

# Desarrollo Backend

## Clase 7: Rutas estáticas y dinámicas



**¿Ponemos a grabar el taller?**



# Agenda de hoy

- A. Diferencia entre rutas estáticas y dinámicas
- B. Manejo de parámetros en el servidor
  - a. Query String
  - b. URL String
- C. Parámetros dinámicos en Node.js
  - a. simples
  - b. múltiples





**Seguimos trabajando con  
rutas**



## Seguimos trabajando con rutas

Como podemos apreciar, a través de todas las unidades que llevamos trabajando con servidores web, el concepto de **rutas** tiene una importancia notable, dado que **es la forma más asertiva de poder llegar al contenido que deseamos visualizar**.

Pero aún nos queda un camino por recorrer dado que, para poder **visualizar datos dinámicos**, el concepto simple de rutas que venimos trabajando no es suficiente.



# Rutas Dinámicas

Para sortear la dificultad de este último punto, las **rutas dinámicas** son la herramienta más efectiva que nos permiten alcanzar a *visualizar información*, de acuerdo a diferentes parámetros o condiciones que debemos tener en cuenta.



# Rutas Dinámicas

Vamos entonces a entender mejor este concepto, junto a los fundamentos de las rutas dinámicas, y a las prácticas utilizando diferentes servicios web que permitirán que afiancemos mejor este concepto, previo a comenzar a desarrollar nuestras soluciones a base de código.





# Diferencia entre rutas estáticas y dinámicas



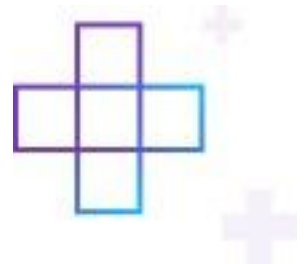


# Diferencia entre rutas estáticas y dinámicas

Hasta ahora, venimos trabajando con rutas para poder servir tanto archivos de proyectos correspondientes a una aplicación frontend como también sirviendo set de datos, algo más cercano a una aplicación backend.

```
Rutas

'http://localhost:5500/'      //ruta raíz
'http://localhost:5500/cursos' //ruta cursos
'http://localhost:5500/contacto' //ruta contacto
```



# Diferencia entre rutas estáticas y dinámicas

Todas estas rutas que hemos venido trabajando hasta el momento, se denominan **rutas estáticas**. Cuando petitionamos a las mismas, éstas nos entregan la información predeterminada que tienen definida.

Pero, en el mundo del backend, tenemos que tener presente que también existen las **rutas dinámicas**; estas usualmente reciben un parámetro por URL, y utilizan el mismo para mostrar información relacionada a este.

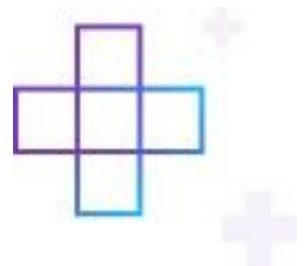


# Diferencia entre rutas estáticas y dinámicas

Imaginemos que disponemos de un array similar al que aquí vemos, y necesitamos crear una ruta que reciba parámetros dinámicos para mostrar información parcial de este Set de datos.

```
Rutas

const cursos = [{id: 1, nombre: 'JavaScript', categoria: 'Programación'},
                 {id: 2, nombre: 'React JS', categoria: 'Programación'},
                 {id: 3, nombre: 'Vue JS', categoria: 'Programación'},
                 {id: 4, nombre: 'SQL', categoria: 'Datos'},
                 {id: 5, nombre: 'MongoDB', categoria: 'Datos'},
                 {id: 6, nombre: 'Ecommerce', categoria: 'Producto'},
                 {id: 7, nombre: 'Customer Experience', categoria: 'Producto'}],
```

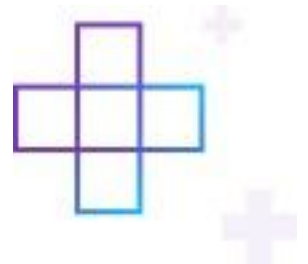


# Diferencia entre rutas estáticas y dinámicas

Nuestra ruta `/cursos` entrega información de todos los cursos almacenados aquí. Podríamos aprovechar la misma ruta, para enviar a esta un parámetro adicional, como ser la **categoría** del curso, y así obtener como resultado los cursos que aplican a dicha categoría.

```
Rutas

const cursos = [{id: 1, nombre: 'JavaScript', categoria: 'Programación'},
  {id: 2, nombre: 'React JS', categoria: 'Programación'},
  {id: 3, nombre: 'Vue JS', categoria: 'Programación'},
  {id: 4, nombre: 'SQL', categoria: 'Datos'},
  {id: 5, nombre: 'MongoDB', categoria: 'Datos'},
  {id: 6, nombre: 'Ecommerce', categoria: 'Producto'},
  {id: 7, nombre: 'Customer Experience', categoria: 'Producto'},]
```



