

Angular 16+ - Edición 2023



## CRUD Angular API en Python

Centro público integrado de formación profesional

Alan Turing





## TS – Temas puntuales de la sección

### ¿Qué veremos en esta sección?

Este es un breve listado de los temas fundamentales:

#### 1. API en Python

Implementamos la persistencia de datos con una API en Python.

Hacemos uso del Framework 'Flask'





Angular 16+ - Edición 2023



# TS – CRUD de datos de Dragon Ball

Nuestro objetivo es conseguir algo tal que así.

Iniciar sesión    Registrar

### CRUD de datos

Agregar dato/s

Filtros de búsqueda

Género	Raza	Afiliación
Todo ▾	Todo ▾	Todo ▾

Buscar

Listar datos

#	Nombre	Fuerza	Acciones
1	Goku	60000000	Borrar
2	Vegeta	54000000	Borrar

### Iniciar sesión

Usuario

Contraseña

Iniciar sesión

Volver al listado

### Registrar

Nombre

Usuario

Contraseña

Registrarse

Volver al listado

CRUD en Angular 16+, desde Docker, accediendo a la API en PHP y Autorización con token

Bienvenido, jota [Cerrar sesión](#)

### CRUD de datos

Agregar dato/s

Filtros de búsqueda

Género	Raza	Afiliación
Todo ▾	Todo ▾	Todo ▾

Buscar

Listar datos

#	Nombre	Fuerza	Acciones
1	Goku	60000000	Borrar
2	Vegeta	54000000	Borrar



## TS – CRUD de datos de Dragon Ball

Antes de nada, recordemos como solucionar los problemas de dependencia para la librería 'idb' (IndexedDB)

- **Instalar la librería idb:**

```
npm install idb
```

- **Actualiza TypeScript (tsconfig.json)**

```
{
  "compilerOptions": {
    "target": "ES2022",
    "module": "ES2022",
    "lib": ["ES2022", "DOM"],
    "skipLibCheck": true
  }
}
```





TS – Nuevo apartado

Angular 16+ - Edición 2023



# Conectando Angular con una API en Flask (Python)





## TS – ¿Cómo se comunican Angular y Flask?

Antes de empezar a escribir código, es importante entender cómo Angular y Flask se comunican.

- Angular (Frontend)
  - Es la interfaz de usuario de la aplicación.
  - Hace solicitudes a la API Flask para obtener, agregar o eliminar datos.
- Flask (Backend)
  - Es el servidor que recibe las solicitudes de Angular.
  - Conecta con la base de datos MySQL para obtener o modificar los datos.
  - Devuelve respuestas en formato JSON.
- Proceso de comunicación
  - Angular envía una solicitud HTTP (GET, POST, DELETE) a Flask.
  - Flask recibe la solicitud, consulta la base de datos y devuelve una respuesta.
  - Angular recibe la respuesta y actualiza la interfaz de usuario.



## TS – Creación de la API con Flask

Para que Angular pueda comunicarse con Flask, primero debemos construir la API.

### Paso 1: Instalación de Flask y dependencias

Si Flask y las librerías necesarias no están instaladas, ejecuta:

```
# pip install flask flask-cors mysql-connector-python
```

### ¿Qué hacen estas librerías?

- **flask**: Nos permite crear la API.
- **flask-cors**: Permite que Angular se comunique con Flask.
- **mysql-connector-python**: Permite que Flask se conecte a MySQL.

También, puede ocurrir que tengas instaladas las librerías pero que el editor de código muestre un warning. En ese caso deberemos añadir `# type: ignore` a cada import

```
from flask import Flask, jsonify, request # type: ignore
import mysql.connector # type: ignore
from flask_cors import CORS, cross_origin # type: ignore
```



## TS – Creación de la API con Flask

### Paso 2: Configurar el servidor Flask

Ahora creamos el archivo app.py y escribimos el código base:

```
from flask import Flask, jsonify, request  
import mysql.connector  
from flask_cors import CORS  
  
app = Flask(__name__)  
CORS(app) # Permitir solicitudes desde Angular
```

### ¿Qué hace este código?

- Flask(\_\_name\_\_) crea el servidor web.
- CORS(app) permite que Angular pueda acceder a la API.



