

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.

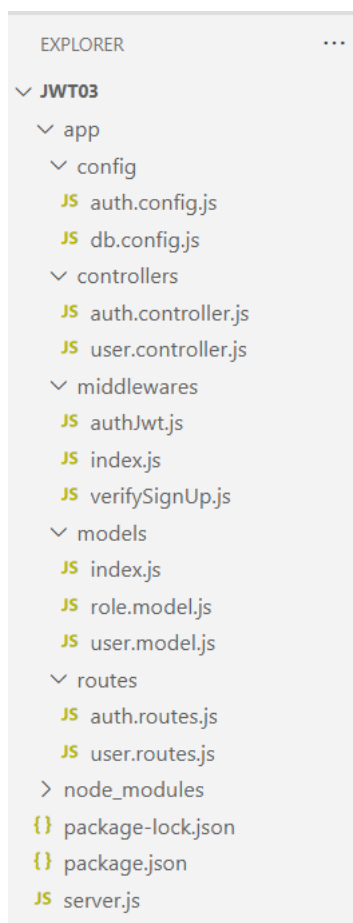
CORS 2 : Intercambio de recursos entre orígenes.

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

middlewares

role.model.js

user.routes.js



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 > [e] default > 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  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 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

app > models > 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
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50

```

```

EXPLORER  JS index.js  X
app > models > JS index.js > ...
26 // Inicializamos los modelos de usuario y rol, pasándoles la instancia de
27 // Sequelize y el objeto Sequelize
28 db.user = userModel(sequelize, Sequelize);
29 db.role = roleModel(sequelize, Sequelize);
30
31 // Definimos una relación de muchos a muchos entre roles y usuarios
32 db.role.belongsToMany(db.user, {
33   through: "user_roles", // Nombre de la tabla intermedia que
34   // almacena las relaciones
35   foreignKey: "roleId", // Clave foránea en la tabla intermedia que
36   // referencia a roles
37   otherKey: "userId", // Clave foránea en la tabla intermedia que
38   // referencia a usuarios
39 });
40
41 // Definimos la relación inversa de muchos a muchos entre usuarios y roles
42 db.user.belongsToMany(db.role, {
43   through: "user_roles", // Nombre de la tabla intermedia
44   // Clave foránea que referencia a usuarios
45   foreignKey: "userId", // Clave foránea que referencia a roles
46   otherKey: "roleId", // Alias para acceder a los roles de un
47   // usuario
48 });
49
50 // Definimos una constante con los posibles roles que se pueden asignar
51 db.ROLES = ["user", "admin", "moderator"];
52
53 // Exportamos el objeto db para que pueda ser utilizado en otras partes de
54 // la aplicación
55 export default db;
56

```

9.- Definir modelos.

```

EXPLORER  JS user.model.js  X
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

app > models > 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 };

```

```

EXPLORER  JS role.model.js  X
app > models > 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.

```

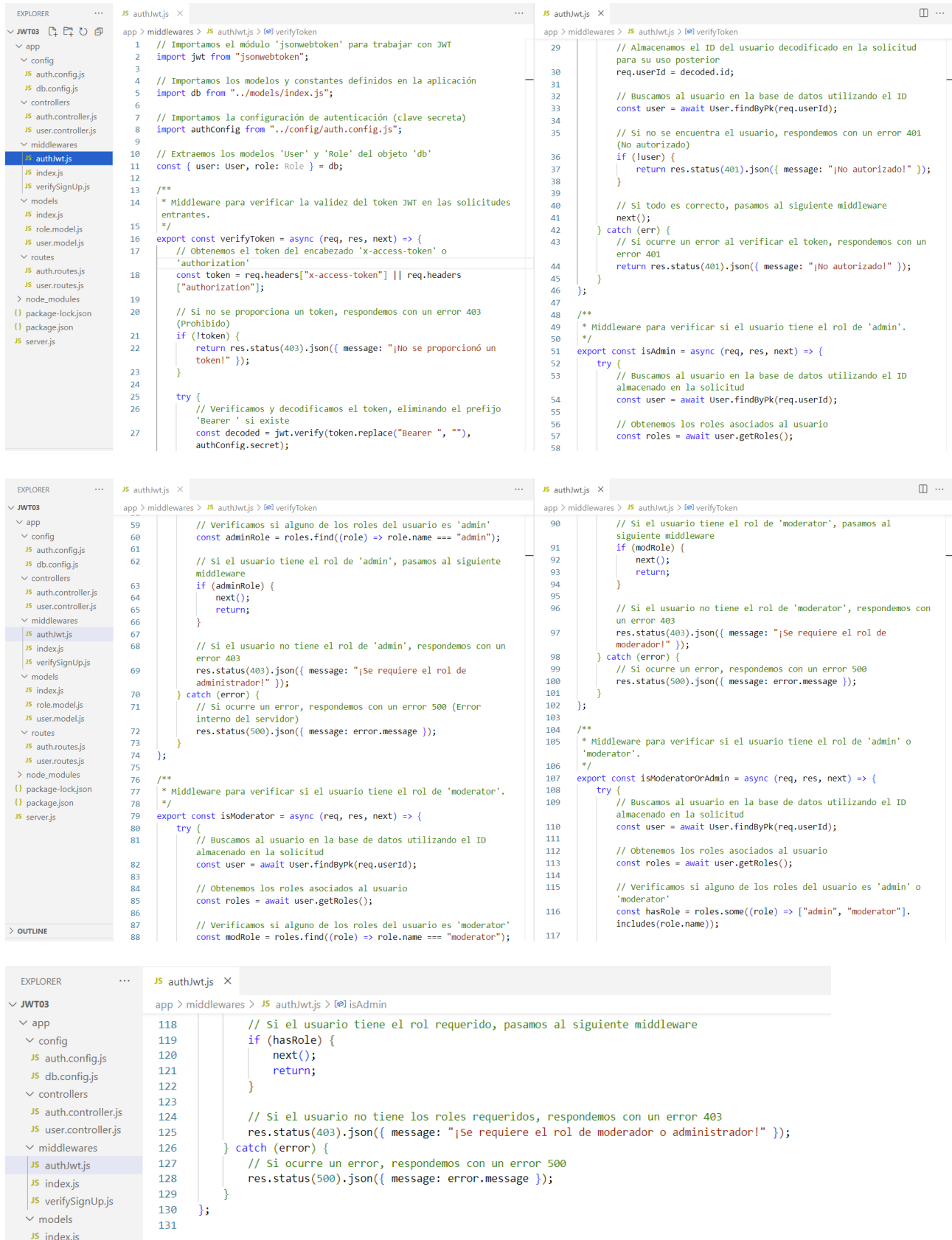
EXPLORER  JS verifySignUp.js  X
JWT03
  app
    config
      JS auth.config.js
      JS db.config.js
    controllers
      JS auth.controller.js
      JS user.controller.js
    middlewares
      JS auth.jwt.js
      JS index.js
      JS verifySignUp.js
    models
      JS index.js
      JS role.model.js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

app > middlewares > 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     // Si se encuentra un usuario con ese nombre de usuario,
20     // respondemos con un error
21     if (userByUsername) {
22       return res.status(400).json({ message: "El nombre de usuario
23       ya está en uso!" });
24     }
25
26     // Buscamos un usuario con el mismo correo electrónico
27     // proporcionado en la solicitud
28     const userByEmail = await User.findOne({
29       where: { email: req.body.email },
30     });
31     // Si se encuentra un usuario con ese correo electrónico,
32     // respondemos con un error
33     if (userByEmail) {
34       return res.status(400).json({ message: "El correo
35       electrónico ya está en uso!" });
36     }
37   } catch (error) {
38     // En caso de error en la base de datos u otro, respondemos con
39     // un error del servidor
40     res.status(500).json({ message: error.message });
41   }
42 }
43
44 /**
45  * Middleware para verificar si los roles proporcionados existen en la
46  * lista de roles permitidos.
47  */
48 export const checkRolesExisted = (req, res, next) => {
49   // Verificamos si se proporcionaron roles en la solicitud
50   if (req.body.roles) {
51     // Iteramos sobre cada rol proporcionado
52     for (const role of req.body.roles) {
53       // Si el rol no existe en la lista de roles permitidos,
54       // respondemos con un error
55       if (!ROLES.includes(role)) {
56         return res.status(400).json({ message: "El rol " + role +
57         " no existe!" });
58       }
59     }
60   }
61   // Si todos los roles son válidos o no se proporcionaron roles,
62   // pasamos al siguiente middleware
63   next();
64 }
65

