

A graphic on the left side of the slide features four horizontal bars of increasing height, colored purple, orange, yellow, and blue from top to bottom. The text 'Agencia de Aprendizaje a lo largo de la vida' is written across these bars in white. The orange bar is shaped like an arrow pointing to the right.

Agencia de  
Aprendizaje  
a lo largo  
de la vida

# **FULL STACK FRONTEND**

## **Clase 28**

Node 4

# Les damos la bienvenida

Vamos a comenzar a grabar la clase

## Clase 27

### Node

- Express
- Servidor estático
- Nodemon
- Rutas

## Clase 28

### Node

- Express Router
- Postman  
GET  
POST  
PUT  
DELETE

## Clase 29

### Node



# NodeJS

## Rutas - Parte 2

# Rutas

Como vimos hasta el momento, la comunicación entre **clientes y servidores** o entre programas que interactúan a través de la web, se hace mediante rutas.

A estas **rutas** se las conoce comúnmente como **ENDPOINTS** y necesitaremos uno por cada flujo que posea nuestro servidor.

**Ahora sí, nuestras rutas se  
van complejizando.**

# Creación del proyecto

Vamos crear un proyecto para simular el consumo de datos desde un frontend.

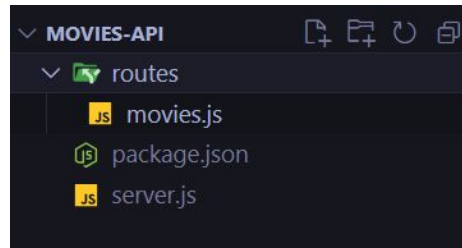
Para ello vamos a el proyecto:

```
npm init -y
```

Instalamos express:

```
npm install express
```

Y luego creamos la estructura:



# Express Router

Es una característica de Express.js que permite crear manejadores de rutas modulares y montables. Un enrutador de Express actúa como una instancia mini de una aplicación completa, lo que permite definir rutas y middleware de manera modular y reutilizable. Esto es especialmente útil para organizar tu aplicación en varios archivos y mantener el código limpio y manejable.



# Express Router

Vamos a crear el archivo server.js con el siguiente código

```
.js server.js > ...  
1  const express = require('express');  
2  const app = express();  
3  const moviesRouter = require('./routes/movies');  
4  
5  app.use(express.json());  
6  
7  app.use('/movies', moviesRouter);  
8  
9  const PORT = 3000;  
10 app.listen(PORT, () => {  
11   console.log(`Server is running on port ${PORT}`);  
12 });  
13
```

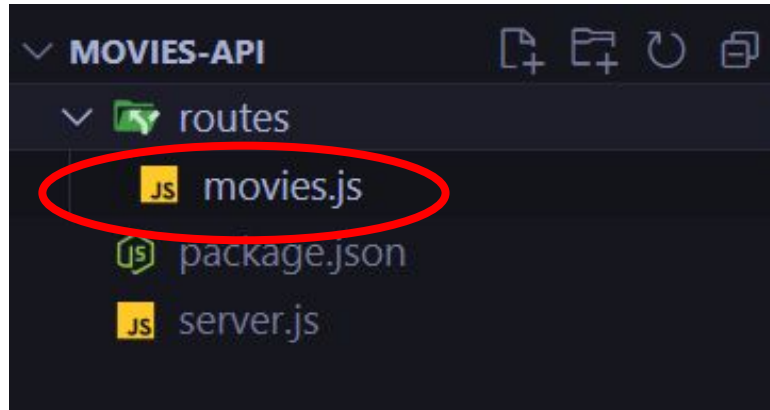
# Express Router

Aquí el mismo código con la explicación:

```
server.js > ...
1 // Importamos el módulo express
2 const express = require('express');
3 // Creamos una instancia de una aplicación de express
4 const app = express();
5 // Importamos el enrutador de películas desde el archivo routes/movies
6 const moviesRouter = require('./routes/movies');
7
8 // Middleware para analizar cuerpos JSON en las solicitudes entrantes
9 app.use(express.json());
10
11 // Definimos la ruta para películas y llamamos al moviesRouter
12 // para manejar las rutas que comiencen con /movies
13 app.use('/movies', moviesRouter);
14
15 // Definimos el puerto en el que nuestro servidor escuchará las solicitudes
16 const PORT = 3000;
17
18 // Iniciamos el servidor y lo configuramos para que escuche en el puerto definido
19 app.listen(PORT, () => {
20   console.log(`Server is running on port ${PORT}`);
21 });
22
```

