

Registro & Login con nodejs (Jwt, bcrypt) , express, y MySQL (Backend)



Ricardo Fanizzi · [Follow](#)

5 min read · Dec 9, 2019



Share



Que usaremos para nuestro registro y login?

- Nodejs v12 librerías : moment, jwt-simple, bcrypt, dotenv, mysql, cors.
- Express v4

- MySQL v5.7
- Creamos una base de datos para el proyecto en MySQL , solo crearemos la tabla “users” con datos simples para poder autenticar y logear.

TABLAS	Campo	Tipo	Longi...	Sin signo	Rellenar ceros	Bina...	Permitir...	Clave
users	id	INT	11	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	PRI
	email	VARCHAR	30	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
	password	VARCHAR	100	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
	name	VARCHAR	20	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
	surname	VARCHAR	30	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	

Base de datos

- Creamos nuestro proyecto con NodeJS y ExpressJS.

```
express --view=pug newlogin
cd newlogin
npm install
```

Creación de proyecto express

- Conectamos nuestra aplicación a la base de datos:
 - Instalamos en node cors, mysql y dotenv.
 - Creamos en nuestro proyecto un archivo *.env*, dentro colocamos los siguientes datos: *host, user, password, port, name* “ Todos de nuestra base de datos antes creada en MySQL ”.
 - Creamos *db.js* , aqui estara la configuración de nuestra base de datos.
 - Requerimos en nuestro *app.js* a *db.js, cors* y *dotenv*.
 - Por ultimo usamos *cors* en *app.js*.

```
npm install mysql dotenv cors
```

Instalación de mysql, cors y dotenv

```
.env
1 DB_HOST=127.0.0.1
2 DB_USER=root
3 DB_PASS=root
4 DB_PORT=8889
5 DB_NAME=newlogin
```

Archivo .env

```
JS db.js > ...
1  const mysql = require('mysql');
2
3  const pool = mysql.createPool({
4    host: process.env.DB_HOST,
5    user: process.env.DB_USER,
6    password: process.env.DB_PASS,
7    port: process.env.DB_PORT,
8    database: process.env.DB_NAME
9  });
```

Configuración db.js (Base de datos)

```
JS app.js > app.use() callback
12  const cors = require('cors');
13
14  require('dotenv').config()
15  require('./db');
16
17  app.use(cors());
```

Require de cors, dotenv y db.js, uso de cors.

- Creamos una carpeta *models* y dentro el modelo *users.js*, donde estarán todas las funciones para las peticiones a la base de datos de usuarios.
- Hacemos la primera función en *users.js* que nos servirá para comprobar la conexión con la base de datos.

```
models > JS users.js > ...
1  const getAll = () => {
2    return new Promise((resolve, reject) => {
3      db.query('SELECT * FROM users', (err, rows) => {
4        if (err) reject(err)
5        resolve(rows);
6      });
7    });
8  };
9
10 module.exports = {
11   getAll: getAll
12 }
```

Función para pedir todos los usuarios a la base de datos

- Creamos un manejador de rutas para poder mostrar un JSON y usar nuestro modelo de usuario en *routes/users.js*.

```

routes > JS users.js > ...
1  const express = require('express');
2  const router = express.Router();
3  const Users = require('../models/users');
4
5  router.get('/', async (req, res) => {
6    const users = await Users.getAll();
7    res.json(users);
8  });
9
10 module.exports = router;

```

“Comprobar a este punto que la conexión es correcta, creando un usuario en mysql y mostrando un JSON en el navegador con sus datos”

- Hacemos otras dos funciones en nuestro *models/users.js* que nos servirán para registrar un usuario y para el login.

```

models > JS users.js > ...
11  /* Registro de usuarios */
12  const insert = ({ email, password, name, surname }) => {
13    return new Promise((resolve, reject) => {
14      db.query('INSERT INTO users ( email, password, name, surname) VALUES (?, ?, ?, ? )', [email, password, name, surname], (err, result) => {
15        if (err) reject(err)
16        if (result) {
17          resolve(result)
18        }
19      });
20    });
21  };
22
23  /* Obtener usuarios por su Email */
24  const getByEmail = (pEmail) => {
25    return new Promise((resolve, reject) => {
26      db.query('SELECT * FROM users WHERE email = ?', [pEmail], (err, rows) => {
27        if (err) reject(err)
28        resolve(rows[0])
29      });
30    });
31  };
32
33
34  module.exports = {
35    getAll: getAll,
36    insert: insert,
37    getByEmail: getByEmail
38  }

```

Funciones insert y getByEmail

- Usaremos *users.js* que ya debe estar en la carpeta *routes*.
- Instalamos bcrypt, jwt-simple y moment en nuestro proyecto.

```
npm install moment bcrypt jwt-simple
```

- Creamos el manejador de rutas `/register` y requerimos las librerías instaladas en `routes/users.js`.

bcrypt nos encripta el password, así que recordar la password para poder verificarla luego en el login.

```
routes > JS users.js > router.post('/login') callback
1  const express = require('express');
2  const router = express.Router();
3  const Users = require('../models/users');
4  const bcrypt = require('bcrypt');
5  const jwt = require('jwt-simple');
6  const moment = require('moment');
7
8  router.post('/register', async (req, res) => {
9    console.log(req.body);
10    req.body.password = bcrypt.hashSync(req.body.password, 10);
11    const result = await Users.insert(req.body);
12    res.json(result);
13  });
```

