

APUNTES NODE 05/05/2022 - Abel Rios

Peaso de documentación: https://fullstackopen.com/es/part3/node_js_y_express#express

Resumen de teoría:

- Concepto de API
- Request, Response
- Verbose HTTP
- Header
- Body
- Content-type
- Express → Node
- Node.js
- Hilos
- Async / Await
- Postman (Restfull client)

➤ Node.js & Express.js

Documentación Node: <https://nodejs.org/es/docs/>

npm es el gestor de paquetes por defecto que trae Node.js. Express.js es la librería (framework) que vamos a usar para programar nuestro backend. Documentación: <https://www.npmjs.com/package/express>

Las **dependencias** son clases que se instalan con ese paquete y nosotros usaremos después, también existen dependencias externas que tendríamos que instalar cuando sean necesarias.

Node.js es un **entorno en tiempo de ejecución**, lo que significa, que procesa el mensaje en tiempo de ejecución y en tiempo real (no tiene necesidad de compilador).

Express se usará normalmente en páginas o APIs:

- De tratamiento de datos simples
- Cantidad pequeña de datos
- Orientadas para SPA (Single Page Application) o aplicaciones simples

La DNS es el sistema de nombre de dominio. Es un sistema que asocia la información o nombre a una dirección IP (servidor).

¿Qué es una REST API?

Es el intermediario entre el usuario (cliente) y la base de datos. El cliente se comunica con la API con los verbos HTTP: GET, POST, PUT y DELETE. La API se comunica con el cliente con los formatos JSON y XML.

Request/Response

- Request: Objeto que contiene la información que se ha mandado de la aplicación a la API
- Response: respuesta de la API según la solicitud que haya recibido y luego de haber procesado los datos recibidos

Principales verbos usados en peticiones

- GET: para obtener información
- POST: para mandar y crear información
- PUT: para actualizar información ya creada
- DELETE: para eliminar información que ya está almacenada

Navegadores usan sólo el GET, sin embargo las Aplicaciones usan el GET, POST, PUT y DELETE.

¿Cómo se mandan los datos? Del cliente a la API. Distintos métodos:

QueryString

Es cuando tenemos una url en la API, ejemplo:

`http://localhost/login?name=Adri&surname=jimenez`

donde lo amarillo (después de un símbolo de ?) son las variables que vamos a usar en el servidor.

Params (o parámetros)

en el navegador: `http://localhost/login/Adrian/Jimenez`

en el servidor: `login/:name/:surname`

ambos Params y QueryString se suelen usar con GET y DELETE

Body

se suele usar para el POST y el PUT, porque suele ser una cantidad masiva de datos. Usaremos el Postman.

