

Apunte 12 - Arquitectura Backend

Introducción

La arquitectura de software define, de manera abstracta, los componentes que llevan a cabo una tarea de computación, sus interfaces y la comunicación entre ellas. Toda arquitectura debe ser implementable en una arquitectura física, que consiste simplemente en determinar qué computadora tendrá asignada una determinada tarea.

Un patrón arquetípico

Las arquitecturas de referencia ayudan a organizar y alinear el trabajo de distintos equipos bajo un patrón arquetípico que engloba tareas de un mismo tipo.



¿Qué es una arquitectura de referencia?

Las arquitecturas de referencia están formadas por estructuras e integraciones recomendadas de productos y servicios de TI para crear una solución. Estas arquitecturas incorporan las mejores prácticas y lineamientos de un determinado sector y se pueden aplicar a diferentes contextos. Una arquitectura de referencia se anticipa y da respuesta a las preguntas más frecuentes que puedan surgir. En consecuencia, ayuda a los equipos a evitar los errores y los retrasos que se podrían producir sin el uso de un conjunto probado de mejores prácticas y soluciones.

Importancia de la selección de una arquitectura de referencia

La selección de una arquitectura de referencia adecuada es muy importante, ya que son los cimientos sobre los que todo el desarrollo se monta. Al igual que con la construcción de un edificio, si los cimientos no son buenos, la construcción de todo el edificio será problemática, llegando incluso a impedir que se termine el edificio. Elegir la mejor arquitectura para solucionar el problema puede garantizar un desarrollo ágil y mantenible, y un producto duradero en el tiempo.

Arquitectura monolítica

Una arquitectura monolítica se refiere al clásico diseño unificado de un software, donde todo está acoplado y unido en un solo elemento. De ahí su nombre, que hace referencia a un monolito (monumento de piedra de una sola pieza).



El software monolítico está diseñado para ser autocontenido, todos sus componentes están interconectados y son interdependientes entre sí.

En esta arquitectura, fuertemente acoplada, cada componente y sus elementos asociados tienen que estar presentes para que el código se ejecute, ya que esta arquitectura concentra toda funcionalidad en un solo punto.

¿Cuáles son los principales beneficios de este tipo de arquitectura?

- Los programas son fáciles de desarrollar.
- El despliegue y la ejecución del software son muy sencillos.
- El costo de desarrollo es bajo en comparación con otras arquitecturas.

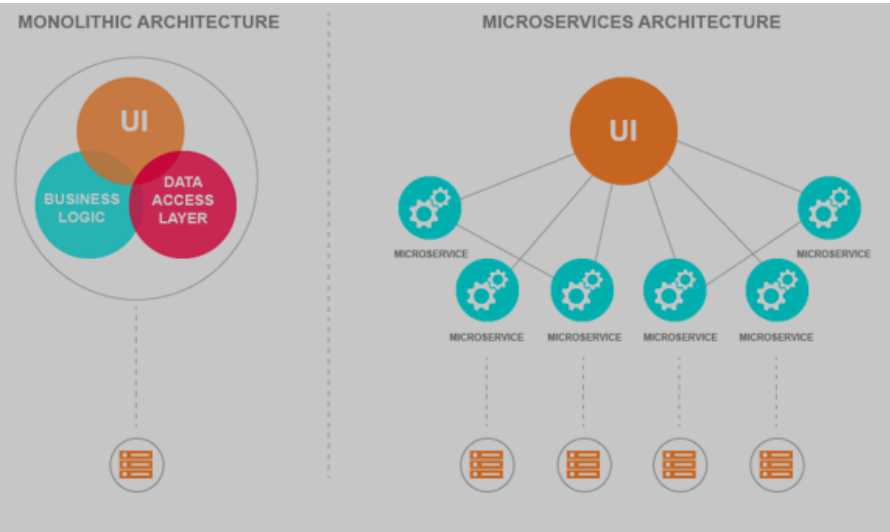
Arquitectura de microservicios

Los microservicios son todo lo opuesto a la arquitectura monolítica, ya que los procesos se distribuyen en múltiples partes, más pequeñas e independientes.

Una de las maneras de implementar este patrón es la distribución de los procesos en bloques separados. Estos bloques se llaman “componentes de servicios” y agrupan de uno a varios componentes enfocados en realizar una tarea o un área de negocio de la aplicación.

Estos componentes de servicios están completamente desacoplados entre sí y funcionan de forma totalmente independiente, lo que permite que cualquiera de ellos pueda ser reemplazado en tiempo real sin afectar el funcionamiento de la aplicación.

¿Qué es un microservicio?

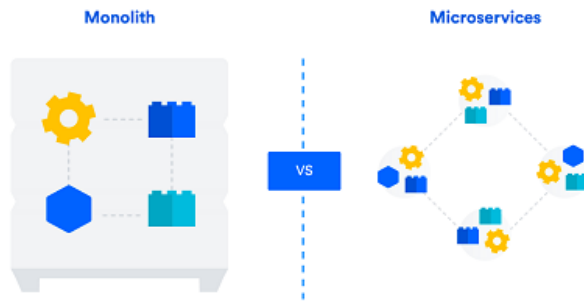


Un microservicio es un componente independiente y autónomo de un sistema o aplicación más amplio, encargado de implementar una funcionalidad completa de negocio. Dicha funcionalidad se expone a través de una API.

El software basado en microservicios puede descomponerse en varias partes funcionales independientes. De esta manera, cada uno de estos servicios puede ser desplegado, modificado y re-desplegado sin comprometer otros aspectos funcionales de la aplicación. Esto trae como resultado que, en caso de necesitarlo, solo tendremos que modificar un par de servicios en lugar de re-desplegar toda la aplicación por completo nuevamente.

Estructura de la aplicación	Autonomía de componentes	Especialización funcional
La arquitectura de microservicios organiza una aplicación como un conjunto de servicios pequeños, cada uno corriendo en su propio proceso y comunicándose mediante mecanismos ligeros, usualmente una API HTTP.	Son componentes completamente autónomos e independientes, lo que permite que sean desarrollados, desplegados y escalados de manera individual.	Cada microservicio está diseñado para realizar una función específica de negocio y opera de manera independiente, lo que mejora la modularidad y facilita la actualización y el mantenimiento.

¿Por qué puede resultar beneficioso trabajar con microservicios?



