CICLO FORMATIVO DE GRADO SUPERIOR

DESARROLLO DE APLICACIONES WEB (DAW)

MÓDULO:

Desarrollo entorno Cliente

Tarea REACT

ALUMNO/A:

Carlos Melenchón Gabarrón

PROFESOR/A:

Andrew Peter Nightingale

CENTRO EDUCATIVO:

CIFP CARLOS III

CURSO ACADÉMICO:

2025 – 2026

Índice

1. Descripción del proyecto.

Este proyecto consiste en el desarrollo de una aplicación web tipo SPA que permite consultar información meteorológica asociada a municipios y provincias de España utilizando la API pública de AEMET.

La aplicación está compuesta por dos partes diferencias:

Frontend: Desarrollado en React Vite, encargado de la interfaz de usuario y de la interacción con el usuario.

Backend: desarrollado con node.js y Express, que actúa como servidor intermedio y gestiona las peticiones hacia la API externa.

El principal objetivo del proyecto no es únicamente mostrar los datos meteorológicos, si no implementar correctamente una arquitectura cliente-servidor real, donde el cliente no accede directamente a la API externa.

2. Arquitectura de la aplicación

La aplicación como nos pide en el ejercicio sigue una modelo de arquitectura cliente-servidor desacoplado.

El frontend, desarrollado con react, se ejecuta en el navegador del usuario y actúa como cliente.

El backend, desarrollado con Node.js y Express, actúa como servidor intermedio encargado de procesar las solicitudes.

El cliente nunca accede directamente a la API de aemet. En su lugar, realiza peticiones HTTP al servidor propio, el cual se encarga de:
Recibir la petición → consultar los datos → procesarlos → devolver la respuesta en formato JSON.

Esta arquitectura permite mayor seguridad, control de datos y escalabilidad de la aplicación.

3. API REST implementada.

El backend implemente una pequeña API REST propia que es consumida por el frontend.

El servidor responde en formato JSON, permitiendo así que React pueda procesar la información y actualizar la interfaz sin recargar la página.

Búsqueda por municipio

GET /municipio?nombre={municipio}

Devuelve la información meteorológica asociada al municipio indicado.

Búsqueda por provincia

GET /provincia?nombre={provincia}

Devuelve la información de los municipios pertenecientes a la provincia seleccionada.

Al usar una API propia nos permite desacoplar completamente el cliente de la API externa.

4. Carga inicial de datos.

En el proyecto he diseñado un script llamado descargarMaestros.js cuya función es realizar la carga inicial de datos y está ubicado en el backend.

Este archivo actúa como proceso de inicialización del sistema. Obtiene información desde una fuente externa y almacena localmente en un archivo llamado municipios.json

Este fichero es una pequeña base de datos persistente que posteriormente es utilizada por el servidor Node.js para responder a las peticiones del cliente sin necesidad de consultar constantemente la API externa. De este modo, el servidor puede acceder rápidamente a la información y procesarla antes de enviarla al frontend.

Con este sistema se reduce el tiempo de respuesta, se evita posibles limitaciones o fallos de la api externa y la aplicación resulta más estable y predecible.

Por otro lado, en el frontend también se dispone de un conjunto de datos estáticos. En la carpeta data he creado un archivo provincias.js, que contiene el listado de todas las provincias de España. Este archivo se utiliza para rellenar los selectores y facilitar la búsqueda por provincias sin necesidad de realizar peticiones adicionales al servidor.

De esta forma podemos diferenciar dos tipos de información:

- Datos dinámicos: municipios resultados meteorológicos a obtenidos a través del backend.
- Datos estáticos: listado de provincias cargado directamente en el cliente.

He realizado uno de cada para ver la diferencia de tenerlo de una manera u otra.

5. Estructura del frontend

El frontend lo he desarrollado utilizando componentes funcionales de React, organizados según su responsabilidad.

