Momento 2:

middleware de autenticación:

ya construidos los modelos, controladores y rutas de nuestro proyecto el siguiente paso es restringir el acceso a cierta información del sistema para determinados usuarios para este caso ,solo los usuarios que estén logueados podrán ingresar al panel de información del sistema.

lo configuraremos y desarrollaremos en 3 pasos :

Paso 1:

Modificar el controlador UsuarioController agregando una nueva función llamada login , la lógica del login ya la hemos construido en la semana 3 .

```
login: async(req, res, next) => {
      try {
         let user = await models.Usuario.findOne({ email: req.body.email, estado: 1 });
         if (user) {
            let match = await bcrypt.compare(req.body.password, user.password);
            if (match) {
               let tokenReturn = await token.encode(user._id);
               res.status(200).json({ user, tokenReturn });
            } else {
               res.status(404).send({
                  message: 'Password Incorrecto'
               });
            }
         } else {
            res.status(404).send({
               message: 'No existe el usuario'
            });
         }
      } catch (e) {
         res.status(500).send({
            message: 'Ocurrió un error'
         });
         next(e);
      }
   }
```

le hicimos algunas mejoras para de buenas prácticas , como lo es

- try y catch.
- separamos la generación de token en otro archivo para así manejar solo la lógica de login en esta función y sea más fácil de entender por diferentes programadores.

- para crear la función de generar el token creamos una carpeta en la raíz de nuestro proyecto llamada services y creamos un archivo llamado token.js y le agregamos el siguiente fragmento de código:

sevices/token.js

```
var jwt = require('jsonwebtoken');
const models = require('../models');

async function checkToken(token) {
    let __id = null;
    try {
        const { _id } = await jwt.decode(token);
        __id = _id;
    } catch (e) {
        return false;
    }
    const user = await models.Usuario.findOne({ where: { id: __id, estado: 1 } });
    if (user) {
        const token = jwt.sign({ _id: __id }, 'secretKeyToGenerateToken', { expiresIn: '1d' });
        return { token, rol: user.rol };
    } else {
        return false;
    }
}

module.exports = {

    //generar el token
    encode: async(_id, rol) => {
        const token = jwt.sign({ _id: _id, rol: rol }, 'secretKeyToGenerateToken', { expiresIn: '1d' });
        return token;
    },
    //permite decodificar el token
    decode: async(token) => {
        try {
            const { _id } = await jwt.verify(token, 'secretKeyToGenerateToken');
            const user = await models.Usuario.findOne({ where: { _id, estado: 1 } });
            if (user) {
                return user;
            } else {
                return false;
            }
        } catch (e) {
            const newToken = await checkToken(token);
            return newToken;
        }
```

```
    }
}
```

checktoken: de checa si es un token de usuario autenticado sigue siendo válido , si el token expiró la función genera uno nuevo
Encode:genera un token para la autenticación.
Decode:decodifica el token.

no nos olvidemos importar la carpeta services en el controlador UsuarioController.js:

```
const token = require('../services/token');
```

Paso 2:

creamos una carpeta en la raíz del proyecto llamada middlewares y creamos un archivo llamado auth.js

auth.js

```
const tokenService = require('../services/token');

module.exports = {
    verifyUsuario: async(req, res, next) => {
        if (!req.headers.token) {
            return res.status(404).send({
                message: 'No token'
            });
        }
        const response = await tokenService.decode(req.headers.token);
        if (response.rol == 'Administrador' || response.rol == 'Vendedor' || response.rol == 'Almacenero') {
            next();
        } else {
            return res.status(403).send({
                message: 'No autorizado'
            });
        }
    },

}
```

lo primero que hacemos es determinar si en la propiedad head (Objeto que contiene encabezados HTTP para servir con el archivo),existe un objeto llamado token , si el token no existe devolvemos un estatus 404 con un mensaje de no token , si el token existe, pasamos a decodificar el token para extraer la información encriptada del rol del usuario y determinar si el usuario tiene rol para ingresar al sistema , si el usuario cuenta con un rol lo

dejamos pasar con la función next(), de lo contrario retornamos un estatus 403 de no autorización.

Paso 3: Uso del middleware creado en el paso 2

para restringir el acceso a las rutas no ubicamos en la carpeta routes y en el archivo articulo.js

sin middleware:

```
const routerx = require('express-promise-router');
const articuloController =
require('../controllers/ArticuloController');
const auth = require('../middlewares/auth');

const router = routerx();

router.post('/add', articuloController.add);
router.get('/query', articuloController.query);
router.get('/queryCodigo', articuloController.queryCodigo);
router.get('/list', articuloController.list);
router.put('/update',articuloController.update);
router.delete('/remove', articuloController.remove);
router.put('/activate',articuloController.activate);
router.put('/deactivate', articuloController.deactivate);

module.exports = router;
```

lo que queremos es restringir estas rutas a solo usuario autenticados y que contenga un rol en el sistema :
lo primero importamos nuestro middleware:

```
const auth = require('../middlewares/auth');
```

luego a cada una de las rutas le pasamos como segundo parámetro el middleware auth:

```
const routerx = require('express-promise-router');
const articuloController = require('../controllers/ArticuloController');
const auth = require('../middlewares/auth');

const router = routerx();

router.post('/add', auth.verifyUsuario, articuloController.add);
router.get('/query', auth.verifyUsuario, articuloController.query);
router.get('/queryCodigo', auth.verifyUsuario, articuloController.queryCodigo);
router.get('/list', articuloController.list);
router.put('/update', auth.verifyUsuario, articuloController.update);
router.delete('/remove', auth.verifyUsuario, articuloController.remove);
router.put('/activate', auth.verifyUsuario, articuloController.activate);
```

```
router.put('/deactivate', auth.verifyUsuario, articuloController.deactivate);

module.exports = router;
```