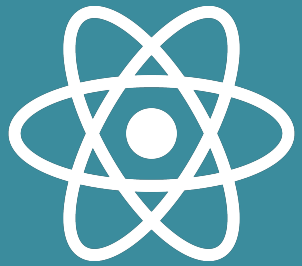
www.cesarguerra.mx



Ciclo de Vida y Llamadas a API

Ciclo de Vida de Componentes Funcionales

CONTENIDO

01

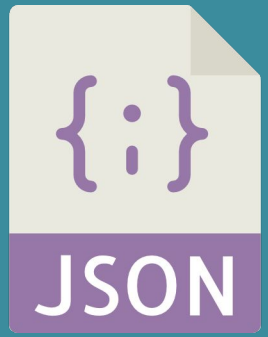
JSON

"JASON" != "JSON"

Cuando inicies a trabajar con bases de datos, escucharás mucho mencionarse **JSON (JavaScript Object Notation)**.

Se utiliza como medio de comunicación entre el Frontend y Backend.

Aunque se parece mucho a la sintaxis literal de un objeto JavaScript, **se puede utilizar independientemente de JavaScript, muchos programas pueden leer y generar documentos JSON.**



JSON

Es formato para el intercambios de datos, básicamente JSON describe los datos con una sintaxis dedicada que se usa para identificar y gestionar los datos. **JSON nació como una alternativa a XML.**

XML vs JSON

XML

```
<empinfo>
   <employees>
      <employee>
         <name>James Kirk</name>
         <age>40></age>
      </employee>
      <employee>
         <name>Jean-Luc Picard</name>
         <age>45</age>
      </employee>
      <employee>
         <name>Wesley Crusher</name>
         <age>27</age>
      </employee>
   </employees>
</empinfo>
```

JSON

```
{  "empinfo" :
      {
            "employees" :  [
            {
                  "name" : "James Kirk",
                  "age" : 40,
            },
            {
                  "name" : "Jean-Luc Picard",
                  "age" : 45,
            },
            {
                  "name" : "Wesley Crusher",
                  "age" : 27,
            }
                              ]
      }
}
```

¿JSON y Objetos?

La sintaxis de JSON se inspiró en la notación literal del objeto JavaScript, pero existen diferencias entre ellos:

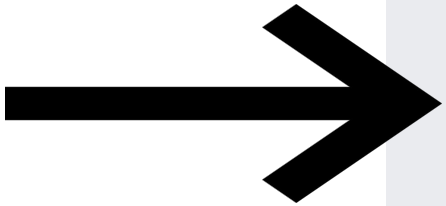
- En JSON todas claves deben estar entre comillas.
- No podemos agregar comentarios a un JSON
- No podemos colocar ;
- Solamente utiliza comillas simples o dobles en un JSON, no las alternes.
- Las comas toman una importancia mayor

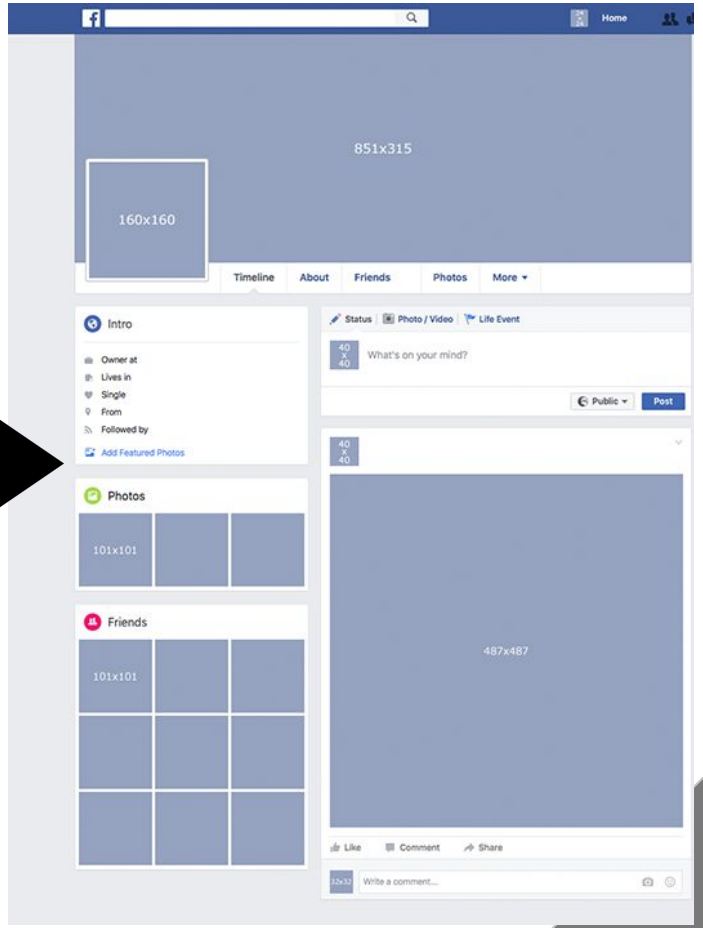
```
// JSON:
{ "foo": "bar" }

// Object literal:
var o = { foo: "bar" };
```

```
// Invalid JSON:
{ "foo": 'bar' }
```

