

Apuntes Node (Login, Status Codes, Encriptación JWT/md5)

- Abel Rios -

Status Code

Nuestra API tiene que devolver un código de status. El 200 es que todo ha ido bien, el 204 es que no ha encontrado datos, el 400 es una petición incorrecta, etc...

<https://developer.mozilla.org/es/docs/Web/HTTP/Status>

****OJO**** Estos ejercicios están hechos con una base de datos en MongoDB. Para SQL debería ser todo igual excepto las queries a la base de datos.

Vamos a practicar con las encriptaciones

Hay dos tipos de nivel de encriptación: reversibles (JWT) y no reversibles (md5). Las reversibles las usaremos para manejar datos menos sensibles en la conversación cliente/API. Las No Reversibles las usaremos para almacenar datos sensibles en nuestra base de datos, de esta forma nunca almacenaremos las contraseñas, sino su encriptación, evitando así (o al menos en teoría) que las contraseñas de nuestros usuarios puedan ser expuestas a hackeos.

Vamos a encriptar contraseñas con la librería nodejs-md5

<https://www.npmjs.com/package/nodejs-md5>

```
7 // 1. endpoint registrar para registrar un usuario con email y password
8 // la password debe encriptarse con MD5
9 // Si email no valido, status code 400
10 // Si todo correcto, status code 200
11 // MD5 - nodejs-md5 (npm install nodejs-md5)
12 // Node JS - Status Code response.status(200).send()
13
14 // 2. endpoint login que acepte email y password, encriptar la password con md5
15 // y comprobar email y password encriptada es la misma del registro
16 // Si email no existe, status code 404
17 // Si email o password son diferentes, status code 401
18
```

Creamos una función de verificar email (esta no es la más óptima pues es de prueba)

```
32 function validarEmail(email){
33
34     let result = false;
35
36     if(email.includes('@') && email.indexOf('@')>0 && email.indexOf('.', email.indexOf('@'))){
37         result = true;
38     }
39
40     return result;
41 }
42
```

Ahora creamos nuestro endpoint. Primero vamos a codificar la password con md5:

```
42
43 app.post("/nuevousuario", async function (request, response){
44
45     let respuesta="";
46     let database = db.db("basedeprueba");
47
48     md5.string.quiet(request.body.password, function(err, md5){
49         if (err) {
50             console.log(err);
51         }
52         else {
53             request.body.password = md5;
54         }
55     })
56
```

Y ahora verificamos el email. Si es un mail válido haremos la llamada a la base de datos y modificaremos el status del response a 200, si el email no es válido el status será 400.

```
56
57     if (validarEmail(request.body.email)){
58         await database.collection("usuarios").insertOne(request.body);
59         response.status(200).send("Email OK");
60     } else {
61         response.status(400).send("Email no valido");
62     }
63 };
64
```

Para testearlo usamos postman:

The screenshot shows the Postman interface for a POST request to `http://localhost:3001/nuevousuario`. The request body is a JSON object: `{ "email": "martin@mail.com", "password": "1234" }`. The response status is `200 OK` with a response body of `Email OK`. The interface includes tabs for Params, Authorization, Headers (9), Body, Pre-request Script, Tests, and Settings. The Body tab is selected, showing the JSON data. The response tab at the bottom shows the status and the response body.

En nuestra base de datos podemos comprobar cómo se ha almacenado (MongoDB Compass):

```
_id: ObjectId('629a44163c0857ffd3a155b8')
email: "abel@mail.com"
password: "MD5 ("1234") = 81dc9bdb52d04dc20036dbd8313ed055"

_id: ObjectId('629a4581d951a0a122114989')
email: "martin@mail.com"
password: "81dc9bdb52d04dc20036dbd8313ed055"
```

En el primer intento, a la hora de encriptar la contraseña no usé la función **.quiet**, por lo que se almacenó la contraseña y la encriptación. La idea de todo esto es que sólo se guarde el encriptado, pues de la misma palabra/contraseña **siempre va a ser el mismo encriptado** (fijarse que tanto en el usuario *abel@mail.com* como en el *martin@mail.com* la contraseña usada es la misma (1234), y también lo es el encriptado) por lo que podemos compararla para verificar que la contraseña es correcta pero no la podemos desencriptar.

