Universidad de Jaén

TFT

Desarrollo de Aplicaciones mediante JAVA EE. Desarrollo Fullstack en Tecnología JAVA EE + Angular

Mena Expósito, Jose Carlos - 53596760-k

Índice

1. Introducción

Al igual que pasa con la sociedad, las tecnologías están en constante evolución con la intención o finalidad de mejorar los servicios y el valor añadido que estos aportan a los usuarios finales.

Hoy en día, son pocos los entornos que se encuentran aislados sin conexión a internet, ya que la mayoría de aplicaciones necesitan descargarse algún tipo de recurso desde internet para su correcto funcionamiento.

Una aplicación realizada en Angular y que conecte con una API (haciendo uso de REST) es un ejemplo de los tipos de aplicaciones más comunes que se desarrollan hoy en día.

El nombre de REST viene dado por **RE**presentational **S**tate **T**ransfer que en español significa Transferencia de Estado Representacional. Este tipo de arquitectura hace uso del estándar HTTP.

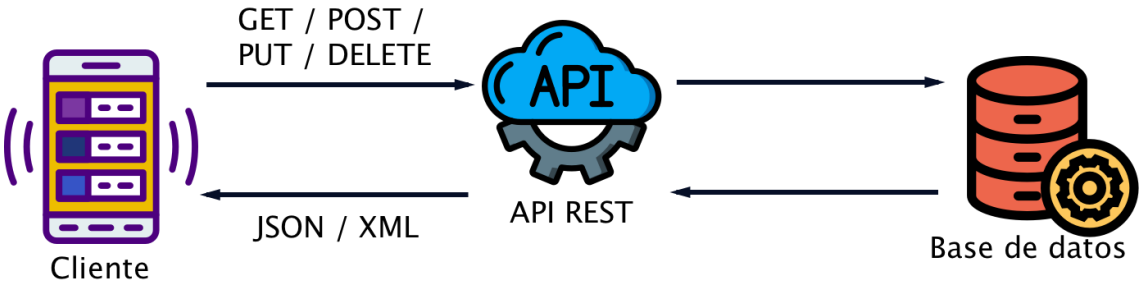

Ilustración 1: Esquema Cliente + API

Como se puede observar en la imagen anterior (ilustración 1.1), el cliente realiza una petición a la API REST haciendo uso de alguno de los métodos del protocolo HTTP (GET, POST, PUT o DELETE), a continuación la API REST se comunica con la base de datos, bien para obtener o para enviar nueva información a la misma. Esta información es enviada de nuevo al cliente normalmente en formato JSON o XML para que este la interprete y la muestre correctamente.

2. Objetivos

Digimon es un término tomado de “Digital Monster” literalmente en español “Monstruos Digitales”, estilizado como DIGIMON que es una franquicia de medios creado por Akiyoshi Hongo en 1997 a partir de una mascota virtual conocida como Digimon Virtual Pet.

Digimon incluye productos como mascotas virtuales, videojuegos, películas, animes, mangas, juguetes y juegos de cartas coleccionables.

Shadow Smith creó e implementó una API gratuita de Digimon para ofrecer todos los datos necesarios de los Digimon a aquellos desarrolladores que estén interesados.

El objetivo principal de este Trabajo Fin de Título (TFT), del Diploma Especializado en “Desarrollo de Aplicaciones mediante JAVA EE. Desarrollo Fullstack en Tecnología JAVA EE + Angular”, es desarrollar una aplicación utilizando el framework Angular para la parte del frontend y una API externa que funcione como backend, mostrando a su vez el proceso de desarrollo y las facilidades y oportunidades que ofrecen este tipo de desarrollos.

3. Funcionalidades

Las funcionalidades principales del sistema a desarrollar son las siguientes:

- Listar objetos.
- Ver información detallada de un objeto.
- Filtrar listado de objetos.

El código y modificaciones necesarias para el desarrollo de estas funcionalidades se mostrarán a continuación en el apartado X.

La estructura visual o de vistas de la aplicación es la mostrada en las siguientes ilustraciones de los storyboards correspondientes a cada apartado.