# Diferencia entre rutas estáticas y dinámicas

La URL final quedará conformada por una estructura similar al ejemplo que vemos arriba. '**Datos**' será la descripción de la categoría que deseamos ver.

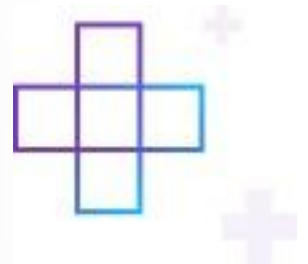
Entonces, la estructura de la ruta a crear en Node.js, debe ser similar a **/cursos/:categoria**.



Rutas

```
'http://localhost:3050/cursos/Datos' //el filtro a aplicar es 'Datos'
```

```
'/cursos/:categoria' //será el nombre de la ruta dinámica en Node.js
```





# Diferencia entre rutas estáticas y dinámicas

En nuestro código de ejemplo: **categoría** es la referencia al parámetro dinámico que utilizaremos para indicarle a la URL del servidor qué categoría de cursos deseamos ver.

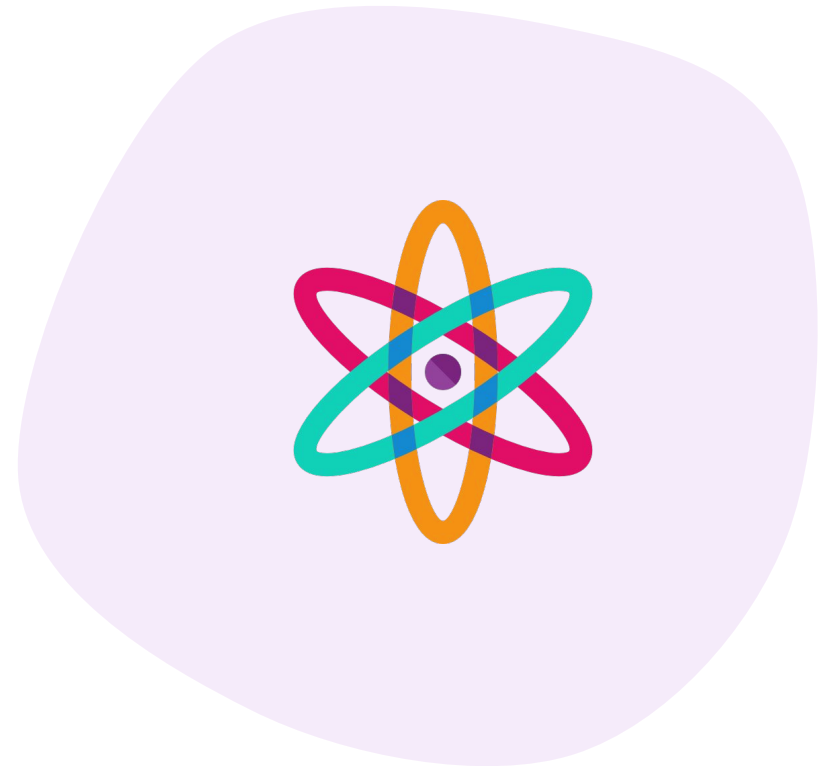
Del lado del servidor, debemos tener la lógica presente para poder leer ese parámetro dinámico y aplicar un filtro sobre los cursos para así obtener y enviar el set de datos correspondiente a aquellos solicitados.



# Diferencia entre rutas estáticas y dinámicas

Pero antes de ver la lógica de programación, para entender mejor el uso de rutas dinámicas y de los diferentes tipos que existen, realizaremos algunas pruebas sobre aplicaciones de backend de ejemplo y de uso público, que nos ayudarán a representar los diferentes escenarios posibles.

Vayamos entonces a los ejemplos.



# Manejo de parámetros en el servidor



# Manejo de parámetros en el servidor

Trabajaremos a continuación con las herramientas listadas a continuación. La idea es que primero veamos los diferentes ejemplos posibles en cuanto a pedir datos específicos, y luego repasemos las bases técnicas para lograr aplicar el concepto de rutas dinámicas en Node.js.