JSON Y FRONTEND


```
{
    "_id": {
        "$oid": "5c3fd50c680caf5787cd11bb"
    },
    "is_important": false,
    "applied_codes": [],
    "is_active": true,
    "first_name": "Raul",
    "last_name": "Prueba Pruebita",
    "phone": "7821076070",
    "email": "dagorik@gmail.com",
    "approaches": [⟷],
    "programs": [],
    "createdAt": {⟷},
    "updatedAt": {
        "$date": "2019-01-28T23:28:11.470Z"
    },
    "__v": 1
}
```



CésarGuerra

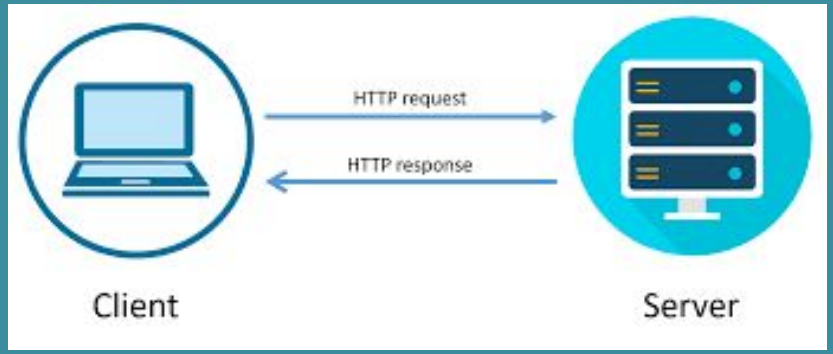
02

PETICIONES

Mensajes HTTP

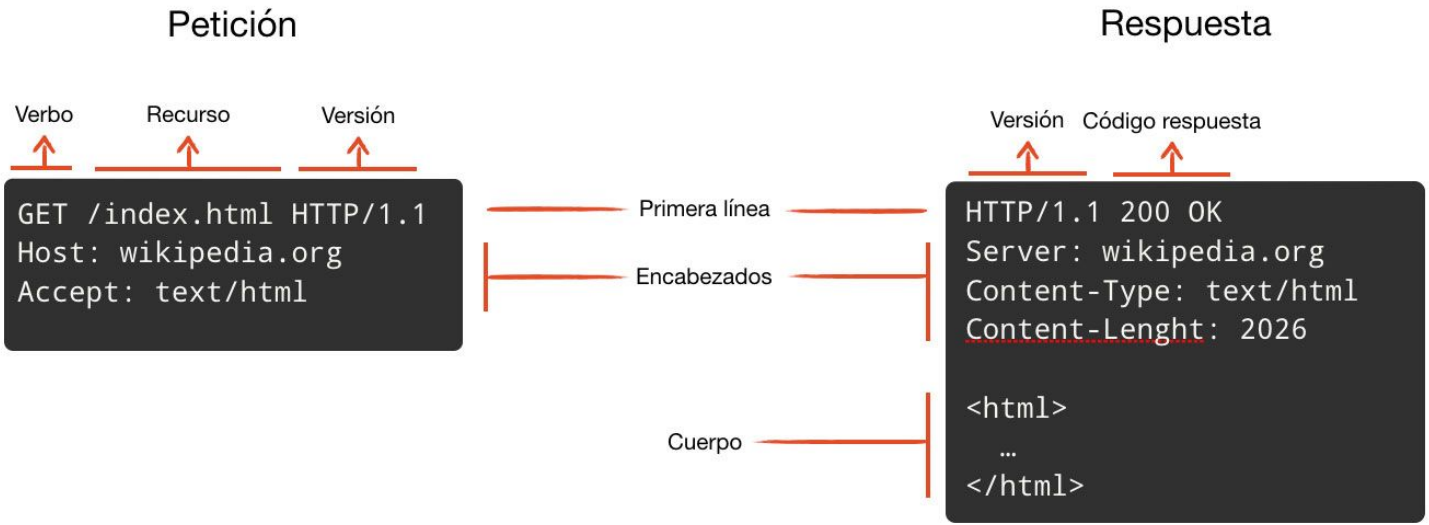
Los mensajes HTTP, son los medios por los cuales se intercambian datos entre servidores y clientes.