Agilidad: Cada microservicio se puede diseñar, desarrollar e implementar de forma independiente de los otros. Esto aporta agilidad, facilitando la implementación frecuente de nuevas versiones de los microservicios.

Diversidad: En un sistema compuesto por múltiples servicios colaborativos, como es el caso de los microservicios, es posible utilizar diferentes lenguajes de programación y tecnologías en cada servicio.

Productividad: Esta diversidad nos permite seleccionar la herramienta más adecuada para cada tipo de tarea, en lugar de depender de una solución estandarizada que debe servir para todo.

Independencia: Cada equipo tiene la capacidad de administrar, desarrollar, implementar y escalar su propio servicio de manera independiente respecto a los demás equipos.

¿Qué es una API?

La abreviatura API significa "interfaz de programación de aplicaciones". Hablando de componentes de software, una API es la interfaz pública de un componente que permite comunicarse con él para hacer uso del mismo. En el contexto de arquitecturas de microservicios, una API describe una interfaz de servicio expuesta a través del protocolo HTTP.

El propósito de una API es intercambiar datos entre diferentes sistemas.

Las APIs le permiten al usuario final utilizar aplicaciones, programas y softwares consultando, cambiando y almacenando datos de diferentes sistemas, como si fuera una especie de traductor.

Un ejemplo de API es el conjunto de funciones que permite a distintos sistemas el envío de notificaciones a clientes por diferentes medios (correo electrónico, sms o mensajes al celular, notificaciones Push, etc.). Las notificaciones Push, por ejemplo, son muy utilizadas en los dispositivos móviles para informar sobre actualizaciones de las apps, novedades, hitos, etc.

Desde una perspectiva de negocio ...



Las APIs exponen y extienden el negocio.

Para el consumidor de la API

Desde una perspectiva técnica ...



Es un contrato y una vía de acceso que permite la interoperabilidad entre sistemas.

El proveedor de la API

Para el consumidor de la API



La misma no es más que una descripción de la interfaz y un endpoint (o conjunto de URLs). El idioma es el protocolo HTTP.

El proveedor de la API



se tienen en cuenta todos los detalles de la infraestructura de implementación de la misma.

Características que debe tener una API

- **Extensibilidad:** Su funcionalidad debe ser fácil de extender. Las personas siempre van a requerir más funcionalidades.
- **Predictibilidad y consistencia:** Usar una API no debería tener comportamientos sorpresa. Si bien puede afectar por perfiles diferentes, el comportamiento debe ser consistente para todos.
- **Errores claros:** Evitar fallos. Contemplar todas las validaciones necesarias.

Algunos usos de APIs

- Usar Google Maps en aplicaciones.
- Integrar videos de Youtube en una página web.
- Consultar el clima en Google.
- Iniciar sesión en una página web a través de tus cuentas de redes sociales.
- Crear bots para chats automáticos (hoy se utilizan en la mayoría de las empresas para poder dar respuesta las 24 horas).

APIs y Microservicios: una relación funcional

- Los microservicios pueden ser un medio para implementar el backend de servicio expuesto por una API.
- Los microservicios generalmente dependen de las API como un medio estándar de comunicación entre ellos en una red interna.

REST

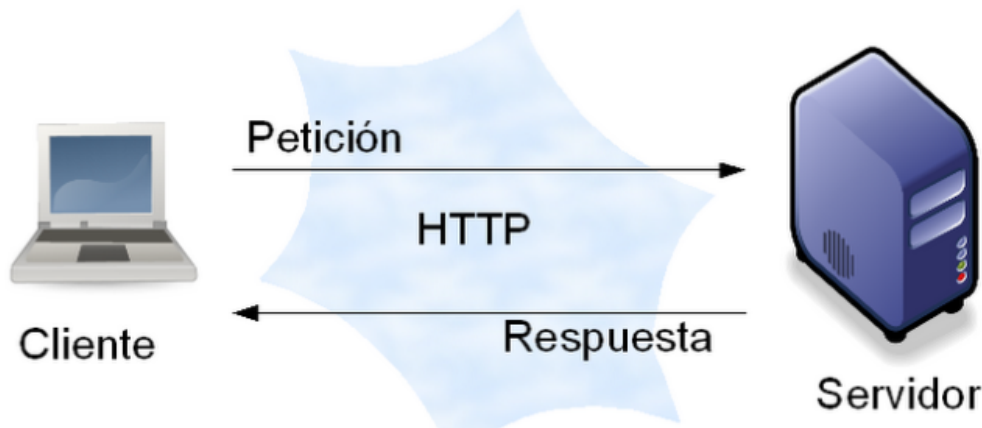
En año 2000, Roy Fielding definió REST como un estilo arquitectónico web basado en el protocolo HTTP, que impone condiciones sobre cómo debe funcionar una API.

Principios REST (*)

- **Cliente / Servidor:** Definen un interface de comunicación entre ambos separando completamente las responsabilidades entre ambas partes.
- **Sin Estado:** Cada petición que se realiza a ellos es completamente independiente de la siguiente.
- **Cache:** El contenido de los servicios REST se puede cachear de tal forma que una vez realizada la primera petición al servicio, pero el resto no sea necesario procesarla.
- **Arquitectura en Capas:** El servidor puede disponer de varias capas para su implementación. Esto ayuda a mejorar la escalabilidad, el rendimiento y la seguridad.
- **Interfaz Uniforme:** Esta restricción indica que cada recurso del servicio REST debe tener una única dirección, "URI".

Arquitectura Cliente Servidor

Las RESTful API funcionan bajo una arquitectura cliente servidor, usando HTTP como protocolo de comunicación.



Protocolo HTTP

HTTP es un protocolo de comunicación utilizado en la web para la transferencia de datos entre diferentes dispositivos y servidores. Se basa en un modelo cliente-servidor. Utiliza diferentes métodos para realizar operaciones en los datos. Utiliza un formato de mensaje simple que consta de una cabecera y un cuerpo.