## Hoppscotch

Es una aplicación online conocida como PWA, que nos permite realizar peticiones a un servidor web específico, y visualizar los resultados del servidor en un panel dedicado.

## JSONPlaceholder

JSONPlaceholder es una herramienta online que nos brinda una API de simulación. Ofrece una variedad de recursos, como usuarios, publicaciones y comentarios, a la cual podemos acceder realizando peticiones HTTP.

## RandomUser

RandomUser es otra herramienta online, similar a JSONPlaceholder, pero limitada a la creación de set de datos con información de usuarios. Este set de datos es muy amplio y dispone de diversas combinaciones para filtrar información de acuerdo a nuestra necesidad.

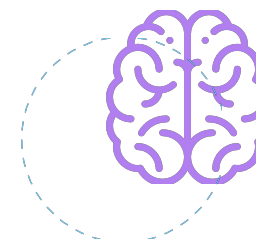
# Sección práctica

Trabajaremos con la herramienta Hoppscotch  
peticionando datos al endpoint RandomUser,  
aplicando diferentes combinaciones de  
acuerdo a las consignas específicas.





# Sección Práctica



Deberás leer la documentación de RandomUser, y realizar las siguientes peticiones a dicho endpoint:

1. obtener un listado de todos los usuarios
2. obtener un listado limitado a 4500 usuarios (*valida tiempo y tamaño de la respuesta*)
3. obtener un listado de 50 usuarios donde:
  - sean mujeres
  - residan en España
4. obtener un listado limitado a 10 usuarios de nacionalidad *australiana*
5. obtener un listado limitado a 40 usuarios de nacionalidad *australiana, brasileña, canadiense y española*
6. obtener un listado limitado a 60 usuarios, donde se limiten los datos a visualizar a (*nombre, nacionalidad, email, teléfono móvil, imagen*)
7. Repetir el punto anterior, pero visualizando el resultado en *formato XML*

Valida en cada caso el tamaño de la respuesta devuelta y el tiempo que demora cada una de estas peticiones. Si no obtienes alguna de las respuestas correctamente, valida con la documentación.



# Puesta a punto del ejercicio práctico

Veamos junto a la profe la resolución de la práctica anterior.



# Manejo de parámetros en el servidor

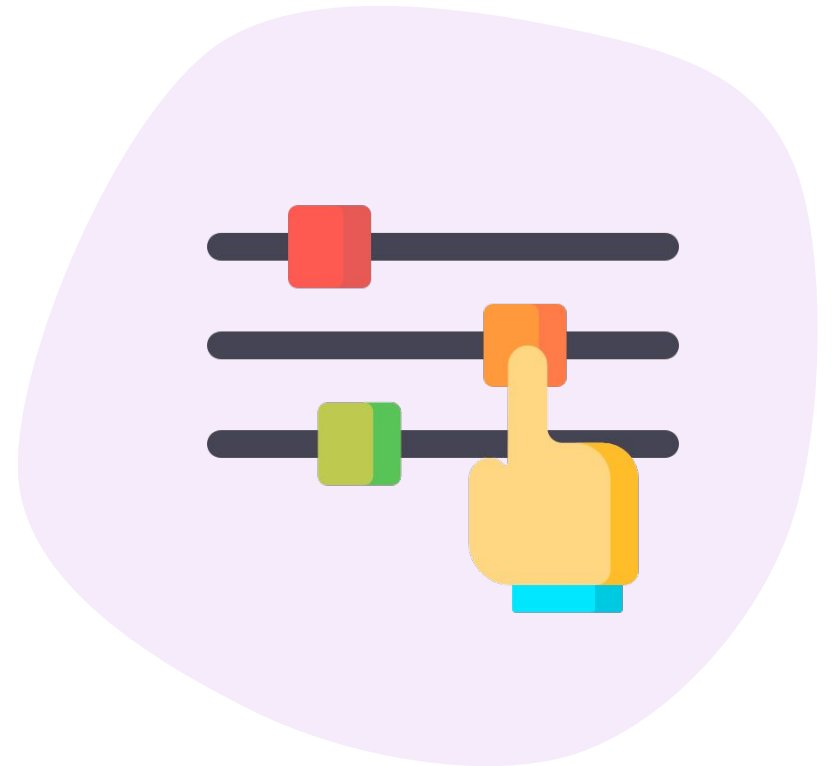
Como pudimos apreciar en este último ejercicio, el servidor web es quien se ocupa de manejar los diferentes parámetros, o rutas dinámicas, que recibe, y responder acorde a la necesidad de la petición con los datos apropiados.

