


Capturando y procesando parámetros de las peticiones

 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

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

```
git checkout master
git merge feature/07-create-api-layout
git checkout -b feature/08-handling-params-from-requests
```

Estamos construyendo un API, por lo cual tendremos que trabajar con otros verbos URL diferentes a GET como POST, PUT y DELETE que no lo podemos hacer directamente con el navegador, ya que cada vez que introducimos la URL están haciendo una petición GET exclusivamente, para solventar esta necesidad existe una herramienta muy popular llamada POSTMAN, la cual ofrece una interfaz gráfica para realizar todas las peticiones (*requests*) y ver las respectivas respuesta (*responses*) el cual veremos con más detalle más adelante, pero en esta sección utilizaremos la línea de comando, probemos obtener el listado de todas las publicaciones:

```
curl http://localhost:3000/api/posts/
```

Se recomienda utilizar otra sesión de la consola ya que *nodemon* está ejecutándose en la consola actual que tenemos

Si observamos en nuestra consola (donde se está ejecutando *nodemon*) *morgan* se ha encargado de registrar el acceso a la ruta, lo cual nos será muy útil para ver las peticiones captadas el servidor:

```
127.0.0.1 - - [15/Apr/2018:17:36:37 +0000] "GET /api/posts/ HTTP/1.1" 200 62
```

Modifiquemos la acción en el archivo `server/api/v1/posts/controller.js` para devolver un *Array* con publicaciones de muestra:

```
...

exports.all = (req, res, next) => {
  res.json([
    { "_id": 1, "title": "Vacation" },
    { "_id": 2, "title": "Jogging" }
  ]);
};

...
```

Nuevamente ejecutamos la petición en la línea de comando:

```
curl http://localhost:3000/api/posts/
```

Por defecto el comando curl utiliza GET

Y en la consola de las peticiones podemos la respuesta (response):

```
[{"_id":1,"title":"Vacation"}, {"_id":2,"title":"Jogging"}]
```

Capturar parametros por la URL

Anteriormente establecimos cual seria la ruta para obtener una tarea:

```
api/posts/:id
```

Por lo cual TODO lo que se envíe en la petición después de `/api/posts/` quedará almacenado en el parámetro `id`, modifiquemos la acción de obtener una tarea en particular en el controlador de tareas (`/server/api/v1/tasks/controller.js`):

```
...  
  
exports.read = (req, res, next) => {  
  res.json({ "_id": req.params.id });  
};  
  
...
```

En el código anterior estamos extrayendo el parámetro `id` de la petición, por defecto `express` deja todos los parámetros enviados por la URL en un objeto llamado `params` en el objeto de la petición (`req`), ejecutamos la petición:

```
curl http://localhost:3000/api/posts/1
```

Podemos observar el resultado en nuestra consola de peticiones:

```
{"_id": "1"}
```

Algo muy importante a tener en cuenta es que no estrictamente el `id` tienen que ser números, de hecho el tomara todo lo que está después del `/`, como por ejemplo:

- `/api/posts/abc1`, entonces `id` será igual a `abc1`
- `/api/posts/filter`, entonces `id` será igual a `filter`

Esto lo utilizaremos más adelante para buscar una publicación en la opción de persistencia que hayamos seleccionado como una base de datos.

Capturar datos desde formularios u objetos json

En las peticiones los datos enviados por los métodos POST and PUT son enviados en el cuerpo (`body`) de la petición, por lo tanto no podemos obtenerlos por `req.params` si no por `req.body`, pero antes de hacer la petición modifiquemos la acción de POST del controlador de tareas (`server/api/v1/posts/controller.js`):

```
...  
  
exports.create = (req, res, next) => {  
  res.json(req.body);  
};  
  
...
```

Ahora vamos a emular enviar datos por el método POST y un objeto json con los datos del usuario:

```
curl http://localhost:3000/api/posts --request POST --data '{ "_id": 3, "title": "Reading a book" }'
```

Para emular el envío de datos mediante un formulario en vez de utilizar `--data` y el payload se podría utilizar `--form _id=3 --form "title=Reading a book"`

Lamentablemente no obtenemos la respuesta que deseamos ya que el cuerpo del documento se debe procesar antes, para esto existe una librería que nos ayudará con esta parte llamada *body parser*, instalamos el paquete como una dependencia de nuestro proyecto:

```
npm install -S body-parser
```

Incluimos las funciones de *middleware* en el archivo de `server/index.js`:

```
...

const logger = require("winston");
const bodyParser = require('body-parser');

const api = require("./api/v1");

const app = express();

app.use(morgan("common"));

app.use(bodyParser.urlencoded({ extended: false }));

app.use(bodyParser.json());

...
```

Ahora intentamos una vez más:

```
curl http://localhost:3000/api/tasks --request POST --data '{ "_id": 3, "title": "Reading a book" }'
```

Esta vez obtenemos la respuesta que esperábamos ya que *body-parser* se encargó de formatear los datos del cuerpo del documento y guardarlo en el objeto `body` dentro del objeto `req`.

Podemos modificar las otras acciones del controlador de usuarios para al menos retornar los parámetros recibidos:

```
exports.all = (req, res, next) => {
  res.json([]);
};

exports.create = (req, res, next) => {
  res.json(req.body);
};

exports.read = (req, res, next) => {
  res.json({
    _id: req.params.id,
  });
};

exports.update = (req, res, next) => {
  res.json({
    _id: req.params.id,
    item: req.body,
  });
};

exports.delete = (req, res, next) => {
  res.json({
    _id: req.params.id,
  });
};
```

Antes de continuar guardamos nuestro progreso:

```
git add .
git commit -m "Handling params from request"
```

Mas información:

[body-parser](#)