[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/)

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

**Ordenamiento**

Otra de las operaciones mas comunes que realiza el usuario cuando consulta los documentos de una colección es ordenarlos, ahora con la fecha de creación y modificación tenemos los campos completos para realizar esta operación, nuevamente crearemos los siguientes parámetros:

sortBy : Nombre del campo por el cual se va a ordenar la coleeción, si el usuario no lo especifica o el nombre no se encuentra dentro del listado de campos del modelo por defecto se establecerá en la fecha de creación.

direction : Indica la dirección en la cual se va a ordenar dicha colección las opciones son ascendentemente y descendentemente, si el usuario no lo especifica o la opción no se encuentra dentro de las opciones predeterminadas por defecto el valor se establecerá en descendentemente.

Vamos a necesitar los campos del modelo por ende modificamos nuestro modelo ( server/api/v1/posts/model.js ) para exportar dichos campos:

...
module.exports = {
Model: mongoose.model( 'post' , post),
fields,
};

En el fragmento de código anterior estamos cambiando la forma como exportamos nuestro modelo, por ende tocará cambiarlo como lo importamos en el controlador, lo cual realizaremos mas adelante.

A continuación añadiremos a nuestro archivo de configuración ( server/api/config/index.js ) las opciones respectivas:

require ( 'dotenv' ).config(); const config = {
...
sort: {
sortBy: { default : 'createdAt' ,
fields: [ 'createdAt' , 'updatedAt' ],
},
direction: { default : 'desc' ,
options: [ 'asc' , 'desc' ],
},
},
}; module .exports = config;

Notesé que estamos indicado en el campo fields los valores que añade la configuración de timestamps en el *Schema* del modelo, pues no hacen parte del listado de campos definidos para el modelo cuando lo exportamos.

Procedemos a modificar nuestro controlador ( server/api/v1/posts/controller.js ):

const logger = require ( 'winston' ); const {
parsePaginationParams,
parseSortParams,
compactSortToStr,
} = require ( './../../../utils' ); const {
Model,
fields,
} = require ( './model' );
...
exports.all = (req, res, next) => { const {
query = {},
} = req; const {
limit,
page,
skip,
} = parsePaginationParams(query); const {
sortBy,
direction,
} = parseSortParams(query, fields); const all = Model
.find()
.sort(compactSortToStr(sortBy, direction))
.limit(limit)
.skip(skip); const count = Model.count();
...
Promise.all([all.exec(), count.exec()])
.then((data) => { const [docs, total] = data; const pages = Math .ceil(total / limit);
res.json({
success: true ,
items: docs,
meta: {
limit,
skip,
total,
page,
pages,
sortBy,
direction,
},
});
})
.catch((err) => {
next( new Error (err));
});
};
...

Detallamos a continuación los cambios añadidos al controlador:

Estamos importando dos nuevas funciones: parseSortParams y compactSortToStr , la primera para realizar el mismo proceso que realizamos con los parámetros de paginación y la segunda es para convertir esos dos campos por lo cuales se va a ordenar a la forma que *Mongoose* los necesita ( [Query.sort](http://mongoosejs.com/docs/api.html#query_Query-sort) ).

Cambiamos la forma en que importamos el modelo y adicionalmente importamos los campos del modelo respectivo.

Creamos las variables sortBy y direction a partir de la función parseSortParams que recibe como parámetros el *queryString* que viene en la petición y los campos. Como su nombre lo indica sortBy será el nombre del campo por el cual se va a ordenar y direction si es ascendentemente o descendentemente.

Añadimos en cadena la función sort después de la función find para efectivamente ordenar los documentos, como parámetro de la función sort invocamos la función compactSortToStr que recibe a su vez los parámetros del campo a ordenar y la dirección.

Finalmente devolvemos en el objeto meta porque llave esta ordenada la colección y en que dirección.

Finalmente añadimos a nuestro archivo de utilidades ( server/utils/index.js ) las funciones mencionadas anteriormente:

const config = require ( './../config' ); const {
pagination,
sort,
} = config;
...
const parseSortParams = ({
sortBy = sort.sortBy.default,
direction = sort.direction.default,
}, fields) => { const whitelist = {
sortBy: [...Object.getOwnPropertyNames(fields), ...sort.sortBy.fields],
direction: sort.direction.options,
}; return {
sortBy: whitelist.sortBy.includes(sortBy) ? sortBy : sort.sortBy.default,
direction: whitelist.direction.includes(direction) ? direction : sort.direction.default,
};
}; const compactSortToStr = (sortBy, direction) => { const dir = direction === sort.direction.default ? '-' : '' ; return ` ${dir} ${sortBy} ` ;
}; module .exports = {
parsePaginationParams,
parseSortParams,
compactSortToStr,
};

Nuevamente expliquemos cada una de las funciones añadidas: primero hablemos de parseSortParams :

De la misma forma que hicimos con la función similar para los parámetros de paginación utilizamos los *Default Values* para asignarle un valor si no son enviados.

Armamos el objeto whitelist como su nombre lo dice contendrá solo el listado de campos y opciones permitidas para cada uno para el campo por el cual se va a ordenar y el sentido respectivamente, para armar la propiedad sortBy utilizamos el *Spread Operator* y colocamos el contenido de cada *Array* dentro de un nuevo *Array* de alguna forma concatenamos el contenido de los campos del modelo y los que están guardados por defecto en la configuración para armar uno solo con todos los campos posibles.

Finalmente retornamos el valor del parámetro si y solo si existe en la lista de campos u opciones permitidas, si no, enviamos el valor por defecto.

La función compactSortStr es bastante sencilla, pero la necesitamos debido a la manera como *Mongoose* pide el parámetro para la función de sort cuando no es un campo especificado en el código, es un campo dinámico, por ejemplo, si el valor de sortBy es: createdAt y es ordenado descendentemente, *Mongoose* lo requerirá de la siguiente forma: '-createdAt' y eso es precisamente lo que hace la función.

Más información:

[Mongoose Query Sort](http://mongoosejs.com/docs/api.html#query_Query-sort)