## TS – Creación de la API con Flask

### Paso 3: Configurar la conexión a MySQL

Añadimos la configuración de la base de datos:

```
db_config = {  
    'host': 'mysql-db',    # Nombre del servicio MySQL en Docker  
    'user': 'root',         # Usuario de la base de datos  
    'password': 'dejame',   # Contraseña  
    'database': 'dbzDB'    # Nombre de la base de datos  
}
```

### ¿Por qué Flask necesita conectarse a MySQL?

Porque los datos de los personajes no se guardan en Flask, sino en la base de datos. Flask solo actúa como un puente entre Angular y MySQL.



## TS – Implementación de los Endpoints en Flask

Ahora creamos las rutas que Angular usará para obtener, agregar y eliminar personajes.

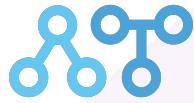
### 1. Obtener todos los personajes (GET /leer)

```
@app.route('/leer', methods=['GET'])

def leer():
    with mysql.connector.connect(**db_config) as conn:
        with conn.cursor(dictionary=True) as cursor:
            cursor.execute("SELECT * FROM personajes")
            personajes = cursor.fetchall()
    return jsonify({
        "success": True,
        "data": personajes,
        "message": "Personajes cargados correctamente"
    })
```

#### Explicación

- **@app.route('/leer', methods=['GET'])**: Define la **ruta** /leer que responde a solicitudes GET.
- **cursor.execute("SELECT \* FROM personajes")**: **Consulta** todos los personajes en la base de datos.
- **jsonify({...})**: Devuelve los personajes en formato **JSON**.



## TS – Implementación de los Endpoints en Flask

### 2. Agregar un personaje (POST /grabar)

```
@app.route('/grabar', methods=['POST'])
def grabar():
    datos = request.json # Recibe datos en formato JSON
    nombre = datos.get('name')
    ki = datos.get('ki')
    max_ki = datos.get('max_ki')
    race = datos.get('race')
    gender = datos.get('gender')
    description = datos.get('description')
    image = datos.get('image')
    affiliation = datos.get('affiliation', 'None')
    deleted_at = datos.get('deleted_at')
```

```
with mysql.connector.connect(**db_config) as conn:
    with conn.cursor() as cursor:
        consulta = """INSERT INTO personajes (name, ki, max_ki, race, gender, description, image, affiliation, deleted_at)
                      VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s)"""
        cursor.execute(consulta, (nombre, ki, max_ki, race, gender, description, image, affiliation, deleted_at))
        conn.commit()
return jsonify({"success": True, "mensaje": "Personaje añadido correctamente"})
```

### Explicación

- `request.json`: Recibe los datos enviados desde Angular.
- `INSERT INTO personajes (...) VALUES (...)`: Agrega un nuevo personaje a la base de datos.



## TS – Implementación de los Endpoints en Flask

### 3. Eliminar un personaje (DELETE /borrar)

```
@app.route('/borrar', methods=['DELETE'])  
def borrar():  
    id_personaje = request.args.get('id', type=int) # Obtiene el ID desde la URL  
  
    if not id_personaje:  
        return jsonify({"success": False, "mensaje": "ID no proporcionado"}), 400  
  
    with mysql.connector.connect(**db_config) as conn:  
        with conn.cursor() as cursor:  
            consulta = "DELETE FROM personajes WHERE id = %s"  
            cursor.execute(consulta, (id_personaje,))  
            conn.commit()  
  
    return jsonify({"success": True, "mensaje": "Personaje eliminado correctamente"})
```

#### Explicación

- `request.args.get('id', type=int)`: Obtiene el ID del personaje a eliminar.
- `DELETE FROM personajes WHERE id = %s`: Borra el personaje de la base de datos.



