Módulo 5 – App de Clima (POO, ES6 y consumo de API)

1) Propósito
Reestructurar la lógica de la App de Clima utilizando Programación Orientada a Objetos (POO) y funcionalidades modernas de JavaScript ES6+, e implementar el consumo de una API de clima mediante programación asíncrona (fetch, promesas, async/await).

En esta iteración, el foco está en cómo está organizado el código JavaScript (clases, módulos lógicos, funciones reutilizables) y en la integración con una API externa para obtener datos reales o simulados de clima para tus lugares. El diseño visual puede reutilizar lo logrado en los módulos 3 y 4.

2) Objetivos de aprendizaje
- Aplicar los **conceptos fundamentales de POO** en JavaScript (clases, objetos, propiedades, métodos).
- Utilizar **funcionalidades ES6+**: let/const, arrow functions, parámetros por defecto, template literals y, cuando sea posible, módulos.
- Implementar **programación asíncrona** con promesas y async/await para consumir datos desde una API.
- Consumir una **API externa de clima** (o endpoint de prueba), procesar su respuesta JSON y actualizar el DOM.
- Gestionar el proyecto con **Git/GitHub** de forma ordenada (commits descriptivos, trabajo incremental, README).

3) Alcance
Sobre la base del Módulo 4 (datos modelados en JS y estadísticas semanales):
- La información de clima de los lugares **deja de estar fija en el código** y pasa a obtenerse (total o parcialmente) desde una **API de clima**.
- Se crea al menos una **clase principal**, por ejemplo WeatherApp o similar, que se encargue de:
  - Gestionar los lugares.
  - Pedir los datos de clima a la API.
  - Actualizar la interfaz (Home y Detalle).
- Se mantiene la sección de **Estadísticas de la semana** en la vista de detalle, pero ahora calculada a partir de los datos obtenidos de la API (cuando corresponda).
- Se agrega una sección sencilla de **"Alertas de clima"** en el detalle de un lugar, basada en reglas simples (por temperatura o cantidad de días de cierto tipo de clima).

4) Requisitos funcionales mínimos
- **Home**:
  - Mostrar un listado de **≥ 5 lugares** con clima actual (temperatura y estado) obtenido desde la API o combinando datos locales + API.
- **Detalle de lugar**:
  - Mostrar el **pronóstico de varios días** (lista o cards) obtenido desde la API (o simulado a partir de la respuesta).
  - Mantener la sección **"Estadísticas de la semana"**:
    - Temperatura mínima, máxima y promedio.
    - Cantidad de días de al menos 2 tipos de clima (ej.: soleado / lluvia).
  - Mostrar una sección **"Alertas de clima"** con al menos **1 regla simple**, por ejemplo:
    - Si el promedio de la semana > X °C → "Alerta de calor".
    - Si hay ≥ N días de lluvia → "Semana lluviosa".
- La navegación entre Home y Detalle puede seguir la misma lógica de módulos anteriores (no se exige SPA ni router aún).

5) Requisitos técnicos

POO y ES6+

- Implementar al menos **una clase principal**, por ejemplo:

```
class WeatherApp {
  constructor(apiClient) {
    this.apiClient = apiClient;
    this.lugares = [];
  }

  async cargarLugares() { /* ... */ }
  async cargarDetalleLugar(id) { /* ... */ }
  calcularEstadisticas(pronosticoSemanal) { /* ... */ }
}
```

- Se pueden definir clases adicionales si tiene sentido (ej.: LugarClima, ApiClient).
- Utilizar **ES6+** en el código:
  - let y const en lugar de var.
  - **Arrow functions** donde sean apropiadas.
  - **Parámetros por defecto** en funciones cuando tenga sentido.
  - **Template literals** para construir cadenas (por ejemplo, fragmentos de HTML o mensajes).

Programación asíncrona y consumo de API

- Utilizar **Fetch API** (o XHR) para obtener datos de clima desde una **API externa**.
- Manejar la respuesta asíncrona mediante:
  - Promesas (then/catch) y/o
  - async/await (recomendado).
- Procesar la respuesta JSON para:
  - Mapear los datos al formato interno de la app (arreglos/objetos).
  - Volver a usar la lógica de estadísticas del Módulo 4.
- Manejar al menos un caso de **error simple**:
  - Mostrar un mensaje en la interfaz si la API no responde o si hay un problema al cargar los da

DOM y actualización de la interfaz

- Utilizar el DOM para:
  - Renderizar dinámicamente el listado de lugares en Home, en base a los datos obtenidos de la API.
  - Renderizar el pronóstico y las estadísticas en la vista de detalle.
  - Mostrar/ocultar mensajes de “Cargando…” o “Error al cargar los datos”.

6) Entregables

- Proyecto comprimido en un único archivo **.zip** que contenga:
  - Archivos HTML.
  - Archivos de estilos (CSS/SASS) ya trabajados en módulos previos.
  - Archivos JavaScript con:
    - Clases y lógica de la App de Clima.
    - Código de consumo de API y cálculo de estadísticas.
  - Recursos adicionales (imágenes, íconos, etc.), si corresponde.

Archivo **README.md** con:

- Descripción breve de la App de Clima y su temática.
- Explicación de la **estructura de clases** (qué clase hace qué).
- Descripción de la **API de clima utilizada** (nombre, URL base o documentación).
- Resumen de cómo se calculan las estadísticas en esta versión.
- **Enlace al repositorio público de GitHub** del proyecto.

Rúbrica de evaluación

Criterio	Excelente (3 pts)	Adecuado (2 pts)	Básico (1 pt)	Insuficiente (0 pts)
POO y ES6+	Clase(s) clara(s), con métodos bien usados; let/const, arrow functions y template literals aplicados de forma consistente.	Hay clase(s) y varias features ES6 usadas, con algunos detalles de orden/estilo.	Se intenta usar clases/ES6, pero predomina código antiguo o desordenado.	No hay clases ni uso relevante de ES6.
Consumo de API y asincronía	API de clima integrada con fetch y async/await/promesas; muestra error en pantalla cuando falla.	API integrada, pero errores se manejan solo en consola o de forma mínima.	Intento de uso de API incompleto o inestable.	No usa API o nunca llega a funcionar.
Datos, estadísticas y alertas	Usa datos de la API para clima y pronóstico; calcula estadísticas y muestra al menos una alerta coherente.	Usa datos de API para clima/pronóstico; estadísticas o alertas funcionan a medias.	Solo muestra datos básicos de API; estadísticas/alertas casi no existen o fallan.	Sigue usando datos fijos o estadísticas/alertas no funcionan.
DOM e interfaz	Home y Detalle se generan dinámicamente desde JS con datos de API; se ve carga y/o error claramente.	Mayor parte de la interfaz viene de JS, pero con algunos elementos fijos o poco claros.	Poco DOM dinámico; mucha info fija en HTML, interfaz confusa.	No actualiza la interfaz con los datos de la API.
Git/GitHub y README	≥3 commits descriptivos; repo público; README explica clases, API usada y cómo ejecutar, con enlace al repo.	≥3 commits aceptables; repo público; README básico con enlace.	Pocos commits o mensajes genéricos; README muy breve.	Sin repo público/enlace; sin README o vacío.