(para el jwt <https://www.npmjs.com/package/jsonwebtoken>)

Ejercicio 2:

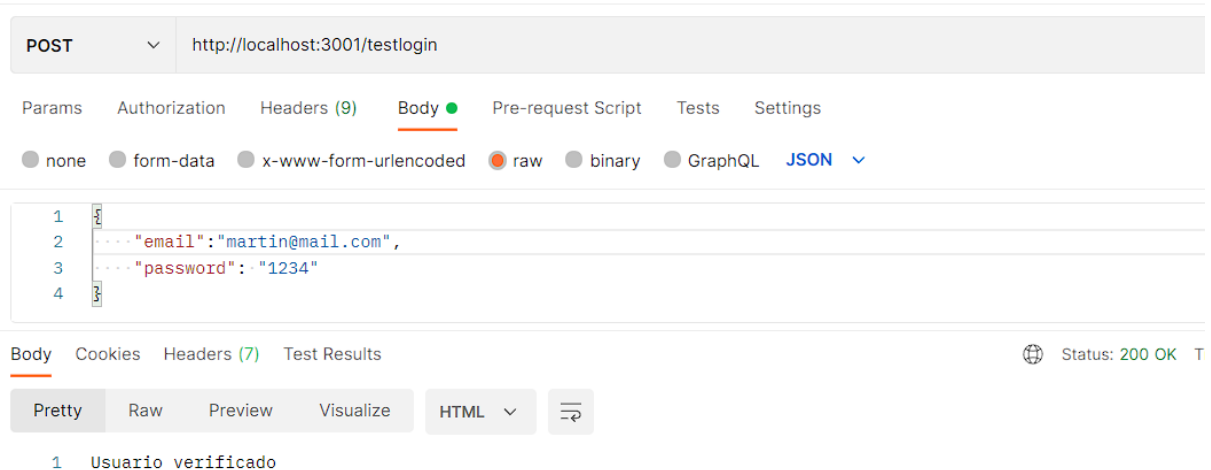
```
74 app.post("/testlogin", async function (request, response){
75
76   let database = db.db("basedeprueba");
77
78   md5.string.quiet(request.body.password, async function(err, md5){
79     if (err) {
80       console.log(err);
81     }
82     else {
83       request.body.password = md5;
84       await database.collection("usuarios").findOne({ email: { $eq: request.body.email } }, function(err,result){
85         if(!result){
86           response.status(404).send("Usuario no existe");
87         } else {
88           if(request.body.password === result.password){
89             response.status(200).send("Usuario verificado");
90           } else {
91             response.status(401).send("Password no valida.")
92           }
93         }
94       })
95     }
96   })
97 });
98
```

Comenzamos conectando a la base de datos 'basedeprueba'. Después usamos md5 para encriptar la contraseña que nos llega por el body de la request, si hubiese un error hacemos console.log del error y si no lo hay sobreescribimos la contraseña del body por el encriptado (puesto que en nuestra base de datos las contraseñas almacenadas están encriptadas).

Una vez hecho ésto buscamos en nuestra base de datos si el email de usuario existe. Si no existiese respondemos con un error 404. Si existe ya sólo nos queda comprobar si coincide la contraseña y devolver un error 401 en caso de que no coincida o un status 200 de todo Ok si ambos campos son válidos. Este endpoint nos sirve para verificar el login de nuestros usuarios.

Aquí unas capturas de Postman verificando que todo funciona en orden:

Status 200:



POST http://localhost:3001/testlogin

Params Authorization Headers (9) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

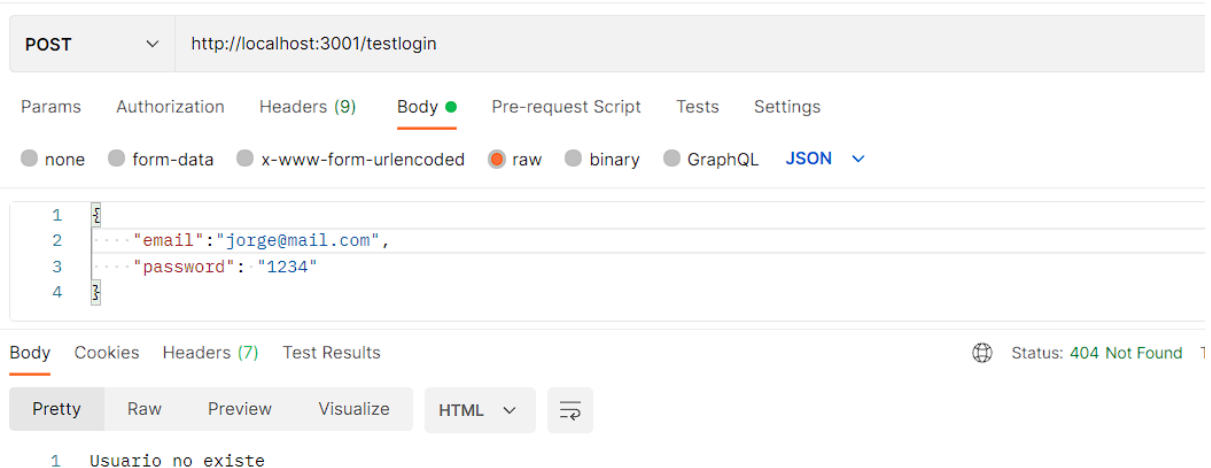
```
1 {
2   "email": "martin@mail.com",
3   "password": "1234"
4 }
```