## TS – Creación del Servicio en Angular

Ahora que tenemos la API en Flask funcionando, vamos a conectar Angular para consumirla.

Vamos a ver cómo Angular envía solicitudes HTTP a Flask y cómo se reciben y procesan las respuestas.

En Angular, un servicio es una clase que maneja la comunicación con la API.

Esto nos permite separar la lógica de negocio del resto de la aplicación.

Paso 1: Crear el servicio

Ejecuta en la terminal:

```
# ng generate service crud/servicios/personaje/comunica-apipython
```

Esto debería de crear dos archivos:

- comunica-apiphp.service
- comunica-apiphp.service.spec

Esto creará un archivo comunica-api.service.ts dentro de la carpeta src/app/services/.



## TS – Creación del Servicio en Angular

Paso 2: Configurar la conexión con la API

Abre `comunica-api.service.ts` y escribe el siguiente código:

```
import { HttpClient, HttpHeaders } from '@angular/common/http';
import { Injectable } from '@angular/core';
import { Personaje } from '../Interfaces/personaje';
import { catchError, firstValueFrom, of } from 'rxjs';
@Injectable({
  providedIn: 'root'
})
export class Comunica ApiService {
  constructor(private http: HttpClient) { }
  private apiUrl = 'http://localhost:5001'; // URL de la API Flask
  private headers = new HttpHeaders({ 'Content-Type': 'application/json; charset=utf-8' });
}
```

### Explicación

- `HttpClient`: Permite hacer solicitudes HTTP.
- `apiUrl`: Define la URL base de la API Flask.
- `headers`: Configura el formato de los datos (JSON).
- `Injectable({ providedIn: 'root' })`: Hace que el servicio esté disponible en toda la aplicación.



## TS – Obtener Personajes desde la API

Paso 1: Crear el método cargarPersonajes()

Agrega este código en comunica-api.service.ts:

```
async cargarPersonajes(): Promise<{ data: Personaje[]; message: string }> {
  try {
    const response = await firstValueFrom(
      this.http.get<{ success: boolean; data: Personaje[]; message: string }>(`${this.apiUrl}/leer`).pipe(
        catchError((error) => {
          console.error('Error al traer datos de la API:', error);
          return of({ success: false, data: [], message: 'Error al obtener personajes' });
        })
      )
    );
  }
};
```

```
return {
  data: response.success ? response.data : [],
  message: response.message || 'Error al obtener datos'
};
} catch (error) {
  console.error('Error inesperado al conectar con la API:', error);
  return { data: [], message: 'Error inesperado' };
}
```

### Explicación

- `this.http.get(...)`: Hace una solicitud GET a Flask.
- `firstValueFrom()`: Convierte el observable en una promesa.
- `catchError(...)`: Captura errores en caso de que la API falle.
- Devuelve un array de personajes en caso de éxito, o un mensaje de error si falla.



## TS – Obtener Personajes desde la API

Paso 2: Llamar al servicio desde un componente

En el componente donde quieras mostrar los personajes, inyecta el servicio:

```
import { Component, OnInit } from '@angular/core';
import { Comunica ApiService } from './services/comunica-
api.service';
import { Personaje } from './Interfaces/personaje';

@Component({
  selector: 'app-personajes',
  templateUrl: './personajes.component.html',
  styleUrls: ['./personajes.component.css']
})
```

```
export class PersonajesComponent implements OnInit {
  personajes: Personaje[] = [];

  constructor(private apiService: Comunica ApiService) {}

  async ngOnInit() {
    const response = await this.apiService.cargarPersonajes();
    this.personajes = response.data;
  }
}
```

### Explicación

- `apiService.cargarPersonajes()`: Llama a la API para obtener los personajes.
- `ngOnInit()`: Carga los datos cuando el componente se inicia.