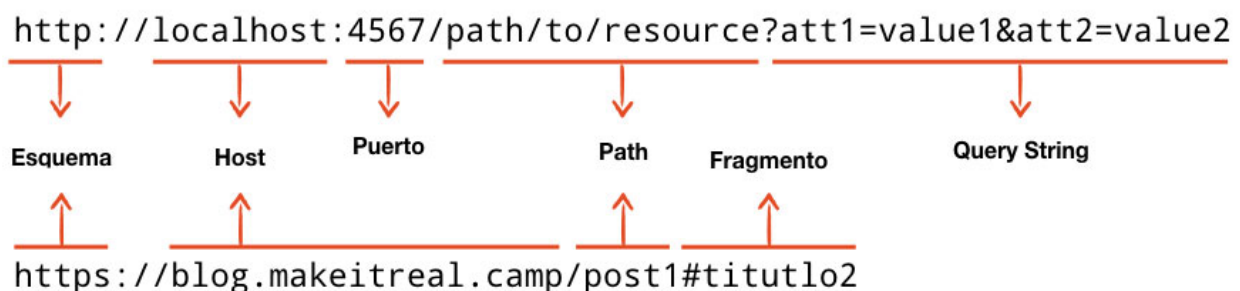


Características del Protocolo HTTP

- Basado en solicitudes y respuestas
- Independencia de la plataforma
- Simplicidad
- Flexibilidad
- Escalabilidad

Estructura URL HTTP

Un URL (Uniform Resource Locator) se utiliza para ubicar un recurso en Internet. Los URLs no solo se pueden utilizar para el protocolo HTTP, se utilizan en muchos otros protocolos.



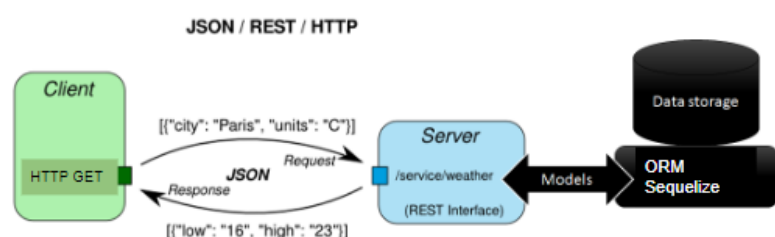
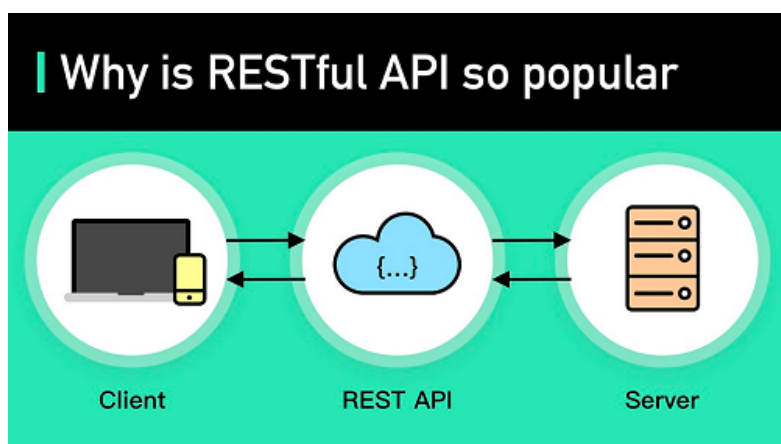
- **Esquema:** El esquema define el protocolo a utilizar, para HTTP puede ser http o https (el protocolo seguro de HTTP).
- **Host:** La IP o el nombre del servidor que se quiere acceder (p.e. 127.0.0.1, localhost, google.com, www.google.com, etc.).
- **Puerto:** El puerto en el que está escuchando el servidor HTTP. Si se omite se asume que es el 80.
- **Path:** La ruta al recurso que se quiere acceder.
- **Query String:** Contiene información adicional para el servidor en forma de propiedades (atributo=valor). Las propiedades se separan por &.
- **Fragmento:** La referencia a una ubicación interna del documento.

Request (Petición) y Response (Respuesta)



¿Qué es un API RESTful?

La API RESTful es una interfaz que dos sistemas de computación utilizan para intercambiar información de manera segura a través de una arquitectura Rest. La mayoría de las aplicaciones para empresas deben comunicarse con otras aplicaciones internas o de terceros para llevar a cabo varias tareas.



Beneficios API RESTful

- **Escalabilidad:** Los sistemas que implementan API REST pueden escalar de forma eficiente porque REST optimiza las interacciones entre el cliente y el servidor.
- **Flexibilidad:** Los servicios web RESTful admiten una separación total entre el cliente y el servidor. Simplifican y desacoplan varios componentes del servidor, de manera que cada parte pueda evolucionar de manera independiente. Los cambios de la plataforma o la tecnología en la aplicación del servidor no afectan la aplicación del cliente. La capacidad de ordenar en capas las funciones de la aplicación aumenta la flexibilidad aún más. Por ejemplo, los desarrolladores pueden efectuar cambios en la capa de la base de datos sin tener que volver a escribir la lógica de la aplicación.
- **Independencia:** Las API REST son independientes de la tecnología que se utiliza. Puede escribir aplicaciones del lado del cliente y del servidor en diversos lenguajes de programación, sin afectar el diseño de la API. También puede cambiar la tecnología subyacente en cualquiera de los lados sin que se vea afectada la comunicación.

Principios de diseño de API Restful

- Manténgalo simple (Keep it simple).
- Usar sustantivos y no verbos.
- Uso de los métodos HTTP adecuados.
- Usar plurales.
- Utilizar parámetros.
- Utilice códigos HTTP adecuados.
- Versiones.
- Usar paginación.
- Formatos soportados.
- Utilizar mensajes de error adecuados.
- Uso de las especificaciones de OpenAPI

Manténgalo simple (Keep it simple)


Necesitamos asegurarnos de que la Resource Path (URI) de la API sea simple. Por ejemplo, si queremos diseñar APIs para productos, debería diseñarse como:

Path	API	Descripción
/products	Productos	API para obtener todos los productos.
/products/A34678	Productos	API para obtener un producto específico con su identificador único (ID).
/users	Usuarios	API para obtener todos usuarios.
/users/234567	Usuarios	API para obtener un usuarios especifico con su identificador único (ID).


Usar sustantivos y no verbos

Muchos desarrolladores cometen el error de olvidar que tenemos **métodos HTTP** para describir mejor las APIs y en su lugar utilizan verbos en las URLs de las APIs. Por ejemplo, API para obtener todos los productos debe ser:

```
1 /products
2
```



```
1 /getAllProducts
2
```



Sin embargo existen excepciones para acciones que no se pueden representar con GET, POST, PUT, DELETE. Ejemplos:

- **/products/123455/activate**: para cambiar el estado de un producto, en este caso a estado "activo".
- **/products/search**: se aplica cuando queremos realizar una consulta compleja que no podemos implementar con un GET.

Convención nombres (Naming Conventions)

En el mundo de la programación existen 3 tipos principales de convenciones de nombre de recursos: CamelCase, snake_case y spinal-case. Son solo una forma de nombrar los recursos para parecerse al lenguaje natural y evitar espacios, apóstrofes y otros caracteres exóticos. Este hábito es universal en los lenguajes de programación donde solo se autoriza un conjunto finito de caracteres para los nombres.

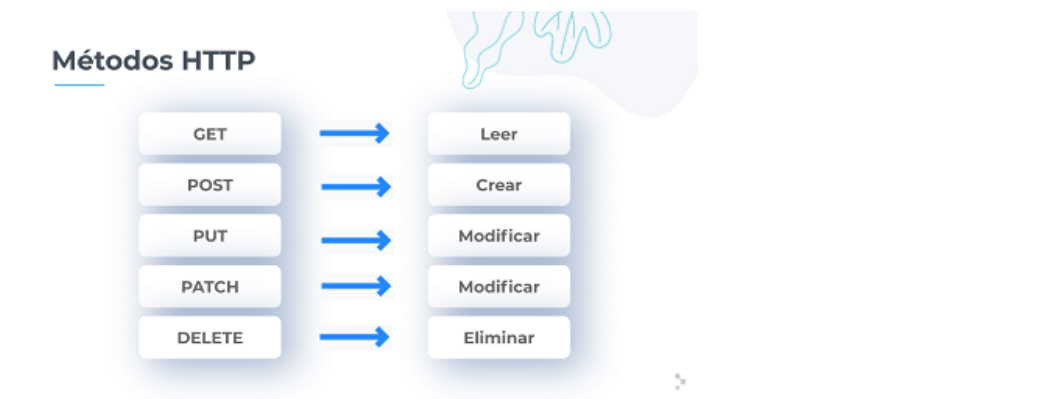
Para APIs utilizaremos **spinal-case** como convención de nombre Urls de APIs. Ejemplo: para la API unidades de negocio, la Url sería /business-unit/12345.

Uso de los métodos HTTP adecuados

Las APIs de RESTful tienen varios métodos para indicar el tipo de operación que vamos a realizar con esta API. Debemos asegurarnos de que usamos el método HTTP correcto para una operación determinada.

Método HTTP	Descripción
GET	Para obtener un recurso o colección de recursos.
POST	Para crear un recurso o colección de recursos.
PUT	Para actualizar todos los atributos de un recurso o conjunto de recursos existentes.

Método HTTP	Descripción
PATCH	Para actualizar solo algunos atributos de un recurso o conjunto de recursos existentes.
DELETE	Para borrar un recurso existente o el conjunto de recursos.



Usar plurales

Mantenerlo en plural evita confusiones sobre si estamos hablando de obtener un solo recurso o una colección.



Utilizar parámetros

A veces necesitamos tener una API que debería contar más que sólo por identificación. Aquí debemos hacer uso de los parámetros de consulta para diseñar la API.

Tipo Parámetro	Ejemplo	Descripción
Path	/products/{id_producto}	Donde id_producto es el identificador del recurso.
QueryString	/products?name='ABC'	Debe ser preferido sobre /getProductsByName.
Body	{"status": "active"}	Debe ser preferido sobre /getProductsByName.
Header	authorization: TOKEN	Debe ser preferido sobre /getProductsByName.

Utilice códigos HTTP adecuados

Tenemos muchos códigos de estado HTTP.



La mayoría terminamos usando dos (200 y 500). Esta no es una buena práctica. A continuación se presentan algunos de los códigos HTTP más utilizados:

Status Code	Descripción
200 OK	Este es el código HTTP más comúnmente utilizado para mostrar que la operación realizada es exitosa.
201 CREATED	Esto se puede utilizar cuando utiliza el método POST para crear un nuevo recurso.

Status Code	Descripción
202 ACCEPTED	Esto se puede utilizar para confirmar la petición enviada al servidor.
400 BAD REQUEST	Esto se puede utilizar cuando falla la validación de entrada del lado del cliente.
401 UNAUTHORIZED	Esto se puede utilizar si el usuario o el sistema no está autorizado para realizar una determinada operación.
403 FORBIDDEN	Esto se puede utilizar si el usuario o el sistema no está autorizado para realizar una determinada operación.
404 NOT FOUND	Esto se puede utilizar si está buscando un recurso determinado y no está disponible en el sistema.
500 INTERNAL SERVER ERROR	Esto nunca debe ser lanzado explícitamente, pero puede ocurrir si el sistema falla.
502 BAD GATEWAY	Esto se puede utilizar si el servidor ha recibido una respuesta no válida del servidor ascendente.

Versiones

La creación de versiones de las API es muy importante. Existen diferentes maneras de implementar versionado de API, algunos utilizan versiones como fechas, mientras que otras utilizan versiones como parámetros de consulta. Lo que mejor funciona es mantenerlo prefijado al recurso. Por ejemplo:

- `/v1/products`
- `/v2/products`

Siempre es una buena práctica mantener **retrocompatibilidad** para que ante un cambio en la versión de la API los **consumidores** tengan suficiente tiempo para pasar a la siguiente versión.

Usar paginación

El uso de la paginación es imprescindible cuando se expone una API que puede retornar grandes cantidades de datos, y si no se hace un balanceo de carga adecuado, el consumidor puede terminar por derribar el servicio. Debemos tener siempre en mente que el diseño de la API debe ser a prueba de tontos.

Es buena práctica usar los parámetros `limit` y `offset`:

- `limit`: valor numérico que representa la cantidad de resultados a retornar.
- `offset`: valor numérico que representa la posición desde la cual se va a retornar la cantidad de valores definido en `limit`.

Por ejemplo, si queremos paginar una API como la de `/products` con paginas de 25 elementos cada pagina se debería consultar de la siguiente manera:

- Pagina 1: `GET /products?limit=25&offset=0`.
- Pagina 2: `GET /products?limit=25&offset=26`.
- Pagina 3: `GET /products?limit=25&offset=51`.

Se aconseja mantener un valor predeterminado de **limit** y **offset**.

Formatos soportados

También es importante elegir cómo responderá su API. La mayoría de las aplicaciones de hoy en día deberían devolver las respuestas de **JSON**, a menos que tenga una aplicación heredada que aún necesite obtener una respuesta **XML**.

XML

vs.

JSON

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <endereco>
3   <cep>31270901</cep>
4   <city>Belo Horizonte</city>
5   <neighborhood>Pampulha</neighborhood>
6   <service>correios</service>
7   <state>MG</state>
8   <street>Av. Presidente Antônio Carlos, 6627</street>
9 </endereco>

```

```

1 {
2   "endereco": {
3     "cep": "31270901",
4     "city": "Belo Horizonte",
5     "neighborhood": "Pampulha",
6     "service": "correios",
7     "state": "MG",
8     "street": "Av. Presidente Antônio Carlos, 6627"
9   }
10 }

```

Utilizar mensajes de error adecuados

Siempre es una buena práctica mantener un conjunto de mensajes de error que la aplicación envía y responder a ellos con la identificación adecuada.

Por ejemplo, si utilizas APIs de gráficos de Facebook, en caso de errores, devuelve un mensaje como este:

```

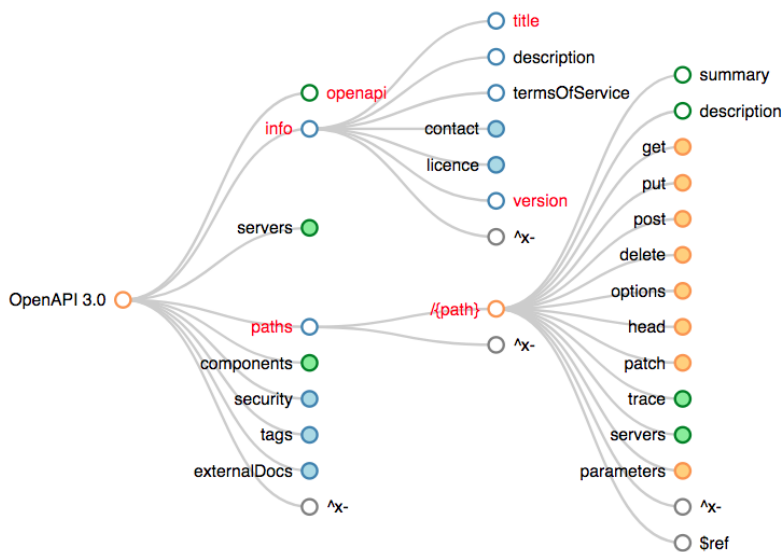
{
  "error": {
    "message": "(#803) Some of the aliases you requested do not exist: products",
    "type": "OAuthException",
    "code": 803,
    "fbtrace_id": "FOX2AhLh80"
  }
}

```

Documentación de API Rest: Swagger

¿Qué es swagger?

- Swagger es una especificación abierta para definir las API REST.
- El documento Swagger especifica la lista de recursos que están disponibles en la API REST y las operaciones que se pueden llamar en esos recursos.
- Swagger fue desarrollado por Reverb, pero actualmente se caracteriza por su neutralidad como código abierto al abrigo de la Fundación Linux, llamada Open API Initiative. Con el tiempo, Swagger se ha renombrado como especificación OpenAPI, aunque sigue siendo conocido de manera no oficial con el nombre de Swagger.
- Editor swagger: <https://swagger.io/tools/swagger-editor/>
- OpenApi Specification: <https://github.com/OAI/OpenAPI-Specification>
- OpenAPI map 3.0:



OpenAPI Object



Modified object!



Allows extension with x- properties



OpenAPI Specification

Description

OpenAPI 3.0 top level object. This is the root document object for the OpenAPI Specification document.

OpenAPI Object Change log



Deleted properties



New properties

What's new

The new OpenAPI Specification version 3.0 offers many welcomed improvements and new features (see OpenAPI [blog post series about this](#)).

Here are the noticeable changes on top level (*navigate through the tree to see what happened on other levels*):

- Bye bye **swagger** and hello **openapi**.
- Reusable definitions are centralized in **components** making the document more consistent (the previous version mixed reusable and

Created by Arnaud Lauret, the API Handyman

Probemos esto

Probar la Tienda de Mascotas en Línea con swagger Podes probar el endpoint GET /pet/{petId} de la API de la tienda de mascotas en línea de Swagger en la página web <https://petstore.swagger.io/#/pet/getPetById>.

Para probar, seguí estos pasos:

1. En un navegador abrir la página web <https://petstore.swagger.io/#/pet/getPetById>.
2. En la sección "Try it out", ingresa un ID de mascota en el campo "petId". Puedes usar cualquier ID de mascota existente, como "1", "2", "3", etc.
3. Haz clic en el botón "Execute" para enviar la solicitud al servidor. En la sección "Server response", se mostrará la respuesta del servidor. Si la solicitud se realizó correctamente, deberías ver los detalles de la mascota con el ID especificado. Ten en cuenta que esta página web es una herramienta de prueba útil para experimentar con las diferentes rutas y métodos de la API de la tienda de mascotas en línea de Swagger.

Name	Description
petId * required integer(int64) (path)	ID of pet to return

2

Execute Clear

Responses Response content type: application/json

Curl

```
curl -X 'GET' \
  'https://petstore.swagger.io/v2/pet/2' \
  -H 'accept: application/json'
```

Request URL

https://petstore.swagger.io/v2/pet/2

Server response

Code	Details
200	<p>Response body</p> <pre>{ "id": 2, "category": { "id": 0, "name": "cat" }, "name": "Tody", "photoUrls": ["string"], "tags": [{ "id": 0, "name": "string" }], "status": "available" }</pre> <p>Download</p>

¿Para quién documentamos las APIs?

- Esta es la primer pregunta que tenemos que resolver como buena práctica de APIs. Las APIs se construyen para resolver problemas, CUs, y contienen funcionalidad específica de negocio que los equipos de desarrollo integrarán a las aplicaciones para dar respuesta al negocio de cara al cliente. Por lo tanto, existen dos tipos de audiencias potenciales de las APIs, que influirán en el uso y en la curva de adopción.
- Con Swagger se puede describir, producir, consumir y visualizar APIs, por lo que es sumamente interesante como desarrollador proyecto.
- Tomadores de decisiones: son las personas que evalúan los servicios que ofrece el API y deciden si tiene sentido que el equipo de desarrollo dedique tiempo a explorar el servicio. Buscan utilizar las APIs para resolver posibles desafíos en su estrategia de producto o servicio. Algunos ejemplos de tomadores de decisiones son los gerentes de producto o PO.
- Usuarios: estas son las personas que trabajarán directamente con las API. Necesitan entender los detalles del API y cómo se aplicaría a su caso de uso. Esto podría significar aprender a invocar e integrarla a otras aplicaciones. Por lo tanto, el API debe ser fácil de usar y tener una excelente documentación para que estos usuarios puedan integrarse con éxito, lo más rápido posible y autogestionarse (sin tener que ponerse en contacto con el owner del API para comprender su utilidad).

API USERS

API DECISION MAKERS





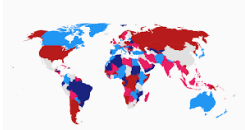

Newcomer:
First-Time User
(e.g. Front End Dev)

Debugger:
Solving a specific issue
(e.g. Back End Dev)

Evaluator:
Deciding whether to
use the API
(e.g. CTO)

Problem Solver:
Is X possible with
this API?
(e.g. PM)

¡APIs famosas!

The Star Wars API	The Marvel Comics API	Rest API Countries	JSON placeholder
			
https://swapi.dev/documentation#start	https://developer.marvel.com/	https://restcountries.com/	https://jsonplaceholder.typicode.com/

Herramientas de desarrollo: Postman/Curl/VSCode

Postman

Postman es una herramienta de colaboración de desarrollo de API que permite a los desarrolladores crear, compartir, probar y documentar API. Es una plataforma para diseñar, probar y depurar API, lo que significa que permite a los desarrolladores crear solicitudes HTTP personalizadas y ver las respuestas del servidor de una manera intuitiva y fácil de usar.

Con Postman, los desarrolladores pueden crear solicitudes HTTP para cualquier tipo de API, ya sea RESTful, SOAP, GraphQL, etc., y probar su funcionalidad y desempeño. La herramienta es útil para simplificar y agilizar el proceso de pruebas de API, ya que permite automatizar pruebas y verificar el funcionamiento de las APIs de manera rápida y fácil.

Además, Postman cuenta con una interfaz gráfica de usuario (GUI) que permite a los desarrolladores visualizar y manipular datos en formato JSON, XML y otros formatos. También ofrece características útiles como la documentación de API, la colaboración y la integración con otras herramientas de desarrollo, lo que ayuda a los equipos de desarrollo a trabajar de manera más eficiente y colaborativa.

curl

Qué es? cURL es una librería de funciones para conectar con servidores para trabajar con ellos. El trabajo se realiza con formato URL. Es decir, sirve para realizar acciones sobre archivos que hay en URLs de Internet, soportando los protocolos más comunes, como http, ftp, https, etc.

- Es una herramienta multiplataforma.
- Soporta los protocolos FTP, FTPS, HTTP, HTTPS, TFTP, SCP, SFTP, Telnet, DICT, FILE y LDAP, entre otros.
- Es un proyecto de software consistente en una biblioteca (libcurl) y un intérprete de comandos (curl) orientado a la transferencia de archivos.
- Es una herramienta válida para simular las acciones de usuarios en un navegador web.
- LibcURL es la biblioteca/API correspondiente que los usuarios pueden incorporar en sus programas donde cURL actúa como un envoltorio (wrapper) aislado para la biblioteca LibcURL.7 LibcURL se usa para proveer capacidades de transferencia de URL a numerosas aplicaciones, tanto libres y open source como privativas. La biblioteca "libcurl" puede ser usada desde más de 30 lenguajes distintos.

Descarga: <https://curl.se/download.html>

```
curl --location --request GET 'https://restcountries.eu/rest/v2/name/bangladesh'
```

Desde postman podemos obtener el comando curl equivalente:

The screenshot shows a Postman REST client interface. The request is a GET to `https://restcountries.com/v2/name`. The response is a 200 OK with a status of 200, a time of 1630 ms, and a body size of 287 B. The response body is displayed in JSON format, showing an array with one object for Mexico: `{ "name": "Mexico", "independent": false }`.

Ejercitación

Ejercicio: Postman RESTful API REST COUNTRIES

El ejercicio consiste en utilizar la herramienta Postman para consumir API RESTful REST Countries. Esta API proporciona información sobre países de todo el mundo, incluyendo detalles como la población, el idioma oficial y la moneda utilizada.

Para realizar este ejercicio, sigue los siguientes pasos:

1. Descarga e instala Postman en tu ordenador si aún no lo has hecho.
2. Abre Postman.
3. Crea una colección con un nombre significativo, por ejemplo: "Desarrollo_de_Software"
4. Crea una nueva solicitud haciendo clic en el botón "Nuevo" en la esquina superior izquierda y seleccionando "Solicitud".

The screenshot shows a Postman REST client interface. The request is a GET to `https://restcountries.com/v2/all`. The response is a 200 OK with a status of 200, a time of 1163 ms, and a body size of 58.09 KB. The response body is displayed in JSON format, showing an array with one object for Afghanistan: `{ "name": "Afghanistan", "topLevelDomain": [".af"], "alpha2Code": "AF", "alpha3Code": "AFG", "callingCodes": [...] }`.

5. En la ventana de solicitud, ingresa la URL de la API REST Countries:

<https://restcountries.com/v2/all>

6. Selecciona el método HTTP GET y haz clic en "Enviar".

restcountries / <https://restcountries.com/v2/all>

GET <https://restcountries.com/v2/all> **Send**

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

Query Params

Key	Value	Description	...	Bulk Edit
Key	Value	Description		

Body Cookies Headers (7) Test Results 200 OK 1140 ms 58.09 KB Save as Example

Pretty Raw Preview Visualize JSON

```

1  {
2    "name": "Afghanistan",
3    "topLevelDomain": [
4      ".af"
5    ],
6    "alpha2Code": "AF",
7    "alpha3Code": "AFG",
8    "callingCodes": [
9      "93"
10   ]
11 }

```

Verás la respuesta de la API en la ventana de respuesta de Postman. La respuesta es un arreglo de objetos, donde cada objeto representa un país y contiene información como el nombre, la población, el idioma, la moneda y la bandera.

7. Podes agregar parámetros a la solicitud para obtener información específica de un país. Por ejemplo, para obtener información sobre México, cambia la URL de la solicitud a:

<https://restcountries.com/v2/name/mexico>

Selecciona el método HTTP GET y haz clic en "Enviar". Verás la respuesta de la API con información específica de México.

restcountries / **GET un pais particular** **Save** **Send**

GET <https://restcountries.com/v2/name/mexico> **Send**

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

Query Params

Key	Value	Description	...	Bulk Edit
Key	Value	Description		

Body Cookies Headers (8) Test Results 200 OK 835 ms 977 B Save as Example

Pretty Raw Preview Visualize JSON

```

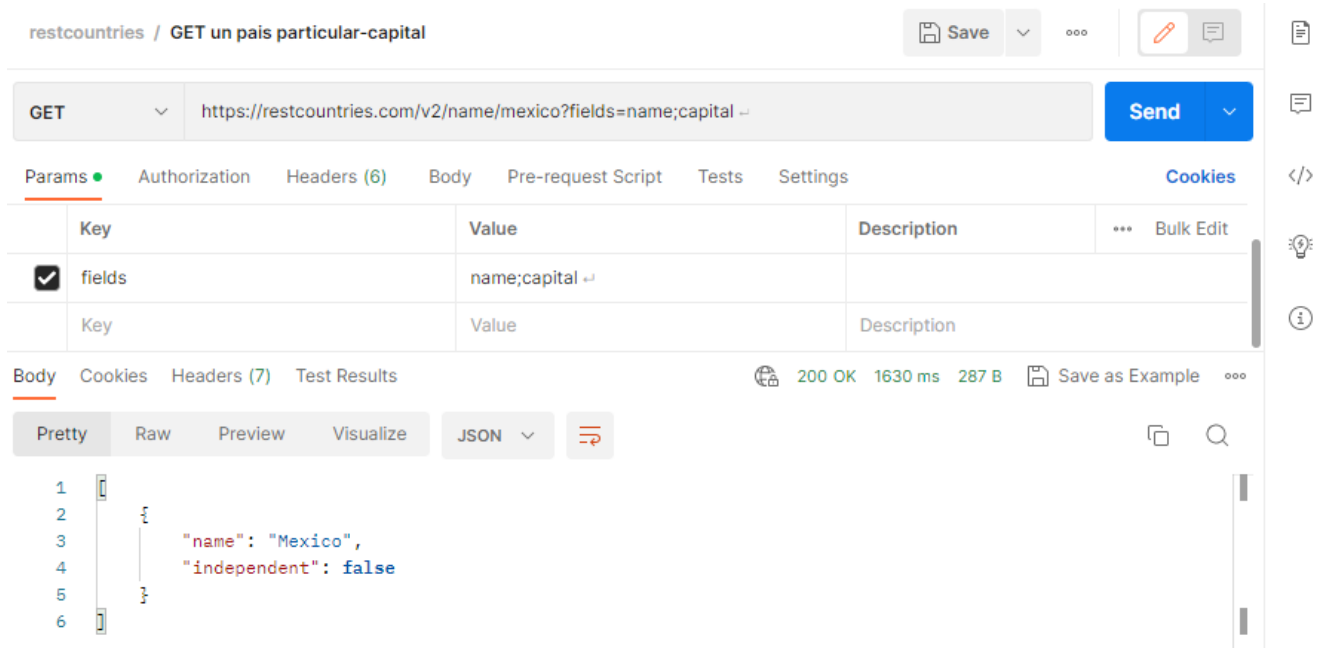
1  {
2    "name": "Mexico",
3    "topLevelDomain": [
4      ".mx"
5    ],
6    "alpha2Code": "MX",
7    "alpha3Code": "MEX",
8    "callingCodes": [
9      "52"
10   ],
11    "capital": "Mexico City",
12    "altSpellings": [
13

```

8. También puedes agregar más parámetros para filtrar la información que se devuelve en la respuesta. Por ejemplo, para obtener solo el nombre y la capital de México, cambia la URL de la solicitud a:

<https://restcountries.com/v2/name/mexico?fields=name;capital>

Selecciona el método HTTP GET y haz clic en "Enviar". Verás la respuesta de la API con solo el nombre y la capital de México.



Has utilizado Postman para consumir la API REST Countries y obtener información sobre países de todo el mundo. Puedes explorar más la API y probar diferentes solicitudes para obtener información específica de los países que te interesen.

Ejercitación extra

Algunos ejercicios de práctica de Postman con la API RestCountries.

- URL: <https://restcountries.com/v3.1/name/united states>
 Obtener información sobre varios países a la vez (por ejemplo, Estados Unidos y Canadá)
 Método: GET
- URL: <https://restcountries.com/v3.1/name/united states;canada>
 Buscar países por subregión (por ejemplo, Europa occidental)
 Método: GET
- URL: <https://restcountries.com/v3.1/subregion/western europe>
 Buscar países por moneda (por ejemplo, dólar estadounidense)
 Método: GET
- URL: <https://restcountries.com/v3.1/currency/usd>
 Buscar países por idioma (por ejemplo, español)
 Método: GET
- URL: <https://restcountries.com/v3.1/lang/es>
 Obtener información sobre la capital de un país en particular (por ejemplo, España)
 Método: GET
- URL: <https://restcountries.com/v3.1/name/spain?fields=capital>
 Obtener información sobre la población de un país en particular (por ejemplo, China)
 Método: GET
- URL: <https://restcountries.com/v3.1/name/china?fields=population>
 Obtener información sobre el área de un país en particular (por ejemplo, Australia)
 Método: GET

- URL: <https://restcountries.com/v3.1/name/australia?fields=area>

Obtener información sobre el idioma oficial de un país en particular (por ejemplo, Canadá)

Método: GET

Ejercicio: Desarrollo API Hello World

Vamos a construir una API Hello Word, seguí estos pasos:

1. Abri Visual Studio Code en tu computadora y asegúrate de tener instalado Node.js.
2. Crea una nueva carpeta "API_Hello_Word" en tu computadora donde puedas guardar el archivo del código que vas a construir.
3. Selecciona la opción "Abrir carpeta" en la página de inicio. Navega a la carpeta que creaste en el paso 2 y abrila.
4. Crea un nuevo archivo en Visual Studio Code y guárdalo con el nombre "index.js" (asegúrate de que la extensión sea ".js").
5. Copia y pega el siguiente código en el archivo "index.js":

```
const http = require('http');

const hostname = '127.0.0.1';
const port = 3000;

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello World DDS!!!');
});

server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`);
});
```

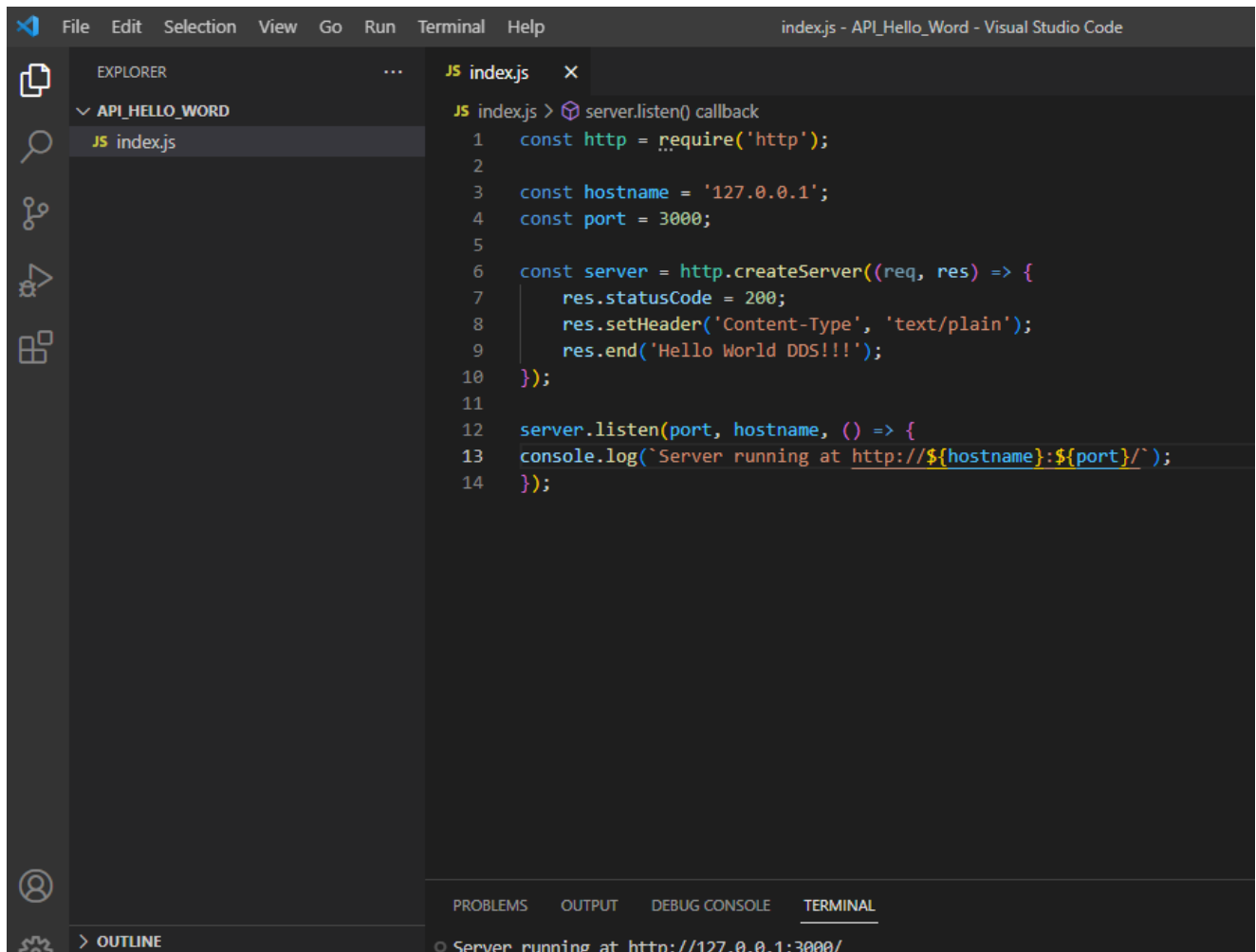
El módulo http se utiliza en node para gestionar el protocolo http:

```
const http = require('http');
```

La constante http ahora puede gestionar peticiones HTTP. La función (anónima) que recibe http.createServer se va a ejecutar cada vez que hay una petición al puerto de escucha.

El servicio http comienza a escuchar en el puerto y hostname configurado con server.listen(port, hostname, () => {...});

6. Abri una terminal en Visual Studio Code. Para hacerlo, selecciona "Terminal" en la barra de menú superior y luego selecciona "Nueva terminal".
7. En la terminal, asegúrate de estar ubicado en la carpeta donde guardaste el archivo "index.js". Para hacerlo, utiliza el comando "cd" seguido de la ruta de la carpeta "API_Hello_Word".
8. Ejecuta el comando "node index.js" en la terminal para ejecutar el código. Si todo está bien, vas a ver un mensaje en la consola que dice "Server running at http://127.0.0.1:3000/".



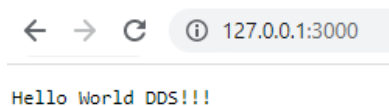
```
index.js - API_Hello_Word - Visual Studio Code

EXPLORER
  API_HELLO_WORD
    JS index.js

JS index.js
1  const http = require('http');
2
3  const hostname = '127.0.0.1';
4  const port = 3000;
5
6  const server = http.createServer((req, res) => {
7    res.statusCode = 200;
8    res.setHeader('Content-Type', 'text/plain');
9    res.end('Hello World DDS!!!');
10 });
11
12 server.listen(port, hostname, () => {
13   console.log(`Server running at http://${hostname}:${port}/`);
14 });

TERMINAL
Server running at http://127.0.0.1:3000/
```

9. Abri tu navegador web y escribe la dirección "<http://127.0.0.1:3000/>" en la barra de direcciones. Deberías ver el mensaje "Hello World DDS!!!" en la pantalla.



Bibliografía