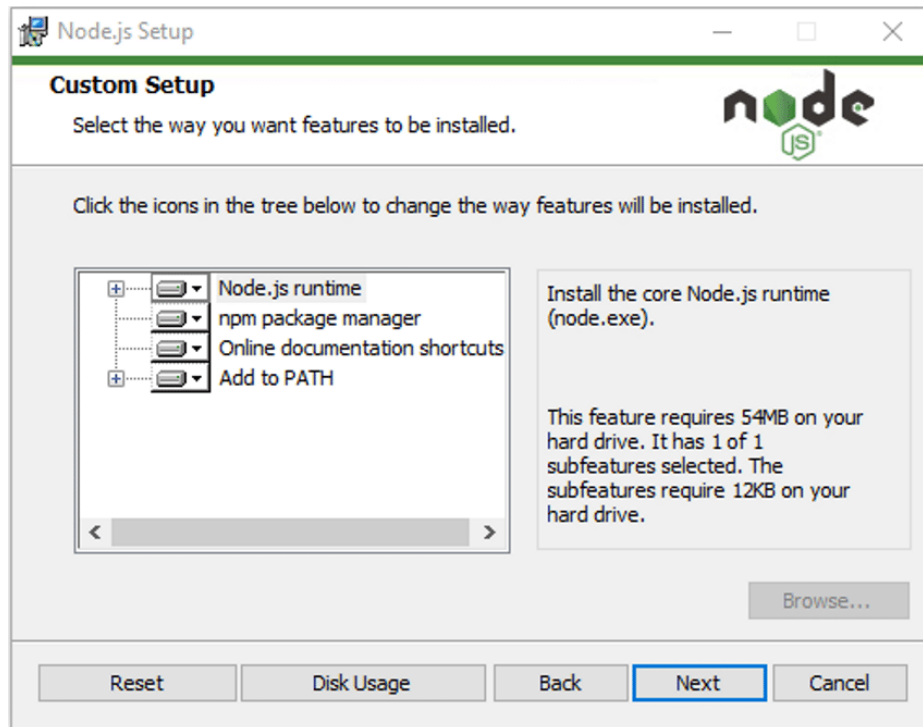
Eso nos abrirá el instalador de node.js, como primer paso tendremos que aceptar los términos.



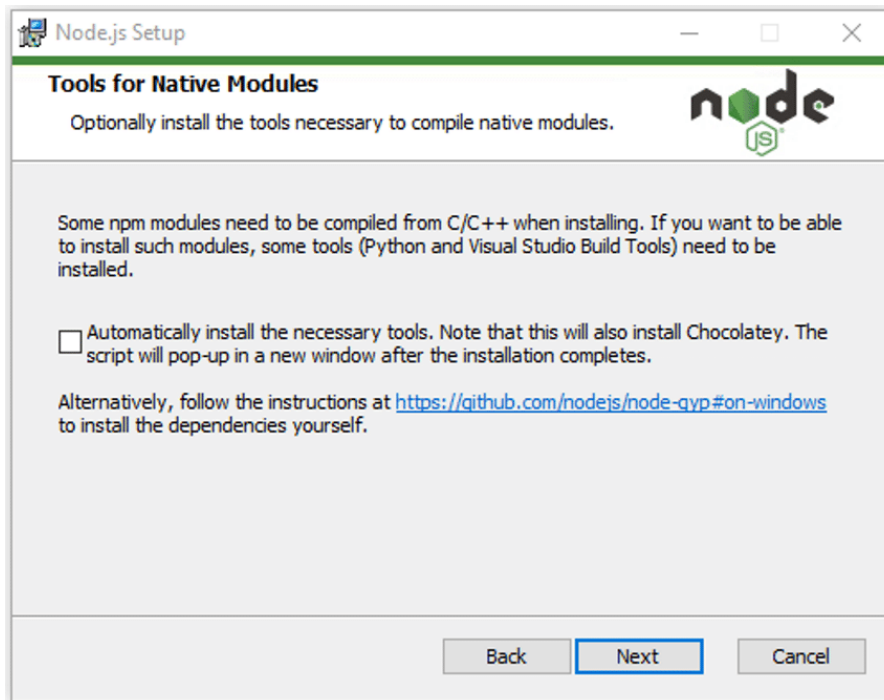
A continuación, selecciona el destino donde quieres instalar Node.js. Si no quieres cambiar el directorio, sigue con la ubicación predeterminada de Windows y vuelve a hacer clic en el botón Siguiente.



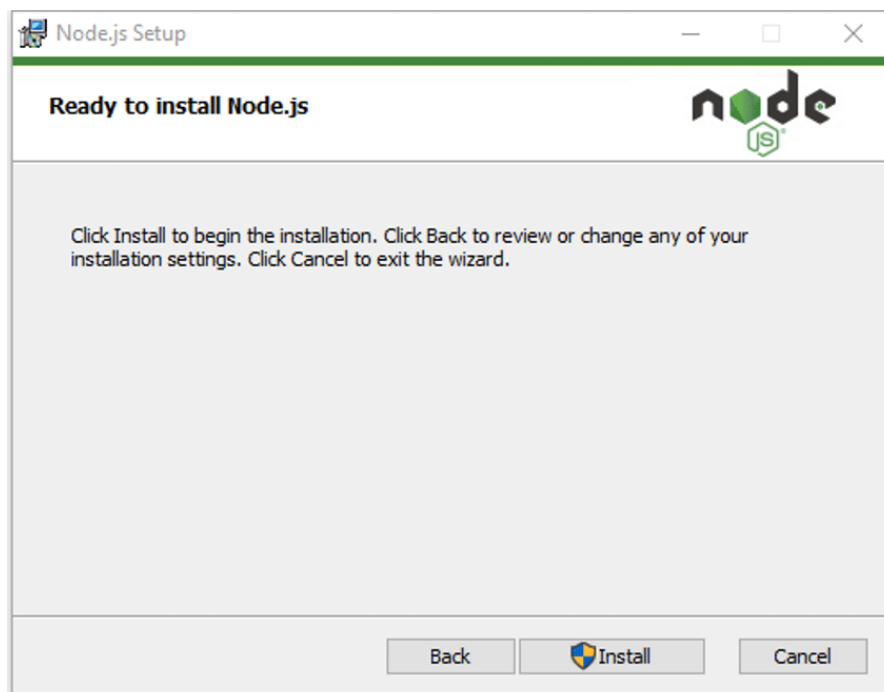
La siguiente pantalla te mostrará las opciones de configuración personalizada. Si quieres una instalación estándar con las características por defecto de Node.js, haz clic en el botón Siguiente. De lo contrario, puedes seleccionar tus elementos específicos en los iconos en el árbol antes de hacer clic en Siguiente:



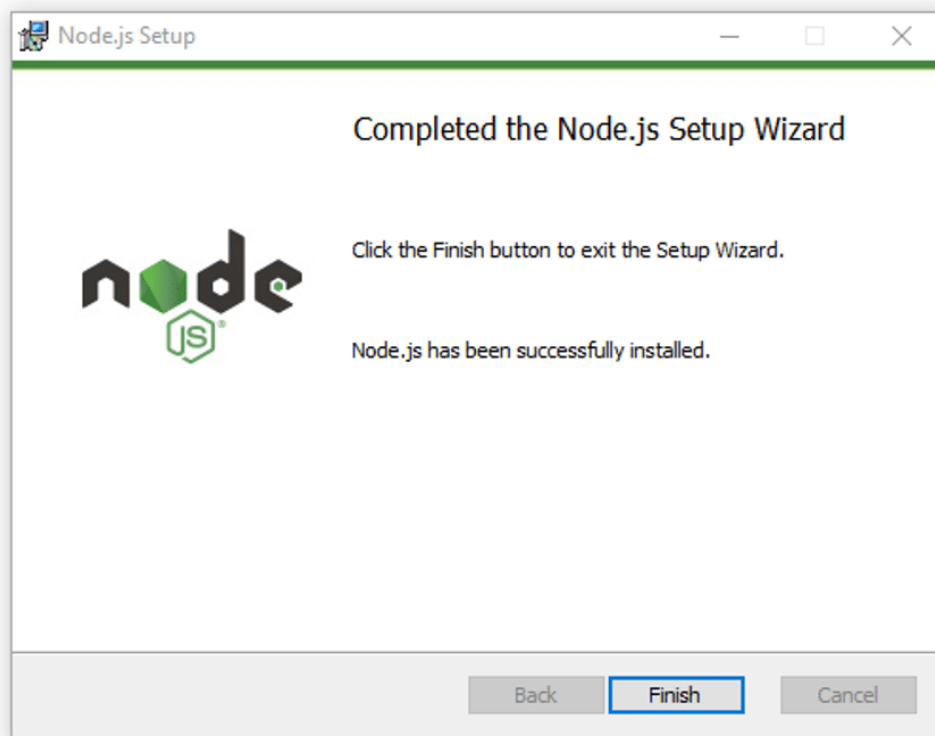
Node.js te ofrece opciones para instalar herramientas para módulos nativos. Si estás interesado en ellas, haz clic en la casilla para marcar tus preferencias, o haz clic en Siguiente para seguir con la opción predeterminada:



Por último, y esta es la parte más fácil de todas, haz clic en el botón Instalar para comenzar el proceso de instalación:



El sistema completará la instalación en unos segundos o minutos y te mostrará un mensaje de éxito. Haz clic en el botón Finalizar para cerrar el instalador de Node.js.

