**Proyecto login y autenticación**

Índice

[Tecnologías usadas 2](#_Toc125542329)

[Criterios a considerar 2](#_Toc125542330)

[Autenticación 2](#_Toc125542331)

[Autorización 3](#_Toc125542332)

[Robo de base de datos 3](#_Toc125542333)

[Carpeta proyecto 3](#_Toc125542334)

[Creación 3](#_Toc125542335)

[Acceder 3](#_Toc125542336)

[Instalación de dependencias 3](#_Toc125542337)

[package.js 3](#_Toc125542338)

[Línea de código 4](#_Toc125542339)

[Index.js 4](#_Toc125542340)

[Importar dependencias 4](#_Toc125542341)

[Conexión a la base de datos 4](#_Toc125542342)

[Ejecución de express 4](#_Toc125542343)

[Modelo de usuario 5](#_Toc125542344)

[Registro de usuario 5](#_Toc125542345)

[Importar modelo de usuario 5](#_Toc125542346)

[Primer endpoint 5](#_Toc125542347)

[Puerto 6](#_Toc125542348)

[Evidencia de funcionamiento 7](#_Toc125542349)

[JsonWebToken 8](#_Toc125542350)

[Encriptar id 8](#_Toc125542351)

[Inicio de sesión 8](#_Toc125542352)

[Segundo endpoint 9](#_Toc125542353)

[Middlewares validación de JWT 10](#_Toc125542354)

[Middleware de buscar y asignar usuario 10](#_Toc125542355)

[Unión Middlewares 10](#_Toc125542356)

[Ruta protegida 10](#_Toc125542357)

[Tercer endpoint 10](#_Toc125542358)

# Tecnologías usadas

Para llevar a cabo estas dos funcionalidades se implementarán varias tecnologías enfocadas a la productividad y a la web las cuales son:

* Node js
* JavaScript
* Express
* Html
* Css

# Criterios a considerar

## Autenticación

Diagrama

Descripción generada automáticamente

## Autorización

Diagrama

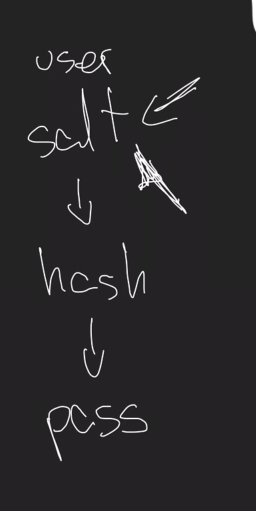
Descripción generada automáticamente

## Robo de base de datos

En caso de robo de base de datos

Encriptar constraseña usando hash

Encriptar con salt, se usara el salt para aplicarlo en el hash



JSON web token es una llave para que un usuario pueda acceder a ciertos elementos

# Carpeta proyecto

## Creación

Creación de la carpeta donde se va a alojar el proyecto



## Acceder

Se accede a el



# Instalación de dependencias

## package.js

Se instala un archivo package.js

 le damos ok a todos



## Línea de código



Se instala express como framework del backend,

mongoose para la conexión con la base de datos

bcrypt para encriptar las contraseñas

jsonwebtoken permite crear json web tokens y gestionarlos

express-jwt permite utilizar jsonwebtoken para una implementación más productiva

# Index.js

Dentro de la carpeta del proyecto se creará un nuevo archivo con el nombre index.js



## Importar dependencias

*//importar dependecias*

const express= require('express');

const mongoose= require('mongoose');

const bcrypt= require('bcrypt');

const jwt =require('jsonwebtoken');

const { expressjwt: expressJwt } = require('express-jwt');

## Conexión a la base de datos

*//conexión a la base de datos*

mongoose.connect('mongodb+srv://miltonXD:Atlas1234@cluster0.venxvyj.mongodb.net/autenticacion?retryWrites=true&w=majority')

## Ejecución de express

*//aplicacion*

const app= express();

*//recivir datos en formato json*

app.use(express.json())

# Modelo de usuario

Se crea un nuevo archivo para hay trabajar en el esquema del modelo de la base de datos



Dentro del archivo

*// importar dependecia de db*

const mongoose= require('mongoose')

*//modelo*

const Usuario=mongoose.model('Usuario',{

*//esquema*

*//son los atributos que tendrá un usuario*

    email:{type:String, require:true},

    password:{type:String, require:true},

    salt:{type:String, require:true},

})

*//exportar el modelo*

module.exports=Usuario;

# Registro de usuario

## Importar modelo de usuario

Se debe importar el modelo de usuario que se creo en el archivo user.js en el archivo index.js



## Primer endpoint

*//primer endpoint*

*//regristro si exite no lo crea*

app.post('/registro', async (*req*,*res*)=>{

*//se saca el body de la req (correo contraseña)*

    const{body}= *req*

    console.log({body})

*//control de errores*

*//trata de hacer*

*try*{

*//realizar la búsqueda*

*//primero buscará en si existe algún usuario con el correo*

*//el cual se encuentra en body*

        const esUsuario=*await* Usuario.findOne({email:body.email})

*//si existe el usuario*

*if*(esUsuario){

*//mensaje*

*return* *res*.status(403).send('el usuario ya existe')

