Sistema de Consulta Meteorológica Integrado con APIs Oficiales

Proyecto de desarrollo backend y frontend con Laravel y Leaflet

ÍNDICE

1.- INTRODUCCIÓN GENERAL DEL PROYECTO

Este proyecto tiene como objetivo ofrecer una plataforma web que permita consultar información meteorológica en tiempo real desde dos fuentes oficiales: Eltiempo.net y AEMET. Para ello se ha desarrollado un sistema basado en Laravel (PHP) y una interfaz web moderna con HTML, CSS y JavaScript. Se integra también un mapa interactivo con Leaflet para visualizar puntos de control meteorológicos en toda España.

Tecnologías utilizadas

Tecnología	Uso principal
Laravel	Backend / API REST
PHP	Lógica del servidor
MySQL	Almacenamiento de datos
JavaScript	Lógica frontend / interactividad
Leaflet	Mapa interactivo
HTML/CSS	Estructura y estilo de la web
Eltiempo.net	Consulta meteorológica
AEMET	Consulta meteorológica

Arquitectura general

Este sistema ha sido diseñado bajo una arquitectura moderna basada en el patrón MVC (Modelo-Vista-Controlador), utilizando el framework Laravel para el backend y tecnologías web estándar (HTML, CSS, JavaScript) para el frontend.

[Usuario]→[Interfaz Web (HTML CSS JS)]→[Rutas y Controladores]→[Base de Datos]

ENLACE DE REPOSITORIO

https://github.com/xZeiwan/ProyectoTrevenque

2.- ARGUMENTACIÓN HTML

El archivo index.html constituye la interfaz principal del sistema meteorológico. En él se organizan todos los elementos visuales que permiten al usuario realizar consultas del tiempo, tanto a través de Eltiempo.net como mediante la API de AEMET. Además, incluye la integración de un mapa interactivo con Leaflet.

Estructura general

El index.html está dividido en tres secciones principales:

- Consultas meteorológicas con Eltiempo.net.
- Consultas meteorológicas con AEMET.
- Visualización de mapa interactivo.

Cada sección está bien diferenciada mediante div y clases específicas para facilitar su diseño y control desde CSS y JavaScript.

```
<!-- Sección ELTIEMPO.NET -->
<div class="seccion seccion-eltiempo">
  <h2>Consultas El-tiempo.net</h2>
  <!-- Por provincia -->
  <div class="bloque-consulta">
    <h3>Por provincia (El-tiempo.net)</h3>
    <select id="provinciaSelect" onchange="cargarMunicipiosDesdeSelect()">
      <option value="">Seleccione provincia</option>
    </select>
    <button type="button" onclick="buscarProvincia()">Buscar</button>
    <div id="resultadoProvincia"></div>
  </div>
  <!-- Por municipio -->
  <div class="bloque-consulta">
    <h3>Por municipio (El-tiempo.net)</h3>
    <select id="municipiosSelect">
      <option value="">Seleccione municipio</option>
    </select>
    <button type="button" onclick="buscarMunicipio()">Buscar</button>
    <div id="resultadoMunicipio"></div>
  </div>
  <!-- Por comunidad -->
  <div class="bloque-consulta">
    <h3>Por comunidad autónoma (El-tiempo.net)</h3>
```

Los selectores se utilizan para que el usuario elija la comunidad autónoma, la provincia y el municipio. El sistema carga los datos dinámicamente desde Laravel y muestra los resultados en los específicos.

El mapa se define un contenedor #map, que es inicializado con Leaflet desde mapa.js.

```
<!-- Sección del mapa -->
<div class="seccion seccion-mapa">
  <h2 style="text-align:center; margin-top: 40px;">Mapa Interactivo de Puntos de Control</h2>
  <div id="map"></div>
</div>
```

Cada marcador en el mapa representa un punto de control meteorológico vinculado a una capital de provincia.

3.- ARGUMENTACIÓN CSS

El archivo styles.css define el estilo visual de todo el sistema meteorológico. Se ha diseñado con el objetivo de ofrecer una interfaz clara, estructurada y profesional, dividiendo el contenido en secciones bien diferenciadas y adaptadas al contenido dinámico.

Mis objetivos de diseño

- Dividir visualmente el área de consulta meteorológica de Eltiempo.net y la de AEMET en dos mitades simétricas.

- Mantener una estética limpia, sin sobrecarga visual.

- Garantizar una buena legibilidad y accesibilidad.

- Adaptarse correctamente al uso de tarjetas, selectores y mapas.

```css
/* ==========================
   Contenedor principal
   ========================== */
.wrapper {
  max-width: 1100px;                        /* Ancho máximo */
  margin: 30px auto;                        /* Centrado y separación superior */
  background: #fff;                           /* Fondo blanco */
  border-radius: 14px;                      /* Bordes redondeados */
  box-shadow: 0 4px 24px rgba(0,0,0,0.08);    /* Sombra suave */
  padding: 32px 24px 24px 24px;             /* Espaciado interno */
}

/* ==========================
   Flex para consultas en dos columnas
   ========================== */
.consultas-flex {
  display: flex;                            /* Flexbox para columnas */
  gap: 32px;                                /* Espacio entre columnas */
  justify-content: space-between;
  align-items: stretch;                     /* Ambas columnas igual de altas */
  margin-bottom: 36px;
}

/* ==========================
   Secciones diferenciadas (columnas)
   ========================== */
.seccion-eltiempo, .seccion-aemet {
  flex: 1 1 0;                              /* Que ambas ocupen el mismo espacio */
  min-width: 0;
  background: #f8fafc;                        /* Fondo ligeramente gris */
  border: 1px solid #e3e7ee;                  /* Borde claro */
  border-radius: 12px;
```

El archivo organiza los estilos según las siguientes áreas:

- Sección #eltiempo-section
- Sección #aemet-section

El mapa se presenta con bordes redondeados y un borde gris para integrarlo visualmente en el diseño. Se añadieron reglas @media para adaptarse mejor a móviles y tablets.