```

11.- Middleware de autenticación JWT

Verifica tokens y verifica los roles de los usuarios.



```

EXPLORER
  ...
  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 > middlewares > JS authJwt.js > verifyToken
1 // Importamos el módulo 'jsonwebtoken' para trabajar con JWT
2 import jwt from "jsonwebtoken";
3
4 // Importamos los modelos y constantes definidos en la aplicación
5 import db from "../models/index.js";
6
7 // Importamos la configuración de autenticación (clave secreta)
8 import authConfig from "../config/auth.config.js";
9
10 // Extraemos los modelos 'User' y 'Role' del objeto 'db'
11 const { user: User, role: Role } = db;
12
13 /**
14  * Middleware para verificar la validez del token JWT en las solicitudes
15  * entrantes.
16  */
17 export const verifyToken = async (req, res, next) => {
18   // Obtenemos el token del encabezado 'x-access-token' o
19   // 'authorization'
20   const token = req.headers["x-access-token"] || req.headers
21     ["authorization"];
22
23   // Si no se proporciona un token, respondemos con un error 403
24   // (Prohibido)
25   if (!token) {
26     return res.status(403).json({ message: "¡No se proporcionó un
27       token!" });
28   }
29
30   try {
31     // Verificamos y decodificamos el token, eliminando el prefijo
32     // 'Bearer ' si existe
33     const decoded = jwt.verify(token.replace("Bearer ", ""),
34       authConfig.secret);
35
36     // Almacenamos el ID del usuario decodificado en la solicitud
37     // para su uso posterior
38     req.userId = decoded.id;
39
40     // Buscamos al usuario en la base de datos utilizando el ID
41     // (No autorizado)
42     const user = await User.findById(req.userId);
43
44     // Si no se encuentra el usuario, respondemos con un error 401
45     // (No autorizado)
46     if (!user) {
47       return res.status(401).json({ message: "¡No autorizado!" });
48     }
49
50     // Si todo es correcto, pasamos al siguiente middleware
51     next();
52   } catch (err) {
53     // Si ocurre un error al verificar el token, respondemos con un
54     // error 401
55     return res.status(401).json({ message: "¡No autorizado!" });
56   }
57 }
58
59 /**
60  * Middleware para verificar si el usuario tiene el rol de 'admin'.
61  */
62 export const isAdmin = async (req, res, next) => {
63   try {
64     // Buscamos al usuario en la base de datos utilizando el ID
65     // almacenado en la solicitud
66     const user = await User.findById(req.userId);
67
68     // Obtenemos los roles asociados al usuario
69     const roles = await user.getRoles();
70
71     // Verificamos si alguno de los roles del usuario es 'admin'
72     const adminRole = roles.find((role) => role.name === "admin");
73
74     // Si el usuario tiene el rol de 'admin', pasamos al siguiente
75     // middleware
76     if (adminRole) {
77       next();
78       return;
79     }
80
81     // Si el usuario no tiene el rol de 'admin', respondemos con un
82     // error 403
83     res.status(403).json({ message: "¡Se requiere el rol de
84       administrador!" });
85   } catch (error) {
86     // Si ocurre un error, respondemos con un error 500 (Error
87     // interno del servidor)
88     res.status(500).json({ message: error.message });
89   }
90 }
91
92 /**
93  * Middleware para verificar si el usuario tiene el rol de 'moderator'.
94  */
95 export const isModerator = async (req, res, next) => {
96   try {
97     // Buscamos al usuario en la base de datos utilizando el ID
98     // almacenado en la solicitud
99     const user = await User.findById(req.userId);
100
101     // Obtenemos los roles asociados al usuario
102     const roles = await user.getRoles();
103
104     // Verificamos si alguno de los roles del usuario es 'admin' o
105     // 'moderator'
106     const hasRole = roles.some((role) => ["admin", "moderator"].
107       includes(role.name));
108
109     // Si el usuario tiene el rol requerido, pasamos al siguiente
110     // middleware
111     if (hasRole) {
112       next();
113       return;
114     }
115
116     // Si el usuario no tiene los roles requeridos, respondemos con un
117     // error 403
118     res.status(403).json({ message: "¡Se requiere el rol de
119       moderador o administrador!" });
120   } catch (error) {
121     // Si ocurre un error, respondemos con un error 500
122     res.status(500).json({ message: error.message });
123   }
124 }
125
126
127
128
129
130
131
  