Veamos a continuación un poco de la teoría que acompaña a las diferentes opciones de **parámetros/rutas dinámicas en Node.js**.



# Manejo de parámetros en el servidor

Cada servidor web debe contar con la posibilidad de responder a diferentes peticiones, manejando a través de las URL que ofician como endpoints, diferentes parámetros que pueden llegar en éstas para luego responder de manera acorde con los datos solicitados.

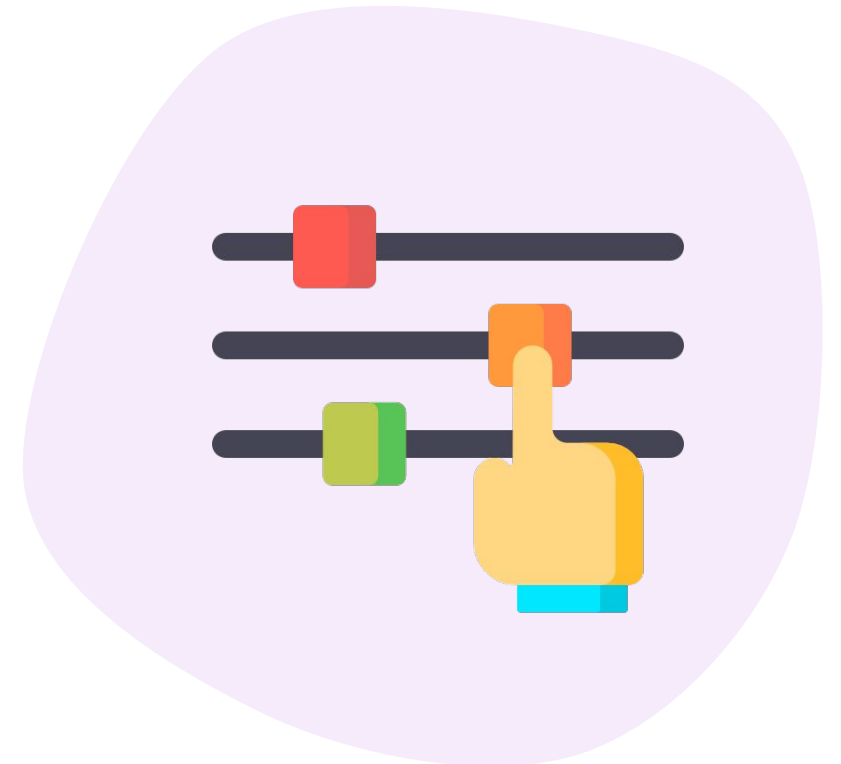


# Manejo de parámetros en el servidor

Y, entre las posibilidades de recibir peticiones dinámicas, tenemos la opción de definir las mismas utilizando los mecanismos:

- **query params**
- **url params**

Estos son dos tipos de parámetros que pueden utilizarse en una aplicación web para transmitir información entre el cliente y el servidor.



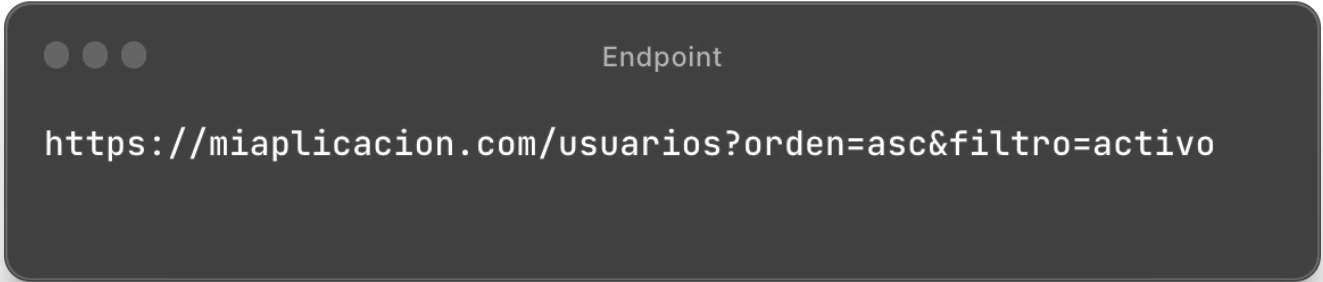


## Query params

# Manejo de parámetros en el servidor

**query params** corresponde a valores que se agregan a la URL de una solicitud HTTP después del signo de interrogación (?).