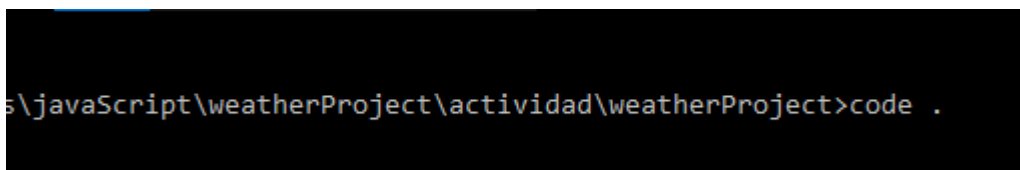


## Weather Project

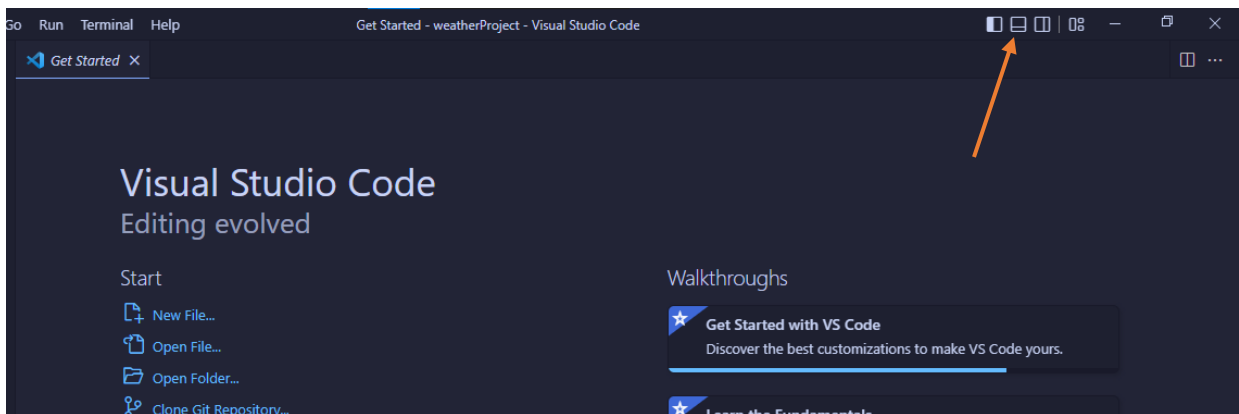
Para abrir el Visual Studio Code, haremos lo siguiente: Dentro de la carpeta raíz, en la barra superior, escribiremos cmd y le damos enter.



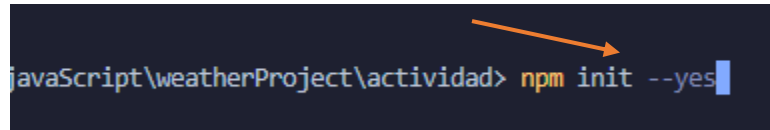
Eso nos abrirá una terminal, solo tendremos que escribir “code .”.



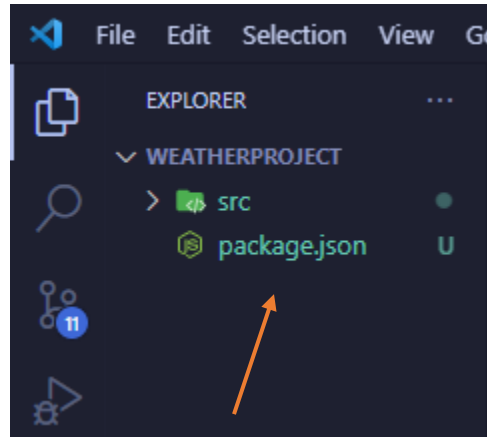
Eso nos abrirá el visual studio code. Ahora, ya que estamos dentro, abriremos la terminal del visual studio code dándole clic en el siguiente icono.



Eso nos abrirá como antes dije, una terminal en la parte baja de nuestro visual studio code, ahí dentro escribiremos el siguiente comando “npm init --yes”, eso nos creará un archivo llamado package.json, luego de haberlo creado, lo abrimos y borramos lo que esté dentro de scripts (línea 7).

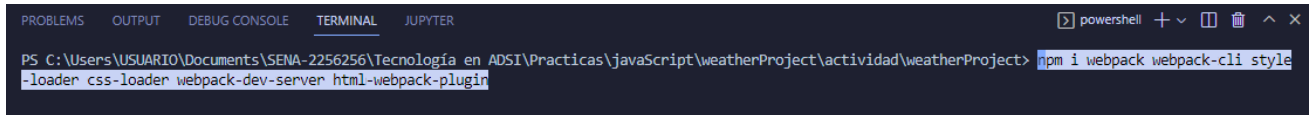


```
JavaScript\weatherProject\actividad> npm init --yes
```



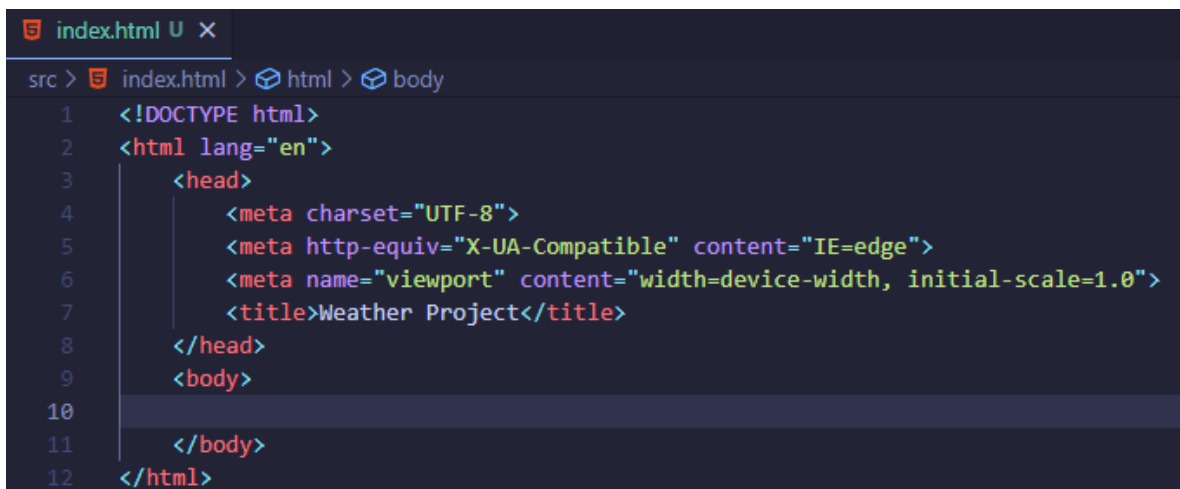
```
package.json U x
package.json > {} scripts
1 {
2   "name": "weatherproject",
3   "version": "1.0.0",
4   "description": "",
5   "main": "index.js",
6   "scripts": {
7     "test": "echo \"Error: no test specified\" && exit 1"
8   },
9   "author": "",
10  "license": "ISC"
11 }
12
```

Luego de haber eliminado esta línea de código, volvemos a la terminal, y escribimos la siguiente lista de comandos: `npm i webpack webpack-cli style-loader css-loader webpack-dev-server html-webpack-plugin`. Esto nos instalará algunas librerías necesarias para la realización del aplicativo.



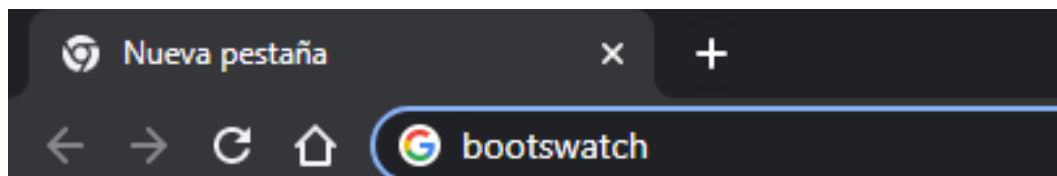
```
PS C:\Users\USUARIO\Documents\SENA-2256256\Tecnología en ADSI\Practicas\javascript\weatherProject\actividad\weatherProject> npm i webpack webpack-cli style-loader css-loader webpack-dev-server html-webpack-plugin
```

Al darle enter, se comenzarán a descargar todas las librerías, mientras se descargan podemos seguir trabajando. Ahora ingresaremos a la carpeta `src` y abriremos el archivo `index.html`, por último, digitamos la estructura del html5 {! enter}



