

Clase 10

- Clase 10
 - API RESTful (*)
 - Beneficios API RESTful
 - Principios de diseño de API Restful (*)
 - Manténgalo simple (Keep it simple) (*)
 - Usar sustantivos y no verbos (*)
 - Convención nombres (Naming Conventions) (*)
 - Uso de los métodos HTTP adecuados (*)
 - Usar plurales (*)
 - Utilizar parámetros (*)
 - Utilice códigos HTTP adecuados (*)
 - Versiones (*)
 - Usar paginación (*)
 - Formatos soportados (*)
 - Utilizar mensajes de error adecuados (*)
 - Documentación de API Rest: Swagger (*)
 - ¿Qué es swagger? (*)
 - ¿Para quién documentamos las APIs?
 - ¡APIs famosas!
 - Herramientas de desarrollo: Postman/Curl/VSCode (*)
 - Postman (*)
 - curl (*)
 - VSCode (*)
 - Ejercitación
 - Ejercicio: Postman RESTful API REST COUNTRIES (*)
 - Ejercicio: Desarrollo API Hello World (*)
 - Ejercicio: Ejemplo simple para obtener un json de una api con fetch (*)
 - Ejercicio: Ejemplo simple para obtener un json de una api con axios (*)

API RESTful (*)

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:



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.

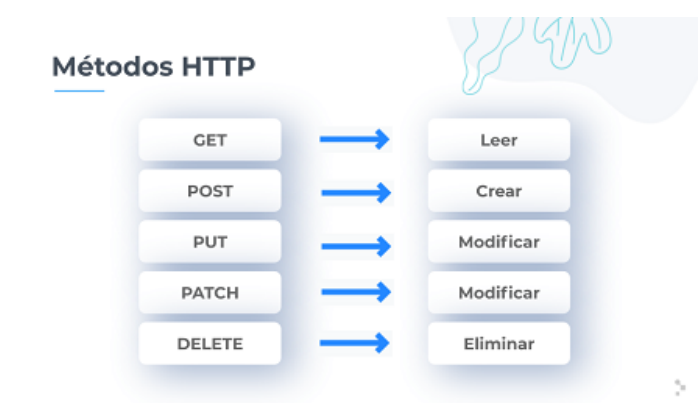
Para APIs utilizaremos **spinal-case** como convención de nombre Urls de APIs. Ejemplo: para la API unidades de negocio, la Url sería /business-unit/12345.

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.

Método HTTP	Descripción
PUT	Para actualizar todos los atributos de un recurso o conjunto de recursos existentes.
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.

1

/products

1

/product

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 Parametro	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.
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.

Status Code	Descripción
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.