4.- ARGUMENTACIÓN CONTROLADORES CSS

Los controladores del proyecto Laravel son los encargados de recibir las solicitudes desde la interfaz web (mediante JavaScript), comunicarse con APIs externas (Eltiempo.net y AEMET) y devolver las respuestas procesadas al frontend. Representan el núcleo de la lógica del sistema y garantizan una separación clara entre presentación e implementación.

Cada controlador se centra en una única API, respetando el principio de responsabilidad única (SRP) y manteniendo el código modular, limpio y fácil de mantener.

Controladores Eltiempo.net

Estos controladores se encargan de consultar los servicios meteorológicos ofrecidos gratuitamente por Eltiempo.net. La lógica se basa en endpoints públicos que no requieren autenticación, lo que permite obtener datos de provincias de forma directa.

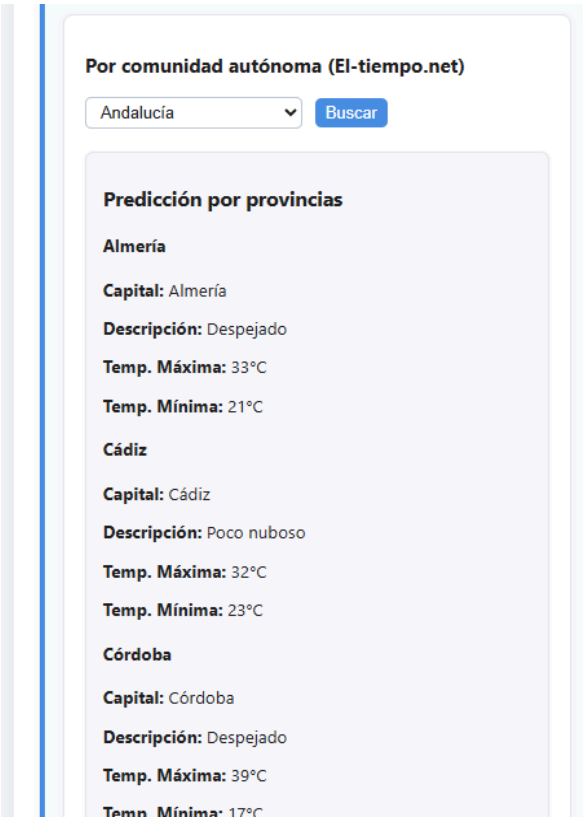
Funcionalidades clave

- Consulta del tiempo por provincia: El sistema permite al usuario seleccionar una provincia directamente y obtener su predicción desde Eltiempo. El controlador consulta el endpoint de capitales de provincia: https://www.el-tiempo.net/api/json/v2/provincias y filtra la provincia correspondiente por nombre, extrayendo sus datos meteorológicos para mostrar en la sección de resultado provincial o en los popups del mapa.

- Consulta de municipios por provincia: Uno de los puntos más importantes del controlador es la gestión dinámica de municipios. Para ello, se ha creado la ruta: /api/municipios/{nombreProvincia}. Este endpoint recibe el nombre de una provincia como parámetro, y busca los municipios asociados en la base de datos local o desde la API si es necesario. La lógica filtra los municipios correspondientes

mediante coincidencia por nombre, lo que permite poblar correctamente el selector dinámico en la interfaz web. Esto se hace así ya que Eltiempo no ofrece consulta directa al tiempo por municipios. La lista de municipios no se carga hasta que se seleccione una provincia.

- Consulta del tiempo por comunidad autónoma: Una de las ampliaciones más relevantes del proyecto es la posibilidad de consultar el tiempo agrupado por comunidad autónoma. Para implementar esta funcionalidad se ha llevado a cabo la creación de la tabla comunidadautonoma en la base de datos y el enlace de cada provincia a una comunidad autónoma, añadiendo una clave foránea o relación lógica.
  También se desarrolló un método en el controlador que, al recibir el nombre de una comunidad, recupera todas las provincias asociadas y sus predicciones una a una, agrupando la información en un formato útil para el frontend.



Control de errores

Se aplica una lógica try/catch en cada petición a Eltiempo.net, con validación del código de respuesta y captura de posibles fallos de conexión. Esto permite mostrar mensajes de error personalizados en el frontend si el proveedor no responde.

```php
try {
    // Realiza la petición HTTP a la API externa
    $response = Http::get($url);
    // Si la respuesta no es exitosa, devuelve error 404
    if (!$response->successful()) {
        return response()->json(['error' => 'No hay datos disponibles para este municipio'], 404);
    }

    // Obtiene los datos de la respuesta
    $data = $response->json();
    // Añade los datos del municipio de la BD al resultado
    $data['municipio'] = $municipio;
    // Devuelve el resultado como JSON
    return response()->json($data);

} catch (\Exception $e) {
    // Si ocurre un error al conectar con la API externa, devuelve error 500
    return response()->json(['error' => 'Fallo al conectar con el-tiempo.net', 'detalle' => $e->getMessage()], 500)
}
```

Controladores AEMET

Los controladores AEMET son los responsables de gestionar las peticiones del sistema que requieren datos meteorológicos desde la API oficial de la Agencia Estatal de Meteorología (AEMET). A diferencia de Eltiempo.net, esta API exige autenticación con una clave personal (API KEY) y trabaja con un sistema de descarga indirecta de datos, lo cual implica una estructura más compleja.

Flujo en dos pasos

La API de AEMET no devuelve los datos directamente. En lugar de eso, el proceso consiste en:

- Primera solicitud: Se realiza una petición autenticada a un endpoint como: https://opendata.aemet.es/opendata/api/prediccion/provincia/dia/{CODAEMET} Esta llamada no devuelve los datos, sino un JSON con una clave datos que contiene una URL temporal de descarga.

