



Agencia de
Aprendizaje
a lo largo
de la vida

FULL STACK FRONTEND

Clase 28

Node 4
(Parte I)

Les damos la bienvenida

Vamos a comenzar a grabar la clase

Clase 27

Node

- Express
- Servidor estático
- Nodemon
- Rutas

Clase 28

Node

- 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.

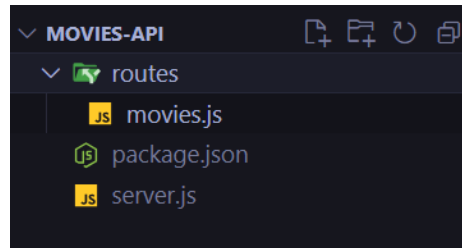
Para ello vamos a el proyecto:

```
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
```

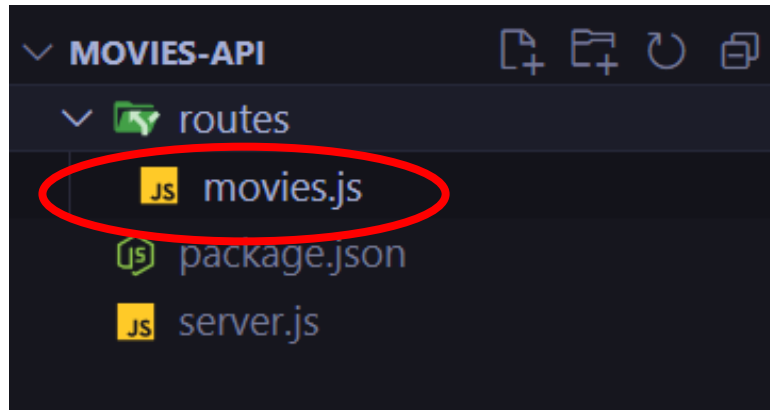
Express Router

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

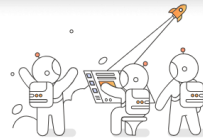
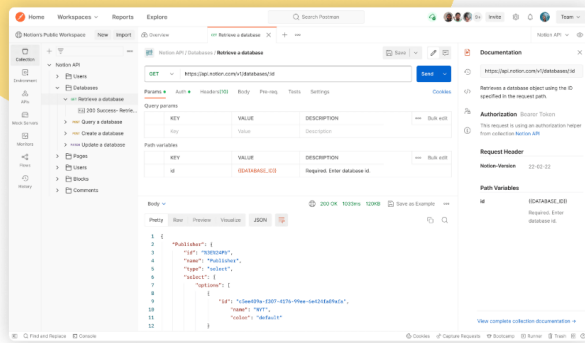
Build APIs together

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

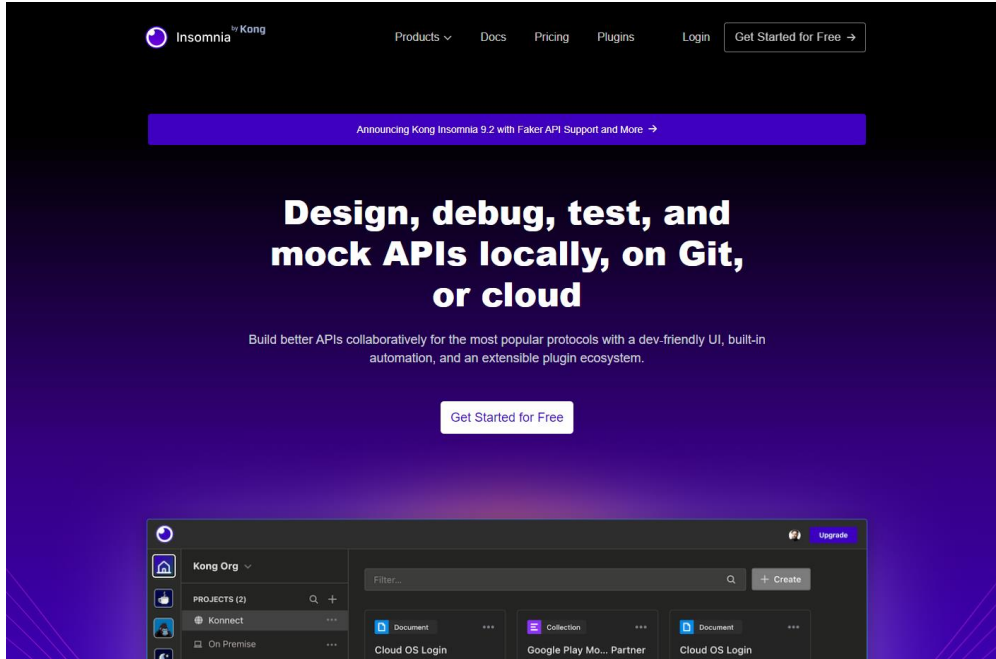
Improve the quality of APIs with governance

