


# Creando el layout del API

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

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

```
git checkout master
git merge feature/06-using-router-and-routes
git checkout -b feature/07-create-api-layout
```

Hasta el momento hemos organizado un poco mejor nuestra API, pero no es suficiente ya que la ruta de `/posts` y todas las que creemos a partir de allí están nuevamente están atadas al prefijo, por lo tanto una buena práctica es crear un directorio por cada recurso, pero también un archivo para cada responsabilidad, el primero sería el módulo de las rutas. Creamos el directorio y archivo correspondiente:

```
mkdir -p server/api/v1/posts
touch server/api/v1/posts/routes.js
```

Copiamos el contenido de `server/api/v1/index.js` a el nuevo archivo creado `server/api/v1/posts/routes.js` pero con un ligero cambio, reemplazar el punto de montaje de `/posts` a `/`

```
const router = require('express').Router();
const logger = require('winston');

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

module.exports = router;
```

Ahora reescribimos nuestro archivo principal del API ( `server/api/v1/index.js` ) y establecemos el punto de montaje para la ruta de `/posts`

```
const router = require('express').Router();

const posts = require('./posts/routes');

router.use('/posts', posts);

module.exports = router;
```

Aquí hemos llegado a la granularidad de los montajes de las rutas y estas son independientes de sus puntos de montaje, por ejemplo si el día de mañana deseamos cambiar `/posts` por `/messages` sencillamente lo hacemos en el archivo principal del API y las rutas seguirán funcionando sin ningún problema, pues nuevamente ya no dependen de prefijos.

Aunque no lo creas todavía no es suficiente, podemos llegar a un nivel de granularidad aún mayor, enfoquémonos ahora en la entidad de `posts`, nuevamente apliquemos lo que hicimos con el punto de montaje del API cambiamos los puntos de montaje por rutas en el archivo `server/api/v1/posts/routes.js` de la siguiente manera:

```
const router = require('express').Router();

router.route('/')
  .get((req, res, next) => {})
  .post((req, res, next) => {});

module.exports = router;
```

Como podemos observar sobre una misma ruta podemos adjuntar diferentes verbos en este caso GET y POST, hemos reducido el *middleware* a su mínima expresión, temporalmente removimos la librería de winston pues en este momento no la necesitamos.

## Contrato REST API

El contrato por defecto de REST API sugiere con que verbos y rutas se deben hacer las operaciones del CRUD con los recursos, la siguiente tabla mostrará como seria para nuestra entidad `posts`:

Ruta	Verbo	Status Code	Operación	Descripción
/api/posts/	POST	201	CREATE	Crear un post
/api/posts/	GET	200	READ	Leer todos los posts
/api/posts/:id	GET	200	READ	Leer un post
/api/posts/:id	PUT / PATCH	200	UPDATE	Actualizar la información de un post
/api/posts/:id	DELETE	200	DELETE	Eliminar un post

Por lo tanto establezcamos el layout del API para la entidad de `posts`, sus respectivas rutas y los verbos asociados, modificamos el archivo `server/api/v1/posts/routes.js`

```
const router = require('express').Router();

router.route('/')
  .post((req, res, next) => {});
  .get((req, res, next) => {});

router.route('/:id')
  .get((req, res, next) => {})
  .put((req, res, next) => {})
  .delete((req, res, next) => {});

module.exports = router;
```

Ahora nuestras rutas para la entidad `posts` tienen al menos el contrato de REST API estándar, pero aún nos falta un paso final para terminar de organizar los *middlewares* asociados a estas.

La cadena `:id` es un parámetro dinámico el cual veremos con más detalle en la próxima sección.

Vamos a introducir un término conceptual que es: el controlador, el cual contendrá las acciones asociadas a cada verbo de la ruta. Procedemos a crear el archivo:

```
mkdir -p server/api/v1/posts/  
touch server/api/v1/posts/controller.js
```

Y colocamos el siguiente contenido en el archivo creado previamente:

```
exports.create = (req, res, next) => {  
  res.json({});  
};  
  
exports.all = (req, res, next) => {  
  res.json({});  
};  
  
exports.read = (req, res, next) => {  
  res.json({});  
};  
  
exports.update = (req, res, next) => {  
  res.json({});  
};  
  
exports.delete = (req, res, next) => {  
  res.json({});  
};
```

Podemos notar varias cosas:

- Las acciones no necesariamente se deben llamarse como los verbos, por ejemplo `all`, el cual vamos a utilizar para obtener el listado de todas las publicaciones cuando hagamos un GET a la raíz de la entidad posts ( `/api/v1/posts` )
- Estamos utilizando *named exports* solo por comodidad para exportar individualmente cada una de las acciones.
- Estamos dejando una respuesta genérica en cada acción que es un objeto JSON vacío.

Finalmente modificamos las siguientes líneas el archivo de montaje de las rutas ( `server/api/v1/posts/routes.js` ) asociando cada acción de nuestro controlador al respectivo e inclusive si fuese el caso pudiéramos utilizar la misma acción para diferentes verbos en diferentes rutas:

```
const router = require('express').Router();

const controller = require('./controller');

...

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

router.route('/:id')
  .get(controller.read)
  .put(controller.update)
  .delete(controller.delete);

module.exports = router;
```

Antes de continuar guardamos nuestro progreso:

```
git add .
git commit -m "Create API layout for Posts"
```