```

12.- Exportación de middleware

```

EXPLORER    ...    JS indexjs  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 indexjs
      └─ JS verifySignUp.js

app > middlewares > JS indexjs
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 indexjs
      └─ JS verifySignUp.js
    └─ models
      └─ JS indexjs
      └─ 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 > [0] 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

```

```

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 indexjs
      └─ JS verifySignUp.js
    └─ models
      └─ JS indexjs
      └─ 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 > [0] 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.

```

EXPLORER  ...  JS user.controller.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
    └─ models
      JS index.js
      JS role.model.js
      JS user.model.js
    └─ routes
      JS auth.routes.js

app > controllers > JS user.controller.js > ...
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

```

EXPLORER  ...  JS auth.routes.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
    └─ models
      JS index.js
      JS role.model.js
      JS user.model.js
    └─ routes
      JS auth.routes.js
      JS user.routes.js
    └─ node_modules
      (l) package-lock.json
      (l) package.json
      JS server.js

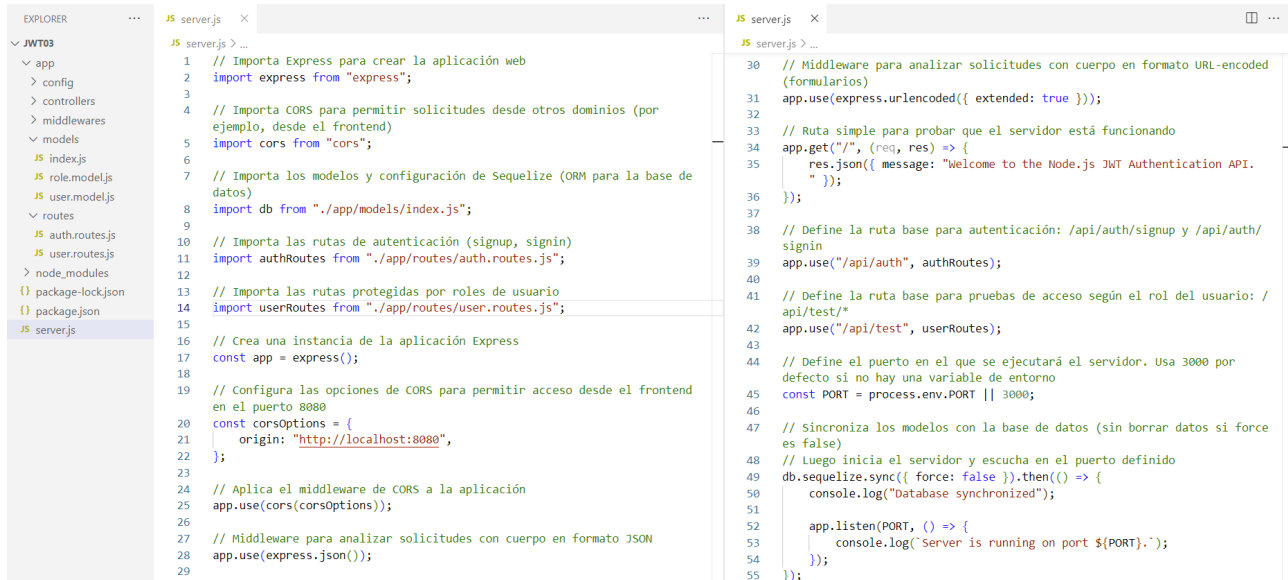
app > routes > JS auth.routes.js > ...
1  // Importa Express para definir rutas
2  import express from "express";
3
4  // Importa las funciones del controlador de autenticación
5  import { signup, signin } from "../controllers/auth.controller.js";
6
7  // Importa los middlewares que verifican datos antes de registrar
  un usuario
8  import {
9    |   checkDuplicateUsernameOrEmail, // Verifica si el username o
10   |   email ya existen
11   |   checkRolesExisted,           // Verifica si los roles
12   |   enviados son válidos
13   | } from "../middlewares/verifySignUp.js";
14
15 // Crea un router de Express para definir las rutas relacionadas
  con autenticación
16 const router = express.Router();
17
18 // Ruta para registrar un nuevo usuario (signup)
19 // Aplica dos middlewares antes de ejecutar la función signup:
20 // 1. checkDuplicateUsernameOrEmail: asegura que el username y el
21 // email no estén repetidos
22 // 2. checkRolesExisted: valida que los roles proporcionados
23 // existan en la base de datos
24 router.post("/signup", [checkDuplicateUsernameOrEmail,
25 |   checkRolesExisted], signup);
26
27 // Ruta para iniciar sesión (signin)
28 // No necesita middlewares previos, va directo al controlador
29 signin
30 router.post("/signin", signin);
31
32 // Exporta el router para poder usarlo en la configuración
  principal de rutas de la app
33 export default router;

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

