Bcrypt, Autenticación

JWT y Middlewares

THE BRIDGE
DIGITAL TALENT ACCELERATOR

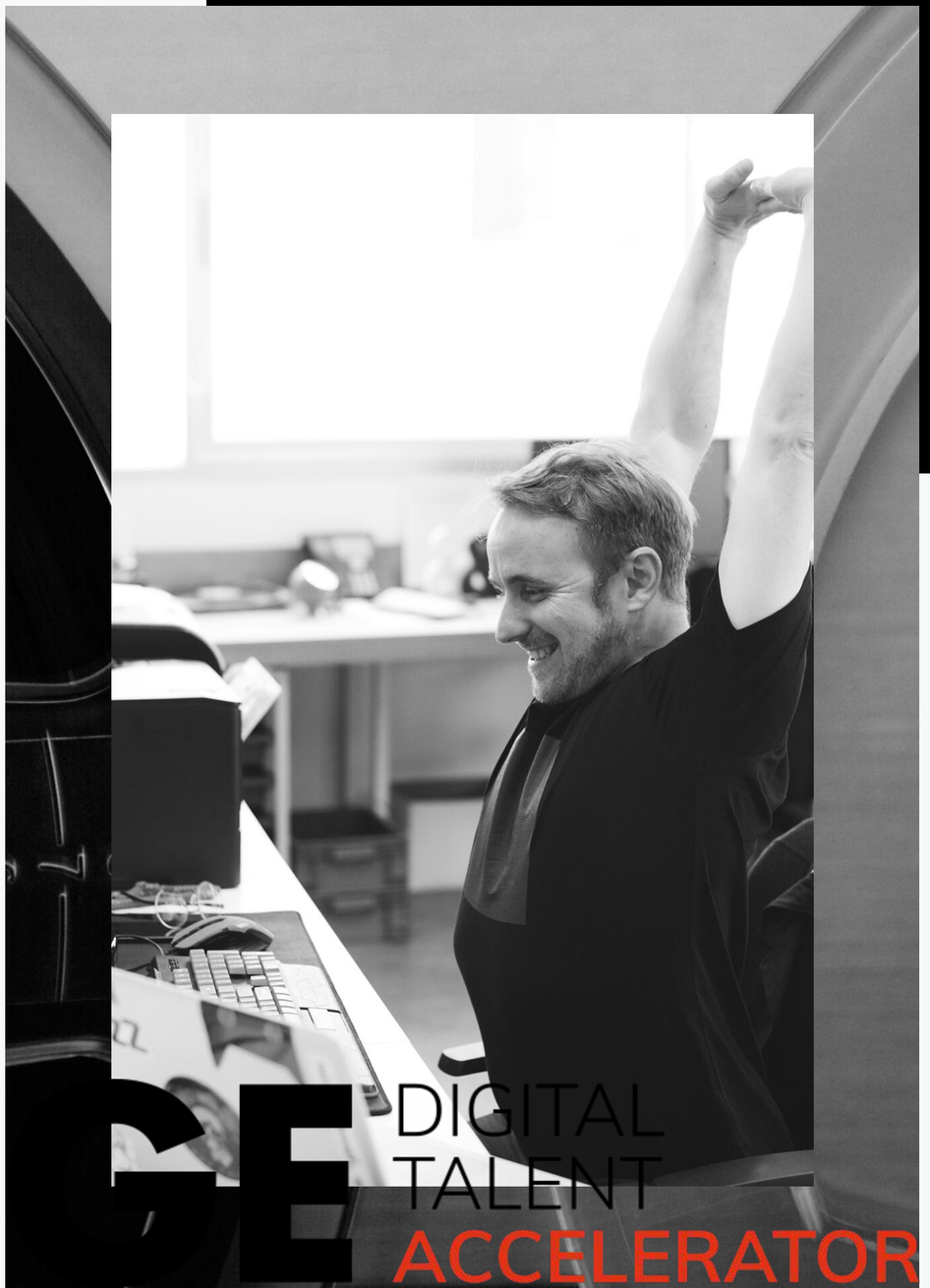
Índice

Bcrypt

Autenticación JWT

Middleware

Roles



THE BRIDGE DIGITAL TALENT ACCELERATOR

Hash

Una función criptográfica hash- usualmente conocida como “hash”- es un algoritmo matemático que **transforma cualquier bloque arbitrario de datos en una nueva serie de caracteres** con una longitud fija.

