

# Taller de Desarrollo Web

## SPORTIFY



Docentes a cargo:

- Zunino Alejandro
- Teyseyre Alfredo

Integrantes:

- Murillo Clara
- Bertoldi Felipe
- Cotti Fernando

Universidad Nacional del Centro de la Provincia de Buenos Aires

Facultad de Ciencias Exactas

<b>Desarrollo.....</b>	<b>3</b>
API Rest.....	3
Base de datos.....	6
DELETE.....	9
PUT.....	9
GET.....	9
POST.....	10
Angular.....	10
Consumo de la API REST.....	11
Product-form.....	11
Product-list.....	12
Vista final.....	13
Conclusiones.....	14

# Desarrollo

## API Rest

Para empezar con la inicialización del proyecto, primero instalamos todas las dependencias y tecnologías que íbamos a utilizar. En nuestro caso, primero realizamos la instalación de Node.js para empezar a construir la API Rest.

Antes de empezar con la instalación de las dependencias y módulos que utilizaremos, empezamos instalando TypeScript, que extiende la sintaxis de JavaScript, añadiendo características adicionales a JavaScript sin reemplazarlo por completo. Pero al utilizarlo, deberemos ir traduciendo el código TypeScript a JavaScript, y lo haremos a través del comando `tsc -w`, utilizando además Nodemon, que estará vigilando el código cada vez que guardemos para ir traduciendo lo que escribamos en TypeScript a JavaScript:

```
"scripts": {  
  "build": "tsc -w",  
  "dev": "nodemon build/index.js"  
},
```

Con `npm run build` y `npm run dev` iremos vigilando el código en TypeScript para ir traduciéndolo a JavaScript

También agregamos algunos módulos que nos iban a ser de utilidad para implementar y agregar determinadas funcionalidades a la aplicación. Para lo que teníamos pensado desarrollar, y luego de instalar Node.js, instalamos **Express**, **Morgan** y **Cors**, que son distintos módulos de Node.js que nos permiten realizar determinadas acciones.

**Express:** Lo utilizamos para configurar servidores y rutas, manejar solicitudes y respuestas HTTP, y para que nos facilite en general el desarrollo de aplicaciones web del lado de los endpoints.

**Morgan:** Es un middleware para registrar eventos en Node.js. Se integra con Express y registra las solicitudes HTTP que se hacen al servidor, donde podemos observar información útil como el método HTTP, el código de estado, et, lo que nos permite un debug mucho más sencillo y rápido.

**Cors:** Es otro middleware de Express que permite controlar el acceso a recursos del servidor desde diferentes orígenes de dominio en navegadores web. Lo utilizamos simplemente para que nos ayude a manejar las solicitudes de recursos entre distintos dominios, evitando errores de seguridad.

Posterior a estas instalaciones, empezaremos a crear la aplicación, a partir del servidor, dentro de nuestra carpeta `src`:

```
import express , { Application } from 'express';
import morgan from 'morgan';
import cors from 'cors';

import indexRoutes from './routes/indexRoutes';
import productsRoutes from './routes/productsRoutes';
```

Importamos los módulos que mencionamos anteriormente para gestionar los endpoints y registrar todas las solicitudes. También importamos otros archivos que representarán páginas en nuestra aplicación, donde estará la página de inicio y donde estarán todos los productos

```
class Server {

  public app: Application;

  constructor(){
    this.app = express();
    this.config();
    this.routes();
  }

  config(): void{
    this.app.set('port', 3000 || process.env.PORT);
    this.app.use(morgan('dev'));
    this.app.use(cors());
    this.app.use(express.json());
    this.app.use(express.urlencoded({extended:false}));
  }

  routes(): void{
    this.app.use(indexRoutes);
    this.app.use('/api/allProducts',productsRoutes);
  }

  start(): void{
    this.app.listen(this.app.get('port'), () =>{
      console.log('Server on port ', this.app.get('port'));
    } );
  }
}

const server = new Server();
server.start();
```

Con estas líneas de código, lo que hacemos es establecer la base de nuestro servidor que estará escuchando solicitudes de los usuarios que accedan a nuestra aplicación web, especificando que el servidor estará escuchando desde el puerto 3000, o el puerto por defecto que tenga el sistema.