```


16.- Crear server.js

Usando la sintaxis ESMODULE , configure el servidor Express :



```

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

JS server.js
1 // Importa Express para crear la aplicación web
2 import express from "express";
3
4 // Importa CORS para permitir solicitudes desde otros dominios (por
  ejemplo, desde el frontend)
5 import cors from "cors";
6
7 // Importa los modelos y configuración de Sequelize (ORM para la base de
  datos)
8 import db from "../app/models/index.js";
9
10 // Importa las rutas de autenticación (signup, signin)
11 import authRoutes from "../app/routes/auth.routes.js";
12
13 // Importa las rutas protegidas por roles de usuario
14 import userRoutes from "../app/routes/user.routes.js";
15
16 // Crea una instancia de la aplicación Express
17 const app = express();
18
19 // Configura las opciones de CORS para permitir acceso desde el frontend
  en el puerto 8080
20 const corsOptions = {
21   origin: "http://localhost:8080",
22 };
23
24 // Aplica el middleware de CORS a la aplicación
25 app.use(cors(corsOptions));
26
27 // Middleware para analizar solicitudes con cuerpo en formato JSON
28 app.use(express.json());
29
30 // Middleware para analizar solicitudes con cuerpo en formato URL-encoded
  (formularios)
31 app.use(express.urlencoded({ extended: true }));
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
  defecto si no hay una variable de entorno
45 const PORT = process.env.PORT || 3000;
46
47 // Sincroniza los modelos con la base de datos (sin borrar datos si force
  es false)
48 // Luego inicia el servidor y escucha en el puerto definido
49 db.sequelize.sync({ force: false }).then(() => {
50   console.log("Database synchronized");
51
52   app.listen(PORT, () => {
53     console.log(`Server is running on port ${PORT}.`);
54   });
55 });
  
```

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

18.- Insertar registro en la tabla roles.

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

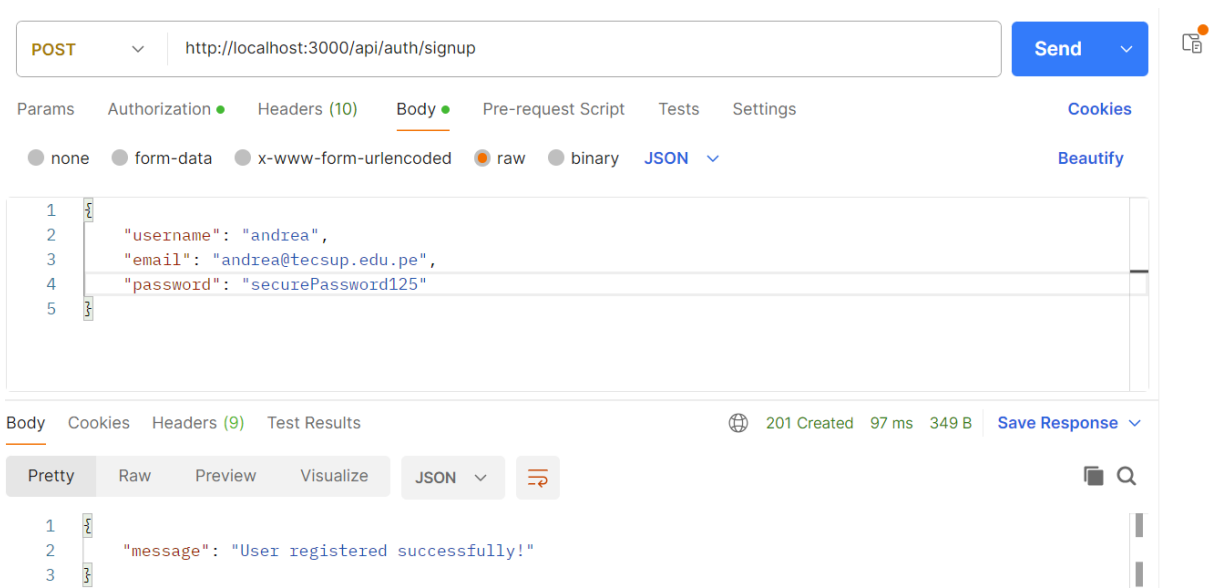
INSERT INTO roles VALUES (2, 'moderator', now(), now());