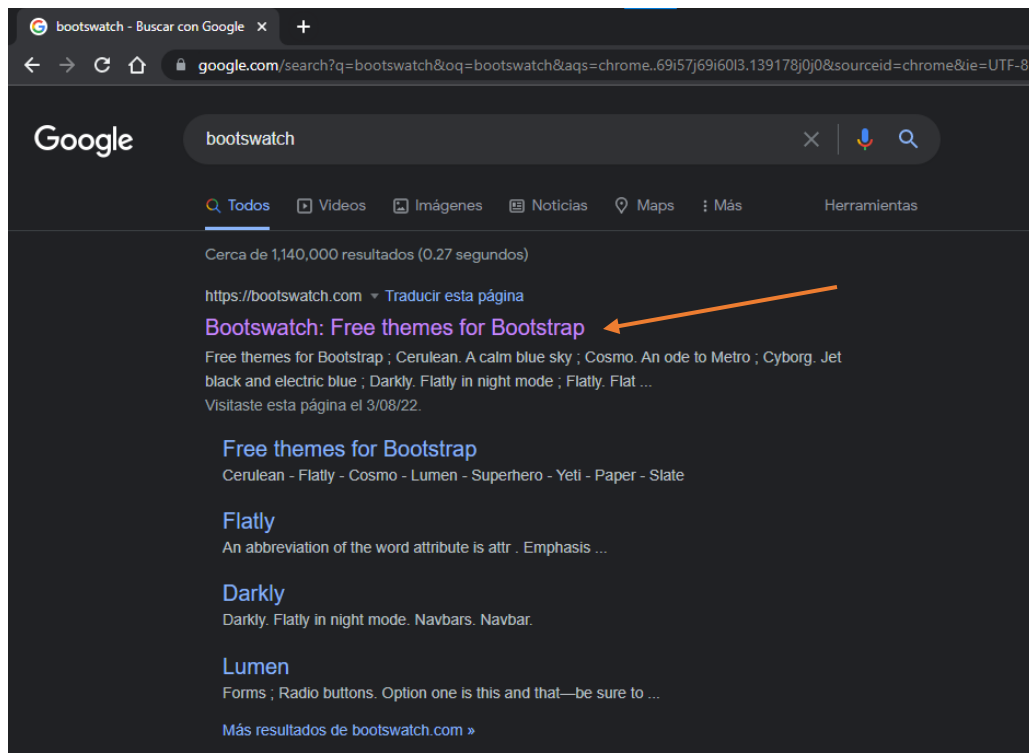
```
src > index.html > html > body
1  <!DOCTYPE html>
2  <html lang="en">
3    <head>
4      <meta charset="UTF-8">
5      <meta http-equiv="X-UA-Compatible" content="IE=edge">
6      <meta name="viewport" content="width=device-width, initial-scale=1.0">
7      <title>Weather Project</title>
8    </head>
9    <body>
10
11  </body>
12 </html>
```

Ahora, iremos a Google y buscaremos “bootswatch” este es una librería que utiliza bootstrap.

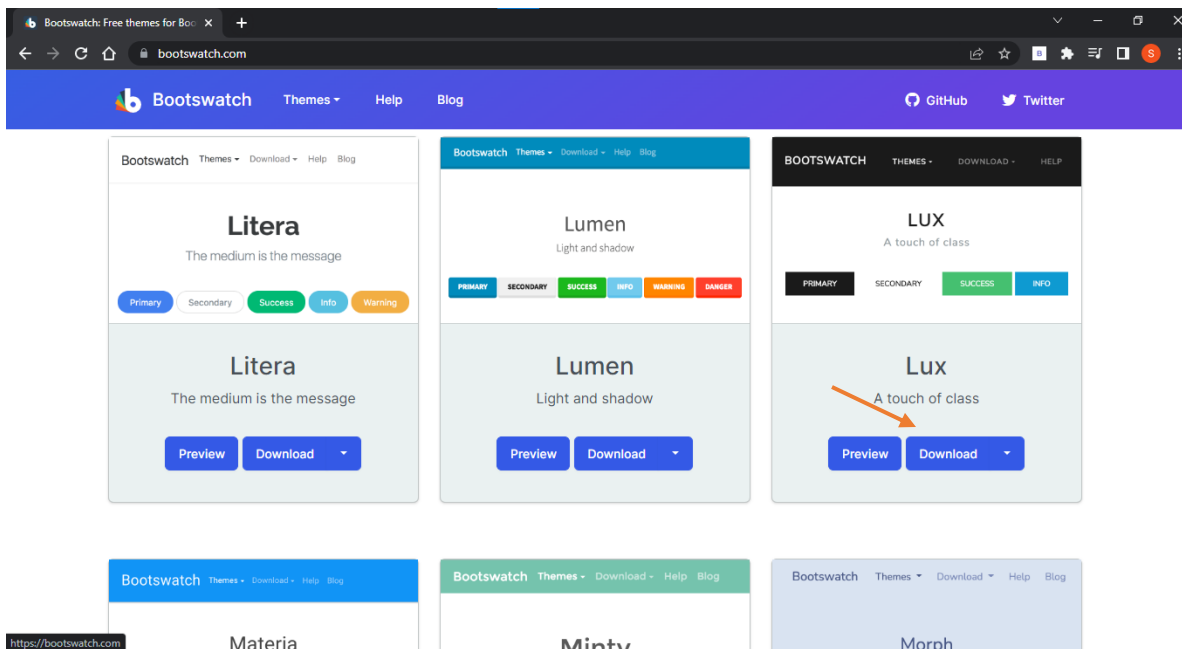




Luego de esto, se selecciona el primer resultado que nos arroja Google.



Entramos, y luego bajamos hasta que nos encontremos uno que se llame “Lux” y le damos en Download.



Esto nos abrirá una página con todo el código css, lo único que necesitaremos será copiar la dirección URL.



Ahora tomamos esa dirección y la enlazamos en el html por medio de la etiqueta link (línea 8).

```
1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <meta charset="UTF-8">
5     <meta http-equiv="X-UA-Compatible" content="IE=edge">
6     <meta name="viewport" content="width=device-width, initial-scale=1.0">
7     <!-- BOOTSTRAP -->
8     <link rel="stylesheet" href="https://bootswatch.com/4/lux/bootstrap.min.css">
9     <title>Weather Project</title>
10  </head>
11  <body>
12
13  </body>
14 </html>
```

Ahora, iremos al body y crearemos tres div, el primero es el contenedor principal (línea 12), donde estará todo del aplicativo, el segundo contenedor es el row (línea 13), este sirve para dividir el contenido, y el último contenedor es para especificar cuánto espacio ocupará en la vista, en este caso, serán 4 columnas y también hacemos que quede centrado todo el contenido (línea 14).

```
11 <body>
12   <div class="container p-4">
13     <div class="row">
14       <div class="col-md-4 mx-auto text-center">
15
16       </div>
17     </div>
18   </div>
19 </body>
20 </html>
```

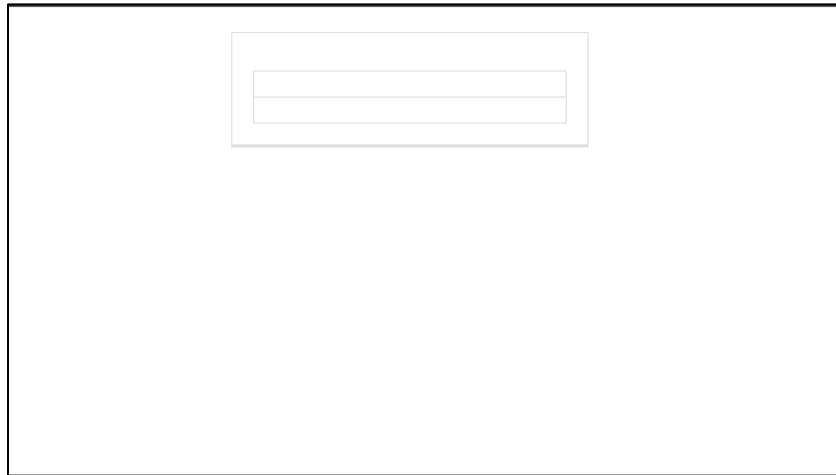
Dentro de estos, crearemos dos tarjetas o “card”, donde en la primera estará toda la información del clima de la ciudad digitada (línea 15), y en la segunda, estará un formulario, ya que ahí es donde el usuario digitará la ciudad y el código del país (línea 18).

```
11 <body>
12   <div class="container p-4">
13     <div class="row">
14       <div class="col-md-4 mx-auto text-center">
15         <div class="card">
16
17
18         </div>
19         <div class="card">
20
21         </div>
22       </div>
23     </div>
24   </body>
```

Dentro del primer card, crearemos un div con la clase “card-body” que será el cuerpo de la tarjeta, dentro de este, crearemos la información que queremos mostrar, en nuestro caso sería de la siguiente manera y orden: localización, descripción del clima, temperatura, humedad y velocidad del viento, estos dos últimos los ponemos en una mini tabla con la etiqueta ul (línea 21).

```
15 <div class="card">
16   <div class="card-body">
17     <!-- Seccion para mostrar informacion -->
18     <h1 id="weather-location" class="h3"></h1>
19     <h3 id="weather-description" class="h4"></h3>
20     <h3 id="weather-string"></h3>
21     <ul class="list-group mt-3">
22       <li id="weather-humidity" class="list-group-item"></li>
23       <li id="weather-wind" class="list-group-item"></li>
24     </ul>
25   </div>
26 </div>
```

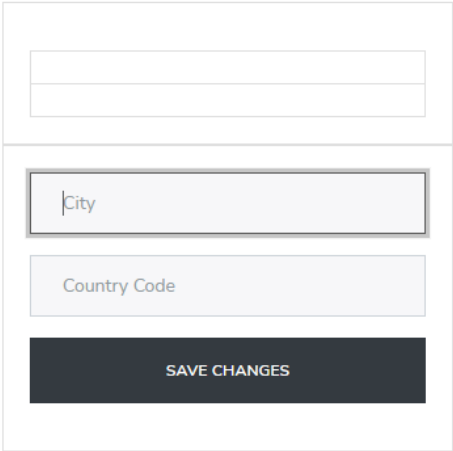
Ahora, podemos abrir el archivo index.html en el navegador, en principio, solo veremos un contenedor grande y uno pequeño dividido en dos, y está vacío, porque no hemos escrito nada dentro de estas etiquetas.