Por ejemplo, en la siguiente URL:




```
Endpoint  
https://miaplicacion.com/usuarios?orden=asc&filtro=activo
```

Los valores **asc** y **activo** son query params que indican el orden y filtro a aplicar en una lista de usuarios.



# Manejo de parámetros en el servidor

Podemos recurrir a enviar solo un parámetro, definiendo el mismo como **parametro=valor**, o definir más de un parámetro tal como muestra la URL de ejemplo.



Endpoint

```
https://miaplicacion.com/usuarios?orden=asc&filtro=activo
```



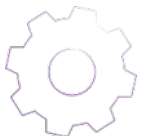
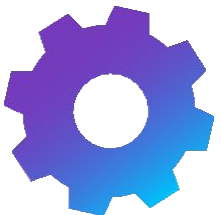
# Manejo de parámetros en el servidor

En el caso de tener que definir más de un parámetro, separamos cada uno de ellos utilizando el caracter ampersand (&): **parametro1=valor1&parametro2=valor2**



Endpoint

```
https://miaplicacion.com/usuarios?orden=asc&filtro=activo
```

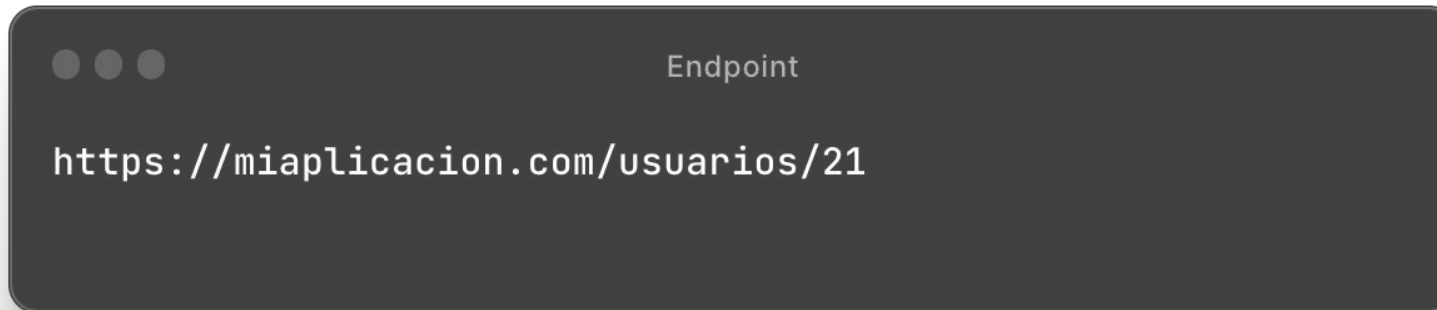


## URL params



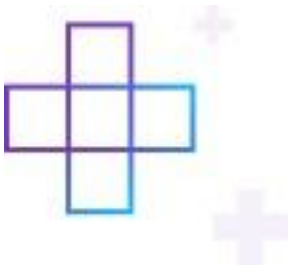
# Manejo de parámetros en el servidor

**URL params** (también conocidos como "*parámetros de ruta*" o "*path params*") son valores que se incluyen en la ruta o URL de una solicitud HTTP. Por ejemplo, en la URL:

A dark-themed terminal window with three gray window control buttons in the top-left corner. The title bar on the right says "Endpoint". The text inside the terminal is a URL: `https://miaplicacion.com/usuarios/21`.

```
Endpoint
https://miaplicacion.com/usuarios/21
```

El valor **21** es una URL param que transmite el ID del usuario solicitado.



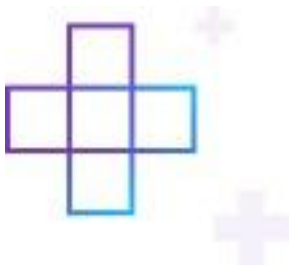
# Manejo de parámetros en el servidor

Estos parámetros se definen en la ruta del endpoint en cuestión, y se pueden capturar en la función de manejo de la ruta en el servidor a través de una sintaxis específica.

