

LABORATORIO N° 07

Desarrollo de aplicaciones Web Avanzado: Seguridad en aplicaciones con JWT.



DOCENTE:

Coello Palomino, Ricardo

CURSO:

Desarrollo de aplicaciones Web
avanzado

5 - C24 - Sección A -B-C-D

DESARROLLO DE APLICACIONES WEB AVANZADO: Seguridad en aplicaciones con JWT.

I. Capacidades

- Implementa aplicaciones usando un stack fullstack y JWT.

II. Seguridad

- En este laboratorio está prohibida la manipulación del hardware, conexiones eléctricas o de red. Así como la ingesta de alimentos y bebidas.
- Ubicar maletines y/o mochilas en lugar destinado para tal fin.
- Dejar la mesa de trabajo y la silla utilizada limpias y ordenadas.

III. Fundamento teórico

- Revise el material de la semana correspondiente antes del desarrollo del laboratorio.

IV. Normas empleadas

- *No aplica*

V. Recursos

- En este laboratorio, cada estudiante trabajará con una computadora con Windows 10.
- La instalación del software requerido se realizará en el equipo virtual.

VI. Metodología para el desarrollo de la tarea

- El desarrollo del laboratorio es individual

VII. Procedimiento

NODE.JS EXPRESS JWT AUTHENTICATION WITH MYSQL & ROLES

Desarrollaremos una aplicación Node.js Express que permita a los usuarios:

Registrarse: Crear una nueva cuenta.

Iniciar sesión: autentiqúese usando nombre de usuario y contraseña.

Acceso basado en roles: acceda a los recursos en función de roles (administrador, moderador, usuario).

TECNOLOGIA

Node.js : entorno de ejecución.

Express 4 : Marco web.

Sequelize 6 : ORM para MySQL.

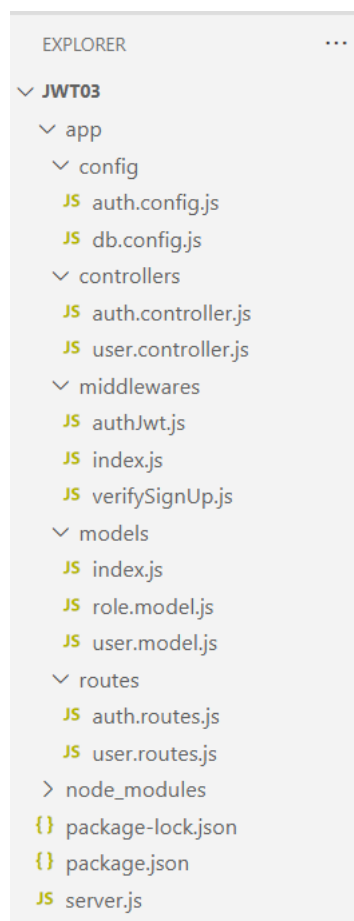
MySQL 8 : Base de datos relacional.

JWT 9 : Autenticación basada en token.

bcryptjs 2 : Hashing de contraseñas.

CORS 2 : Intercambio de recursos entre orígenes.

1.- Abrir el programa Visual Studio y crear la siguiente estructura de carpetas:



2.- Crear paquete JSON: **npm init -y**

3.- Instalar dependencias: **npm install express sequelize mysql2 cors jsonwebtoken bcryptjs**

4.- Configurar package.json:

```
"scripts": {  
  "start": "node server.js"  
}
```

5.- Actualización package.json de los módulos ES

```
{  
  ...  
  "type": "module",  
  ...  
}
```

CONFIGURACIÓN DE LA BASE DE DATOS

7.- Crear archivo de configuración

```
JS db.config.js X  
app > config > JS db.config.js > [icon] default > [icon] PORT  
1 // app/config/db.config.js  
2 export default {  
3   HOST: "localhost",  
4   USER: "root",  
5   PASSWORD: "",  
6   DB: "db",  
7   PORT: 3306,  
8   dialect: "mysql",  
9   pool: {  
10    max: 5,  
11    min: 0,  
12    acquire: 30000,  
13    idle: 10000,  
14  },  
15 };
```

```
JS auth.config.js X  
app > config > JS auth.config.js > ...  
1 // app/config/auth.config.js  
2 export default {  
3   secret: "your-secret-key",  
4 };  
5
```