# El primer archivo de rutas

Para mantener nuestro Proyecto organizado y que pueda crecer de manera prolija, vamos a generar un archivo que maneje todas las rutas de una película:



# Creando nuestro primer endpoint

```
routes > js movies.js > ...  
4 // Creamos una instancia del router de Express  
5 const router = express.Router();  
6  
7 // Definimos una ruta para obtener todas las películas  
8 // Cuando se haga una solicitud GET a la ruta ('/movies/'), se ejecutará esta función  
9 router.get('/', (req, res) => {  
10   // Envía una respuesta en formato JSON con un mensaje  
11   res.json({ mensaje: 'Hola desde la ruta de películas!'});  
12 });  
13  
14 // Exporta el router para que pueda ser utilizado en otros archivos  
15 module.exports = router;
```

# Probemos nuestros endpoint.

# POSTMAN

Product Pricing Enterprise Resources and Support Public API Network Search Postman Contact Sales Sign In Sign Up for Free

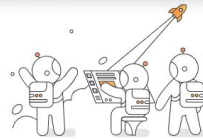
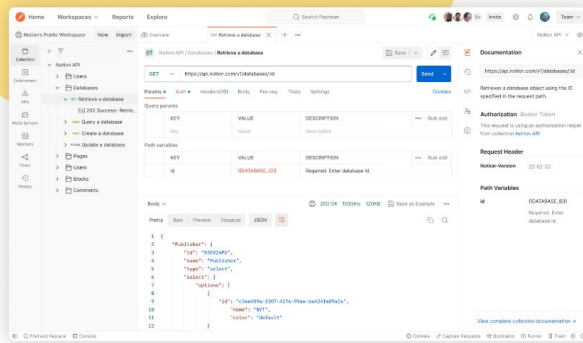
## Build APIs together

Over 30 million developers use Postman. Get started by signing up or downloading the desktop app.

name@company.com

Sign Up for Free

Download the desktop app for



## What is Postman?

Postman is an API platform for building and using APIs. Postman simplifies each step of the API lifecycle and streamlines collaboration so you can create better APIs—faster.

**API Tools**  
A comprehensive set of tools that help

**API Repository**  
Easily store, iterate and collaborate around

**Workspaces**  
Organize your API work and collaborate

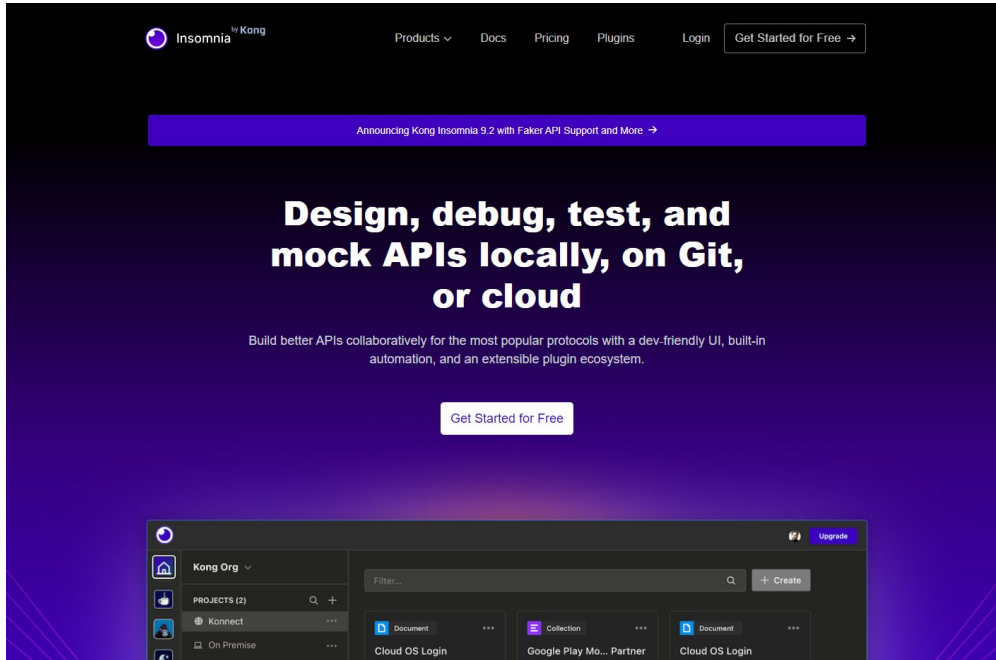
**Governance**  
Improve the quality of APIs with governance

Postman es una popular herramienta utilizada para probar APIs, permitiendo a los desarrolladores enviar peticiones a servicios web y ver respuestas

<https://www.postman.com/>



# INSOMNIA



Es otra herramienta que se ha vuelto popular con el paso del tiempo. Es una alternativa a Postman

<https://insomnia.rest/>

# Postman / Insomnia

Lo importante es que ambas herramientas nos permiten “simular” estas peticiones a través de los **distintos métodos HTTP**.

