Esta clase va a ser

- grabada

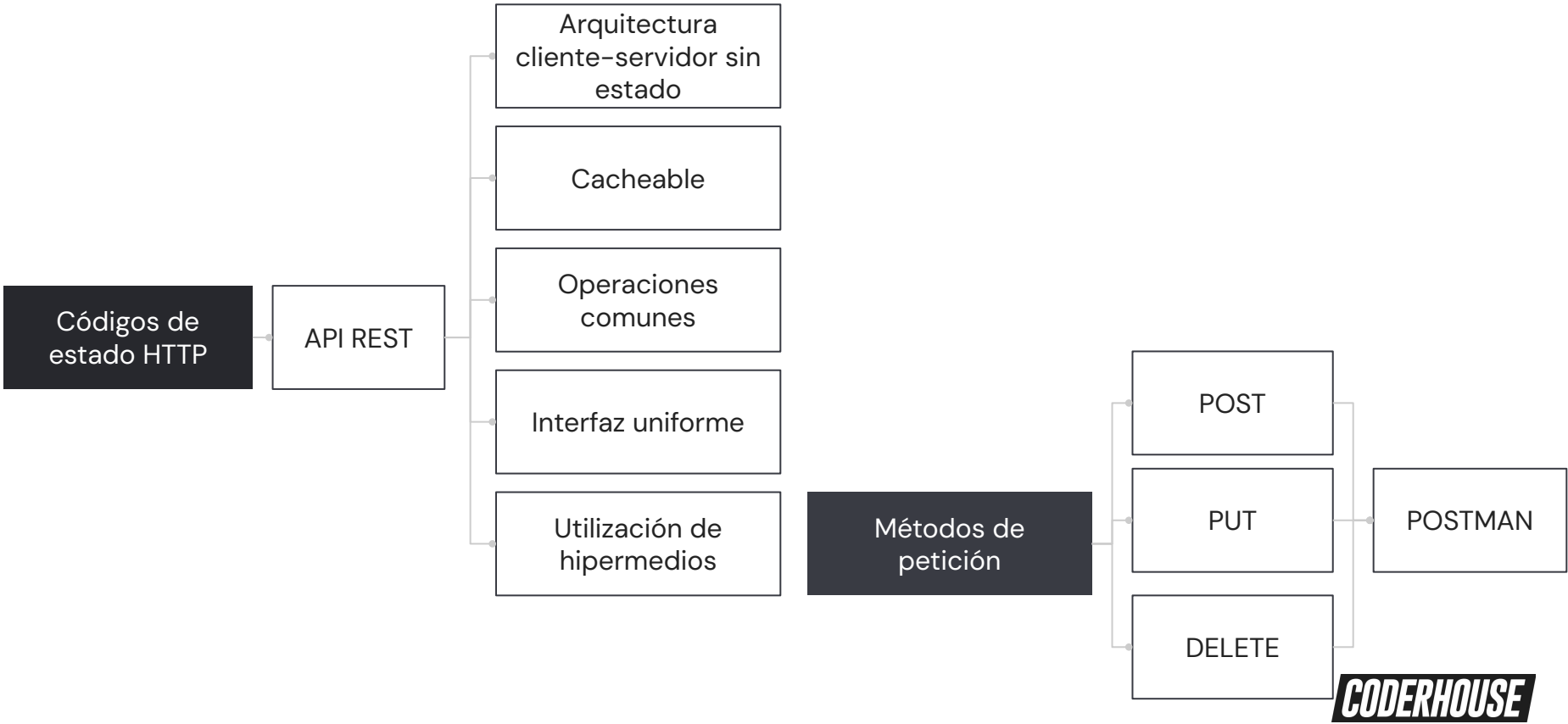
Clase 07. DESARROLLO AVANZADO DE BACKEND

Express avanzado

Objetivos de la clase

- **Conocer** los status del protocolo HTTP
- **Comprender** el concepto de API REST
- **Conocer** los métodos POST, PUT, DELETE y utilizarlos con POSTMAN o ThunderClient
- **Profundizar** de manera práctica sobre los métodos POST, PUT, DELETE

MAPA DE CONCEPTOS



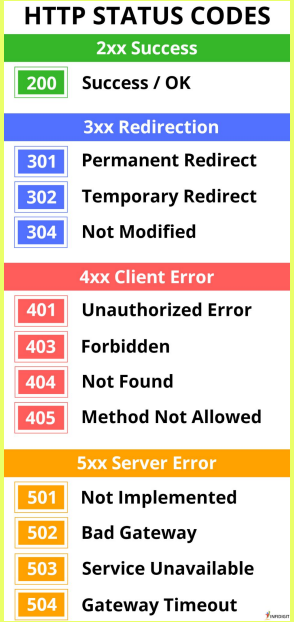
Códigos de estado en HTTP

Códigos de estado HTTP

¿Cómo funciona?