8.- Inicializar Sequelize y definir asociaciones de modelos:

```

EXPLORER
  JWT03
    app
      config
        JS auth.config.js
        JS db.config.js
      controllers
        JS auth.controller.js
        JS user.controller.js
      middlewares
        JS auth.jwt.js
        JS index.js
        JS verifySignUp.js
      models
        JS index.js
        JS role.model.js
        JS user.model.js
      routes
        JS auth.routes.js
        JS user.routes.js
      node_modules
      package-lock.json
      package.json
      server.js

JS index.js
1 // Importamos Sequelize, que es el ORM que utilizaremos para interactuar
2 // con la base de datos
3 import Sequelize from "sequelize";
4
5 // Importamos la configuración de la base de datos desde un archivo
6 // externo
7 import dbConfig from "../config/db.config.js";
8
9 // Importamos los modelos de usuario y rol
10 import userModel from "../user.model.js";
11 import roleModel from "../role.model.js";
12
13 // Creamos una instancia de Sequelize con los parámetros de configuración
14 const sequelize = new Sequelize(dbConfig.DB, dbConfig.USER, dbConfig.
15   PASSWORD, {
16     host: dbConfig.HOST, // Dirección del servidor de la base
17     // de datos
18     dialect: dbConfig.dialect, // Tipo de base de datos (por
19     // ejemplo, 'mysql', 'postgres')
20     pool: dbConfig.pool, // Configuración del pool de
21     // conexiones
22     port: dbConfig.PORT, // Puerto en el que se conecta a la
23     // base de datos
24   });
25
26 // Creamos un objeto para almacenar los modelos y la instancia de
27 // Sequelize
28 const db = {};
29
30 // Asignamos Sequelize y la instancia sequelize al objeto db
31 db.Sequelize = Sequelize;
32 db.sequelize = sequelize;
33
34 // Inicializamos los modelos de usuario y rol, pasándoles la instancia de
35 // Sequelize y el objeto Sequelize
36 db.user = userModel(sequelize, Sequelize);
37 db.role = roleModel(sequelize, Sequelize);
38
39 // Definimos una relación de muchos a muchos entre roles y usuarios
40 db.role.belongsToMany(db.user, {
41   through: "user_roles", // Nombre de la tabla intermedia que
42   // almacena las relaciones
43   foreignKey: "roleId", // Clave foránea en la tabla intermedia que
44   // referencia a roles
45   otherKey: "userId", // Clave foránea en la tabla intermedia que
46   // referencia a usuarios
47 });
48
49 // Definimos la relación inversa de muchos a muchos entre usuarios y roles
50 db.user.belongsToMany(db.role, {
51   through: "user_roles", // Nombre de la tabla intermedia
52   // foreignKey: "userId", // Clave foránea que referencia a usuarios
53   // otherKey: "roleId", // Clave foránea que referencia a roles
54   // as: "roles", // Alias para acceder a los roles de un
55   // usuario
56 });
57
58 // Definimos una constante con los posibles roles que se pueden asignar
59 db.ROLES = ["user", "admin", "moderator"];
60
61 // Exportamos el objeto db para que pueda ser utilizado en otras partes de
62 // la aplicación
63 export default db;

```

9.- Definir modelos.

```

EXPLORER
  JWT03
    app
      config
        JS auth.config.js
        JS db.config.js
      controllers
        JS auth.controller.js
        JS user.controller.js
      middlewares
        JS auth.jwt.js
        JS index.js
        JS verifySignUp.js
      models
        JS index.js
        JS role.model.js
        JS user.model.js
      routes
        JS auth.routes.js
        JS user.routes.js
      node_modules
      package-lock.json
      package.json
      server.js

JS user.model.js
1 // app/models/user.model.js
2 export default (sequelize, Sequelize) => {
3   const User = sequelize.define("users", {
4     username: {
5       type: Sequelize.STRING,
6       unique: true,
7     },
8     email: {
9       type: Sequelize.STRING,
10      unique: true,
11    },
12    password: {
13      type: Sequelize.STRING,
14    },
15  });
16
17   return User;
18 };

JS role.model.js
1 // app/models/role.model.js
2 export default (sequelize, Sequelize) => {
3   const Role = sequelize.define("roles", {
4     id: {
5       type: Sequelize.INTEGER,
6       primaryKey: true,
7       autoIncrement: true,
8     },
9     name: {
10      type: Sequelize.STRING,
11    },
12  });
13
14   return Role;
15 };

```

IMPLEMENTACIÓN DE FUNCIONES DE MIDDLEWARE

10.- Verificar el middleware de registro.

Comprobamos si hay nombres de usuario o correos electrónicos duplicados y valida roles.