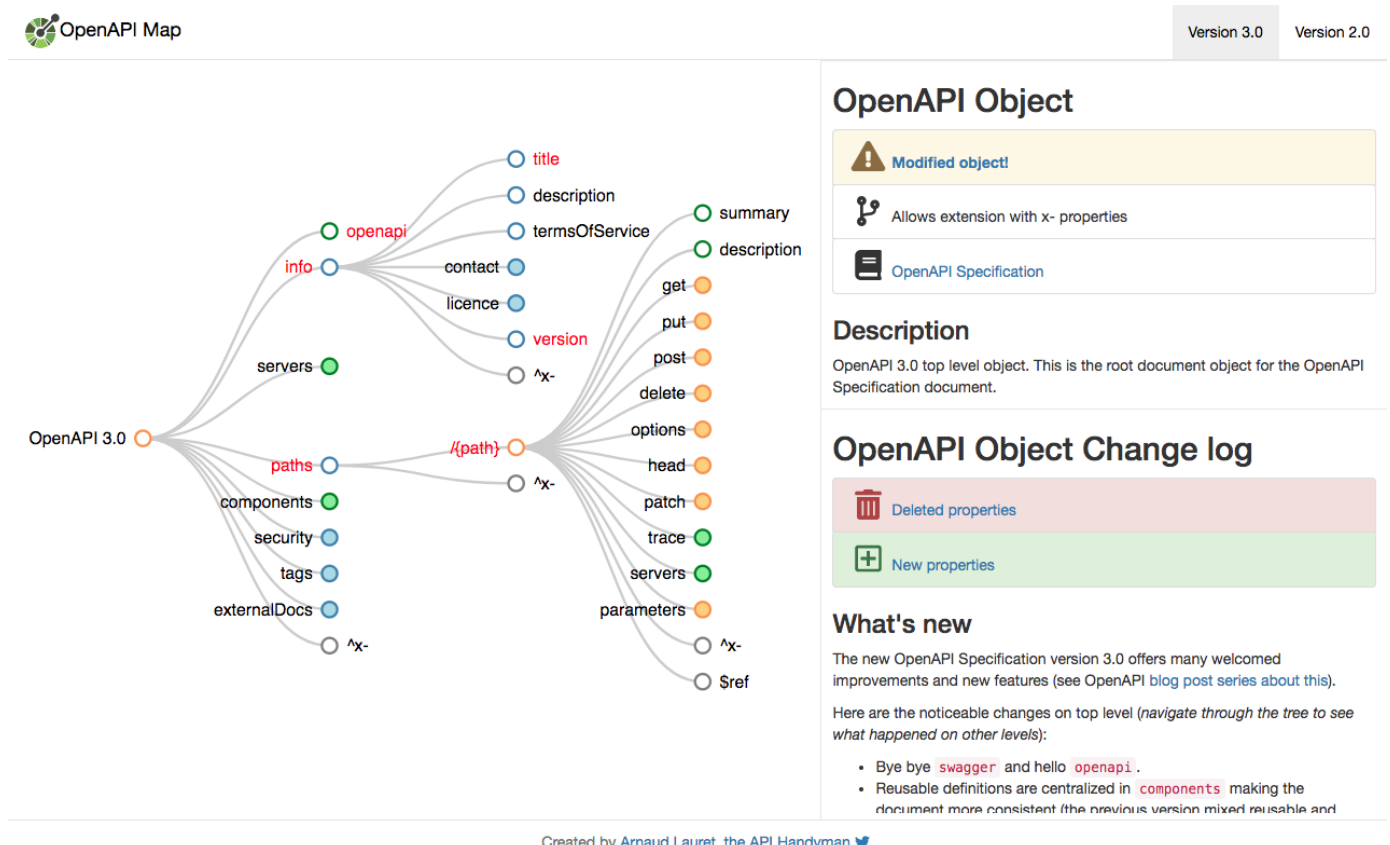
Por ejemplo, si utilizas APIs de gráficos de Facebook, en caso de errores, devuelve un mensaje como este:

```
{
  "error": {
    "message": "(#803) Some of the aliases you requested do not exist: products",
    "type": "OAuthException",
    "code": 803,
    "fbtrace_id": "FOXX2AhLh80"
  }
}
```

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:
-



Ejercitación

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>.

- 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.
- 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. Tené 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

ExecuteClear

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

¿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





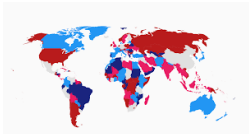

Newcomer:
First-Time User
(e.g. Front End Dev)

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 the Postman interface. At the top, the request method is 'GET' and the URL is 'https://restcountries.com/v2/name'. The 'Send' button is visible. Below the URL bar, the 'Params' section is expanded, showing a table with one parameter:

Key	Value	D...	...	Bulk Edit
fields	name;capital			

Below the params table, the 'Body' section is expanded, showing the response in 'Pretty' JSON format:

```
{
  "name": "Mexico",
  "independent": false
}
```

On the right side, the 'Code snippet' panel is open, showing the equivalent curl command:

```
curl --location 'https://restcountries.com/v2/name/mexico?fields=name%3Bcapital%0A'
```

VSCode (*)

Visual Studio Code es un editor potente y en gran parte por las extensiones. Las extensiones nos permiten personalizar y agregar funcionalidad adicional de forma modular y aislada. Descarga: <https://code.visualstudio.com/>

