



AE.U3 REALIZA EL TUTORIAL DE ANGULAR



Nombre y apellidos: Cristina Sandoval Laborde

Curso: 2º DAM

Asignatura: Optativa

Índice

1.Introducción y contextualización.....	3
2.Desarrollo	3
2.1.Configuración del Routing y providers	3
2.2.Data Binding	4
2.2.1. One-way binding (Unidireccional)	4
2.2.2. Two-way (Bidireccional).....	6
2.3. Directivas Estructurales	6
2.3.1 Iteración de elementos.....	6
2.3.2. Visualización condicional.....	6
2.3.3. Implementación de CRUD Estático.....	6
2.4. Integración con el Backend y servicios	9
2.4.1. Consumo del servicio en el componente	10
2.5 Configuración del entorno y servidor.....	12
2.5.1 Transformación a API REST (Modificación de 01-express.js).....	13
2.5.2 Configuración de CORS	13
2.5.3 Implementación del modelo y rutas de datos.....	13
2.5.4 Estado final de la integración	13
2.6 Desarrollo CRUD	14
2.6.2. Implementación de Operaciones CRUD (Frontend)	16
2.6.3. Sincronización y Detección de Cambios	18
2.7. Navegación y Experiencia de Usuario.....	20
3.Conclusión	21
4.Enlaces github.....	21

1. Introducción y contextualización

En esta actividad se aborda la transición hacia el desarrollo web moderno mediante Angular. El objetivo es consolidar el uso de arquitecturas basadas en componentes y la comunicación asíncrona con servicios Backend desarrollados en Node.js. Se exploran las ventajas de las aplicaciones de página única (SPA) para mejorar la experiencia de usuario y la eficiencia en la carga de datos.

2. Desarrollo

2.1. Configuración del Routing y providers

Como se menciona en el manual, el Router intercepta las acciones del usuario para mostrar u ocultar componentes sin recargar la página completa. Al centrar los proveedores en appConfig, me aseguro que toda la aplicación tenga acceso a las herramientas de navegación y conexión a datos externos desde el comienzo.

En el archivo app.config.ts

He utilizado el modelo de configuración de Angular (**Standalone**) mediante el objeto ApplicationConfig, eliminando así la necesidad de los antiguos **NgModules**, lo que simplifica la estructura del proyecto.

He comenzado añadiendo **provideRouter(routes)** que es el motor de navegación. Este proveedor es el que mapea las URL que el usuario introduce en la barra de direcciones o en los enlaces que hace clic para asociarlos a los componentes de la aplicación.

He añadido **provideHttpClient()** para permitir la comunicación con el servidor, permitiendo así las operaciones CRUD asíncronas mediante el protocolo HTTP.

También he añadido **ProvideClientHydration()** para mejorar el rendimiento de carga y la renderización con el cliente, al permitir que la aplicación sea más rápida al “hidratar” el contenido renderizado en el servidor.

```

OptativaPokemon > src > app > ts app.config.ts > ...
1 import { ApplicationConfig, provideBrowserGlobalErrorListeners } from '@angular/core';
2 import { provideRouter } from '@angular/router';
3 import { provideHttpClient } from '@angular/common/http';
4 import { routes } from './app.routes';
5 import { provideClientHydration, withEventReplay } from '@angular/platform-browser';
6
7 export const appConfig: ApplicationConfig = {
8   providers: [
9     provideBrowserGlobalErrorListeners(),
10    provideRouter(routes),
11    provideHttpClient(),
12    provideClientHydration(withEventReplay())
13  ]
14};
15

```

Una vez configurados los proveedores globales, he definido el diccionario de rutas en el archivo app.routes.ts. En este archivo es donde se mapea el camino con su componente correspondiente, permitiendo que Angular separa que vista carga cuando el usuario navega a una URL específica.

En el archivo app.routes.ts

```

OptativaPokemon > src > app > app.routes.ts > ...
1 import { Routes } from '@angular/router';
2 import { Pokemon } from './components/pokemon/pokemon';
3
4 export const routes: Routes = [
5   {path: 'pokemon', component: Pokemon, title:'Pokemon'}
6 ];
7

```

2.2.Data Binding

El data binding es el mecanismo fundamental de Angular que permite la comunicación o enlace de datos entre el archivo TypeScript del componente y su plantilla HTML.

Según el manual, hay dos formas de gestionar esta comunicación:

2.2.1. One-way binding (Unidireccional)

Es el envío de datos desde el código hacia la plantilla. Para esto he implementado dos tipos:

- Interpolación{{expresión}}: Que se utiliza para mostrar valores dinámicos. En este proyecto la he usado para pintar el nombre y el tipo de cada Pokémon en la vista.
- Property binding [target]=""expresión": Que sirve para controlar propiedades de elementos HTML.

