PROYECTOS

[1- PROYECTO 1: App películas 2](#_Toc127180522)

[HTML Y CSS 2](#_Toc127180523)

[COMPONENTES 2](#_Toc127180524)

[COMPONENTE Crear.js 3](#_Toc127180525)

[HELPERS 3](#_Toc127180526)

[COMPONENTE Listado.js 4](#_Toc127180527)

[Modificar COMPONENTE Crear.js 4](#_Toc127180528)

[BORRAR películas: modifico componente Listado.js 4](#_Toc127180529)

[EDITAR películas: modifico componente Listado.js 5](#_Toc127180530)

[EDITAR películas: componente Editar.js 5](#_Toc127180531)

[COMPONENTE Buscador.js 5](#_Toc127180532)

[2- PROYECTO 2: Portafolio 7](#_Toc127180533)

[3- PROYECTO 3: Blog 8](#_Toc127180534)

[1- Instalaciones previas 8](#_Toc127180535)

[2- Crear proyecto en node 8](#_Toc127180536)

[3- Dependencias 8](#_Toc127180537)

[4- Configuración previa 8](#_Toc127180538)

[5- Arrancar el proyecto 8](#_Toc127180539)

[6- Conectar BBDD con el proyecto Node 9](#_Toc127180540)

[7- Crear servidor Node 9](#_Toc127180541)

[8- Visualizar el proyecto 10](#_Toc127180542)

[9- Crear rutas 10](#_Toc127180543)

[10- Probando la api con Postman 11](#_Toc127180544)

[11- MVC 11](#_Toc127180545)

[12- MODELO 12](#_Toc127180546)

[13- CONTROLADOR 12](#_Toc127180547)

[14- Añadir otra ruta 13](#_Toc127180548)

[15- CREAR ARTÍCULOS 14](#_Toc127180549)

[1- Crear la ruta al endpoint 14](#_Toc127180550)

[2- Recibir datos con form-urlencoded 15](#_Toc127180551)

[3- Validar datos 16](#_Toc127180552)

[4- Crear el objeto a guardar 17](#_Toc127180553)

[5- Asignar valores a objeto basado en el modelo 17](#_Toc127180554)

[6- Guardar en BBDD y devolver el resultado 17](#_Toc127180555)

[7- Resultado completo de controllers/article función crear 18](#_Toc127180556)

[16- LISTAR ARTÍCULOS 19](#_Toc127180557)

[17- Filtros en la consulta 21](#_Toc127180558)

[18- Pasar parámetro a la petición en la BBDD 21](#_Toc127180559)

[19- MOSTRAR UN ARTÍCULO 22](#_Toc127180560)

[20- BORRAR ARTÍCULO 23](#_Toc127180561)

[21- EDITAR ARTÍCULO 24](#_Toc127180562)

[22- HELPER de Validación 25](#_Toc127180563)

[23- Subir ficheros e imágenes con NodeJS 26](#_Toc127180564)

[24- Actualizar y mostrar imágenes 28](#_Toc127180565)

[25- BUSCAR ARTÍCULO 29](#_Toc127180566)

[26- FRONT 30](#_Toc127180567)

[27- Probando la api con Postman 30](#_Toc127180568)

[28- Probando la api con Postman 30](#_Toc127180569)

[29- Probando la api con Postman 30](#_Toc127180570)

[30- Probando la api con Postman 30](#_Toc127180571)

[31- Probando la api con Postman 31](#_Toc127180572)

[32- Probando la api con Postman 31](#_Toc127180573)

[33- Probando la api con Postman 31](#_Toc127180574)

[34- Probando la api con Postman 31](#_Toc127180575)

[35- Probando la api con Postman 31](#_Toc127180576)

[36- Probando la api con Postman 31](#_Toc127180577)

[37- Probando la api con Postman 31](#_Toc127180578)

[38- Probando la api con Postman 32](#_Toc127180579)

[39- Probando la api con Postman 32](#_Toc127180580)

[4- PROYECTO 4: Red social 33](#_Toc127180581)

[FUNCIONES 34](#_Toc127180582)

# PROYECTO 1: App películas

## HTML Y CSS

**Borro** el fichero **app.css**.  
Pongo los **estilos** CSS en el fichero **index.css**.

Pego el contenido del body del html dentro del return del **app.js**.  
Modifico los elementos class por className.

## COMPONENTES

Creo el componente **Listado.js** y le pego todo el contenido del ‘content’. En el content, llamo al componente Listado con su etiqueta.

Creo el componente **Buscador.js** y le pego el contenido del aside, la parte del buscador.

Creo el componente **Crear.js** y le pego el contenido del aside, la parte de crear película.

## COMPONENTE Crear.js

Creo el estado **peli** con las propiedades **id, title y description** vacíos ‘’.

Hago un **e.preventDefault()** para que no envíe el form y recargue la página.

Por otro lado, **asigno** los **datos** del form a una variable y llamo al **setPeli**.

Por último, **muestro** en un div los **datos** de la peli creada **sólo si ésta tiene datos**.

## HELPERS

Creo en src la carpeta helpers. Aquí voy a guardar funcionalidades que no sean exclusivas de un componente, como por ejemplo, guardar en localStorage.

/\*  Guardar en el localStorage. Recibe la lista y el elem a guardar. \*/

export const GuardarStorage = (lista, elem) => {

    /\* Conseguir los datos de localStorage \*/

    let elementos = JSON.parse(localStorage.getItem(lista));  
 /\* JSON.parse Convierte JSON a objeto \*/

    /\* Si elementos no es un array, lo creo. \*/

    if(!Array.isArray(elementos)){

        elementos = [];

    }

    /\* Añado al array el elemento nuevo \*/

    elementos.push(elem);

    /\* Guardo en localStorage el array modificado \*/

    localStorage.setItem(lista, JSON.stringify(elementos));

    /\* OPCIONAL: Devolver objeto \*/

    return elem;

}

## COMPONENTE Listado.js

Creo un **estado** en App.js para la **lista** de películas. Lo creo ahí porque necesito que se comunique entre los distintos componentes (Crear, Listado,…).  
A la hora de llamar a los componentes, le paso **2 props**: el **estado de lista** de películas y el **setLista** para que se pueda llamar desde dentro del componente.

Creo un **useEffect** para que **cuando cargue** el **componente**, se carguen las películas y me **actualice** el **estado** de **lista** de películas.