Ejercitación

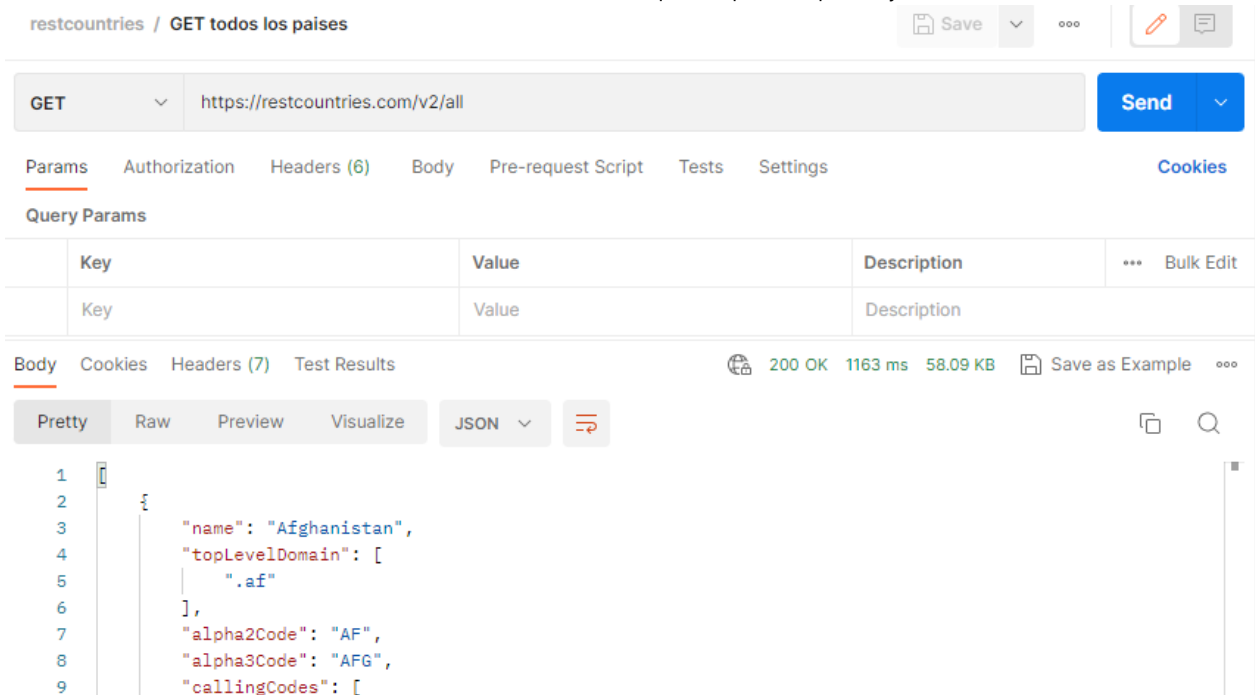
Ejercicio: Postman RESTful API REST COUNTRIES (*)

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:

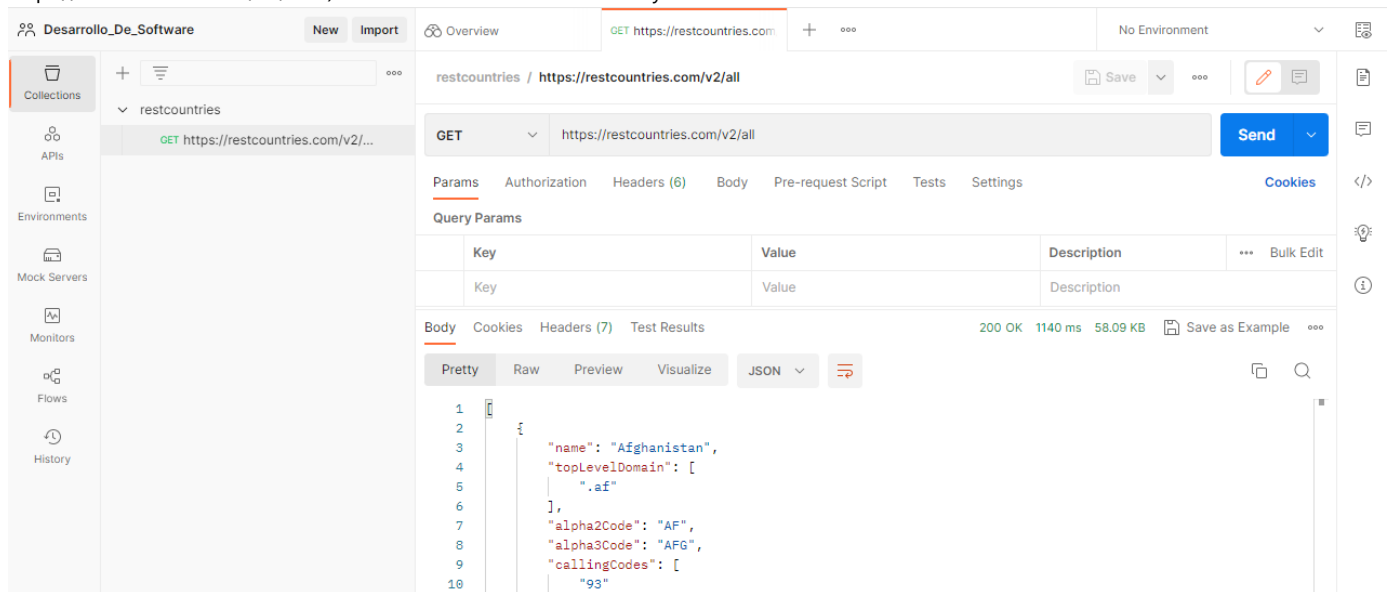
1. Descarga e instala Postman en tu ordenador si aún no lo has hecho.
2. Abre Postman.
3. Crea una colección con un nombre significativo, por ejemplo: "Desarrollo_de_Software"