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**.

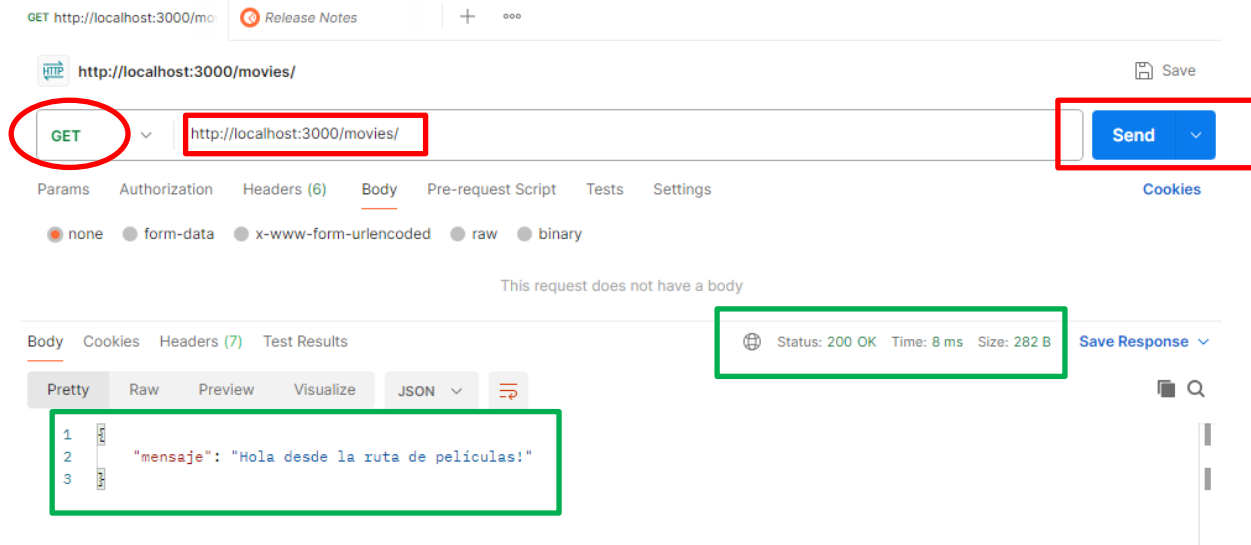
Estas herramientas facilitan enviar consultas a nuestro servidor sin **depende de un Frontend armado**.

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.