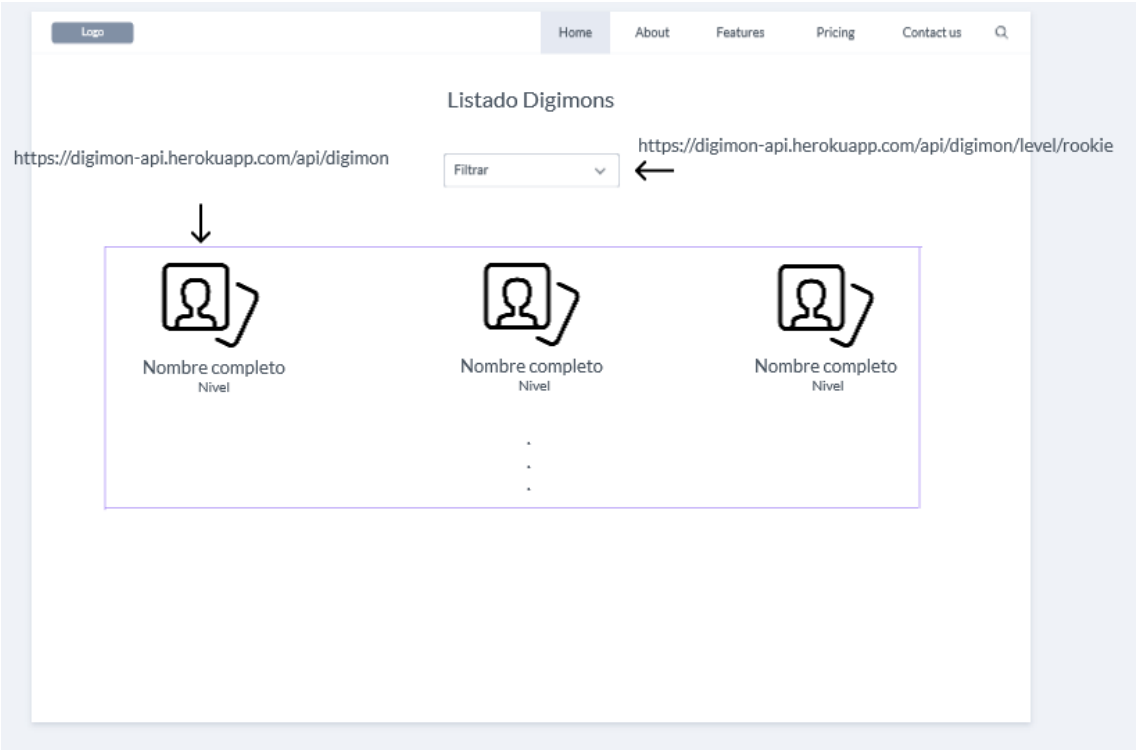

Ilustración 2: Storyboard listado

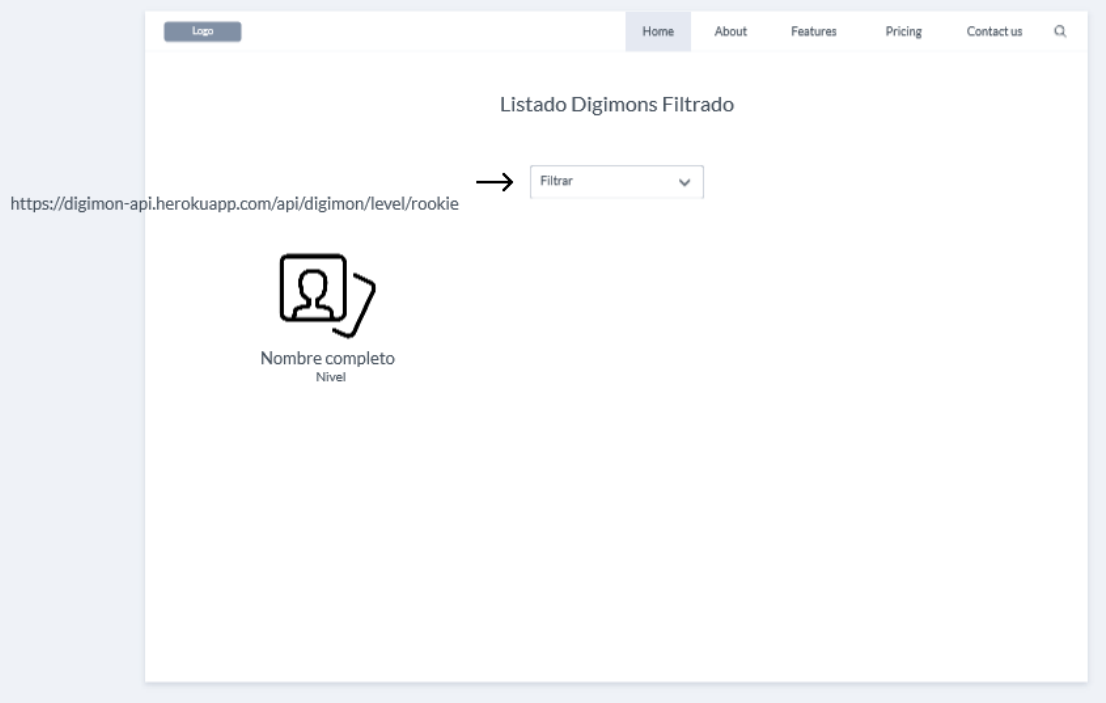

Ilustración 3: Storyboard filtrar listado

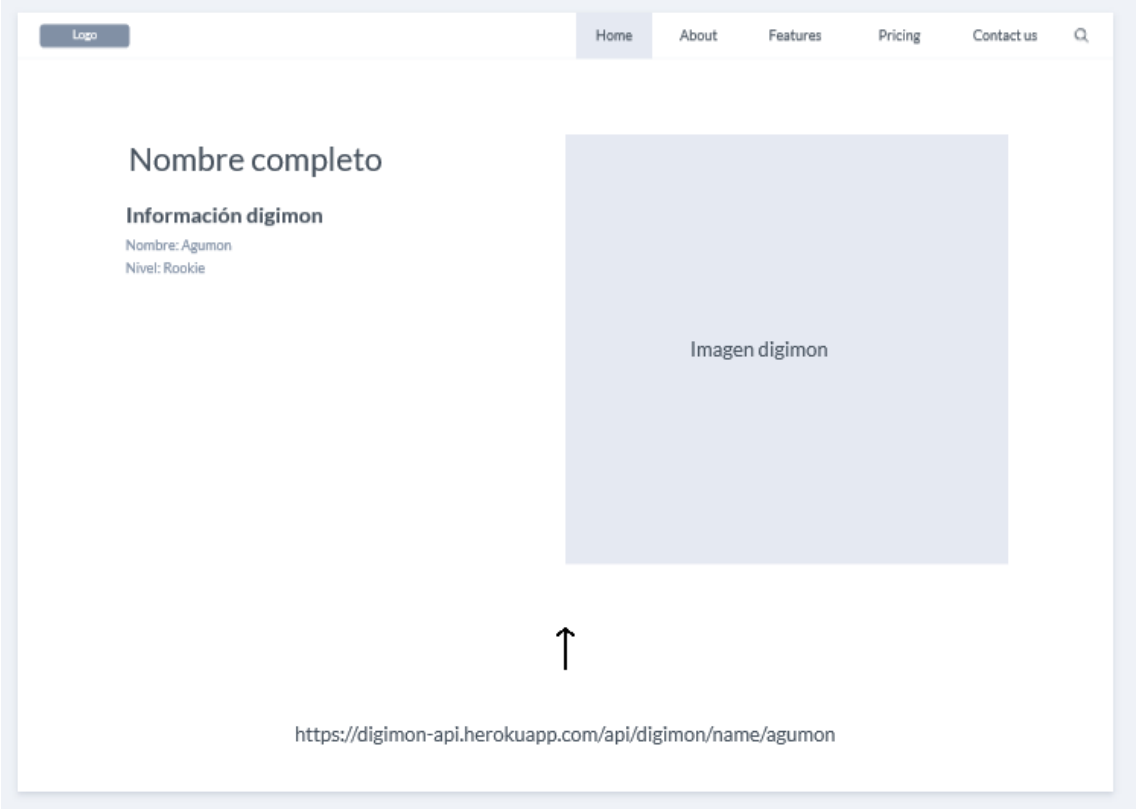

Ilustración 4: Storyboard información detallada

4. Requisitos básicos

Para poder realizar el proyecto necesitamos contar previamente con:

- **API**. Yo utilizaré una API que se encuentra disponible de forma gratuita en el siguiente enlace: https://digimon-api.herokuapp.com/.
- **Node** y **npm**. Será necesario instalar node para poder instalar Angular.
- **IDE**. En mi caso utilizaré Visual Studio Code.

5. Descripción tecnologías

a. Angular

Angular es una plataforma de desarrollo, construida sobre TypeScript. Como plataforma, Angular incluye:

- Un marco basado en componentes para crear aplicaciones web escalables.
- Una colección de bibliotecas bien integradas que cubren una amplia variedad de características, como son: enrutamiento, administración de formularios y comunicación cliente-servidor entre otras.
- Un conjunto de herramientas para desarrolladores que ayudan a desarrollar, compilar, probar y actualizar código.

Ilustración 5: Logo Angular

Angular permite desarrollar tanto proyectos de un solo desarrollador como aplicaciones de nivel empresarial. Está diseñado para que la actualización sea lo más sencilla posible y pueda aprovechar las ventajas de las últimas actualizaciones con un mínimo de esfuerzo.