Estas herramientas facilitan enviar consultas a nuestro servidor sin **depende de un Frontend armado**.

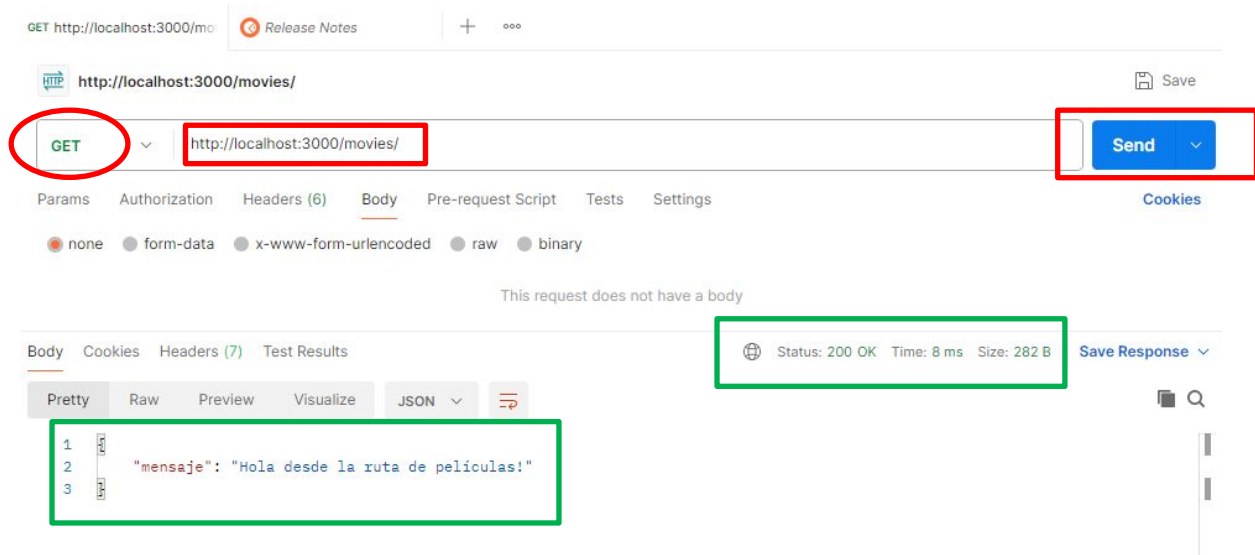
En su versión gratuita nos permite realizar todo el trabajo necesario para validar el funcionamiento de nuestros endpoints, mientras que con la versión paga tenemos funciones adicionales que nos permiten trabajar en equipo con mayor facilidad.



# Cómo usar POSTMAN

## Método GET

Dentro de **POSTMAN** podemos crear una nueva petición con el botón de **+**, luego seleccionamos el método (en este caso **GET**) y la **url** a consultar.

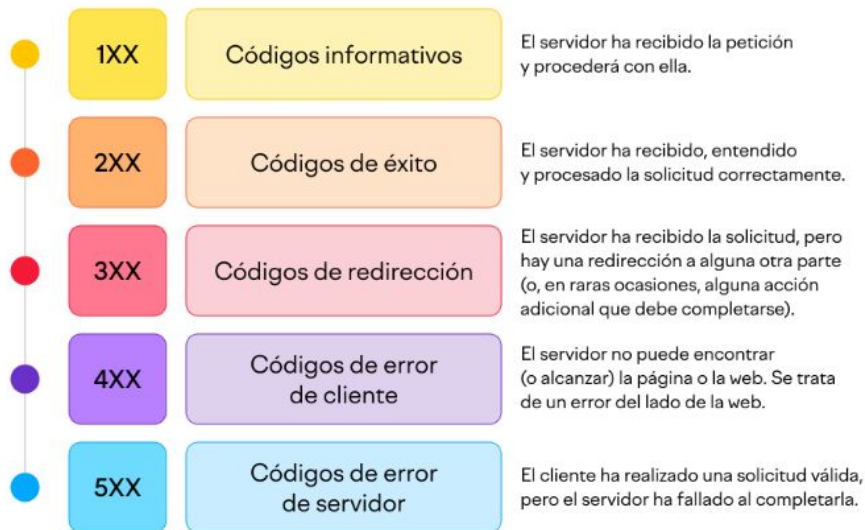


Finalmente presionamos **SEND** y esperamos la **respuesta** del servidor

De esta manera probamos  
nuestro **backend** sin un  
**frontend** y comunicamos  
nuestra aplicación con el  
mundo exterior.