Hay dos tipos de mensajes: peticiones, enviadas por el cliente al servidor, para pedir el inicio de una acción; **y respuestas**, que son la respuesta del servidor.



Mensajes HTTP

La línea de inicio y las cabeceras HTTP, del mensaje, son conocidas como la cabeza de la peticiones, mientras que su contenido en datos se conoce como el cuerpo del mensaje.

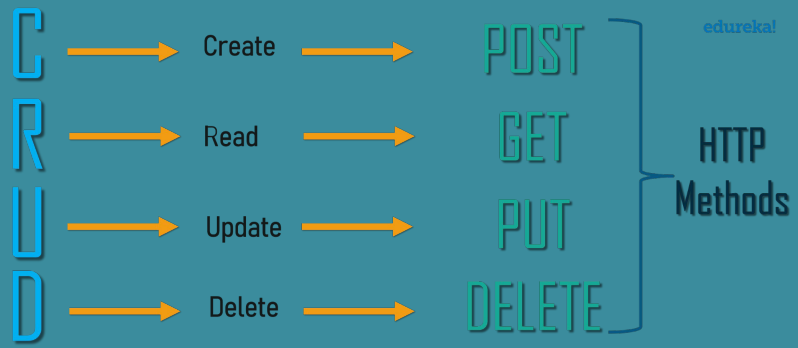


VERBOS HTTP

La primera línea de un mensaje de petición empieza con un verbo (también se le conoce como método). **Los verbos definen la acción que se quiere realizar sobre el recurso.**

Los verbos más comunes son:

- **GET:** Solicitar un recurso.
- **POST:** Publicar un recurso.
- **PUT:** Reemplazar un recurso.
- **DELETE:** Eliminar un recurso.
- **PATCH:** Actualizar un recurso



Nota: Cuando ingresas a una página desde un navegador, por debajo el navegador envía un mensaje GET, lo mismo cuando oprimes un vínculo a otra página

Códigos de Respuesta

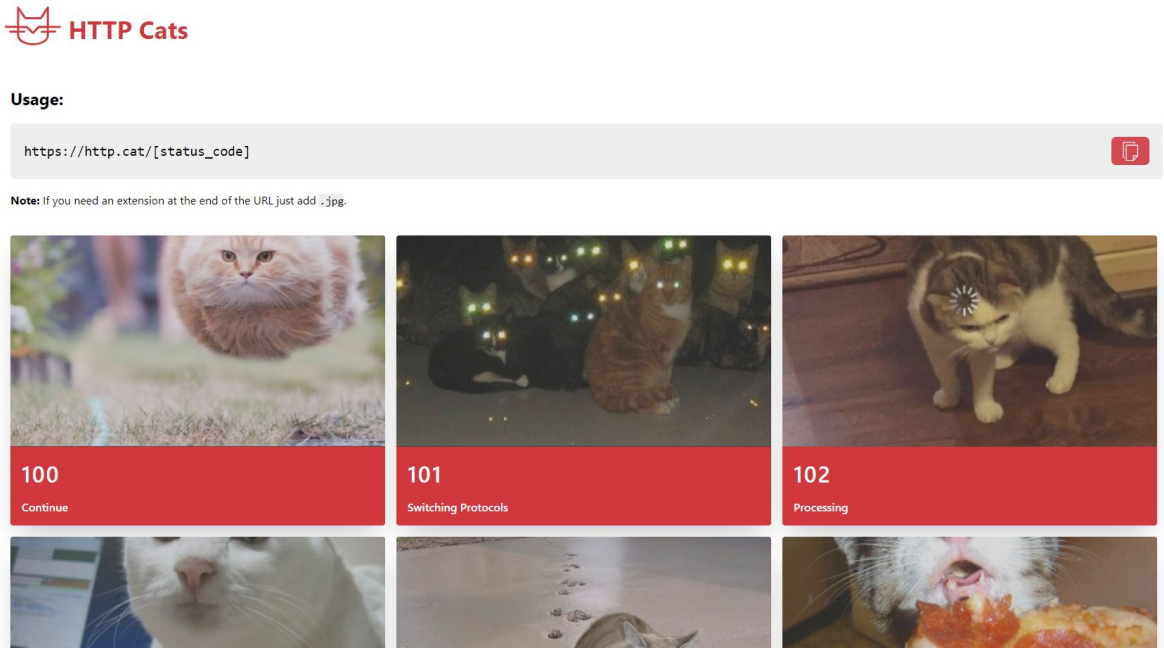
La primera línea de un mensaje de respuesta tiene un código de 3 dígitos que le indica al cliente cómo interpretar la respuesta.