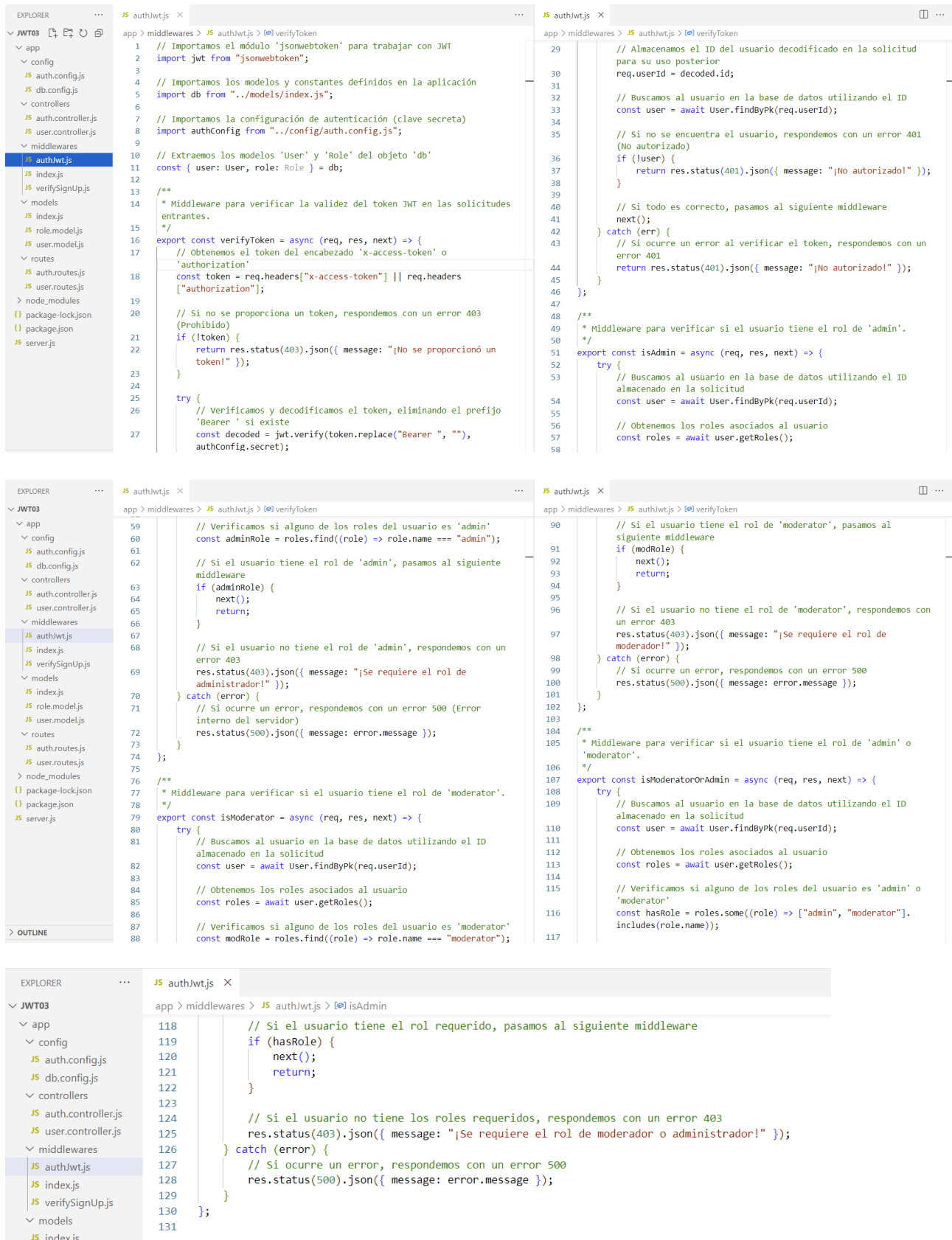
```

EXPLORER
  JWT03
    app
      config
        JS auth.config.js
        JS db.config.js
      controllers
        JS auth.controller.js
        JS user.controller.js
      middlewares
        JS auth.jwt.js
        JS index.js
        JS verifySignUp.js
      models
        JS index.js
        JS role.model.js
        JS user.model.js
      routes
        JS auth.routes.js
        JS user.routes.js
      node_modules
      package-lock.json
      package.json
      server.js

JS verifySignUp.js
1 // Importamos el objeto 'db' que contiene los modelos y constantes
2 // definidos en la aplicación
3 import db from "../models/index.js";
4
5 // Extraemos la constante ROLES y el modelo User del objeto db
6 const { ROLES, user: User } = db;
7
8 /**
9  * Middleware para verificar si el nombre de usuario o el correo
10  * electrónico ya están en uso.
11  */
12 export const checkDuplicateUsernameOrEmail = async (req, res, next) => {
13   try {
14     // Buscamos un usuario con el mismo nombre de usuario
15     // proporcionado en la solicitud
16     const userByUsername = await User.findOne({
17       where: { username: req.body.username },
18     });
19
20     // Si se encuentra un usuario con ese nombre de usuario,
21     // respondemos con un error
22     if (userByUsername) {
23       return res.status(400).json({ message: "¡El nombre de usuario
24       ya está en uso!" });
25     }
26
27     // Buscamos un usuario con el mismo correo electrónico
28     // proporcionado en la solicitud
29     const userByEmail = await User.findOne({
30       where: { email: req.body.email },
31     });
32
33     // Si se encuentra un usuario con ese correo electrónico,
34     // respondemos con un error
35     if (userByEmail) {
36       return res.status(400).json({ message: "¡El correo
37       electrónico ya está en uso!" });
38     }
39
40     // Si no se encontraron duplicados, pasamos al siguiente
41     // middleware
42     next();
43   } catch (error) {
44     // En caso de error en la base de datos u otro, respondemos con
45     // un error del servidor
46     res.status(500).json({ message: error.message });
47   }
48 };
49
50 /**
51  * Middleware para verificar si los roles proporcionados existen en la
52  * lista de roles permitidos.
53  */
54 export const checkRolesExisted = (req, res, next) => {
55   // Verificamos si se proporcionaron roles en la solicitud
56   if (req.body.roles) {
57     // Iteramos sobre cada rol proporcionado
58     for (const role of req.body.roles) {
59       // Si el rol no existe en la lista de roles permitidos,
60       // respondemos con un error
61       if (!ROLES.includes(role)) {
62         return res.status(400).json({ message: `¡El rol ${role}
63         no existe!` });
64       }
65     }
66   }
67
68   // Si todos los roles son válidos o no se proporcionaron roles,
69   // pasamos al siguiente middleware
70   next();
71 };

```

Verifica tokens y verifica los roles de los usuarios.



12.- Exportación de middleware

```

EXPLORER    ...    JS index.js  X
└─ JWT03
  └─ app
    └─ config
      └─ JS auth.config.js
      └─ JS db.config.js
    └─ controllers
      └─ JS auth.controller.js
      └─ JS user.controller.js
    └─ middlewares
      └─ JS authJwt.js
      └─ JS index.js
      └─ JS verifySignUp.js

app > middlewares > JS index.js
1  // Importa todo lo exportado desde 'authJwt.js' como un objeto llamado 'authJwt'
2  // Esto incluye funciones como verifyToken, isAdmin, isModerator, etc., si están exportadas desde ese archivo
3  import * as authJwt from "../authJwt.js";
4
5  // Importa directamente las funciones 'checkDuplicateUsernameOrEmail' y 'checkRolesExisted'
6  // desde el archivo 'verifySignUp.js'. Estas funciones probablemente validan datos del usuario durante el registro.
7  import { checkDuplicateUsernameOrEmail, checkRolesExisted } from "../verifySignUp.js";
8
9  // Reexporta los middlewares importados para que puedan ser accedidos fácilmente desde otros archivos
10 // Por ejemplo, puedes hacer: `import { authJwt, checkDuplicateUsernameOrEmail } from "../middlewares/index.js"`
11 export { authJwt, checkDuplicateUsernameOrEmail, checkRolesExisted };
12

```

CREACIÓN DE CONTROLADORES

13.- Controlador de autenticación.