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 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": "andrea",
3   "email": "andrea@tecups.edu.pe",
4   "password": "securePassword125"
5
  
```

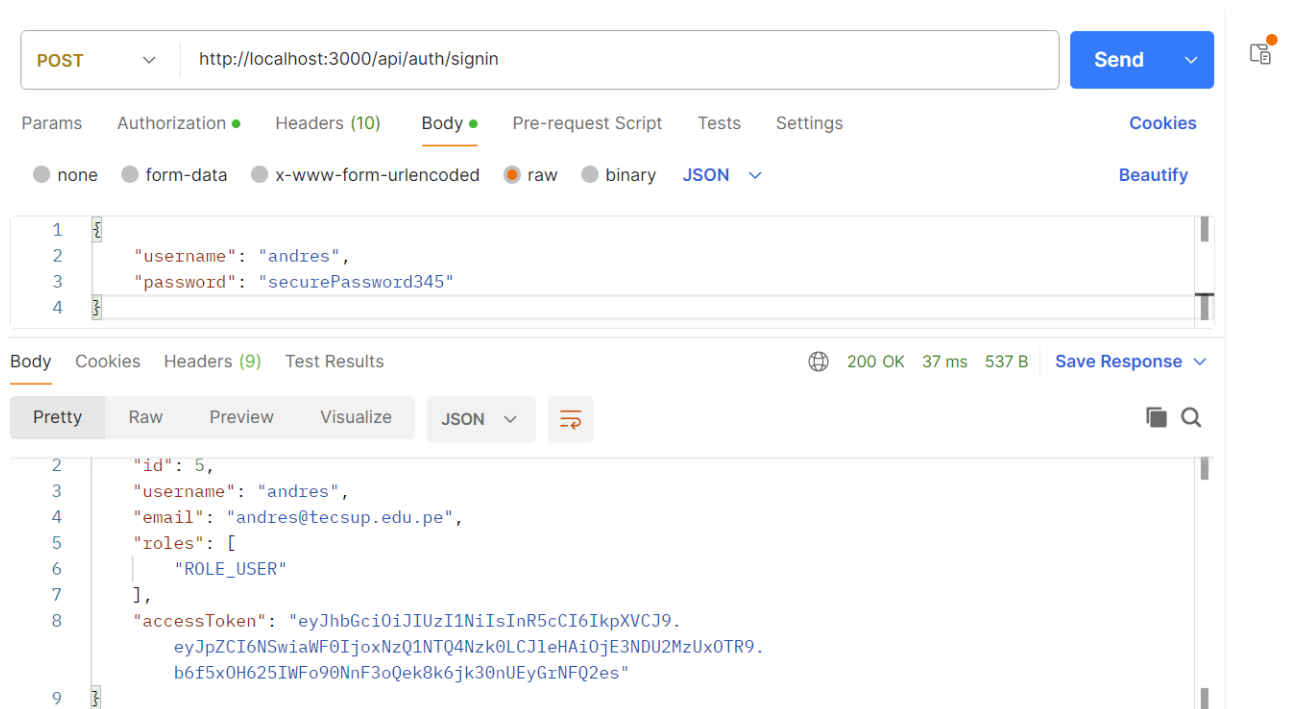
Body Cookies Headers (9) Test Results 201 Created 97 ms 349 B Save Response ⌵

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

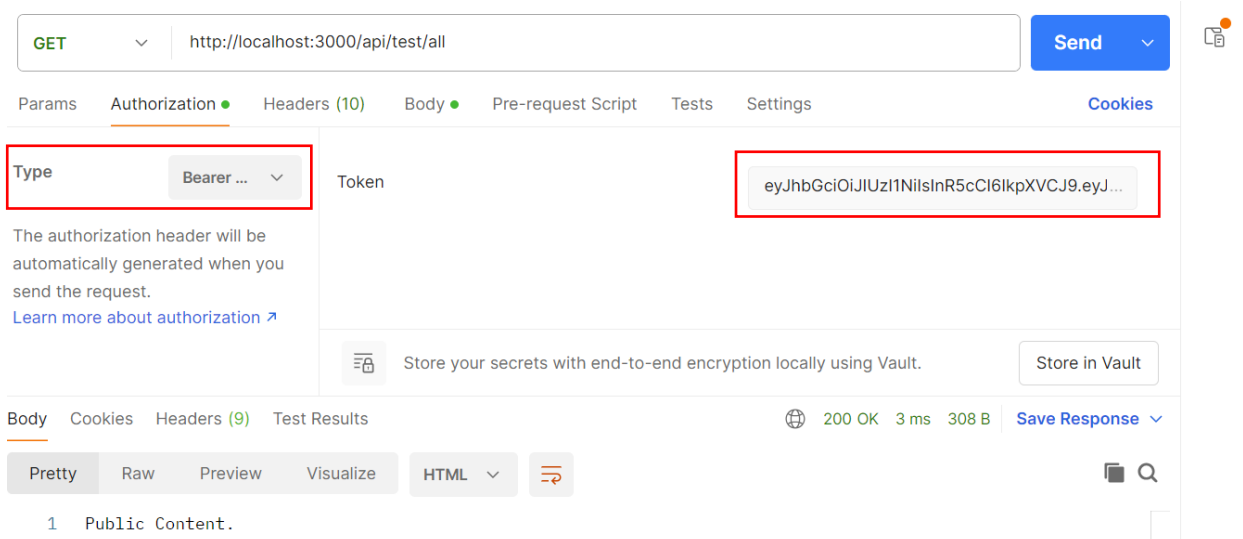
```