Ahora, vamos al segundo card, donde también crearemos un cuerpo de la tarjeta, dentro de este haremos un formulario, crearemos tres div, en los dos primeros, habrá un input, en el primero, será donde el usuario ingresará la ciudad; y en el segundo, será donde el usuario ingresará el código del país, para terminar, en el último div será donde estará el botón para consultar.

```
27 <div class="card">
28   <div class="card-body">
29     <form id="w-form">
30       <!-- Div - input para ingresar la ciudad -->
31       <div class="form-group">
32         <input type="text" id="city" class="form-control" placeholder="City" autofocus>
33       </div>
34       <!-- Div - input para ingresar el codigo del pais -->
35       <div class="form-group mt-3">
36         <input type="text" id="country-code" class="form-control" placeholder="Country Code">
37       </div>
38       <!-- Div - button para guardar cambios -->
39       <div class="form-group mt-3">
40         <button class="btn btn-dark btn-block" id="w-change-btn">
41           Save Changes
42         </button>
43       </div>
44     </form>
45   </div>
46 </div>
```

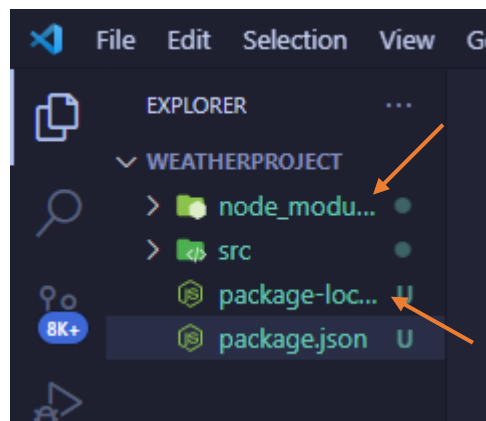
Luego de crear este formulario, también iremos al navegador para ver los cambios realizados.



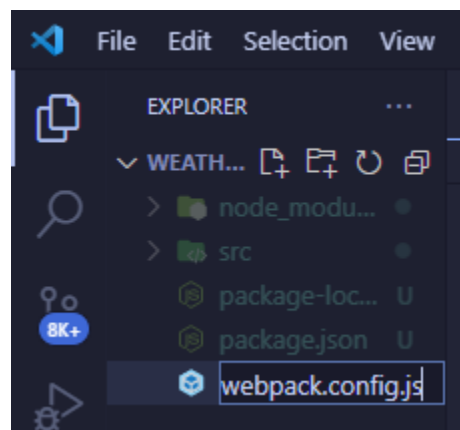
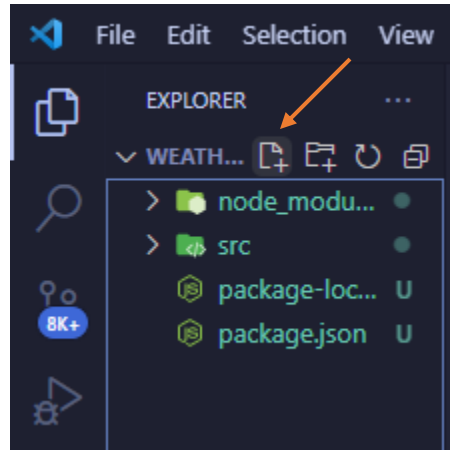
The image shows a web form with the following elements:

- Two empty input fields at the top.
- A text input field labeled "City".
- A text input field labeled "Country Code".
- A dark button labeled "SAVE CHANGES".

Al terminar la codificación de todo el html, ya debió de terminar la instalación de todas las librerías necesarias y se deberían ver de la siguiente manera, primero una carpeta llamada “node\_modules” y un nuevo archivo .json en la carpeta raíz.



Luego de hacer esto, vamos a configurar webpack, que es una herramienta que anteriormente habíamos descargado. Nos devolvemos a la carpeta raíz, y creamos un archivo nuevo llamado “webpack.config.js” de la misma manera que creamos el CSS.



Luego de que hayamos creado este archivo, vamos a configurar el webpack, le damos la configuración de entrada y salida, también configuramos el servidor de desarrollo; para terminar creamos las reglas, para poder leer el css dentro del js y lo mismo hacemos con el html, ya que llamamos una librería para que nos lo lea y le indicamos la ubicación de nuestro archivo html. Hay que explicar algo todo esto, es para crear tanto la carpeta de distribución, como el servidor de desarrollo.

```
webpack.config.js X
webpack.config.js
1  const path = require('path');
2  const HtmlWebpackPlugin = require('html-webpack-plugin');
3
4  module.exports = {
5    entry: './src/app/index.js',
6    output: {
7      path: path.join(__dirname, 'dist'),
8      filename: 'bundle.js'
9    },
10   devServer: {
11     port: 3000
12   },
13   module: {
14     rules: [
15       {
16         test: /\.css$/,
17         use: ['style-loader', 'css-loader']
18       }
19     ]
20   },
21   plugins: [
22     new HtmlWebpackPlugin({
23       template: './src/index.html'
24     })
25   ],
26 };
```

Ahora, vamos al archivo package.json y modificamos dentro de scripts (línea 6); el build es la instrucción que nos crea la carpeta dist, y el dev, es la instrucción que nos crea el servidor de desarrollo.

```
1  {
2    "name": "weatherproject",
3    "version": "1.0.0",
4    "description": "",
5    "main": "index.js",
6    "scripts": {
7      "build": "webpack --mode production",
8      "dev": "webpack server --mode development"
9    },
10   "author": "",
11   "license": "ISC",
12   "dependencies": {
13     "css-loader": "^6.7.1",
14     "html-webpack-plugin": "^5.5.0",
15     "style-loader": "^3.3.1",
16     "webpack": "^5.74.0",
17     "webpack-cli": "^4.10.0",
18     "webpack-dev-server": "^4.11.0"
19   }
20 }
```

Ahora, tendremos que abrir de nuevo la terminal dentro del visual studio code, así que le daremos al mismo ícono que en el paso 3. Ahora, tendremos que digitar la siguiente instrucción en la terminal: "npm run dev", esto nos inicializará el servidor de desarrollo.

```
actividad\weatherProject> npm run dev
```

Tendremos que esperar algunos segundos mientras se inicializa, cuando haya terminado, va a salir un mensaje parecido al siguiente:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  JUPYTER

./node_modules/webpack/hot/log.js 1.34 KiB [built] [code generated]
+ 2 modules
modules by path ./node_modules/html-entities/lib/*.js 81.3 KiB
./node_modules/html-entities/lib/index.js 7.74 KiB [built] [code generated]
./node_modules/html-entities/lib/named-references.js 72.7 KiB [built] [code generated]
+ 2 modules
./node_modules/ansi-html-community/index.js 4.16 KiB [built] [code generated]
./node_modules/events/events.js 14.5 KiB [built] [code generated]
./src/app/index.js 1.46 KiB [built] [code generated]
webpack 5.74.0 compiled successfully in 4919 ms
```

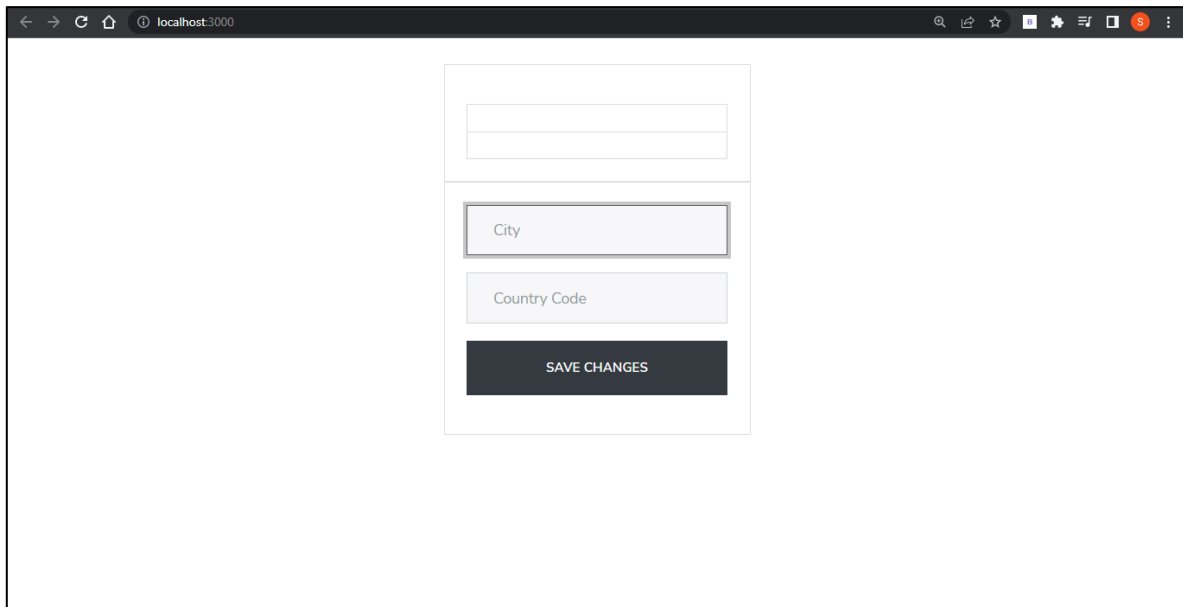


Y podremos ver en donde está ubicado el servidor, si vamos hasta la parte superior del mensaje hasta donde diga algo parecido a lo siguiente, para abrirlo, dejamos la tecla control presionada, y le damos clic:

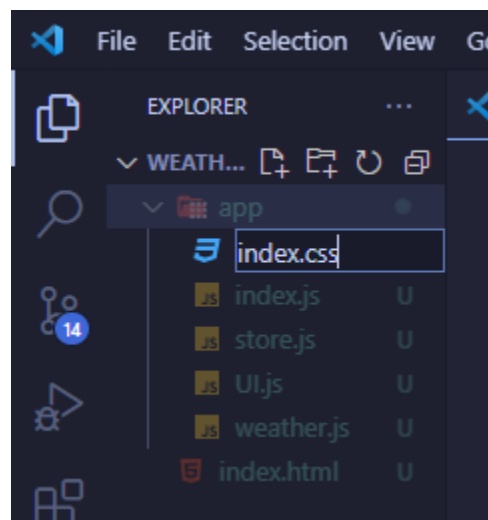
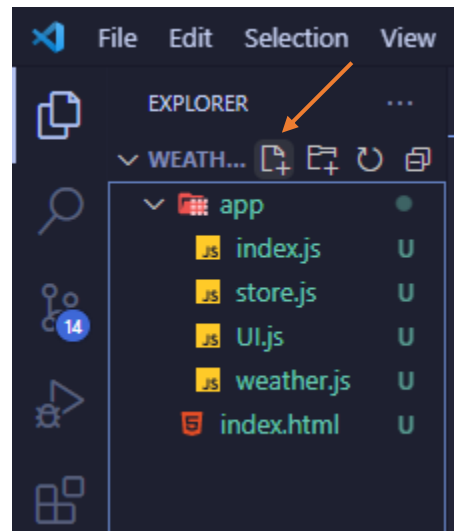
```
PS C:\Users\USUARIO\Documents\SENA-2256256\Tecnología en ADSI\Practicas\Actividades\javascript\weatherProject\actividad\weatherProject> npm run dev
> weatherproject@1.0.0 dev
> webpack server --mode development