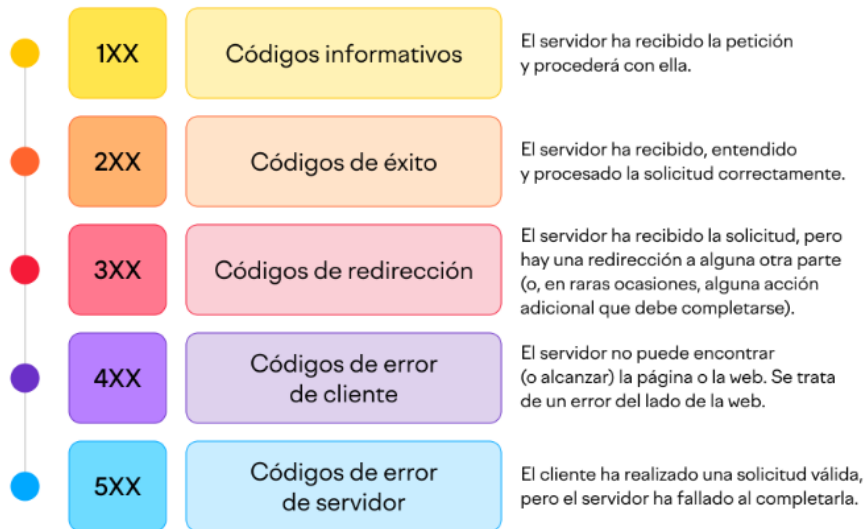


Finalmente presionamos **SEND** y esperamos la **respuesta** del servidor

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.

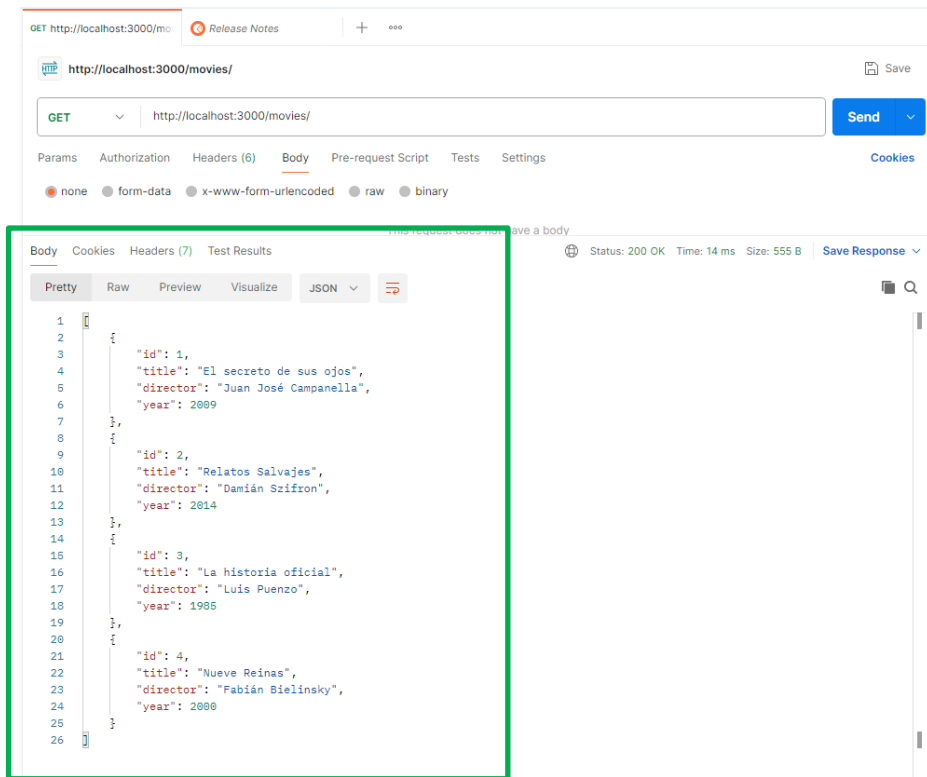
Estas respuestas en ocasiones, las programaremos nosotros desde el backend para informar determinados resultados al frontend.