        }

*//no es necesario agregar else*

*//si no existe el usuario se creará un salt*

*//usando la dependecia bcrypt*

*//con el método genSalt()*

        const salt=*await* bcrypt.genSalt()

*//contrasena encriptada*

*//usando la dependecia bcrypt*

*//con el metodo hash*

*//se envia la contraseña*

*//y luego se envia el salt*

        const hashed= *await* bcrypt.hash(body.password,salt)

*//crear el usuario*

*//se envía el email*

*//la contraseña encriptada*

*//y el salt*

        const usuario= *await* Usuario.create({email:body.email, password:hashed,salt})

*//muestra el id de usuario*

*res*.send({\_id:usuario.\_id})

    }*catch*(err){*//si ocurre un error*

        console.log(err)

*//mensaje para el cliente*

*res*.status(500).send(err.message)

    }

})

## Puerto

Puerto por el cual escucha la aplicación

*//puerto que escucha*

app.listen(3000,()=>{

    console.log('Escuchando por el puerto 3000')

})

## Evidencia de funcionamiento

En la terminal

Texto

Descripción generada automáticamente

En postman

Configuración

Captura de pantalla de un celular

Descripción generada automáticamente

Resultado postman

Captura de pantalla de un teléfono celular

Descripción generada automáticamente



# JsonWebToken

## Encriptar id

Primero se crea la función que se encargará de firmar el jsonwebtoken

*//funcion de firma de jsonWebtoken*

*//recive un \_id*

*//encripta el parametro con el valor siguiente*

const firmaToken=*\_id* =>jwt.sign({\_id},'secreto')

Esta porción de Código va después de la creación de usuario, es parte del primer endpoint

*//encriptar el id*

*//se guarda en una constante el jsonWebToken firmado*

*//se llama a la funcion*

*//se envia el parametro*

            const firmado= firmaToken(usuario.\_id)

*//muestra el id de usuario*

*//envia un jsonWeb token*

*res*.status(201).send(firmado)

# Inicio de sesión

## Segundo endpoint

Se crea el segundo endpoint para el inicio de sesión   
*//inicio de sesión*

*//Segundo endpoint*

app.post('/acceso', async (*req*,*res*)=>{

*//sacar el cuerpo de la petición*

*//se enviara el usuario y la contraseña en post*

    const {body} =*req*

*//control de excepciones*

*try*{

*//guardar usuario*

*//se buscar por le valor de email*

*// este se encontrar en body*

        const usuario=*await* Usuario.findOne({email:body.email})

*//si no existe usuario*

*if*(!usuario){

*//mensaje de error*

*res*.status(403).send('usuario o contraseña invalidas')

        }*else*{

*//compara la contraseña que existe en el usuario*

*//con la que existe en el post*

*//se usa bcrypt*

*//compara la contraseña sin enciptar con la encriptada*

            const seEmpareja=*await* bcrypt.compare(body.password,usuario.password)

*//condición*

*if* (seEmpareja){*//si es que coinciden*

*//se firma el jsonwebtoken*

*//se envia el id*

                const firmado=firmaToken(usuario.\_id)

*//se envía un estatus y el jwt firmado*

*res*.status(200).send(firmado)

            }*else*{

*//si no*

*//envía un estatus 403*

*//y un mensaje de error*

*res*.status(403).send('Usuario y/o contraseña no encontrados ')

            }

        }

    }*catch*(err){

*//capturar el error*

*res*.status(500).send(err.message)

    }

})

# Middlewares validación de JWT

*//Middleware de validacion*

*//función de validacion JsonWebToken*

*//se usa la dependecia expressJWt*

*//se envia un objeto con el string secreto*

*//algoritmo que encripta*

const validacionJwt = expressJwt({ secret: 'secreto', algorithms: ['HS256'] })

# Middleware de buscar y asignar usuario

*//funcion de buscar y asignar usuario*

const buscarAsignarUsuario = async (*req*, *res*, *next*) => {

*//*

*try* {

*//va a buscar en la base de datos al usuario*

        const usuario = *await* Usuario.findById(*req*.auth.\_id)

*if* (!usuario) {*//si no existe*

*return* *res*.status(401).end()

        }

*//si lo encuentra*

*//asigna a la propieda usuario*

*req*.usuario = usuario

        next()

    } *catch* (e) {

*//si existe un error*

        next(e)

    }

}

# Unión Middlewares

Mediante una función de express se pueden unir varios middlewares y convertirlo en uno

*//unir middlewares*

*//se usa dependecia espress, el método Router y el metodo use*

*// se ingresan los middlewares que se desean unir*

const estaAutenticado=express.Router().use(validacionJwt,buscarAsignarUsuario)

# Ruta protegida

## Tercer endpoint

*//endpoint ruta protegida*

*//se ingresa la ruta 'lele'*

*//se envia el middleware unido*

*//al final la lógica del endpoint*

app.get('/lele', estaAutenticado, (*req*, *res*) => {

*//manejo del endpoint*

*//devuleve el usuario completo (todos los atributos )*

*res*.send(*req*.usuario)

})

# Variables de entorno ocultas

En este caso se ocultará el string secreto de encriptación

En la terminal



const validacionJwt = expressJwt({ secret: process.env.secret, algorithms: ['HS256'] })

const firmaToken = *\_id* => jwt.sign({ \_id }, process.env.secret)