<i> [webpack-dev-server] Project is running at:
<i> [webpack-dev-server] Loopback: http://localhost:3000/
<i> [webpack-dev-server] On Your Network (IPv4): http://192.168.20.21:3000/
<i> [webpack-dev-server] On Your Network (IPv6): http://\[fe80::ccf:9c31:911e:35f1\]:3000/
<i> [webpack-dev-server] Content not from webpack is served from 'C:\Users\USUARIO\Documents\SENA-2256256\Tecnología en ADSI\Practicas\Actividades\ja
weatherProject\actividad\weatherProject\public' directory
```

Ahora tenemos el servidor de desarrollo, este nos sirve para que todo lo que hagamos se actualice en tiempo real, ya que se depura y se ejecuta luego de que hagamos cambios en el aplicativo.



Al terminar todo esto, iremos dentro de la carpeta app, y crearemos un archivo más, llamado index.css. Lo que haremos es darle estando dentro de app, clic en el siguiente ícono y lo llamaremos index.css.



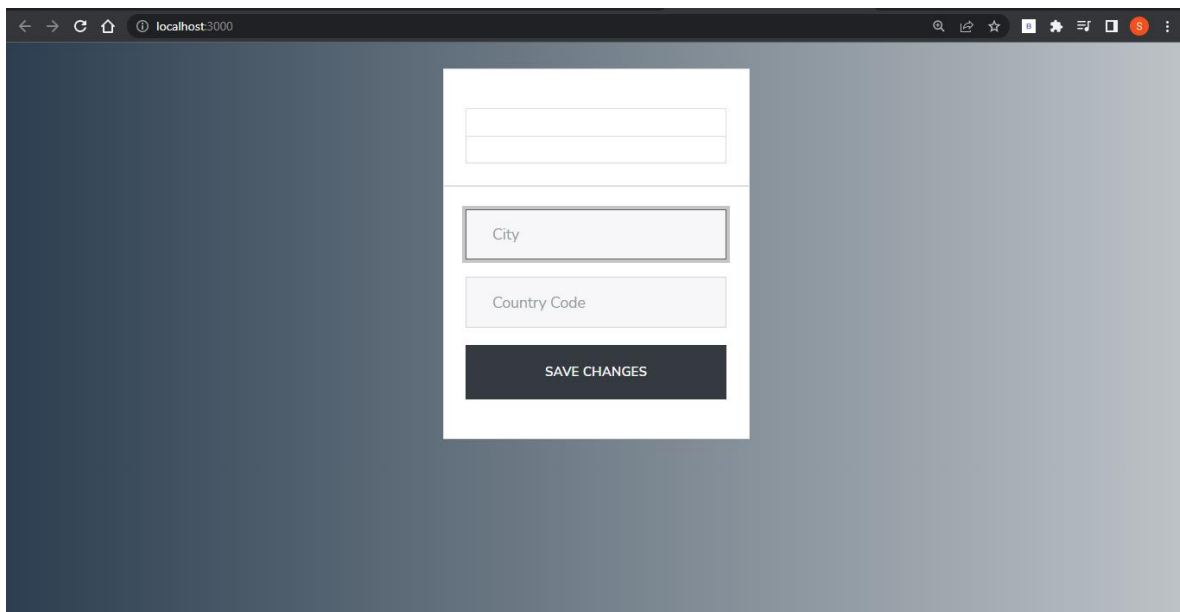
Ahora, vamos a este archivo para darle estilos; dentro de este, llamamos la etiqueta `body`, y le ponemos en este caso un color de fondo gradiente.

```
index.css U X
src > app > index.css > body
1  /* Estilo para el fondo del aplicativo */
2  body {
3      background: #bdc3c7;
4      background: -webkit-linear-gradient(to right, #2c3e50, #bdc3c7);
5      background: linear-gradient(to right, #2c3e50, #bdc3c7);
6  }
7  }
```

Antes hemos creado el css, pero ahora vemos que no lo está teniendo en cuenta, ¿por qué? Porque tendremos que ir al archivo `index.js` dentro de la carpeta `app`, y llamar el css.

```
1  // Se importan los estilos
2  require('./index.css');
```

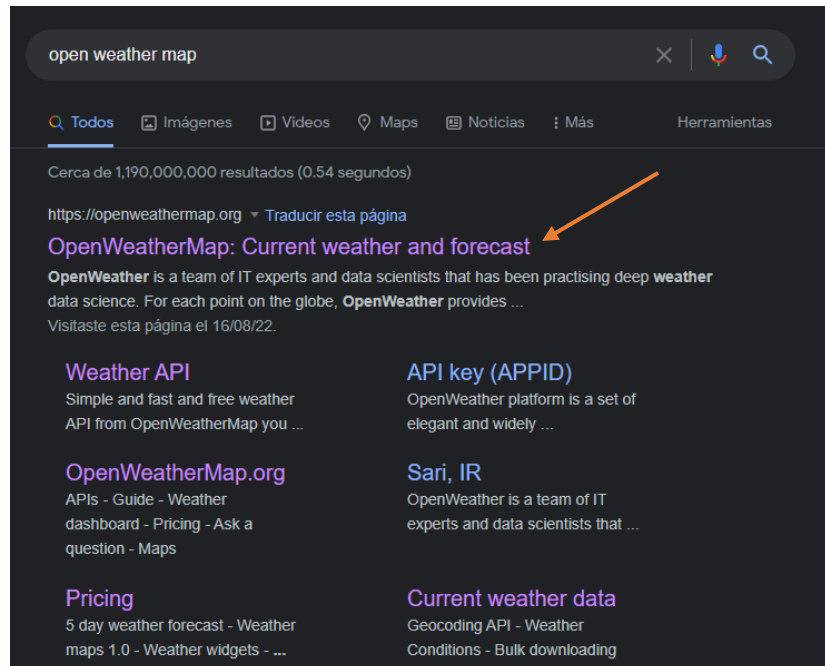
Guardamos y vamos al navegador para ver los cambios, si recordamos, solo le modificamos el color al `body` y esto es lo que nos muestra.



Ahora, vamos de nuevo al archivo index.js y creamos una función llamada fetchWeather, luego, creamos una escucha del evento del documento, el cual deberá ser que se recargue el dom, y ejecutamos la función que acabamos de crear.