Cuando el servidor responde con un código de estado, esto permite saber qué ocurrió con la consulta que estábamos haciendo, y da información al cliente sobre qué ha ocurrido.

- ✓ **1xx: Status “informativo”**
- ✓ **2xx : Status “ok”**
- ✓ **3xx: Status de redirección.**
- ✓ **4xx: Status de error de cliente.**
- ✓ **5xx: Status de error en servidor.**



¿Sabías que...?

En 1998 Se agregó un código de estado 418 con significado “I’m a teapot” (Soy una tetera) como broma del día de los inocentes para el Hyper Text Coffee Pot Control Protocol. Este estado significa que el servidor se rehúsa a realizar la tarea solicitada, porque es una tetera.

Ver el documento oficial

¡Importante!

Recuerda que tú eres el desarrollador del servidor, entonces es tu responsabilidad reconocer cuándo colocar cada código de status. Si no configuramos nuestro servidor para devolver múltiples status, entonces será mucho más difícil rastrear los problemas.

Comprendiendo una API REST

API (Application Programming Interface)

Es un conjunto de definiciones y reglas que permiten que dos equipos puedan integrarse para trabajar juntos. La mejor analogía que hay para comprender ésto es que una API funge como un “contrato” entre el front y el back.

Veamos las etapas en el siguiente Genially

La API permite entonces que se respondan preguntas como:

- ✓ ¿A qué endpoint debo apuntar para la tarea que necesito?
- ✓ ¿Qué método debo utilizar para ese recurso?
- ✓ ¿Qué información debo enviar para realizar correctamente mi petición?

El cliente necesita algo del servidor, por lo que tiene que realizar una petición (request)

Para que la petición llegue correctamente al servidor, deberá apuntar al endpoint correcto, con el método correcto, con la información correcta

El servidor recibe la petición. Si se cumplieron todas las especificaciones de la API, el procesamiento se podrá llevar a cabo con éxito

Petición

Procesamiento

Respuesta

El cliente, al haber cumplido con lo que especificaba la API, podrá obtener su resultado satisfactoriamente y utilizarlo.

Resultado

Cumplir con el contrato de la API asegura (en la mayoría de los casos) que habrá un resultado satisfactorio

REST

Ya tenemos las reglas para comunicarse, ¿Pero qué tal la estructura del mensaje? Cuando hacemos una petición o cuando recibimos una respuesta, ésta debe tener un **formato**. REST (**RE**presentational **S**tate **T**ransfer) permite definir **la estructura** que deben tener los datos para poder transferirse.

La API respondía a preguntas sobre cómo comunicarse correctamente, sin embargo, REST define cómo debe ser el cuerpo del mensaje a transmitir. (puedes llegar a hablar con el presidente si cumples con el protocolo (HTTP) y las reglas (API), pero ¿de qué nos servirá si la forma en que estructuramos nuestro mensaje (REST) no es correcta?)

Los dos formatos más importantes son **JSON** y **XML**.
La utilización de la estructura dependerá de las necesidades del proyecto. Nosotros utilizaremos **JSON**. Como notarás, ¡un JSON parece un objeto! así que es mucho más amigable la sintaxis.

✓ XML

```xml
<factura>
  <cliente>Gomez</cliente>
  <emisor>Perez S.A.</emisor>
  <tipo>A</tipo>
  <items>
    <item>Producto 1</item>
    <item>Producto 2</item>
    <item>Producto 3</item>
  </items>
</factura>
```