## Modificar COMPONENTE Crear.js

Como he puesto Listado como estado global (en App.js), ahora al crear una peli, además de actualizar el estado peli y guardar en localStorage, necesito actualizar el estado listado.

setPeli(peliNueva);

setLista( (estado\_previo) => {

    return [...estado\_previo, peliNueva];

});

GuardarStorage('pelis', peliNueva);

## BORRAR películas: modifico componente Listado.js

<button className="delete" onClick={ () => borrarPeli(peli.id) }>

Borrar </button>

const borrarPeli = (id) => {

    /\* Obtengo el listado de películas \*/

    let pelis = getLista();

    /\* Busco la que quiero eliminar. Creo una copia del array sin ella \*/

    let pelis\_res = pelis.filter( (peli) => peli.id !== parseInt(id) );

    /\* La elimino del estado listado \*/

    setLista(pelis\_res);

    /\* Actualizo localStorage \*/

    localStorage.setItem('pelis', JSON.stringify(pelis\_res));  
}

## EDITAR películas: modifico componente Listado.js

Creo un estado editar que inicialmente valga 0.

const [editar, setEditar] = useState(0);

<button className="edit" onClick={ () => setEditar(peli.id) }>

Editar </button>

Justo debajo de los botones, pongo un div condicional, que si el estado editar es igual al id de la película, me aparezca un componente que voy a crear, llamado Editar.js, que va a tener el formulario para editar la película.  
Le paso como props: la **peli** que estoy editando, **setEditar**, **setLista** y **getLista**.

## EDITAR películas: componente Editar.js

Tiene un formulario, el cual al darle a submit nos lleva a una función:

const editarPeli = (e, id) => {

    e.preventDefault();

    /\* Consigo la lista de películas \*/

    const pelis = getLista();

    /\* Conseguir los datos del formulario para actualizar la peli \*/

    let peliNueva = {

        id,

        title: e.target.title.value,

        description: e.target.description.value

    }

    /\* Busco la peli a modificar \*/

    const indice = pelis.findIndex(peli => peli.id === id);

    /\* Modifico la película en el listado de pelis. \*/

    pelis[indice] = peliNueva;

    /\* Actualizo estado lista \*/

    setLista(pelis);

    /\* Actualizo estado editar \*/

    setEditar(0);

    /\* Guardo en localStorage \*/

    localStorage.setItem('pelis', JSON.stringify(pelis));

}

## COMPONENTE Buscador.js

Recibe las props **lista** y **setLista**.  
Creo el **estado búsqueda**, **estado encontrado**, y un **useEffect** que llama a la función buscarPeli.

const buscarPeli = () => {

    /\* Filtro para encontrar pelis \*/  
    let resultado = lista.filter( (peli) => {  
        return peli.title.toLowerCase().includes(busqueda.toLowerCase())  
    })  
  
    /\* Si no encuentra o no ha escrito nada, muestra localStorage. \*/  
    if(resultado <= 0 || busqueda.length <=0){  
        resultado = JSON.parse(localStorage.getItem('pelis'));  
        setEncontrado(false);  
    }  
    else{  
      setEncontrado(true);  
    }  
      
    setLista(resultado);  
}

onChange={(e) => setBusqueda(e.target.value)}

# PROYECTO 2: Portafolio

Creo la carpeta components.  
Creo componentes: Contacto, Curriculum, Error, Inicio, Portafolio, Servicios.

Creo la carpeta router. Dentro creo el componente Rutas.

Creo la carpeta components/layout. Dentro creo los componentes: Header, Nav y Footer.

Creo la carpeta data. Dentro creo trabajos.js que contiene un array de objetos con los proyectos que he realizado.

Creo la carpeta public/images donde voy a meter una imagen para cada proyecto.

Creo el componente Proyecto dentro de components.

Creo el componente ListadoTrabajos dentro de components.

# PROYECTO 3: Blog

## Instalaciones previas

* **NodeJS**: Descargar e instalar NodeJS LTS.
* **MondoDB (+ Servicio y Compass)**: Descargo de su web el Community Server.
* **Postman**: Descargar e instalar de su web.

## Crear proyecto en node

* Creo una carpeta en mi equipo.
* Abro la carpeta en VSCode.
* En la consola ejecuto **npm init**

## Dependencias

* **npm i express --save** Framework para manejar peticiones HTTP (GET, POST , PUT).
* **npm i mongoose --save**  Librería de MongoDB que nos aporta muchas funcionalidades.
* **npm i multer --save** Para subir archivos.
* **npm i validator --save** Para validar datos.
* **npm i cors --save** Para hacer peticiones entre dominios cruzados.
* **npm i nodemon --save-dev** Watcher de cambios, para no parar y arrancar el server.

--save guardo la dependencia dentro del proyecto.  
--save-dev guardo la dependencia dentro del proyecto como devDependencia.

## Configuración previa

En el package.json en la parte de scripts voy a crear un nuevo comando para ejecutar en nodemon el archivo principal de la aplicación.



## Arrancar el proyecto

* Para arrancar en nodemon: **npm start**
* Para arrancar sin nodemon: **node index.js**

## Conectar BBDD con el proyecto Node

* En la raíz del proyecto, creo la carpeta database
* Dentro creo connection.js

const mongoose = require('mongoose')

mongoose.set('strictQuery',true) //Si da fallo poner esto

const connection = async (url, bbdd) => {

    try{

        await mongoose.connect(`mongodb://${url}/${bbdd}`)

        console.log('Conectado a BBDD')

    }

    catch(error){

        console.log(error)

        throw new Error('Error al conectar a la BBDD')

    }

}

module.exports = { connection }

* Importo la conexión a la BBDD en el index.js

const { connection } = require('./database/connection')

//Inicializar app

console.log('App de node inicializada.')

//Conectar a BBDD

const url = '127.0.0.1:27017'

const bbdd = 'blog'

connection(url, bbdd)

## Crear servidor Node

A falta de crear las rutas, me tiene que quedar así el index.js

const { connection } = require('./database/connection')

const express = require('express')

const cors = require('cors')

//Inicializar app

console.log('App de node inicializada.')