2 {
3   "id": 5,
4   "username": "andres",
5   "email": "andres@tecsup.edu.pe",
6   "roles": [
7     "ROLE_USER"
8   ],
9   "accessToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6NSwiaWF0IjoxNzQ1NTQ4Nzk0LCJleHAiOiE3NDU2MzUxOTR9.b6f5x0H625IWFo90NnF3oQek8k6jk30nUEyGrNFQ2es"

```

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.eyJpZCI6NSwiaWF0IjoxNzQ1NTQ4Nzk0LCJleHAiOiE3NDU2MzUxOTR9.b6f5x0H625IWFo90NnF3oQek8k6jk30nUEyGrNFQ2es

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

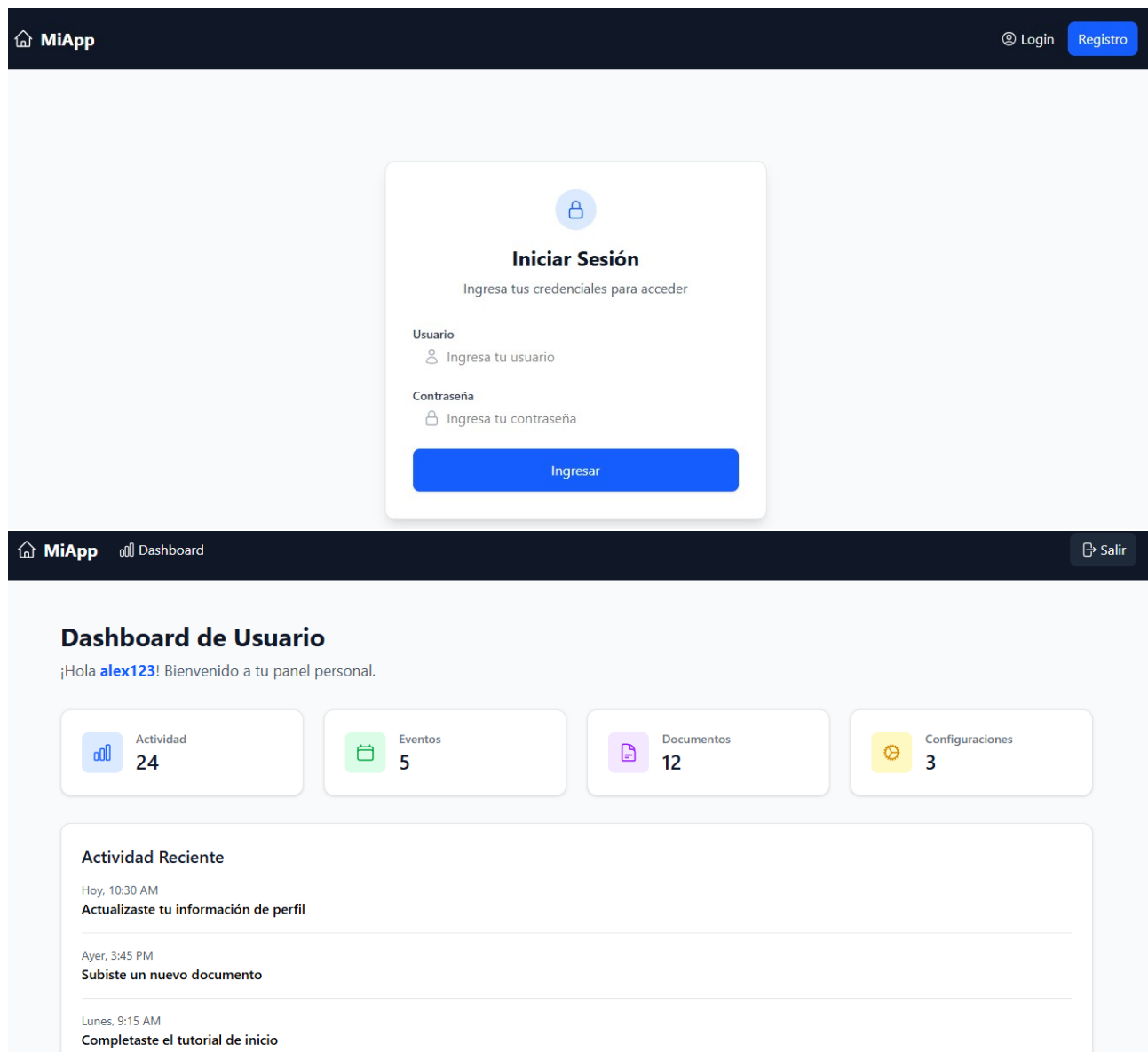
- GET /api/test/admin- Administrador

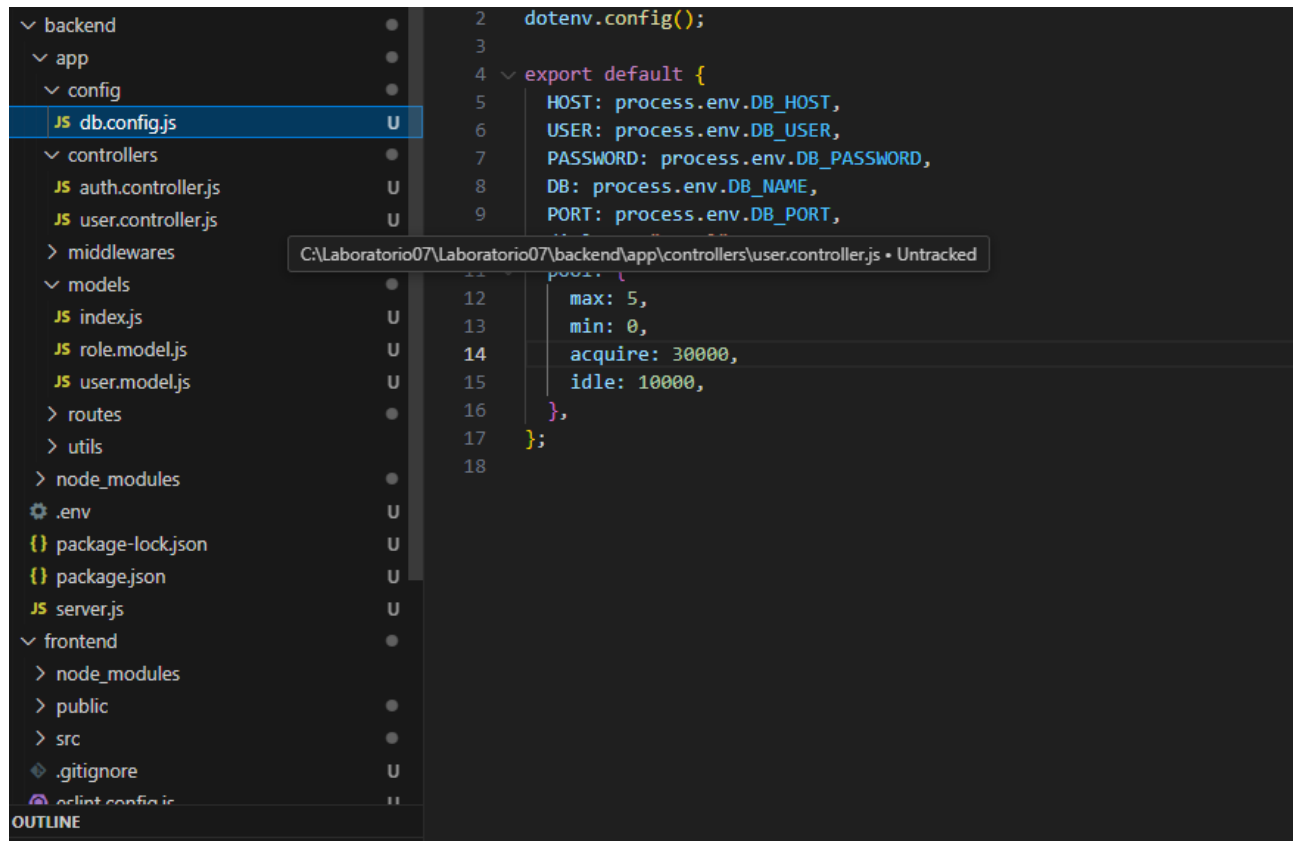
Ejercicios de aplicación

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.






