[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/)

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

**Utilizando el Router y Routes en Express**

Antes de continuar vamos a mezclar el *feature* anterior y crear una rama para trabajar:

git checkout master
git merge feature/05-handle-errors-and-logs
git checkout -b feature/06-using-router-and-routes

**Definiendo las rutas con Route**

Como ya vimos anteriormente hemos definido la ruta para la raíz de nuestra API en el archivo :

...
app.get( '/' , (req, res, next) => {
logger.info( 'API root' );
res.json({
message: 'Welcome to the API' ,
});
});
...

De la misma manera manera podemos definir todas las rutas que necesitamos para nuestra API, modificamos la definición anterior de la ruta raíz en el archivo server/index.js de la siguiente manera:

...
app.get( '/api/posts' , (req, res, next) => {
logger.info( 'GET all posts' );
res.json({
message: 'GET all posts' });
});
...

De esta manera se pueden definir todas las rutas que necesitemos y dentro de cada una de ellas hacer todo el procesamiento que necesitemos, pero imaginemos por un momento que hemos definido 10 rutas más con diferentes verbos HTTP para las diferentes operaciones relacionadas con este recurso ( *posts* ) y por algún motivo nos toca cambiar el nombre del recurso de /api/posts a /api/messages/ nos tocaría modificar todas las rutas lo cual no es muy dinámico e inclusive se puede incurrir en errores, para esto *express* tiene un objeto llamado ***Route*** el cual permite establecer un punto de montaje para el recursos y adjuntarle todas las operaciones con los verbos que queramos, entonces tomando nuevamente el ejemplo anterior solo nos tocaría modificar el recurso en el punto de montaje y no en todas las rutas.

Veamos cómo se utiliza realizando los cambios en nuestro archivo de server/index.js :

Cambiar la definición de / api/posts , introduciendo el route de *express* :

...
app.route( '/api/posts' )
.get((req, res, next) => {
logger.info( 'GET all posts' );
res.json({
message: 'GET all posts' });
});
...

Tenemos un cambio significativo en la estructura de la definición, el punto de montaje se establece con el objeto route y le "anexamos" el método GET con el *middleware* correspondiente, pero también se le puede anexar diferentes métodos como: GET, POST, PUT, DELETE, etc. Lo interesante es que solo dependen del primer parámetro: el nombre de la ruta.

Creamos un directorio que contendrá todos los directorios y archivos relacionados con el API

mkdir -p server/api
touch server/api/index.js

Editamos el último archivo creado server/api/index.js , cortamos la definición de la ruta del archivo server/index.js y lo convertimos en un módulo con el siguiente contenido:

const router = require ( 'express' ).Router(); const logger = require ( 'winston' );
router.route( '/api/posts' )
.get((req, res, next) => {
logger.info( 'GET all posts' );
res.json({
message: 'GET all posts' });
}); module .exports = router;

En el fragmento de código anterior estamos importando directamente el *Router* no toda la librería de *express* aunque se puede hacer el dos lineas de código también.

Modificamos el archivo server/index.js importando el módulo definido anteriormente y reemplazando la definición de la ruta por el punto de montaje:

const express = require ( 'express' ); const morgan = require ( 'morgan' ); const api = require ( './api' ); const app = express(); // Setup middleware app.use(morgan( 'common' )); // Setup router and routes app.use( '/api' , api);
...

Hemos realizado unos cambios muy importantes, en cuanto a organización, responsabilidad y modularidad:

Hemos creado un Router para manejar todas las operaciones relacionadas con el recurso de *posts* , pero estas a su vez son independientemente del prefijo que se vaya a utilizar, es decir, ya no tiene marcado en el código antes de cada ruta el prefijo /api .

Hemos creado el punto de montaje /api pero esta vez le hemos adjuntado el *Router* que contiene todas las operaciones del recurso /posts (por el momento), es decir que al final para acceder al recurso sigue siendo /api/posts .

Si podemos visualizar esto nos brinda una enorme ventaja a la hora de organizar los recursos y renombrarlos.

Ahora si vamos al navegador a la dirección localhost:3000/api/tasks obtenemos el siguiente resultado:

{
"message": "Get all posts"
}

Antes de continuar guardamos nuestro progreso:

git add .
git commit -m "Setting an API mount point and basic router for posts"

Mas información:

[Express Routing](http://expressjs.com/es/guide/routing.html)

**Manejo de versiones de API**

Es normal que un API tenga diferentes versiones en el tiempo, pues surgen nuevas funcionalidades o cambios importantes, pero debemos asegurar cuando vayamos a realizar estos cambios no dañar la versión actual del API que ya está publicada y la pueden estar utilizando muchas personas en ese momento. Con los cambios que introducimos anteriormente será más sencillo manejar las versiones del API.

Es recomendable siempre versionar el API desde la creación de la misma, para lo cual ejecutamos los siguientes comandos:

mkdir -p server/api/v1
mv server/api/index.js server/api/v1/index.js

Finalmente cambiamos el punto de montaje de nuestra API en el archivo server/index.js , modificando las siguientes líneas:

...
const api = require ( './api/v1' );
... // Setup router and routes app.use( '/api' , api);
app.use( '/api/v1' , api);
...

Este cambio nos permite utilizar dos puntos de montaje para el mismo API. Normalmente la última versión del API se utiliza bajo el prefijo /api solamente, y si el usuario quiere utilizar alguna versión específica le coloca el prefijo /api/v1 . Aprovechando al máximo la independencia que provee el *Router* de *express* .

Si en el futuro lanzamos la versión 2 del API solo tendríamos que crear el directorio llamado v2 (al mismo nivel que v1) y asociarlo con el punto de montaje /api/v2 y a su vez /api , garantizando así que los usuarios que desean acceder a la versión "legacy" (Versión 1) de nuestra API sería exclusivamente con el prefijo /api/v1 .

Antes de continuar guardamos nuestro progreso:

git add .
git commit -m "Versioning API"