- Segunda solicitud: Se utiliza la URL incluida en datos para realizar una nueva llamada, que ya devuelve el contenido meteorológico. Este flujo está encapsulado dentro del método principal del controlador, de forma que el frontend recibe los datos ya procesados sin preocuparse del sistema de doble llamada.

```php
public function consultarPorCodigo($codigo)
{
    // Llamada a AEMET con el endpoint correcto
    $response = Http::withHeaders([
        'api_key' => env('e[redacted]
    ])->get("https://opendata.aemet.es/opendata/api/prediccion/provincia/hoy/" . urlencode($codigo));

    if ($response->failed()) {
        return response()->json([
            'error' => 'Error al consultar la API de AEMET',
            'codigo' => $codigo,
            'estado' => $response->status(),
            'respuesta' => $response->body()
        ], 500);
    }

    // Obtener URL de datos JSON
    $datosUrl = $response->json()['datos'] ?? null;
```

Funcionalidades clave

- Consulta del tiempo por municipio: Permite consultar la predicción meteorológica diaria por municipios concretos, utilizando el identificador indicativo oficial de AEMET para cada uno. Se obtiene un JSON estructurado con la predicción para los próximos días, que se muestra en el frontend a través de aemet.js.

- Consulta del tiempo por provincias: Esta funcionalidad permite consultar predicción meteorológica diaria a nivel de provincia, utilizando el código oficial de AEMET. Esta responde con una URL de descarga como en el caso anterior.

  Particularidad: AEMET devuelve un archivo de texto plano, lo que exige un tratamiento especial desde el cliente.
  Solución: El controlador devuelve directamente el cuerpo plano (body()) al frontend, y el archivo aemet.js se encarga de leer el texto plano, decodificarlo manualmente y filtrar solo los campos deseados (temperaturas, cielo, viento, etc).

- Consulta por comunidades autónomas: La consulta por comunidades se realiza de forma similar a la usada con Eltiempo, pero accediendo a los datos desde la estructura de AEMET. Se basa en la tabla comunidadautonomaaemet, donde se han enlazado todas las provincias correspondientes a cada comunidad:

  Al recibir una comunidad autónoma por parámetro, se recuperan todas las provincias asociadas a ella.
  Por cada provincia, se realiza una consulta meteorológica individual.
  Se compilan las predicciones por provincia y se devuelven al frontend.

5.- ARGUMENTACIÓN JAVASCRIPT

Los archivos JavaScript de este proyecto son responsables de la interacción del usuario con la interfaz, la obtención dinámica de datos desde las APIs expuestas en Laravel y su presentación visual ordenada. Se han dividido en dos archivos independientes, uno para cada proveedor de datos: Eltiempo.net (tiemponet.js) y AEMET (aemet.js), siguiendo el principio de separación de responsabilidades.

Código Eltiempo.net

Funcionalidades clave

- Normalizar texto: Normaliza un texto eliminando acentos y convirtiéndolo en minúsculas. Es útil para comparar nombres de provincias o municipios de forma insensible a mayúsculas, minúsculas y tildes.

```
/**
 * Normaliza un texto eliminando acentos y convirtiendo a minúsculas.
 * Útil para comparar nombres de provincias o municipios.
 */
function normalizarTexto(texto) {
  return texto.normalize("NFD").replace(/[\u0300-\u036f]/g, "").toLowerCase().trim();
}
```

- Carga de provincias y comunidades: al cargar la página, se obtienen y muestran las provincias y comunidades autónomas disponibles, rellenando los <select> correspondientes.

```
/**
 * Carga el listado de provincias desde la API y las añade al <select> correspondiente.
 * Guarda el listado en la variable global listaProvincias.
 */
async function cargarProvincias() {
  console.log("cargarProvincias llamada");
  try {
    const res = await fetch(`${apiBase}/provincias`);
    const data = await res.json();
    console.log(data);
    listaProvincias = data.provincias;

    const select = document.getElementById('provinciaSelect');
    // Añade cada provincia como opción en el select
    listaProvincias.forEach(p => {
      const option = new Option(p.NOMBRE_PROVINCIA, p.NOMBRE_PROVINCIA);
      select.appendChild(option);
    });
  } catch (err) {
    alert("No se pudieron cargar las provincias");
  }
}
```

Las funciones cargarProvincias y cargarComunidades realizan peticiones a la API y rellenan los selectores.

- Selección y carga de municipios: cuando el usuario selecciona una provincia, se cargan los municipios asociados.

```
/**
 * Carga el listado de municipios de la provincia seleccionada y los añade al <select> correspondiente.
 * Se ejecuta cuando el usuario selecciona una provincia.
 */
async function cargarMunicipiosDesdeSelect() {
  const valor = document.getElementById('provinciaSelect').value;
  const provincia = listaProvincias.find(p => p.NOMBRE_PROVINCIA === valor);
  if (!provincia) return;

  try {
    const res = await fetch(`${apiBase}/municipios/${provincia.NOMBRE_PROVINCIA}`);
    const municipios = await res.json();

    const select = document.getElementById('municipiosSelect');
    select.innerHTML = '<option value="">Elige un municipio</option>';
    // Añade cada municipio como opción en el select
    municipios.forEach(m => {
      const option = new Option(m.NOMBRE, `${m.CODPROV}-${m.CODIGOINE}`);
      select.appendChild(option);
    });
  } catch {
    alert("Error cargando municipios");
  }
}
```

Cuando el usuario selecciona una provincia, esta función consulta la API para obtener los municipios de esa provincia y los añade al <select> de municipios.

- Consulta y visualización de predicción provincial: busca la predicción meteorológica de la provincia seleccionada llamando a la API y, si la respuesta es válida, muestra los datos usando la función mostrarProvincia.

```
/**
 * Busca y muestra la predicción meteorológica de la provincia seleccionada.
 * Llama a la API y muestra los datos en el div correspondiente.
 */
async function buscarProvincia() {
  const valor = document.getElementById('provinciaSelect').value;
  const provincia = listaProvincias.find(p => p.NOMBRE_PROVINCIA === valor);
  if (!provincia) return;

  try {
    const response = await fetch(`${apiBase}/tiempo/${provincia.CODPROV}`);
    const data = await response.json();
    mostrarProvincia(data);
  } catch {
    alert("Error consultando la provincia");
  }
}
```