Maneja el registro e inicio de sesión del usuario.

```

EXPLORER    ...    JS auth.controller.js
└─ JWT03
  └─ app
    └─ config
      └─ JS auth.config.js
      └─ JS db.config.js
    └─ controllers
      └─ JS auth.controller.js
      └─ JS user.controller.js
    └─ middlewares
      └─ JS authJwt.js
      └─ JS index.js
      └─ JS verifySignUp.js
    └─ models
      └─ JS index.js
      └─ JS role.model.js
      └─ JS user.model.js
    └─ routes
      └─ JS auth.routes.js
      └─ JS user.routes.js
  > node_modules
  > package-lock.json
  > package.json
  > server.js

app > controllers > JS auth.controller.js > signup
1  // Importa el objeto de modelos (User, Role, etc.) desde la carpeta models
2  import db from "../models/index.js";
3
4  // Importa la librería jsonwebtoken para generar tokens JWT
5  import jwt from "jsonwebtoken";
6
7  // Importa bcryptjs para encriptar y comparar contraseñas
8  import bcrypt from "bcryptjs";
9
10 // Importa la configuración del secreto JWT desde un archivo de configuración
11 import authConfig from "../config/auth.config.js";
12
13 // Extrae los modelos User y Role desde el objeto db
14 const { user: User, role: Role } = db;
15
16 // Controlador para el registro de usuarios
17 export const signup = async (req, res) => {
18   try {
19     // Extrae los datos enviados en el cuerpo de la solicitud
20     const { username, email, password, roles } = req.body;
21
22     // Encripta la contraseña antes de guardarla en la base de datos
23     const hashedPassword = await bcrypt.hash(password, 8);
24
25     // Busca el rol "user" en la base de datos para asignarlo por defecto
26     const userRole = await Role.findOne({ where: { name: "user" } });
27
28     // Crea un nuevo usuario con los datos proporcionados y la contraseña encriptada
29     const user = await User.create({
30       username,
31       email,
32       password: hashedPassword,
33     });
34
35     // Asocia el rol encontrado al usuario (relación muchos a muchos)
36     await user.setRoles([userRole]);
37
38     // Devuelve respuesta exitosa
39     res.status(201).json({ message: "User registered successfully!" });
40   } catch (error) {
41     // Si ocurre un error, responde con código 500 y el mensaje del error
42     res.status(500).json({ message: error.message });
43   }
44 }
45
46 // Controlador para el inicio de sesión
47 export const signin = async (req, res) => {
48   try {
49     const { username, password } = req.body;
50
51     // Busca el usuario por su nombre de usuario, incluyendo sus roles
52     const user = await User.findOne({
53       where: { username },
54       include: { model: Role, as: "roles" },
55     });
56
57     // Si no se encuentra el usuario, responde con error 404
58     if (!user) {
59       return res.status(404).json({ message: "User Not found." });
60     }
61
62     // Compara la contraseña proporcionada con la almacenada (ya encriptada)
63     const passwordIsValid = await bcrypt.compare(password, user.password);
64
65     // Si la contraseña no es válida, responde con error 401
66

```

```

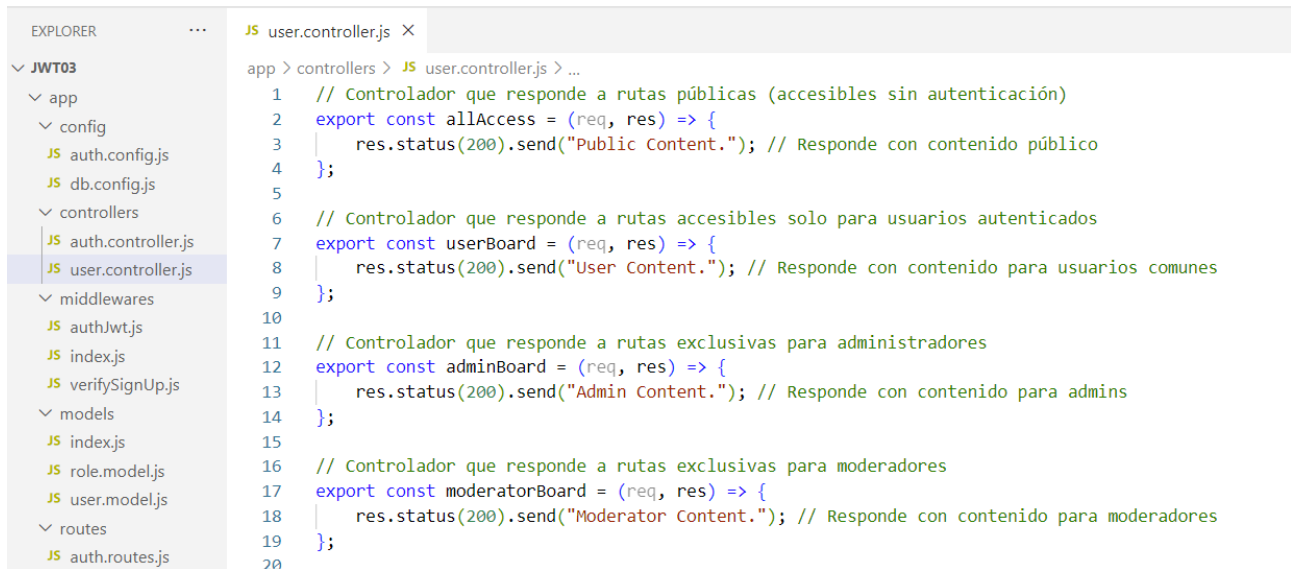
EXPLORER    ...    JS auth.controller.js
└─ JWT03
  └─ app
    └─ config
      └─ JS auth.config.js
      └─ JS db.config.js
    └─ controllers
      └─ JS auth.controller.js
      └─ JS user.controller.js
    └─ middlewares
      └─ JS authJwt.js
      └─ JS index.js
      └─ JS verifySignUp.js
    └─ models
      └─ JS index.js
      └─ JS role.model.js
      └─ JS user.model.js
    └─ routes
      └─ JS auth.routes.js
      └─ JS user.routes.js
  > node_modules
  > package-lock.json
  > package.json
  > server.js

app > controllers > JS auth.controller.js > signup
65 // Si la contraseña no es válida, responde con error 401
66 if (!passwordIsValid) {
67   return res.status(401).json({
68     accessToken: null,
69     message: "Invalid Password!",
70   });
71 }
72
73 // Si la contraseña es válida, genera un token JWT que expira en 24 horas
74 const token = jwt.sign({ id: user.id }, authConfig.secret, {
75   expiresIn: 86400, // 24 horas
76 });
77
78 // Crea un array con los roles del usuario en el formato 'ROLE_ADMIN', 'ROLE_USER', etc.
79 const authorities = user.roles.map((role) => `ROLE_${role.name.toUpperCase()}`);
80
81 // Responde con la información del usuario y el token de acceso
82 res.status(200).json({
83   id: user.id,
84   username: user.username,
85   email: user.email,
86   roles: authorities,
87   accessToken: token,
88 });
89 } catch (error) {
90   // Si ocurre un error en el proceso, responde con código 500 y el mensaje del error
91   res.status(500).json({ message: error.message });
92 }
93
94