```
2  dotenv.config();
3
4  export default {
5    HOST: process.env.DB_HOST,
6    USER: process.env.DB_USER,
7    PASSWORD: process.env.DB_PASSWORD,
8    DB: process.env.DB_NAME,
9    PORT: process.env.DB_PORT,
10  };
11
12  pool: {
13    max: 5,
14    min: 0,
15    acquire: 30000,
16    idle: 10000,
17  },
18  };
19  };
20  };
21  };
22  };
23  };
24  };
25  };
26  };
27  };
28  };
29  };
30  };
31  };
32  };
33  };
34  };
35  };
36  };
37  };
38  };
39  };
40  };
41  };
42  };
43  };
44  };
45  };
46  };
47  };
48  };
49  };
50  };
51  };
52  };
53  };
54  };
55  };
56  };
57  };
58  };
59  };
60  };
61  };
62  };
63  };
64  };
65  };
66  };
67  };
68  };
69  };
70  };
71  };
72  };
73  };
74  };
75  };
76  };
77  };
78  };
79  };
80  };
81  };
82  };
83  };
84  };
85  };
86  };
87  };
88  };
89  };
90  };
91  };
92  };
93  };
94  };
95  };
96  };
97  };
98  };
99  };
100  };
101  };
102  };
103  };
104  };
105  };
106  };
107  };
108  };
109  };
110  };
111  };
112  };
113  };
114  };
115  };
116  };
117  };
118  };
119  };
120  };
121  };
122  };
123  };
124  };
125  };
126  };
127  };
128  };
129  };
130  };
131  };
132  };
133  };
134  };
135  };
136  };
137  };
138  };
139  };
140  };
141  };
142  };
143  };
144  };
145  };
146  };
147  };
148  };
149  };
150  };
151  };
152  };
153  };
154  };
155  };
156  };
157  };
158  };
159  };
160  };
161  };
162  };
163  };
164  };
165  };
166  };
167  };
168  };
169  };
170  };
171  };
172  };
173  };
174  };
175  };
176  };
177  };
178  };
179  };
180  };
181  };
182  };
183  };
184  };
185  };
186  };
187  };
188  };
189  };
190  };
191  };
192  };
193  };
194  };
195  };
196  };
197  };
198  };
199  };
200  };
201  };
202  };
203  };
204  };
205  };
206  };
207  };
208  };
209  };
210  };
211  };
212  };
213  };
214  };
215  };
216  };
217  };
218  };
219  };
220  };
221  };
222  };
223  };
224  };
225  };
226  };
227  };
228  };
229  };
230  };
231  };
232  };
233  };
234  };
235  };
236  };
237  };
238  };
239  };
240  };
241  };
242  };
243  };
244  };
245  };
246  };
247  };
248  };
249  };
250  };
251  };
252  };
253  };
254  };
255  };
256  };
257  };
258  };
259  };
260  };
261  };
262  };
263  };
264  };
265  };
266  };
267  };
268  };
269  };
270  };
271  };
272  };
273  };
274  };
275  };
276  };
277  };
278  };
279  };
280  };
281  };
282  };
283  };
284  };
285  };
286  };
287  };
288  };
289  };
290  };
291  };
292  };
293  };
294  };
295  };
296  };
297  };
298  };
299  };
300  };
301  };
302  };
303  };
304  };
305  };
306  };
307  };
308  };
309  };
310  };
311  };
312  };
313  };
314  };
315  };
316  };
317  };
318  };
319  };
320  };
321  };
322  };
323  };
324  };
325  };
326  };
327  };
328  };
329  };
330  };
331  };
332  };
333  };
334  };
335  };
336  };
337  };
338  };
339  };
340  };
341  };
342  };
343  };
344  };
345  };
346  };
347  };
348  };
349  };
350  };
351  };
352  };
353  };
354  };
355  };
356  };
357  };
358  };
359  };
360  };
361  };
362  };
363  };
364  };
365  };
366  };
367  };
368  };
369  };
370  };
371  };
372  };
373  };
374  };
375  };
376  };
377  };
378  };
379  };
380  };
381  };
382  };
383  };
384  };
385  };
386  };
387  };
388  };
389  };
390  };
391  };
392  };
393  };
394  };
395  };
396  };
397  };
398  };
399  };
400  };
401  };
402  };
403  };
404  };
405  };
406  };
407  };
408  };
409  };
410  };
411  };
412  };
413  };
414  };
415  };
416  };
417  };
418  };
419  };
420  };
421  };
422  };
423  };
424  };
425  };
426  };
427  };
428  };
429  };
430  };
431  };
432  };
433  };
434  };
435  };
436  };
437  };
438  };
439  };
440  };
441  };
442  };
443  };
444  };
445  };
446  };
447  };
448  };
449  };
450  };
451  };
452  };
453  };
454  };
455  };
456  };
457  };
458  };
459  };
460  };
461  };
462  };
463  };
464  };
465  };
466  };
467  };
468  };
469  };
470  };
471  };
472  };
473  };
474  };
475  };
476  };
477  };
478  };
479  };
480  };
481  };
482  };
483  };
484  };
485  };
486  };
487  };
488  };
489  };
490  };
491  };
492  };
493  };
494  };
495  };
496  };
497  };
498  };
499  };
500  };
501  };
502  };
503  };
504  };
505  };
506  };
507  };
508  };
509  };
510  };
511  };
512  };
513  };
514  };
515  };
516  };
517  };
518  };
519  };
520  };
521  };
522  };
523  };
524  };
525  };
526  };
527  };
528  };
529  };
530  };
531  };
532  };
533  };
534  };
535  };
536  };
537  };
538  };
539  };
540  };
541  };
542  };
543  };
544  };
545  };
546  };
547  };
548  };
549  };
550  };
551  };
552  };
553  };
554  };
555  };
556  };
557  };
558  };
559  };
560  };
561  };
562  };
563  };
564  };
565  };
566  };
567  };
568  };
569  };
570  };
571  };
572  };
573  };
574  };
575  };
576  };
577  };
578  };
579  };
580  };
581  };
582  };
583  };
584  };
585  };
586  };
587  };
588  };
