Documentación del Proyecto: Aplicación Meteorológica

Alumno: Carlos Enguí García

Módulo: Desarrollo Web en Entorno Cliente (DWEC)

Curso: 2025-26

1. Introducción y Objetivos

El objetivo principal de este proyecto es desarrollar una aplicación web moderna (Single Page Application) utilizando **React** para el frontend y **Node.js con Express** para el backend. La finalidad es consumir datos reales de la API de **AEMET (Agencia Estatal de Meteorología)** de forma segura y eficiente.

Uno de los requisitos más importantes que he tenido en cuenta es la arquitectura de la aplicación. Para no exponer mi clave privada de la API de AEMET en el navegador del cliente (algo inseguro), he implementado un servidor intermedio (proxy) en Node.js. Así, la comunicación sigue este flujo:
`Cliente React -> Mi Servidor Express -> API AEMET` .

2. Arquitectura de Archivos y Componentes

A continuación, se detalla la estructura técnica del proyecto y la responsabilidad de cada archivo clave.

2.1 Frontend (`/frontend/src/`)

El frontend es una SPA construida con **React** y estilizada con **Tailwind CSS**.

A. Configuración y Entrada

- **`main.jsx`** : Punto de entrada de React. Monta la aplicación en el DOM y envuelve a `App` con los proveedores de contexto (`LanguageProvider` , `ThemeProvider`).
- **`App.jsx`** : Componente raíz. Define la estructura visual base (Layout) y orquesta la aplicación usando el hook `useWeather` . No contiene lógica compleja, solo visualización.

B. Componentes (`/components`)

- **`Search.jsx`** : Buscador de municipios con autocompletado y validación.
- **`WeatherCard.jsx`** : Muestra el clima **actual**. Usa sub-componentes como `StatCard` para métricas (humedad, viento, UV).
- **`HourlyForecast.jsx`** : Carrusel de predicción por horas (scroll horizontal).
- **`WeatherForecast.jsx`** : Lista vertical de predicción a 7 días.
- **`Header.jsx`** : Barra superior con selector de idioma y tema.

C. Custom Hooks (`/hooks`)

- **`useWeather.js`** : **[CEREBRO]** Centraliza la lógica de negocio. Gestiona estados (`loading` , `error` , `data`), llama al servicio API y expone métodos a la vista.

D. Servicios y Utilidades

- **`services/api.js`** : Cliente HTTP que conecta con nuestro Backend.
- **`services/dataAdapter.js`** : **[TRANSFORMADOR]** Convierte el JSON complejo de AEMET en objetos simples para la UI.
- **`utils/weatherUtils.jsx`** : Funciones puras para decidir iconos (`getWeatherIcon`) y formatear fechas.

2.2 Backend (`/backend/`)

Servidor **Node.js + Express** que actúa como Proxy.

- **`server.js`** : Configura Express, CORS y rutas.
- **`routes/weatherRoutes.js`** : Define endpoints públicos (`/api/municipios` , `/api/prediccion/...`).
- **`services/aemetService.js`** : Gestiona la comunicación segura con AEMET usando la API Key oculta en `.env` .

3. Detalles de Implementación Técnica

3.1 Flujo de Datos

1. **Inicio**: React carga y pide configuración.
2. **Búsqueda**: Usuario escribe -> `Search.jsx` pide lista a Backend.
3. **Selección**: `Search` notifica a `useWeather`.
4. **Petición**: `useWeather` pide datos al Proxy (Backend).
5. **Proxy**: Backend inyecta API Key y pide a AEMET.
6. **Respuesta**: Backend devuelve datos limpios (UTF-8) a Frontend.
7. **Render**: React pinta las tarjetas.

3.2 Configuración y Estilos (Tailwind CSS)

Se ha utilizado **Tailwind CSS** instalado vía `npm`.

1. **Configuración (`tailwind.config.js`)**: Configurado el `content` para escanear archivos `.jsx` (Tree Shaking).
2. **Estilos**: Inyectados mediante directivas `@tailwind` en `index.css`.
3. **Procesamiento**: Vite + PostCSS generan el CSS final optimizado.

3.3 Implementación del Tema (Oscuro/Claro)

Manejado mediante `ThemeContext.jsx`:

- Detecta preferencia del S.O. (`prefers-color-scheme`).
- Guarda preferencia en `localStorage`.
- Aplica clase `.dark` al `<html>`, activando estilos condicionales (`dark:bg-slate-800`).