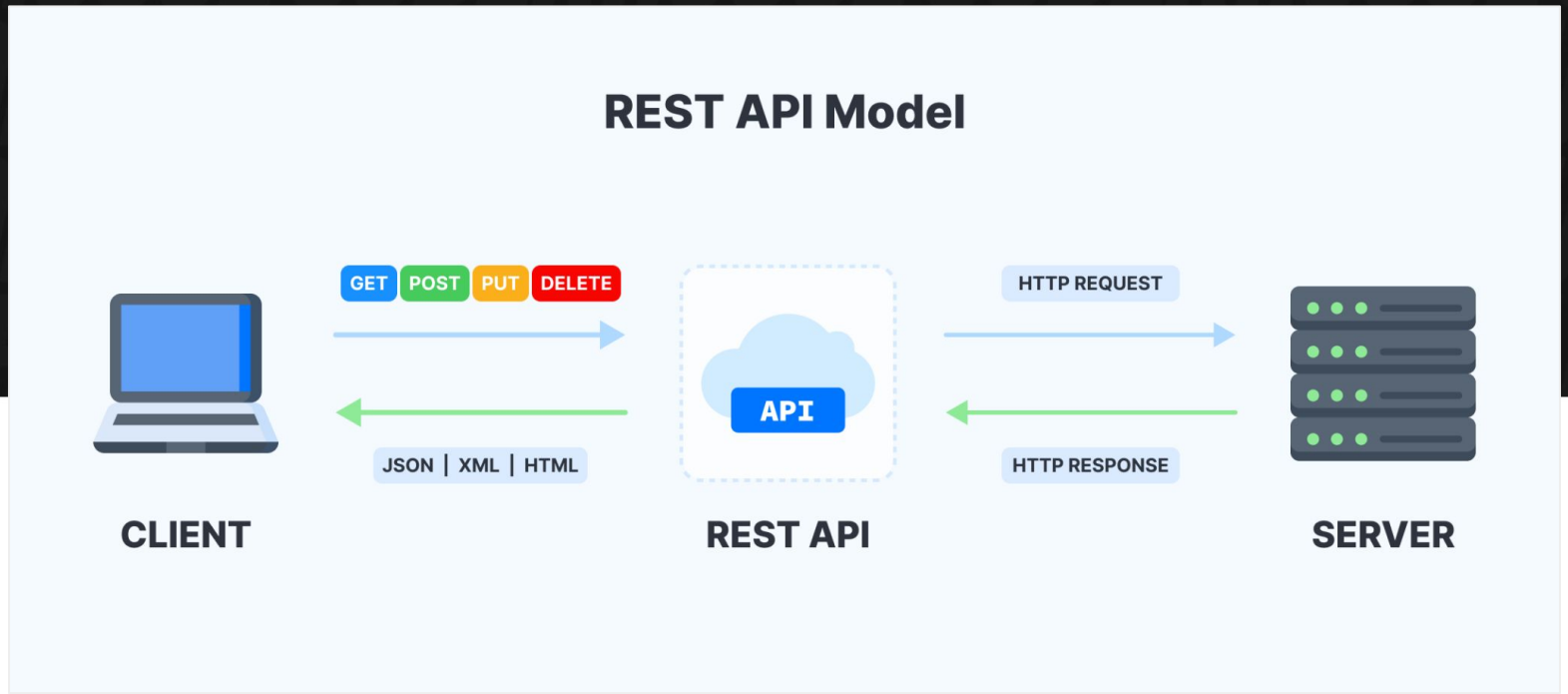
✓ JSON

```json
{
  "cliente":"Gomez",
  "emisor":"Perez S.A.",
  "tipo":"A",
  "items": [
    "Producto 1",
    "Producto 2",
    "Producto 3"
  ]
}
```

Entonces una API REST es...

Un modelo completo para tener perfectamente estipulados los protocolos, las reglas, e incluso la estructura de la información, con el fin de poder hacer un sistema de comunicación completo entre las computadoras.

Modelo de una API REST



Break

¡10 minutos y volvemos!

Métodos de petición

Métodos de petición

Un método es una definición que forma parte del protocolo HTTP, el cual nos sirve para canalizar el tipo de petición que estoy realizando sobre un cierto endpoint. De esta manera, el cliente puede llamar al mismo endpoint, **pero con diferentes métodos**, indicando qué operación quiere realizar con dicho recurso. Los principales métodos son:

- ✓ GET: Obtener un recurso
- ✓ POST: Crear o añadir un recurso
- ✓ PUT: Modificar un recurso
- ✓ DELETE: Eliminar un recurso

CODERHOUSE

Ejemplos sobre cómo declarar correctamente endpoints

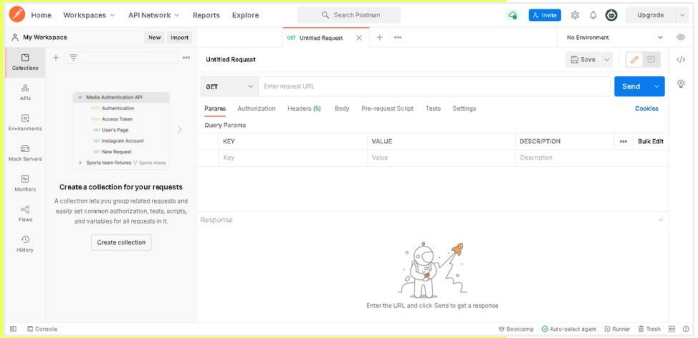
Sí	No	¿Por qué?
GET *api/perritos* GET *api/perritos/:pid*	GET *api/perritos/obtener* GET *api/perritos/obtener/:pid*	GET ya significa “obtener” entonces no hace sentido la redundancia con el método y el endpoint.
POST *api/perritos*	POST *api/perritos/añadir*	POST ya hace referencia a crear un nuevo perrito, entonces es redundante el método y el endpoint

“api/perritos” es perfectamente funcional, sin necesidad de declarar cosas adicionales, entonces podemos reutilizar el endpoint, siempre y cuando sean diferentes sus métodos.