//Conectar a BBDD

const url = '127.0.0.1:27017'

const bbdd = 'blog'

connection(url, bbdd)

//Crear servidor Node

const app = express()

//Configurar CORS

app.use(cors())

//Convertir body a JSON

app.use(express.json())

//Crear rutas

// Escuchar peticiones HTTP

const puerto = 3900

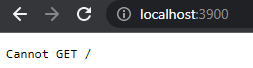
app.listen(puerto, () => {

    console.log(`Servidor corriendo en el puerto ${puerto}.`)

})

## Visualizar el proyecto

Voy a la web y escribo localhost:3900



Veo esto porque no tengo rutas asociadas aún

## Crear rutas

Voy a crear 2 rutas: la ruta ‘/’ y la ruta ‘/probando’.

//Crear rutas

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

    return res.status(200).send( '<h1>Bienvenido a la API REST</h1>' )

})

app.get('/probando', (req,res) => {

    console.log('Se ha ejecutado el endpoint PROBANDO.')

    return res.status(200).json(

        [

            { nombre: 'Kevin', apellidos: 'Arcas Monteiro', edad: 29 },

            { nombre: 'Alejandra', apellidos: 'Ramos Fdez', edad: 27 }

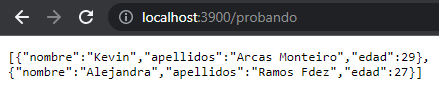
        ]

    )

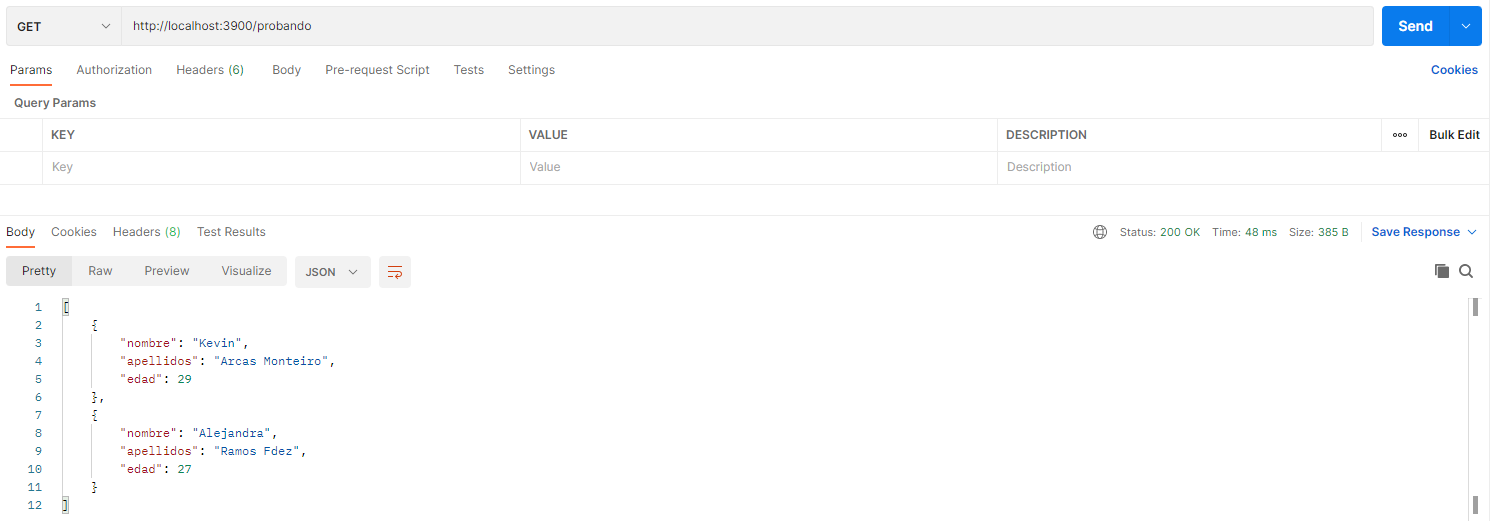
})

La ruta ‘/’ manda html, para ello pongo **send**.

  
  
La ruta ‘/probando’ manda json, por eso en lugar de send utilizo **json**.



## Probando la api con Postman



Muestra los datos que devuelve la api.

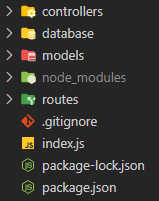
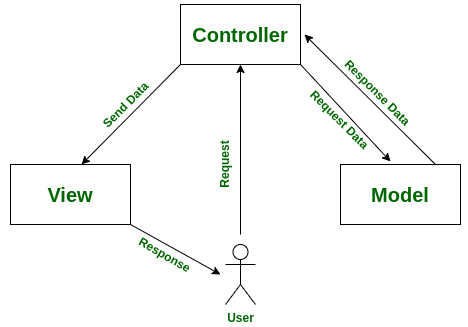
## MVC

La arquitectura del proyecto es MVC.

**Modelo**: Crea nuevos artículos y enlace a BBDD.

**Vista**: En una API no hay una vista como tal, es un JSON que devuelve la api.

**Controlador**: Cada una de las acciones y la lógica de la aplicación. Recibe peticiones HTTP, procesa la información, llama al modelo y ese se encargará de la BBDD. Cada acción será una ruta/endpoint de la api.  
También tendremos rutas, middlewares y helpers.

## MODELO

Dentro de **models** creo **Article.js** y ahí dentro creo el modelo para crear, borrar, editar artículos.

const { Schema, model } = require('mongoose')

//Defino el formato de cada Article.

const ArticleSchema = Schema({

    title: String,

    content: {

        type: String,

        required: true

    },

    date: {

        type: Date,

        default: Date.now

    },

    image: {

        type: String,

        default: 'default.png'

    }

})

module.exports = model('Article', ArticleSchema, 'articles')

// Article es el nombre que va a tener mi modelo.

// ArticleSchema es el esquema que va a utilizar.

// articles (parámetro opcional) indica el nombre de la collection en MongoDB.

## CONTROLADOR

Dentro de **controllers** creo **article.js** y ahí dentro creo todas las acciones que va a tener el controlador Article.

const prueba = (req,res) => {

    return res.status(200).json({

        mensaje: 'Acción prueba en el controlador de artículos.'

    })

}

module.exports = {

    prueba

}

Voy a crear una ruta para acceder a prueba. En routes, creo el archivo **article.js**

const express = require('express')

const router = express.Router()

//Cargo mi controlador

const ArticleController = require('../controllers/article')

//Ruta de prueba

router.get('/ruta-prueba', ArticleController.prueba)

module.exports = router

En el **index.js** voy a poner mi nueva ruta.

