[Creando APIs con Node JS, Express y MongoDB](https://legacy.gitbook.com/book/gmoralesc/creando-apis-con-node-js-express-y-mongodb)

Portada

[Introducción](https://gmoralesc.gitbooks.io/creando-apis-con-node-js-express-y-mongodb/content/)

Configurando el proyecto con express

[Crear un simple Web Server](https://gmoralesc.gitbooks.io/creando-apis-con-node-js-express-y-mongodb/content/configurando-el-proyecto-con-express/crear-un-simple-web-server.html)

[Utilizando Express](https://gmoralesc.gitbooks.io/creando-apis-con-node-js-express-y-mongodb/content/configurando-el-proyecto-con-express/utilizando-express.html)

[Herramientas de desarrollo](https://gmoralesc.gitbooks.io/creando-apis-con-node-js-express-y-mongodb/content/configurando-el-proyecto-con-express/configurando-el-ambiente-de-desarrollo.html)

[Configuración y variables de entorno](https://gmoralesc.gitbooks.io/creando-apis-con-node-js-express-y-mongodb/content/configurando-el-proyecto-con-express/configuracion-y-variables-de-entorno.html)

[Middleware, manejo de errores y logs](https://gmoralesc.gitbooks.io/creando-apis-con-node-js-express-y-mongodb/content/configurando-el-proyecto-con-express/middleware-manejo-de-errores-y-logs.html)

Router y Routes en Express

[Utilizando el Router y Routes de Express](https://gmoralesc.gitbooks.io/creando-apis-con-node-js-express-y-mongodb/content/router-y-routes-en-express/utilizando-el-router-y-routes-de-express.html)

[Creando el layout del API](https://gmoralesc.gitbooks.io/creando-apis-con-node-js-express-y-mongodb/content/router-y-routes-en-express/creando-el-layout-del-api.html)

[Capturando y procesando parámetros de las peticiones](https://gmoralesc.gitbooks.io/creando-apis-con-node-js-express-y-mongodb/content/router-y-routes-en-express/capturando-y-procesando-parametros-de-las-peticiones.html)

Persistencia de datos con MongoDB

[Instalando y configurando MongoDB](https://gmoralesc.gitbooks.io/creando-apis-con-node-js-express-y-mongodb/content/persistencia-de-datos-con-mongodb/instalando-mongo-db.html)

[Conectando con MongoDB](https://gmoralesc.gitbooks.io/creando-apis-con-node-js-express-y-mongodb/content/persistencia-de-datos-con-mongodb/conectando-con-mongodb.html)

[Mongoose Models](https://gmoralesc.gitbooks.io/creando-apis-con-node-js-express-y-mongodb/content/persistencia-de-datos-con-mongodb/mongoose-models.html)

[Procesando parámetros comunes con middleware](https://gmoralesc.gitbooks.io/creando-apis-con-node-js-express-y-mongodb/content/persistencia-de-datos-con-mongodb/procesando-parametros-comunes-con-middleware.html)

[Estandarización de la respuesta](https://gmoralesc.gitbooks.io/creando-apis-con-node-js-express-y-mongodb/content/persistencia-de-datos-con-mongodb/estandarizacion-de-la-respuesta.html)

[Paginación](https://gmoralesc.gitbooks.io/creando-apis-con-node-js-express-y-mongodb/content/persistencia-de-datos-con-mongodb/paginacion.html)

[Mongoose Schemas](https://gmoralesc.gitbooks.io/creando-apis-con-node-js-express-y-mongodb/content/persistencia-de-datos-con-mongodb/mongoose-schemas.html)

[Ordenamiento](https://gmoralesc.gitbooks.io/creando-apis-con-node-js-express-y-mongodb/content/persistencia-de-datos-con-mongodb/ordenamiento.html)

[Creando recursos](https://gmoralesc.gitbooks.io/creando-apis-con-node-js-express-y-mongodb/content/persistencia-de-datos-con-mongodb/creando-recursos.html)

[Relaciones entre recursos](https://gmoralesc.gitbooks.io/creando-apis-con-node-js-express-y-mongodb/content/persistencia-de-datos-con-mongodb/relaciones-entre-colecciones.html)

[Consultar recursos anidados](https://gmoralesc.gitbooks.io/creando-apis-con-node-js-express-y-mongodb/content/persistencia-de-datos-con-mongodb/consultar-recursos-anidados.html)

[Añadir recursos anidados](https://gmoralesc.gitbooks.io/creando-apis-con-node-js-express-y-mongodb/content/persistencia-de-datos-con-mongodb/anadir-recursos-anidados.html)