Los códigos de respuesta se dividen en cinco categorías dependiendo del dígito con el que inician:

- **1XX:** Información
- **2XX:** Éxito
- **3XX:** Redirección
- **4XX:** Error en el cliente
- **5XX:** Error en el servidor

Mensajes HTTP

https://http.cat/



03

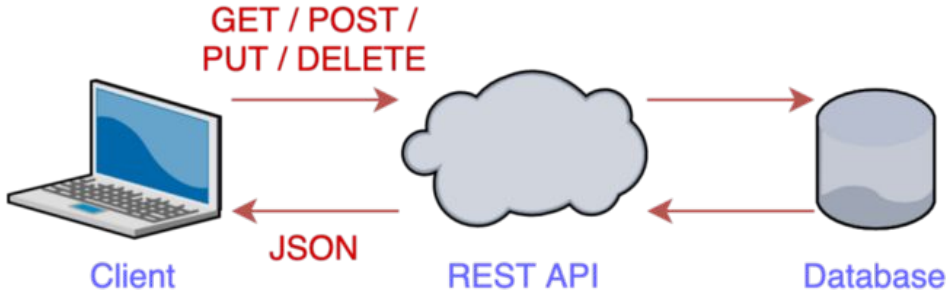
API REST

API + REST = API REST

API REST

API Rest es el conjunto de buenas prácticas utilizadas en las peticiones HTTP realizadas por una API en una aplicación web (GET, POST, etc).

Es decir, cuando se habla de API Rest, significa utilizar una API para acceder a aplicaciones back-end, de manera que esa comunicación se realice con los estándares definidos por el estilo de arquitectura Rest.



Nombrando Endpoints en API Rest

/users // lista todos los usuarios

/users/123 // lista a un usuario en específico

/users/123/orders // lista los pedidos de un usuario específico

/users/123/orders/0001 // lista una orden especifica de un usuario específico

ENDPOINTS Y VERBOS HTTP

Un mismo Endpoint, hara diferentes acciones dependiendo el verbo http usado para accederlo

Recuros / Estado	POST	GET	PUT	DELETE
/casas	Crea una casa	Devuelve una lista de todos las casas	Modifica casas	Borra todas las casas
/casas/123	error	Devuelve los detalles de la casa **123**	Modifica la casa	Borra la casa