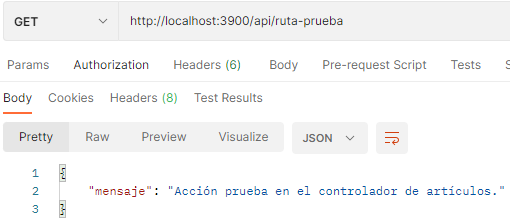
//RUTAS

const routes\_article = require('./routes/article')

//Cargo las rutas

app.use('/api', routes\_article)

Pruebo con Postman



## Añadir otra ruta

1. Voy a controllers/article.js y añado

const personas = (req,res) => {

    return res.status(200).json(

        [

            { nombre: 'Kevin', apellidos: 'Arcas Monteiro', edad: 29 },

            { nombre: 'Alejandra', apellidos: 'Ramos Fdez', edad: 27 }

        ]

    )

}

Y en el module.exports añado personas para que se exporte.

module.exports = {

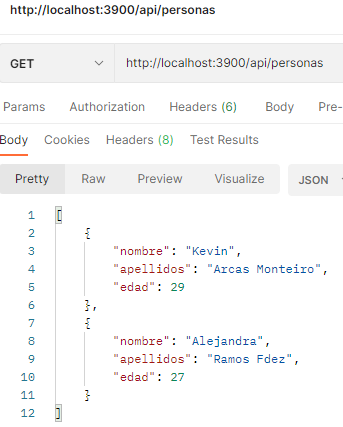
    prueba,

    personas

}

1. Voy a routes/article.js y añado

router.get('/personas', ArticleController.personas)

1. Pruebo la api con Postman  
     
   

## CREAR ARTÍCULOS

### Crear la ruta al endpoint

En controllers/article.js

const crear = (req,res) => {

    return res.status(200).json({

        mensaje: 'Acción de crear'

    })

}

module.exports = {

    prueba,

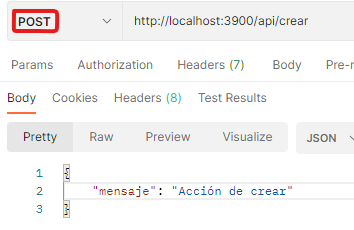
    personas,

    crear

}

En routes/articles pongo la ruta pero con **POST** en lugar de GET, porque va a recibir datos.

router.post('/crear', ArticleController.crear)

Lo pruebo en Postman  
  


### Recibir datos con form-urlencoded

En controllers/article

const crear = (req,res) => {

    // Recoger parámetros por POST a guardar

    let parametros = req.body

    // Devolver el resultado

    return res.status(200).json({

        mensaje: 'Acción de crear',

        parametros

    })

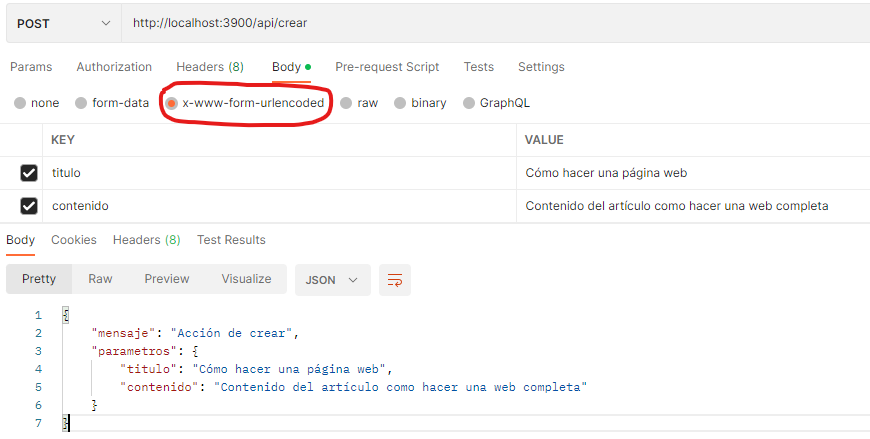
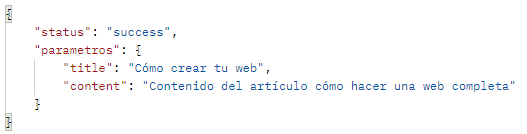
}

index.js

//Convertir body a JSON

app.use(express.json()) // recibir datos con content-type app/json

app.use(express.urlencoded({extended: true})) // datos con form-urlencoded

Lo pongo así, porque en este caso no hay que convertir a JSON, va en formato form-urlencoded.  
  
  
Lo pruebo en Postman  
  


### Validar datos

En controllers/article

Importo la dependencia validator

const validator = require('validator')

Voy a validar los campos que se envían en la petición POST.

// Validar datos

try{

    let titleOk = !validator.isEmpty(parametros.title)

    let contentOk = !validator.isEmpty(parametros.content)

    if(!titleOk || !contentOk){

        throw new Error('No se ha validado la información')

    }

}catch(error){

    return res.status(400).json({

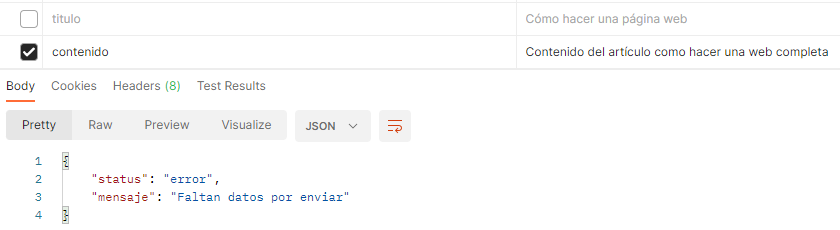
        status: 'error',

        mensaje: 'Faltan datos por enviar'

    })

}

Si mando una petición POST sin el título, me manda este mensaje

  
  
La validación del título va a consistir en:  
 - Comprobar que no esté vacío  
 - Longitud mínima 5 caracteres, máxima no tiene.

let titleOk = !validator.isEmpty(parametros.title)

    && validator.isLength(parametros.title, {min: 5, max: undefined})

### Crear el objeto a guardar

Importo el modelo del article.

const Article = require('../models/Article')

Creo el objeto a guardar

const articulo = new Article()

### Asignar valores a objeto basado en el modelo

Si lo hiciera manualmente, tendría que poner: articulo.title = parametros.title

Una línea por cada uno de los parámetros que quisiera asignar.