```


14.- Controlador del usuario

Maneja el acceso a recursos protegidos.



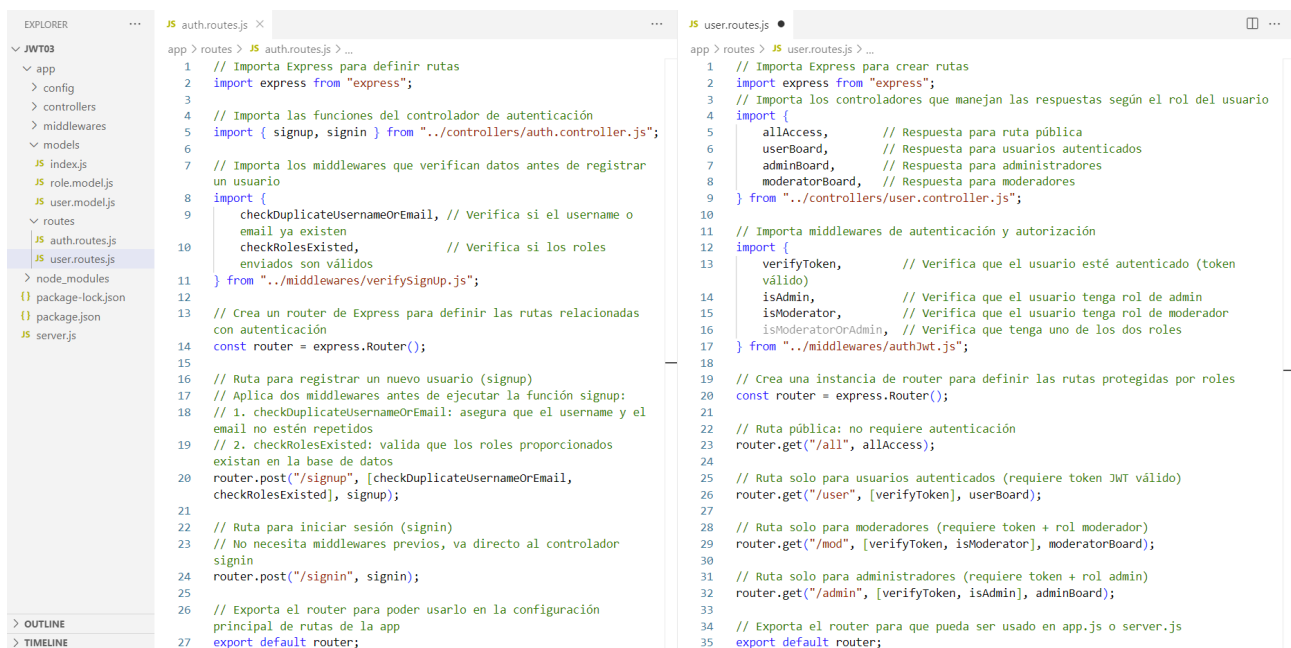
```

1 // Controlador que responde a rutas públicas (accesibles sin autenticación)
2 export const allAccess = (req, res) => {
3   res.status(200).send("Public Content."); // Responde con contenido público
4 };
5
6 // Controlador que responde a rutas accesibles solo para usuarios autenticados
7 export const userBoard = (req, res) => {
8   res.status(200).send("User Content."); // Responde con contenido para usuarios comunes
9 };
10
11 // Controlador que responde a rutas exclusivas para administradores
12 export const adminBoard = (req, res) => {
13   res.status(200).send("Admin Content."); // Responde con contenido para admins
14 };
15
16 // Controlador que responde a rutas exclusivas para moderadores
17 export const moderatorBoard = (req, res) => {
18   res.status(200).send("Moderator Content."); // Responde con contenido para moderadores
19 };
20

```

DEFINICIÓN DE RUTAS

15.- Rutas de autenticación y Rutas de usuario



```

1 // Importa Express para definir rutas
2 import express from "express";
3
4 // Importa las funciones del controlador de autenticación
5 import { signup, signin } from "../controllers/auth.controller.js";
6
7 // Importa los middlewares que verifican datos antes de registrar un usuario
8 import {
9   checkDuplicateUsernameOrEmail, // Verifica si el username o email ya existen
10  checkRolesExisted,             // Verifica si los roles enviados son válidos
11 } from "../middlewares/verifySignup.js";
12
13 // Crea un router de Express para definir las rutas relacionadas con autenticación
14 const router = express.Router();
15
16 // Ruta para registrar un nuevo usuario (signup)
17 // Aplica dos middlewares antes de ejecutar la función signup:
18 // 1. checkDuplicateUsernameOrEmail: asegura que el username y el email no estén repetidos
19 // 2. checkRolesExisted: valida que los roles proporcionados existan en la base de datos
20 router.post("/signup", [checkDuplicateUsernameOrEmail, checkRolesExisted], signup);
21
22 // Ruta para iniciar sesión (signin)
23 // No necesita middlewares previos, va directo al controlador signin
24 router.post("/signin", signin);
25
26 // Exporta el router para poder usarlo en la configuración principal de rutas de la app
27 export default router;