- BuscadorMunicipio.jsx: Se encarga de recoger el texto introducido por el usuario y lanzar la petición al servidor.
- BuscadorProvincias.jsx: Permite seleccionar una provincia y genera la solicitud correspondiente al backend.
- ResultadoMunicipio.jsx: Muestra la información obtenida tras la búsqueda de un municipio.
- ResultadoProvincias.jsx: Renderiza la lista de municipios pertenecientes a la provincia seleccionada.

La separación en componentes facilita la reutilización del código y mejora el mantenimiento de la aplicación.

6. Funcionamiento.

Para comprender como interactúan las distintas parte de la aplicación, a continuación se describe el flujo completo que sigue una consulta desde que el usuario selecciona un dato en la interfaz hasta que se muestran los resultados en pantalla. Este proceso nos permite observar la comunicación entre el cliente (React) y el servidor (Node.js con express), así como el tratamiento de la información antes de ser presentada al usuario.

I. El usuario introduce un municipio o selecciona una provincia.
II. React captura el evento.
III. El frontend realiza una petición HTTP mediante fetch.
IV. El servidor Express recibe la solicitud.
V. El servidor consulta los datos o la API de AEMET.
VI. El servidor procesa la información.
VII. Devuelve una respuesta JSON.
VIII. React actualiza el estado mediante useState.
IX. Los componentes se renderizan automáticamente mostrando el resultado.

7. Gestión de estados.

La aplicación gestiona diferentes estados para mejorar la experiencia de usuario.

- Estado de carga: se muestra un mensaje mientras se espera la respuesta del servidor.
- Estado de error: Se informa si ocurre un problema en la petición.
- Sin resultados: se notifica cuando no se ha encontrado ningún dato.

8. Instrucciones de instalación y ejecución

8.1 Requisitos previos

Para poder ejecutar correctamente la aplicación desarrollada en este proyecto, es necesario disponer de una serie de herramientas básicas que permiten ejecutar el entorno de trabajo que he desarrollado durante este trabajo de React.

En primer lugar, es imprescindible contar con Node.js, ya que tanto el fronted como el backend de la aplicación se apoyan en este entorno para su funcionamiento. Node.js permite ejecutar código JavaScript fuera del navegador.

Junto a Node.js se utiliza npm, el gestor de paquetes que se incluye por defecto, y que se encarga de instalar y gestionar todas las dependencias necesarias del proyecto. Gracias a npm podemos tener automáticamente todas las librerías requeridas tanto en la parte cliente como en el entorno de servidor.

Para el desarrollo del proyecto he usado Visual Studio Code ya que se integra muy bien con JavaScript y React.

También necesitaremos disponer de una clave personal de acceso a la API de AEMET, la cual es necesaria para realizar las consultas meteorológicas. Esta clave se configura únicamente en nuestro backend mediante el archivo .env

8.2 Instalación del backend (Node + Express)

Para poner en marcha el backend del proyecto, primero se debe acceder a la carpeta correspondiente desde la terminal en mi caso es: “c:\aemet\react”

Una vez dentro, instalamos las dependencias necesarias ejecutando el siguiente comando: “npm install”.

A continuación, se crea un archivo .env en la raíz de nuestro backend donde se va almacenar la clave de acceso a la API de AEMET, con el siguiente contenido:

AEMET_API_KEY=TU_CLAVE_DE_AEMET donde TU_CLAVE_DE_AEMET será la clave que hemos solicitado que nos envíen a nuestro correo.

Este archivo nos permite mantener la clave de la API fuera del código y así evitamos que se exponga en el frontend.

Una vez configurado el archivo .env , el servidor se inicia ejecutando:
“node server.js”

Si el backend se ha iniciado correctamente, el servidor se quedará escuchando en el puerto configurado y estará preparado para recibir las peticiones que se realicen desde el fronted de la aplicación.

8.3 Instalación del frontend (React)

Para ejecutar el fronted de la aplicación, es necesario acceder a la carpeta correspondiente del proyecto React, en mi caso la carpeta está en c:\aemet\meteo-frontend

Una vez situados en dicha carpeta, se instalan las dependencias del proyecto ejecutando el siguiente comando: “npm install”

Este comando descarga todas las librerías necesarias para el funcionamiento de la aplicación Ract.

