GUIA DE AUTENTICACION CON JWT

1. Creamos la base de datos en PostgreSQL

2. Crear un nuevo proyecto

```
npm init adonis-ts-app@latest .
```

3. Instalación de ORM

```
npm i @adonisjs/lucid@alpha
node ace invoke @adonisjs/lucid
```

4. Actualizar el archivo .env

5. Creación de migraciones

```
node ace make:migration usuarios
```

6. Modificar las migraciones

```
public async up () {
  this.schema.createTable(this.tableName, (table) => {
    table.increments('id').primary()
    table.string('nombres', 180).notNullable()
    table.string('correo', 200).notNullable().unique()
    table.string('contrasena', 255).notNullable()
    table.timestamps(true)
```

7. Correr las migraciones creadas

```
node ace migration:run
```

8. Crear los modelos

```
node ace make:model Usuario
```

9. Crear los controladores

```
node ace make:controller Usuario
```

10. Instalar el paquete jwt
    npm i jsonwebtoken

11. Instalamos bcrypt.js

```
npm install bcryptjs
```

12. Modificamos el modelo

```
export default class Usuario extends BaseModel {
  @column({ isPrimary: true }) public id: number
  @column() public nombres: string
  @column() public correo: string
  @column() public contrasena: string
  @column.dateTime({ autoCreate: true }) public createdAt: DateTime
  @column.dateTime({ autoCreate: true, autoUpdate: true })
  public updatedAt: DateTime
}
```

13. Modificamos el controlador

```
import type { HttpContextContract } from '@ioc:Adonis/Core/HttpContext'
import Usuario from 'App/Models/Usuario'
import jwt from 'jsonwebtoken'
import Env from '@ioc:Adonis/Core/Env'
const bcryptjs = require('bcryptjs')

export default class UsuariosController {

  public async registrar({request}: HttpContextContract){
    const {nombres, correo, contrasena} = request.all();
    const salt = bcryptjs.genSaltSync();
    const usuario = new Usuario();
    usuario.nombres = nombres;
    usuario.correo = correo;
    usuario.contrasena = bcryptjs.hashSync( contrasena, salt );;
    await usuario.save();
    return{usuario, "msg": "Usuario registrado"}
  }
```

```
public async login({request, response}: HttpContextContract){
  const correo = request.input('correo');
  const contrasena = request.input('contrasena');
  try {
    //consultar si existe usuario con ese correo
    const user = await Usuario.findBy('correo', correo)
    if(!user){
      return response.status(400).json({msj: 'El usuario no existe'})
    }

    const validPassword = bcryptjs.compareSync( contrasena, user.contrasena );
    if ( !validPassword ) {
      return response.status(400).json({msj: 'Los datos de acceso no son correctos'})
    }
    //Validar si la contraseña ingresada es igual a la del usaurio
    const payload ={
      'nombres': user.nombres,
      'id': user.id
    }
    const token:string = this.generarToken(payload);

    response.status(200).json({
      token,
      "msg": "Usuario logueado"})
  } catch (error) {
    response.json({"msg": "Credenciales invalidas"});
  }
}
```

```
public generarToken(payload: any):string{
  const opciones = {
    expiresIn: "5 mins"
  }
  return jwt.sign(payload, Env.get('JWT_SECRET_KEY'), opciones)
}

public verificarToken(authorizationHeader:string){
  let token = authorizationHeader.split(' ')[1]
  console.log(token)
  token = jwt.verify(token, Env.get('JWT_SECRET_KEY'), (error)=>{
      if(error){
          throw new Error("Token expirado");

      }
  })
  return true
```

14. Instalar Middleware: node ace make:middleware AuthJwt
15. Modificar Middleware

```
import type { HttpContextContract } from '@ioc:Adonis/Core/HttpContext'
import UsuariosController from 'App/Controllers/Http/UsuariosController'

export default class Authjwt {
  public async handle(ctx: HttpContextContract, next: () => Promise<void>) {
    const authorizationHeader = ctx.request.header('authorization')

    if(authorizationHeader == undefined){
      return ctx.response.status(400).send({
        mensaje: "Falta el token de autorización",
        estado: 401,
      })
    }

    //const cuerpoPeticion = {
      const token = authorizationHeader
    //}

    try{
      const usuariosController = new UsuariosController()
      usuariosController.verificarToken(token)
      await next()
    }catch(error){
      ctx.response.status(400).send("Falla en token")
    }

  }
}
```

16. Modificar en Kernel

```
Server.middleware.registerNamed({
  auth: () => import('App/Middleware/Authjwt')
})
```

17. Crear rutas:

```
import Route from '@ioc:Adonis/Core/Route'

Route.get('/', async () => {
  return { hello: 'world' }
}).middleware("auth")

Route.group(() =>{
  Route.post("/register", "UsuariosController.registrar");
  Route.post("/login", "UsuariosController.login");
}).prefix("api")
```

18. Realizar pruebas en Postman