Body Cookies Headers (7) Test Results Status: 200 OK

Pretty Raw Preview Visualize HTML

```
1 Usuario verificado
```

Status 404:



POST http://localhost:3001/testlogin

Params Authorization Headers (9) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

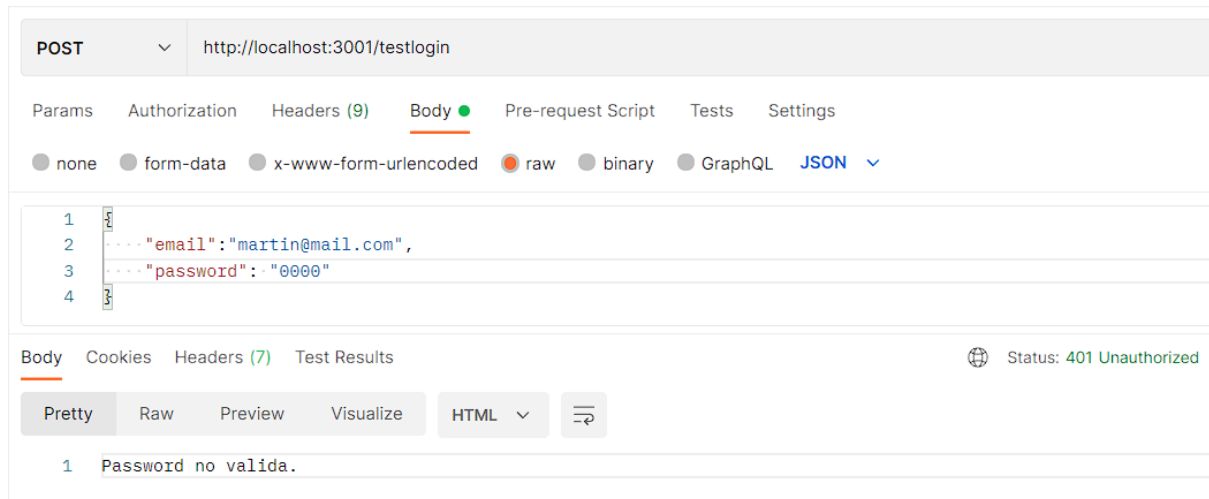
```
1 {
2   "email": "jorge@mail.com",
3   "password": "1234"
4 }
```

Body Cookies Headers (7) Test Results Status: 404 Not Found

Pretty Raw Preview Visualize HTML

```
1 Usuario no existe
```

Status 401:



Ejercicio 3:

```
2 // 3. Generar un token que contenga el id del usuario, email y tiempo de expiracion
3 // El token expira a los 60 segundos
4 // Formato del token idUsuario:email:expiracion
5 // El token se devuelve al hacer login resultado 200
6
```

```
99 app.post("/logintoken", async function (request, response){
100
101   let database = db.db("basedeprueba");
102
103   md5.string.quiet(request.body.password, async function(err, md5){
104     if (err) {
105       console.log(err);
106     }
107     else {
108       request.body.password = md5;
109       await database.collection("usuarios").findOne({ email: { $eq: request.body.email } }, function(err,result){
110         if(!result){
111           response.status(404).send("Usuario no existe");
112         } else {
113           if(request.body.password === result.password){
114             const accessToken = jwt.sign({ idObject: result.ObjectID, email:result.email}, "relevante", {expiresIn: 60});
115             response.status(200).send(accessToken);
116           } else {
117             response.status(401).send("Password no valida.")
118           }
119         }
120       })
121     }
122   })
123 });
124
```

(Podéis hacer zoom en el pdf, que la imagen está en buena calidad, o si preferís, pedidme el archivo y os lo paso)

Este endpoint es exactamente igual que el anterior, con la única diferencia de que, cuando encontramos que el usuario sí existe (está registrado) en nuestra base de datos, generamos (y devolvemos en la respuesta) un token JWT, compuesto por el id del usuario, su email, la "palabra clave" del encriptado y un tiempo de expiración del token de 60 segundos.