Para hacerlo de manera automática, al crear el Article, le paso los parámetros dentro del new Article.

const articulo = new Article(parametros)

### Guardar en BBDD y devolver el resultado

// Guardar el artículo en la BBDD

articulo.save((error, articuloGuardado) => {

    if(error || !articuloGuardado){

        return res.status(400).json({

            status: 'error',

            mensaje: 'No se ha guardado el artículo',

            error

        })

    }

    // Devolver el resultado

    return res.status(200).json({

        status: 'success',

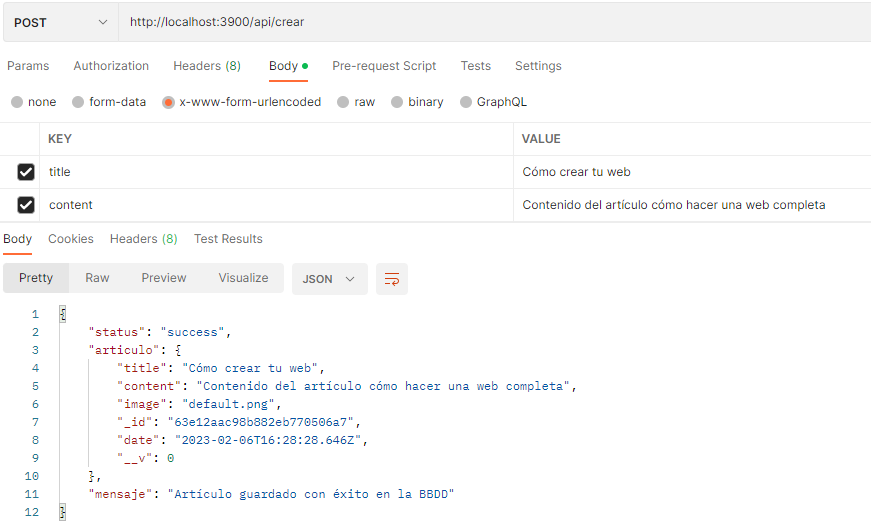
        articulo: articuloGuardado,

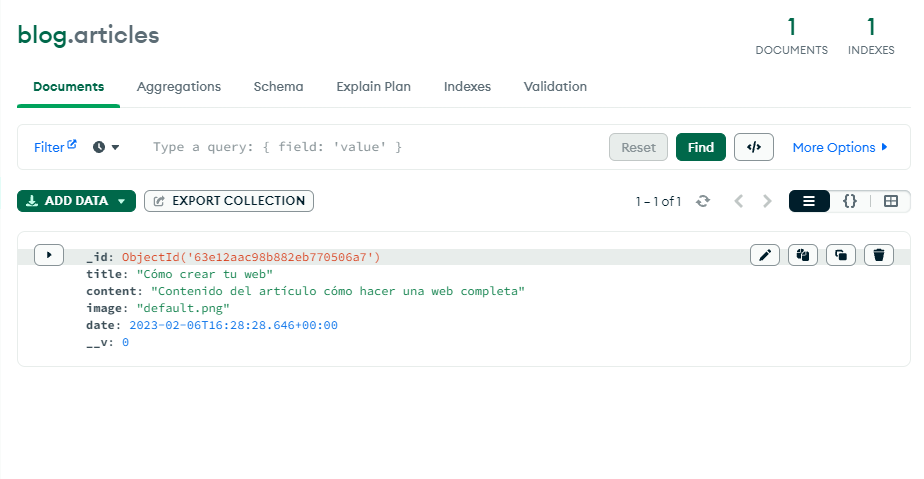
        mensaje: 'Artículo guardado con éxito en la BBDD'

    })

})

Prueba final de Postman.



MongoDB Compass  


### Resultado completo de controllers/article función crear

const crear = (req,res) => {

    // Recoger parámetros por POST a guardar

    let parametros = req.body

    // Validar datos

    try{

        let titleOk = !validator.isEmpty(parametros.title)

            && validator.isLength(parametros.title, {min: 5, max: undefined})

        let contentOk = !validator.isEmpty(parametros.content)

        if(!titleOk || !contentOk){

            throw new Error('No se ha validado la información')

        }

    }catch(error){

        return res.status(400).json({

            status: 'error',

            mensaje: 'Faltan datos por enviar'

        })

    }

    // Crear el objeto a guardar

    const articulo = new Article(parametros)

    // Asignar valores al objeto basado en el modelo (manual o auto)

        /\* articulo.titulo = parametros.titulo

        así sería de manera manual. Una línea por parámetro a asignar. \*/

    // Guardar el artículo en la BBDD

    articulo.save((error, articuloGuardado) => {

        if(error || !articuloGuardado){

            return res.status(400).json({

                status: 'error',

                mensaje: 'No se ha guardado el artículo',

                error

            })

        }

        // Devolver el resultado

        return res.status(200).json({

            status: 'success',

            articulo: articuloGuardado,

            mensaje: 'Artículo guardado con éxito en la BBDD'

        })

    })

}

## LISTAR ARTÍCULOS

Defino una ruta GET en routes/article.js

router.get('/articulos', ArticleController.listar)

En el controllers/article.js

const listar = (req,res) => {

    let consulta = Article.find({}).exec( (error, articulos) => {

        if(error || !articulos){

            return res.status(404).json({

                status: 'error',

                mensaje: 'No se han encontrado artículos',

                error

            })

        }

        return res.status(200).send({

            status: 'success',

            articulos

        })

    })

}

Y lo exporto

module.exports = {

    prueba,

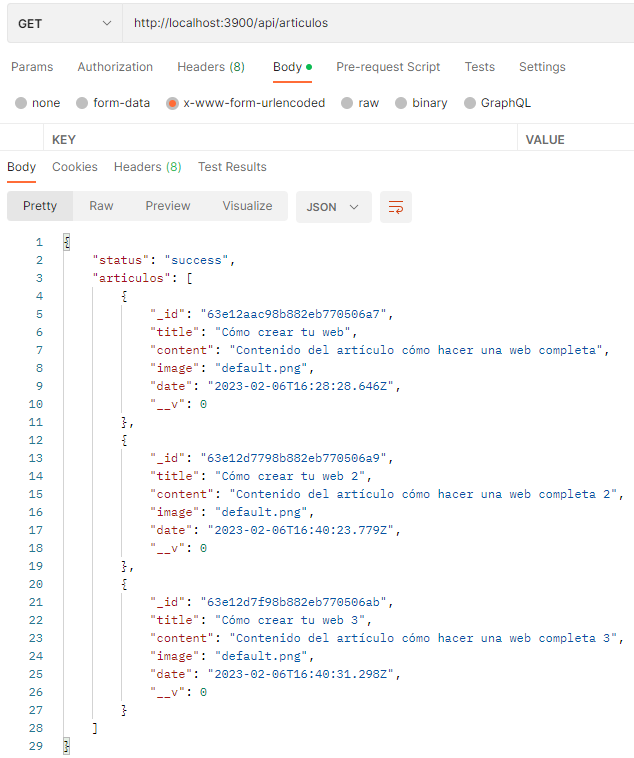
    personas,

    crear,

    listar

}

NOTA:  
En el Article.find({}) le puedo pasar parámetros para filtrar la búsqueda, pero como quiero que me muestre todo el listado no pongo ningún parámetro.