Lo mejor de todo es que el ecosistema Angular consta de un grupo diverso de más de 1.7 millones de desarrolladores, autores de librerías y creadores de contenido.

Las aplicaciones desarrolladas con Angular cuentan con 3 partes esenciales dentro del proyecto: Componentes, Plantillas e Inyección de dependencias.

i. Componentes

Un componente en Angular es un bloque de código re-utilizable, que consta básicamente de 3 archivos: un CSS, un HTML (también conocido como plantilla o en inglés, template) y un archivo TypeScript. La carpeta app con la que viene Angular por defecto es un componente, aunque un tanto especial.

Como normal general todos los componentes de un proyecto de Angular deben ser subcarpetas de la carpeta principal app.

Para crear un componente dentro de nuestro proyecto debemos seguir los siguientes pasos:

1. Abrir una consola de comandos o terminal de Windows y acceder a la carpeta del proyecto y a su subcarpeta src/app.

```
cd D:/Proyectos/angular/src/app
```

Ilustración 6: Acceder a la carpeta app del proyecto

2. Generamos el componente de forma automática con el siguiente código:

```
ng generate component nombre_componente
```

Ilustración 7: Generar componente automáticamente

3. ¡Listo! El componente ya ha sido creado correctamente en tu proyecto Angular.

ii. Plantillas

Cada componente tiene una plantilla HTML que declara cómo se representa visualmente ese componente.

Angular extiende HTML con sintaxis adicional que le permite insertar valores dinámicos desde su componente. Actualiza automáticamente el DOM cuando cambia el estado de su componente. Una aplicación de esta función es la inserción de texto dinámico, como se muestra en el siguiente ejemplo.

```
<p>{{ message }}</p>
```

El valor del mensaje proviene de la clase de componente:

```
import { Component } from '@angular/core';

@Component ({
  selector: 'hello-world-interpolation',
  templateUrl: './hello-world-interpolation.component.html'
})
export class HelloWorldInterpolationComponent {
    message = 'Hello, World!';
}
```

Cuando la aplicación carga el componente y su plantilla, el usuario ve lo siguiente:

```
<p>Hello, World!</p>
```

iii. Inyección de dependencias

La inyección de dependencias le permite declarar las dependencias de sus clases de TypeScript sin ocuparse de su instanciación. Angular maneja la instanciación por nosotros lo que nos permite escribir código más comprobable y flexible.

Su uso es muy sencillo, solo es necesario agregar un “import” en las primeras líneas del componente tal y como se muestra a continuación:

```
import { Injectable } from '@angular/core';
```

b. API

El término API es una abreviatura de Application Programming Interfaces, que en español significa interfaz de programación de aplicaciones. Se trata de un conjunto de definiciones y protocolos que se utiliza para desarrollar e integrar el software de las aplicaciones, permitiendo la comunicación entre dos aplicaciones de software a través de un conjunto de reglas.

Así pues, podemos hablar de una API como una especificación formal que establece cómo un módulo de un software se comunica o interactúa con otro para cumplir una o muchas funciones. Todo dependiendo de las aplicaciones que las vayan a utilizar, y de los permisos que les dé el propietario de la API a los desarrolladores de terceros.

Ilustración 8: API

Una de las principales funciones de las API es poder facilitarle el trabajo a los desarrolladores y ahorrarles tiempo y dinero. Por ejemplo, si estás creando una aplicación que es una tienda online, no necesitarás crear desde cero un sistema de pagos u otro para verificar si hay stock disponible de un producto. Podrás utilizar la API de un servicio de pago ya existente, por ejemplo PayPal, y pedirle a tu distribuidor una API que te permita saber el stock que ellos tienen.

Con ello, no será necesario tener que reinventar la rueda con cada servicio que se crea, ya que podrás utilizar piezas o funciones que otros ya han creado. Imagínate que cada tienda online tuviera que tener su propio sistema de pago,

para los usuarios normales es mucho más cómodo poder hacerlo con los principales servicios que casi todos utilizan.

También son útiles para cuando lo único que se quiere es utilizar deliberadamente las funciones de determinado servicio para ofrecer ventajas a sus usuarios o atraer a los usuarios de ese servicio a que utilicen tu aplicación.

6. Aspectos de la implementación

a. Creación del proyecto

Lo primero que debemos hacer para crear el proyecto es acceder a la carpeta del espacio de trabajo y una vez allí ejecutar el siguiente comando de la ilustración 6 para crear el proyecto base con el nombre que queramos.

```
ng new nombre-aplicacion
```

Ilustración 9: Comando para crear un nuevo proyecto

Una vez ejecutado el comando correctamente tenemos un proyecto base tal y como se muestra en la ilustración 7.