# Status de de una respuesta HTTP

En el contexto de las respuestas HTTP, el status (estado) es un código numérico que el servidor devuelve para indicar el resultado de la solicitud realizada por el cliente. Los códigos de estado HTTP están divididos en cinco categorías, cada una representando un tipo diferente de respuesta



# Status de de una respuesta HTTP

Dentro de las respuestas más comunes, encontramos:

**200 OK:** La solicitud ha tenido éxito. La información solicitada está en el cuerpo de la respuesta.

**404 Not Found:** El recurso solicitado no se pudo encontrar en el servidor.

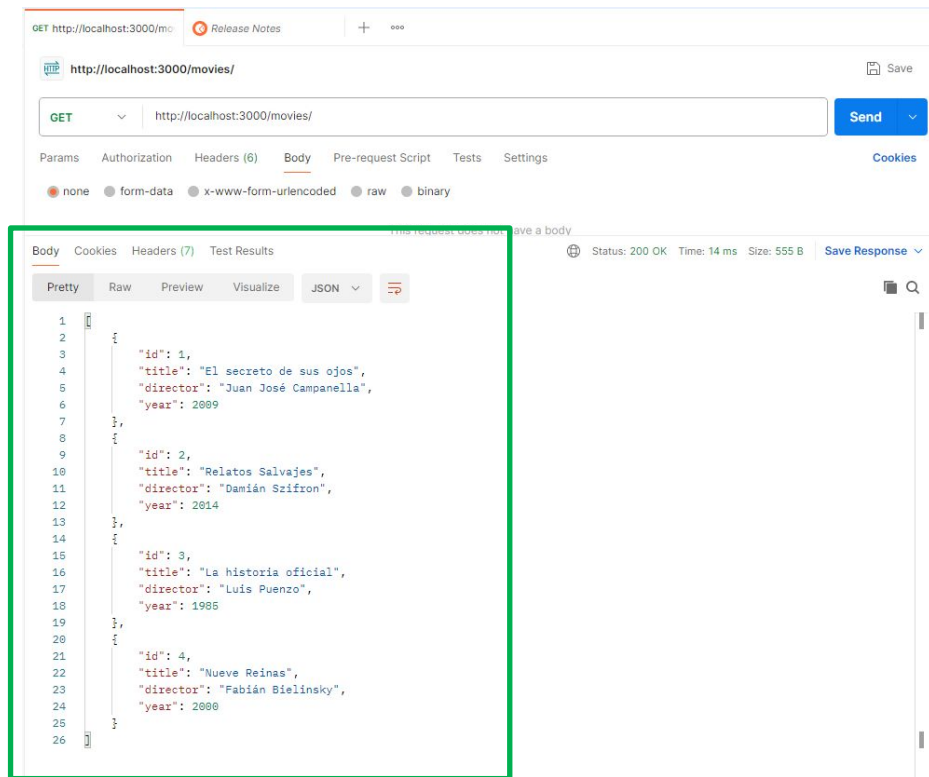
**500 Internal Server Error:** El servidor encontró una condición inesperada que le impidió completar la solicitud.

Estas respuestas en ocasiones, las programaremos nosotros desde el backend para informar determinados resultados al frontend.

routes &gt; JS movies.js &gt; [🔗] movies

```
1  const express = require('express');
2  const router = express.Router();
3
4  // Datos ficticios de películas
5  let movies = [
6    { id: 1, title: "El secreto de sus ojos", director: "Juan José Campanella", year: 2009 },
7    { id: 2, title: "Relatos Salvajes", director: "Damián Szifron", year: 2014 },
8    { id: 3, title: "La historia oficial", director: "Luis Puenzo", year: 1985 },
9    { id: 4, title: "Nueve Reinas", director: "Fabián Bielinsky", year: 2000 }
10 ];
11
12 // Obtener todas las películas
13 router.get('/', (req, res) => {
14   res.json(movies);
15 });
16
17 module.exports = router;
18
```