## Filtros en la consulta

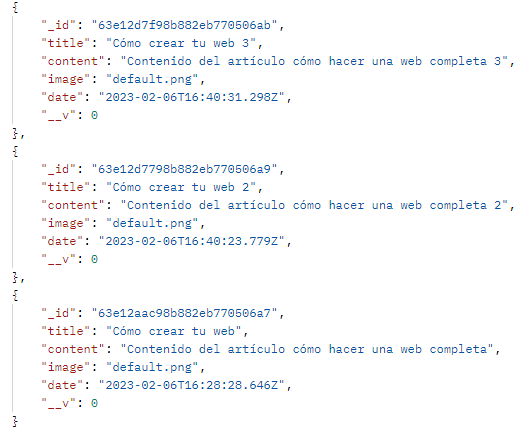
Para crear filtros, en el controller/article.js modifico el Article.find

Ordenar los artículos de más nuevos a más antiguos

let consulta = Article.find({})

                      .sort({date: -1})

                      .exec( (error, articulos) => {



Mostrar solamente 2 artículos.

let consulta = Article.find({})

                            .sort({date: -1})

                            .limit(2)

                            .exec( (error, articulos) => {

## Pasar parámetro a la petición en la BBDD

Voy a hacer que en la URL le pueda pasar como parámetro cuántos artículos quiero mostrar. Pero quiero que ese parámetro sea opcional.

En routes/article

router.get('/articulos/:cantidad?', ArticleController.listar)

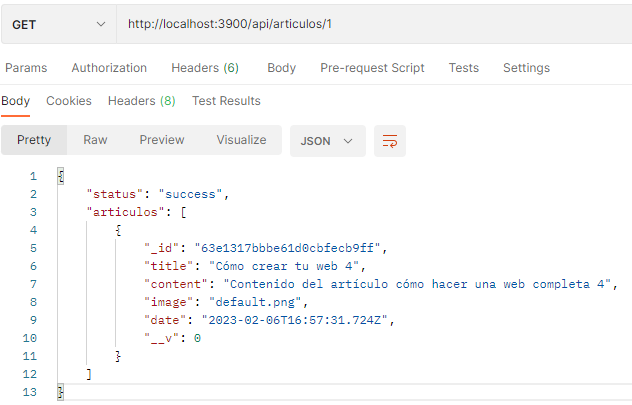
Con **?** le indico que es un parámetro opcional.

En controllers/article recojo el parámetro como

if(req.params.cantidad){

    consulta.limit(req.params.cantidad)

}



Me muestra solo 1 artículo.

Si no le paso ningún parámetro, me muestra todos.

## MOSTRAR UN ARTÍCULO

En routes/article añado la ruta:

router.get('/articulo/:id', ArticleController.uno)

En controllers/article creo el método para mostrar un solo artículo.

const uno = (req,res) => {

    // Recibir id por URL

    let id = req.params.id

    // Buscar

    Article.findById(id, (error, articulo) => {

        // Devolver error

        if(error || !articulo){

            return res.status(404).json({

                status: 'error',

                mensaje: 'No se ha encontrado el artículo',

                error

            })

        }

        // Devolver artículo

        return res.status(200).send({

            status: 'success',

            articulo

        })

    })

}

Y lo exporto

module.exports = {

    prueba,

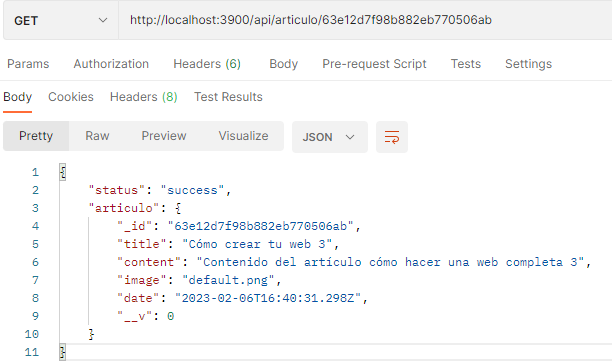
    personas,

    crear,

    listar,

    uno

}



## BORRAR ARTÍCULO

El proceso es similar a la función de buscar un artículo, con un par de diferencias:

DIFERENCIAS en el controller/article

// Buscar

    Article.findById(id, (error, articulo) => {

// Buscar para borrar

    Article.findOneAndDelete({\_id: paramId}, (error, articulo) => {

DIFERENCIAS en el routes/article

router.get('/articulo/:id', ArticleController.uno)

router.delete('/articulo/:id', ArticleController.borrar)

## EDITAR ARTÍCULO

En routes/article añado la ruta:

router.put('/articulo/:id', ArticleController.editar)

En controllers/article actualizo el artículo.

const editar = (req,res) => {

    // Recibir id por URL

    let paramId = req.params.id

    // Recoger datos del body

    let parametros = req.body

    // Validar datos

    try{

        let titleOk = !validator.isEmpty(parametros.title)

            && validator.isLength(parametros.title, {min: 5, max: undefined})

        let contentOk = !validator.isEmpty(parametros.content)

        if(!titleOk || !contentOk){

            throw new Error('No se ha validado la información')

        }

    }catch(error){

        return res.status(400).json({

            status: 'error',

            mensaje: 'Faltan datos por enviar'

        })

    }

    // Buscar y actualizar

    Article.findOneAndUpdate({\_id: paramId},parametros, {new: true},(error, articulo) => {  
  
        // Devolver error

        if(error || !articulo){

            return res.status(500).json({

                status: 'error',

                mensaje: 'Error al actualizar'