Salt

Un **salt** es una **string aleatoria.** Al **encriptar una contraseña de texto sin formato más un salt**, la salida del algoritmo hash **ya no es predecible**. La misma contraseña ya no producirá el mismo hash.

Instalación

Bcrypt

```
$ npm i bcryptjs
```

Importando el módulo bcrypt

Para usar bcrypt, debemos importarnos el módulo.

```
const { User, Post } = require('../models/index.js');
const bcrypt = require ('bcryptjs');

const UserController = {
    create(req, res) {
        req.body.role = "user";
        const password = bcrypt.hashSync(req.body.password,10)
        User.create({...req.body, password:password })
            .then(user => res.status(201).send({ message: 'Usuario creado con éxito', user }))
            .catch(err => console.error(err))
    },
. . .
```

importamos bcrypt

creamos el hash de forma síncrona

definimos el salt

Login

Creamos nuestro primer login de la siguiente forma con bcrypt:

```
const UserController = {
. . .
    login(req,res){
        User.findOne({
            where:{
                email:req.body.email
            }
        }).then(user=>{
            if(!user){
                return res.status(400).send({message:"Usuario o contraseña incorrectos"})
            }
            const isMatch = bcrypt.compareSync(req.body.password, user.password);
            if(!isMatch){
                return res.status(400).send({message:"Usuario o contraseña incorrectos"})
            }
            res.send(user)
        })
    },
. . .
```

buscamos al usuario que intenta logearse por email

comparamos la contraseña que le pasamos por el body con la que tenemos en base de datos

Cambiar una columna de una Tabla

Creamos una nueva migración para cambiar una columna de la tabla users:

```
$ sequelize migration:generate --name changeUserColumn
```

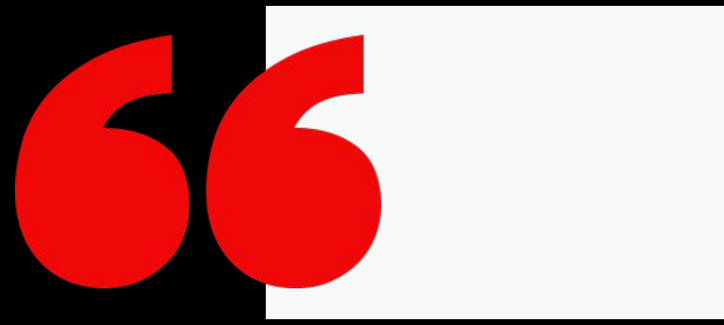
Cambiar una columna de una Tabla

Ahora en el nuevo archivo que hemos creado le diremos que el campo email de la tabla users sea de tipo string y único:

```
async up(queryInterface, Sequelize) {
    return queryInterface.changeColumn("Users", "email",  {
      type:Sequelize.STRING,
      unique:true
    });
  },
```

definimos el campo como único

JWT(JSON WEB TOKEN)



JWT

Sirve para la **creación de tokens** de acceso que permiten la propagación de identidad de un determinado usuario y privilegios.

JWT se crea con una clave secreta y esa clave secreta es privada para ti (tu servidor), lo que significa que nunca se revelará al público ni se inyectará dentro del token JWT. **Cuando recibe un JWT del cliente, puede verificar ese JWT con esta clave secreta almacenada en el servidor**.

JWT

Un JWT es simplemente una string pero contiene tres partes distintas separadas por puntos .
https://jwt.io/

HEADER: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9 (indica el algoritmo y tipo de Token, en nuestro caso: HS256 y JWT).

PAYLOAD: eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiYWRtaW4iOnRydWV9 (datos de usuario, fecha creación...).

SIGNATURE: TJVA95OrM7E2cBab30RMHrHDcEfxjoYZgeFONFh7HgQ (la firma, para verificar que el token es válido, aquí lo importante es el "secret" con el que firmamos).

La **firma se usa para verificar que el remitente del JWT es quien dice ser** y para asegurarse de que el mensaje no haya cambiado en el camino.