Ejemplos sobre cómo declarar correctamente endpoints

Sí	No	¿Por qué?
PUT *api/perritos/:pid*	PUT *api/perritos/actualizar/:pid*	PUT ya hace referencia a una actualización, hay redundancia entre el método y el endpoint
DELETE *api/perritos*	DELETE *api/perritos/eliminar*	DELETE ya significa eliminar, entonces hay redundancia entre método y endpoint.

"api/perritos" es perfectamente funcional, sin necesidad de declarar cosas adicionales, entonces podemos reutilizar el endpoint, siempre y cuando sean diferentes sus métodos.



POSTMAN

El problema: El navegador sólo puede enviar peticiones con método GET desde la url, (por ello es que podíamos utilizarlo desde el navegador sin problema la clase pasada), sin embargo, para poder utilizar el resto de métodos, no será posible con el navegador

La solución: POSTMAN es un software profesional que nos permitirá gestionar peticiones simulando ser un cliente. De esta manera rompemos la limitante del navegador y podemos probar todos nuestros endpoints.

[Descargar Postman]

CODERHOUSE

Método POST

Sirve para “crear” recursos, POST se utiliza para operaciones donde no necesitamos obtener un recurso, sino añadir uno. Algunos de los casos donde se utilizan son:

- ✓ Registrar un usuario
- ✓ Loguear un usuario
- ✓ Crear un producto
- ✓ Crear una mascota
- ✓ Crear un carrito de compra
- ✓ Enviar información para un correo electrónico.

Se apoya del recurso **req.body**, donde el body representa la información que el cliente envía para crear.

¡Importante!

Para que nuestro servidor express pueda interpretar en forma automática mensajes de tipo JSON en formato urlencoded al recibirlos, debemos indicarlo en forma explícita, agregando las siguiente líneas luego de crearlo.

```
app.use(express.json())
app.use(express.urlencoded({ extended: true }))
```

Probando el endpoint

Método PUT

Ya vimos cómo crear un recurso, ahora ¿Cómo podríamos modificar dicho recurso? El método PUT sirve para ésto.
Para poder trabajar con PUT, no sólo enviamos el body en el request, sino que además mandamos por params el id, nombre, o cualquier identificador para que el servidor sepa qué recurso específicamente debe actualizar.

Hay dos formas de actualizar un recurso: actualizar sólo los campos requeridos, o bien mandar a actualizar el objeto completo, ambas formas son válidas cuando hablamos de actualización, y dependerá del contexto.

Método DELETE

Como bien lo indica el nombre, este método lo utilizamos cuando queremos eliminar algún recurso. Aquí no es necesario enviar nada desde el body, sin embargo, sí es importante indicar en el req.params el identificador para que el servidor reconozca qué recurso debe eliminar.

Ejemplo en vivo: Integrando todos los métodos

- ✓ Se agregará al código de explicación un método GET al mismo endpoint, con el fin de completar los 4 métodos principales.
- ✓ Se realizará un flujo completo con POSTMAN donde podremos ver trabajando a todos los endpoints en conjunto, revisando,

ACTIVIDAD EN CLASE

Servidor con GET, POST, PUT, DELETE

Dada la frase: “Frase inicial”, realizar una aplicación que contenga un servidor en express, el cual cuente con los siguientes métodos:

- ✓ GET '/api/frase': devuelve un objeto que como campo ‘frase’ contenga la frase completa
- ✓ GET '/api/palabras/:pos': devuelve un objeto que como campo ‘buscada’ contenga la palabra hallada en la frase en la posición dada (considerar que la primera palabra es la #1).

ACTIVIDAD EN CLASE

- ✓ POST '/api/palabras': recibe un objeto con una palabra bajo el campo ‘palabra’ y la agrega al final de la frase. Devuelve un objeto que como campo ‘agregada’ contenga la palabra agregada, y en el campo ‘pos’ la posición en que se agregó dicha palabra.
- ✓ PUT '/api/palabras/:pos': recibe un objeto con una palabra bajo el campo ‘palabra’ y reemplaza en la frase aquella hallada en la posición dada. Devuelve un objeto que como campo ‘actualizada’ contenga la nueva palabra, y en el campo ‘anterior’ la anterior.
- ✓ DELETE '/api/palabras/:pos': elimina una palabra en la frase, según la posición dada (considerar que la primera palabra tiene posición #1).
- ✓ Utilizar POSTMAN para probar funcionalidad

¿Preguntas?

Opina y valora esta clase

Muchas gracias.