Proyecto 2. Extensión de Navegador

Construirá una extensión de navegador que funcione en Edge, Chrome y Firefox. Esta extensión, que es como un minisitio web que está diseñado para una tarea muy específica, verifica la API de señal C02 para el uso de electricidad y la intensidad de carbono de una región determinada, y devuelve una lectura sobre la huella de carbono de la región.

Paso 1.
Comencemos por construir el HTML para el formulario y darle estilo con CSS.

En la carpeta /dist, creará un formulario y un área de resultados. En el archivo index.html, complete el área delineada del formulario:

```
<form class="form-data" autocomplete="on">
     <div>
          <h2>¿Nuevo? Agrega tu información</h2>
     </div>
     <div>
     <label>Nombre de la región</label>
          <input type="text" required class="region-name" />
     </div>
     <div>
          <label>Tu clave API de tmrow</label>
          <input type="text" required class="api-key" />
     </div>
     <button class="search-btn">Enviar</button>
</form>
```

Este es el formulario donde se ingresará la información guardada y se guardará en el almacenamiento local.

Paso 2.
A continuación, cree el área de resultados; debajo de la etiqueta de formulario final, agregue algunos divs:

```
<div class="result">
     <div class="loading">cargando...</div>
     <div class="errors"></div>
     <div class="data"></div>
     <div class="result-container">
          <p><strong>Región:       </strong><span       class="my-
region"></span></p>
```

```
          <p><strong>Uso de carbono: </strong><span class="carbon-
usage"></span></p>
          <p><strong>Porcentaje     de     combustible     fósil:
</strong><span class="fossil-fuel"></span></p>
     </div>
     <button class="clear-btn">Cambia región</button>
</div>
```

Paso 3.
Trabajando en su archivo index.js, comience creando algunas variables const para contener los valores asociados con varios campos:

```
// form fields
const form = document.querySelector('.form-data');
const region = document.querySelector('.region-name');
const apiKey = document.querySelector('.api-key');

// results
const errors = document.querySelector('.errors');
const loading = document.querySelector('.loading');
const results = document.querySelector('.result-container');
const usage = document.querySelector('.carbon-usage');
const fossilfuel = document.querySelector('.fossil-fuel');
const myregion = document.querySelector('.my-region');
const clearBtn = document.querySelector('.clear-btn');
```

Paso 4.
A continuación, agregue detectores de eventos al formulario y el botón de borrar que restablece el formulario, de modo que si un usuario envía el formulario o hace clic en ese botón de restablecimiento, algo sucederá y agregue la llamada para inicializar la aplicación en la parte inferior del archivo:

```
form.addEventListener('submit', (e) => handleSubmit(e));
clearBtn.addEventListener('click', (e) => reset(e));
init();
```

Paso 5.
Ahora vas a construir la función que inicializa la extensión, que se llama init():

```
function init() {
     //si hay algo en localStorage, recójalo
     const storedApiKey = localStorage.getItem('apiKey');
     const storedRegion = localStorage.getItem('regionName');
```

```
      //establecer el icono en verde genérico
      //todo

      if (storedApiKey === null || storedRegion === null) {
            //si no tenemos las claves, mostrar el formulario
            form.style.display = 'block';
            results.style.display = 'none';
            loading.style.display = 'none';
            clearBtn.style.display = 'none';
            errors.textContent = '';
      } else {
         //si hemos guardado claves / regiones en localStorage,
mostrar los resultados cuando se cargan
         displayCarbonUsage(storedApiKey, storedRegion);
            results.style.display = 'none';
            form.style.display = 'none';
            clearBtn.style.display = 'block';
      }
};

function reset(e) {
      e.preventDefault();
      //borrar almacenamiento local solo para la región
      localStorage.removeItem('regionName');
      init();
}
```

Paso 6.
Cree una función llamada handleSubmit que acepte un argumento de evento (e). Detenga la propagación del evento (en este caso, queremos que el navegador no se actualice) y llame a una nueva función, setUpUser, pasando los argumentos apiKey.value y region.value. De esta manera, utiliza los dos valores que se introducen a través del formulario inicial cuando se completan los campos correspondientes.

```
function handleSubmit(e) {
      e.preventDefault();
      setUpUser(apiKey.value, region.value);
}
```