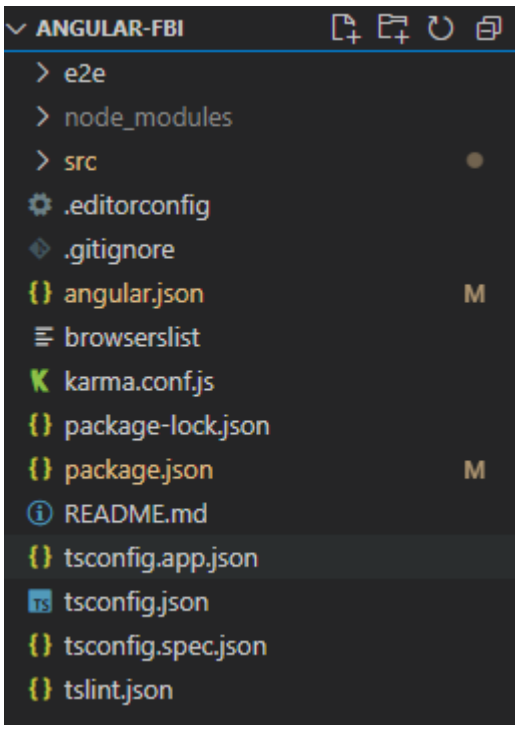

Ilustración 10: Proyecto base

b. Cargar listado de Digimons

La carga de todos los digimons del sistema se realiza en el método getDigimons() creado en el componente digimon-all (digimon-all.component.ts)

haciendo uso del servicio digimonService, el cuál realiza la petición GET a la API.

En primer lugar podemos observar el código del método getDigimons()

```
getDigimons() {
    this.digimonService.getDigimons().subscribe(
      result => {
        this.digimons = result;
      }, error => {
        console.log("Error carga componente get digimons");
      }
    );
  }
```

Este método se lanza en el constructor del componente tal y como se puede observar a continuación:

```
constructor(
    private digimonService: DigimonService,
  ) {
    this.getDigimons();
  }
```

El servicio digimonService realiza la petición GET a la API de la siguiente forma:

```
constructor(private http: HttpClient) {
    this.urlDigimons = "https://digimon-api.herokuapp.com/api/";
  }

  getDigimons(): Observable<any> {
    return this.http.get<any>(this.urlDigimons + "digimon");
  }
```

Por último, el código HTML necesario para mostrar el número total de resultados encontrados y listar los digimons devueltos por la API es el siguiente:

```
<div class="row">
    <p><b>{{ digimons.length }}</b> Resultados</p>
    <div *ngFor="let digimon of digimons" class="col-6 col-md-4 col-lg-3
thumbnail">
        <a class="link-unstyled" [href]="digimon.name">
            <img class="img-thumbnail" [src]="digimon.img" alt="digimon"
/>
            <div class="caption">
                <p class="link-dark text-center">{{ digimon.name }}</p>
            </div>
```

```
        </a>
    </div>
</div>
```

c. Obtener información de un Digimon

Cuando el usuario selecciona uno de los digimons que se muestran en el listado principal el usuario es redirigido a la url /nombreDigimon donde se mostrará la información del digimon junto a su imagen. Esto se consigue gracias al enrutado de urls.

> El Enrutamiento en Angular es la manera en la que navegamos entre las vistas de nuestra aplicación, en una web normal nosotros navegamos entre páginas HTML, pero en Angular navegamos entre las vistas que hemos generado a base de módulos y componentes.

Para agregar el módulo que se encargar del enrutado a un proyecto de Angular es necesario ejecutar el comando mostrado en la ilustración 8 en la consola:

```
ng generate module app-routing --flat --module=app
```

Ilustración 11: Agregar módulo routing a proyecto Angular

Una vez ejecutado correctamente el módulo abrimos el archivo app-routing.module.ts y agregamos el siguiente código:

```
const routes: Routes = [
  {path: '', component: DigimonAllComponent},
  {path: ':name', component: DigimonComponent},
  {path: '**', pathMatch: 'full', redirectTo: ''}
];
```

que se encargará de cargar el componente DigimonComponent para mostrar la información de un digimon cada vez que accedamos al perfil de uno de ellos.

