

```
<!--Ingeniería en Sistemas de la Información-->  
<!--Universidad Tecnológica nacional Facultad  
Regional de Rosario-->  
<!--Desarrollo de Software-->
```

Open API/REST (Swagger) {

```
<Por="Lurati Ignacio, Olivieri  
Luca"/>
```

}



Contenidos

01

Introducción

02

Funcionamiento

03

Muestra técnica

04

Ventajas y Desventajas

05

Conclusión

Qué es una API REST {

API: Application Programming Interface

- Conjunto de definiciones y protocolos que se utiliza para desarrollar e integrar los sistemas de software de las aplicaciones.
- Contrato entre el usuario y el proveedor de información
- Establece el contenido que se necesita por parte del consumidor (la llamada) y el que requiere el productor (la respuesta).

REST: Representational State Transfer

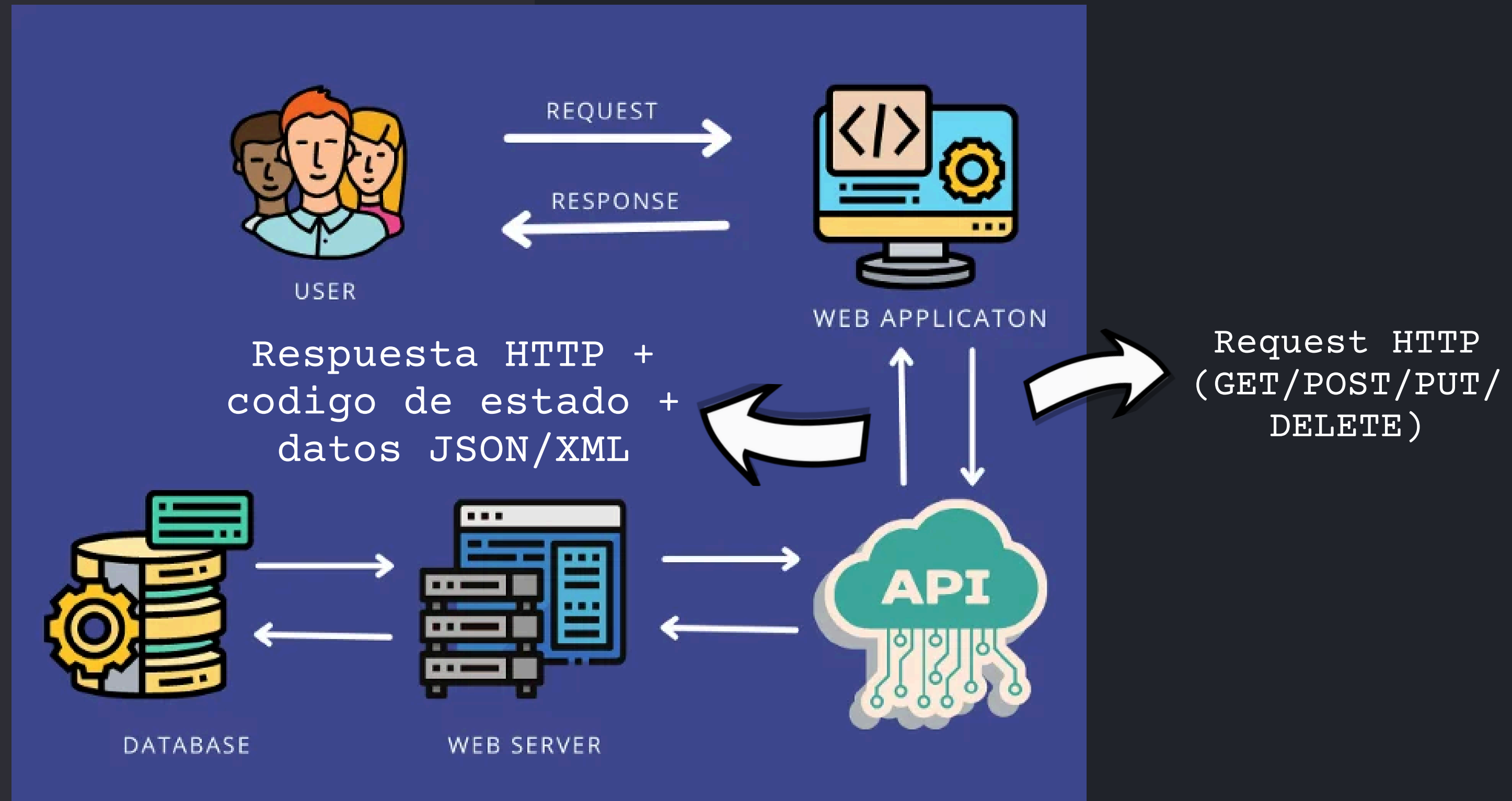
- Estilo de arquitectura de desarrollo de APIs
- Basado en el uso de estándares y Protocolos HTTP
- Permite el intercambio de información de manera ligera, escalable y sin estado (stateless)

API REST (RESTful API)

- Interfaz de programación de aplicaciones que sigue los principios de la arquitectura REST.
- Permite la comunicación entre diferentes sistemas a través de peticiones HTTP de forma ligera, escalable y sin estado.
- Los recursos se identifican mediante URLs (URIs).
- Se utilizan los métodos HTTP estándar (GET, POST, PUT/PATCH DELETE)

}

API REST (RESTful API)



OpenAPI

- La Especificación OpenAPI (ex Swagger) es un formato de descripción de API para API REST.
- Es un contrato escrito en JSON/YAML que describe cómo consumir y qué esperar de una API REST.
- Surge del proyecto predecesor Swagger. La empresa que lo desarrolló sometió la especificación existente a una licencia abierta, dejando su mantenimiento y desarrollo en manos de la iniciativa OpenAPI, cuyos miembros incluyen a Google, IBM, Microsoft y la Fundación Linux

OpenAPI permite documentar de manera clara

- Recursos que expone la API (Endpoints disponibles) con sus métodos HTTP
- Datos que se deben enviar en las peticiones (parametros, cuerpo)
- Respuestas del servidor (codigos de estado, estructura del JSON,etc)
- Métodos de autenticación
- Información de contacto, licencia, condiciones de uso y otra información.

}

OpenAPI

- Una especificación (estándar)
- Define cómo describir una API REST
- Más que una herramienta, es el lenguaje común para documentar APIs

El QUÉ

Swagger

- Conjunto de herramientas que trabajan con OpenAPI
- Facilita crear, generar y visualizar documentación.
- Componentes principales:
 - Swagger Editor → escribir y validar especificaciones OpenAPI.
 - Swagger UI → mostrar documentación en formato web interactivo.
 - swagger-jsdoc / swagger-codegen → generar especificación o código a partir de tu proyecto.

El CÓMO

Funcionamiento {

La especificación OpenAPI define una serie de propiedades que pueden utilizarse para desarrollar una API propia. Estas propiedades se agrupan en los llamados objetos (objects). En la actual versión 3.0.3, OpenAPI define la estructura de los siguientes objetos, entre otros:

- **Info Object:** versión, nombre, etc. de la API.
- **Contact Object:** datos de contacto del proveedor de la API.
- **License Object:** licencia bajo la cual la API proporciona sus datos.
- **Server Object:** nombres del host, estructura del URL y puertos del servidor a través del cual se dirige la API.
- **Components Object:** componentes encapsulados que pueden utilizarse varias veces dentro de una definición de API.
- **Paths Object:** rutas relativas a los puntos finales de la API que se utilizan junto con el servidor del objeto.
- **Path Item Object:** operaciones permitidas para una ruta específica como GET, PUT, POST, DELETE.
- **Operation Object:** especifica, entre otras cosas, los parámetros y las respuestas del servidor que se esperan de una operación.

}

Funcionamiento {


```
const swaggerOptions = {
  definition: {
    openapi: "3.0.0",
    info: {
      title: "Mi API",
      version: "1.0.0",
    },
    description: "Documentación de todos los CRUDs",
    components: {
      securitySchemes: {
        bearerAuth: {
          // nombre del esquema
          type: "http",
          scheme: "bearer",
          bearerFormat: "JWT",
        },
      },
      security: [
        {
          bearerAuth: [], // aplica a todos los endpoints por defecto
        },
      ],
    },
    apis: ["../src/**/*.routes.ts"],
  };
};
```



Info Object



Components Object



Propia de swagger-jsdoc,
permite generar el Paths
Object leyendo las rutas
que estén documentadas

}

Cómo iniciar Swagger en TS {

Pasos a seguir

- Importar **swaggerUI** y **swaggerJsDoc**
- Definir configuración general
- Validar archivo json formato swagger con **JSDoc**
- Definir endpoint donde mostrar documentación
- Documentar endpoints a mostrar

```
import swaggerUi from "swagger-ui-express";  
import swaggerJsdoc from "swagger-jsdoc";
```

```
const swaggerOptions = {}
```

```
const swaggerSpec = swaggerJsdoc(swaggerOptions);
```

```
app.use("/api-docs", swaggerUi.serve,  
  | swaggerUi.setup(swaggerSpec));
```

```
@openapi  
/api/cities/:  
  get:  
    summary: Obtener todas las ciudades
```

```
}
```

Importante {

Para funcionar automáticamente, swagger requiere generar un documento **swagger.json** con la documentación de la api

Si este documento puede ser generado de cualquier otra forma, basta con usarlo en el **swaggerUI.setup(doc)**

Veremos entonces que existen alternativas a tener que hacerlo **a mano**

}

El Código {

Si utilizamos **swaggerAutogen** podemos detectar todas las rutas y generar documentación automática sobre ellas con el siguiente script



```
import swaggerAutogenPkg from "swagger-autogen";

const swaggerAutogen = swaggerAutogenPkg();

swaggerAutogen("./swagger-output.json",
  ["/src/app.ts"], doc)
  .then(() => {
    console.log("Swagger JSON generado!");
  });
```

Lo levantamos directamente desde el app.ts de la siguiente manera



```
const swaggerPath = path.join(
  process.cwd(), "swagger-output.json");

const swaggerDocument = JSON.parse(
  fs.readFileSync(swaggerPath, "utf-8"));

const app = express();
app.use("/api-docs", swaggerUi.serve,
  swaggerUi.setup(swaggerDocument));
```

}

Problemas de este método {

NO RECONOCE TIPOS

Nos mostrará que todos los parámetros y propiedades de la request son de type: "any"

NO FUNCIONA SIN NEST.JS

Requiere un framework sólido de desarrollo de apis y controladores para que detecte y documente correctamente sin nuestra intervención

COMPORTAMIENTO INCONSISTENTE

Endpoints con middlewares que añaden propiedades no andarán, tokens tienden a fallar, no detecta respuestas

}

Limitaciones de JS {

En algunos lenguajes orientados al backend existen paquetes que permiten documentar automáticamente los endpoints y generar un archivo swagger

Cómo levantar el endpoint de swagger en **ASP.NET**

.NET

```
if (app.Environment.IsDevelopment())  
{  
    app.UseSwagger();  
    app.UseSwaggerUI();  
}
```

```
builder.Services.AddEndpointsApiExplorer();  
builder.Services.AddSwaggerGen();
```

Cómo autogenerar el swagger.json con **SwashBuckle**

}

¿Qué más permite OpenAPI {

Existen herramientas que a través de la documentación generada permiten múltiples funciones

Permiten generar **servidores de prueba**, crear **sanitizadores**, crear **clientes axios**, **testing** y muchísimo más!

Los invitamos a que se fijen por sí mismos

<https://tools.openapis.org/categories/sdk.html>

}

```
/** @openapi
```

Gracias {

```
saludar("Lurati Ignacio, Olivieri Luca")
```

}

```
*/
```