```
4 // Se crea la funcion asincrona fetchWeather
5 function fetchWeather () {
6
7 }
8
9 // Se escucha el evento y se corre la funcion
10 document.addEventListener('DOMContentLoaded', fetchWeather);
```

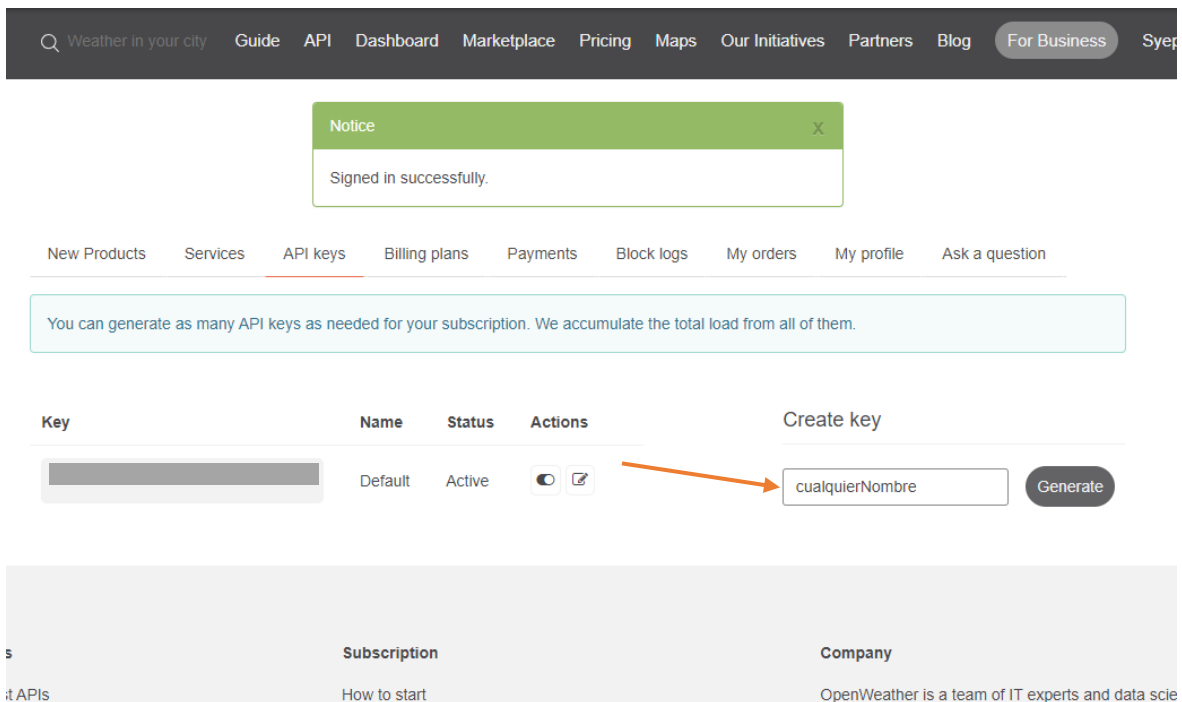
Ahora, ingresaremos a la página web oficial de la api que consumiremos, esta se llama OpenWeatherMap, escribimos esto en Google y seleccionamos el primer resultado.




Luego de que entremos a la página, nos tendremos que registrar e iniciar sesión, ya que, para poder utilizar la api, necesitamos una llave que nos genera esta página. Luego de haberse registrado o iniciado sesión, pulsaremos donde sale nuestro nombre, y se abrirá un menú desplegable, ahí pulsaremos la opción que dice My API keys.



Esto nos abrirá una sección donde deberemos escribir el nombre de la api key y posteriormente le damos clic a “Generate”.



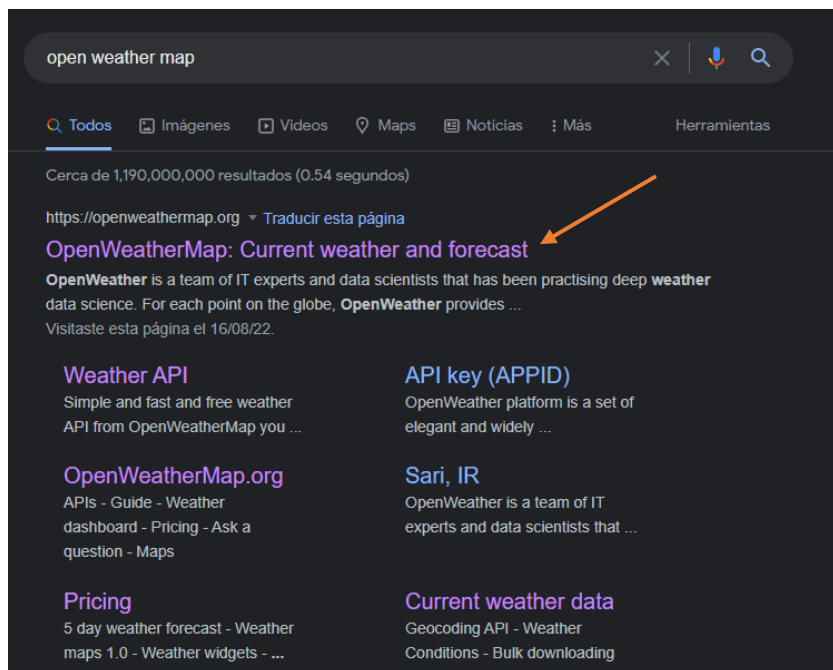
Esto nos generará la api key que necesitaremos en nuestro aplicativo, así que la vamos a copiar en el portapapeles para más adelante utilizarla.

Key	Name	Status	Actions
743520de48ea9fa036be06cde99f81e4	cualquierNombre	Active	  

Ahora vamos al archivo weather.js y creamos una clase que será exportada, esta se llamará Weather. Dentro de esta, crearemos un método constructor, que es el método que se ejecuta, donde vamos a definir la apiKey, aquí pegamos la key que en el paso anterior habíamos copiado (línea 7), la ciudad (línea 8) y el código del país (línea 9).

```
1 // Exportamos la clase para poder importarla en el index
2 export class Weather {
3
4     // Metodo constructor (Se ejecuta solo)
5     constructor(city, countryCode)
6     {
7         this.apikey = '3cbf896a0e1f91f5d85ab195e7681dee';
8         this.city = city;
9         this.countryCode = countryCode;
10    }
```

Ahora, ingresaremos de nuevo a la página web oficial de la api que consumiremos, esta se llama OpenWeatherMap, escribimos esto en Google y seleccionamos el primer resultado.



Ahora le tendremos que dar donde dice API en la parte superior.



Luego, bajaremos hasta donde dice Current Weather Data, y le damos en el botón API Doc.

You can read the [How to Start](#) guide and enjoy using our powerful weather APIs right now.

## Current & Forecast weather data collection

### Current Weather Data

[API doc](#) [Subscribe](#)

- Access current weather data for any location including over 200,000 cities
- We collect and process weather data from different sources such as global and local weather models, satellites, radars and a vast network of weather stations
- JSON, XML, and HTML formats
- Included in both free and paid subscriptions

### Hourly Forecast 4 days

[API doc](#) [Subscribe](#)

- Hourly forecast is available for 4 days
- Forecast weather data for 96 timestamps
- JSON and XML formats
- Included in the Developer, Professional and Enterprise subscription plans

### Daily Forecast 16 days

[API doc](#) [Subscribe](#)

- 16 days forecast is available for any location on the globe
- 1-day step for 16 days
- JSON and XML formats
- Included in all paid subscription plans

### Climatic Forecast 30 days

[API doc](#) [Subscribe](#)

- Forecast weather data for 30 days
- JSON format

### Bulk Download

[API doc](#) [Subscribe](#)

- You may request current weather and forecasts in bulk with a variable data

### Global Weather Alerts Push notifications

[Doc](#) [Get access](#)

- Get all the warnings from national

Esto nos enviará a toda la documentación de la api, lo único que tendremos que realizar aquí es en la parte derecha de la pantalla, darle donde dice “Built-in API request by city name”.

## Current weather data

[Home](#) / [API](#) / [Current weather](#)

Access current weather data for any location on Earth including over 200,000 cities! We collect and process weather data from different sources such as global and local weather models, satellites, radars and a vast network of weather stations. Data is available in JSON, XML, or HTML format.

### Call current weather data

#### How to make an API call

**API call**

```
https://api.openweathermap.org/data/2.5/weather?lat={lat}&lon={lon}&appid={API key}
```

**Parameters**

Parameter	Required	Description
lat, lon	required	Geographical coordinates (latitude, longitude). If you need the geocoder to automatic convert city names and zip-codes to geo coordinates and the other way around, please use our <a href="#">Geocoding</a>

- Call current weather data
  - How to make an API call
- Bulk downloading
- Weather fields in API response
  - JSON
  - XML
  - List of condition codes
  - Min/max temperature in current weather
  - API and forecast API
- Other features
  - Geocoding API
  - Built-in geocoding
    - [Built-in API request by city name](#)
    - Built-in API request by city ID
    - Built-in API request by ZIP code
  - Format
  - Units of measurement
  - Multilingual support
  - Call back function for JavaScript code



Al darle clic allí, nos llevará a la documentación de cómo llamar la api, allí tomaremos la segunda opción, ya que tiene todo lo que realmente necesitaremos que es el nombre del país, luego el código del país y por último, tendremos que poner allí la key que nos dio la api. Copiaremos esta segunda opción.

OpenWeather

Weather in your city Guide API Dashboard Marketplace Pricing Maps Our Initiatives Partners Blog For Business Sign in Support

Built-in API request by city name

You can call by city name or city name, state code and country code. Please note that searching by states available only for the USA locations.

API call

`https://api.openweathermap.org/data/2.5/weather?q={city name}&appid={API key}`

`https://api.openweathermap.org/data/2.5/weather?q={city name},{country_code}&appid={API key}`

`https://api.openweathermap.org/data/2.5/weather?q={city name},{state_code},{country_code}&appid={API key}`

Parameters

q required City name, state code and country code divided by comma, Please, refer to ISO 3166 for the state codes or country codes. You can specify the parameter not only in English. In this case, the API response should be returned in the same language as the language of requested location name if the location is in our predefined list of more than 200,000 locations.

Call current weather data  
How to make an API call  
Bulk downloading  
Weather fields in API response  
JSON  
XML  
List of condition codes  
Min/max temperature in current weather  
API and forecast API  
Other features  
Geocoding API  
Built-in geocoding  
Built-in API request by city name  
Built-in API request by city ID  
Built-in API request by ZIP code  
Format  
Units of measurement  
Multilingual support  
Call back function for JavaScript code

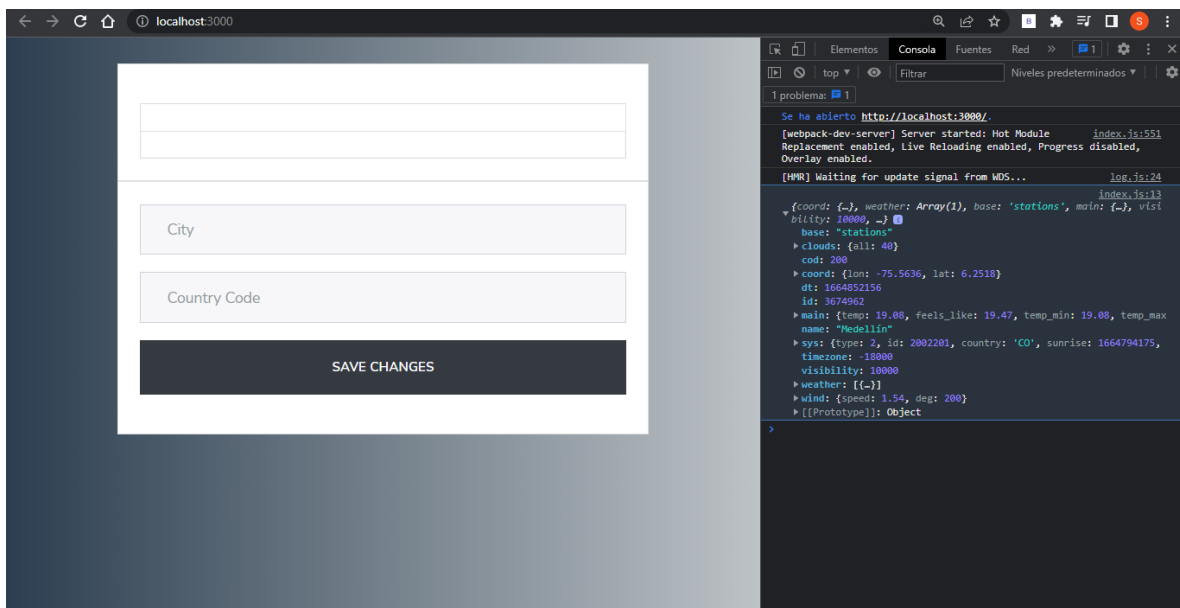
Ya tenemos el link para poder hacer uso de la api, ahora vamos a codificar el método desde donde la llamaremos. Entonces, crearemos un método asíncrono, donde en principio crearemos una constante llamada URI, donde irá la url que copiamos en el paso anterior, es importante aclarar que este debe ir dentro de backticks (`), ya que dentro de la url tendremos que llamar la ciudad, el código del país y por último la key, la última parte de la url, sirve para que los grados, en el momento de llamar la api, nos los dé en grados Celsius. Luego creamos una petición mediante el fetch y esto también con await, para especificar que la petición puede tardar en realizarse, para terminar con este método, simplemente convertimos esta petición de objeto a json, para poder tomar los datos después, y retornamos los datos.