- Mostrar provincias: muestra en pantalla la predicción meteorológica de la capital de la provincia seleccionada, incluyendo descripción del cielo y temperaturas máxima y mínima.

```
/**
 * Muestra en pantalla la predicción meteorológica de la provincia recibida.
 * Busca la capital de la provincia y muestra su información.
 */
function mostrarProvincia(data) {
  const ciudad = data.ciudades?.find(c => c.name === data.provincia?.CAPITAL_PROVINCIA);
  const div = document.getElementById('resultadoProvincia');
  if (!ciudad) {
    div.innerHTML = `<p style="color:red;">No se encontraron datos</p>`;
    return;
  }

  div.innerHTML = `
    <h2>Tiempo en ${ciudad.name} (provincia de ${data.provincia.NOMBRE_PROVINCIA})</h2>
    <p><strong>Descripción:</strong> ${ciudad.stateSky?.description || 'No disponible'}</p>
    <p><strong>Temp. Máxima:</strong> ${ciudad.temperatures?.max ?? 'N/A'}ºC</p>
    <p><strong>Temp. Mínima:</strong> ${ciudad.temperatures?.min ?? 'N/A'}ºC</p>
  `;
}
```

- Buscar y mostrar municipios: busca la predicción meteorológica del municipio seleccionado llamando a la API y muestra los datos usando la función mostrarMunicipio.

```
/**
 * Busca y muestra la predicción meteorológica del municipio seleccionado.
 * Llama a la API y muestra los datos en el div correspondiente.
 */
async function buscarMunicipio() {
  const valor = document.getElementById('municipiosSelect').value;
  if (!valor) return;
  const [codprov, codigoine] = valor.split('-');

  try {
    const response = await fetch(`${apiBase}/tiempo-municipio/${codprov}/${codigoine}`);
    const data = await response.json();

    if (data.error) {
      document.getElementById('resultadoMunicipio').innerHTML = `<p style="color:red;">${data.error}</p>`;
      return;
    }

    mostrarMunicipio(data);
  } catch {
    alert("Error consultando el municipio");
  }
}

/**
 * Muestra en pantalla la predicción meteorológica del municipio recibido.
 */
function mostrarMunicipio(data) {
  const div = document.getElementById('resultadoMunicipio');
  div.innerHTML = `
    <h2>Tiempo en ${data.municipio.NOMBRE}</h2>
    <p><strong>Descripción:</strong> ${data.stateSky?.description || 'No disponible'}</p>
    <p><strong>Temperatura:</strong> ${data.temperatura_actual ?? 'N/A'}°C</p>
    <p><strong>Humedad:</strong> ${data.humedad ?? 'N/A'}%</p>
    <p><strong>Viento:</strong> ${data.viento ?? 'N/A'} km/h</p>
  `;
}
```

- Cargar y buscar comunidad: Obtiene el listado de comunidades autónomas desde la API y las añade como opciones al <select> correspondiente.

```
/**
 * Carga el listado de comunidades autónomas desde la API y las añade al <select> correspondiente.
 */
async function cargarComunidades() {
  try {
    const res = await fetch(`${apiBase}/comunidades`);
    const comunidades = await res.json();

    const select = document.getElementById('comunidadSelect');
    // Añade cada comunidad como opción en el select
    comunidades.forEach(c => {
      const option = new Option(c.NOMBRE, c.CODIGO);
      select.appendChild(option);
    });
  } catch {
    alert("No se pudieron cargar las comunidades");
  }
}

/**
 * Busca y muestra la predicción meteorológica para todas las provincias de la comunidad seleccionada.
 * Llama a la API y muestra los datos en el div correspondiente.
 */
async function buscarComunidad() {
  const codigo = document.getElementById('comunidadSelect').value;
  if (!codigo) return;

  try {
    const response = await fetch(`${apiBase}/tiempo-comunidad/${codigo}`);
    const data = await response.json();

    if (data.error) {
      document.getElementById('resultadoComunidad').innerHTML = `<p style="color:red;">${data.error}</p>`;
      return;
    }

    mostrarProvinciasDeComunidad(data);
  } catch {
    alert("Error consultando la comunidad");
```

- Mostrar provincias de comunidad: muestra en pantalla la predicción meteorológica de todas las provincias de una comunidad autónoma, mostrando para cada una la información de su capital.

```
/**
 * Muestra en pantalla la predicción meteorológica de todas las provincias de una comunidad.
 * Para cada provincia, muestra la información de su capital.
 */
function mostrarProvinciasDeComunidad(provincias) {
  const div = document.getElementById('resultadoComunidad');
  div.innerHTML = `<h3>Predicción por provincias</h3>`;

  provincias.forEach(p => {
    // Busca la capital de la provincia
    const ciudad = p.datos.ciudades?.find(c => c.name === p.datos.provincia?.CAPITAL_PROVINCIA);

    const html = `
      <div class="prediccion-dia">
        <h4>${p.provincia}</h4>
        <p><strong>Capital:</strong> ${ciudad?.name || 'Desconocido'}</p>
        <p><strong>Descripción:</strong> ${ciudad?.stateSky?.description || 'No disponible'}</p>
        <p><strong>Temp. Máxima:</strong> ${ciudad?.temperatures?.max ?? 'N/A'}ºC</p>
        <p><strong>Temp. Mínima:</strong> ${ciudad?.temperatures?.min ?? 'N/A'}ºC</p>
      </div>
    `;

    div.innerHTML += html;
  });
}
```

Código AEMET

Funcionalidades clave

- Cargar provincias: Esta función realiza una petición al backend para obtener el listado de provincias desde la API de AEMET. Una vez recibidos los datos, limpia el <select> de provincias y añade una opción por cada provincia recibida. Si ocurre un error en la petición, lo muestra por consola.

```
// Carga el listado de provincias AEMET en el <select> correspondiente
async function cargarProvinciasAemet() {
  try {
    // Petición a la API para obtener las provincias
    const res = await fetch(`${aemetApiBase}/aemet/provincias`);
    const data = await res.json();

    // Selecciona el elemento <select> donde se mostrarán las provincias
    const select = document.getElementById('provinciaAemetSelect');
    if (!select) return;

    // Limpia el select y añade la opción por defecto
    select.innerHTML = '<option value="">Seleccione provincia</option>';
    // Añade cada provincia como opción en el select
    data.provincias.forEach(p => {
      const option = document.createElement('option');
      option.value = p.CODPROV;
      option.text = p.NOMBRE;
      select.appendChild(option);
    });
  } catch (err) {
    // Si hay error en la petición, lo muestra por consola
    console.error("Error cargando provincias AEMET:", err);
  }
}
```