Asegurando el API

[Añadir y remover campos de un documento](https://gmoralesc.gitbooks.io/creando-apis-con-node-js-express-y-mongodb/content/asegurando-el-api/anadir-y-remover-campos-de-un-documento.html)

[Encriptando información sensible](https://gmoralesc.gitbooks.io/creando-apis-con-node-js-express-y-mongodb/content/asegurando-el-api/encriptando-informacion-sensible.html)

[Publicado con GitBook](https://www.gitbook.com/)

[Procesando parámetros comunes con middleware](https://gmoralesc.gitbooks.io/creando-apis-con-node-js-express-y-mongodb/content/)

**Procesando parámetros comunes con middleware**

Una vez hemos creado y leemos objetos de la base de datos, ahora es la oportunidad para de realizar otras operaciones como buscar un documento en especifico, editar y eliminar, pero estas 3 operaciones básicas tienen un elemento en común primero debemos buscar el documento basado en el id y luego ejecutar la operación.

Primero comencemos con la operación de buscar un documento, para ellos vamos a editar la acción de read de nuestro controlador ( server/api/v1/posts/controller.js ):

...
exports.read = (req, res, next) => { const { params } = req; const { id } = params;
Model.findById(id)
.then( (doc) => { if (doc) {
res.json(doc);
} else { const message = 'Post not found' ;
logger.info(message);
res.json({
message
});
}
})
.catch( err => {
next( new Error (err));
});
};
...

Podemos observar en el fragmento de código anterior que esta vez utilizamos la función findById de *Mongoose* que es una abreviatura específica de la función find pero esta vez espera explicitamente el parámetro id .

Si fueramos a editar la accion de update nos damos cuenta que lo primero que tenemos que hacer es buscar el documento por el id y lo mismo en la acción de delete, estariamos repitiendo código y lo cual indica que debemos optimizarlo.

Para solucionar esto *Express* tiene un *middleware* para procesar los parámetros y pasar al siguiente *middleware* , para esto primero editamos el archivo de rutas ( server/api/v1/posts/routes.js ) y añadimos la siguiente línea:

...
router
.param( 'id' , controller.id);
router
.route( '/' )
.get(controller.all)
.post(controller.create);
...

Con este cambio le estamos indicando a express que cuando existe el parámetro id en la ruta actual llame a una función del controlador llamada id la cual creamos a continuación en nuestro archivo del controlador ( server/api/v1/posts/controller.js ):

...
const Model = require ( './model' );
exports.id = (req, res, next, id) => {
Model.findById(id).exec()
.then((doc) => { if (doc) {
req.doc = doc;
next();
} else { const message = ` ${Model.modelName} not found` ;
logger.info(message);
res.json({
message,
});
}
})
.catch((err) => {
next( new Error (err));
});
};
...

Aquí podemos observar que tomamos las mismas líneas de código de la acción read pero con sutiles cambios:

Una vez encontramos el documento lo guardamos en el mismo objeto de la petición req.doc = doc

Una vez tenemos el documento llamamos a la función next() para que el próximo *middleware* que sigue procese esta petición en este caso la acción read .

Si no existe ningún documento con el id enviado creamos un mensaje de respuesta extrayendo el nombre del modelo actual para que sea genérico y el flujo termina pues estamos dando respuesta a la petición.

Entonces como quedaria nuestra acción de read si transportamos la logica de buscar el documento al middleware que procesa el parámetro id ? Pues veamos:

...
exports.read = (req, res, next) => { const { doc } = req;
res.json(doc);
};
...

Lo interesante de este patrón de *middleware* es cuando se realicé una petición para buscar el documento por el id primero llega al *middleware* que procesa dicho parámetro, si y solo sí existe el documento lo almacena en la misma petición y deja que el siguiente middleware siga procesando la petición ( next ), en caso contrario nunca llegará al siguiente *middleware* ya que termina la petición dandole una respuesta al usuario, realmente este el poder de los *middleware* , entenderlos y organizarlos puede ser muy ventajoso para la aplicación.

Con esto en cuenta ahora modificamos nuestra acción update :

...
exports.update = (req, res, next) => { const { doc, body } = req; Object .assign(doc, body);
doc.save()
.then((updated) => {
res.json(updated);
})
.catch((err) => {
next( new Error (err));
});
};
...

Finalmente aprovechamos el procesamiento del usuario y creamos nuestra acción dd delete :

exports.delete = (req, res, next) => { const { doc } = req;
doc.remove()
.then((removed) => {
res.json(removed);
})
.catch((err) => {
next( new Error (err));
});
};

Con esto hemos terminado nuestra operaciones básicas de CRUD para nuestros usuarios.

Antes de continuar guardamos nuestro progreso:

git add .
git commit -m "Process id param, update and delete for Posts"