Como se observa en la imagen siguiente he definido una interface PokemonData asegurando que los datos que fluyen por el binding tengan la estructura correcta(nombre,tipo y descripción).

En el archivo **pokemon.html** he usado la interpolación para mostrar los datos del array Pokémon en el navegador de forma dinámica.

```
OptativaPokemon > src > app > components > pokemon > pokemon.html > div.pokedex
1  <div class="pokedex">
2    <h1>Mis Pokémon</h1>
3
4    @for (p of listaPokemon; track p.nombre) {
5      <div class="pokemon-card">
6        <h2>{{ p.nombre }}</h2>
7        <p>Tipo: {{ p.tipo }}</p>
8        <p><i>{{ p.descripcion }}</i></p>
9      </div>
10     } @empty {
11       <p>No hay pokémon en la lista.</p>
12     }
13   </div>
```

En el archivo **pokemon.ts**

```
OptativaPokemon > src > app > components > pokemon > pokemon.ts > Pokemon
1  import { Component, OnInit, inject } from '@angular/core';
2  import { CommonModule } from '@angular/common';
3
4  // Definimos una interfaz
5  interface PokemonData {
6    nombre: string;
7    tipo: string;
8    descripcion: string;
9  }
10
11 @Component({
12   selector: 'app-pokemon',
13   standalone: true,
14   imports: [CommonModule],
15   templateUrl: './pokemon.html',
16   styleUrls: ['./pokemon.css'],
17 })
18 export class Pokemon implements OnInit {
19
20   listaPokemon: PokemonData[] = [];
21
22
23   ngOnInit(): void {
24     // Los datos cumplen con la interfaz PokemonData
25     this.listaPokemon = [
26       { nombre: "Caterpie", tipo: "Bicho", descripcion: "Es lamentable" },
27       { nombre: "Weedle", tipo: "Bicho", descripcion: "También es lamentable" },
28       { nombre: "Magikarp", tipo: "Agua", descripcion: "Qué cosa más mala" }
29     ];
30   }
31 }
32
```

2.2.2. Two-way (Bidireccional)

Permite que la comunicación fluya en ambas direcciones, es decir, los cambios en el código actualizan la vista y las interacciones del usuario en la vista actualizan el código.

Se utiliza la sintaxis “banana in a box” [(target)]=”expresión”

2.3. Directivas Estructurales

Las directivas estructurales son herramientas que permiten modificar la estructura del DOM basándose en los datos del componente.

2.3.1 Iteración de elementos

He utilizado la sintaxis moderna @for para recorrer el array listaPokemon.

He incluido la propiedad track p.nombre, que permite a Angular identificar de forma única cada elemento.

Por cada iteración, se genera una tarjeta que contiene la información del Pokémon y los botones de control.

2.3.2. Visualización condicional

En este proyecto he utilizado la potencia del bloque @empty, que actúa como condicional automático, si la lista de Pokémon no contiene información, Angular oculta el bucle y simplemente muestra un mensaje alterativo (“No hay Pokémon en la lista”).

Esto hace que la interfaz siempre proporcione feedback al usuario.

2.3.3. Implementación de CRUD Estático

Dentro de la estructura generada mediante el bucle @for, he integrado los elementos esenciales para las operaciones CRUD(Create, Read, Update, Delete), asegurando que cada elemento de la lista cuente con su propia interfaz de gestión.

- **Read:** Esta operación ya es funcional, se realiza mediante la interpolación de los datos del array listaPokemon dentro de cada tarjeta. Al iterar sobre la lista, Angular extrae propiedades de la interfaz y las muestra en el DOM de forma automática.
- **Update/Delete:** Ahora tengo la maquetación de los botones específicos para estas acciones dentro de cada tarjeta de pokemon.
 - **Estado actual:** En esta fase de maquetación y estructura, los botones son solo estéticos. Están correctamente posicionados y vinculados visualmente a cada registro, pero no tienen ningún tipo de lógica funcional por el momento.