4. Crea una nueva solicitud haciendo clic en el botón "Nuevo" en la esquina superior izquierda y seleccionando "Solicitud".



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".



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

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.

restcountries / GET un pais particular-capital

GET

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

Send

Params

Authorization

Headers (6)

Body

Pre-request Script

Tests

Settings

Cookies

	Key	Value	Description	...	Bulk Edit
<input checked="" type="checkbox"/>	fields	name;capital			
	Key	Value	Description		

Body

Cookies

Headers (7)

Test Results

200 OK

1630 ms

287 B

Save as Example

Pretty

Raw

Preview

Visualize

JSON

```

1  {
2    "name": "Mexico",
3    "independent": false
4  }
5
6

```

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%20states Obtener información sobre varios países a la vez (por ejemplo, Estados Unidos y Canadá) Método: GET URL: https://restcountries.com/v3.1/name/united%20states;canada Buscar países por subregión (por ejemplo, Europa occidental) Método: GET URL: https://restcountries.com/v3.1/subregion/western%20europe 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 URL: <https://restcountries.com/v3.1/name/canada?fields=languages.official>

Ejercicio: Desarrollo API Hello World (*)

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

1. Abrí 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 ábrela.
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!!!');
});

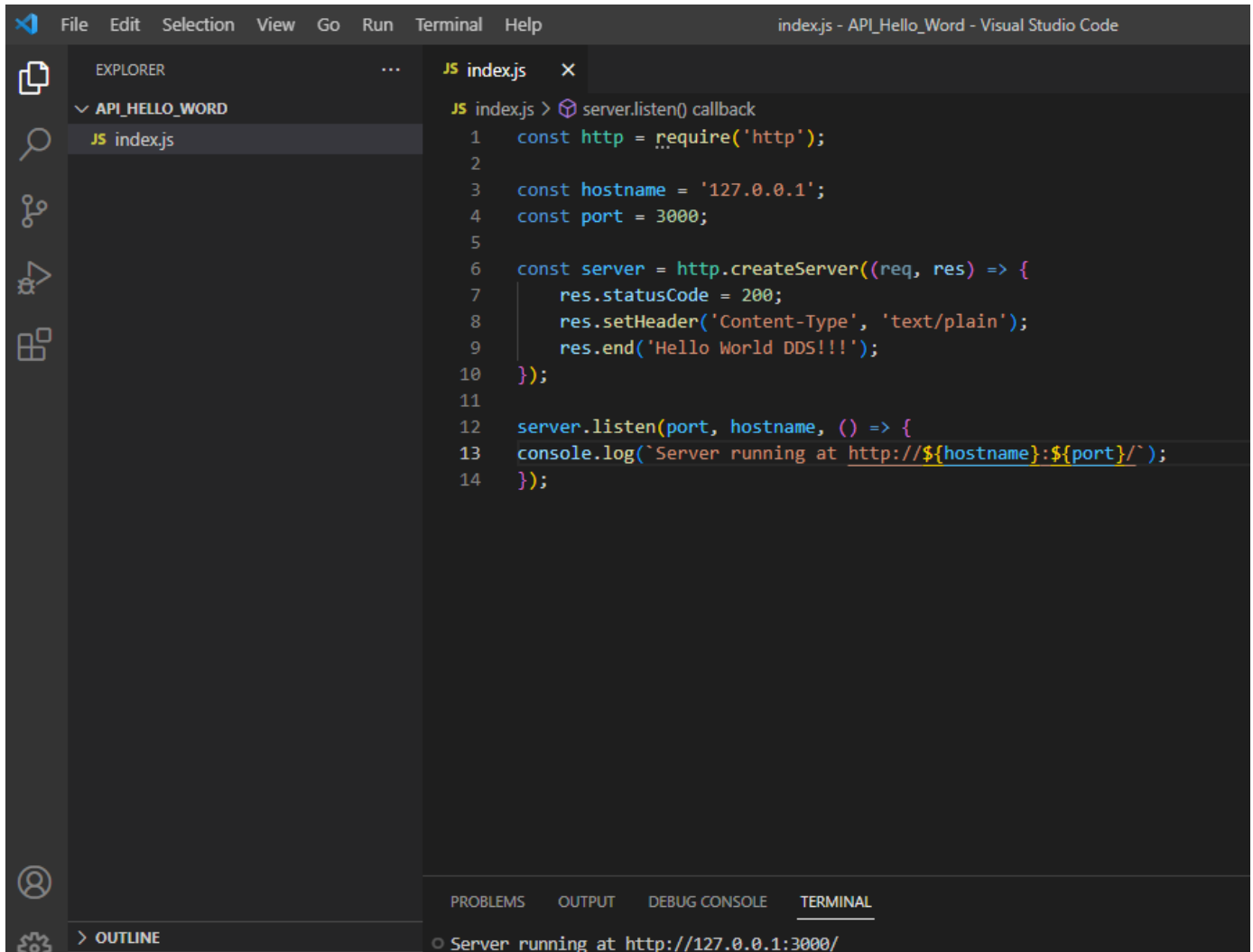
server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`);
});
```

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. Abrí 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, asegurate 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/".



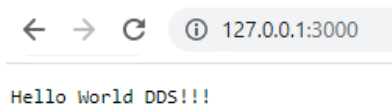
```
File Edit Selection View Go Run Terminal Help
index.js - API_Hello_Word - Visual Studio Code