# Desde Postman...



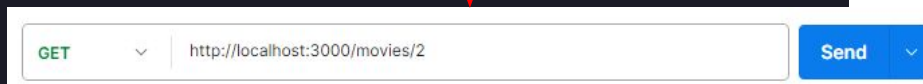
Si esta informaci\u00f3n la recibimos en un frontend, como resultado a una petici\u00f3n fetch, podremos manipular la informaci\u00f3n para mostrarla en el documento HTML de acuerdo a las necesidades de la aplicaci\u00f3n.

```

4 // Datos ficticios de películas
5 let movies = [
6   { id: 1, title: "El secreto de sus ojos", director: "Juan José Campanella", year: 2009 },
7   { id: 2, title: "Relatos Salvajes", director: "Damián Szifron", year: 2014 },
8   { id: 3, title: "La historia oficial", director: "Luis Puenzo", year: 1985 },
9   { id: 4, title: "Nueve Reinas", director: "Fabián Bielinsky", year: 2000 }
10 ];
11
12 // Obtener todas las películas
13 router.get('/', (req, res) => {
14   res.json(movies);
15 });
16
17 // Obtener una película por ID
18 router.get('/:id', (req, res) => {
19
20 });
21
22
23 module.exports = router;
24
25
26

```

Lo primero que hacemos es definir una nueva ruta, donde a través de los ":" estamos indicando el argumento que vamos a recibir.



Si desde postman a la ruta **movies** agregamos un **/2** ese valor se recibirá en el backend en el id.

# Utilizando parámetros en la ruta

Ahora, solo debemos agregar un poco de Javascript para obtener del array de películas, la película que estamos buscando, teniendo en cuenta que en **req.params.id** está el valor que necesitamos:

```
17 // Obtener una película por ID
18 router.get('/:id', (req, res) => {
19     const movie = movies.find(m => m.id === parseInt(req.params.id));
20 });
```

Tenemos que contemplar el caso en que el id recibido no exista dentro del arreglo de películas. En ese caso, utilizando la convención de status, devolveremos un status 404.

```
17 // Obtener una película por ID
18 router.get('/:id', (req, res) => {
19     const movie = movies.find(m => m.id === parseInt(req.params.id));
20     if (!movie) return res.status(404).send('Movie not found');
21
22 });
```



# Utilizando parámetros en la ruta

Finalmente, si movie tiene una película, procedemos a retornarla como fin de la función.

```
17 // Obtener una película por ID
18 router.get('/:id', (req, res) => {
19     const movie = movies.find(m => m.id === parseInt(req.params.id));
20     if (!movie) return res.status(404).send('Movie not found');
21     res.json(movie);
22 });
```

El código escrito anteriormente, es sólo una de las formas en que podemos recorrer un array y obtener la información que necesitamos.

Existen varias formas de poder llegar a este resultado y cada programador puede pensarlo de una forma diferente.

# Utilizando parámetros en la ruta

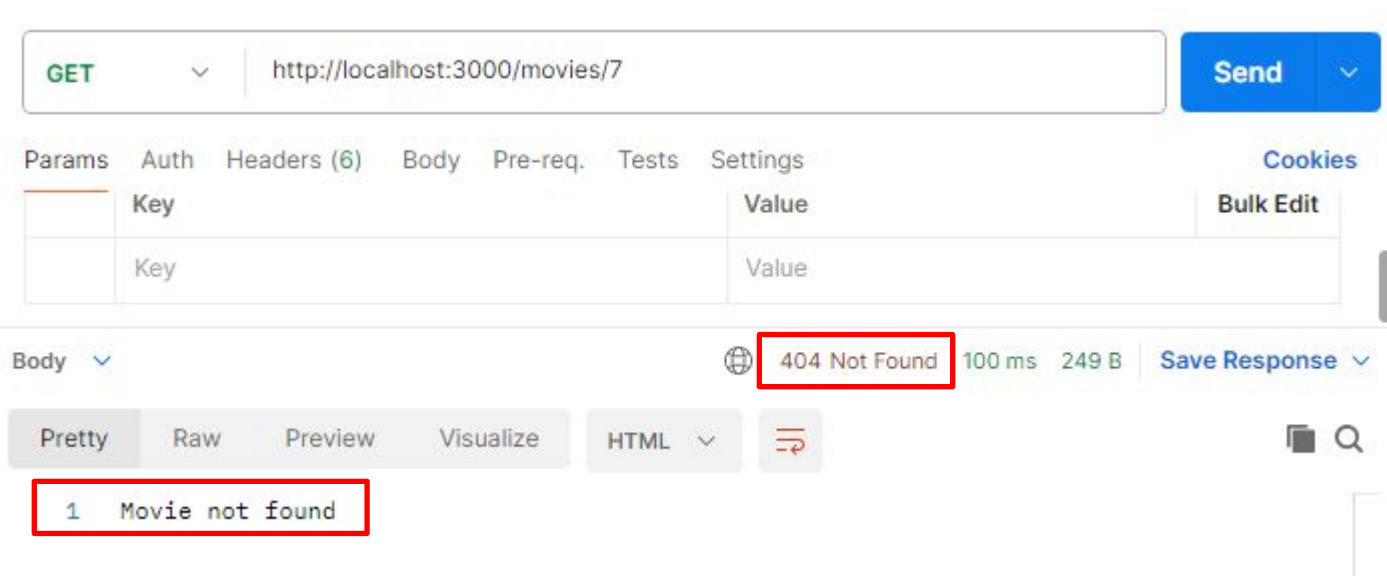
Si desde postman ahora solicitamos el id 2, obtendremos el siguiente resultado:

The screenshot shows the Postman interface. At the top, a GET request is configured to `http://localhost:3000/movies/2`. Below the request bar, the 'Params' tab is active, showing an empty table with columns 'Key' and 'Value'. The 'Body' tab is also visible. The response section shows a status of 200 OK, a time of 7 ms, and a size of 311 B. The response body is displayed in JSON format:

```
1 {
2   "id": 2,
3   "title": "Relatos Salvajes",
4   "director": "Damián Szifron",
5   "year": 2014
6 }
```

# Utilizando parámetros en la ruta

Y si ingresamos un id que no existe:



The screenshot shows a REST client interface with the following details:

- Method:** GET
- URL:** http://localhost:3000/movies/7
- Response Status:** 404 Not Found (highlighted with a red box)
- Response Time:** 100 ms
- Response Size:** 249 B
- Response Body:** 1 Movie not found (highlighted with a red box)

Key	Value
Key	Value

# Ahora probemos con POST.

# Parseando datos recibidos

Hasta el momento venimos trabajando con **GET**, pero para que nuestro servidor pueda recibir peticiones por **POST** necesitamos convertir los **datos recibidos** en el **BODY** a un formato que **entienda el servidor**.

Usando los **middlewares** nativos `.urlencoded()` y `.json()` podemos convertir la data de estos formatos a uno que el **servidor pueda manejar**.

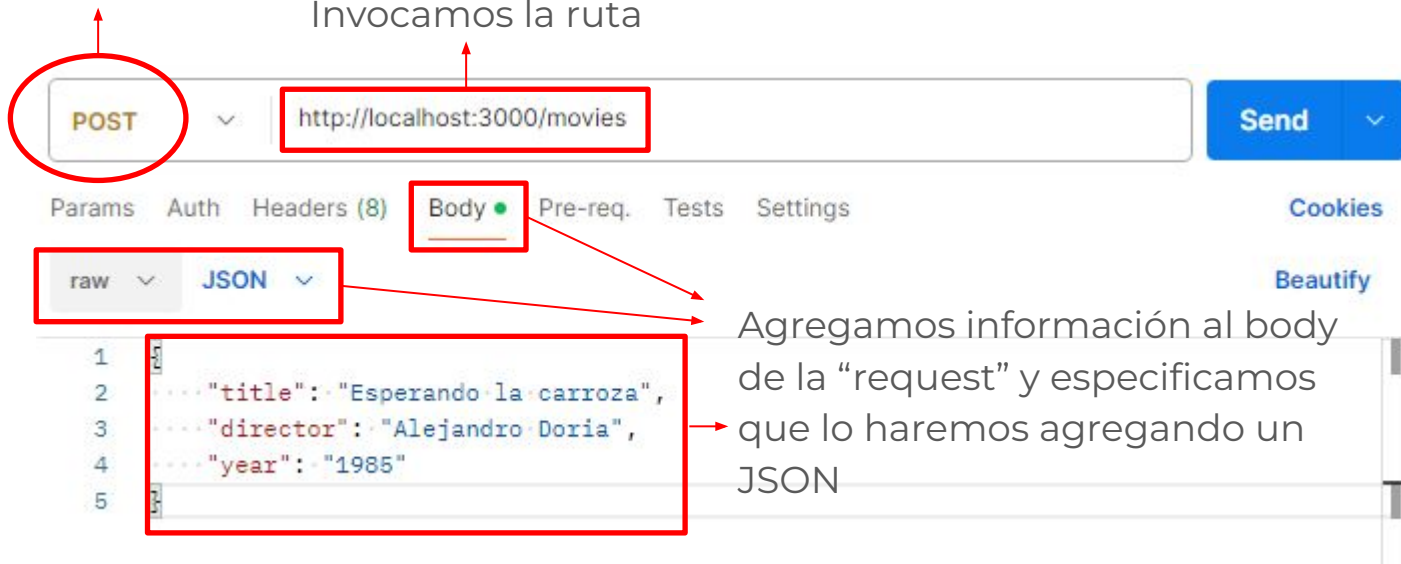
Para poder armar una request en POSTMAN que nos permita enviar una nueva película para cargar a nuestro array de películas, deberemos proceder de la siguiente manera:

*\*nota: en versiones previas a **express 4.16.0** se utilizaba una librería llamada **body-parser** para este propósito.*

# Enviando la petición al servidor

Indicamos el método

Invocamos la ruta



# Programando en el servidor el POST

Al igual que en los casos anteriores, creamos la ruta en la cual recibiremos la request con la información para agregar a nuestro array.

```
24 // Crear una nueva película
25 router.post('/', (req, res) => {
26
27 });
```

En este caso, tenemos dentro de req, la propiedad body, que en su interior recibe los parámetros enviados a través del POST en el frontend (o en este caso desde postman).

Podemos acceder a los valores que necesitamos a través de:

**req.body.title**

**req.body.director**

**req.body.year**

```
25 router.post('/', (req, res) => {  
26     const newMovie = {  
27         id: movies.length + 1,  
28         title: req.body.title,  
29         director: req.body.director,  
30         year: req.body.year  
31     };  
32     movies.push(newMovie);  
33  
34 });
```

gar a

Al igual que en el GET, la manera de resolver cómo agregar la película al array puede diferir de acuerdo a la solución que estemos pensando.



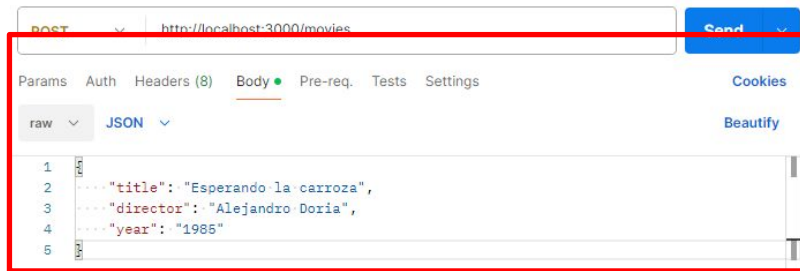
# Programando en el servidor el POST

Ahora, lo único que falta es determinar que vamos a responder como resultado de esta petición. Por convención, en general en un pedido “post” se suele devolver el id generado o bien el objeto completo. En este caso, vamos a devolver el objeto completo bajo el status 201.

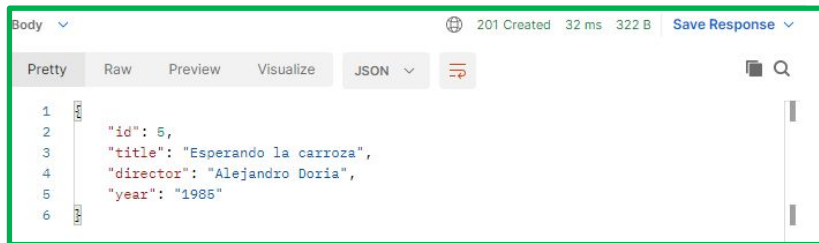
```
24 // Crear una nueva película
25 router.post('/', (req, res) => {
26     const newMovie = {
27         id: movies.length + 1,
28         title: req.body.title,
29         director: req.body.director,
30         year: req.body.year
31     };
32     movies.push(newMovie);
33     res.status(201).json(newMovie);
34 });
```

# Probando la petición en POSTMAN

Ahora, lo único que falta es determinar que vamos a responder como resultado de esta petición. Por convención, en general en un pedido “post” se suele devolver el id generado o bien el objeto completo. En este caso, vamos a devolver el objeto completo bajo el status 201.