Apuntes Node

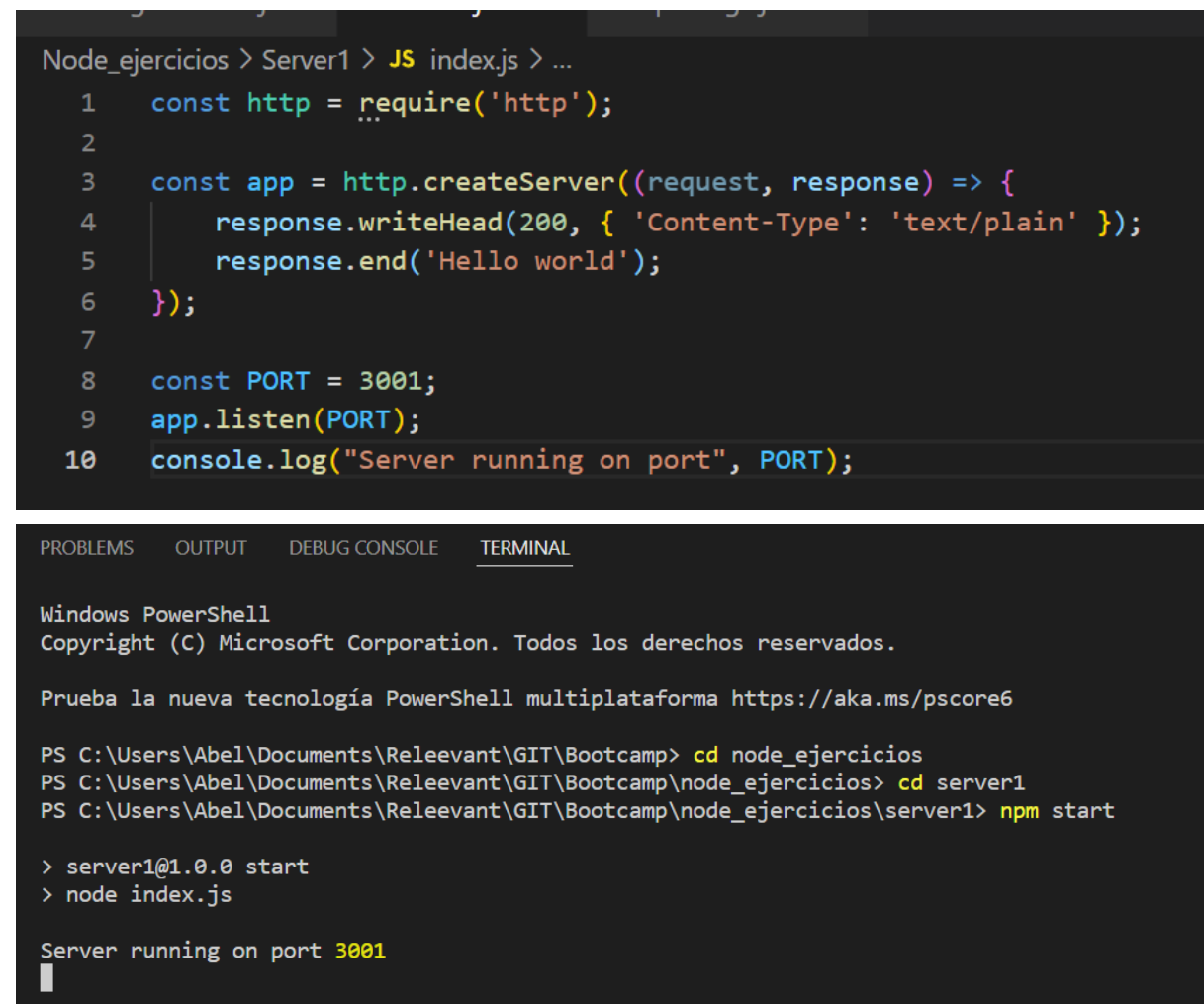
06/05/2022

Resumen de clase:

- Querystring, Params, Body
- Postman
- GET; POST, PUT, DELETE
- Trabajar con datos hardcodeados

La semana que viene ejercicios a tope con SQL, la siguiente con MONGO DB y la siguiente repaso y JSON webtokens.

Abrimos el visual studio y la carpeta donde hemos guardado el servidor. En la terminal ponemos el comando: npm start



The image shows a screenshot of the Visual Studio Code editor. The top part displays a file named `index.js` with the following code:

```
Node_ejercicios > Server1 > JS index.js > ...
1  const http = require('http');
2
3  const app = http.createServer((request, response) => {
4      response.writeHead(200, { 'Content-Type': 'text/plain' });
5      response.end('Hello world');
6  });
7
8  const PORT = 3001;
9  app.listen(PORT);
10 console.log("Server running on port", PORT);
```

The bottom part shows the integrated terminal window with the following output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Prueba la nueva tecnología PowerShell multiplataforma https://aka.ms/pscore6

PS C:\Users\Abel\Documents\Relevant\GIT\Bootcamp> cd node_ejercicios
PS C:\Users\Abel\Documents\Relevant\GIT\Bootcamp\node_ejercicios> cd server1
PS C:\Users\Abel\Documents\Relevant\GIT\Bootcamp\node_ejercicios\server1> npm start