- Buscar provincia: obtiene el código de provincia seleccionado y realiza una petición al backend para obtener la predicción. Según el tipo de contenido de la respuesta (application/json o texto plano), procesa los datos de forma diferente:
  - Si es JSON, muestra la predicción de cada municipio.
  - Si es texto, procesa el texto para extraer fecha, ciudad, información y temperaturas, y lo muestra estructurado.

```
// Busca y muestra la predicción de una provincia AEMET (puede ser JSON o texto plano)
async function buscarProvinciaAemet() {
  // Obtiene el código de provincia seleccionado
  const select = document.getElementById('provinciaAemetSelect');
  const codprov = select.value;
  if (!codprov) return;

  // Elemento donde se mostrará el resultado
  const div = document.getElementById('resultadoProvinciaAemet');
  div.innerHTML = '<p>Cargando datos de la provincia...</p>';

  try {
    // Petición a la API para obtener la predicción de la provincia
    const res = await fetch(`${aemetApiBase}/aemet/provincia/${codprov}`);
    const contentType = res.headers.get("content-type");

    // Si la respuesta es JSON (predicción por municipios)
    if (contentType && contentType.includes("application/json")) {
      const data = await res.json();

      // Si hay error en la respuesta, lo muestra
      if (data.error) {
        div.innerHTML = `<p style="color:red;">${data.error}</p>`;
        return;
      }

      // Si no hay datos, lo indica
      if (!Array.isArray(data) || data.length === 0) {
        div.innerHTML = `<p style="color:red;">No hay datos para esta provincia.</p>`;
        return;
      }

      // Muestra la predicción de cada municipio de la provincia
      div.innerHTML = `<h3>Predicción para municipios de la provincia</h3>`;
      data.forEach(item => {
        div.innerHTML += `
          <div class="prediccion-dia" style="border:1px solid #ccc; padding:10px; margin:10px 0; border-radius:8px;">
            <h4>${item.municipio.NOMBRE}</h4>
            <p><strong>Descripción:</strong> ${item.datos?.prediccion?.dia?.[0]?.estadoCielo?.[0]?.descripcion ?? 'No disponible'}</p>
            <p><strong>Descripción:</strong> ${item.datos?.prediccion?.dia?.[0]?.estadoCielo?.[0]?.descripcion ?? 'No disponible'}</p>
            <p><strong>Temp. Máxima:</strong> ${item.datos?.prediccion?.dia?.[0]?.temperatura?.maxima ?? 'N/A'}ºC</p>
            <p><strong>Temp. Mínima:</strong> ${item.datos?.prediccion?.dia?.[0]?.temperatura?.minima ?? 'N/A'}ºC</p>
          </div>
        `;
      });
    } else {
      // Si la respuesta es texto plano (predicción provincial)
      const texto = await res.text();
      let lineas = texto.split('\n');

      // Filtrar líneas no deseadas (cabeceras, etc.)
      lineas = lineas.filter(linea =>
        !linea.startsWith('AGENCIA ESTATAL DE METEOROLOGÍA') &&
        !linea.startsWith('PREDICCIÓN PARA LA PROVINCIA') &&
        !linea.startsWith('PREDICCIÓN VÁLIDA')
      );

      // Unir el texto y normalizar espacios
      const textoPersonalizado = lineas.join(' ').replace(/\s+/g, ' ').trim();

      // Extraer secciones: fecha (hasta "HORA OFICIAL"), ciudad y el resto
      const fechaMatch = textoPersonalizado.match(/^(DÍA.*?HORA OFICIAL)/);
      const fecha = fechaMatch ? fechaMatch[1] : '';
      let resto = textoPersonalizado.replace(fecha, '').trim();

      // Extraer ciudad (mayúsculas al principio)
      const ciudadMatch = resto.match(/^([A-ZÁÉÍÓÚÑ ]+)\s/);
      const ciudad = ciudadMatch ? ciudadMatch[1].trim() : '';
      resto = resto.replace(ciudad, '').trim();

      // Separa el bloque de temperaturas del resto del contenido
      const tempTitulo = 'TEMPERATURAS MÍNIMAS Y MÁXIMAS PREVISTAS (°C):';
      const tempIndex = resto.indexOf(tempTitulo);
      let info = '';
      let temps = '';
```

```
    let temps = '';
    if (tempIndex !== -1) {
      info = resto.substring(0, tempIndex).trim();
      temps = resto.substring(tempIndex + tempTitulo.length).trim();
    } else {
      info = resto;
    }

    // Procesa el bloque de temperaturas para mostrarlo bonito
    let tempsHTML = '';
    if (temps) {
      // Suponemos que el bloque de temperaturas tiene la estructura: "Ciudad MinTemp MaxTemp" repetido
      const tokens = temps.split(/\s+/);
      for (let i = 0; i < tokens.length - 2; i += 3) {
        const ciudadTemp = tokens[i];
        const minima = tokens[i + 1];
        const maxima = tokens[i + 2];
        tempsHTML += `
          <div style="display: flex; justify-content: center; padding: 4px 0; border-bottom: 1px dotted #ccc;">
            <div style="width: 50%; text-align: center;"><strong>${ciudadTemp}</strong></div>
            <div style="width: 50%; text-align: center;">${minima}°C / ${maxima}°C</div>
          </div>
        `;
      }
    }

    // Mostrar la predicción provincial estructurada: fecha, ciudad, info y temperaturas
    div.innerHTML = `
      <div style="max-width:800px; margin: 0 auto; text-align: center; font-size: 1.2em; box-shadow: 2px 2px 8px #666; padding: 1
        ${fecha ? `<div style="margin-bottom: 10px;"><strong>${fecha}</strong></div>` : ''}
        ${(ciudad || info) ? `
          <div style="margin-bottom: 10px;">
            ${ciudad ? `<div><strong>${ciudad}</strong></div>` : ''}
            ${info ? `<div>${info}</div>` : ''}
          </div>
        ` : ''}
        ${tempsHTML ? `
          <div style="margin-bottom: 10px;"><strong>${tempTitulo}</strong></div>
```

```
          ${tempsHTML ? `
            <div style="margin-bottom: 10px;"><strong>${tempTitulo}</strong></div>
            <div>${tempsHTML}</div>
          ` : ''}
        </div>
      `;
    }
  } catch (err) {
    // Si hay error en la petición, lo muestra
    div.innerHTML = `<p style="color:red;">Error al consultar la provincia en AEMET.</p>`;
    console.error("Error al buscar provincia AEMET:", err);
  }
}
```