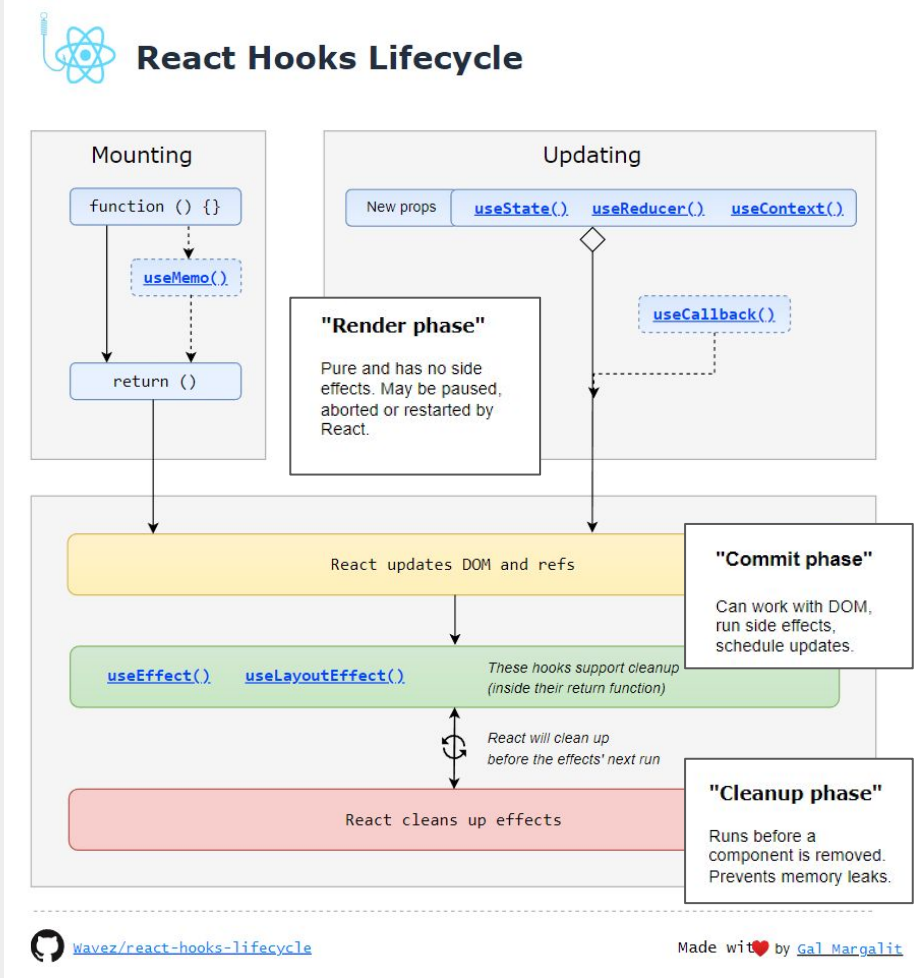
04

Ciclo de Vida

Ciclo de Vida de Hooks

El ciclo de vida de los componentes en React se refiere a ejecutar código en diferentes fases: montaje, actualización y desmontaje.

Con los **hooks** también podremos acceder a ese ciclo de vida en nuestros **functional components** usando **useEffect**.



05

UseEffect

USEEFFECT

Es un **hook** que **recibe como parámetro una función que se ejecutará cada vez que nuestro componente se renderice**, ya sea por un cambio de estado, por recibir props nuevas o, y esto es importante, porque es la primera vez que se monta.

Nota: Para usarlo es importante importarlo: **import { useEffect } from 'react';**

```
import React, { useEffect } from 'react';

function Ejemplo() {
  useEffect(function () {
    console.log('render!')
  },[])

  return <span>Ejemplo de useEffect</span>
}
```

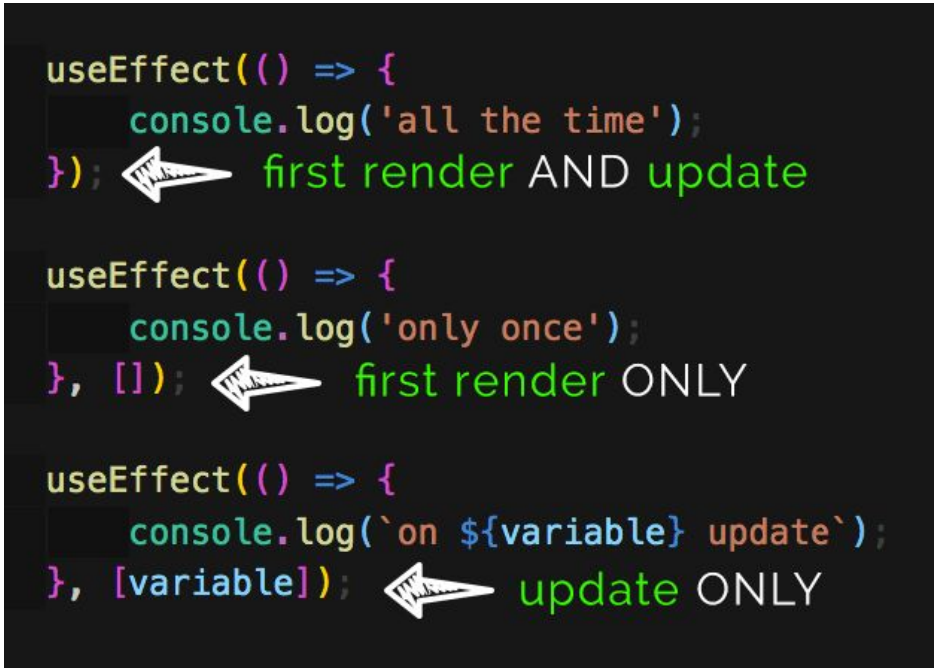
Ciclo de Vida de Hooks

Un functional component de React utiliza props y/o state para calcular la salida. Si el functional component realiza cálculos que no tienen como objetivo el valor de salida, estos cálculos se denominan **side-effects**.

Ejemplos de **side-effects** son las llamadas a API, la manipulación directa del DOM, el uso de funciones de temporización como setTimeout(), etc.

```
import { useEffect } from 'react';

function Saludar({ nombre }) {

  const mensaje = `Hola, ${nombre}!`;   // Calculates output
  useEffect(() => {
    document.title = `Saludos a ${nombre}`; // Side-effect!
  }, [nombre]);

  return <div>{mensaje}</div>;          // Calculates output
}
```

Formas de usar useEffect



DEMOSTRACIÓN