Ahora, lo que nos hacen falta son los controladores de las páginas que vamos a implementar. En nuestro caso, vamos a tener *indexController* (para la página de inicio) y *productsController* (para la página donde estarán todos los productos). Pero antes, creamos una carpeta “*database*” para empezar a crear la base de datos y establecer qué se va a devolver a la hora de que se realicen solicitudes a la API.

## Base de datos

Entonces, dentro de la carpeta “**database**”, vamos a tener el código para la creación de la base de datos y de la tabla con toda la información de los productos, en un archivo *database.sql*:

```
CREATE DATABASE ng_sportify_db;

USE ng_sportify_db;

CREATE TABLE product(
  id INT(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,
  productName VARCHAR(180),
  description VARCHAR(255),
  image VARCHAR(200),
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

DESCRIBE product;
```

Luego, este código lo ejecutaremos con el comando *mysql -u root -p "password"*. Una vez que ingresemos la contraseña, copiaremos la creación de la base de datos y la tabla dentro de la consola de MySQL.

Para validar la conexión, debemos tener el usuario y la contraseña establecidos en algún lado, en nuestra caso, en un archivo *keys.ts*:

```
export default{
  database: {
    host: 'localhost',
    user: 'root',
    password: '!',
    database: 'ng_sportify_db',
  }
}
```

Al que debemos exportar para luego establecer la conexión desde el controlador que tendremos de los productos, *productsController*, de esta manera:

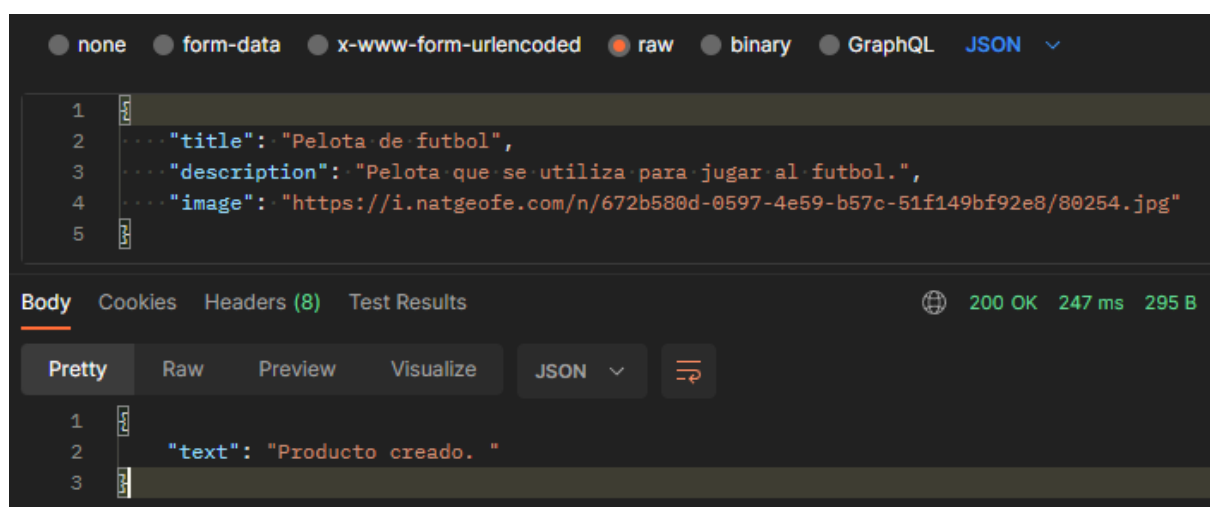
```
import { Request, Response } from 'express';
import db from '../database';

class ProductsController {

  public products(req: Request, res:Response){
    db.query('DESCRIBE product');
    res.json('Acá estarán los datos de todos los productos.');
```

Para testear el endpoint de *post*, que sirve para cargar nuevos productos en la página web, utilizamos **Postman**, que nos permite verificar la funcionalidad de cada uno de los endpoints. En este caso, mandamos un archivo json crudo para probar el funcionamiento de la carga de datos. En el siguiente código, devolvemos un mensaje por pantalla con el producto que se cargó con todos sus datos, y un mensaje como respuesta notificando al usuario que el producto fue creado con éxito.

```
public async createProduct(req: Request, res:Response){
  console.log(req.body);
  res.json({text: 'Producto creado. '})
}
```