- Cargar municipios: solicita el listado de municipios a la API y los añade al <select> correspondiente, validando que el código del municipio tenga el formato correcto (5 dígitos).

```
157   // Carga el listado de municipios AEMET en el <select> correspondiente
158   async function cargarMunicipiosAemet() {
159     try {
160       // Petición a la API para obtener los municipios
161       const res = await fetch(`${aemetApiBase}/municipios-aemet`);
162       const data = await res.json();
163
164       // Selecciona el elemento <select> donde se mostrarán los municipios
165       const select = document.getElementById('municipiosAemetSelect');
166       if (!select) return;
167
168       // Añade cada municipio como opción en el select
169       data.forEach(m => {
170         if (!m.codigo || !/^\d{5}$/.test(m.codigo)) return;
171
172         const option = document.createElement('option');
173         option.value = m.codigo;
174         option.text = m.NOMBRE;
175         select.appendChild(option);
176       });
177     } catch (err) {
178       // Si hay error en la petición, lo muestra por consola
179       console.error("Error cargando municipios AEMET:", err);
180     }
181   }
```

- Buscar municipios: valida el código del municipio, consulta la predicción meteorológica y muestra los datos de cada día en el contenedor correspondiente. Si no hay datos, muestra un mensaje de error.

```
183   // Busca y muestra la predicción de un municipio AEMET
184   async function buscarMunicipioAemet() {
185     // Obtiene el código del municipio seleccionado
186     let codigo = document.getElementById('municipiosAemetSelect').value;
187     codigo = codigo.trim();
188
189     // Valida el código del municipio
190     if (!/^\d{5}$/.test(codigo)) {
191       console.error("Código de municipio inválido:", codigo);
192       return;
193     }
194
195     try {
196       // Petición a la API para obtener la predicción del municipio
197       const res = await fetch(`${aemetApiBase}/aemet/municipio/${codigo}`);
198       const datos = await res.json();
199
200       // Elemento donde se mostrará el resultado
201       const contenedor = document.getElementById('resultadoMunicipioAemet');
202       contenedor.innerHTML = ''; // Limpiar contenido previo
203
204       // Verificar predicción disponible
205       const dias = datos?.prediccion?.dia;
206       if (!Array.isArray(dias)) {
207         contenedor.innerHTML = `<p style="color:red;">No se encontraron datos para este municipio.</p>`;
208         return;
209       }
210
211       // Mostrar cabecera
212       contenedor.innerHTML += `<h3>Predicción para ${datos.nombre} (${datos.provincia})</h3>`;
213
214       // Mostrar cada día de predicción
215       dias.forEach(d => {
216         contenedor.innerHTML += `
217           <div class="prediccion-dia" style="border:1px solid #ccc; padding:10px; margin:10px 0; border-radius:8px;">
218             <h4>${d.fecha}</h4>
219             <p><strong>Temperatura:</strong> ${d.temperatura.minima}°C / ${d.temperatura.maxima}°C</p>
220             <p><strong>Estado del cielo:</strong> ${d.estadoCielo[0]?.descripcion ?? 'No disponible'}</p>
221             <p><strong>Viento:</strong> ${d.viento[0]?.direccion ?? '?'} ${d.viento[0]?.velocidad ?? '?'} km/h</p>
```

```
            <p><strong>Temperatura:</strong> ${d.temperatura.minima}°C / ${d.temperatura.maxima}°C</p>
            <p><strong>Estado del cielo:</strong> ${d.estadoCielo[0]?.descripcion ?? 'No disponible'}</p>
            <p><strong>Viento:</strong> ${d.viento[0]?.direccion ?? '?'} ${d.viento[0]?.velocidad ?? '?'} km/h</p>
            <p><strong>Humedad:</strong> ${d.humedadRelativa.minima}% - ${d.humedadRelativa.maxima}%</p>
          </div>
        `;
      });
    } catch (err) {
      // Si hay error en la petición, lo muestra por consola
      console.error("Error al buscar predicción AEMET:", err);
    }
  }
```

- Cargar comunidades autónomas: solicita el listado de comunidades autónomas y las añade al <select> correspondiente, permitiendo al usuario seleccionar una comunidad para consultar su predicción.

```
// Carga el listado de comunidades autónomas desde la API de AEMET y las añade al <select> correspondiente
async function cargarComunidadesAemet() {
  try {
    // Petición a la API para obtener las comunidades
    const res = await fetch(`${aemetApiBase}/aemet/comunidades`);
    const data = await res.json();

    // Selecciona el elemento <select> donde se mostrarán las comunidades
    const select = document.getElementById('comunidadAemetSelect');
    if (!select) return;

    // Limpia el select y añade la opción por defecto
    select.innerHTML = '<option value="">Seleccione comunidad</option>';

    // Añade cada comunidad como opción en el select
    data.forEach(c => {
      const option = document.createElement('option');
      option.value = c.codigo;
      option.text = c.nombre;
      select.appendChild(option);
    });
  } catch (err) {
    // Si hay error en la petición, lo muestra por consola
    console.error("Error cargando comunidades AEMET:", err);
  }
}
```