3.4 Internacionalización (i18n)

Manejado mediante `LanguageContext.jsx`:

- Sistema ligero con diccionario JSON (`translations.js`).
- Función `t('clave')` disponible en toda la app vía hook `useLanguage()`.

3.5 Algoritmo de Búsqueda (`Search.jsx`)

Algoritmo de filtrado cliente que:

1. **Normaliza**: Elimina acentos y mayúsculas (`Málaga -> malaga`).
2. **Multi-campo**: Busca en Nombre, Provincia e ID.

3. **Ranking**: Prioriza coincidencias exactas y por inicio de palabra.

3.6 Creación del Custom Hook (`useWeather.js`)

Para limpiar el componente principal `App.jsx` y separar la lógica de la vista, se creó este hook personalizado siguiendo estos pasos:

1. **Encapsulamiento de Estado**: Se movieron todos los `useState` (`weatherData` , `loading` , `error` , etc.) desde `App.jsx` al hook.
2. **Lógica de Negocio**: Se trasladó la función `handleSelection` (que llama a la API y gestiona errores) dentro del hook.
3. **Exposición de API**: El hook retorna un objeto con solo lo necesario para la vista:

```
return {
  selectedMunicipality, // Estado del municipio actual
  weatherData,          // Datos de AEMET
  hourlyData,           // Datos por horas
  loading,              // Booleano de carga
  error,                // Mensaje de error si lo hay
  handleSelection,      // Función para cambiar de municipio
  clearError            // Función para limpiar errores
};
```

De esta forma, `App.jsx` pasa de tener 100 líneas de lógica compleja a simplemente llamar a `const { ... } = useWeather();` .

3.7 Tooltips y Accesibilidad

Se ha priorizado la accesibilidad y la sencillez en la implementación de tooltips:

1. **Nativo**: Se utiliza el atributo estándar `title` de HTML en los iconos y botones. Esto garantiza que todos los navegadores muestren la descripción al pasar el ratón sin necesidad de librerías JavaScript pesadas.
2. **Lectores de Pantalla**: Los iconos incluyen atributos `aria-label` o textos ocultos para asegurar que las personas que usan lectores de pantalla entiendan el significado de cada elemento visual (ej: "Humedad", "Viento").

4. Desafíos y Soluciones

Durante el desarrollo, se resolvieron varios retos:

- **Codificación (Charset)**: AEMET envía en `ISO-8859-1`. Se implementó decodificación a `UTF-8` en el Backend para corregir tildes.
- **Datos Horarios**: Se creó un adaptador (`dataAdapter.js`) para aplanar la estructura de AEMET y un componente con scroll para visualizar 24h sin saturar.
- **API Key Segura**: Se movió toda la petición a AEMET al backend, usando variables de entorno `.env`, para que la clave nunca llegue al navegador.

5. Instalación

1. **Backend**: `cd backend -> npm install ->` Renombrar `.env.example a .env -> npm run dev`.
2. **Frontend**: `cd frontend -> npm install -> npm run dev`.

6. Conclusiones

Este proyecto ha sido fundamental para comprender la arquitectura real de una aplicación web moderna.

Principales aprendizajes:

- **Seguridad y Arquitectura**: He entendido por qué no se debe llamar a APIs externas desde el cliente (exposición de claves, CORS) y cómo un **Backend Intermediario (Proxy)** soluciona estos problemas de raíz.
- **React Avanzado**: La creación del Hook personalizado `useWeather` me ha enseñado a separar la lógica de la interfaz, haciendo que los componentes sean mucho más limpios y mantenibles.
- **Manejo de Datos Reales**: Los datos de AEMET no venían "listos para usar" (problemas de codificación, estructuras anidadas complejas). Implementar **Adaptadores** (`dataAdapter.js`) ha sido clave para normalizar esta información antes de pintarla.
- **Experiencia de Usuario**: Detalles como la normalización en la búsqueda (ignorar tildes) o el feedback visual de carga mejoran drásticamente la percepción de calidad del software.

En conclusión, la aplicación cumple con todos los requisitos funcionales y técnicos, demostrando un dominio sólido de la pila React + Node.js.