🛠️ Ejercicio Práctico

📌 **Título:** *Implementación de Autenticación Segura y Control de Acceso en Aplicaciones Web utilizando JWT, Bcrypt y Buenas Prácticas Criptográficas*

🎯 Objetivo del ejercicio:

Implementar un sistema completo de **autenticación** utilizando **JWT** para gestionar la sesión del usuario y un sistema de **roles** (Administrador, Editor, Usuario) para gestionar el **acceso controlado** a los recursos de la aplicación web.

📝 Escenario:

Estás desarrollando una **aplicación de gestión de proyectos** que tiene tres tipos de usuarios:

- **Administrador**: Puede ver, crear, editar y eliminar proyectos.
- **Editor**: Puede ver, crear y editar proyectos, pero no eliminarlos.
- **Usuario**: Solo puede ver los proyectos que le han sido asignados.

El sistema debe permitir a los usuarios autenticarse mediante **JWT**, y en función de su rol, proporcionar acceso a las funcionalidades de la aplicación.

🔍 Tu tarea:

Paso 1 – Instalar dependencias necesarias:
Instala las librerías necesarias para trabajar con **JWT** y **bcrypt** en **Node.js** (o el equivalente en el lenguaje que prefieras):

```
npm install bcrypt jsonwebtoken express
```

1.

Paso 2 – Crear el sistema de registro de usuarios:

1. **Formulario de registro**: Los usuarios deben proporcionar un **nombre de usuario** y una **contraseña**.
    - La contraseña debe ser **hasheada** utilizando **bcrypt** antes de ser almacenada en la base de datos.
    - Almacena un **rol** para cada usuario (por defecto, asigna **Usuario**).
2. **Ejemplo de implementación de registro (Node.js):**

```
const bcrypt = require('bcrypt');
const jwt = require('jsonwebtoken');
const saltRounds = 10;

const registrarUsuario = async (nombre, contraseña, rol = 'usuario') => {
  const hash = await bcrypt.hash(contraseña, saltRounds);
  // Guardar nombre, hash de contraseña y rol en la base de datos
  // Aquí se simula el guardado en la base de datos.
  const nuevoUsuario = { nombre, contraseña: hash, rol };
  return nuevoUsuario; // Simula un registro exitoso
};
```

Paso 3 – Crear el sistema de inicio de sesión:

1. **Inicio de sesión**: Los usuarios deben ingresar **nombre de usuario** y **contraseña**.
    - Si las credenciales son correctas, se debe generar un **JWT** que contenga el **rol** y el **ID del usuario**.
    - El **JWT** debe tener una fecha de expiración para mejorar la seguridad.
2. **Ejemplo de implementación de inicio de sesión (Node.js):**

```
const loginUsuario = async (nombre, contraseñaIngresada, usuarioRegistrado) => {
  const contraseñasCoinciden = await bcrypt.compare(contraseñaIngresada,
usuarioRegistrado.contraseña);
  if (contraseñasCoinciden) {
    const token = jwt.sign({ id: usuarioRegistrado.id, rol: usuarioRegistrado.rol }, 'secreto123',
{ expiresIn: '1h' });
    return token;
  } else {
    throw new Error('Credenciales incorrectas');
  }
```

};

Paso 4 – Middleware de autenticación con JWT:

1. **Crear un middleware de autenticación** que verifique si el **JWT** enviado en las cabeceras de la solicitud es válido.
    - Si el token es válido, el usuario tiene acceso a las rutas protegidas.
    - Si el token es inválido o ha expirado, el servidor debe devolver un error de autenticación.
2. **Ejemplo de implementación del middleware (Node.js):**

```
const verificarToken = (req, res, next) => {
  const token = req.headers['authorization']?.split(' ')[1];
  if (!token) {
    return res.status(403).send('Acceso denegado');
  }

  jwt.verify(token, 'secreto123', (err, decoded) => {
    if (err) {
      return res.status(403).send('Token inválido');
    }
    req.user = decoded;
    next();
  });
};
```