Instalación

JWT

```
$ npm i jsonwebtoken
```

jwt_secret

La firma para verificar que el remitente del JWT es quien dice ser, la guardaremos en nuestro config.json:

```
"development": {
   "username": "root",
   "password": "123456",
   "database": "database_sequelize",
   "host": "127.0.0.1",
   "dialect": "mysql",
   "jwt_secret": "mimamamemima"
 },
```

definimos nuestro secretito

Generando token

- Nos importamos en nuestro UserController jwt y nuestro secretito.
- Entonces generaremos el token al logearnos correctamente

```
const { User, Post } = require('../models/index.js');
const bcrypt = require ('bcryptjs');
const jwt = require('jsonwebtoken');
const { jwt_secret } = require('../config/config.json')['development']

const UserController = {
. . .
    login(req,res){
        User.findOne({
            where:{
                email:req.body.email
            }
        }).then(user=>{
            if(!user){
                return res.status(400).send({message:"Usuario o contraseña incorrectos"})
            }
            const isMatch = bcrypt.compareSync(req.body.password, user.password);
            if(!isMatch){
                return res.status(400).send({message:"Usuario o contraseña incorrectos"})
            }
            const token = jwt.sign({ id: user.id }, jwt_secret)
            res.send({ message: 'Bienvenid@ ' + user.name, user, token });
        })
. . .
    },
```

importamos jwt y nuestro secretito

generamos el token

Creando modelo Token

- Lo siguiente es crear una tabla en nuestra base de datos para ir almacenando los diferentes tokens que tenga un usuario.
- Ahora vamos a crear nuestro modelo y migración de nuestra nueva tabla tokens:

```
$ sequelize model:generate --name Token --attributes
token:string,UserId:integer

$ sequelize db:migrate
```

Token

Ahora al hacer login también le decimos que nos guarde el token en nuestra nueva tabla:

```
const { User, Post, Token } = require('../models/index.js');

. . .
login(req,res){
        User.findOne({
            where:{
                email:req.body.email
            }
        }).then(user=>{
            if(!user){
                return res.status(400).send({message:"Usuario o contraseña incorrectos"})
            }
            const isMatch = bcrypt.compareSync(req.body.password, user.password);
            if(!isMatch){
                return res.status(400).send({message:"Usuario o contraseña incorrectos"})
            }
            let token = jwt.sign({ id: user.id }, jwt_secret);
            Token.create({ token, UserId: user.id });
            res.send({ message: 'Bienvenid@' + user.name, user, token });
        })
. . .
```

nos importamos el modelo Token

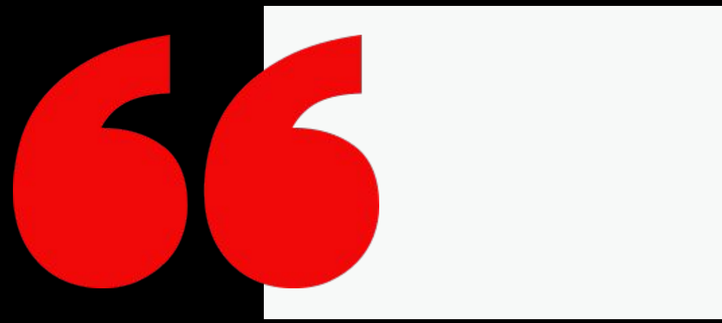
guardamos el token en la tabla tokens

Añadimos la ruta de login

Vamos a nuestro archivo de rutas de usuario y añadimos la nueva ruta:

```
router.post('/login',UserController.login)
```

Middleware



Middleware

Un middleware es una **función que se puede ejecutar antes o después del manejo de una ruta**. Esta función tiene acceso al objeto Request, Response y la función next().

Las funciones middleware suelen ser utilizadas como mecanismo para verificar niveles de acceso antes de entrar en una ruta, manejo de errores, validación de datos, etc.