REQUEST

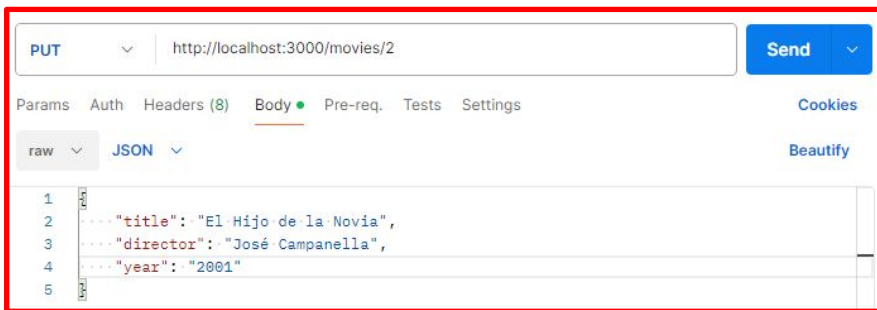


RESPONSE

# Metodo PUT

El método PUT se utiliza para reemplazar completamente un recurso existente, a diferencia de PATCH que realiza modificaciones parciales.

En este caso, en el método PUT enviamos por parámetro en la URL el id de la película que queremos modificar y en el body el nuevo objeto que reemplazara al anterior:



REQUEST



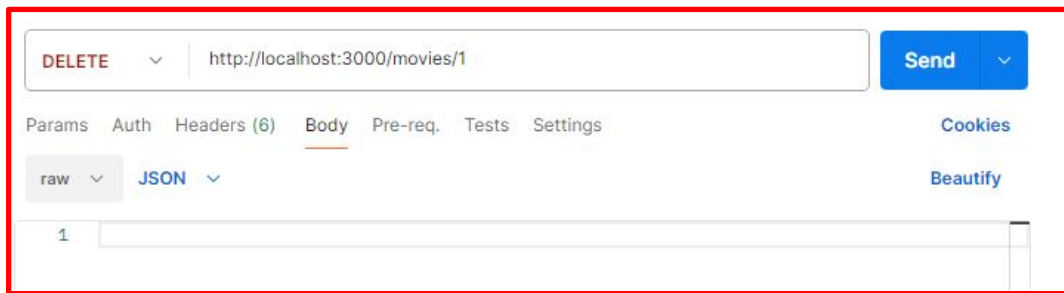
RESPONSE

Y del lado del servidor, a través de Javascript, gestionamos la modificación de la película correspondiente. La respuesta en el status será un 200, lo que indica que se procesó exitosamente. Como es la respuesta por defecto, no es necesario hacerla explícita en el código.

```
36 // Actualizar una película por ID
37 router.put('/:id', (req, res) => {
38     const movie = movies.find(m => m.id === parseInt(req.params.id));
39     if (!movie) return res.status(404).send('Movie not found');
40
41     movie.title = req.body.title || movie.title;
42     movie.director = req.body.director || movie.director;
43     movie.year = req.body.year || movie.year;
44
45     res.json(movie);
46 });
```

# Método DELETE

Se utiliza para eliminar un recurso específico en el servidor. Después de realizar la solicitud, el recurso se eliminará. En el método DELETE, se puede optar por una respuesta standard o devolver el objeto que se eliminó.



REQUEST



RESPONSE

Se utiliza para eliminar un recurso específico en el servidor. Después de realizar la solicitud, el recurso se eliminará. En el método DELETE, se puede optar por una respuesta standard o devolver el objeto que se eliminó.

```
48 // Eliminar una película por ID
49 router.delete('/:id', (req, res) => {
50     const movieIndex = movies.findIndex(m => m.id === parseInt(req.params.id));
51     if (movieIndex === -1) return res.status(404).send('Movie not found');
52
53     const deletedMovie = movies.splice(movieIndex, 1);
54     res.json(deletedMovie);
55 });
56
```



# Middlewares

Casi de forma instintiva, ya **utilizamos varios middlewares** para configurar nuestro programa.

## ¿Qué es un middleware entonces?

Son simplemente funciones que se ejecutan antes o después de otras y los hay de distintos tipos:

- **Middlewares de nivel de aplicación:** Son middlewares que se aplican a toda la aplicación y se configuran utilizando **app.use()**
- **Middlewares de nivel de ruta:** Son middlewares que se aplican a una ruta específica.
- **Middlewares de manejo de errores:** Son middlewares especiales que se utilizan para manejar errores en la aplicación.

**A lo largo de las clases iremos viendo distintos middlewares y también aprenderemos a crear los propios.**

# No te olvides de dar el presente

## **Recordá:**

- **Revisar la Cartelera de Novedades.**
- **Hacer tus consultas en el Foro.**
- **Realizá los ejercicios obligatorios.**

**Todo en el Aula Virtual.**