Paso 5 – Control de acceso según el rol:

1. **Crear funciones que controlen el acceso** a las rutas según el rol del usuario.
    - Los **Administradores** pueden realizar todas las acciones.
    - Los **Editores** solo pueden ver, crear y editar proyectos.
    - Los **Usuarios** solo pueden ver los proyectos que han sido asignados.
2. **Ejemplo de implementación de control de acceso (Node.js):**

```
const accesoAdministrador = (req, res, next) => {
  if (req.user.rol !== 'administrador') {
    return res.status(403).send('Acceso restringido');
  }
```

```
  next();
};

const accesoEditor = (req, res, next) => {
  if (req.user.rol === 'usuario') {
    return res.status(403).send('Acceso restringido');
  }
  next();
};
```

Paso 6 – Proteger las rutas de la API:

1. Protege las rutas que solo deben ser accesibles para los usuarios autenticados y con los permisos adecuados.
    - **GET /proyectos**: Solo accesible para **Administradores**, **Editores** y **Usuarios** asignados.
    - **POST /proyectos**: Solo accesible para **Administradores** y **Editores**.
    - **DELETE /proyectos**: Solo accesible para **Administradores**.
2. **Ejemplo de implementación de rutas protegidas (Node.js):**

```
app.get('/proyectos', verificarToken, accesoEditor, (req, res) => {
  // Lógica para obtener proyectos
  res.send('Proyectos obtenidos');
});

app.post('/proyectos', verificarToken, accesoEditor, (req, res) => {
  // Lógica para crear un proyecto
  res.send('Proyecto creado');
});

app.delete('/proyectos/:id', verificarToken, accesoAdministrador, (req, res) => {
  // Lógica para eliminar un proyecto
  res.send('Proyecto eliminado');
});
```

Paso 7 – Cerrar sesión:

1. **Cerrar sesión**: Implementa una función que invalide el **JWT** en el lado del cliente (eliminando el token de las cookies o almacenamiento local).

✅ Resultado esperado:

- Un sistema completo de **autenticación con JWT** que permite a los usuarios registrarse, iniciar sesión y acceder a recursos protegidos según su rol.
- Implementación de un sistema de **roles** que controla el acceso a las funcionalidades de la aplicación (Administrador, Editor, Usuario).
- Uso de **middleware de autenticación** para verificar el **JWT** y el **control de acceso por roles** para asegurar las rutas.
- **Control de acceso** que impide a los usuarios no autorizados realizar acciones no permitidas (como eliminar proyectos o acceder a proyectos que no les corresponden).

📄 Entrega sugerida:

1. **Código fuente** de la implementación de la **autenticación con JWT** y el **control de acceso por roles**.
2. **Capturas de pantalla** o una demostración funcional del sistema de autenticación y control de acceso según el rol.
3. **Informe breve** que explique cómo se implementó la gestión de sesiones, el control de acceso y el uso de **JWT**.

🧰 Herramientas recomendadas:

- **Node.js** y **Express.js** para el backend (puedes usar cualquier otro framework backend de tu preferencia).
- **JWT** (puedes usar la librería **jsonwebtoken** para crear y verificar los tokens).
- **bcrypt** para el hashing de contraseñas.
- **MongoDB** o **MySQL** para almacenar los **usuarios**, **roles**, y **proyectos**.
- **Postman** o **Insomnia** para probar las rutas de la API.

📝 Sugerencia para extender el ejercicio (opcional):

- **Autenticación Multifactor (MFA)**: Implementa una capa adicional de seguridad utilizando un segundo factor (como un código enviado por correo electrónico o SMS).

- **Token de actualización (Refresh Token)**: Añade la posibilidad de usar un **refresh token** para mantener la sesión del usuario activa después de que el JWT haya expirado, evitando que tengan que volver a autenticarse.