- Buscar comunidad: Cuando el usuario selecciona una comunidad, la función pide el listado de provincias de esa comunidad y, para cada provincia, solicita y muestra la predicción meteorológica. Si la respuesta es JSON, muestra la predicción por municipios; si es texto, procesa y estructura la información para mostrarla de forma clara.

```
// Busca y muestra la predicción para todas las provincias de una comunidad seleccionada
async function buscarComunidadAemet() {
  // Obtiene el código de la comunidad seleccionada
  const select = document.getElementById('comunidadAemetSelect');
  const codigo = select.value;
  const div = document.getElementById('resultadoComunidadAemet');
  div.innerHTML = 'Cargando...';

  // Si no hay comunidad seleccionada, limpia el resultado y sale
  if (!codigo) {
    div.innerHTML = '';
    return;
  }
```

```
try {
  // Pide a la API el listado de provincias de la comunidad seleccionada
  const res = await fetch(`${aemetApiBase}/aemet/comunidad/${codigo}/provincias`);
  const provincias = await res.json();

  // Si hay error en la respuesta, lo muestra
  if (provincias.error) {
    div.innerHTML = `<p style="color:red;">${provincias.error}</p>`;
    return;
  }
  // Si no hay provincias, lo indica
  if (!Array.isArray(provincias) || provincias.length === 0) {
    div.innerHTML = `<p>No hay provincias para esta comunidad.</p>`;
    return;
  }

  // Prepara el contenedor para mostrar las provincias y sus predicciones
  div.innerHTML = `<h4>Provincias:</h4><div id="provinciasTiempo"></div>`;
  const provinciasTiempoDiv = document.getElementById('provinciasTiempo');

  // Para cada provincia, pide la predicción y la muestra
  for (const provincia of provincias) {
    // Crea un div para mostrar la predicción de la provincia
    const provDiv = document.createElement('div');
    provDiv.innerHTML = `<h5>${provincia.nombre}</h5><div>Cargando predicción...</div>`;
    provinciasTiempoDiv.appendChild(provDiv);

    try {
      // Pide la predicción de la provincia a la API
      const resProv = await fetch(`${aemetApiBase}/aemet/provincia/${provincia.codigo}`);
      const contentType = resProv.headers.get("content-type");

      // Si la respuesta es JSON, la procesa como predicción por municipios
      if (contentType && contentType.includes("application/json")) {
        const data = await resProv.json();
        if (data.error) {
          provDiv.innerHTML = `<h5>${provincia.nombre}</h5><p style="color:red;">${data.error}</p>`;
          continue;
```

```
          continue;
        }
        if (!Array.isArray(data) || data.length === 0) {
          provDiv.innerHTML = `<h5>${provincia.nombre}</h5><p>No hay datos para esta provincia.</p>`;
          continue;
        }
        // Muestra la predicción de cada municipio de la provincia
        let html = '';
        data.forEach(item => {
          html += `
            <div class="prediccion-dia" style="border:1px solid #ccc; padding:10px; margin:10px 0; border-radius:8px;">
              <h6>${item.municipio.NOMBRE}</h6>
              <p><strong>Descripción:</strong> ${item.datos?.prediccion?.dia?.[0]?.estadoCielo?.[0]?.descripcion ?? 'No dispo
              <p><strong>Temp. Máxima:</strong> ${item.datos?.prediccion?.dia?.[0]?.temperatura?.maxima ?? 'N/A'}ºC</p>
              <p><strong>Temp. Mínima:</strong> ${item.datos?.prediccion?.dia?.[0]?.temperatura?.minima ?? 'N/A'}ºC</p>
            </div>
          `;
        });
        provDiv.innerHTML = `<h5>${provincia.nombre}</h5>${html}`;
      } else {
        // Si la respuesta es texto plano, la procesa y muestra de forma estructurada
        const texto = await resProv.text();
        let lineas = texto.split('\n');

        // Filtra las líneas que no se quieren mostrar (cabeceras, etc.)
        lineas = lineas.filter(linea =>
          !linea.startsWith('AGENCIA ESTATAL DE METEOROLOGÍA') &&
          !linea.startsWith('PREDICCIÓN PARA LA PROVINCIA') &&
          !linea.startsWith('PREDICCIÓN VÁLIDA')
        );

        // Une el texto y normaliza los espacios
        const textoPersonalizado = lineas.join(' ').replace(/\s+/g, ' ').trim();

        // Extrae la fecha (hasta "HORA OFICIAL"), la ciudad y el resto del texto
        const fechaMatch = textoPersonalizado.match(/^(DÍA.*?HORA OFICIAL)/);
        const fecha = fechaMatch ? fechaMatch[1] : '';
        let resto = textoPersonalizado.replace(fecha, '').trim();
```

```
344          const textoPersonalizado = lineas.join(' ').replace(/\s+/g, ' ').trim();
345
346          // Extrae la fecha (hasta "HORA OFICIAL"), la ciudad y el resto del texto
347          const fechaMatch = textoPersonalizado.match(/^(DÍA.*?HORA OFICIAL)/);
348          const fecha = fechaMatch ? fechaMatch[1] : '';
349          let resto = textoPersonalizado.replace(fecha, '').trim();
350
351          const ciudadMatch = resto.match(/^([A-ZÁÉÍÓÚÑ ]+)\s/);
352          const ciudad = ciudadMatch ? ciudadMatch[1].trim() : '';
353          resto = resto.replace(ciudad, '').trim();
354
355          // Separa el bloque de temperaturas del resto del contenido
356          const tempTitulo = 'TEMPERATURAS MÍNIMAS Y MÁXIMAS PREVISTAS (°C):';
357          const tempIndex = resto.indexOf(tempTitulo);
358          let info = '';
359          let temps = '';
360          if (tempIndex !== -1) {
361            info = resto.substring(0, tempIndex).trim();
362            temps = resto.substring(tempIndex + tempTitulo.length).trim();
363          } else {
364            info = resto;
365          }
366
367          // Procesa el bloque de temperaturas para mostrarlo bonito
368          let tempsHTML = '';
369          if (temps) {
370            // Busca patrones de "Ciudad Min Max" y los muestra en filas
371            const regex = /([^\d]+?)\s+(\d{1,2})\s+(\d{1,2})(?=\s|$)/g;
372            let match;
373            let bloques = [];
374            while ((match = regex.exec(temps)) !== null) {
375              const nombre = match[1].trim();
376              const min = match[2];
377              const max = match[3];
378              bloques.push(`
379                <div style="display: flex; justify-content: center; align-items: center; padding: 4px 0; border-bottom: 1px do
380                  <div style="width: 60%; text-align: left; font-weight: bold; color: #222;">${nombre}</div>
381                  <div style="width: 40%; text-align: right; font-weight: bold; color: #222;">${min}°C / ${max}°C</div>
382                </div>
383              `);
384            }
385            // Si no se detectó ningún bloque, muestra el texto tal cual
386            if (bloques.length === 0) {
387              tempsHTML = `<div style="text-align:left; font-weight:bold; color:#222;">${temps.replace(/\s+/g, ' ')}</div>`;
388            } else {
389              tempsHTML = bloques.join('');
390            }
391          }
392
393          // Muestra la predicción provincial estructurada: fecha, ciudad, info y temperaturas
394          provDiv.innerHTML = `
395          <div style="max-width:700px; margin: 10px auto; text-align: center; font-size: 1em; box-shadow: 2px 2px 8px #666; paddir
396            ${fecha ? `<div style="margin-bottom: 10px;"><strong>${fecha}</strong></div>` : ''}
397            ${(ciudad || info) ? `
398              <div style="margin-bottom: 10px;">
399                ${ciudad ? `<div><strong>${ciudad}</strong></div>` : ''}
400                ${info ? `<div>${info}</div>` : ''}
401              </div>
402            ` : ''}
403            ${tempsHTML ? `
404              <div style="margin-bottom: 10px;"><strong>${tempTitulo}</strong></div>
405              <div>${tempsHTML}</div>
406            ` : ''}
407          </div>
408        `;
409        }
410      } catch (err) {
411        // Si hay error al consultar la predicción de la provincia, lo muestra
412        provDiv.innerHTML = `<h5>${provincia.nombre}</h5><p style="color:red;">Error al consultar el tiempo.</p>`;
413      }
414    }
415  } catch (err) {
416    // Si hay error general al consultar la comunidad, lo muestra
417    div.innerHTML = `<p style="color:red;">Error al consultar la comunidad en AEMET.</p>`;
418    console.error("Error al buscar comunidad AEMET:", err);
419  }
```