Authentication middleware

Tenemos definida una ruta a la cual solo usuarios logeados pueden entrar, por lo tanto, **se necesita comprobar antes de entrar a esa ruta, si el usuario está o no logeado.**

Nuestro primer Middleware

Creamos una carpeta middleware y dentro de ella un archivo que se llame **authentication.js** que contendrá el siguiente código:

```
const { User, Token, Sequelize } = require('../models');
const { Op } = Sequelize;
const jwt = require('jsonwebtoken');
const  {jwt_secret}  = require('../config/config.json')['development']

const authentication = async(req, res, next) => {
    try {
        const token = req.headers.authorization;
        const payload = jwt.verify(token, jwt_secret);
        const user = await User.findByPk(payload.id);
        const tokenFound = await Token.findOne({
            where: {
                [Op.and]: [
                    { UserId: user.id },
                    { token: token }
                ]
            }
        });
        if (!tokenFound) {
            return res.status(401).send({ message: 'No estas autorizado' });
        }
        req.user = user;
        next();
    } catch (error) {
        console.log(error)
        res.status(500).send({ error, message: 'Ha habido un problema con el token' })
    }
}
module.exports = { authentication }
```

Implementando el middleware

Vamos a nuestro archivo de rutas de usuario y nos importamos el middleware que hemos creado y lo implementamos en aquellas rutas que queremos que solo se acceda si estas logeado:

```
const express = require('express');
const router = express.Router();
const UserController = require('../controllers/UserController')
const {authentication} = require('../middleware/authentication')

router.post('/',UserController.create)
router.get('/',authentication,UserController.getAll)
router.delete('/:id',authentication,UserController.delete)
router.put('/:id',authentication,UserController.update)
router.post('/login',UserController.login)

module.exports = router;
```

importamos el middleware de autenticación

implementamos el middleware

isAdmin middleware

Tenemos definida una ruta a la cual solo usuarios administradores pueden ingresar, por lo tanto, **se necesita comprobar antes de entrar a esa ruta, si el usuario es o no, un administrador.**

isAdmin Middleware

Ahora crearemos un middleware que chequeara el rol del usuario para saber si es admin:

Ruta:

***middleware/authentication.js**

```
. . .
const isAdmin = async(req, res, next) => {
    const admins = ['admin','superadmin'];
    if (!admins.includes(req.user.role)) {
        return res.status(403).send({
            message: 'No tienes permisos'
        });
    }
    next();
}

module.exports = { authentication, isAdmin }
```

Implementando el middleware isAdmin

Vamos a nuestro archivo de rutas de posts y nos importamos el middlewar de autenticacion y el de isAdmin y lo implementamos en aquellos endpoints que queremos que solo se acceda si estas logeado y tienes el rol de admin :

```
const express = require('express');
const router = express.Router();
const PostController = require('../controllers/PostController')
const {authentication, isAdmin} = require('../middleware/authentication')

router.get('/', PostController.getAll)
router.get('/:id', PostController.getById)
router.get('/title/:title', PostController.getOneByName)
router.post('/',authentication, PostController.create)
router.delete('/:id',authentication, isAdmin, PostController.delete)
router.put('/:id',authentication, PostController.update)

module.exports = router;
```

importamos el middleware de autenticación y el de isAdmin

implementamos el middleware

Logout

Para crear nuestro logout le decimos que nos destruya el Token donde el UserId sea igual que el que pasamos y el token igual al que le pasamos por headers:

```
const { User, Post, Token, Sequelize } = require('../models/index.js');
const bcrypt = require ('bcryptjs');
const jwt = require('jsonwebtoken');
const  {jwt_secret}  = require('../config/config.json')['development']
const { Op} = Sequelize;

const UserController = {
. . .
       async logout(req, res) {
        try {
            await Token.destroy({
                where: {
                    [Op.and]: [
                        { UserId: req.user.id },
                        { token: req.headers.authorization }
                    ]
                }
            });
            res.send({ message: 'Desconectado con éxito' })
        } catch (error) {
            console.log(error)
            res.status(500).send({ message: 'hubo un problema al tratar de desconectarte' })
        }
    }
}

module.exports = UserController
```

Implementando el logout

Vamos a nuestro archivo de rutas de usuario e implementamos la ruta para el logout pasando el middleware de autenticación:

```
const express = require('express');
const router = express.Router();
const UserController = require('../controllers/UserController')
const {authentication} = require('../middleware/authentication')

router.post('/',UserController.create)
router.get('/',authentication,UserController.getAll)
router.delete('/id/:id',authentication,UserController.delete)
router.put('/:id',authentication,UserController.update)
router.post('/login',UserController.login)
router.delete('/logout',authentication,UserController.logout)

module.exports = router;
```

añadimos la ruta junto con el middleware