> server1@1.0.0 start
> node index.js

Server running on port 3001
```

Esto es sólo por repasar lo de ayer. Ahora empezamos con express (documentación: https://fullstackopen.com/es/part3/node_js_y_express#express)

En nuestro archivo index ponemos:

```
Node_ejercicios > Server > JS index.js > app.get('/api/notes') callback
1  const express = require('express')
2  const app = express()
3
4  let persona = {
5    name: "Abel",
6    surname: "Rios"
7  }
8
9  app.get('/', (request, response) => {
10    response.send('<h1>Hello World!</h1>')
11  })
12
13  app.get('/api/notes', (request, response) => {
14    response.json(persona)
15  })
16
17  const PORT = 3001
18  app.listen(PORT, () => {
19    console.log(`Server running on port ${PORT}`)
20  })
```

con el comando 'require' le decimos qué librería vamos a usar, en este caso Express, que al instalarlo nos sale en el package.json en forma de "dependencia":

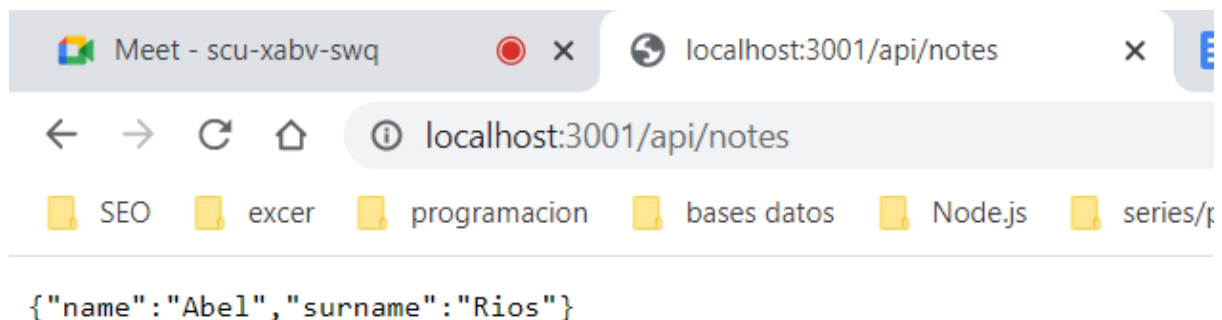
```
10    "author": "AbelRios",
11    "license": "MIT",
12    "dependencies": {
13      "express": "^4.18.1"
14    }
```

Ahora para hacer una llamada al servidor lo hacemos así:

```
app.get('/api/notes', (request, response) => {
```

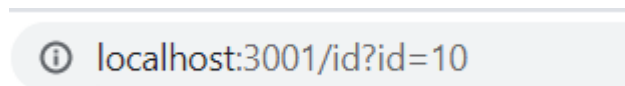
Usamos **app**, que hemos definido en la línea 2, que es nuestra llamada a la librería (express). Después usamos el **verbo** (get, post, put, delete). Después la **url** ('/api/notes') eso por dentro sería <http://localhost:8000/api/notes>, pero nuestra primera parte de la url queda obviada. Esta url puede ser la que nosotros queramos, le ponemos el "nombre" a la url que prefiramos o bueno, que sea el propio. Después va el **request** que es el objeto que tiene mi información y después el **response** que es el objeto que manda la respuesta (objeto que quiero que reciba mi aplicación).

Con esto, si vamos a <http://localhost:3001/api/notes> nos saldrá:

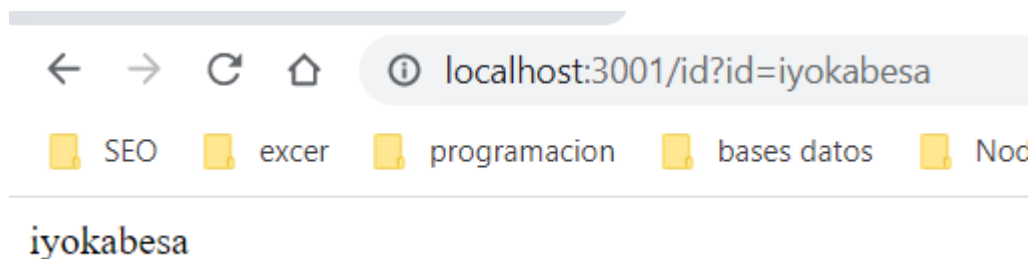


Querystring

Ahora vamos a probar a hacer una nueva url que se llame "id". En el navegador vamos a llamarla tal que así, pasándole una variable por querystring:



y en el servidor, vamos al objeto "request" que es el que tiene la información que estamos pasando y le ponemos "query" y la variable que creamos (id). ¿Qué hace eso? Pues vemos:



¿Porqué hace eso? Pues vemos:

```
32
33   app.get('/id', (request, response) => {
34
35       let id = request.query.id;
36       console.log(id);
37
38       response.send(id);
39   })
```

Y al hacer el console log en ese get, nuestra consola va almacenando los valores distintos que le vamos a dar a la variable id desde nuestra url del navegador:

```
Server running on port 3001
10
gorrion
iyokabesa
```

Podemos hacer esto mismo con dos variables distintas y usando querystring también:

← → ↻ 🏠 ⓘ localhost:3001/id?id=iyokabesa&aux=olakease

SEO excer programacion bases datos Node.js

```
{"nombre":"iyokabesa","apellido":"olakease"}
```

```
Server running on port 3001
iyokabesa olakease
```

```

32
33   app.get('/id', (request, response) => {
34
35       let id = request.query.id;
36       let aux = request.query.aux;
37
38       console.log(id, aux);
39
40       const respuesta = {
41           nombre: id,
42           apellido: aux
43       }
44
45       response.send(respuesta);
46   })

```

Ahora para el siguiente ejercicio instalamos Nodemon con el comando:

```
npm install nodemon
```

Y modificamos el package.json y añadimos un nuevo script llamado dev:

```

6   "scripts": {
7       "dev": "nodemon index.js",
8       "start": "node index.js",
9       "test": "echo \"Error: no test specified\" && exit 1"
10  },

```

Y ahora hacemos en la terminal: npm run dev

Con nodemon arrancado, si modificamos alguna de las url's que hemos creado anteriormente y guardamos el archivo, vemos como nodemon resetea el servidor y así no tenemos que pararlo y volverlo a iniciar manualmente nosotros (seguro que tiene más medidas que aprenderemos más adelante).

Params (pasar info a través de los Params)

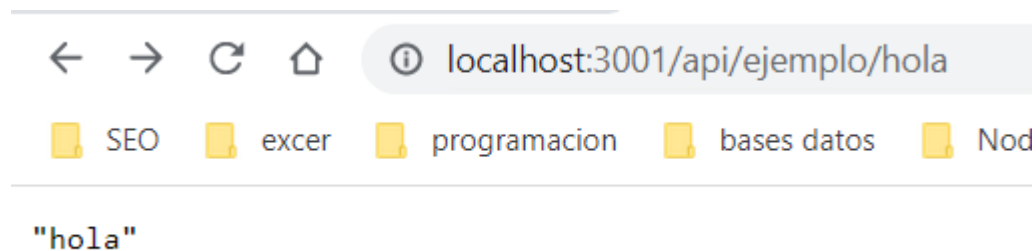
Pongamos como ejemplo la url **/api/note/10** , esta es la forma que se ve en el navegador, pero en el servidor nuestra url se ve tal que **'api/note/:numero'**. Eso lo usaremos con el request, que es un objeto, y añadir params.