Mejorando la respuesta con GET

Próximamente veremos como conectar nuestro backend con una base de datos. Por el momento vamos a crear una variable que contenga un listado de películas. De esta manera nuestro backend podrá devolver información estructurada en formato JSON a las peticiones que se realicen desde Postman o desde el frontend.

```
routes > movies.js > 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 información la recibimos en un frontend, como resultado a una petición fetch, podremos manipular la información para mostrarla en el documento HTML de acuerdo a las necesidades de la aplicación.

Utilizando parámetros en la ruta

Es posible utilizar el GET para obtener una película específica. Podemos enviar en la petición, por ejemplo, el id 2 y trabajar con ese parámetro recibido para obtener la película y devolverla como respuesta.

```
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 // Obtener una película por ID
18 router.get('/:id', (req, res) => {
19   // ...
20 });
21
22 module.exports = router;
```

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



GET Send

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 response time of 7 ms, and a size of 311 B. The response body is displayed in JSON format, showing details for a movie with id 2.

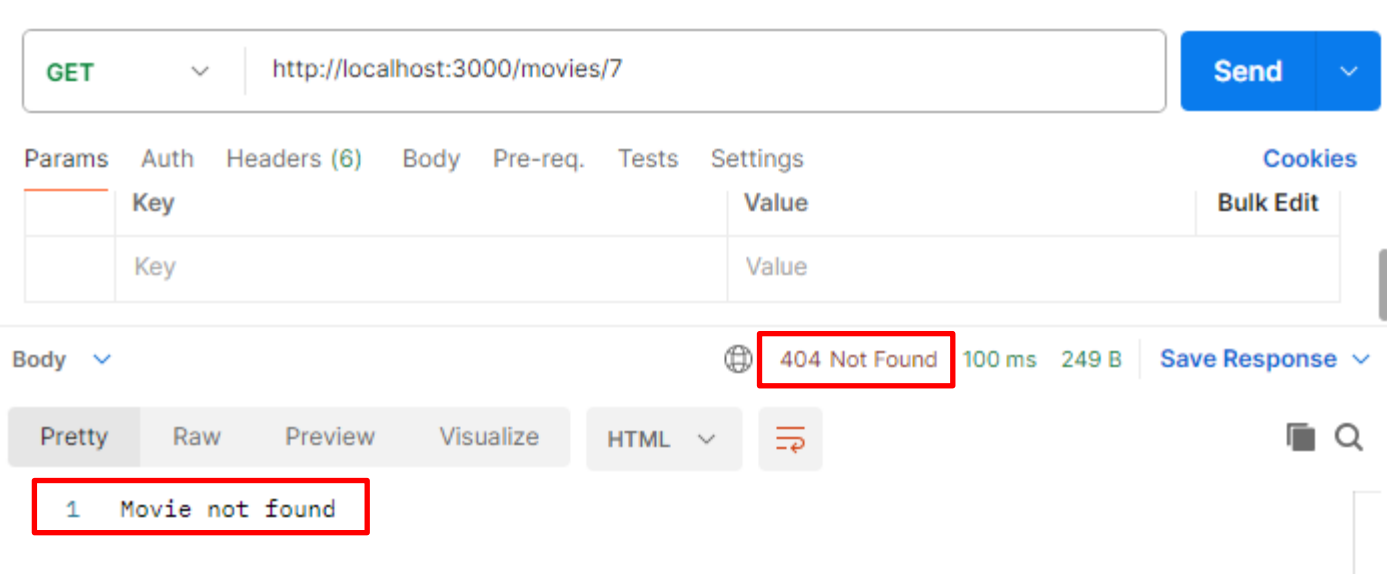
Key	Value

Body 200 OK 7 ms 311 B [Save Response](#)

```
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 components:

- Request Bar:** Method `GET` and URL `http://localhost:3000/movies/7`. A `Send` button is on the right.
- Params Tab:** A table with columns `Key` and `Value`. It contains one empty row.
- Response Bar:** Shows a `404 Not Found` status (highlighted with a red box), `100 ms` time, `249 B` size, and a `Save Response` button.
- Response Body:** A tabbed interface with `Pretty` selected. The response content is `1 Movie not found` (highlighted with a red box).

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.