```
12 // Metodo asincrono y dentro un await (Que la peticion puede tardar)
13 async getWeather()
14 {
15     // URL de la api
16     const URI = `https://api.openweathermap.org/data/2.5/weather?q=${this.city},${this.countryCode}&appid=${this.apikey}&units=metric`;
17     // Se hace la peticion con fetch y await es para especificar que puede tardar
18     const response = await fetch(URI);
19
20     const data = await response.json();
21     return data;
22 }
```

Ahora, vamos al archivo index.js y traeremos la clase que hemos acabado de realizar, llamado Weather y luego indicamos la ubicación del archivo; luego, creamos una nueva instancia de la clase Weather y le enviamos una ciudad de ejemplo y un código de país también de ejemplo. Ahora, dentro de la función fetchWeather vamos a llamar el método getWeather y lo almacenamos en una constante llamada data, ahora le tendremos que poner async a la función y await cuando llamamos el método getWeather, para terminar imprimimos por medio de la consola todos los datos.

```
4  Importamos la clase Weather desde el archivo weather
5  const { Weather } = require('./weather');
6
7  // Se crea una nueva instancia Weather
8  const weather = new Weather('Medellin', 'CO');
9
10 // Se crea la funcion asincrona fetchWeather
11 async function fetchWeather () {
12   ...const data = await weather.getWeather();
13   ...console.log(data);
14 }
```

Podremos ver en el navegador los cambios que hemos hecho, primero le tenemos que dar a la tecla f12 para abrir las herramientas de desarrollador en el navegador y tendremos que entrar donde dice consola, allí estarán todos los datos del clima de esa ciudad.



Ahora, vamos al archivo UI.js, este nos servirá para manipular el dom, o sea, llamaremos por su id a los elementos que queramos cambiarle el valor y mostrarlo por pantalla. Empezamos creando la clase UI y exportándola, para poder utilizarla en otros módulos, luego, haremos el método constructor dentro de esta y llamamos por medio de su id, todos los elementos que habíamos definido en el html para la información del clima.

```
1 // Exportamos la clase para poder importarla en el index
2 export class UI {
3
4     // Metodo constructor (Se ejecuta solo)
5     constructor()
6     {
7         // Se traen todos los elementos necesarios
8         this.location = document.getElementById('weather-location');
9         this.desc = document.getElementById('weather-description');
10        this.string = document.getElementById('weather-string');
11        this.humidity = document.getElementById('weather-humidity');
12        this.wind = document.getElementById('weather-wind');
13    }
14 }
```

También, en este archivo (UI.js), crearemos otro método llamado render, que nos servirá para renderizar la vista, y le ponemos que requiera la información del clima para poder ejecutarse.

```
14
15     // Metodo para mostrar en la vista los datos
16     render(weather)
17     {
18
19     }
20
21 }
```

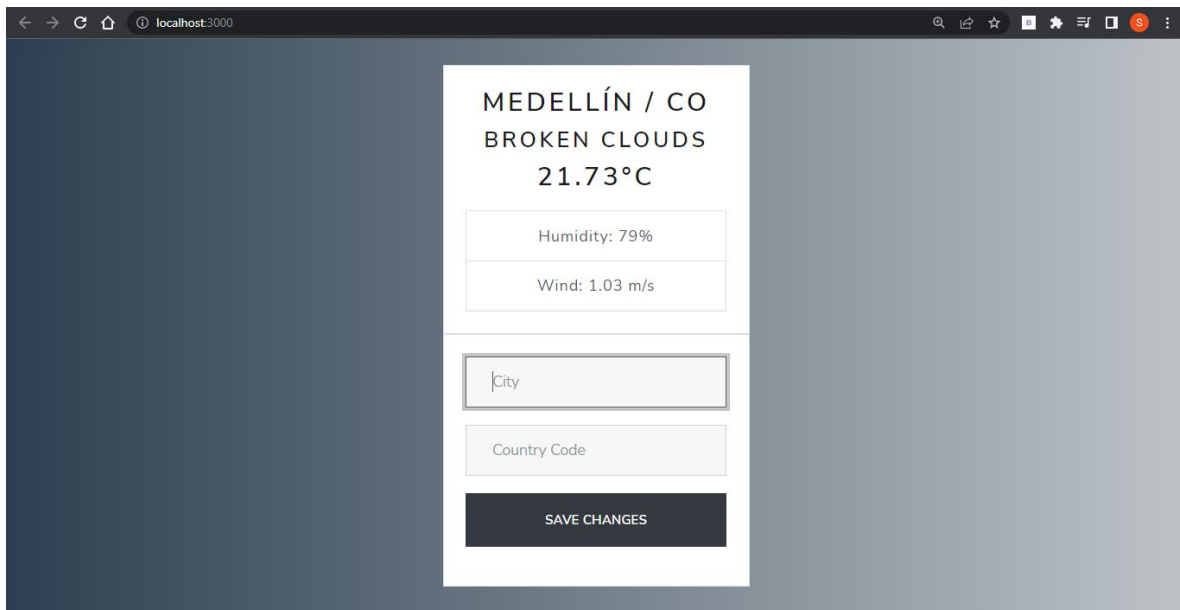
Ahora, vamos a llamar esta clase UI e instanciarla dentro del index.js, así que como primer paso la llamamos (línea 7), luego la instanciamos o llamamos para poder ejecutarla más adelante (línea 12) y para terminar, tendremos que llamar el método render, y enviarle lo que antes habíamos puesto a imprimir por consola (línea 18).

```
1 // Se importan los estilos
2 require('./index.css');
3
4 // Importamos la clase Weather desde el archivo weather
5 const { Weather } = require('./weather');
6 // Importamos la clase UI desde el archivo UI
7 const { UI } = require('./UI');
8
9 // Se crea una nueva instancia Weather
10 const weather = new Weather('Medellin', 'CO');
11 // Se crea una nueva instancia UI
12 const ui = new UI();
13
14 // Se crea la funcion asincrona fetchWeather
15 async function fetchWeather () {
16     const data = await weather.getWeather();
17     console.log(data);
18     ui.render(data);
19 }
20
21 document.addEventListener('DOMContentLoaded', fetchWeather);
```

Ahora, estamos llamando el método render y le enviamos los datos del clima, pero el método no tiene nada, así que vamos al archivo UI.js, y vamos al método render, lo que tendremos que hacer, es llamar una por una las variables e irles poniendo el dato correspondiente, por ejemplo, en la línea 19, llamamos la variable que creamos en el método constructor, llamada location, llamamos también la función textContent, que lo que hace es insertar texto y por último, llamamos desde los datos del clima, exactamente lo que se necesita mostrar, en este ejemplo fue el nombre y el código del país.

```
15 // Metodo para mostrar en la vista los datos
16 render(weather)
17 {
18     // Con la funcion textContent insertamos texto en el elemento
19     this.location.textContent = weather.name + ' / ' + weather.sys.country;
20     this.desc.textContent = weather.weather[0]['description'];
21     this.string.textContent = weather.main.temp + '°C';
22     this.humidity.textContent = 'Humidity: ' + weather.main.humidity + '%';
23     this.wind.textContent = 'Wind: ' + weather.wind.speed + ' m/s';
24 }
25
26 }
```

Ahora, podremos ir al aplicativo y veremos que nos va a mostrar la información de la ciudad que habíamos definido antes (Paso 40), así vamos al servidor de desarrollo para ver los cambios.



Ahora, sólo nos va a mostrar esta ciudad, así que vamos al archivo weather.js y crearemos un nuevo método, para cambiar la localización o la ciudad de la que vemos la información, tendremos que enviarle la ciudad y el código del país, y actualizamos la variable.