- Fase posterior: La lógica de programación, el manejo de eventos de clic y la comunicación con los servicios para ejecutar cambios reales en la base de datos los implementare en el apartado Desarrollo CRUD Pokemon.

```

OptativaPokemon > src > app > components > pokemon > 📄 pokemon.html > 🌐 div.container.mt-5
1  <div class="container mt-5">
2    <h1 class="text-center mb-4 custom-title">Mis Pokémon</h1>
3
4    <div class="d-flex justify-content-end mb-4">
5      <button class="btn btn-lg btn-add-custom shadow" type="button">
6        + Agregar Pokémon
7      </button>
8    </div>
9
10   <div class="row g-4">
11     @for (p of listaPokemon; track p.nombre) {
12       <div class="col-12 col-md-6 col-lg-4">
13         <div class="card h-100 shadow-sm border-0 pokemon-card">
14           <div class="card-body p-4">
15             <h2 class="card-title h4 pokemon-name">{{ p.nombre }}</h2>
16             <p class="card-text"><strong>Tipo:</strong> {{ p.tipo }}</p>
17             <p class="card-text text-muted"><em>{{ p.descripcion }}</em></p>
18
19             <div class="d-flex gap-2 mt-4">
20               <button class="btn btn-actualizar flex-fill" type="button">Actualizar</button>
21               <button class="btn btn-eliminar flex-fill" type="button">Eliminar</button>
22             </div>
23           </div>
24         </div>
25       } @empty {
26         <div class="col-12 text-center">
27           <p class="alert alert-info">No hay pokémon en la lista.</p>
28         </div>
29       }
30     </div>
31   </div>
32 </div>

```

Estilos CSS

```
OptativaPokemon > src > app > components > pokemon > pokémon.css > .btn-eliminar
1  :host {
2    --purple-main: #d145d1;
3    --pink-accent: #e352e3;
4    --dark-bg: #212121;
5  }
6
7  .custom-title {
8    font-weight: bold;
9    color: #444;
10 }
11
12
13 .pokemon-card {
14   border-radius: 15px !important;
15   transition: transform 0.2s;
16 }
17
18 .pokemon-card:hover {
19   transform: translateY(-5px);
20 }
21
22 .pokemon-name {
23   color: var(--purple-main);
24   border-bottom: 1px solid #eee;
25   padding-bottom: 10px;
26 }
27
28 /* BOTONES CON COLORES */
29 .btn-add-custom {
30   background-color: #007bff;
31
32   color: white;
33   border-radius: 30px;
34   padding: 10px 25px;
35 }
36
37 .btn-actualizar {
38   background-color: var(--purple-main);
39   color: white;
40   font-weight: bold;
41 }
42
43 .btn-eliminar {
44   background-color: #c98bc9;
45   color: white;
46   font-weight: bold;
47 }
48
49 .btn:hover {
50   opacity: 0.9;
51   color: white;
52 }
```

[Ver mi Pokédex](#)

Mis Pokémon

+ Agregar Pokémon

Caterpie

Tipo: Bicho

Es lamentable

[Actualizar](#)

[Eliminar](#)

Weedle

Tipo: Bicho

También es lamentable

[Actualizar](#)

[Eliminar](#)

Magikarp

Tipo: Agua

Qué cosa más mala

[Actualizar](#)

[Eliminar](#)

2.4. Integración con el Backend y servicios

Con el siguiente comando creamos la carpeta service y los archivos pokemon.service.ts y poemon.service.spec.ts

```
Component update sent to client(s).
PS C:\Users\alumno\Desktop\2º Dam\Optativa\Optativa_pokemon\OptativaPokemon> ng generate service services/pokemon.service
CREATE src/app/services/pokemon.service.spec.ts (378 bytes)
CREATE src/app/services/pokemon.service.ts (127 bytes)
PS C:\Users\alumno\Desktop\2º Dam\Optativa\Optativa_pokemon\OptativaPokemon>
```

Ahora hay que sacar la interfaz de pokemon.ts y pasarla a pokemon.service.ts

```

OptativaPokemon > src > app > components > pokemon > ts pokemon.ts > ...
1 import { Component, OnInit, inject } from '@angular/core';
2 import { CommonModule } from '@angular/common';
3 import { PokemonService, PokemonData } from '../../../../../services/pokemon.service';
4
5 @Component({
6   selector: 'app-pokemon',
7   standalone: true,
8   imports: [CommonModule],
9   templateUrl: './pokemon.html',
10  styleUrls: ['./pokemon.css'],
11 })
12 export class Pokemon implements OnInit {
13   listaPokemon: PokemonData[] = [];
14   private pokemonService = inject(PokemonService);
15
16   ngOnInit(): void {
17
18     this.listaPokemon = [
19       { nombre: "Caterpie", tipo: "Bicho", descripcion: "Es lamentable" },
20       { nombre: "Weedle", tipo: "Bicho", descripcion: "También es lamentable" },
21       { nombre: "Magikarp", tipo: "Agua", descripcion: "Qué cosa más mala" }
22     ];
23   }
24 }

```

```

OptativaPokemon > src > app > services > ts pokemon.service.ts > PokemonService
1 import { Injectable, inject } from '@angular/core';
2 import { HttpClient } from '@angular/common/http';
3 import { Observable } from 'rxjs';
4
5
6 export interface PokemonData {
7   nombre: string;
8   tipo: string;
9   descripcion: string;
10 }
11
12 @Injectable({
13   providedIn: 'root'
14 })
15 export class PokemonService {
16   private http = inject(HttpClient);
17   private urlPokemon = 'http://localhost:4000/pokemon';
18
19   getPokemons(): Observable<PokemonData[]> {
20     return this.http.get<PokemonData[]>(this.urlPokemon);
21   }
22 }

```

2.4.1. Consumo del servicio en el componente

En este apartado se detalla la implementación lógica necesario para que el componente Pokemon deje de utilizar datos locales y pase a recibir información de una fuente externa.