● ● ●

Endpoint

```
https://miaplicacion.com/usuarios/21
```





# Descanso

Nos vemos en 5 minutos



# Parámetros dinámicos en Node.js



# Parámetros dinámicos en Node.js

Trabajaremos un ejemplo de servidor web utilizando este modelo de datos. Realizaremos una aplicación web de servidor con **Express**, la cual sirva estos datos a través de la ruta **/cursos**, y que luego se pueda especificar un filtro de datos, tomando como parámetro a la propiedad **categoría**.

```
Rutas

const cursos = [{id: 1, nombre: 'JavaScript', categoria: 'Programación'},
                 {id: 2, nombre: 'React JS', categoria: 'Programación'},
                 {id: 3, nombre: 'Vue JS', categoria: 'Programación'},
                 {id: 4, nombre: 'SQL', categoria: 'Datos'},
                 {id: 5, nombre: 'MongoDB', categoria: 'Datos'},
                 {id: 6, nombre: 'Ecommerce', categoria: 'Producto'},
                 {id: 7, nombre: 'Customer Experience', categoria: 'Producto'},]
```

# Parámetros dinámicos en Node.js

```
Rutas dinámicas

const express = require('express');
const cursos = require('./cursos');
const app = express();
const dotenv = require('dotenv');
  dotenv.config();
const PORT = process.env.PORT || 3000;

app.get('/', (req, res) => {
  res.send('Bienvenid@s al servidor web con rutas dinámicas!');
});

app.get('/cursos', (req, res) => {
  res.json(cursos);
});

app.get('/cursos/:categoria', (req, res) => {

});

app.get('*', (req, res) => {
  res.status(404).send('Lo siento, la página que buscas no existe.');
```

Nuestra estructura de servidor web básica, es la de siempre. Creamos una ruta raíz “/”, que responde a la URL principal del servidor web.

Una segunda ruta que responde a **/cursos**, donde se enviará todo el set de datos de los cursos en cuestión.

Una nueva ruta **/cursos/**, con el agregado de que recibirá un parámetro adicional; en este caso **:categoria**.

Y, por supuesto, el control de errores sobre rutas inexistentes.

# Parámetros dinámicos en Node.js

El objeto **request** cuenta con un objeto simple denominado **params**. Este es el encargado de recibir todo parámetro que llegue en una petición mediante la URL que oficia de endpoint.

```
Rutas dinámicas

//definición del parámetro esperado
"/cursos/:categoria"

//dónde encontrar el parámetro definido en la URL que peticiona datos.
req.params.categoria;
```

En el momento en el cual definimos el parámetro de la ruta, éste se agrega dentro del objeto simple **params** para que, luego, podamos consultar el valor que recibe.



# Parámetros dinámicos en Node.js

Si queremos probar su funcionamiento, simplemente agregamos un `console.log()` dentro de la ruta de parámetro, y enviamos cualquier dato desde una pestaña del navegador web. Veremos el parámetro reflejado en la **Terminal**.

```
Rutas dinámicas

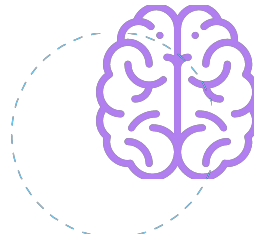
app.get('/cursos/:categoria', (req, res) => {
  console.log(req.params.categoria);
});
```



## Algunas condiciones a tener en cuenta



# Algunas condiciones a tener en cuenta



Cuando creamos rutas dinámicas, debemos tener presente que los parámetros que recibiremos pueden llegar de diferentes formas, por lo tanto, debemos contemplar los posibles diferentes escenarios cuando tomamos los mismos para procesarlos.

Aquí un ejemplo de posibles diferentes parámetros que llegan a través del endpoint:



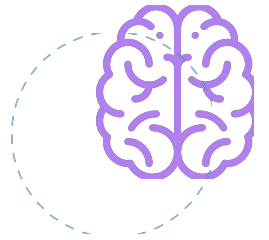
Rutas dinámicas

```
http://localhost:3008/cursos/datos
```

```
http://localhost:3008/cursos/diseño gráfico
```

```
http://localhost:3008/cursos/PROGRAMACION
```

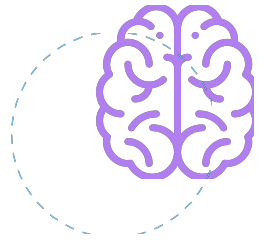
# Algunas condiciones a tener en cuenta



El parámetro que recibimos en esta primera opción, llega escrito todo en minúsculas. El mismo será leído por **request.params.categoria**, tal cual lo recibimos por URL.

```
Rutas dinámicas
http://localhost:3008/cursos/datos
http://localhost:3008/cursos/diseño gráfico
http://localhost:3008/cursos/PROGRAMACION
```

# Algunas condiciones a tener en cuenta



El parámetro que recibimos en esta segunda opción, es un parámetro que combina dos palabras. Ante este caso, los motores de navegadores web reemplazan los caracteres de espacio ' ', por el carácter %20.