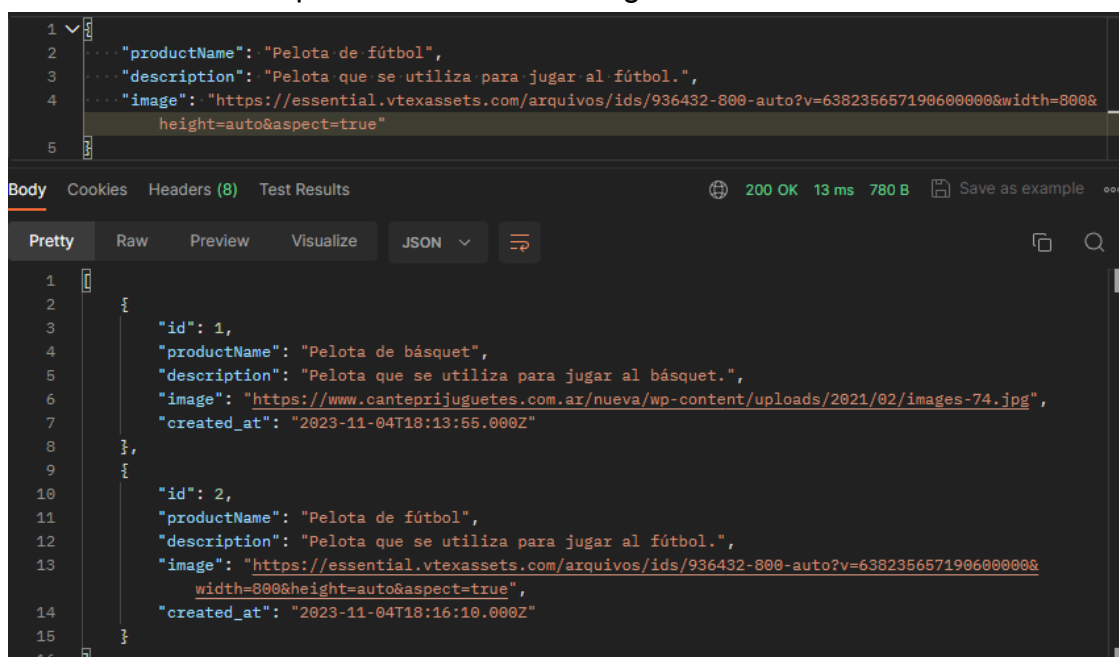


Enviamos el producto con sus datos a través de **Postman**, para testear el endpoint de *post*.

Y por la consola se muestra el objeto con los datos que se cargaron:

```
[nodemon] starting `node build/index.js`
Server on port 3000
DB is connected
{
  title: 'Pelota de futbol',
  description: 'Pelota que se utiliza para jugar al futbol.',
  image: 'https://i.natgeoefe.com/n/672b580d-0597-4e59-b57c-51f149bf92e8/80254.jpg'
}
POST /api/allProducts 200 212.475 ms - 28
GET /api/allProducts 304 2.598 ms - -
```

Agregamos otro elemento para mostrar que también se puede observar desde **Postman** con el endpoint GET como se cargaron correctamente:



```
1  ✓
2  ... "productName": "Pelota de fútbol",
3  ... "description": "Pelota que se utiliza para jugar al fútbol.",
4  ... "image": "https://essential.vtexassets.com/arquivos/ids/936432-800-auto?v=638235657190600000&width=800&
5  height=auto&aspect=true"
6
7
8
9
10
11
12
13
14
15
16
```

Body Cookies Headers (8) Test Results 200 OK 13 ms 780 B Save as example

Pretty Raw Preview Visualize JSON

```
1  {
2    "id": 1,
3    "productName": "Pelota de básquet",
4    "description": "Pelota que se utiliza para jugar al básquet.",
5    "image": "https://www.canteprijuguetes.com.ar/nueva/wp-content/uploads/2021/02/images-74.jpg",
6    "created_at": "2023-11-04T18:13:55.000Z"
7  },
8  {
9    "id": 2,
10   "productName": "Pelota de fútbol",
11   "description": "Pelota que se utiliza para jugar al fútbol.",
12   "image": "https://essential.vtexassets.com/arquivos/ids/936432-800-auto?v=638235657190600000&
13   width=800&height=auto&aspect=true",
14   "created_at": "2023-11-04T18:16:10.000Z"
15 }
16
```