```

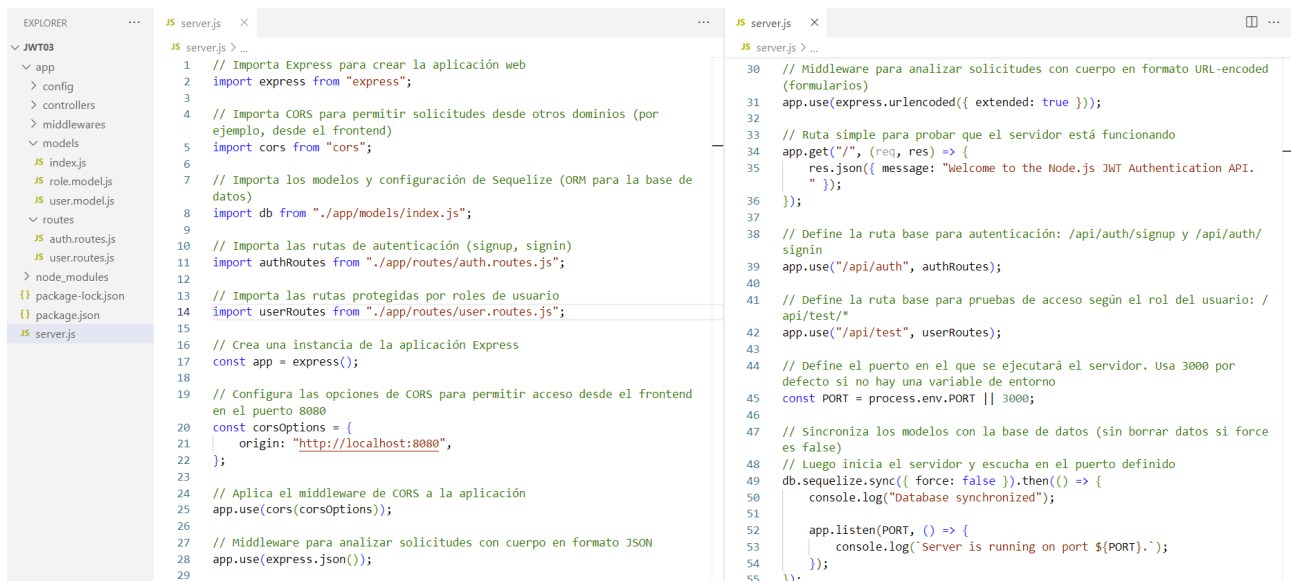
```

1 // Importa Express para crear rutas
2 import express from "express";
3 // Importa los controladores que manejan las respuestas según el rol del usuario
4 import {
5   allAccess,           // Respuesta para ruta pública
6   userBoard,           // Respuesta para usuarios autenticados
7   adminBoard,          // Respuesta para administradores
8   moderatorBoard,     // Respuesta para moderadores
9 } from "../controllers/user.controller.js";
10
11 // Importa middlewares de autenticación y autorización
12 import {
13   verifyToken,         // Verifica que el usuario esté autenticado (token válido)
14   isAdmin,             // Verifica que el usuario tenga rol de admin
15   isModerator,         // Verifica que el usuario tenga rol de moderador
16   isModeratorOrAdmin,  // Verifica que tenga uno de los dos roles
17 } from "../middlewares/authJwt.js";
18
19 // Crea una instancia de router para definir las rutas protegidas por roles
20 const router = express.Router();
21
22 // Ruta pública: no requiere autenticación
23 router.get("/all", allAccess);
24
25 // Ruta solo para usuarios autenticados (requiere token JWT válido)
26 router.get("/user", [verifyToken, userBoard]);
27
28 // Ruta solo para moderadores (requiere token + rol moderador)
29 router.get("/mod", [verifyToken, isModerator, moderatorBoard]);
30
31 // Ruta solo para administradores (requiere token + rol admin)
32 router.get("/admin", [verifyToken, isAdmin, adminBoard]);
33
34 // Exporta el router para que pueda ser usado en app.js o server.js
35 export default router;

```


16.- Crear server.js

Usando la sintaxis ESMODULE , configure el servidor Express :



```

1 // Importa Express para crear la aplicación web
2 import express from "express";
3
4 // Importa CORS para permitir solicitudes desde otros dominios (por
5 // ejemplo, desde el frontend)
6 import cors from "cors";
7
8 // Importa los modelos y configuración de Sequelize (ORM para la base de
9 // datos)
10 import db from "./app/models/index.js";
11
12 // Importa las rutas de autenticación (signup, signin)
13 import authRoutes from "./app/routes/auth.routes.js";
14
15 // Importa las rutas protegidas por roles de usuario
16 import userRoutes from "./app/routes/user.routes.js";
17
18 // Crea una instancia de la aplicación Express
19 const app = express();
20
21 // Configura las opciones de CORS para permitir acceso desde el frontend
22 // en el puerto 8080
23 const corsOptions = {
24   origin: "http://localhost:8080",
25 };
26
27 // Aplica el middleware de CORS a la aplicación
28 app.use(cors(corsOptions));
29
30 // Middleware para analizar solicitudes con cuerpo en formato JSON
31 app.use(express.json());
32
33 // Ruta simple para probar que el servidor está funcionando
34 app.get("/", (req, res) => {
35   res.json({ message: "Welcome to the Node.js JWT Authentication API." });
36 });
37
38 // Define la ruta base para autenticación: /api/auth/signup y /api/auth/signin
39 app.use("/api/auth", authRoutes);
40
41 // Define la ruta base para pruebas de acceso según el rol del usuario: /api/test/*
42 app.use("/api/test", userRoutes);
43
44 // Define el puerto en el que se ejecutará el servidor. Usa 3000 por
45 // defecto si no hay una variable de entorno
46 const PORT = process.env.PORT || 3000;
47
48 // Sincroniza los modelos con la base de datos (sin borrar datos si force
49 // es false)
50 db.sequelize.sync({ force: false }).then(() => {
51   console.log("Database synchronized");
52 });
53
54 app.listen(PORT, () => {
55   console.log("Server is running on port ${PORT}.");
56 });

```

17.- Ejecutar la aplicación: **npm start**

18.- Insertar registro en la tabla roles.