También será necesario crear el componente Digimon tal y como se ha mostrado en el apartado “5.a.i Componentes”. Una vez creado el componente se agrega el siguiente código el cuál cargará los datos del digimon haciendo uso del servicio digimonService.

```
constructor(
    private activatedRoute: ActivatedRoute,
    private digimonService: DigimonService
  ) {
    this.digimonService.getDigimon(this.activatedRoute.snapshot.params['n
ame']).subscribe(
      result => {
        this.digimons = result;
      }, error => {
        console.log("Error carga componente digimon");
```

```
      }
    );
  }
```

También será necesario agregar el correspondiente código html a la vista del perfil del digimon seleccionado:

```
<div *ngIf="digimons">
    <div class="row mt-4 justify-content-center" *ngFor="let digimon of
digimons">
        <div class="ps-5 col-7 col-lg-4">
            <h1>{{ digimon.name }}</h1>
            <p>Nivel: {{ digimon.level }}</p>
        </div>
        <img class="col-5 img-thumbnail" [src]="digimon.img"
[alt]="digimon.name" />
    </div>
</div>
```

Incluso se puede agregar una opción para que el usuario pueda volver a la página de inicio siempre que lo deseé, lo que se conseguiría agregando lo siguiente:

```
<a class="link-dark link-unstyled" href="/">< Volver</a>
```

Una vez realizadas todas estas modificaciones si accedemos al perfil de un digimon obtendremos una vista como la de la ilustración 9.

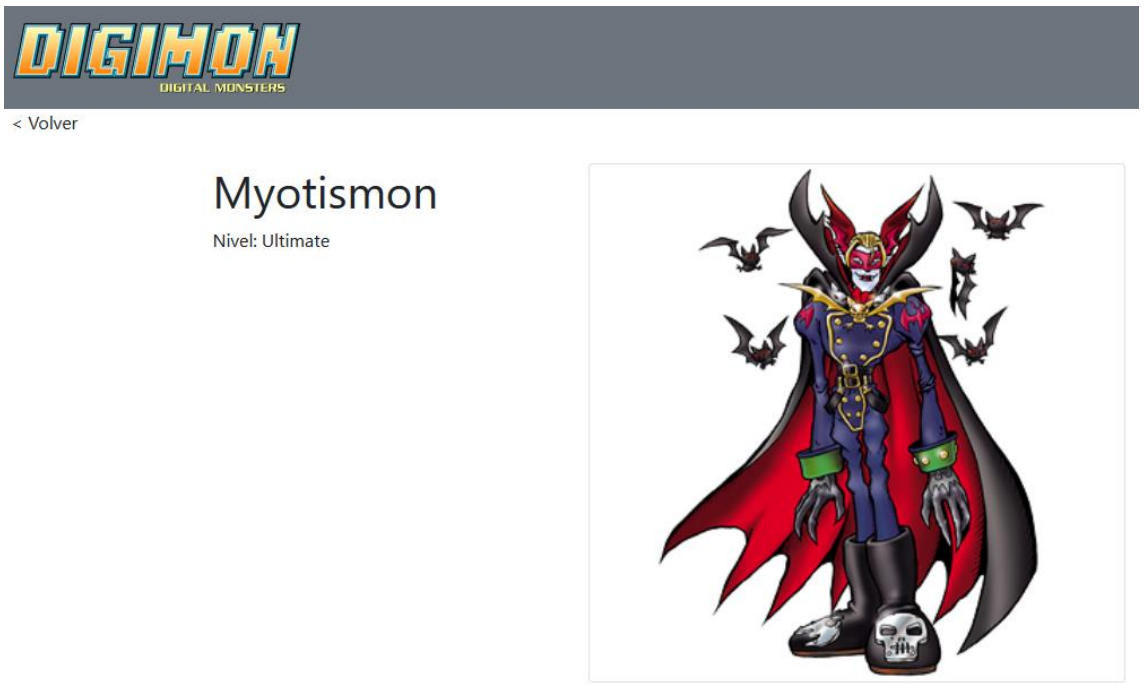

Ilustración 12: Página de información sobre un digimon

d. Filtrar por tipo de Digimon

Para poder filtrar por el tipo de digimon es necesario crear un nuevo método en el componente digimon-all (digimon-all.component.ts), ya que en él se cargan todos los digimons de la API.