```

48 app.get('/api/ejemplo/:numero', (request, response) => {
49
50     request.params.numero;
51
52     response.json(request.params.numero);
53 })

```

Y nos devuelve:

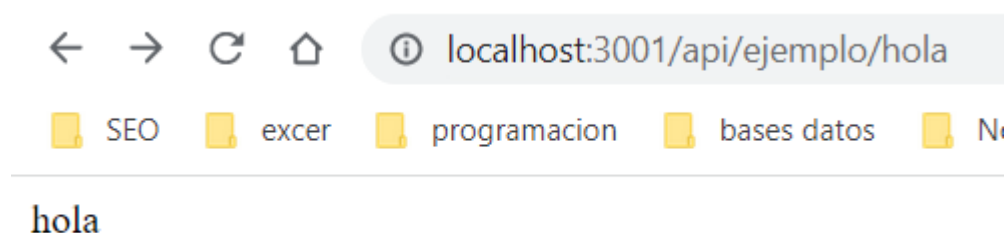


y si en lugar de hacer response.json hacemos response.send, no nos lo envía como objeto:

```

51 app.get('/api/ejemplo/:numero', (request, response) => {
52
53     console.log(request.params.numero);
54
55     response.send(request.params.numero);
56 })

```

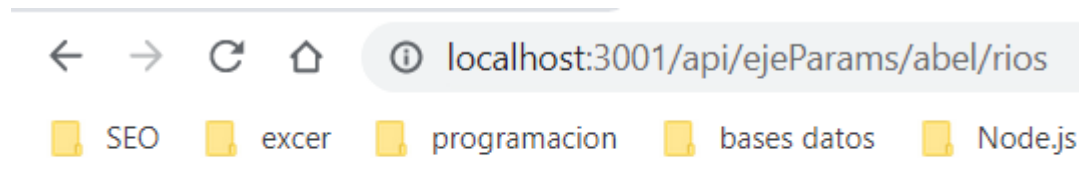


y para hacerlo concatenando dos params distintos:

```

58 app.get('/api/ejeParams/:nombre/:apellido', (request, response) => {
59
60     console.log(request.params);
61
62     response.send(request.params);
63 })

```

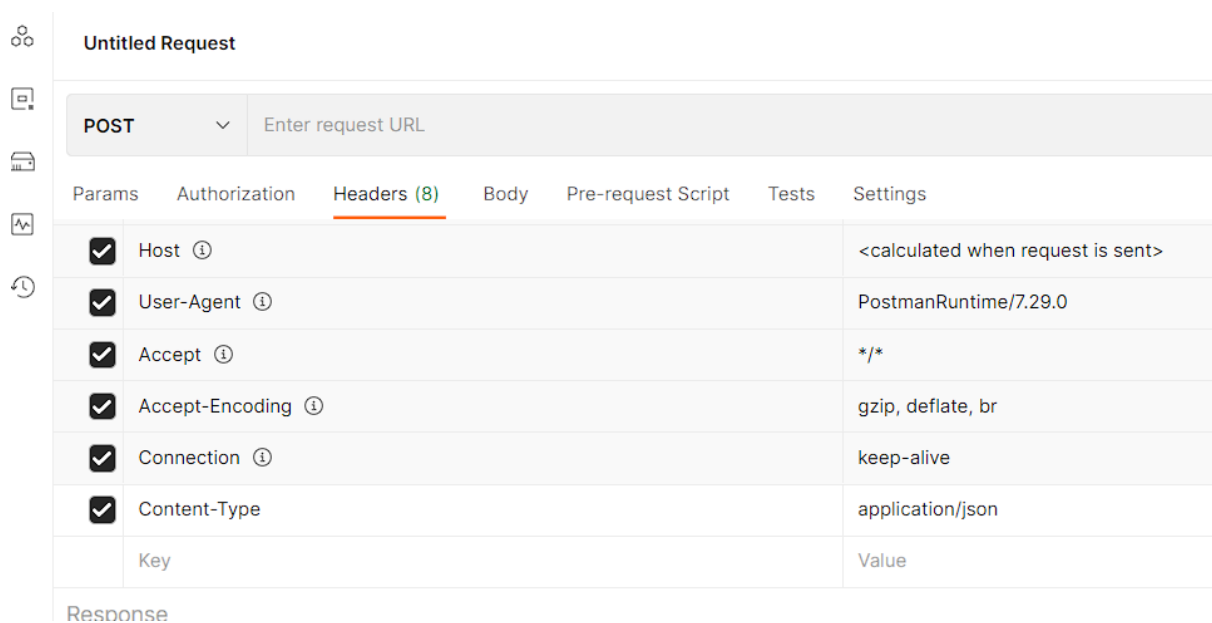