INSERT INTO roles VALUES (1, 'user', now(), now());

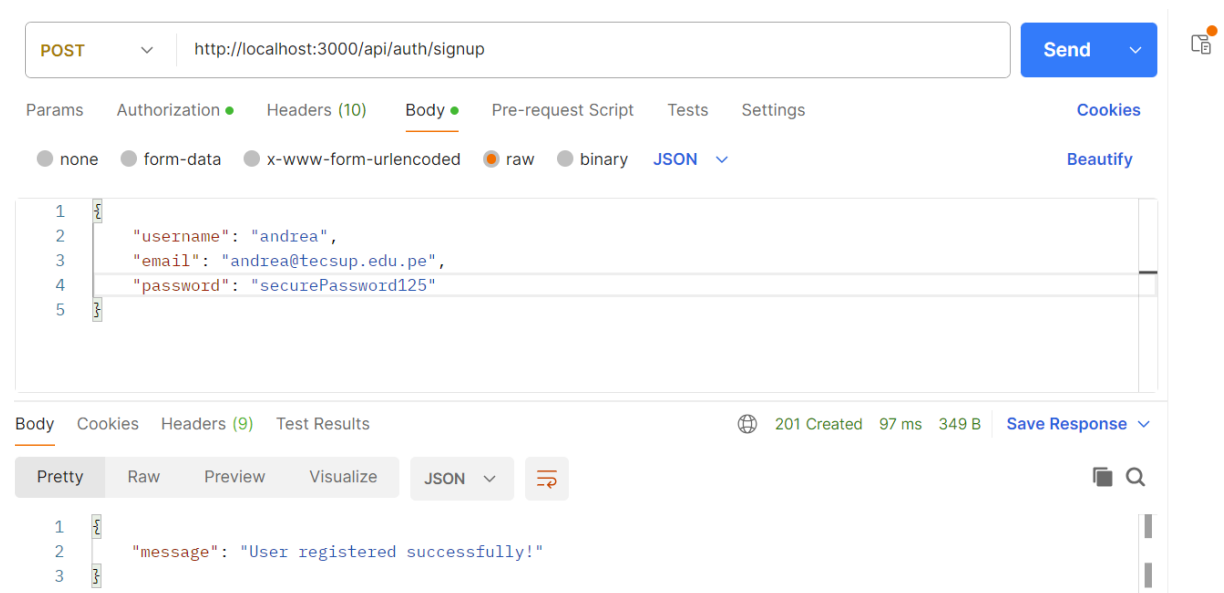
INSERT INTO roles VALUES (2, 'moderator', now(), now());

INSERT INTO roles VALUES (3, 'admin', now(), now());

PRUEBAS CON POSTMAN

19.- Ejecute Postman

20.- Registrar un usuario



POST http://localhost:3000/api/auth/signup

Params Authorization Headers (10) Body Pre-request Script Tests Settings Cookies Beautify

none form-data x-www-form-urlencoded raw binary JSON

```

1 {
2   "username": "andrea",
3   "email": "andrea@tecsup.edu.pe",
4   "password": "securePassword125"
5 }

```

Body Cookies Headers (9) Test Results 201 Created 97 ms 349 B Save Response

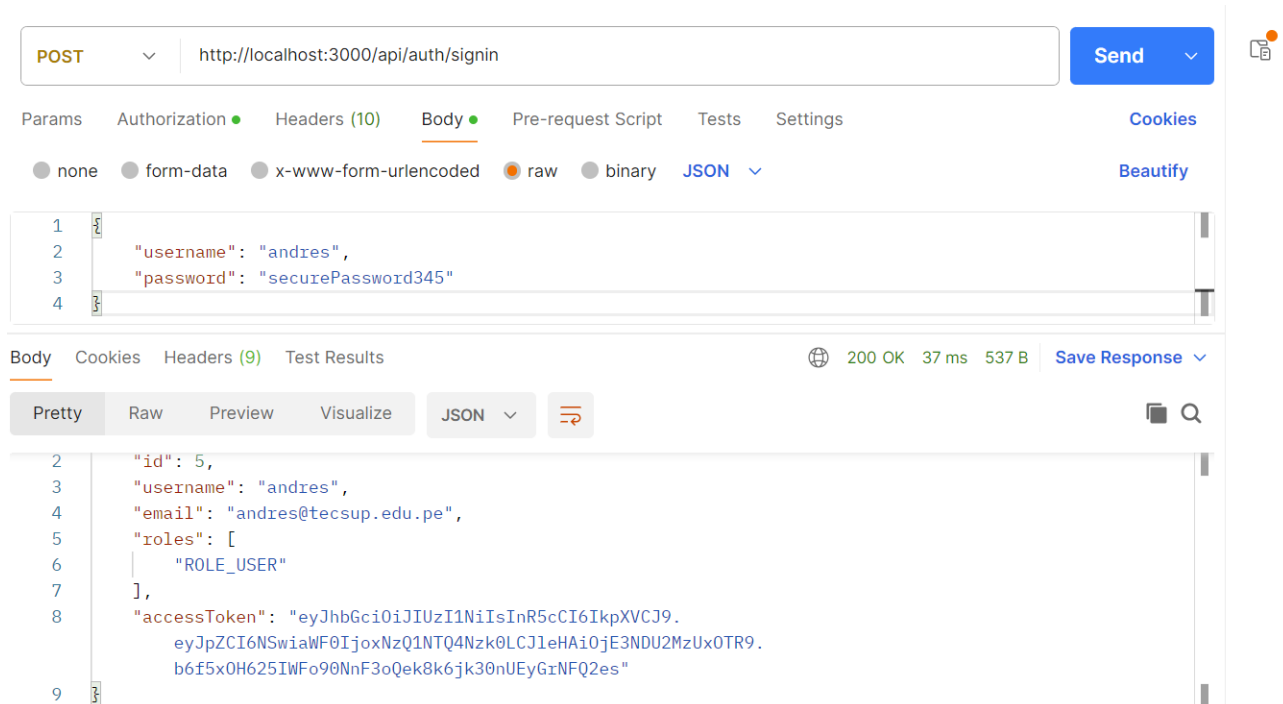
Pretty Raw Preview Visualize JSON

```

1 {
2   "message": "User registered successfully!"
3 }