Para conectar la vista con la base de datos, he implementado el patrón Observer mediante el método `.subscribe()`

He utilizado la función `inject(PokemonService)` para dar acceso al componente a los métodos de comunicación HTTP.

La petición se dispara automáticamente al cargar el componente, asegurando que el usuario vea la lista actualizada de inmediato gracias al uso de **ngOnInit**

Al recibir la respuesta del servidor, al variable **listaPokemon** se actualiza.

```
OptativaPokemon > src > app > components > pokemon > ts pokemon.ts > ↗ Pokemon
1 import { Component, OnInit, inject } from '@angular/core';
2 import { CommonModule } from '@angular/common';
3 import { PokemonService, PokemonData } from '../../../../../services/pokemon.service';
4
5 @Component({
6   selector: 'app-pokemon',
7   standalone: true,
8   imports: [CommonModule],
9   templateUrl: './pokemon.html',
10  styleUrls: ['./pokemon.css'],
11 })
12 export class Pokemon implements OnInit {
13   // 1Empezamos con la lista vacía
14   listaPokemon: PokemonData[] = [];
15
16   // Inyectamos el servicio
17   private pokemonService = inject(PokemonService);
18
19   ngOnInit(): void {
20     // Consumimos el servicio de forma asíncrona
21     this.pokemonService.getPokemons().subscribe({
22       next: (datos) => {
23         // Cuando lleguen los datos de MongoDB, actualizamos la lista
24         this.listaPokemon = datos;
25       },
26       error: (e) => {
27         console.error('Error al recuperar los Pokémon:', e);
28       }
29     });
30   }
31
32   agregarPokemon(): void {
33     console.log('Botón añadir pulsado: Lógica de creación preparada.');
34   }
35 }
36 }
```

Al probarlo podemos comprobar que funciona, porque todavía no me he conectado a la base de datos y por eso me sale el mensaje de “No hay Pokémon en la lista”.

Ver mi Pokédex

Mis Pokémon

+ Agregar Pokémon

No hay pokémon en la lista.

2.5 Configuración del entorno y servidor

Estoy utilizando Node.js con el framework Express, utilizando un proyecto anterior que ya teníamos de pokemon.

Dependencias principales: express,mongoose,dotenv

Puerto del servidor: Configurado en el puerto 4000

Variables de Entorno: Uso de un archivo .env para proteger las credenciales de conexión al clúster.

Actualización del archivo pokemon.ts en angular

```
OptativaPokemon > src > app > components > pokemon > ts pokemon.ts > ↵ Pokemon > ⌂ agregarPokemon
1 import { Component, OnInit, inject } from '@angular/core';
2 import { CommonModule } from '@angular/common';
3 import { PokemonService, PokemonData } from '../../../../../services/pokemon.service';
4
5 @Component({
6   selector: 'app-pokemon',
7   standalone: true,
8   imports: [CommonModule],
9   templateUrl: './pokemon.html',
10   styleUrls: ['./pokemon.css'],
11 })
12 export class Pokemon implements OnInit {
13   // Array para los datos de la base de datos
14   listaPokemon: PokemonData[] = [];
15
16   // Inyectamos el servicio
17   private pokemonService = inject(PokemonService);
18
19   ngOnInit(): void {
20     this.obtenerPokemons();
21   }
22
23   obtenerPokemons(): void {
24     this.pokemonService.getPokemons().subscribe({
25       next: (datos) => {
26         this.listaPokemon = datos;
27       },
28       error: (e) => {
29         console.error('Error al recuperar los Pokémon:', e);
30       }
31     });
32
33   }
34
35   agregarPokemon(): void {
36     // Objeto directo sin validación
37     const nuevoPokemon: PokemonData = {
38       nombre: "Pikachu",
39       tipo: "Eléctrico",
40       descripción: "Ratón eléctrico amarillo"
41     };
42
43     this.pokemonService.postPokemon(nuevoPokemon).subscribe({
44       next: (res) => {
45         console.log('Pokémon guardado:', res);
46         this.obtenerPokemons();
47       },
48       error: (e) => console.error('Error al guardar:', e)
49     });
50
51
52   }
53 }
```