El código del método necesario es el siguiente:

```
filtrarDigimons(level: String) {
    this.digimonService.filtrarDigimons(level).subscribe(
      result => {
        this.digimons = result;
      }, error => {
        console.log("Error carga componente filtrar digimons");
      }
    );
  }
```

Y al igual que para cargar el listado de digimons es necesario utilizar un servicio que se comunique con la API. En este caso el método del servicio para filtrar es el siguiente:

```
filtrarDigimons(level: String): Observable<any> {
    return this.http.get<any>(this.urlDigimons + "digimon/level/" +
level);
  }
```

Y se llamará a este método siempre y cuando el usuario pulse en uno de los checkbox que se muestran en la página principal de la aplicación. Para agregar los checkbox al código html (digimon-all.component.html) hay que agregar el siguiente código:

```
<span>Filtrar por: </span>

    <div class="form-check form-check-inline">
        <input class="form-check-input" type="radio"
name="flexRadioDefault" id="flexRadioDefault1"
            (click)="getDigimons()">
        <label class="form-check-label"
for="flexRadioDefault1">Ninguno</label>
    </div>
    <div class="form-check form-check-inline">
        <input class="form-check-input" type="radio"
name="flexRadioDefault" id="flexRadioDefault1"
            (click)="filtrarDigimons('Fresh')">
        <label class="form-check-label"
for="flexRadioDefault1">Fresh</label>
    </div>
```

```
    <div class="form-check form-check-inline">
        <input class="form-check-input" type="radio"
name="flexRadioDefault" id="flexRadioDefault1"
            (click)="filtrarDigimons('In training')">
        <label class="form-check-label" for="flexRadioDefault1">In
training</label>
    </div>
    <div class="form-check form-check-inline">
        <input class="form-check-input" type="radio"
name="flexRadioDefault" id="flexRadioDefault1"
            (click)="filtrarDigimons('Rookie')">
        <label class="form-check-label"
for="flexRadioDefault1">Rookie</label>
    </div>
    <div class="form-check form-check-inline">
        <input class="form-check-input" type="radio"
name="flexRadioDefault" id="flexRadioDefault1"
            (click)="filtrarDigimons('Champion')">
        <label class="form-check-label"
for="flexRadioDefault1">Champion</label>
    </div>
    <div class="form-check form-check-inline">
        <input class="form-check-input" type="radio"
name="flexRadioDefault" id="flexRadioDefault1"
            (click)="filtrarDigimons('Armor')">
        <label class="form-check-label"
for="flexRadioDefault1">Armor</label>
    </div>
    <div class="form-check form-check-inline">
        <input class="form-check-input" type="radio"
name="flexRadioDefault" id="flexRadioDefault1"
            (click)="filtrarDigimons('Ultimate')">
        <label class="form-check-label"
for="flexRadioDefault1">Ultimate</label>
    </div>
    <div class="form-check form-check-inline">
        <input class="form-check-input" type="radio"
name="flexRadioDefault" id="flexRadioDefault1"
            (click)="filtrarDigimons('Mega')">
        <label class="form-check-label"
for="flexRadioDefault1">Mega</label>
    </div>
```

Una vez hecho esto comprobamos que el filtro de la aplicación funciona correctamente como se puede ver en el ejemplo de la ilustración 10.

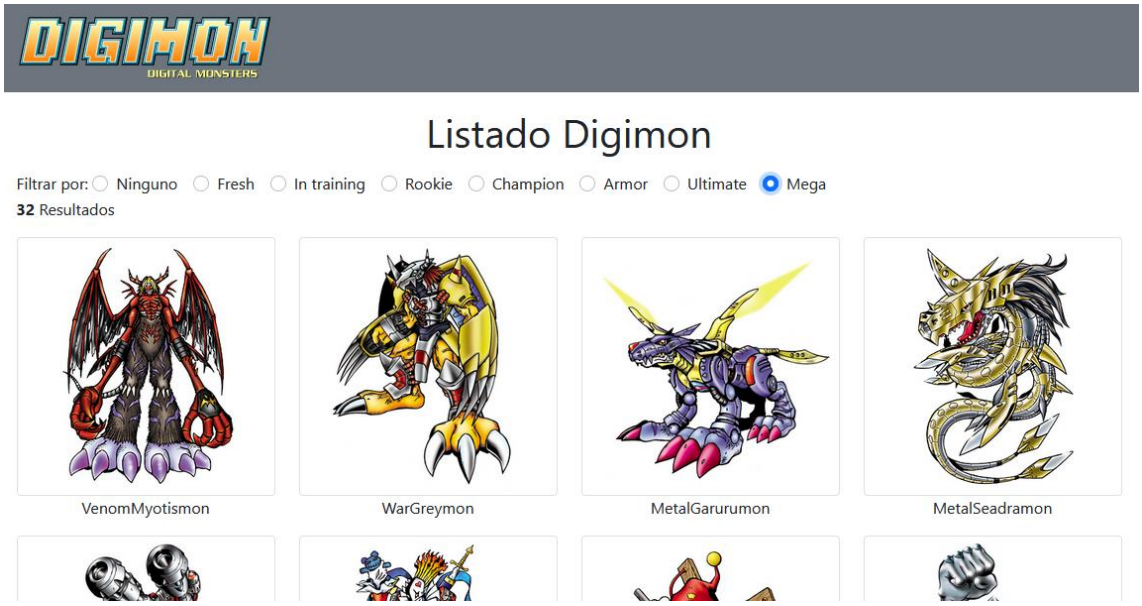

Ilustración 13: Filtrado de digimons

7. Cómo lanzar la aplicación

En primer lugar debemos descargarnos la aplicación del repositorio https://github.com/jcme0003/AngularFBI.git. Para ello podemos acceder directamente al enlace y bajarnos el código o utilizar algún cliente git (como SmartGit) que nos ayude a sincronizar nuestro equipo con el repositorio.