```
24      // Metodo para cambiar la ciudad consultada
25      changeLocation(city, countryCode)
26      {
27          this.city = city;
28          this.countryCode = countryCode;
29      }
30
31 }
```

Vamos entonces ahora al archivo principal (index.js), donde llamaremos debajo de la función fetchWeather, el botón para guardar la información y escucharemos su evento click, y corremos una función, donde como primer paso tomamos los valores que el usuario escribe como ciudad y el código del país (línea 24,25); luego ejecutamos el método changeLocation de la clase Weather (línea 27) y le enviamos estos dos valores (ciudad, código del país), luego ejecutamos la función fetchWeather para poder actualizar el render de la vista (línea 28) y como última cosa, hacemos que no se recargue la página con preventDefault (línea 30).

```
21 // Se obtiene el elemento del boton, se escucha el evento y se realiza una funcion
22 document.getElementById('w-change-btn').addEventListener('click', (e) => {
23     // Se obtienen los elementos de ciudad y codigo del pais
24     const city = document.getElementById('city').value;
25     const countryCode = document.getElementById('country-code').value;
26
27     weather.changeLocation(city, countryCode);
28     fetchWeather();
29
30     e.preventDefault();
31 })
```

Ahora iremos a store.js, para guardar los datos en el localStorage y que cuando ingresemos al aplicativo nos muestre la última ciudad consultada, o que si entramos por primera vez, nos muestre una ciudad predefinida, en este caso será Medellín. Luego de ingresar al archivo creamos una clase que vamos a exportar también llamada Store, y en principio, crearemos su clase constructora, donde definimos las variables city, para la ciudad; countryCode, para el código del país; defaultCity, para la ciudad por defecto cuando se ingrese por primera vez; y defaultCountry, para el código de país por defecto cuando se ingrese por primera vez.

```
1  // Exportamos la clase para poder importarla en el index
2  export class Store {
3
4      // Metodo constructor (Se ejecuta solo)
5      constructor()
6      {
7          this.city;
8          this.countryCode;
9          // Se definen los valores predeterminados del aplicativo
10         this.defaultCity = 'Medellin';
11         this.defaultCountry = 'CO';
12     }
13 }
```

Ahora, crearemos dentro de este mismo archivo, un nuevo método llamado `getLocationData`, donde se evalúa si dentro del `localStorage`, no hay nada guardado, enviamos los valores por defecto de la ciudad con su respectivo código del país, en cambio si ya tiene algo, traemos desde el `localStorage` esos datos mediante la instrucción `getItem`, y los guardamos en las variables locales, en la línea 20 se guarda en la variable local el valor de la ciudad y en la línea 26, se guarda en la variable local el código del país. Para terminar, retornamos como ciudad, la ciudad que tengamos en el `localStorage`, o la predeterminada, y lo mismo con el código del país.

```
14 // Metodo para traer del localStorage la informacion antes guardada
15 getLocationData()
16 {
17     if (localStorage.getItem('city') === null) {
18         this.city = this.defaultCity;
19     } else {
20         this.city = localStorage.getItem('city');
21     }
22
23     if (localStorage.getItem('countryCode') === null) {
24         this.countryCode = this.defaultCountry;
25     } else {
26         this.countryCode = localStorage.getItem('countryCode');
27     }
28
29     return{
30         city: this.city,
31         countryCode: this.countryCode
32     }
33 }
34 }
```



Ahora, crearemos otro método, llamado `setLocationData`, le tendremos que enviar la ciudad y el código del país, ya que este será el encargado de enviar la información y guardarla en el `localStorage`, para guardar información en el `localStorage`, utilizamos el `setItem` y le tenemos que enviar dos valores, el primero es el nombre donde lo guardaremos, y el segundo es el valor que queremos guardar; primero guardamos la ciudad (línea 38), y luego guardamos el código del país (línea 39).

```
35 // Metodo para guardar informacion en el localStorage
36 setlocationData(city,countryCode)
37 {
38     localStorage.setItem('city' , city);
39     localStorage.setItem('countryCode', countryCode);
40 }
41 }
```

Ahora, tendremos que ir al archivo `index.js`, para llamar esta clase y poder utilizar sus métodos, como ya lo hemos hecho anteriormente con las demás clases. Así que ahora, requerimos el archivo poniendo la ubicación de este y traemos exactamente la clase `Store` (línea 9); luego, instanciamos la clase `Store` (línea 12), y para terminar, traemos los valores `city` y `countryCode` llamando el método `getLocationData` de la clase `Store`. Y como último paso, le pasaremos estos datos a la instancia del clima (línea 15), o sea, dentro de los paréntesis, le enviaremos estas dos variables.

```
1 // Se importan los estilos
2 require('./index.css');
3
4 // Importamos la clase Weather desde el archivo weather
5 const { Weather } = require('./weather');
6 // Importamos la clase UI desde el archivo UI
7 const { UI } = require('./UI');
8 // Importamos la clase store desde el archivo store
9 const { Store } = require('./store');
10
11 // Se crea una nueva instancia store
12 const store = new Store();
13 const { city, countryCode } = store.getLocationData();
14 // Se crea una nueva instancia Weather
15 const weather = new Weather(city, countryCode);
16 // Se crea una nueva instancia UI
17 const ui = new UI();
```

Ahora, en este mismo archivo, bajaremos hasta donde escuchamos el evento del botón y lo único que tendremos es llamar al método setLocationData de la clase Store y le enviamos los valores que el usuario digita en los campos de ciudad y código del país, al llamar este método lo que hacemos es guardar la información que el usuario digitó en el localStorage.

```
26 // Se obtiene el elemento del boton, se escucha el evento y se realiza una funcion
27 document.getElementById('w-change-btn').addEventListener('click', (e) => {
28     // Se obtienen los elementos de ciudad y codigo del pais
29     const city = document.getElementById('city').value;
30     const countryCode = document.getElementById('country-code').value;
31
32     weather.changeLocation(city, countryCode);
33     store.setLocationData(city, countryCode);
34     fetchWeather();
35
36     e.preventDefault();
37 })
```

Hemos terminado el desarrollo de nuestro aplicativo, ahora lo que tendremos que hacer es ir a la terminal dentro del visual studio code (Paso 3), cuando abramos la terminal, estará algo tipo así:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  JUPYTER
asset index.html 2.49 KiB [emitted]
asset main.1294e46583efde0a9d89.hot-update.json 28 bytes [emitted] [immutable] [hmr]
Entrypoint main 274 KiB = bundle.js 271 KiB main.1294e46583efde0a9d89.hot-update.js 2.65 KiB
cached modules 174 KiB [cached] 35 modules
runtime modules 27.3 KiB 13 modules
./src/app/index.js 1.31 KiB [built] [code generated]
webpack 5.74.0 compiled successfully in 103 ms
```

Esto pasa es porque tenemos activo el servidor de desarrollo, lo que tendremos que hacer es darle a la tecla control y c (Ctrl + C), nos aparecerá un mensaje preguntando que si queremos terminar el trabajo por lotes; tendremos que digitar "S". Al hacer esto, cerrará el servidor de desarrollo y nos mostrará de nuevo la terminal común.

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  JUPYTER
webpack 5.74.0 compiled successfully in 103 ms
<i> [webpack-dev-server] Gracefully shutting down. To force exit, press ^C again. Please wait...
¿Desea terminar el trabajo por lotes (S/N)? S
PS C:\Users\USUARIO\Documents\SENA-2256256\Tecnología en ADSI\Practicas\Actividades\javascript\weatherProject\actividad\weatherProject>
```

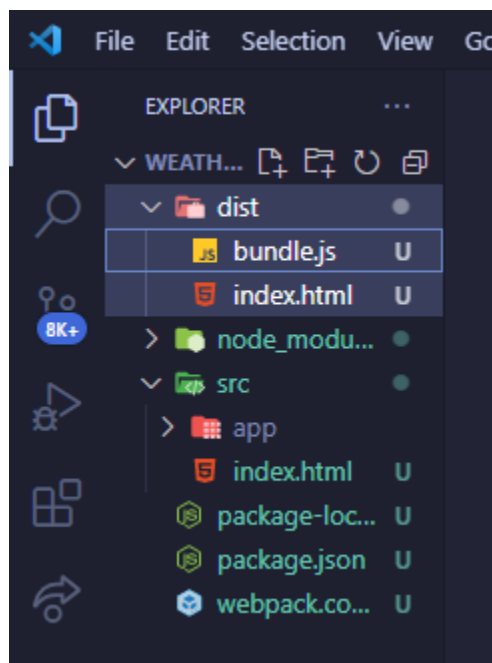
Ahora, para terminar tendremos que digitar ahí en la terminal el siguiente comando: “npm run build”, esto nos creará la carpeta de distribución.

```
PS C:\Users\USUARIO\Documents\SENA-2256256\Tecnología en ADSI\Practicas\Actividades\JavaScript\weatherProject\actividad\weatherProject> npm run build

> weatherproject@1.0.0 build
> webpack --mode production

asset bundle.js 6.68 KiB [emitted] [minimized] (name: main)
asset index.html 1.14 KiB [emitted]
runtime modules 972 bytes 5 modules
cacheable modules 14.4 KiB
  modules by path ./node_modules/ 8.07 KiB
    modules by path ./node_modules/style-loader/dist/runtime/*.js 5.75 KiB 6 modules
```

Al final nos mostrará un mensaje de satisfacción, y aparecerá la nueva carpeta en nuestro explorador de archivos llamada “dist”, dentro sólo habrá dos archivos, el primero llamado index.html, que es donde estará toda la estructura de nuestro aplicativo, y bundle.js, donde está toda la lógica que antes habíamos creado.



Hemos terminado el aplicativo y se debería ver de la siguiente manera:

