**MongoDB**

<https://www.mongodb.com/download-center/community>

sudo apt update

sudo apt install -y mongodb

sudo systemctl status mongodb

mongo --eval 'db.runCommand({ connectionStatus: 1 })'

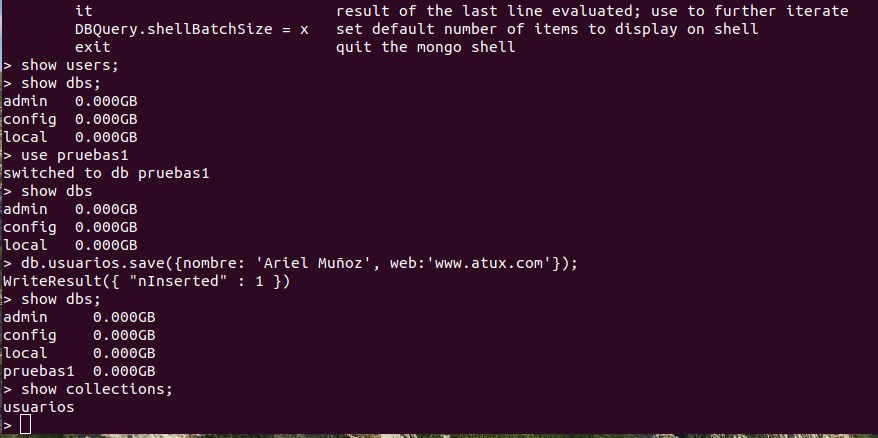
\*\* callback, función que se ejecuta dentro de otra

Crear BBDD:

use pruebas1

Insertar datos:

db.usuarios.save({nombre: 'Ariel Muñoz', web:'www.atux.com'});



show collections; --veo que tablas o colecciones tiene la BBDD

--metodo find() permite buscar documentos dentro de la BBDD



Instalamos un gestor visual de mongo (gui - interfaz gráfica de usuario, graphical user interface):

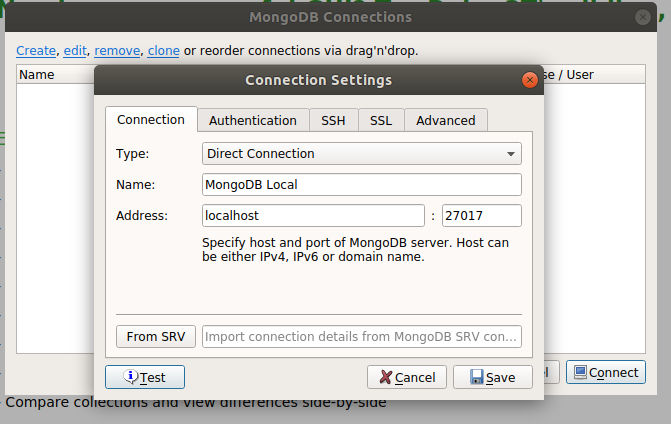
<https://robomongo.org/download>

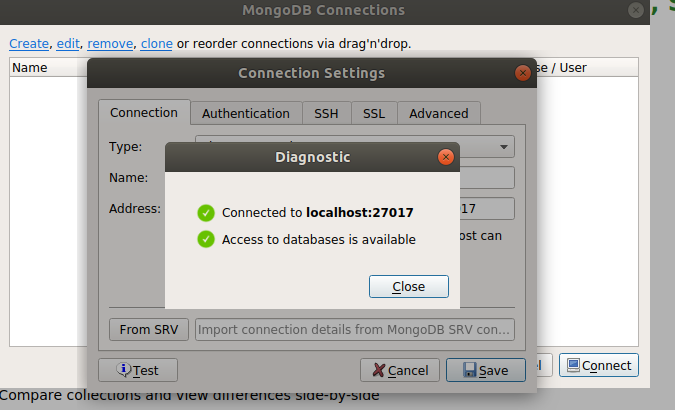
Descargamos sólo el Robo 3T, descomprimimos,

entramos a /bin y ejecutamos:

./robo3t

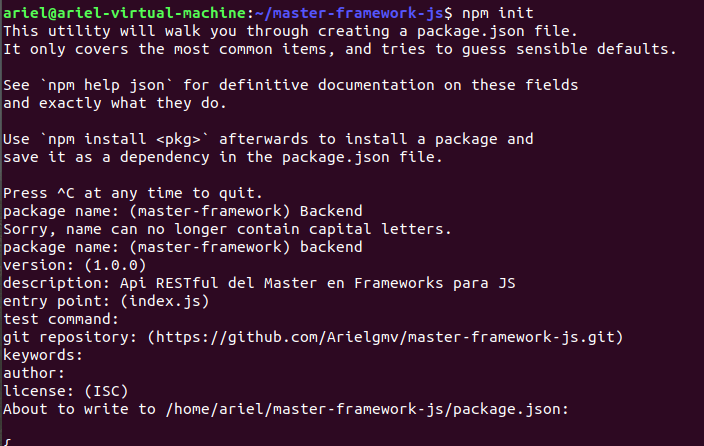
Create

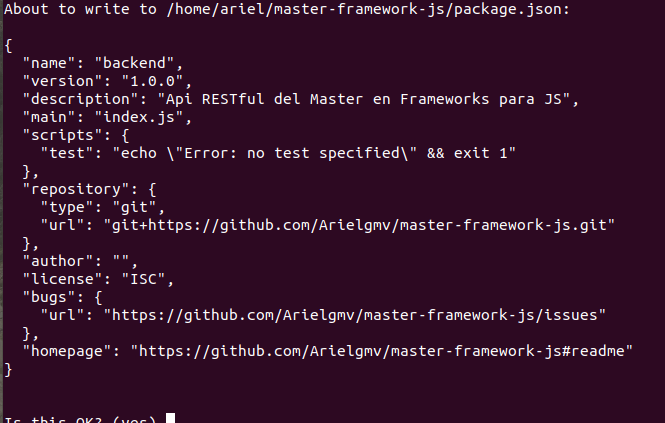




**Nodejs**

npm init





Creo una carpeta (backend), llevo ahí el package.json e instalo las dependencias:

\*\* por lo regular crea una carpeta

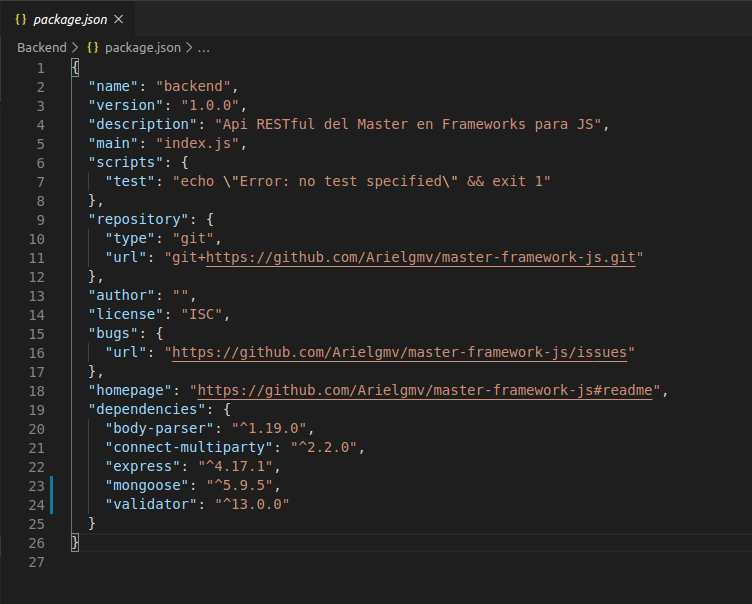
npm install --save express

npm install --save body-parser (convierte datos de un formulario a objeto json)

npm install --save mongoose (para el mongo)

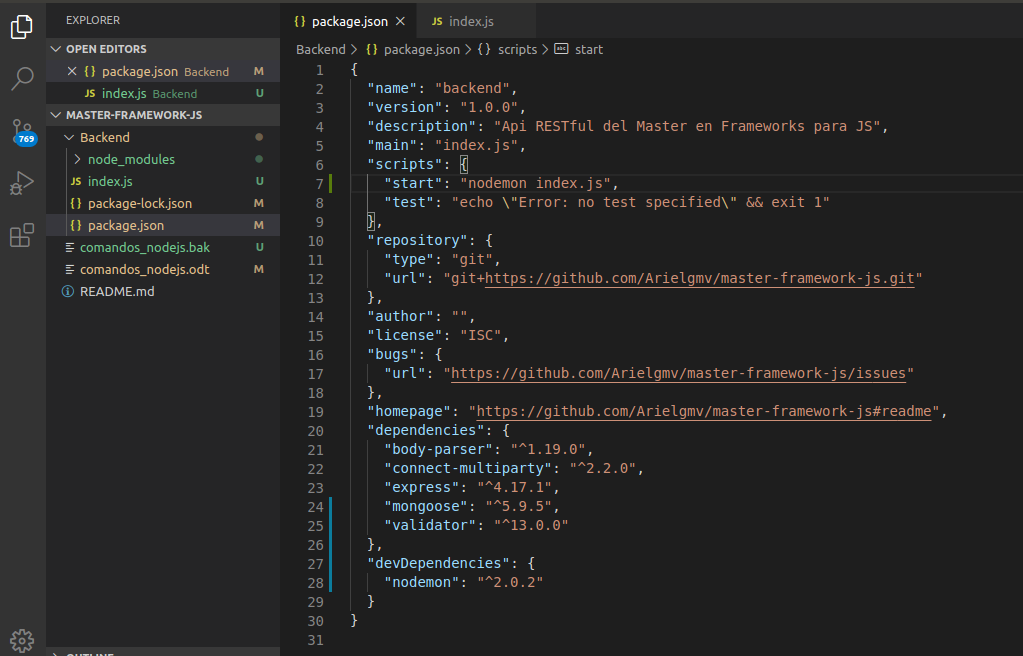
npm install --save connect-multiparty (para gestión de archivos)

npm install --save validator



**Conectar con MongoDB**

npm install --save-dev nodemon (permite reiniciar el servidor a cada cambio de codigo y se ve los cambios)



**Postman**

métodos GET -> navegador procesa

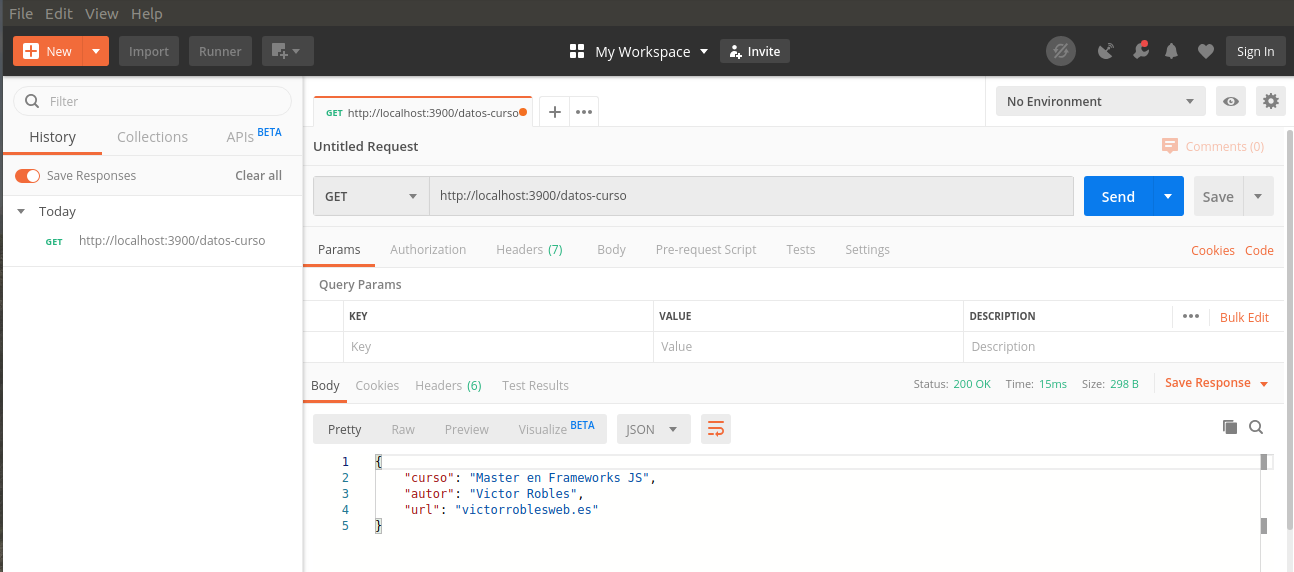
métodos POST -> crear formularios y demás...

métodos PUT ->

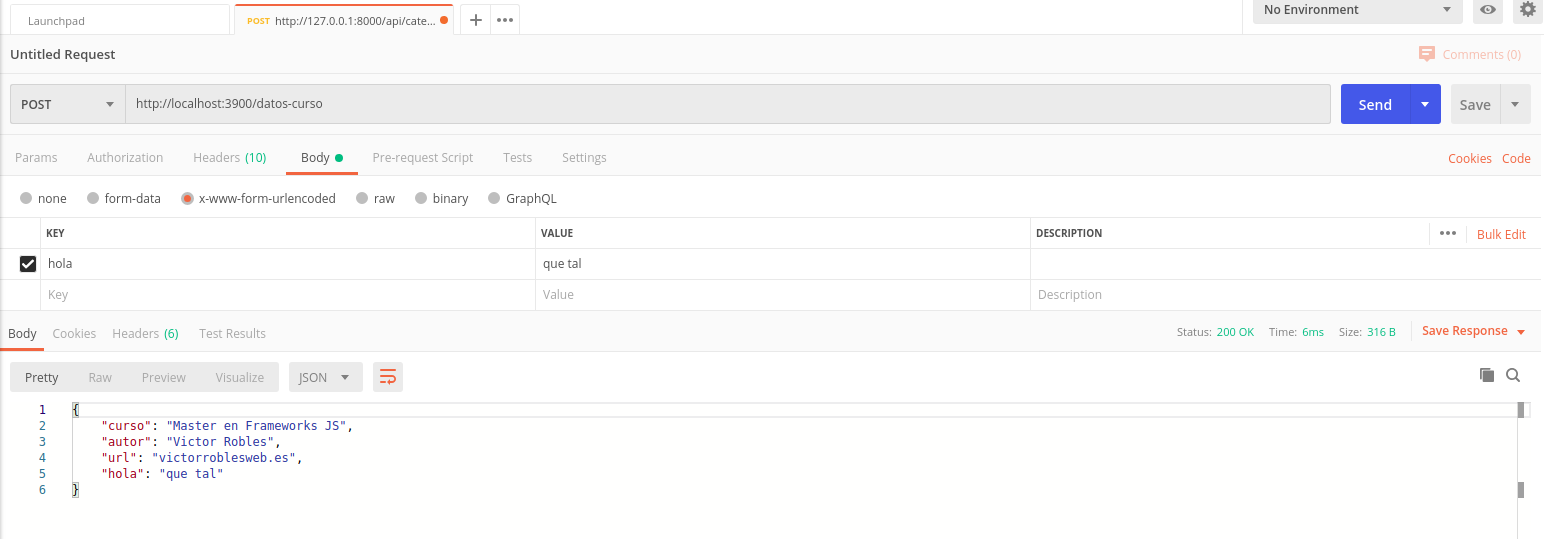
métodos DELETE ->

testear con navegador para todos los métodos no funciona bien, para eso el postman -> hacer pruebas

Postman es un Cliente RESTful



Enviando datos desde el postman (POST)



recibo este parametro en el backend (node) y me esta devolviendo el resultado

Postman me permite testear el API, una vez la tenga lista el API con los diferentes metodos (GET, POST, etc.), podre utilizar esta API en las diferentes aplicaciones del front end o aplicaciones del movil o lo que sea, un API REST puede ser consumida por cualquier dispositivo que entienda el protocolo HTTP.

<https://blog.makeitreal.camp/el-protocolo-http/>

**Arquitectura de la aplicacion web**

mas o menos un MVC

Modelos -> capa de abstraccion con la que nos conectamos a la BBDD, nos devuelve datos o resultados de la BBDD

Controladores -> mediante las rutas vamos a recibir datos, procesarlos y devolver un dato al cliente

Vistas -> no habra vistas como tal, las vistas seran los JSON que nos devuelve cada metodo del API

Tambien separaremos nuestro codigo en rutas, si es necesario tambien en servicios, incluso tambien middlewares para procesar una informacion antes de cargar una ruta o no

**Crear modelos**

modelo no es mas que una clase que nos da un modelo para crear diferentes objetos utilizando ese modelo

capa de abstraccion intermedia que me va a dar diferentes metodos, diferentes funcionalidades, para trabajar con la coleccion de datos o articulos que he creado (BBDD), me servira como molde para generar nuevos articulos o para generar nuevos objetos de este tipo.

article.js es el modelo

si me creo un modelo de un articulo, lo que hago es crear un esquema de ese articulo, definir que propiedades va a tener ese articulo, cada uno de los objetos que yo cree, que estructura va a tener el documento cuando yo lo guarde en la coleccion de mongo, y sobre este modelo yo voy a poder estar conectandome a la coleccion de articulos, voy a poder interactuar con la bbdd

**Controladores y rutas**

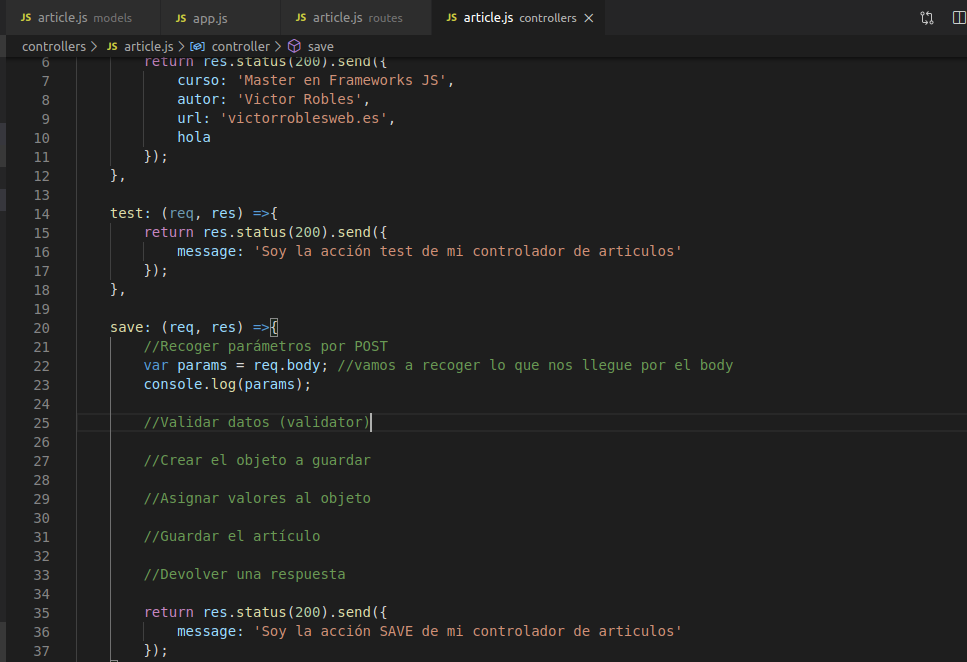
article.js

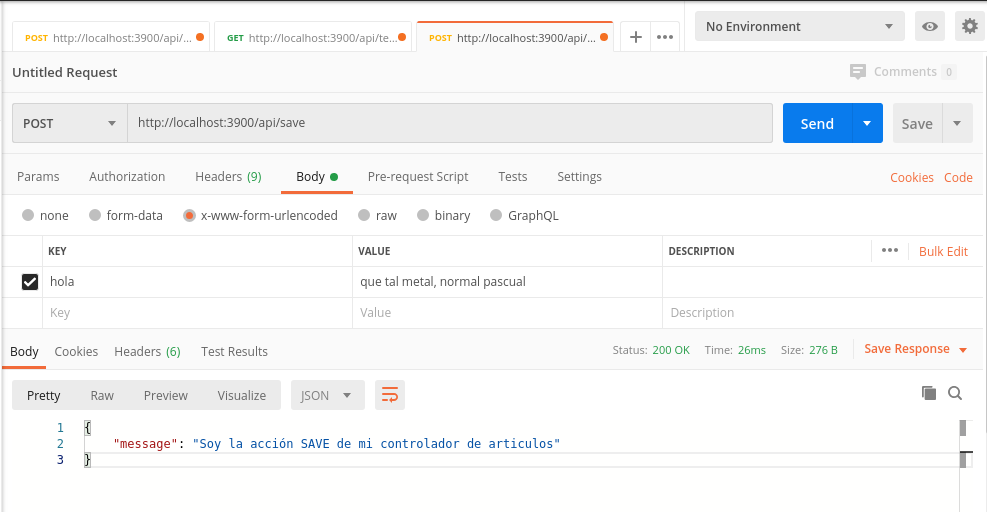
es una clase en la cual vamos a tener los diferentes metodos y rutas relacionadas con articulos de nuestro api, de nuestro backend

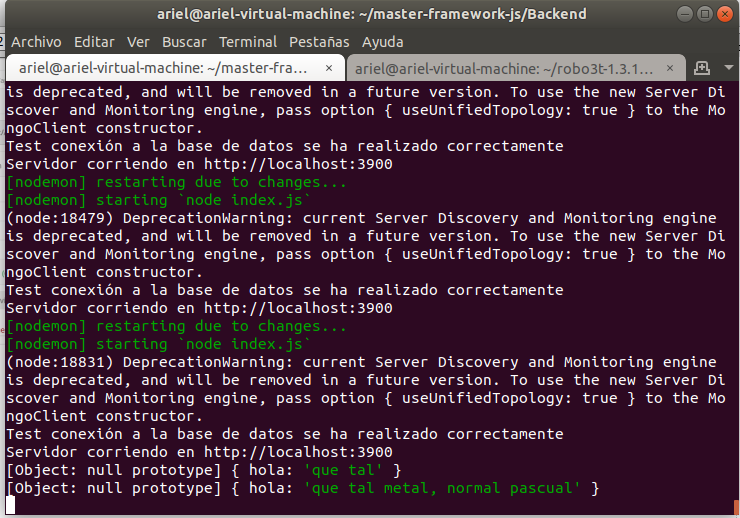
definir ese objeto y definir todos los metodos que va a tener el API

Recoger parámetros por POST:

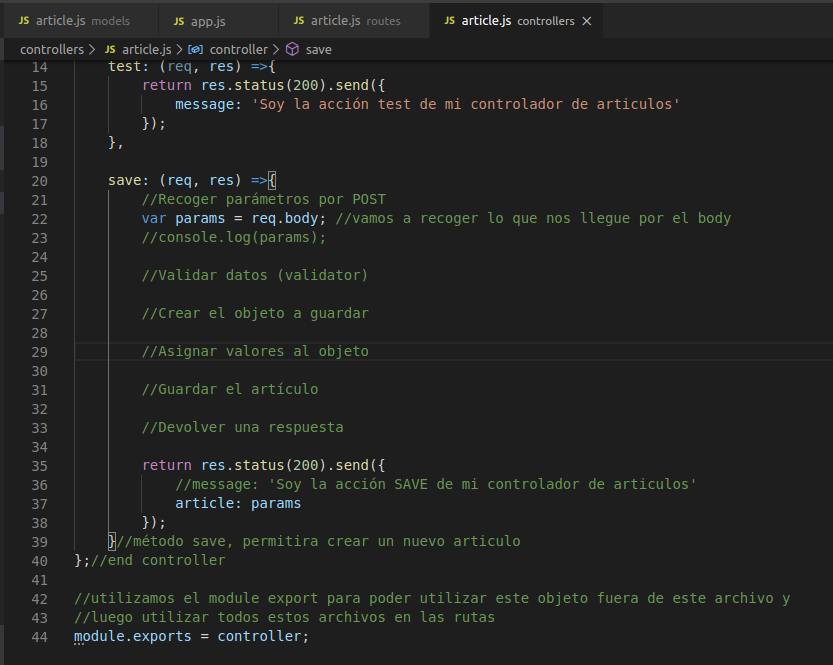
\* Mostramos los datos por consola

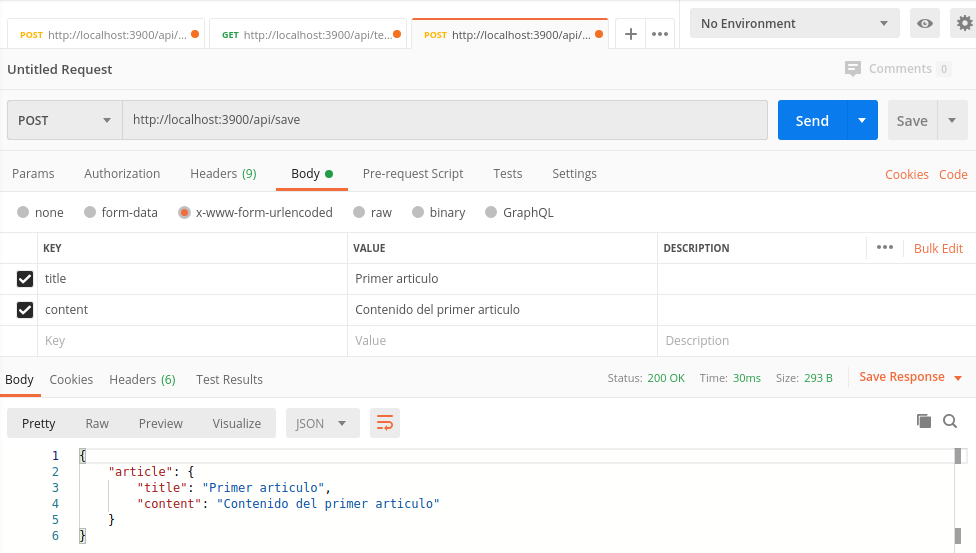






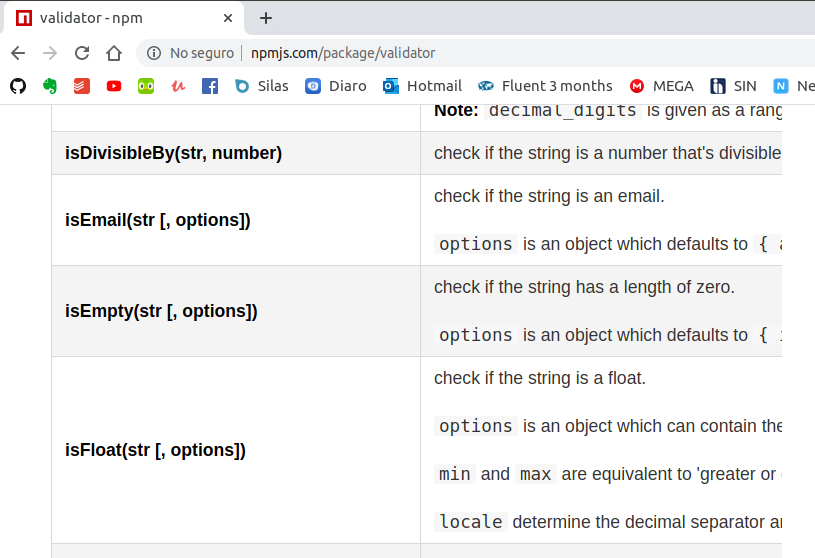
\* Mostramos los datos por postman

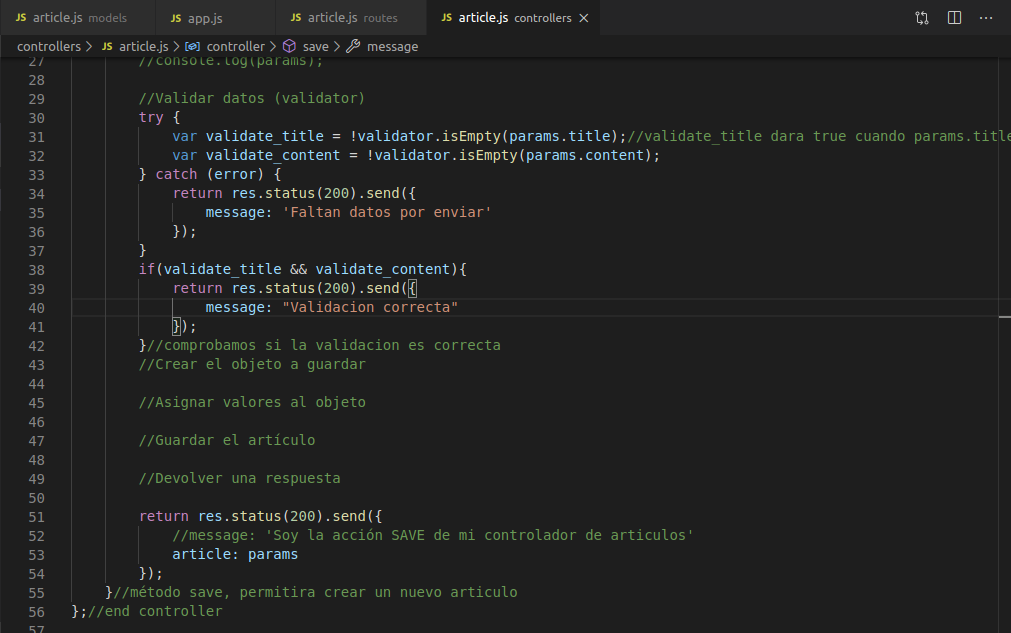


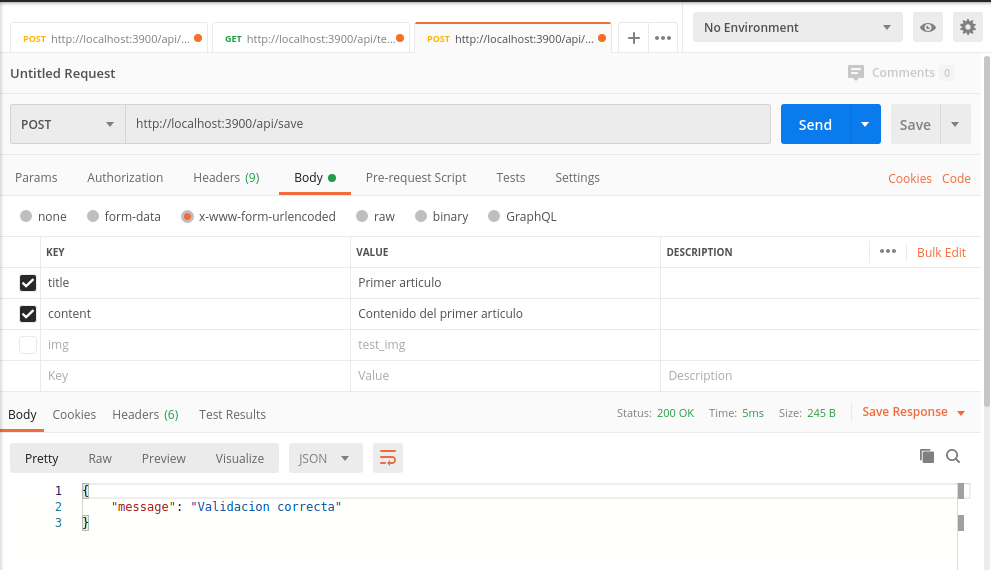


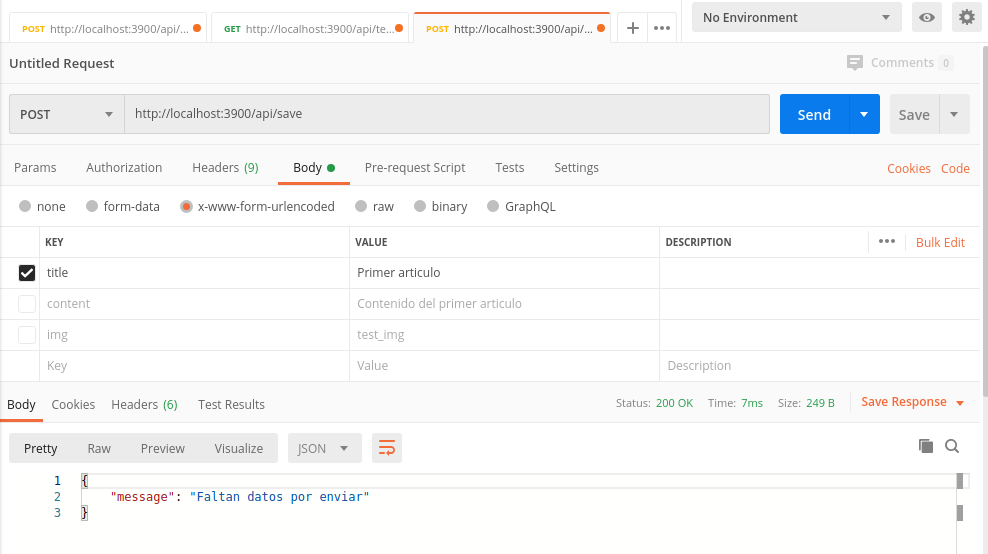
**Validaciones node**

<https://www.npmjs.com/package/validator>

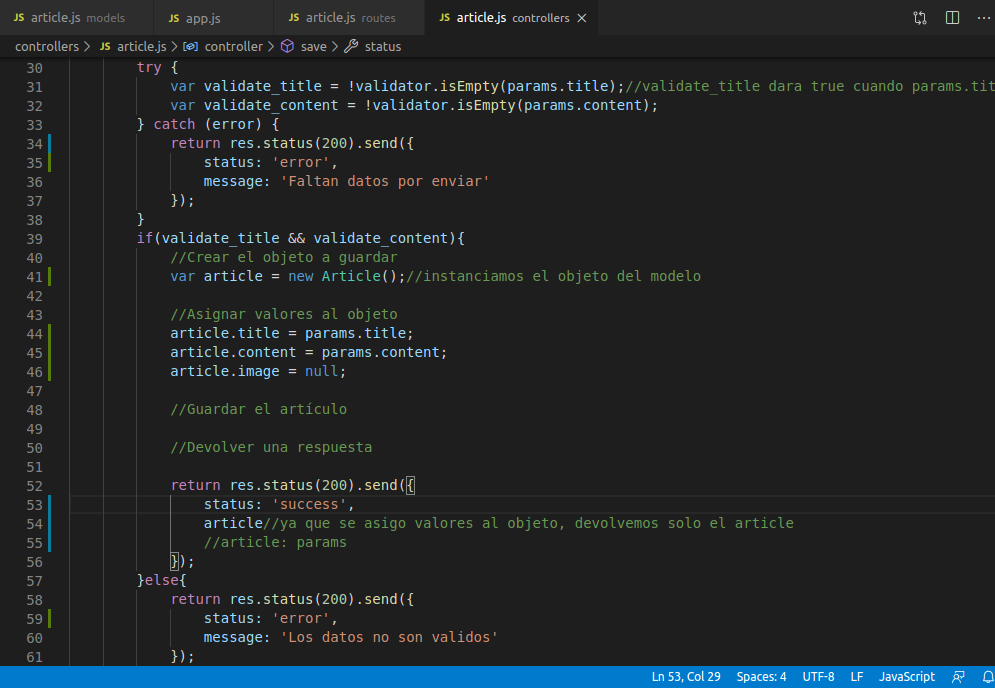


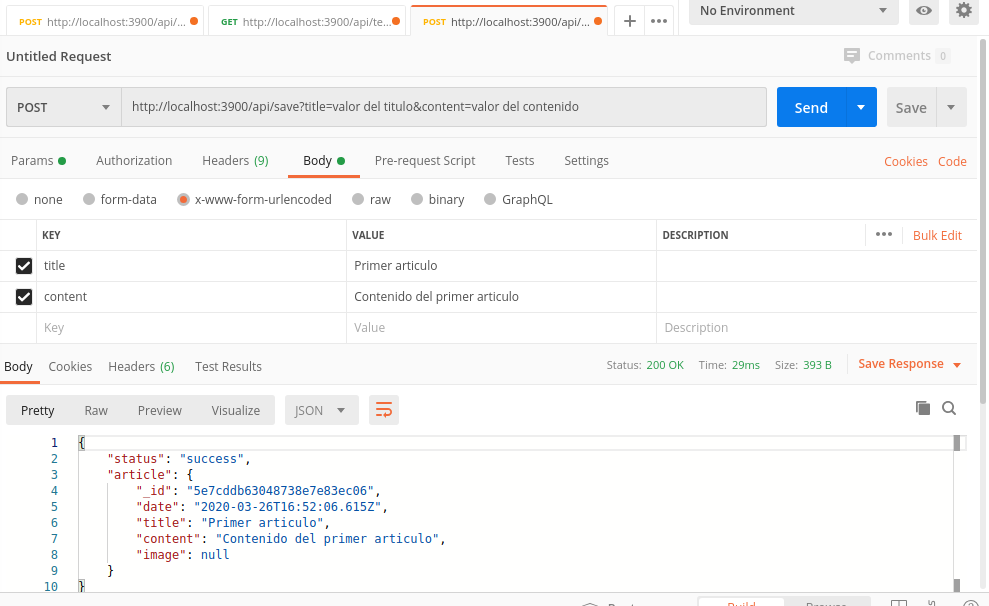






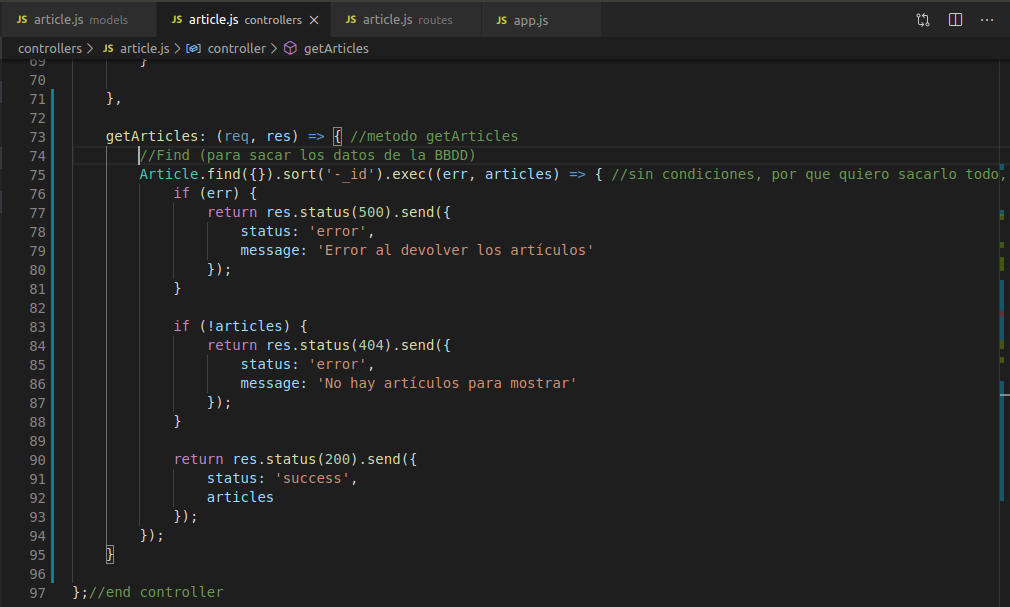
**Jueves 26-3-20**



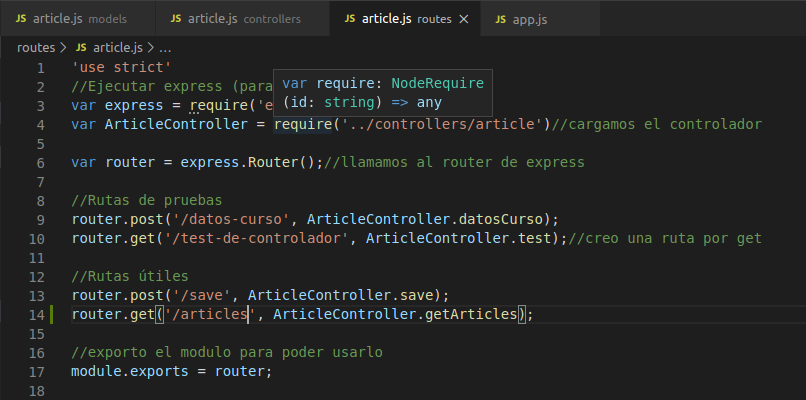


**Metodo que devuelve todos los artículos de la BBDD**

Controller

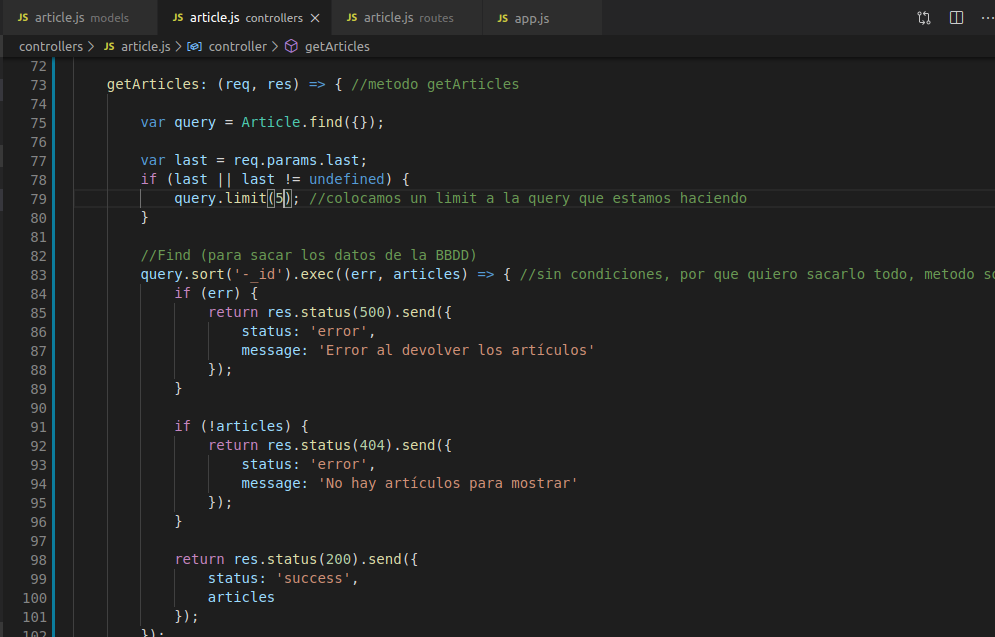


Routes



**Sacar últimos artículos**

Controller

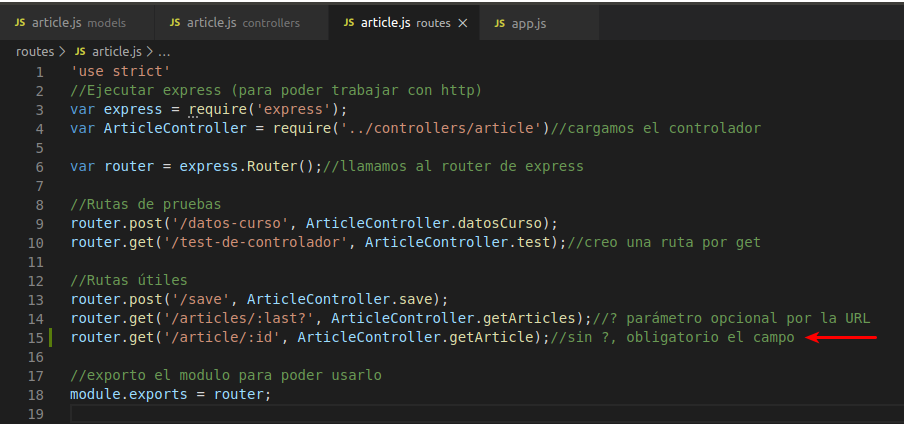


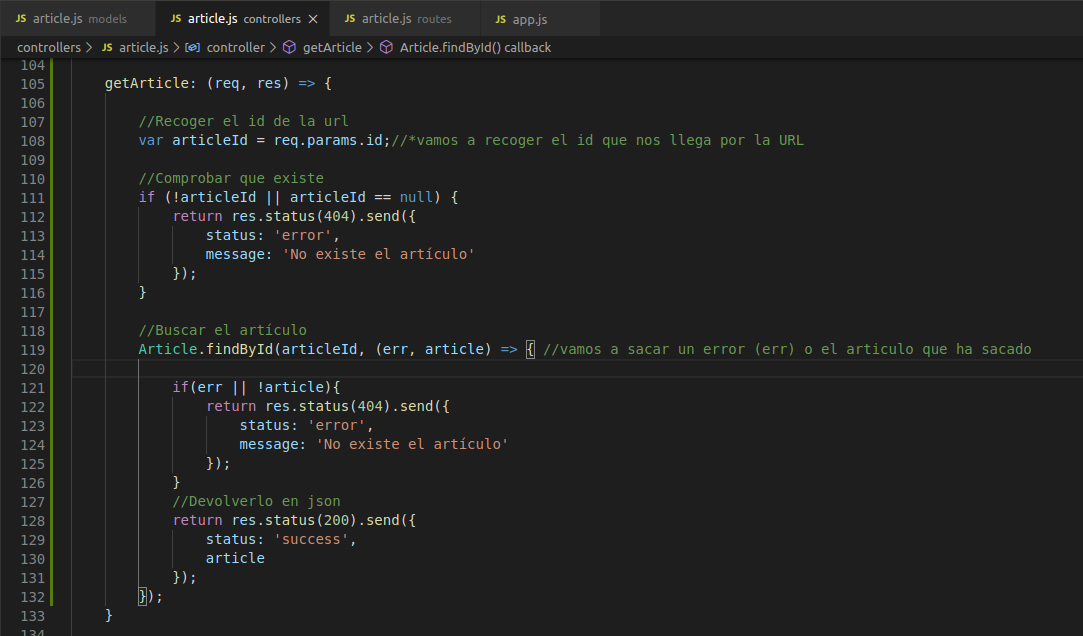
Routes

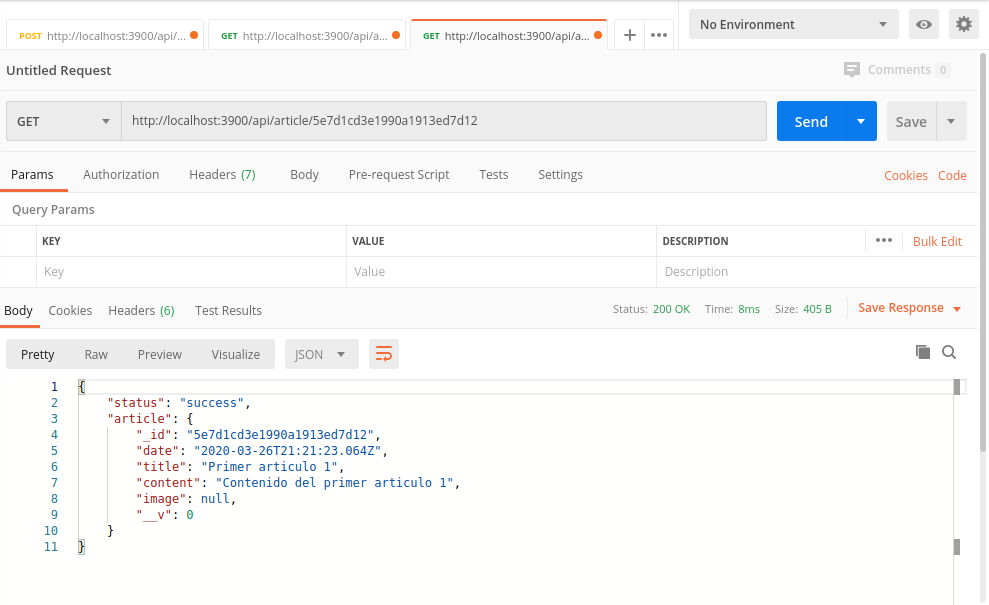


**Viernes 27-3-20**

**Devolver un solo artículo**



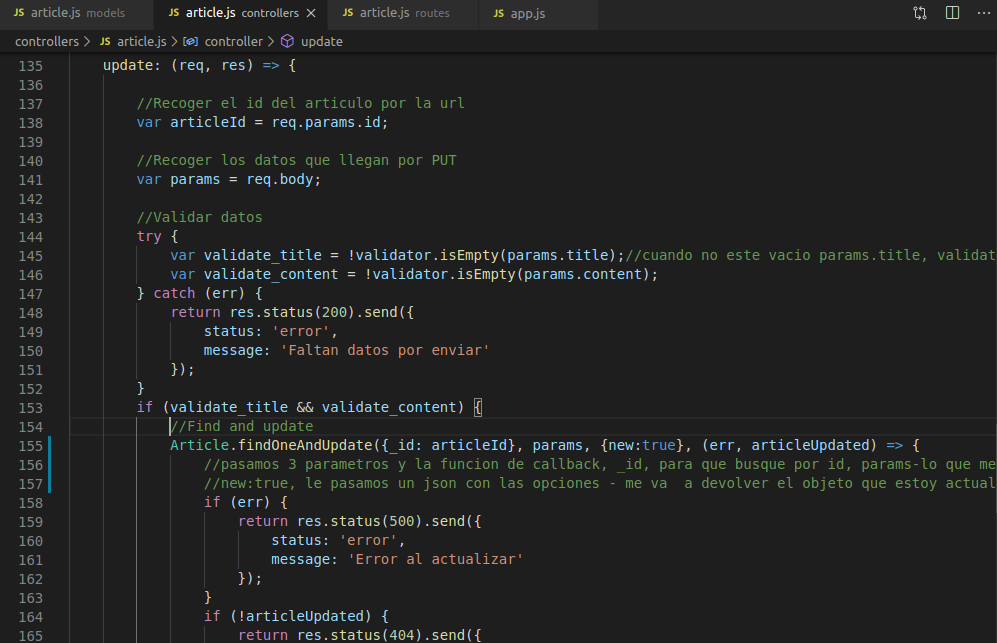


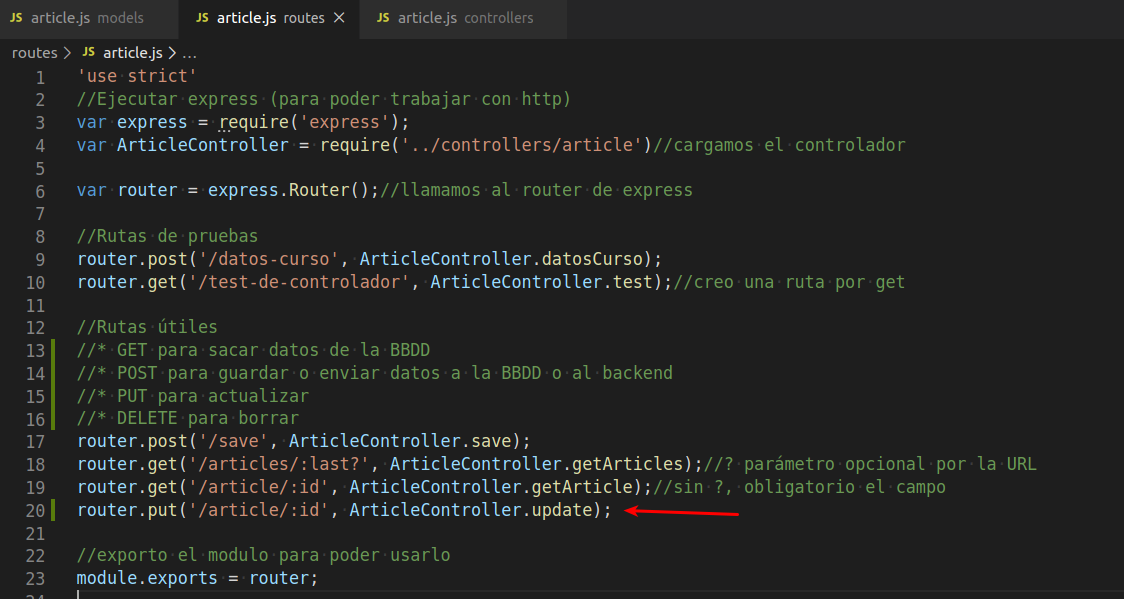


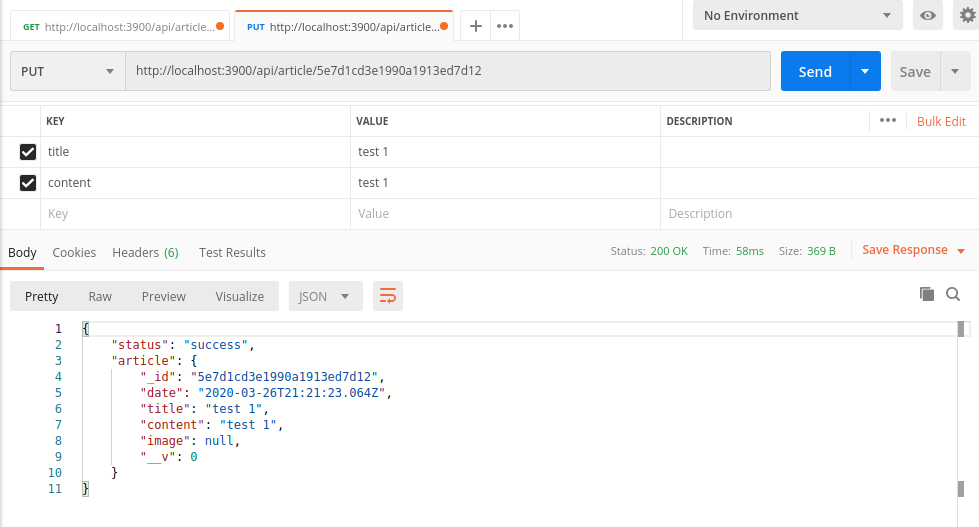
**Sabado 28-3-20**

**Método para actualizar articulos**

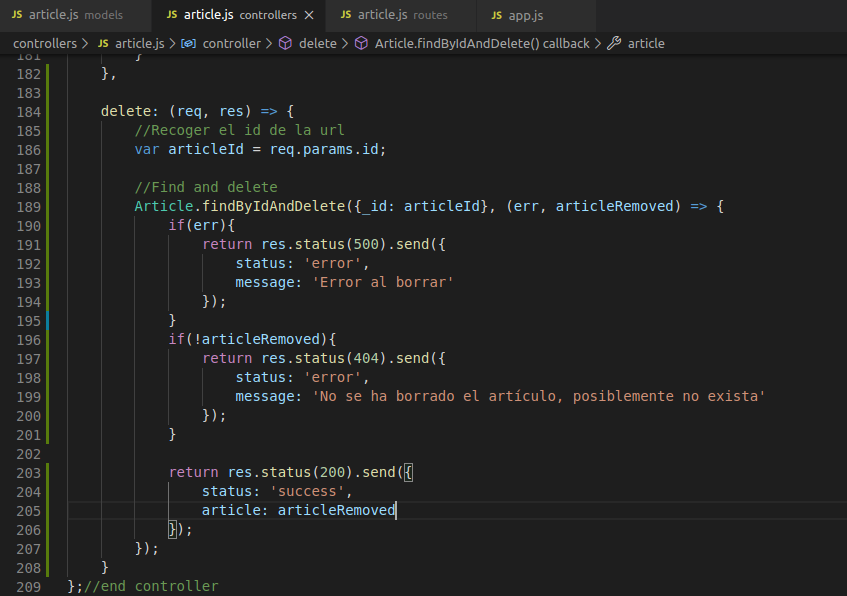
(actualizar o modificar los datos de un articulo)

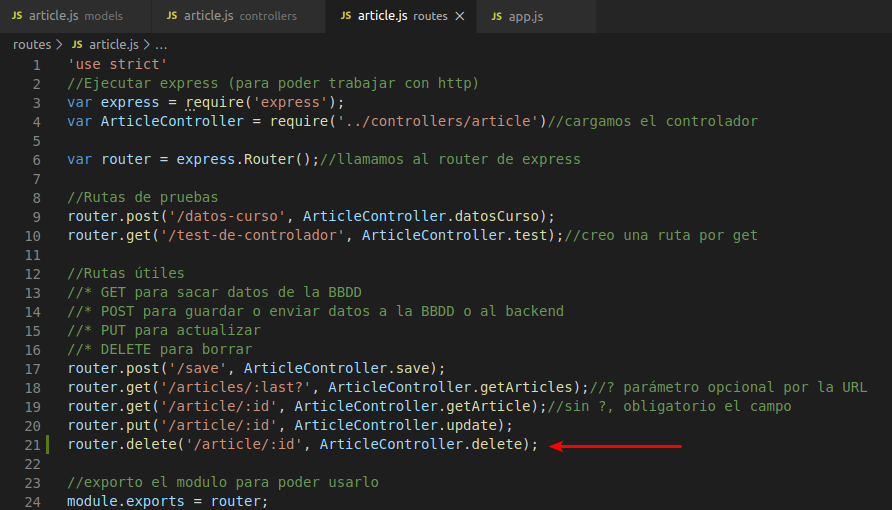


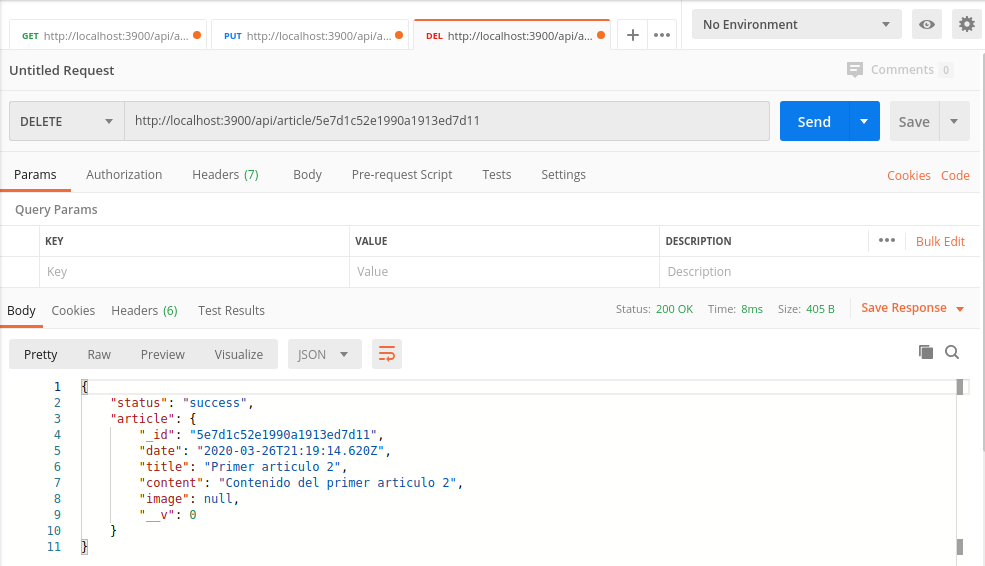




**Eliminar articulos**

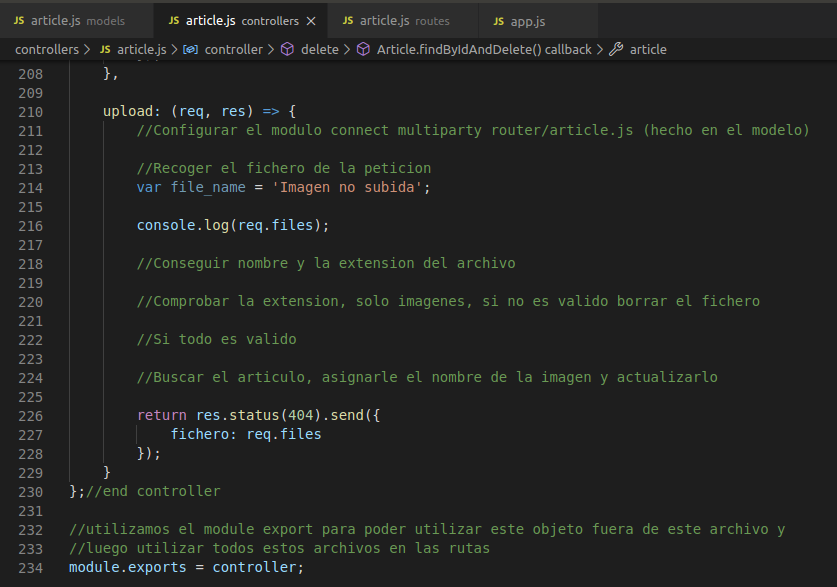


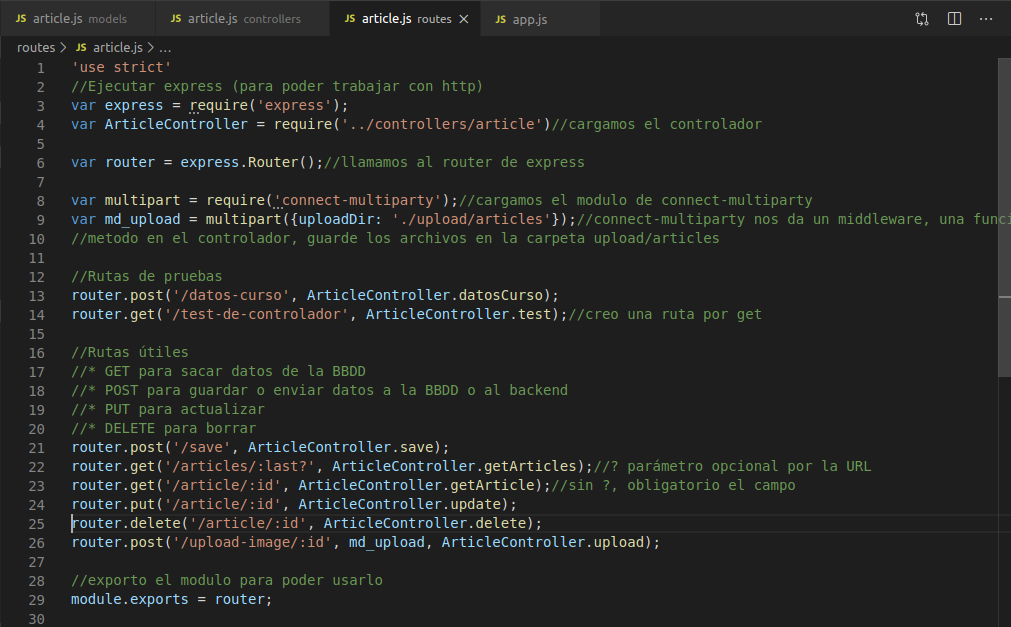


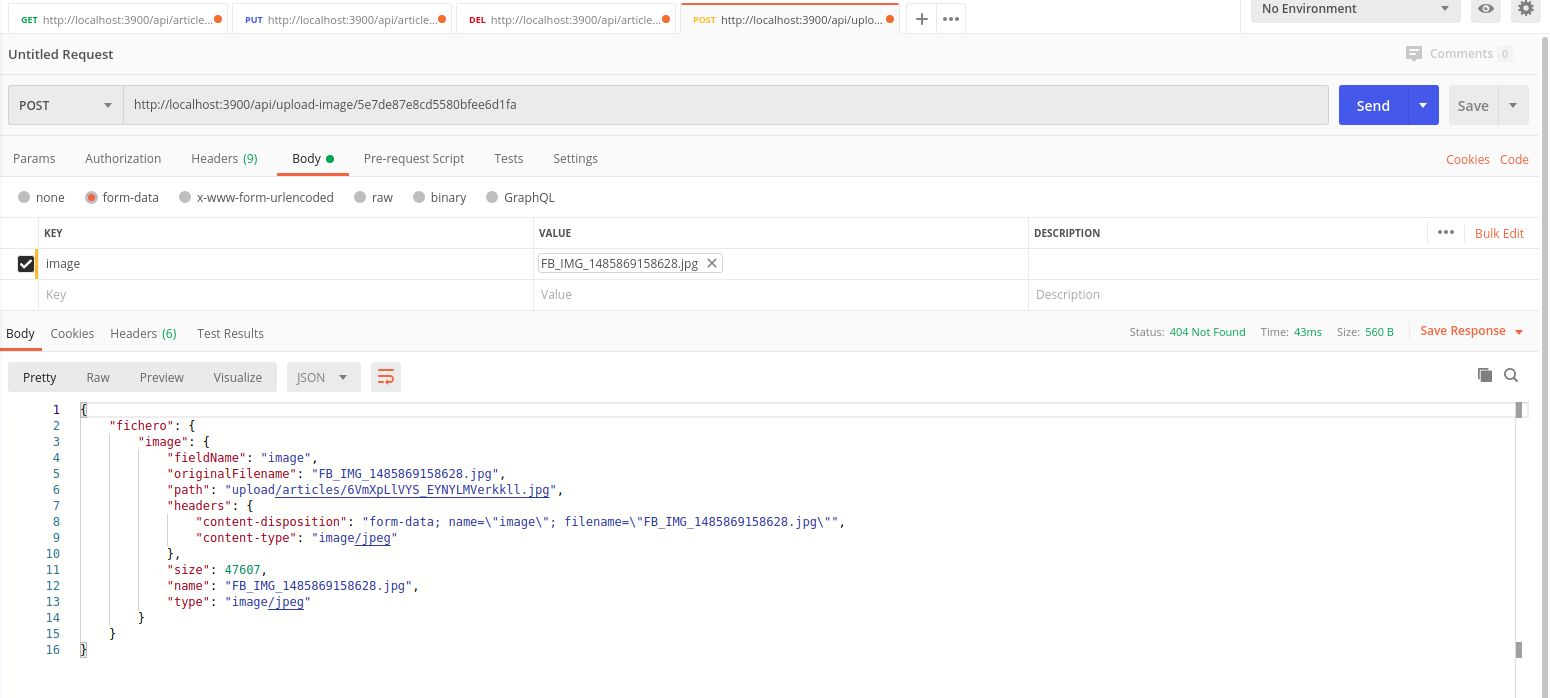


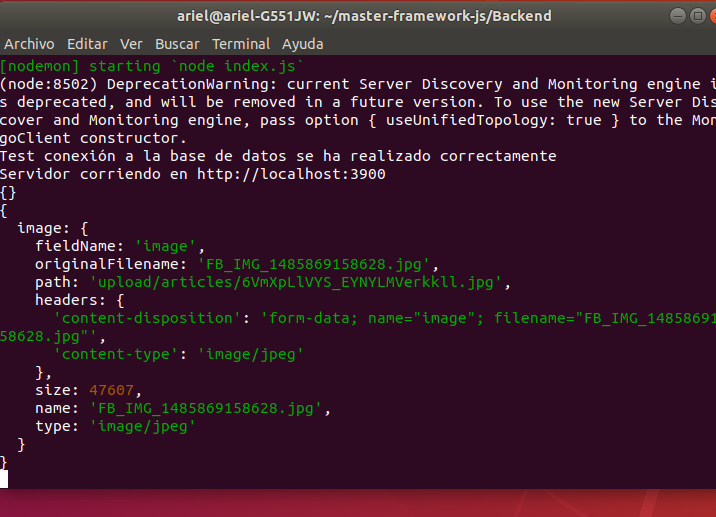
**Domingo 29-3-20**

**Subida de archivos**



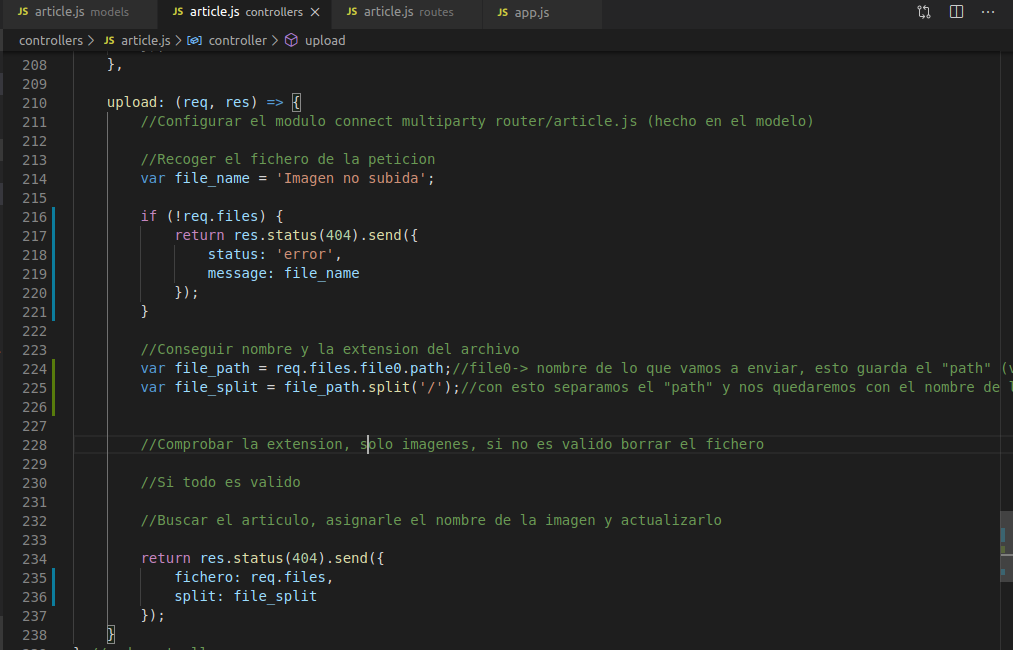


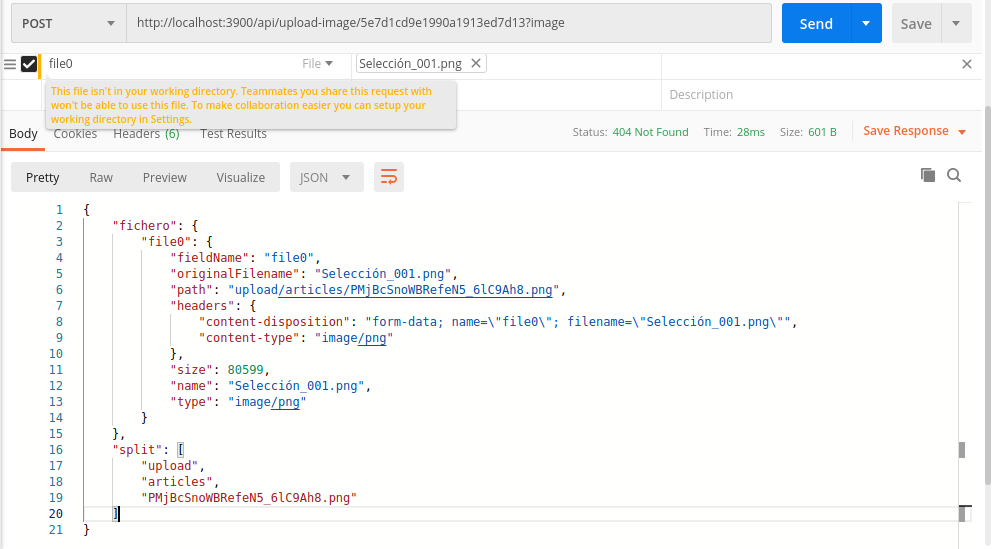




**Miércoles 01/04/2020**

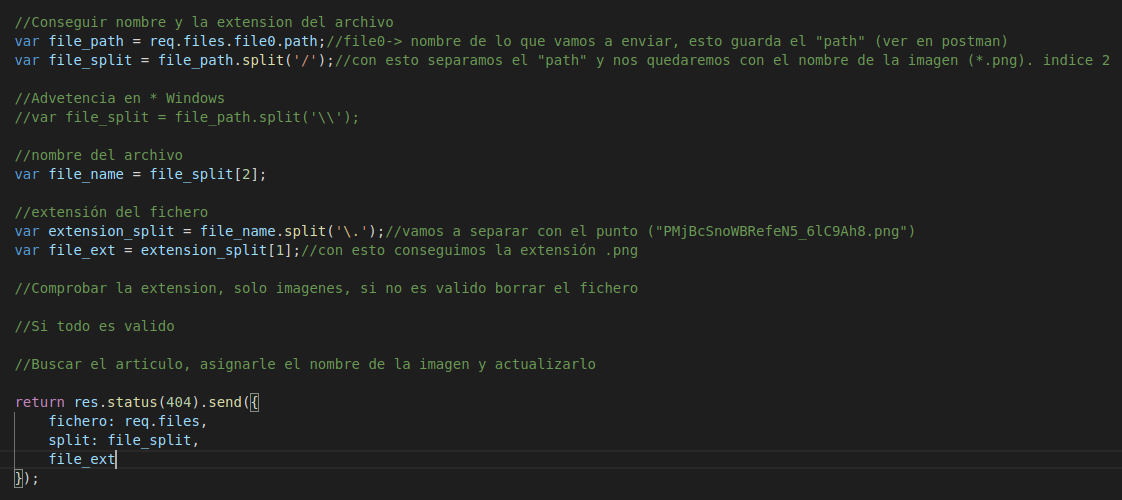
separando el «path»





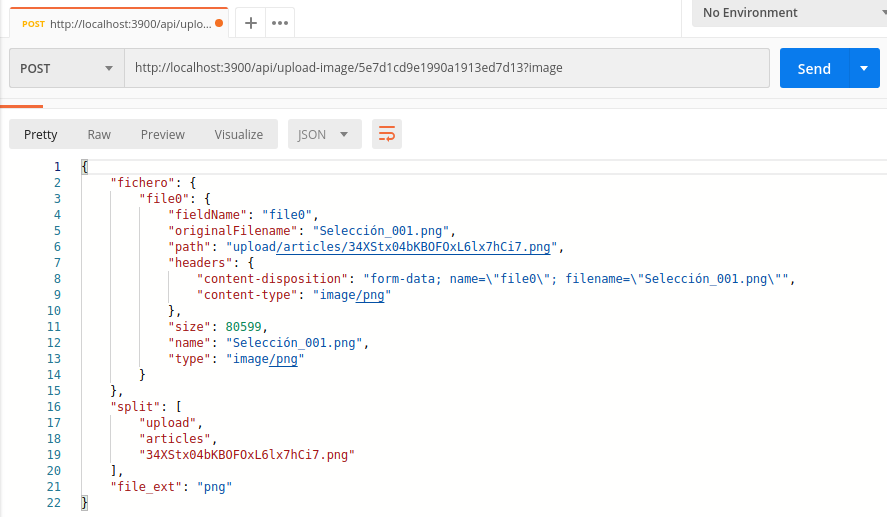
nos entrega «split», me separa el string «path» en un array de 3 elementos

Ahora conseguimos la extensión:



El nombre (file\_name) lo usaremos para guardarlo en la BBDD

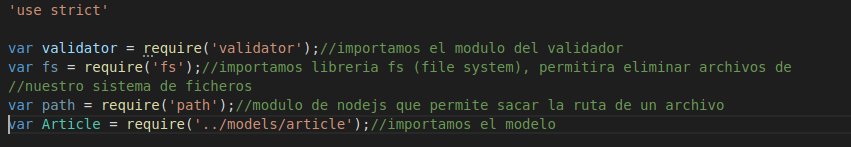
y la extensión (file\_ext) para comprobar si es un fichero valido



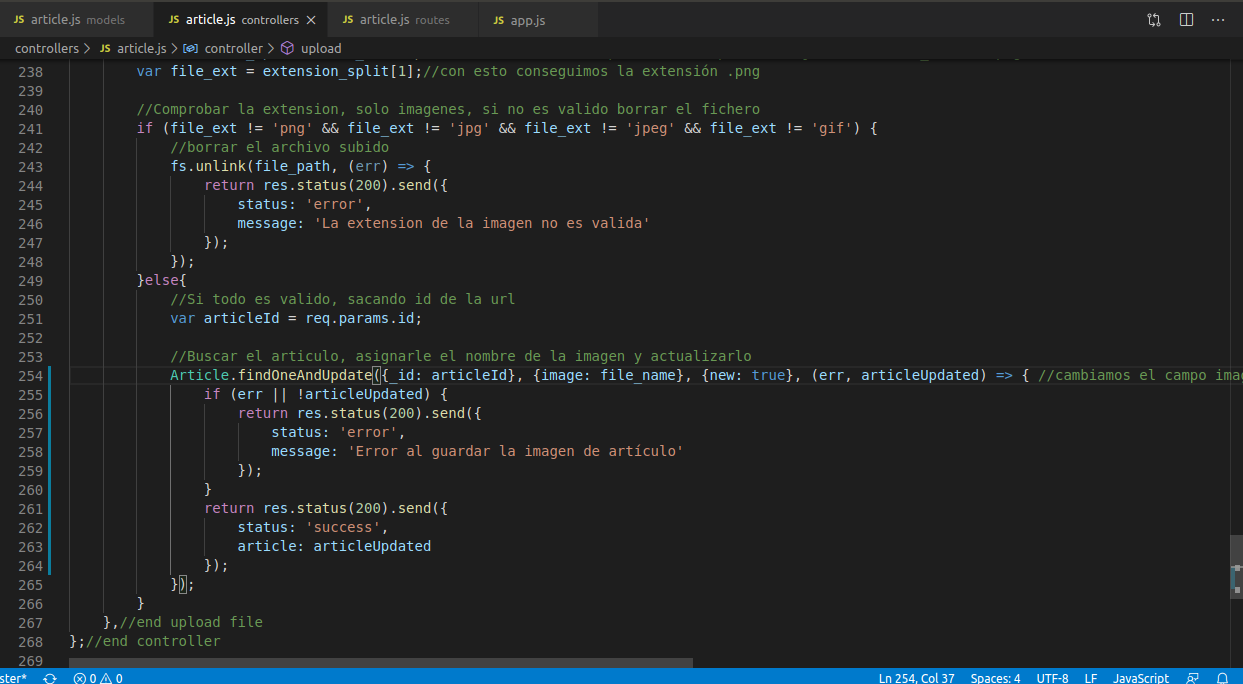
**Domingo 05/04/2020**

**Subir archivos con Node. Segunda parte**

Importamos las librerias fs y path:

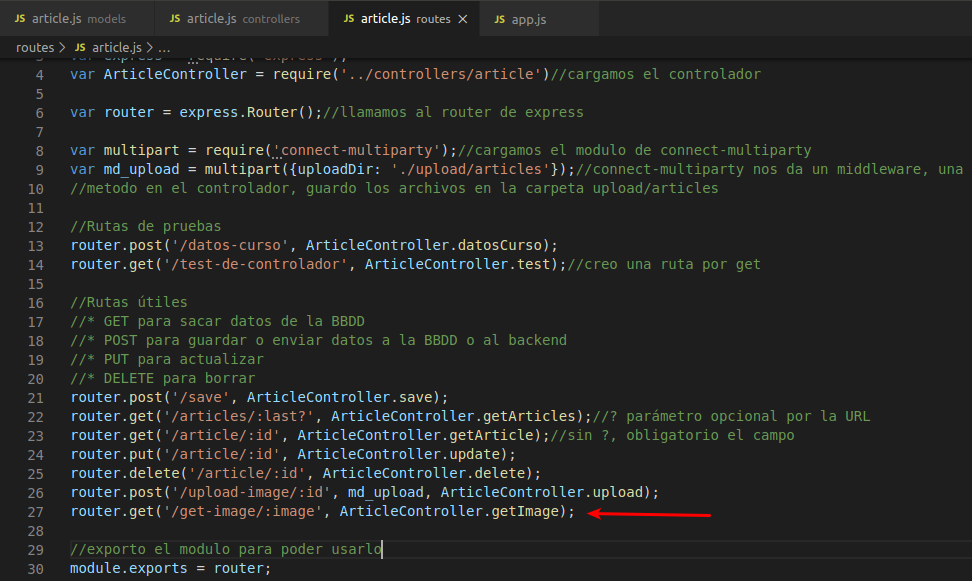


**Martes 07/04/2020**



**Miercoles 8/4/20**

Ruta de obtener imagen:



Controlador de obtener imagen:

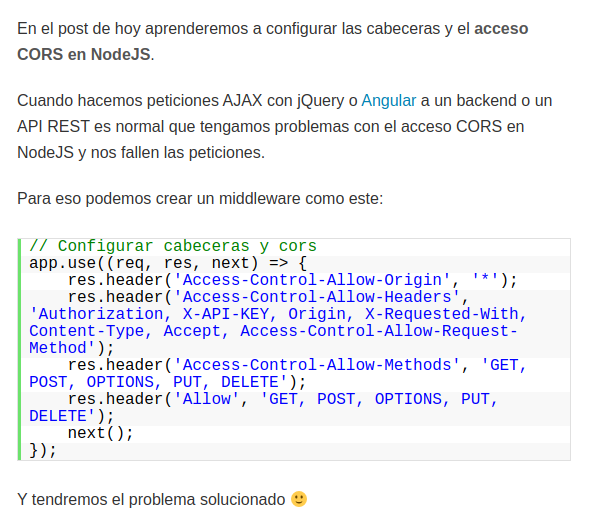


Ruta de :

**Sabado 18-4-20**

**CORS:**

<https://victorroblesweb.es/2018/01/31/configurar-acceso-cors-en-nodejs/>

[](https://victorroblesweb.es/2018/01/31/configurar-acceso-cors-en-nodejs/)