Actualización del archivo pokemon.service

```
OptativaPokemon > src > app > services > ts pokemon.service.ts > Pokemonservice
1 import { Injectable, inject } from '@angular/core';
2 import { HttpClient } from '@angular/common/http';
3 import { Observable } from 'rxjs';
4
5 export interface PokemonData {
6   _id?: string;
7   nombre: string;
8   tipo: string;
9   descripcion: string;
10 }
11
12 @Injectable({
13   providedIn: 'root'
14 })
15 export class Pokemonservice {
16   private http = inject(HttpClient);
17   private urlPokemon = 'http://localhost:4000/pokemon';
18
19   // Obtener los Pokémon (Read)
20   getPokemons(): Observable<PokemonData[]> {
21     return this.http.get<PokemonData[]>(this.urlPokemon);
22   }
23
24
25   postPokemon(data: PokemonData): Observable<PokemonData> {
26
27     return this.http.post<PokemonData>(this.urlPokemon, data);
28   }
29 }
```

2.5.1 Transformación a API REST (Modificación de 01-express.js)

Para que Angular pueda procesar la información, he modificado la lógica del servidor en el archivo 01-express.js.

He sustituido el envío de vistas renderizadas (HTML) por el método res.json(), permitiendo que el servidor actúe como una API de datos pura.

Compatibilidad con Angular: Angular espera recibir objetos o arrays de datos para que sus directivas, como @for, puedan iterar sobre ellos en el frontend.

2.5.2 Configuración de CORS

Al tener el servidor de Angular en un puerto (4200) y el servidor de Node.js en otro (puerto 4000), he tenido que habilitar la política de intercambio de recursos de origen cruzado (CORS).

2.5.3 Implementación del modelo y rutas de datos

He vinculado el esquema de Mongoose definido en ejercicios anteriores con las rutas del servidor.

El servicio de Angular (pokemon.service.ts) realiza una petición GET a esta ruta, recibe el JSON y lo inyecta en el componente mediante la suscripción al Observable.

2.5.4 Estado final de la integración

El servidor se conecta al **cluster de Cristina** usando mongoose y las variables de .env.

Angular solicita los datos al cargar el componente mediante ngOnInit.

Si la base de datos devuelve un array vacío, el bloque @empty muestra el mensaje de feedback informativo: "No hay pokémon en la lista".

```

JS 01-express.js ● | .gitignore | .env | JS pokemon.js ● | JS rutas.js | JS play ⏎ ...
JS 01-express.js > ...
1  const express = require('express');
2  const bodyParser = require('body-parser');
3  const cors = require('cors');
4  const mongoose = require('mongoose');
5  require('dotenv').config();
6
7  const app = express();
8
9  // --- CONFIGURACIÓN DE MIDDLEWARES ---
10 app.use(cors());
11
12 // Body-parser para que Node entienda los JSON que envía Angular
13 app.use(bodyParser.urlencoded({ extended: false }));
14 app.use(bodyParser.json());
15
16 const port = process.env.PORT || 3000;
17
18 // Configuración de vistas
19 app.set('view engine', 'ejs');
20 app.use('views', express.static(__dirname + '/views'));
21 app.use(express.static(__dirname + '/public'));
22
23 // --- CONEXIÓN A MONGODB ---
24 const uri = `mongodb+srv://${process.env.USER}:${process.env.PASSWORD}@cristina.rxmgmup`;
25
26 mongoose.connect(uri)
27   .then(() => console.log('Conectado a MongoDB en el cluster de Cristina'))
28   .catch(e => console.log('Error de conexión: ', e));
29
30 // --- LLAMADAS A LAS RUTAS ---
31 app.use('/', require('./router/rutas'));
32 app.use('/pokemon', require('./router/pokemon')); // Esta es la que usa Angular
33
34 // --- RUTAS DE PRUEBA / VISTAS EJS ---
35 app.get('/pruebas', (req, res) => {
36   res.render('pruebas', { titulo: 'mi título dinámico', Descripcion: 'Esto es una descripción' });
37 });
38
39 app.get('/pruebas2', (req, res) => {
40   res.render('pruebas2', { titulo: 'mi título dinámico 2', Descripcion: 'Esto es una descripción' });
41 });
42

```

2.6 Desarrollo CRUD

2.6.1. Implementación de la API y Reglas de Negocio (Backend)

Para que el CRUD sea funcional y profesional, he configurado el servidor Express para que actúe como una API JSON. Una de las aportaciones adicionales más importantes es la **gestión automática del contador de IDs**:

- **Ruta de Borrado:** Se ha implementado en router/pokemon.js utilizando findByIdAndDelete.
- **Lógica de Reinicio:** Al eliminar un registro, el servidor ejecuta countDocuments() para verificar si la colección está vacía
- **Regla Especial:** Si el conteo es 0, el sistema está programado para reiniciar la secuencia de IDs al valor **001**, asegurando que la Pokédex siempre comience con un orden lógico tras una limpieza total

```

1  const express = require('express');
2  const router = express.Router();
3
4  const Pokemon = require('../models/pokemon');
5
6  // OBTENER TODOS LOS POKÉMON (GET /pokemon)
7  router.get('/', async (req, res) => {
8      try {
9          const arrayPokemonDB = await Pokemon.find();
10         // Devolvemos JSON para que Angular lo reciba bien
11         res.json(arrayPokemonDB);
12     } catch (error) {
13         console.error("Error en GET:", error);
14         res.status(500).json({ mensaje: 'Error al obtener datos' });
15     }
16 });
17
18 // CREAR UN POKÉMON (POST /pokemon)
19 router.post('/', async (req, res) => {
20     const body = req.body;
21     try {
22         const pokemonDB = new Pokemon(body);
23         await pokemonDB.save();
24         res.status(201).json(pokemonDB);
25     } catch (error) {
26         console.log('Error en POST:', error);
27         res.status(400).json({ mensaje: 'Error al guardar' });
28     }
29 });
30
31 // ELIMINAR Y ACTUALIZAR
32 router.delete('/:id', async (req, res) => {
33     try {
34         // Eliminamos el Pokémon
35         const pokemonDB = await Pokemon.findByIdAndDelete(req.params.id);
36
37         // Comprobamos cuántos quedan en la colección
38         const contador = await Pokemon.countDocuments();
39
40         // Si no queda ninguno, ejecutamos la regla especial de reinicio
41         if (contador === 0) {
42             console.log("Pokedex vacía. Reiniciando contador a 001...");
43         }
44     }
45
46     res.json({
47         estado: !!pokemonDB,
48         totalRestante: contador
49     });
50 } catch (error) {
51     res.status(500).send(error);
52 }
53 });
54
55 router.put('/:id', async (req, res) => {
56     try {
57         const pokemonDB = await Pokemon.findByIdAndUpdate(req.params.id, req.body, { new: true });
58         res.json({ estado: true, pokemon: pokemonDB });
59     } catch (error) {
60         res.status(500).send(error);
61     }
62 });
63
64 module.exports = router;

```

2.6.2. Implementación de Operaciones CRUD (Frontend)

Una vez establecidas las reglas en el servidor, he procedido a activar la lógica en Angular para que los botones, anteriormente estéticos, realicen cambios reales en la base de datos.

- **Create (Creación):** He desarrollado un componente específico (PokemonForm) que captura los datos mediante un formulario dinámico.
- **Update (Actualización):** He implementado una lógica de edición integral. Al pulsar "Actualizar", el sistema redirige al usuario a una nueva página donde se recuperan los datos actuales de MongoDB para ser modificados simultáneamente.
- **Delete (Borrado):** Se ha vinculado el botón de eliminación con una alerta de confirmación nativa. Tras la aprobación del usuario, se invoca el método DELETE del servicio, disparando la lógica de reinicio de IDs en el backend si la colección queda vacía.

```

OptativaPokemon > src > app > components > pokemon-form > pokémon-forms > PokemonForm
1  import { Component, OnInit, inject } from '@angular/core';
2  import { CommonModule } from '@angular/common';
3  import { FormsModule } from '@angular/forms';
4  import { ActivatedRoute, Router } from '@angular/router';
5  import { PokemonService, PokemonData } from '../../../../../services/pokemon.service';
6
7  @Component({
8    selector: 'app-pokemon-form',
9    standalone: true,
10   imports: [CommonModule, FormsModule],
11   templateUrl: './pokemon-form.html',
12 })
13 export class PokemonForm implements OnInit {
14   nuevoNombre: string = '';
15   nuevoTipo: string = '';
16   nuevaDescripcion: string = '';
17   idEditar: string | null = null; // Para saber si estamos editando
18
19   private pokemonService = inject(PokemonService);
20   private router = inject(Router);
21   private route = inject(ActivatedRoute); // Para leer el ID de la URL
22
23   ngOnInit(): void {
24     // Comprobamos si hay un ID en la URL
25     this.idEditar = this.route.snapshot.paramMap.get('id');
26
27     if (this.idEditar) {
28       // Si existe, pedimos los datos a Node.js para llenar el formulario
29       this.pokemonService.getPokemonById(this.idEditar).subscribe({
30         next: (p) => {
31           this.nuevoNombre = p.nombre;
32           this.nuevoTipo = p.tipo;
33           this.nuevaDescripcion = p.descripcion;
34         },
35         error: (e) => console.error('Error al cargar datos:', e)
36       });
37     }
38   }
39
40   guardar(): void {
41     const datos: PokemonData = {
42       nombre: this.nuevoNombre,
43       tipo: this.nuevoTipo,
44       descripcion: this.nuevaDescripcion
45     };
46
47
48     if (this.idEditar) {
49       // MODO EDICIÓN
50       this.pokemonService.actualizarPokemon(this.idEditar, datos).subscribe({
51         next: () => this.router.navigate(['/pokemon']),
52         error: (e) => console.error(e)
53       });
54     } else {
55       // MODO CREACIÓN
56       this.pokemonService.postPokemon(datos).subscribe({
57         next: () => this.router.navigate(['/pokemon']),
58         error: (e) => console.error(e)
59       });
60     }
61   }
62
63   cancelar(): void {
64     this.router.navigate(['/pokemon']);
65   }
66 }