```
{"nombre":"abel","apellido":"rios"}
```

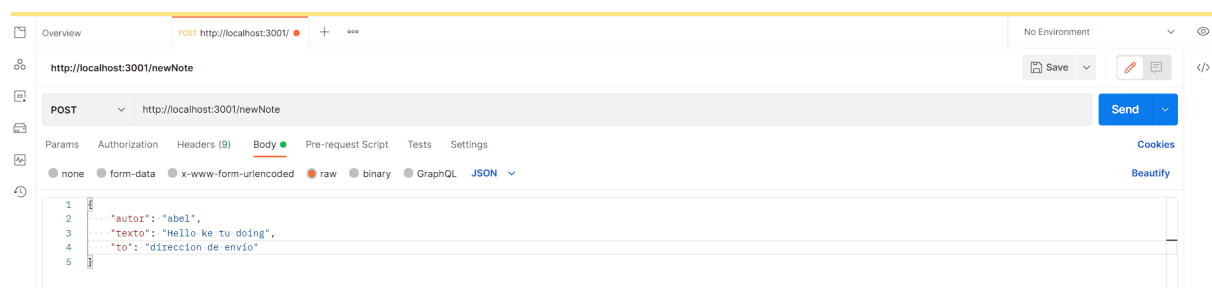
Postman

Cuando mandamos algo a través de postman podemos hacer un header, y en el header mandamos los metadatos, que sirven para indicar al servidor el tipo de archivos o datos que está recibiendo.

Vamos a crearlo, en Key decimos que sea Content-Type, y en Value que sea application/json. Con esto estamos diciendo que la información que vamos a enviar y que el servidor va a recibir es un json.



Ahora vamos a crear el body y enviarlo:



Si creamos también un nuevo endpoint:

```
68   app.post('/newNote', (request, response) => {  
69     console.log(request.body);  
70     response.json(request.body);  
71   })
```

y pulsamos 'send' en el postman, en la terminal del visual studio nos dará undefined, porque no es capaz de leer el json que estamos mandando en el body de la request. Para ello añadimos al principio de nuestro index.js la línea 5:

```
Node_ejercicios > Server > JS index.js > [🔗] PORT  
1   const { response } = require('express')  
2   const express = require('express')  
3   const app = express()  
4  
5   app.use(express.json())  
6
```

Ahora al enviar desde el postman sí nos llegan los datos:

```
[nodemon] starting node index.js  
Server running on port 3001  
{ autor: 'abel', texto: 'Hello ke tu doing', to: 'direccion de envío' }  
█
```

EJERCICIOS:

1. Crear un array de notas, cada nota tendrá como parámetros: id, autor, texto y receptor. Sacar por navegador la nota número 3.