            })

        }

        // Devolver artículo actualizado

        return res.status(200).send({

            status: 'success',

            mensaje: 'Artículo eliminado con éxito',

            articulo

        })

    })

}

Y lo exporto

module.exports = {

    prueba,

    personas,

    crear,

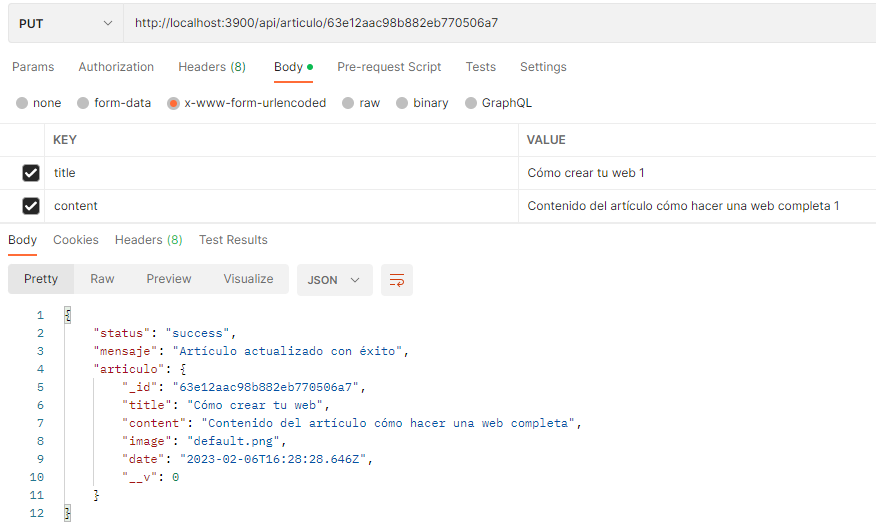
    listar,

    uno,

    borrar,

    editar

}



Lo actualiza bien, pero me devuelve el antiguo. Para que me devuelva el artículo tras actualizarse añado el parámetro **{ new : true }** entre parametros y el callback ( error, articulo )

## HELPER de Validación

En controllers/article creo la función pegando el contenido de las validaciones que hago tanto en CREAR como en EDITAR artículo.

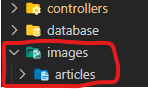
const validarArticulo = (res, parametros) => {

Le tengo que pasar además de los parámetros, el **res**, ya que necesita lanzar un res si hay error (catch).

Ahora en CREAR y EDITAR pongo la llamada a validarArticulo validarArticulo(res, parametros)

## Subir ficheros e imágenes con NodeJS

1. **Creo la carpeta** donde se van a subir las imágenes. La creo en la raíz, y dentro creo una subcarpeta.



1. En routes/article importo multer, configuro el middleware y creo la ruta.

const multer = require('multer')

const almacenamiento = multer.diskStorage({

    destination: function(req, file, cb){

        cb(null, './images/articles')

    },

    filename: function(req, file, cb){

        cb(null, 'articulo' + Date.now() + file.originalname)

    }

})

const uploads = multer({storage: almacenamiento})

En destination cb le indico la ruta donde se va a subir el fichero.  
En filename cb le indico el nombre que va a tener el fichero a subir.

router.post('/subir-imagen/:id', [uploads.single('file0')], ArticleController.subir)

uploads.single porque solo se va a subir un fichero, y file0 es el nombre del campo que le llega por POST.

1. En controllers/article importo fs para manejar ficheros de Windows, creo y exporto el método subir.

const fs = require('fs')

const subir = (req,res) => {

    // Comprobar que se ha enviado un archivo

    if(!req.file && !req.files){

        return res.status(404).json({

            status: 'error',

            mensaje: 'No se ha cargado ningún archivo'

        })

    }

    // Recoger el fichero de imagen subido

    let archivo = req.file.originalname.split('.')

    // Conseguir extensión del archivo

    let extension = archivo[1].toLowerCase()

    // Comprobar extensión correcta

    if(extension!== 'jpg' && extension!== 'jpeg' && extension!== 'png'){

        //Si no es correcta, borro archivo y doy respuesta.

        fs.unlink(req.file.path, (error) => {

            return res.status(400).json({

                status: 'error',

                mensaje: 'Extensión de imagen inválida'

            })

        })

    }

    else{ // Extensión correcta

        // Recibir id por URL

        let paramId = req.params.id

        // Buscar y actualizar

        Article.findOneAndUpdate({\_id: paramId}, {image: req.file.filename},

{new: true}, (error, articulo) =>

{

            // Devolver error

            if(error || !articulo){

                return res.status(500).json({

                    status: 'error',

                    mensaje: 'Error al actualizar'

                })

            }

            // Devolver artículo actualizado

            return res.status(200).send({

                status: 'success',

                mensaje: 'Fichero subido con éxito',

                articulo,

                files: req.file

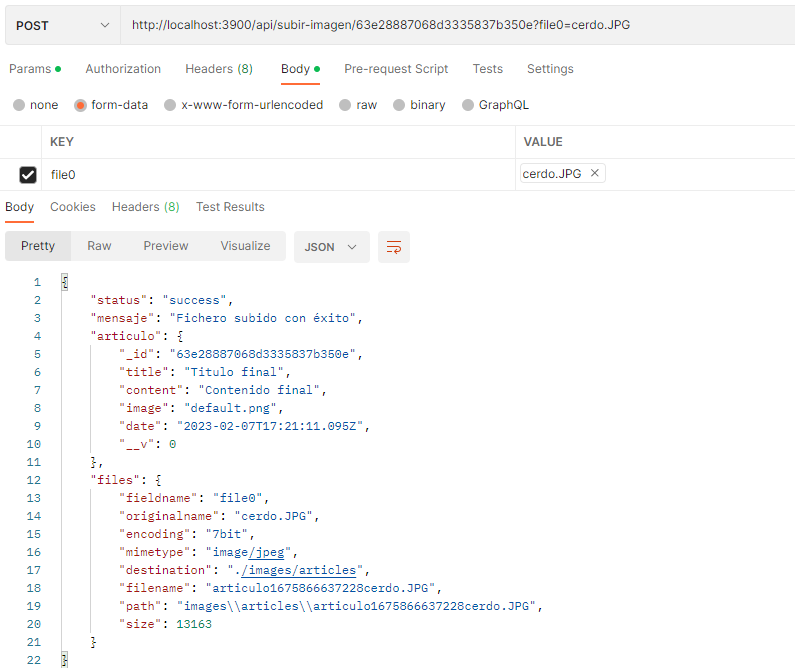
            })