Una vez descargado es el momento de lanzarlo haciendo uso del comando: ng serve –open. Este comando lanzará la aplicación y la abrirá en el navegador correspondiente.

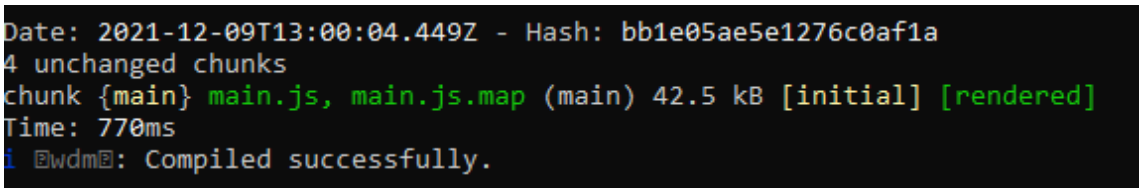

Ilustración 14: Resultado proyecto lanzado correctamente

Si no ha habido ningún error durante el proceso la consola de comando mostrará la salida que se indica en la ilustración 11 y se mostrará la aplicación en el navegador de la misma forma que se muestra en la siguiente ilustración 12.

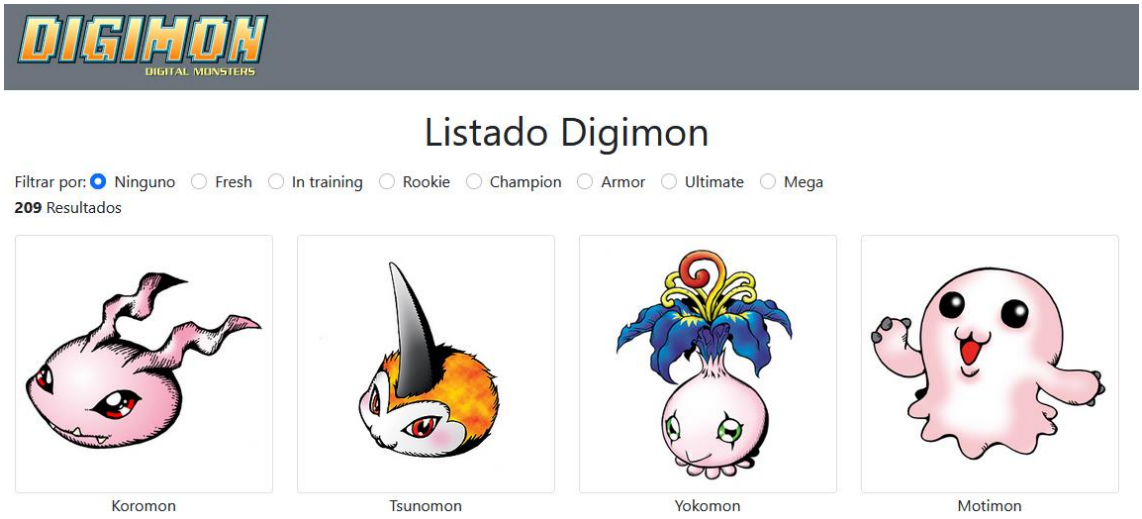

Ilustración 15: Aplicación lanzada

8. Conclusión

A lo largo de este TFT se ha pasado por las diferentes etapas de un proyecto Angular, comenzando por la preparación de los requisitos del mismo hasta llegar a tener una aplicación Angular conectada con un API.

Durante este proceso se ha podido comprobar lo fácil que es realizar una aplicación con Angular siempre y cuando tengas un backend robusto y sin errores, como es nuestro caso (API).

Partiendo de una API pública que provee al frontend de los datos y recursos necesarios se ha realizado una web que permite visualizar los datos de un elemento e incluso filtrar por un tipo de dato.

En conclusión, podemos afirmar que Angular es un Framework de JavaScript que nos permite realizar una aplicación web de forma ágil y sencilla haciendo uso de un backend ya creado.