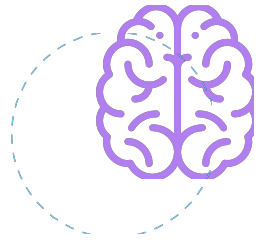
Rutas dinámicas

```
http://localhost:3008/cursos/datos
```

```
http://localhost:3008/cursos/diseño gráfico
```

```
http://localhost:3008/cursos/PROGRAMACION
```

# Algunas condiciones a tener en cuenta



El parámetro que recibimos en este tercer ejemplo, es un **parámetro simple**, de una sola palabra, pero escrita toda en mayúsculas. Se deberá contemplar estas posibles situaciones también del lado del servidor.



Rutas dinámicas

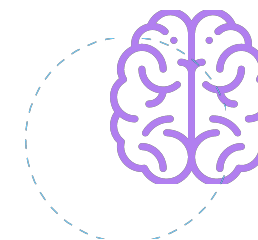
```
http://localhost:3008/cursos/datos
```

```
http://localhost:3008/cursos/diseño gráfico
```

```
http://localhost:3008/cursos/PROGRAMACION
```

## Parámetro simple (URL params)

# Parámetro simple (URL params)



Volvemos a la construcción del endpoint: el parámetro recibido lo almacenaremos en una variable interna, previo a filtrar la información con la que debemos responder. Aprovechando este paso, aplicaremos para mayor seguridad, el uso del método `.trim()` seguido del método `.toLowerCase()`.

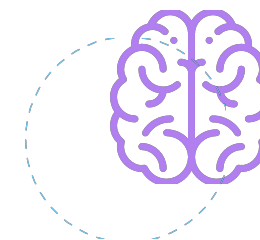
De esta forma minimizamos cualquier posible error, y nos ocupamos de evitar el uso combinado de mayúsculas/minúsculas en los valores, para así lograr una comparación de datos más asertiva.

Rutas dinámicas

```
let parametro = req.params.categoria.trim().toLowerCase();
```



# Parámetro simple (URL params)



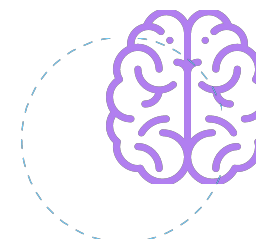
Recorremos el array **cursos** utilizando la cláusula **for...of**. Luego comparamos la propiedad **categoria** de cada uno de los objetos, aplicando también el método **toLowerCase()**, con el parámetro capturado mediante la URL del endpoint en cuestión.

Si coincide el objeto iterado, agregamos el mismo a un array **resultado**, declarado e inicializado vacío **[]**.

```
Rutas dinámicas

if (parametro !== "") {
  let resultado = [];
  for (let curso of cursos) {
    if (curso.categoria.toLowerCase() == parametro) {
      resultado.push(curso)
    }
  }
}
```

## Parámetro simple (URL params)



Finalizada la iteración, nos queda evaluar si el array **resultado** tiene o no elementos, mediante su propiedad **.length**. En caso afirmativo, retornamos el array en cuestión utilizando el método **.json()** del objeto **response**, sino, retornamos un **mensaje de error** como respuesta para que el cliente evalúe cómo manejar esto.



Rutas dinámicas

```
resultado.length > 0 ?  
  res.json(resultado) :  
  res.json([{id: 'Error', descripcion: 'No se encontraron coincidencias.'}]);
```

## Parámetro simple (URL params)

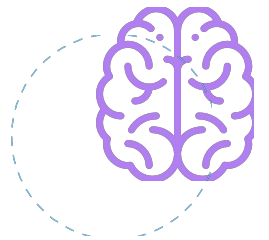
Si bien estructurar la respuesta correspondiente nos lleva más de 10 líneas de código, en nuestro próximo encuentro podremos aprovechar las diferentes funciones de orden superior, para simplificar al máximo esta tarea de filtrar datos.

El beneficio que obtenemos con este ejercicio de iteración, es ir conociendo cómo se comportan internamente las funciones de orden superior.