Ejercicio 4:

```
36
37 // 4. Crear endpoint /me que comprueba si el token no ha expirado y devuelve los
38 // datos del usuario de la base de datos excepto la password
39 // si el token ha expirado, status code 401
40
41
```

```
125 app.get("/me", (request, response) => {
126
127     const token = request.get("authorization");
128
129     jwt.verify(token, "releevant", function(err, decoded){
130         if(err){
131             response.status(401).send("Token expirado");
132         } else {
133             response.json(decoded.email);
134         }
135     });
136 })
137
```

En este endpoint lo que hacemos es sacar de la request el token que tenemos almacenado en el header 'authorization'. Una vez tenemos el token lo desencriptamos con nuestra 'palabra clave' y devolvemos en la respuesta los datos almacenados en el token (email, he

intentado sacar también el ObjectId pero no me sale, creo que el error lo tengo en el endpoint anterior a la hora de almacenar el id en el token pero habría que preguntarle a Israel). En caso de error generamos un status 401 'Token Expirado'.

Ejemplo de token expirado:

ejerciciosEncriptacion / loginchecktoken

GET http://localhost:3001/me

Params Authorization Headers (8) Body Pre-request Script Tests Settings

☒ Connection ⓘ keep-alive

☒ Authorization eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlbWFpbCI6Im...

Key	Value	Description
-----	-------	-------------

Body Cookies Headers (7) Test Results

Status: 401 Unauthorized Time:

Pretty Raw Preview Visualize HTML

1 Token expirado

Ejemplo de token recién creado:

ejerciciosEncriptacion / loginchecktoken

GET http://localhost:3001/me

Params Authorization Headers (8) Body Pre-request Script Tests Settings

☒ Connection ⓘ keep-alive

☒ Authorization eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlbWFpbCI6Im...

Key	Value	Description
-----	-------	-------------

Body Cookies Headers (7) Test Results

Status: 200 OK Time:

Pretty Raw Preview Visualize JSON

1 "martin@mail.com"

**** Solucionado el problema de no recibir el id del objeto en Mongo ****

```
99 app.post("/logintoken", async function (request, response){
100
101     let database = db.db("basedeprueba");
102
103     md5.string.quiet(request.body.password, async function(err, md5){
104         if (err) {
105             console.log(err);
106         }
107         else {
108             request.body.password = md5;
109             await database.collection("usuarios").findOne({ email: { $eq: request.body.email } }, function(err,result){
110                 if(!result){
111                     response.status(404).send("Usuario no existe");
112                 } else {
113                     if(request.body.password === result.password){
114                         const accessToken = jwt.sign({ id: result._id, email:result.email}, "releevant", {expiresIn: 120});
115                         response.status(200).send(accessToken);
116                     } else {
117                         response.status(401).send("Password no valida.")
118                     }
119                 }
120             })
121         }
122     });
123 });
```

En el token metemos el id del objeto y ahora en el siguiente endpoint lo sacamos del token:

```
125 app.get("/me", (request, response) => {
126
127     const token = request.get("authorization");
128
129     jwt.verify(token, "releevant", function(err,decoded){
130         if(err){
131             response.status(401).send("Token expirado");
132         } else {
133             response.json({email:decoded.email, id:decoded.id});
134         }
135     });
136 })
137
138
```

El problema que había es que a la función .json no se le pueden pasar varios valores, se tiene que desestructurar como un objeto. Respuesta en Postman:

GET http://localhost:3001/me

Params Authorization Headers (8) Body Pre-request Script Tests Settings

Authorization eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjYyOW...

Key Value Description

Body Cookies Headers (7) Test Results Status: 200 OK Tim

Pretty Raw Preview Visualize JSON

```
1 {
2   "email": "martin@mail.com",
3   "id": "629a4581d951a0a122114989"
4 }
```