Tras completar la instalación, el fronted se inicia con el siguiente comando: “npm start”. Al ejecutarse correctamente, la aplicación se abrirá automáticamente en el navegador web, en mi caso con la dirección “http://localhost:3000” .

9. Decisiones técnicas

9.1 Separación entre frontend y backend.

El proyecto esta dividido en dos partes independientes: un fronten dessarrolado en react y un backend desarrollado con Node.js y Express.
Al tener separado el frontend del backend nos permite que el frontend no acceda directamente a la API de AEMET y que todas las peticiones pasen primero por el servidor.

De esta forma, se evita exponer la clave de API en el cliente y se cumplimos con el requisito de utilizar una arquitectura cliente-servidor correctamente.

9.2 Uso de express como servidor intermedio.

He utilizado express para crear un servidor sencillo que expone sus propias ruta consumidas por el fronted.

El backend se encarga de realizar las peticiones a la API de AEMET, procesar la respuesta y devolver al frontend únicamente los datos necesarios en formato JSON.

9.3 Uso de variables de entorno para la clave de la API.

La clave de acceso a la API de AEMET se almacena en un archivo .env, que es leído únicamente por el backend.

De este modo, la clave no aparece en el código y no es expuesta en el navegado.

9.4 Organización del frontend en componentes.

El frontend se ha estructurado en componentes funcionales de React, separando la lógica de búsqueda y la visualización de resultados.

Está organización me ha facilitado la lectura del código, mejora de su mantenimiento y me permite reutirlizar componentes si se deseo ampliar la aplicación como ha sido el caso ya que solo había hecho provincias y después he puesto municipios.

9.5 Uso de hooks para la gestión del estado

Para la gestión del estado de la aplicación se han utilizado hooks de React como useState y useEffect.
Gracias a estos hooks se controla el estado de los datos recibidos, los posibles errores y el estado de carga durante las peticiones al backend.

9.6 Gestión de estados de carga y error.

Se ha implementado una gestión básica de errores y estados de carga para mejorar la experiencia del usuario. Mientras se realizan las peticiones se muestra un mensaje de carga, y en caso de error se informa al usuario de forma clara.

9.7 Uso de archivos JSON locales y Javascript locales.

Para facilitar las búsquedas y validaciones, se han utilizado archivos JSON y Javascript locales con información de provincias y municipios.

Esto nos permite realizar y ofrecer búsquedas más controladas y reducir errores en las peticiones al backend.

9.8 Estructura y organización del proyecto

En todo momento se ha mantenido una estructura clara de carpetas, separando frontend y backend en directorios distintos y organizando los archivos de forma lógica.

10.Conclusiones

Este proyecto me ha permitido aprender a comprender el funcionamiento real de una aplicación web moderna basada en arquitectura cliente-servidor.

Lo primero que he tenido que hacer es trabajar el consumo de APIS, aprendiendo a solicitar información a un servicio externo y a tratar la respuesta recibida en formato JSON para poder mostrarla al usuario de una forma clara y comprensible.

También he aplicado el uso de la programación asíncrona, ya que las peticiones HTTP al servidor no siempre son inmediatas. Mediante el uso de promesas y de los hooks de React he gestionado la espera de la respuesta sin bloquear la ejecución de la aplicación, permitiendo que la interfaz continúe siendo interactiva mientras reciben todos los datos.

Tambien he trabajado la seguridad en la gestión de claves, evitando exponer la clave de acceso a la API de AEMET en el lado del cliente. Para ello la clave se almacena en variables de entorno dentro del servidor, garantizando que solo accede a ella el backend.

Tambien he entendido la importancia de usar un backend intermedio. El servidor propio actúa como intermediario entre el cliente y la API externa, permitiendo controlar las peticiones, procesar la información y proteger los datos sensibles, además de facilitar futuras ampliaciones de la aplicación.