6.- MAPA INTERACTIVO LEAFLET

El sistema de mapa interactivo permite visualizar en un mapa de España los puntos de control almacenados en la base de datos. Al hacer clic en cada punto, se muestra un popup con la predicción meteorológica de la provincia asociada, obtenida en tiempo real desde la API de el-tiempo.net.

- El sistema usa Leaflet para mostrar los puntos de control meteorológico.

- Cada marcador está vinculado a un PuntoControl en base de datos (latitud, longitud).

- Al hacer clic en un marcador se hace una llamada dinámica a la API de Laravel que consulta el tiempo actual desde Eltiempo.net.

- Se usan popups personalizados para mostrar información meteorológica por provincia.

- Al cargar la página, se crea un mapa centrado en España usando la librería Leaflet y se añade la capa base de OpenStreetMap.

```
window.onload = async () => {
  // Inicializa el mapa centrado en España
  const map = L.map("map").setView([40.4168, -3.7038], 6);

  // Añade la capa base de OpenStreetMap
  L.tileLayer("https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png", {
    attribution:
      'Map data &copy; <a href="https://www.openstreetmap.org/">OpenStreetMap</a>',
  }).addTo(map);
```

- Se obtiene la lista de puntos de control desde el backend y se crea un marcador en el mapa para cada uno, usando sus coordenadas.

```
  try {
    // 1. Obtener todos los puntos de control de tu base de datos
    const res = await fetch("http://192.168.56.101:8000/api/puntos-control");
    const puntos = await res.json();

    // Recorre cada punto de control recibido
    puntos.forEach((punto) => {
      // Convierte latitud y longitud a número
      const lat = Number(punto.latitud);
      const lng = Number(punto.longitud);

      // Si no son válidos, ignora este punto
      if (isNaN(lat) || isNaN(lng)) return;

      // Crea un marcador en el mapa para cada punto
      const marker = L.marker([lat, lng]).addTo(map);
```

- Cuando el usuario hace clic en un marcador, se solicita al backend la predicción meteorológica de la provincia asociada al punto.
  - Si la respuesta es correcta, se busca la capital de la provincia y se muestra su predicción (descripción del cielo, temperatura máxima y mínima) en un popup.
  - Si ocurre un error, se muestra un mensaje de error en el popup.

```
28      // Al hacer clic en el marcador, muestra el tiempo de la ciudad/provincia asociada
29      marker.on("click", async () => {
30        try {
31          // Llama a tu backend para obtener el tiempo de la provincia asociada al punto
32          const resTiempo = await fetch(`http://192.168.56.101:8000/api/puntos-control/${punto.id}/tiempo-provincia`);
33          if (!resTiempo.ok) {
34            // Si la respuesta no es correcta, muestra error en el popup
35            marker.bindPopup(`<strong>${punto.nombre}</strong><br><em>No se pudo obtener el tiempo (${resTiempo.status})</
36            return;
37          }
38          // Parsea la respuesta como JSON
39          const datos = await resTiempo.json();
40
41          // Busca la capital de la provincia en el array de ciudades (igual que en tiemponet.js)
42          const capital = datos.ciudades?.find(
43            c => c.name === datos.provincia?.CAPITAL_PROVINCIA
44          );
45
46          // Construye el contenido del popup
47          let popupContent = `<strong>${punto.provincia}</strong><br>`;
48          if (capital) {
49            popupContent += `
50              <strong>Capital:</strong> ${capital.name}<br>
51              <strong>Descripción:</strong> ${capital.stateSky?.description || 'No disponible'}<br>
52              <strong>Temp. Máxima:</strong> ${capital.temperatures?.max ?? 'N/A'}ºC<br>
53              <strong>Temp. Mínima:</strong> ${capital.temperatures?.min ?? 'N/A'}ºC
54            `;
55          } else {
56            popupContent += `<em>No se encontraron datos de la capital</em>`;
57          }
58
59          // Muestra el popup en el marcador
60          marker.bindPopup(popupContent).openPopup();
61        } catch (err) {
62          // Si ocurre un error, muestra mensaje de error en el popup
63          marker.bindPopup(`<strong>${punto.nombre}</strong><br><em>Error al cargar el tiempo</em>`).openPopup();
64        }
65      });
```