        })

    }

}

1. Probar en Postman



## Actualizar y mostrar imágenes

En controllers/article importo path para manejar rutas, y creo imagen para visualizar imagen en la petición. Al final exporto el método imagen.

const path = require('path')

const imagen = (req, res) => {

    let fichero = req.params.fichero

    let ruta = './images/articles/' + fichero

    fs.stat(ruta, (error, existe) => {

        if(existe){

            return res.sendFile(path.resolve(ruta))

        }

        else{

            return res.status(404).json({

                status: 'error',

                mensaje: 'La imagen no existe'

            })

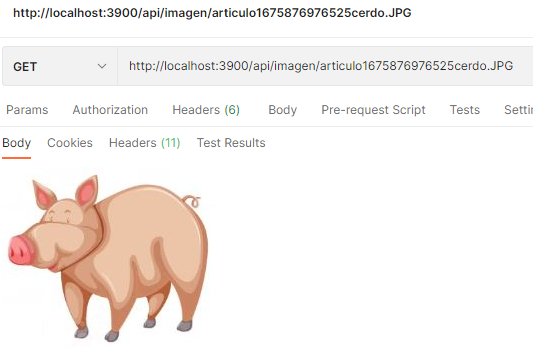
        }

    })

}

En routes/article añado la ruta

router.get('/imagen/:fichero', ArticleController.imagen)



## BUSCAR ARTÍCULO

En routes/article añado la ruta

router.get('/buscar/:busqueda', ArticleController.buscador)

En controllers/article creo la función para buscar un artículo, y exporto el módulo buscador.

const buscador = (req,res) => {

    // Obtener el string a buscar

    let busqueda = req.params.busqueda

    // Find y orden

    Article.find({ "$or": [

        {'title': {'$regex': busqueda, '$options': 'i' }},

        {'content': {'$regex': busqueda, '$options': 'i' }}

    ]})

    .sort({date: -1})

    .exec((error, articulosEncontrados) => {

        if(error || !articulosEncontrados || articulosEncontrados.length <= 0){

            return res.status(404).json({

                status: 'error',

                mensaje: 'No se han encontrado artículos'

            })

        }

        //Devolver resultados

        return res.status(200).json({

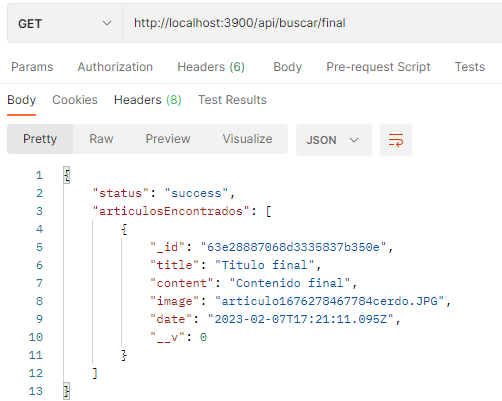
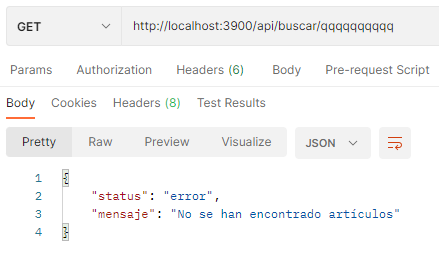
            status: 'success',

            articulosEncontrados

        })

    })

}



## FRONT

### Crear proyecto VITE

Si lo hiciera

### Asignar valores a objeto basado en el modelo

Si lo hiciera

### Asignar valores a objeto basado en el modelo

Si lo hiciera

### Asignar valores a objeto basado en el modelo

Si lo hiciera

### Asignar valores a objeto basado en el modelo

Si lo hiciera

### Asignar valores a objeto basado en el modelo

Si lo hiciera

### Asignar valores a objeto basado en el modelo

Si lo hiciera

### Asignar valores a objeto basado en el modelo

Si lo hiciera

### Asignar valores a objeto basado en el modelo

Si lo hiciera

### Asignar valores a objeto basado en el modelo

Si lo hiciera

### Asignar valores a objeto basado en el modelo

Si lo hiciera

### Asignar valores a objeto basado en el modelo

Si lo hiciera

### Asignar valores a objeto basado en el modelo

Si lo hiciera

### Asignar valores a objeto basado en el modelo

Si lo hiciera

### Asignar valores a objeto basado en el modelo

Si lo hiciera

### Asignar valores a objeto basado en el modelo

Si lo hiciera

### Asignar valores a objeto basado en el modelo

Si lo hiciera

### Asignar valores a objeto basado en el modelo

Si lo hiciera

### Asignar valores a objeto basado en el modelo

Si lo hiciera

### Asignar valores a objeto basado en el modelo

Si lo hiciera

### Asignar valores a objeto basado en el modelo

Si lo hiciera

### Asignar valores a objeto basado en el modelo

Si lo hiciera

### Asignar valores a objeto basado en el modelo

Si lo hiciera

### Asignar valores a objeto basado en el modelo

Si lo hiciera

### Asignar valores a objeto basado en el modelo

Si lo hiciera

### Asignar valores a objeto basado en el modelo

Si lo hiciera

### Asignar valores a objeto basado en el modelo

Si lo hiciera

### Asignar valores a objeto basado en el modelo

Si lo hiciera

### Asignar valores a objeto basado en el modelo

Si lo hiciera

### Asignar valores a objeto basado en el modelo

Si lo hiciera

### Asignar valores a objeto basado en el modelo

Si lo hiciera

### Asignar valores a objeto basado en el modelo

Si lo hiciera

### Asignar valores a objeto basado en el modelo

Si lo hiciera

### Asignar valores a objeto basado en el modelo

Si lo hiciera

## Probando la api con Postman

# PROYECTO 4: Red social

Asasassasa

# FUNCIONES

class Toggle extends React.Component {  
 constructor(props) {  
 super(props);  
 this.state = {isToggleOn: true};  
 }  
}



this.setState(prevState => ({  
 isToggleOn: !prevState.isToggleOn

}));



cambioInput = event => {  
 this.setState({temperatura: event.target.value});  
}

…

<input type="text" value={this.state.temperatura} onChange={this.cambioInput} />