Manejador de ruta `/register`

id	email	password	name	surname
2	test@gmail.com	\$2b\$10\$Pzz0mKqct2FeDzN0rAPWLOol8GEDmQDy42/pNF5W7pLWgO/ZSr/lu	test	surnametest

Password (1234) encriptada en base de datos

El registro podemos probarlo con ayuda de el Postman

<https://www.getpostman.com/>

- Creamos la función para generar un *Token* en `routes/users.js`, el cual nos permitirá comprobar cuando el usuario esté logueado con ayuda del navegador.
 - Agregar al `.env` el `TOKEN_KEY="Token-Auth"`

```
routes > JS users.js > ...
41  const createToken = (user) => {
42    let payload = {
43      userId: user.id,
44      createdAt: moment().unix(),
45      expiresAt: moment().add(1, 'day').unix()
46    }
47    return jwt.encode(payload, process.env.TOKEN_KEY);
48  };
```

Función `createToken`

```

.env
1 DB_HOST=127.0.0.1
2 DB_USER=root
3 DB_PASS=root
4 DB_PORT=8889
5 DB_NAME=newlogin
6 TOKEN_KEY="Token-Auth"

```

Archivo .env

- Procedemos a crear el manejador para la ruta */login*.

```

routes > JS users.js > router.post('/login') callback
15 router.post('/login', async (req, res) => {
16   const user = await Users.getByEmail(req.body.email)
17   if (user === undefined) {
18     res.json({
19       error: 'Error, email or password not found'
20     })
21   } else {
22     const equals = bcrypt.compareSync(req.body.password, user.password);
23     if (!equals) {
24       res.json({
25         error: 'Error, email or password not found'
26       });
27     } else {
28       res.json({
29         succesfull: createToken(user),
30         done: 'Login correct'
31       });
32     }
33   }
34 });

```

El login pueden comprobarlo con ayuda del Postman nuevamente

Ya tenemos tanto el *login* que nos genera un *Token* y el *registro* que encripta nuestra *password*, para finalizar con *nodeJS* y nuestro backend, nos falta crear un *middleware*, este se encargará de verificar cada petición que le hagan a */users* y esperar el *Token* como *header*, si el *Token* existe obtendremos el *id* del usuario y de esa manera su información para que en cada ruta de nuestra aplicación tengamos activo al *Usuario logueado*.

- Creamos nuestro *middleware.js* en la carpeta *routes*.

```

routes > JS middleware.js > [⌘] checkToken
1  const jwt = require('jwt-simple');
2  const moment = require('moment');
3
4  const checkToken = (req, res, next) => {
5      if (!req.headers['user_token'])
6          return res.json({
7              error: "You must include the header"
8          });
9
10     const token = req.headers['user_token'];
11     let payload = null
12     try {
13         payload = jwt.decode(token, process.env.TOKEN_KEY)
14     } catch (err) {
15         return res.json({
16             error: 'Invalid token'
17         });
18     }
19
20     if (moment().unix() > payload.expiresAt) {
21         return res.json({ error: 'Expired token' });
22     };
23
24     req.userId = payload.userId;
25
26     next();
27 };
28
29 module.exports = {
30     checkToken: checkToken
31 }

```

middleware.js

- Usamos el *middleware* en *routes/users.js* después de nuestro manejadores de rutas */login* y */register*, si lo usáramos antes, nos pediría el Token en el header de la petición que aun no tendríamos generado ya que este Token se crea en el *login*.

```

routes > JS users.js > ...
37  router.use(middleware.checkToken);

```

- Creamos una nueva función en *models/users.js* para obtener un usuario a partir de su *id*.

```

models > JS users.js > ...
33  /* Obtener usuarios por su ID */
34  const getById = (pId) => {
35      return new Promise((resolve, reject) => {
36          db.query('SELECT * FROM users WHERE id = ?', [pId], (err, rows) => {
37              if (err) reject(err)
38              resolve(rows[0])
39          });
40      });
41  };
42
43
44  module.exports = {
45      getAll: getAll,
46      insert: insert,
47      getByEmail: getByEmail,
48      getById: getById
49  }

```

Función getById

Para finalizar nuestro *login* en el *backend* necesitaremos un manejador de ruta que utilice el *Token* que nos enviaran en el header de la petición y nos devuelva el usuario que se ha logueado, que posteriormente usaremos en nuestro frontend con *Angular* o cualquier otro framework.

- Creamos un nuevo manejador de ruta */mainUser* en *routes/users.js*, este recibirá el id del usuario en el header gracias a nuestro middleware.

```

routes > JS users.js > [⌘] createToken > [⌘] payload
37  router.use(middleware.checkToken);
38
39  router.get('/mainUser', (req, res) => {
40      Users.getById(req.userId)
41      .then(rows => {
42          res.json(rows);
43      })
44      .catch(err => console.log(err));
45  });

```

Manejador /mainUser

Con el postman puedes comprobar que con nuestro Token generado en el login, con la ruta */mainUsers*, puedes tener siempre los datos del usuario logueado, recuerda agregar el *Token* en el header del postman.

Con esto ya tienes tu login y autenticación de usuarios.

Muchas gracias por leer este post.