```

```

OptativaPokemon > src > app > services > pokemon.service.ts > ...
2   import { HttpClient } from '@angular/common/http';
3   import { Observable } from 'rxjs';
4
5   export interface PokemonData {
6     _id?: string;
7     nombre: string;
8     tipo: string;
9     descripcion: string;
10    }
11
12  @Injectable({
13    providedIn: 'root'
14  })
15  export class PokemonService {
16    private http = inject(HttpClient);
17    private urlPokemon = 'http://localhost:4000/pokemon';
18
19    // Obtener los Pokémon (Read)
20    getPokemons(): Observable<PokemonData[]> {
21      return this.http.get<PokemonData[]>(this.urlPokemon);
22    }
23
24
25    postPokemon(data: PokemonData): Observable<PokemonData> {
26
27      return this.http.post<PokemonData>(this.urlPokemon, data);
28    }
29
30    // Obtener un solo registro para editar
31    getPokemonById(id: string): Observable<PokemonData> {
32      return this.http.get<PokemonData>(` ${this.urlPokemon}/${id}`);
33    }
34
35    // Actualizar registro completo
36    actualizarPokemon(id: string, data: PokemonData): Observable<any> {
37      return this.http.put(` ${this.urlPokemon}/${id}`, data);
38    }
39
40    eliminarPokemon(id: string): Observable<any> {
41      return this.http.delete(` ${this.urlPokemon}/${id}`);
42    }
43  }
44

```

2.6.3. Sincronización y Detección de Cambios

Para garantizar que la aplicación cargue los datos "del tirón" tras realizar una operación CRUD, he aplicado técnicas avanzadas de renderizado:

- **Refresco Automático:** Tras cada inserción, edición o borrado, el componente ejecuta una nueva suscripción al método getPokemons(), actualizando la galería morada de forma instantánea.
- **ChangeDetectorRef:** He integrado la detección de cambios manual para asegurar que, tras recibir el array de datos desde el puerto 4000, la interfaz de Angular se "hidrate" correctamente y elimine el estado de carga inicial.

```

OptativaPokemon > src > app > components > pokemon > pokemons > Pokemon > obtenerPokemons
1 import { Component, OnInit, inject, PLATFORM_ID, ChangeDetectorRef } from '@angular/core';
2 import { CommonModule, isPlatformBrowser } from '@angular/common'; // Importamos estas utilidades
3 import { Router } from '@angular/router';
4 import { PokemonService, PokemonData } from '../../../../../services/pokemon.service';
5
6 @Component({
7   selector: 'app-pokemon',
8   standalone: true,
9   imports: [CommonModule],
10  templateUrl: './pokemon.html',
11  styleUrls: ['./pokemon.css'],
12})
13 export class Pokemon implements OnInit {
14   listaPokemon: PokemonData[] = [];
15
16   // Inyectamos el ID de la plataforma para evitar fallos de carga inicial
17   private platformId = inject(PLATFORM_ID);
18   private pokemonService = inject(PokemonService);
19   private router = inject(Router);
20   private cd = inject(ChangeDetectorRef);
21
22   ngOnInit(): void {
23     if (isPlatformBrowser(this.platformId)) {
24       this.obtenerPokemons();
25     }
26   }
27
28   obtenerPokemons(): void {
29     this.pokemonService.getPokemons().subscribe({
30       next: (datos) => {
31         // 1. Forzamos una nueva referencia del array
32         this.listaPokemon = [...datos];
33         console.log(`¡Datos recibidos!`, this.listaPokemon);
34
35         // 2. OBLIGAMOS a la interfaz a refrescarse
36         this.cd.detectChanges();
37       },
38       error: (e) => console.error("Error:", e)
39     });
40   }
41
42   irAAgregar(): void {
43     this.router.navigate(['/nuevo']);
44   }
45
46   irAEstar(id: string | undefined): void {
47     if (id) this.router.navigate(['/editar', id]);
48   }
49
50   borrarPokemon(id: string | undefined): void {
51     if (id && confirm('¿Eliminar este Pokémon?')) {
52       this.pokemonService.eliminarPokemon(id).subscribe({
53         next: () => this.obtenerPokemons(),
54         error: (e) => console.error(e)
55       });
56     }
57   }
58 }