589  };
590  };
591  };
592  };
593  };
594  };
595  };
596  };
597  };
598  };
599  };
600  };
601  };
602  };
603  };
604  };
605  };
606  };
607  };
608  };
609  };
610  };
611  };
612  };
613  };
614  };
615  };
616  };
617  };
618  };
619  };
620  };
621  };
622  };
623  };
624  };
625  };
626  };
627  };
628  };
629  };
630  };
631  };
632  };
633  };
634  };
635  };
636  };
637  };
638  };
639  };
640  };
641  };
642  };
643  };
644  };
645  };
646  };
647  };
648  };
649  };
650  };
651  };
652  };
653  };
654  };
655  };
656  };
657  };
658  };
659  };
660  };
661  };
662  };
663  };
664  };
665  };
666  };
667  };
668  };
669  };
670  };
671  };
672  };
673  };
674  };
675  };
676  };
677  };
678  };
679  };
680  };
681  };
682  };
683  };
684  };
685  };
686  };
687  };
688  };
689  };
690  };
691  };
692  };
693  };
694  };
695  };
696  };
697  };
698  };
699  };
700  };
701  };
702  };
703  };
704  };
705  };
706  };
707  };
708  };
709  };
710  };
711  };
712  };
713  };
714  };
715  };
716  };
717  };
718  };
719  };
720  };
721  };
722  };
723  };
724  };
725  };
726  };
727  };
728  };
729  };
730  };
731  };
732  };
733  };
734  };
735  };
736  };
737  };
738  };
739  };
740  };
741  };
742  };
743  };
744  };
745  };
746  };
747  };
748  };
749  };
750  };
751  };
752  };
753  };
754  };
755  };
756  };
757  };
758  };
759  };
760  };
761  };
762  };
763  };
764  };
765  };
766  };
767  };
768  };
769  };
770  };
771  };
772  };
773  };
774  };
775  };
776  };
777  };
778  };
779  };
780  };
781  };
782  };
783  };
784  };
785  };
786  };
787  };
788  };
789  };
790  };
791  };
792  };
793  };
794  };
795  };
796  };
797  };
798  };
799  };
800  };
801  };
802  };
803  };
804  };
805  };
806  };
807  };
808  };
809  };
810  };
811  };
812  };
813  };
814  };
815  };
816  };
817  };
818  };
819  };
820  };
821  };
822  };
823  };
824  };
825  };
826  };
827  };
828  };
829  };
830  };
831  };
832  };
833  };
834  };
835  };
836  };
837  };
838  };
839  };
840  };
841  };
842  };
843  };
844  };
845  };
846  };
847  };
848  };
849  };
850  };
851  };
852  };
853  };
854  };
855  };
856  };
857  };
858  };
859  };
860  };
861  };
862  };
863  };
864  };
865  };
866  };
867  };
868  };
869  };
870  };
871  };
872  };
873  };
874  };
875  };
876  };
877  };
878  };
879  };
880  };
881  };
882  };
883  };
884  };
885  };
886  };
887  };
888  };
889  };
890  };
891  };
892  };
893  };
894  };
895  };
896  };
897  };
898  };
899  };
900  };
901  };
902  };
903  };
904  };
905  };
906  };
907  };
908  };
909  };
910  };
911  };
912  };
913  };
914  };
915  };
916  };
917  };
918  };
919  };
920  };
921  };
922  };
923  };
924  };
925  };
926  };
927  };
928  };
929  };
930  };
931  };
932  };
933  };
934  };
935  };
936  };
937  };
938  };
939  };
940  };
941  };
942  };
943  };
944  };
945  };
946  };
947  };
948  };
949  };
950  };
951  };
952  };
953  };
954  };
955  };
956  };
957  };
958  };
959  };
960  };
961  };
962  };
963  };
964  };
965  };
966  };
967  };
968  };
969  };
970  };
971  };
972  };
973  };
974  };
975  };
976  };
977  };
978  };
979  };
980  };
981  };
982  };
983  };
984  };
985  };
986  };
987  };
988  };
989  };
990  };
991  };
992  };
993  };
994  };
995  };
996  };
997  };
998  };
999  };
1000  };
```

 Dashboard Admin Salir

Panel de Administración


Bienvenido **admin**. Gestiona toda la plataforma desde aquí.



Gestión de Usuarios

Administra todos los usuarios del sistema


[Ver detalles →](#)



Configuración del Sistema

Ajusta los parámetros globales de la plataforma

[Configurar →](#)



Estadísticas

Visualiza métricas y análisis de uso

[Ver reportes →](#)

Actividad Reciente del Sistema

FECHA	EVENTO	USUARIO
10/05/2023 14:30	Nuevo usuario registrado	usuario_nuevo
09/05/2023 09:15	Configuración actualizada	admin



Panel de Moderador

¡Hola **moderator!** Desde aquí puedes moderar comentarios, publicaciones y reportes de usuarios.

Conclusiones:

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

- Integramos un stack completo (Node.js + React + MySQL + JWT) garantizando autenticación segura y control de acceso basado en roles.
- Pudimos organizar el proyecto de forma ordenada, separando claramente las responsabilidades del frontend y del backend, lo que facilita su mantenimiento a futuro.
- Aplicamos validaciones desde el frontend, mejorando la calidad de los datos que llegan al servidor y ofreciendo una mejor experiencia a los usuarios.
- Comprendimos mejor cómo proteger rutas y manejar accesos usando JWT y control de roles, algo clave en sistemas reales.
- Comprendimos la importancia de proteger rutas y gestionar sesiones, para garantizar un flujo de usuario seguro y controlado.

https://drive.google.com/file/d/1zdnHY7EaU_KySpmg0P4VmjYpxotWuwwq/view