Paso 7.
Pasando a la función setUpUser, aquí es donde configura los valores de almacenamiento local para apiKey y regionName. Agrega una nueva función:

```
function setUpUser(apiKey, regionName) {
      localStorage.setItem('apiKey', apiKey);
      localStorage.setItem('regionName', regionName);
      loading.style.display = 'block';
      errors.textContent = '';
      clearBtn.style.display = 'block';
      //make initial call
      displayCarbonUsage(apiKey, regionName);
}
```

Esta función establece un mensaje de carga para mostrar mientras se llama a la API. ¡En este punto, ha llegado a crear la función más importante de esta extensión de navegador!

Paso 8.
Cree una nueva función para consultar la API C02Signal:

```
import axios from '../node_modules/axios';

async function displayCarbonUsage(apiKey, region) {
      try {
            await axios
                  .get('https://api.co2signal.com/v1/latest', {
                        params: {
                              countryCode: region,
                        },
                        headers: {
                              'auth-token': apiKey,
                        },
                  })
                  .then((response) => {
                        let                    CO2                    =
Math.floor(response.data.data.carbonIntensity);

                        //calculateColor(CO2);

                        loading.style.display = 'none';
                        form.style.display = 'none';
                        myregion.textContent = region;
                        usage.textContent =

      Math.round(response.data.data.carbonIntensity)   +   '   grams
(grams C02 emitted per kilowatt hour)';
                        fossilfuel.textContent =

      response.data.data.fossilFuelPercentage.toFixed(2) +
```

```
                        '% (percentage of fossil fuels used to
generate electricity)';
                    results.style.display = 'block';
                });
     } catch (error) {
          console.log(error);
          loading.style.display = 'none';
          results.style.display = 'none';
          errors.textContent = 'Sorry, we have no data for the
region you have requested.';
     }
}
```

Paso 9.
Trabajando en / src / index.js, agregue una función llamada calculateColor() después de la serie de variables const que estableció para obtener acceso al DOM:

```
function calculateColor(value) {
     let co2Scale = [0, 150, 600, 750, 800];
     let colors = ['#2AA364', '#F5EB4D', '#9E4229', '#381D02',
'#381D02'];

     let closestNum = co2Scale.sort((a, b) => {
          return Math.abs(a - value) - Math.abs(b - value);
     })[0];
     console.log(value + ' is closest to ' + closestNum);
     let num = (element) => element > closestNum;
     let scaleIndex = co2Scale.findIndex(num);

     let closestColor = colors[scaleIndex];
     console.log(scaleIndex, closestColor);

     chrome.runtime.sendMessage({ action: 'updateIcon', value: {
color: closestColor } });
}
```

Paso 10.
Ahora, en la función init (), configure el ícono en verde genérico para comenzar nuevamente llamando a la acción updateIcon de Chrome:

```
chrome.runtime.sendMessage({
     action: 'updateIcon',
          value: {
               color: 'green',
          },
```

```
});
```

Paso 11.
A continuación, llame a la función que acaba de crear agregándola a la promesa devuelta por la API C02Signal:

```
//let CO2...
calculateColor(CO2);
```

Paso 12.
Finalmente, en /dist/background.js, agregue el oyente para estas llamadas de acción en segundo plano:

```
chrome.runtime.onMessage.addListener(function (msg, sender, sendResponse) {
     if (msg.action === 'updateIcon') {
          chrome.browserAction.setIcon({ imageData: drawIcon(msg.value) });
     }
});
//tomado de la extensión Energy Lollipop, ¡buena característica!
function drawIcon(value) {
     let canvas = document.createElement('canvas');
     let context = canvas.getContext('2d');

     context.beginPath();
     context.fillStyle = value.color;
     context.arc(100, 100, 50, 0, 2 * Math.PI);
     context.fill();

     return context.getImageData(50, 50, 100, 100);
}
```

Paso 12.
Construya su extensión con el comando: npm run build