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 });

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Server running at http://127.0.0.1:3000/
```

9. Abrió 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.



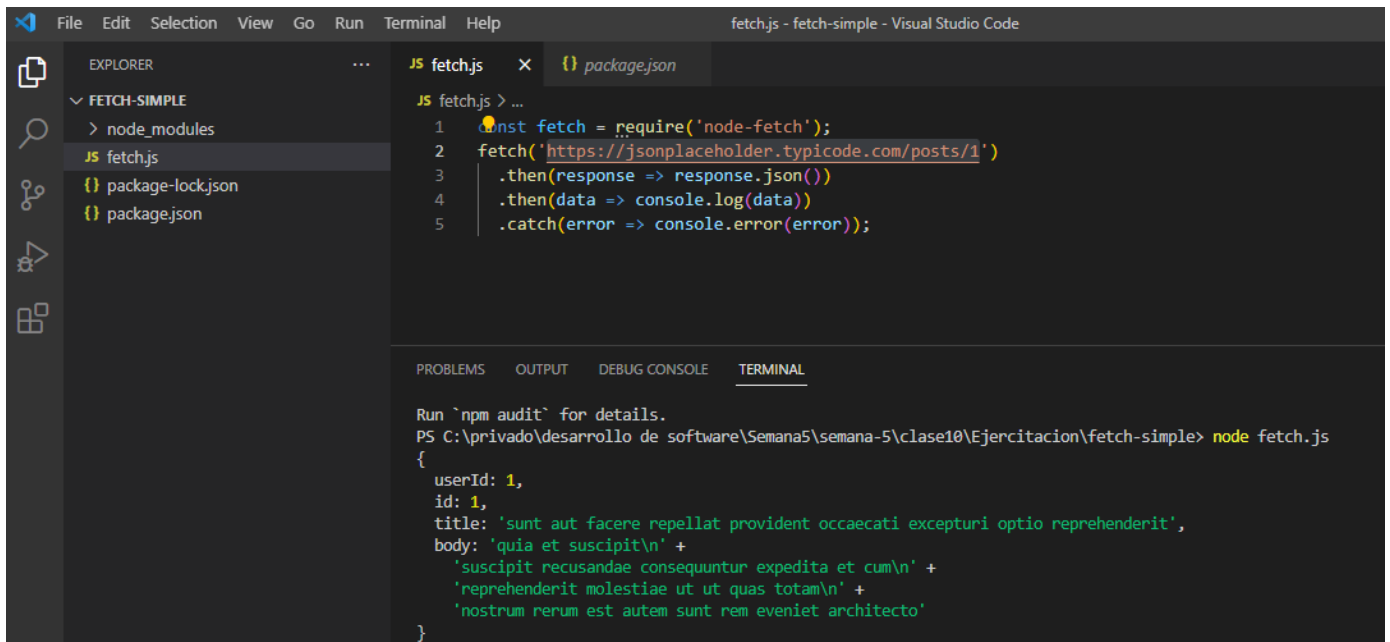
Ejercicio: Ejemplo simple para obtener un json de una api con fetch (*)

Este es un ejemplo básico para usar fetch en Node.js con Visual Studio

1. Abrió Visual Studio Code en tu computadora y asegúrate de tener instalado Node.js.
2. Crea una nueva carpeta "fetch-simple" en tu computadora donde puedas guardar el archivo del código que vas a construir.
3. Instalá la librería fetch: abrí la terminal de Visual Studio Code y navegá hasta la carpeta del proyecto. Ejecutá el siguiente comando: `npm i node-fetch@2.6.1` o `npm i node-fetch@2.6.7`
4. Creá un archivo para realizar la petición con fetch: Creá un nuevo archivo en la carpeta del proyecto y nombrarlo fetch.js. En el archivo, importá la librería node-fetch y escribí el código para realizar la petición usando fetch. Por ejemplo, para obtener los datos de una API de prueba:

```
const fetch = require('node-fetch'); fetch('https://jsonplaceholder.typicode.com/posts/1') .then(response => response.json()) .then(data => console.log(data)) .catch(error => console.error(error));
```

5. Guardá el archivo.
6. Ejecutá el archivo desde la terminal: Navegá hasta la carpeta del proyecto en la terminal. Ejecutá el comando: `node fetch.js` Debería mostrarse en la consola la respuesta de la API.



```
File Edit Selection View Go Run Terminal Help fetch.js - fetch-simple - Visual Studio Code

EXPLORER
  FETCH-SIMPLE
    > node_modules
    JS fetch.js
    {} package-lock.json
    {} package.json

JS fetch.js > ...
1 const fetch = require('node-fetch');
2 fetch('https://jsonplaceholder.typicode.com/posts/1')
3   .then(response => response.json())
4   .then(data => console.log(data))
5   .catch(error => console.error(error));

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Run `npm audit` for details.
PS C:\privado\desarrollo de software\Semana5\semana-5\clase10\Ejercitacion\fetch-simple> node fetch.js
{
  userId: 1,
  id: 1,
  title: 'sunt aut facere repellat provident occaecati excepturi optio reprehenderit',
  body: 'quia et suscipit\n' +
  'suscipit recusandae consequuntur expedita et cum\n' +
  'reprehenderit molestiae ut ut quas totam\n' +
  'nostrum rerum est autem sunt rem eveniet architecto'
}
```

Ejercicio: Ejemplo simple para obtener un json de una api con axios (*)

1. Abri Visual Studio Code en tu computadora y asegúrate de tener instalado Node.js.
2. Crea una nueva carpeta "axios-simple" en tu computadora donde puedas guardar el archivo del código que vas a construir.
3. Instalá la librería axios: abrí la terminal de Visual Studio Code y navegá hasta la carpeta del proyecto. Ejecutá el siguiente comando:

```
npm install axios
```

4. Creá un archivo JavaScript para realizar la petición con axios: Creá un nuevo archivo en la carpeta del proyecto y nombrarlo axios.js.
5. En el archivo, importá la librería axios y escribí el código para realizar la petición usando axios. Por ejemplo, para obtener los datos de una API de prueba:

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

axios.get('https://jsonplaceholder.typicode.com/posts/1')
  .then(response => {
    console.log(response.data);
  })
  .catch(error => {
    console.error(error);
  });
```

6. Guardá el archivo.
7. Ejecutá el archivo desde la terminal: Navegá hasta la carpeta del proyecto en la terminal. Ejecutá el comando:

```
node axios.js
```

8. Debería mostrarse en la consola la respuesta de la API.

Referencias:

- <https://aws.amazon.com/es/what-is/restful-api/>
- <https://www.ma-no.org/es/programacion/como-construir-una-api-restful-guia-paso-a-paso>