## Parámetros múltiples (query params)

# Parámetro múltiples (query params)



A diferencia de **URL params**, **Query params** no requiere que definamos **:parametros** en la URL del endpoint.

Si llegan los parámetros en cuestión, estos se alojan dentro del objeto simple **query**, contenido en el objeto **request**, respondiendo de igual manera que URL (*con el nombre del parámetro*).

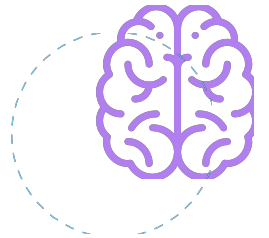
Entonces, la lógica de respuesta nos lleva a modificar el endpoint **/cursos**, donde debemos validar si llegan parámetros o no, para responder de la forma correcta ante cada escenario posible.



Rutas dinámicas

```
req.query.nombre //devuelve el valor del parámetro nombre  
req.query.categoria //devuelve el valor del parámetro categoría
```

# Parámetro múltiples (query params)



Recurrimos a **Object.keys** para validar si el objeto **request.query**, posee claves (*propiedades*) internas.

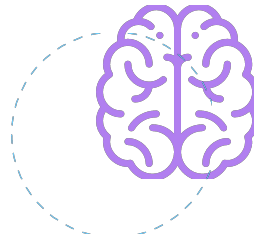
Si el resultado de la constante **queryParams** nos da 0, entonces enviamos el set de datos completo.

Caso contrario es porque llegan parámetros. Allí nos ocupamos de iterar el array **cursos**, buscando que coincidan las propiedades **nombre** y **categoría** con los datos recibidos en **request.query**.

```
Rutas dinámicas

const queryParams = Object.keys(req.query);
if (queryParams.length === 0) {
  console.log('No llegan parámetros. Envío el set de datos');
  res.send('Llegan parámetros');
} else {
  console.log('Llegan los parámetros nombre y categoría');
}
```

# Parámetro múltiples (query params)



En la iteración del array **cursos**, podemos validar mediante la cláusula **if()** aplicando un condicional combinado con el operador lógico AND (**&&**). Sólo si se cumplen ambas coincidencias, entonces agregamos el curso iterado a nuestro array **resultado**.

El resto de la estructura queda igual a la que utilizamos en el ejemplo de URL params.

```
Rutas dinámicas

for (let curso of cursos) {
  if (curso.nombre.toLowerCase().includes(req.query.nombre.toLowerCase())
      && curso.categoria.toLowerCase().includes(req.query.categoria.toLowerCase())) {
    resultado.push(curso)
  }
}
```

## Parámetro múltiples (query params)

Si bien podemos establecer múltiples combinaciones y apoyarnos en **Object.keys** para evaluar qué parámetros elige el usuario al peticionar el endpoint utilizando múltiples parámetros, lo ideal es elegir correctamente qué propiedades (*campos*) de un set de datos serán los apropiados para filtrar información, y acotar los posibles escenarios de búsquedas.

Esto nos ayudará a minimizar errores y que nuestro código no termine siendo demasiado extenso dentro de un endpoint.





# Desafío

Para la próxima clase:

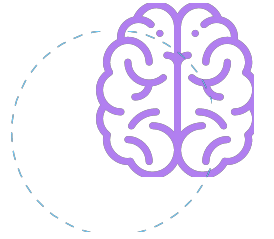
Al servidor de ejemplo que utilizamos en esta unidad, le agregaremos algunos endpoints adicionales, que permitan buscar tanto por código de curso como también por nombre.

Veamos a continuación el detalle de la consigna completa.

La profe te compartirá el código base del servidor web express en donde debes trabajar este desafío.



# Prácticas



En el servidor web debemos definir las siguientes rutas:

1. `"/curso/codigo/:id"`

2. `"/curso/nombre/:nombre"`

1. El código de curso debe recibir un parámetro numérico. Convertimos al mismo en un valor numérico utilizando **parseInt()**, verificamos que así sea utilizando **typeof** y finalmente buscamos el curso en cuestión iterando el array. Cuando encontramos el mismo, lo agregamos al array **resultado** y lo retornamos como respuesta interrumpiendo la posible continuidad de una iteración innecesaria.
2. Cuando buscamos por nombre, la búsqueda deberá ser parcial, o sea, que se pueda enviar parte del nombre del curso. El endpoint podrá devolver uno o más cursos resultantes.



¿Alguna consulta?







¡Muchas gracias!

FUNDACIÓN  
**YPF**