Luego, realizamos el resto de endpoints y testeamos la funcionalidad de manera similar en **Postman**, quedando todos de la siguiente manera:

## DELETE

```
public deleteGame(req: Request, res: Response) {
  const { id } = req.params;
  db.query('DELETE FROM product WHERE id = ?', id, (err, result) => {
    if (err) {
      res.status(400).json({ error: 'Error al eliminar el
producto.' });
    } else {
      res.json({ message: 'El producto fue eliminado.' });
    }
  });
}
```



## PUT

```
public updateGame(req:Request, res:Response){
    const { id } = req.params;
    db.query('UPDATE product set ? WHERE id = ?', [req.body, id], (err,
result) => {
        if (err) {
            res.status(400).json({ error: 'Error al actualizar el
producto.' });
        } else {
            res.json({message: 'El producto fue actualizado' });
        }
    });
}
```

## GET

```
public async products(req: Request, res: Response): Promise<void> {
    db.query('SELECT * FROM product', (err, result) => {
        if (err) {
            console.error('Error al obtener los productos:', err);
            res.status(500).json({ error: 'Error al obtener los
productos' });
        } else {
            res.json(result);
        }
    });
}
```

## POST

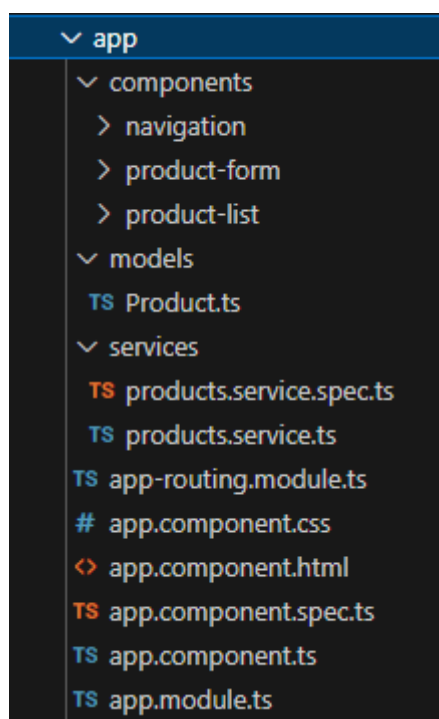
```
public async createProduct(req: Request, res:Response):Promise<void>{
    await db.query('INSERT INTO product set ?', [req.body]);
    res.json({message: 'Producto creado. '});
}
```

## Angular

Una vez ya tenemos todo el servidor funcional, empezamos a hacer la parte del front end, para poder consumir todos los endpoints que hicimos anteriormente.

Generamos el proyecto de Angular con **ng new frontend** para crear todas las librerías y dependencias necesarias para empezar a diseñar las vistas que van a tener los servicios que realizamos.

Creamos los componentes y servicios necesarios para llamar a los métodos que realizan la funcionalidad con la base de datos: la carga, la eliminación, actualización y demás.



**components:** en el componente de *navigation*, tenemos todo lo que corresponde a la navegación de la página, como la barra de navegación donde están las opciones que ofrece la app. En *product-form* tenemos implementada la funcionalidad que guarda el producto en la base de datos a la hora de agregarlo desde la aplicación, y en *product-list* el resto de endpoints que utilizamos para mostrar todos los productos.

**models:** en esta carpeta tenemos el tipo de dato Producto, que lo necesitamos para guardar y mostrar los productos con TypeScript y que coincidan los campos a la hora de guardarlo en la BD.

**services:** en esta carpeta tenemos los ruteos especificados cuando usamos cada endpoint, cada uno con su respectivo método y funcionalidad.

## Consumo de la API REST

Con todos los componentes creados, lo que nos queda por hacer es implementar el consumo desde el frontend. Primero, explicaremos la parte de la obtención de los datos, que lo implementamos en el componente *product-form*, donde tenemos toda la lógica para eliminar y obtener los datos desde nuestro servidor.

En `ngOnInit(){...}` aplicamos la lógica para que desde cada nueva carga de datos verifique que tenga el id correspondiente y lo guarde para nuestra variable `product`.