```

2.7. Navegación y Experiencia de Usuario

He transformado la arquitectura de una aplicación estática a una **Single Page Application (SPA)** con navegación por rutas:

- **Rutas Dinámicas:** En el archivo app.routes.ts, he definido caminos para el listado (/pokemon), la creación (/nuevo) y la edición parametrizada (/editar/:id).
- **Navegación Programática:** Utilizando el servicio Router, la aplicación redirige al usuario automáticamente a la lista principal una vez que el servidor confirma que el Pokémon ha sido guardado o actualizado con éxito.
- **Feedback Visual:** Se mantiene el uso del bloque @empty que, en combinación con el contador del backend, informa al usuario cuando la Pokédex ha sido reiniciada a sus valores por defecto.

```
OptativaPokemon > src > app > routes.ts > ...
1 import { Routes } from '@angular/router';
2 import { Pokemon } from './components/pokemon/pokemon';
3 import { PokemonForm } from './components/pokemon-form/pokemon-form';
4
5 export const routes: Routes = [
6   { path: '', redirectTo: 'pokemon', pathMatch: 'full' },
7   { path: 'pokemon', component: Pokemon },
8   { path: 'nuevo', component: PokemonForm },
9   { path: 'editar/:id', component: PokemonForm }
10];

```

```

OptativaPokemon > src > app > components > pokemon > pokemon.html > div.container.mt-5 > div.row.g-4
1  <div class="container mt-5">
2    <h1 class="text-center mb-4 custom-title">Mis Pokémon</h1>
3
4    <div class="d-flex justify-content-end mb-4">
5      <button (click)='irAAgregar()' class="btn btn-lg btn-add-custom shadow" type="button">
6        + Agregar Pokémon
7      </button>
8    </div>
9
10   <div class="row g-4">
11     @for (p of listaPokemon; track $index) {
12       <div class="col-12 col-md-6 col-lg-4">
13         <div class="card h-100 shadow-sm border-0 pokemon-card">
14           <div class="card-body p-4">
15             <h2 class="card-title h4 pokemon-name">{{ p.nombre }}</h2>
16             <p class="card-text"><strong>Tipo:</strong> {{ p.tipo }}</p>
17             <p class="card-text text-muted"><em>{{ p.descripcion }}</em></p>
18
19             <div class="d-flex gap-2 mt-4">
20               <button (click)='irAEstar(p._id)' class="btn btn-actualizar flex-fill" type="button">
21                 Actualizar
22               </button>
23               <button (click)='borrarPokemon(p._id)' class="btn btn-eliminar flex-fill" type="button">
24                 Eliminar
25               </button>
26             </div>
27           </div>
28         </div>
29       </div>
30     } @empty {
31       <div class="col-12 text-center py-5">
32         <div class="spinner-border text-purple" role="status">
33           <span class="visually-hidden">Cargando...</span>
34         </div>
35         <p class="mt-3 text-muted">Buscando Pokémon en la base de datos...</p>
36       </div>
37     }
38   </div>
39 </div>

```

3. Conclusión

La realización de este proyecto **me ha permitido consolidar** la integración de un ecosistema **Full-Stack** moderno, utilizando **Angular** como frontend y **Node.js** como backend.

A través de la implementación de componentes **Standalone** y el uso de **ApplicationConfig**, **he comprobado** que esta arquitectura supone una evolución mucho más eficiente respecto a los modelos tradicionales, ya que **me ha simplificado** enormemente la inyección de dependencias críticas, como el cliente HTTP y el enrutador. Asimismo, al desarrollar **mi propia API REST**, **he logrado** tener un control total sobre las reglas de negocio, destacando especialmente la gestión automatizada de IDs y la persistencia de datos en el clúster de **MongoDB Atlas**.

En definitiva, **he conseguido crear** una aplicación reactiva, escalable y con una experiencia de usuario fluida, cumpliendo con los estándares de calidad que exige el desarrollo web profesional hoy en día.

4. Enlaces github

<https://github.com/crisanlab94/2-Dam/tree/main/Optativa/03Express>

https://github.com/crisanlab94/2-Dam/tree/main/Optativa/Optativa_pokemon