## TS – Agregar un Personaje

### Paso 1: Crear el método agregarPersonaje()

```
async agregarPersonaje(personaje: Personaje): Promise<{ success: boolean;  
message: string }> {  
  try {  
    const response = await firstValueFrom(  
      this.http.post<{ success: boolean; message: string  
>(`${this.apiUrl}/grabar`, personaje, { headers: this.headers }).pipe(  
        catchError((error) => {  
          console.error('Error al enviar datos a la API:', error);  
          return of({ success: false, message: 'Error al enviar datos a la API' });  
        })  
    );  
  }  
}  
return {  
  success: response.success,  
  message: response.message || 'Operación realizada'  
};  
} catch (error) {  
  console.error('Error inesperado al conectar con la API:', error);  
  return { success: false, message: 'Error inesperado al conectar con la API'  
};  
}  
}
```

### Explicación

- `this.http.post(...)`: Envía un nuevo personaje a la API Flask.
- `catchError(...)`: Captura errores en caso de fallo.



## TS – Agregar un Personaje

Paso 2: Crear un formulario en HTML

En el componente donde agregas personajes (personajes.component.html):

```
<form (submit)="agregar()">
  <input type="text" placeholder="Nombre" [(ngModel)]="nuevoPersonaje.name">
  <input type="text" placeholder="Ki" [(ngModel)]="nuevoPersonaje.ki">
  <button type="submit">Agregar Personaje</button>
</form>
```

Paso 3: Agregar el método agregar() en el componente

```
nuevoPersonaje: Personaje = { name: '', ki: '' };

async agregar() {
  await this.apiService.agregarPersonaje(this.nuevoPersonaje);
  this.nuevoPersonaje = { name: '', ki: '' }; // Resetear formulario
}
```

Explicación

- Vinculamos el formulario con ngModel para capturar los datos.
- Llamamos al servicio agregarPersonaje() al presionar el botón.
- Limpiamos los campos del formulario tras enviar los datos.



## TS – Eliminar un Personaje

### Paso 1: Crear el método eliminarPersonaje()

```
async eliminarPersonaje(id: number): Promise<{ success: boolean; message: string }> {
  try {
    const response = await firstValueFrom(
      this.http.delete<{ success: boolean; message: string }>(`${this.apiUrl}/borrar?id=${id}`, { headers: this.headers }).pipe(
        catchError((error) => {
          console.error('Error al eliminar el personaje:', error);
          return of({ success: false, message: 'Error al eliminar el personaje' });
        })
      )
  );
}

return {
  success: response.success,
  message: response.message || 'Operación realizada'
};
} catch (error) {
  console.error('Error inesperado al conectar con la API:', error);
  return { success: false, message: 'Error inesperado al conectar con la API' };
}
}
```

### Explicación

- `this.http.delete(...)`: Hace una solicitud DELETE a la API Flask.

### Paso 2: Agregar un botón para eliminar

```
<button (click)="eliminar(personaje.id)">Eliminar</button>
```



## TS – Creación del Servicio en Angular

### Paso 3: Crear el método en el componente

```
async eliminar(id: number) {  
  await this.apiService.eliminarPersonaje(id);  
}
```

### Explicación

- Cada personaje tiene un botón "Eliminar".
- Al presionarlo, se ejecuta `eliminarPersonaje()` y se elimina en la API.



TS – Nuevo apartado

Angular 16+ - Edición 2023



# Ejercicio 1





## TS – Ejercicio

Angular 16+ - Edición 2023



Ahora te toca a ti.

**Título:** Añadir la persistencia de datos en Planetas.

**Objetivo:**

Del mismo modo que hemos implementado la persistencia de datos para Personajes.

¿Qué tal si haces lo mismo para Planetas?

Extra:

- ¿Cómo sería para transformaciones?
- ¿Tendría sentido?





TS – Nuevo apartado

Angular 16+ - Edición 2023



# Ejercicio 1 (Solución)





Angular 16+ - Edición 2023



# TS – Ejercicio (Solución)

...