```
73  class Nota {
74      constructor(id, autor, texto, receptor){
75          this.id=id,
76          this.autor=autor,
77          this.texto=texto,
78          this.receptor=receptor
79      }
80  };
81
82  let miArray = [];
83
84  let nota1 = new Nota(1,"Paco", "texto1", "Jaime");
85  miArray.push(nota1);
86  let nota2 = new Nota(2, "Marta", "texto2", "Rocio");
87  miArray.push(nota2);
88  let nota3 = new Nota(3, "Lucia", "texto3", "Pedro");
89  miArray.push(nota3);
90  let nota4 = new Nota(4, "Adri", "texto4", "Ramón");
91  miArray.push(nota4);
92  let nota5 = new Nota(5, "Miguel", "texto5", "Sara");
93  miArray.push(nota5);
94
```

```
app.get('/nota/:idnota', (request, response) => {

    let aux = request.params.idnota;

    let respuesta = miArray.find(item => item.id==aux);

    console.log(respuesta);

    response.send(respuesta);
})
```

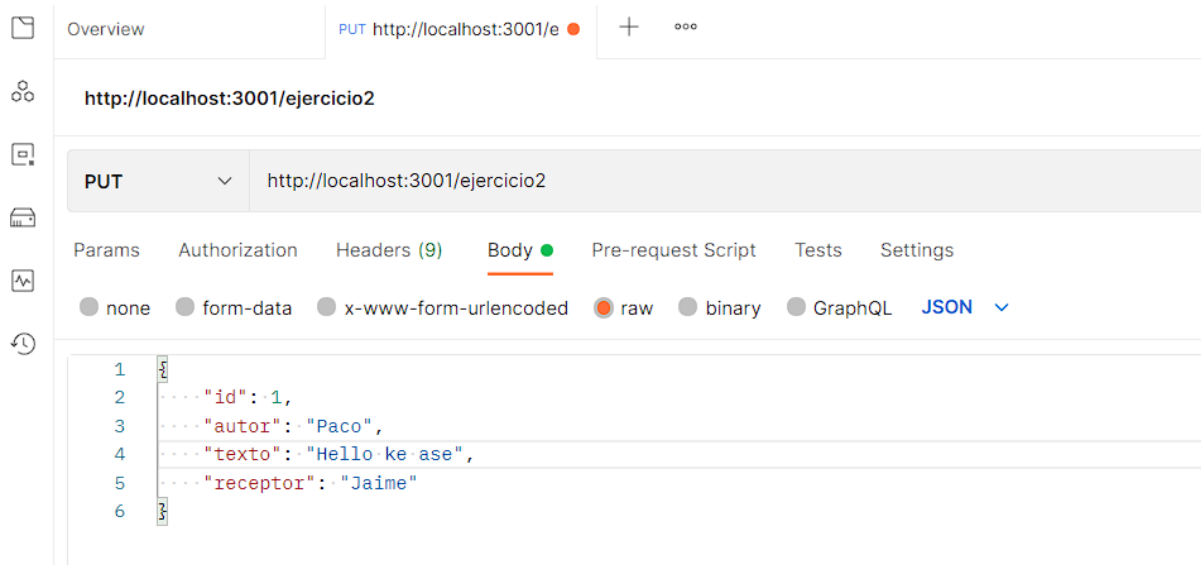
****OJO**** en el find usamos dos iguales porque lo que nos llega en request.params.idnota es un string con el número que sea, no un número. Por lo que no podemos poner los tres iguales.

2. Modificar el texto de una nota (actualizar) dándonos el Autor. Desde el postman mandar una nota con un texto distinto pero un autor que ya tenemos y luego enviar al navegador (response) el array de notas.

****OJO****

Postman sólo sirve para mandar los datos. Y nosotros le indicamos cómo los mandamos (put, post, get, delete) + URL

El servidor es donde tratamos los datos (actualizar, insertar, borrar, obtener).



```
107 app.put('/ejercicio2', (request, response) => {
108
109   let aux = request.body;
110   let indice = miArray.findIndex(item => item.id === aux.id)
111
112   miArray[indice].texto=request.body.texto;
113
114   response.send(miArray);
115
116   console.log(request.body);
117 })
```

o también:

```
119  app.put('/ejercicio2', (request, response) => {
120
121      let aux = request.body;
122      let nota = miArray.find(item => item.id === aux.id)
123
124      nota.texto=aux.texto;
125
126      response.send(miArray);
127
128      console.log(request.body);
129  })
```