```

21.- Iniciar sesión como usuario



POST ▼ http://localhost:3000/api/auth/signin Send ▼

Params Authorization Headers (10) **Body** Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded **raw** binary JSON ▼ Beautify

```

1
2  "username": "andres",
3  "password": "securePassword345"
4

```

Body Cookies Headers (9) Test Results 200 OK 37 ms 537 B Save Response ▼

Pretty Raw Preview Visualize JSON ▼ 🔍

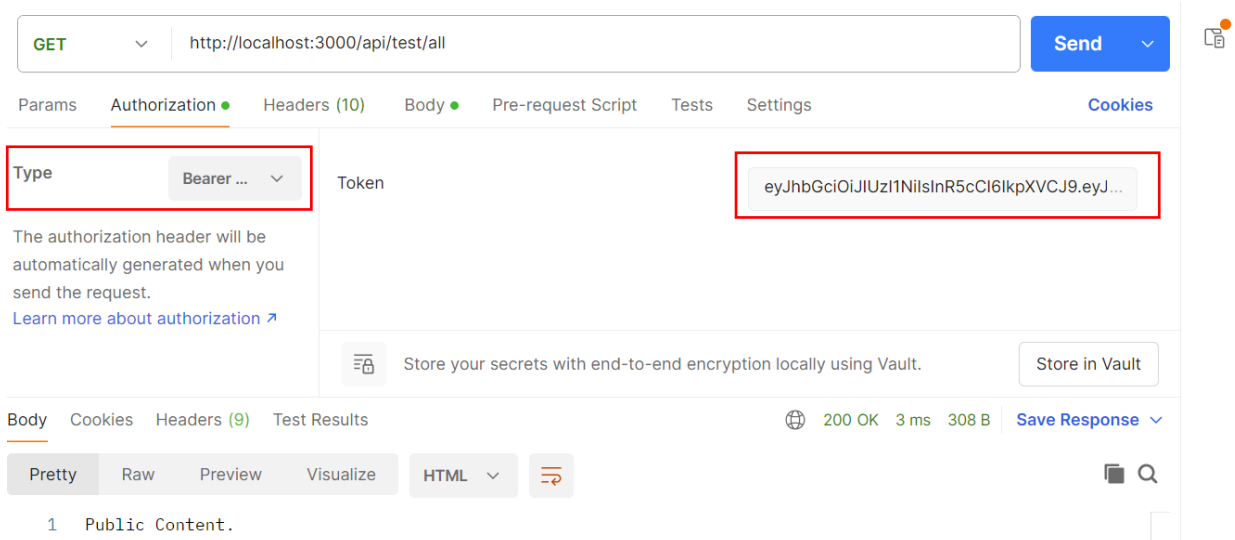
```

2  "id": 5,
3  "username": "andres",
4  "email": "andres@tecsup.edu.pe",
5  "roles": [
6    "ROLE_USER"
7  ],
8  "accessToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6NSwiaWF0IjoxNzQ1NTQ4Nzk0LCJleHAiOjE3NDU2MzUxOTR9.b6f5x0H625IWFo90NnF3oQek8k6jk30nUEyGzNFQ2es"
9

```

22.- Acceso a rutas protegidas.

Utilice lo recibido accessToken en el Authorization encabezado:



GET ▼ http://localhost:3000/api/test/all Send ▼

Params Authorization Headers (10) **Body** Pre-request Script Tests Settings Cookies

Type Bearer ... ▼ Token eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6NSwiaWF0IjoxNzQ1NTQ4Nzk0LCJleHAiOjE3NDU2MzUxOTR9.b6f5x0H625IWFo90NnF3oQek8k6jk30nUEyGzNFQ2es

The authorization header will be automatically generated when you send the request.
[Learn more about authorization](#) ↗

🔒 Store your secrets with end-to-end encryption locally using Vault. Store in Vault

Body Cookies Headers (9) Test Results 200 OK 3 ms 308 B Save Response ▼

Pretty Raw Preview Visualize HTML ▼ 🔍

```

1 Public Content.

```

Puntos finales:

- GET /api/test/all- Público
- GET /api/test/user- Usuario, Moderador, Administrador
- GET /api/test/mod- Moderador
- GET /api/test/admin- Administrador

Ejercicios de aplicación

Construir una aplicación React en la que:

- Hay páginas de inicio/cierre de sesión y registro.
- Los datos del formulario serán validados por el front-end antes de enviarse al back-end.
- Dependiendo de los roles del usuario (administrador, moderador, usuario), la barra de navegación cambia sus elementos automáticamente.

Presentar capturas de pantalla de la ejecución y código de su proyecto.

Conclusiones:

Indicar 5 conclusiones que llegó después de los temas tratados de manera práctica en este laboratorio.
