ExpressJS: Peticiones, respuestas y rutas

Gabriel Rodríguez Flores

November 16, 2021

ADA ITS
INSTITUTO TECNOLÓGICO SUPERIOR

    - Atender a peticiones
  - Manejar las rutas
  - Organización de rutas
  - Métodos
  - Params en rutas (/:id)

Contents

1 Teoría

1.1 Request

El objeto con toda la información de la petición entrante por parte del cliente

1.1.1 headers

Datos de la cabecera, son datos informativos que no suelen tener relación con la petición.

El uso más extendido es para el token de autenticación.

1.1.2 query

Parámetros opcionales que se definen con el formato `clave=valor` a la derecha de la ruta a partir del carácter `?`, encadenados por el carácter `&`

- Ejemplo: `localhost:3000?name=Gabri&age=28`

1.1.3 params

Son valores integrados en la ruta que pueden ser dinámicos y extraidos como variables.

- Ejemplo: `localhost:3000/names/Gerardo`
- Ejemplo: `localhost:3000/names/Gema`
- Ejemplo: `localhost:3000/names/Gabriel`

1.1.4 body

El cuerpo de la petición, usado en los métodos POST, PUT y PATCH.

> Nota: Para poder obtener los datos en el objeto `req.body` hay que instalar un módulo llamado `body-parser` y usarlo como middleware para el servidor. (Vease el apartado ejemplos)

```
1 const bodyParser = require('body-parser');
2 server.use(bodyParser.json());
3 server.use(bodyParser.urlencoded({ extended: true }));
```

> A partir de la versión 4.16.0 de ExpressJS viene integrado directamente

```
server.use(express.json());
server.use(express.urlencoded({ extended: true }));
```

1.2 Response

El objeto que contiene la información y métodos para enviar la respuesta al cliente.

- método `status` o propiedad `statusCode`

```
res.status(200);
res.statusCode = 200;
```

- método `send` o método `json`

```
res.send();
res.json();
```

1.3 Router

1.3.1 Método `route`

```
app.route('/book')
  .get(function(req, res) {
    res.send('Get a random book');
  })
  .post(function(req, res) {
    res.send('Add a book');
  })
  .put(function(req, res) {
    res.send('Update the book');
  });
```

1.3.2 Clase `Router`

El enrutador se utiliza para agrupar rutas en un elemento a parte, que será incluido en el servidor (u otro enrutador) para tener una estructura en arbol y facilitar su gestión.

Su uso es similar a implementar la rutas en el servidor, sólo que usando el objeto enrutador, que será el que se use en el objeto servidor.

```
const express = require('express');

const server = express();
const router = express.Router();

server.use('/birds', birds);

server.listen(3000, () => {
  console.log(`Example app listening`)
});
```

2 Ejemplos

2.1 Request

2.1.1 Recoger los datos de la cabecera

```
const express = require('express');

const server = express();

server.get('/', function(req,res){
  const { mycustomheader } = req.headers;
  res.status(200).send(mycustomheader);
});

server.listen(3000, () => {
  console.log(`Example app listening`)
});
```

- Ejecución para la prueba

```
curl localhost:3000/name -H "myCustomHeader: Here can be any value"
```

2.1.2 Recoger los datos de la url

```
const express = require('express');

const server = express();

server.get('/:name', function(req,res){
  const { name } = req.params;
  res.status(200).send(`Hello ${name}`);
});
```

```
9
10 server.listen(3000, () => {
11   console.log(`Example app listening`)
12 });
```

- Ejecución para la prueba (o acceder con el navegador)

```
1 curl localhost:3000/Gabriel
```

2.1.3 Recoger los datos de la query

```
1  const express = require('express');
2
3  const server = express();
4
5  server.get('/', function(req,res){
6    const { name = 'World', age } = req.query;
7    res.status(200).send(`Hello ${name}${age ? `, you are ${age} years
       old` : '!'}`);
8  });
9
10 server.listen(3000, () => {
11   console.log(`Example app listening`)
12 });
```

- Ejecución para la prueba (o acceder con el navegador)

```
1 curl localhost:3000?name=Gabriel&age=28
```

2.1.4 Recoger los datos del body

```
1  const express = require('express');
2  const bodyParser = require('body-parser');
3
4  const server = express();
5  server.use(bodyParser.json());
6  server.use(bodyParser.urlencoded({ extended: true }));
7
8  server.post('/:key', function(req,res){
9    const { body, params: { key } } = req;
10   res.status(200).send(body[key]);
11 });
12
13 server.listen(3000, () => {
14   console.log(`Example app listening`)
15 });
```

- Ejecución para la prueba

```
curl localhost:3000/name -H "Content-Type: application/json" -d '{"name
    ":"Gabriel"}'
```

2.2 Response

2.2.1 Responder con un 200 OK

```
server.get('/', function(req,res){
  res.status(200).send('Ok');
});
```

2.2.2 Responder con un 404 Not Found

```
server.get('/', function(req,res){
  const errorObject = {
    code: 404,
    error: "Not Found",
    message: "Error: Path not found"
  };
  res.status(404).send(errorObject);
});
```

2.3 Router

2.3.1 Usando enrutador para agrupar endpoints en la ruta `/birds`

- bird.js

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

// middleware that is specific to this router
router.use(function timeLog(req, res, next) {
  console.log('Time: ', Date.now());
  next();
});
// define the home page route
router.get('/', function(req, res) {
  res.send('Birds home page');
});
// define the about route
```

```
14 router.get('/about', function(req, res) {
15   res.send('About birds');
16 });
17
18 module.exports = router;
```

- index.js

```
1  const express = require('express');
2  const birds = require('./birds');
3
4  const server = express();
5
6  server.use('/birds', birds);
7
8  server.listen(3000, () => {
9    console.log(`Example app listening`)
10 });
```

- Ejecución para la prueba

```
1 curl localhost:3000/birds
2 curl localhost:3000/birds/about
```

3 Ejercicios

Realizar un solo servidor en el que programar todos los ejercicios, separados por sus rutas:

1. `/header` recoger he imprimir por consola un parámetro llamado `token`

- Si no está definido, devolver la respuesta con el código `401` y un objeto

```
1 {
2   "code": 401,
3   "error": "Unauthorized",
4   "message": "Error: Set a token to login"
5 }
```

2. `/params` crear un parámetro llamado `name` en la ruta y devolver `Hola ${name}`
3. `/query` enviar un número `n` y devolver la suma de todos los números desde el 1 hasta el número recibido.

- Si el número no se define, se tomará `100` por defecto.

4. `/body` Imprimir todo el objeto entrante en una lista HTML en el que se muestren todos los parámetros: su clave y su valor.

5. `/animals` Crear un enrutador bajo dicha ruta establecer los siguientes endpoint:

- `/animals/dog` devuelve un objeto `{ "grow": "guau guau"}`
- `/animals/cat` devuelve un objeto `{ "grow": "miau"}`
- `/animals/bird` devuelve un objeto `{ "grow": "pio pio"}`

6. El resto de rutas, han de devolver el código 404 con un objeto en la respuesta:

```
{
  "code": 404,
  "error": "Not Found",
  "message": "Error: Path not found"
}
```

4 Entregables

4.1 En clase

4.2 Tarea

- Todos los ejercicios