## Product-form

```
export class ProductFormComponent {

  @HostBinding('class') classes = 'row';

  product:any = {
    id: 0,
    productName: '',
    description: '',
    image: '',
    created_at: new Date()
  }

  constructor(private productService: ProductsService, private
router:Router, private activeroute:ActivatedRoute){ }

  ngOnInit(){
    const params = this.activeroute.snapshot.params;
    if(params['id']){
      this.productService.getProduct(params['id']).subscribe(
        res => {
          this.product = res;
        },
        err => console.error(err)
      );
    }
  }

  saveProduct(){
    delete this.product.created_at;
    delete this.product.id;
    this.productService.saveProduct(this.product).subscribe(
      res => {
        console.log(res);
        this.router.navigate(['./products']);
      },
      err => console.error(err)
    );
  }
}
```

Luego, en `saveProduct(){...}` implementamos el guardado del producto cuando se quiera agregar uno desde la aplicación, y una vez guardado lo reenviamos a la página principal con todos los productos cargados. Eliminamos los parámetros “created\_at” e “id” porque esos los carga automáticamente la base de datos.

## Product-list

```
export class ProductListComponent {

  @HostBinding('class') classes = 'row';

  products: any = [];

  constructor(private productService:ProductsService){
  }

  ngOnInit(){
    this.getProducts();
  }

  getProducts(){
    this.productService.getProducts().subscribe(
      res => {
        this.products = res;
        console.log('Producto creado');
      },
      err => console.error(err)
    );
  }


  deleteProduct(id:string){
    this.productService.deleteProduct(id).subscribe(
      res => {
        console.log();
        this.getProducts();
      },
      err => console.error(err)
    );
  }
}
```


De la misma manera trabajamos en el mostrado de los productos, primero, en el método `ngOnInit(){...}` mostramos los productos en la página principal apenas se abre la aplicación, para visualizarlos directamente en el inicio.

## Vista final

Home


SPORTIFY  PRODUCTOS  AGREGAR PRODUCTO


Pelota de fútbol 



Pelota que se utiliza para jugar al fútbol.


EDITAR PRODUCTO


Baston de Hockey 



Con su diseño ligero y equilibrado, ofrece un control excepcional y una excelente sensación de la pelota.

EDITAR PRODUCTO



Aro de básquet 



ARO BASKET METÁLICO CAÑO PESADO CON RED Y RESORTE . Diámetro: 46 cm. . Resorte para volcadas . Doble planchuela de fijación . Origen: Importado

EDITAR PRODUCTO

## Agregar producto

SPORTIFY  PRODUCTOS  AGREGAR PRODUCTO

Pelota de tenis
Pelota de tenis
<a href="https://files.cults3d.com/uploaders/19942810/iU">https://files.cults3d.com/uploaders/19942810/iU</a>
GUARDAR



0 . PELOTA DE TENIS

Pelota de tenis

09-11-2023

## Conclusiones

El proyecto de desarrollo de Sportify nos sirvió como experiencia que nos permitió adquirir conocimiento en el mundo de las aplicaciones web. A lo largo de este desarrollo, se exploraron diversos conceptos y se utilizaron tecnologías clave para la creación de una aplicación completa y funcional.

El proceso de inicio del proyecto incluyó la comprensión y aplicación de tecnologías fundamentales. Desde la configuración inicial del entorno de desarrollo con Node.js hasta la implementación de TypeScript, explorando herramientas como Express, Morgan y Cors para la creación de la API Rest. Esta etapa fue fundamental para comprender la importancia de la configuración del servidor y la gestión de rutas, así como el control de acceso a recursos entre diferentes dominios web.

El desarrollo del front end con Angular nos permitió aplicar los conceptos aprendidos en la construcción de interfaces de usuario interactivas. La creación de componentes, modelos y servicios para conectar y consumir los endpoints de la API Rest fue esencial. Además, se aprendió a sincronizar de manera efectiva las acciones del usuario en la interfaz con las operaciones de la base de datos, garantizando una experiencia fluida y eficiente para el usuario final.

En conclusión, el proyecto nos brindó una profunda experiencia en el desarrollo web, permitiéndonos no solo adquirir habilidades técnicas específicas, sino también comprender la importancia de la planificación, la implementación y la integración de tecnologías para construir una aplicación completa y funcional. Pudimos conseguir profundizar nuestra comprensión sobre la arquitectura de aplicaciones web, la interacción entre el servidor y el cliente, así como las mejores prácticas para el desarrollo de software.