



TECNOLÓGICO
NACIONAL DE MÉXICO



TECNOLÓGICO NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE TLAXIACO

ASIGNATURA:

Seguridad y virtualización

DOCENTE:

Ing. Edward Osorio Salinas

INTEGRANTES DEL EQUIPO:

Luz Arleth López Bautista 21620036

Saúl López Bautista 21620073

ACTIVIDAD:

PRÁCTICA 2: Autorización y Autenticación

CARRERA:

INGENIERIA EN SISTEMAS COMPUTACIONALES

SEPTIMO SEMESTRE

GRUPO:

7US

Heroica ciudad de Tlaxiaco. A 07 de septiembre del 2024.

Contenido

INTRODUCCIÓN.....	5
EJERCICIO 1. Crea una aplicación [web mobile escritorio] que permita loguearse con un usuario y contraseña.....	6
EJERCICIO 2. Implementa un mecanismo de autorización que permita o deniegue el acceso a ciertas rutas de la aplicación, en este caso la página de perfil y la página de administración si en dado caso el usuario no ha iniciado sesión o no tiene el rol de administrador.	19
EJERCICIO 3. Implementa un mecanismo de autenticación que permita a los usuarios registrarse, iniciar sesión y cerrar sesión.	23
EJERCICIO 4. Implementa un mecanismo para cerrar sesión de un usuario si ha pasado un tiempo determinado sin actividad de 5 mins.	29
CONCLUSION	31
INVESTIGACION SERVICIOS DE AUTENTICACIÓN	32
LDAP (LIGHTWEIGHT DIRECTORY ACCESS PROTOCOL)	32
PRINCIPIOS CLAVE DE LDAP	32
TIPOS DE AUTENTICACIÓN EN LDAP	33
EJEMPLO DE CASOS DE USO	34
VENTAJAS DE LDAP.....	35
DESVENTAJAS DE LDAP	35
EJEMPLO DE UNA BÚSQUEDA LDAP.....	35
RADIUS (REMOTE AUTHENTICATION DIAL-IN USER SERVICE).....	36
FUNCIONES PRINCIPALES DE RADIUS:.....	36
ARQUITECTURA DE RADIUS:	37
PROCESO DE AUTENTICACIÓN DE RADIUS.....	37
PROTOCOLOS Y SEGURIDAD	38
CARACTERÍSTICAS DE RADIUS	39
VENTAJAS DE RADIUS	40
DESVENTAJAS DE RADIUS	40
EJEMPLOS DE USO DE RADIUS	41
TACACS+ (TERMINAL ACCESS CONTROLLER ACCESS-CONTROL SYSTEM PLUS)	42
FUNCIONES DE TACACS+	43

PROCESO DE AUTENTICACIÓN TACACS+	44
VENTAJAS DE TACACS+	44
DESVENTAJAS DE TACACS+	45
KERBEROS.....	46
COMPONENTES CLAVE:	46
PROCESO DE AUTENTICACIÓN DE KERBEROS.....	47
CARACTERÍSTICAS CLAVE DE KERBEROS	47
VENTAJAS DE KERBEROS	48
DESVENTAJAS DE KERBEROS:	49
CASOS DE USO DE KERBEROS:	50
INVESTIGACIÓN DE LOS SERVICIOS DE AUTORIZACIÓN	51
ACL (ACCESS CONTROL LIST).....	51
TIPOS DE PERMISOS EN UNA ACL	52
TIPOS DE ACL	52
VENTAJAS DE ACL.....	52
DESVENTAJAS DE ACL	53
USO DE ACL EN DIFERENTES CONTEXTOS.....	53
EJEMPLO PRÁCTICO DE ACL EN UN SISTEMA DE ARCHIVOS	54
ACL EN DISPOSITIVOS DE RED	54
RBAC (ROLE-BASED ACCESS CONTROL)	55
COMPONENTES CLAVE DE RBAC:	55
VENTAJAS DE RBAC	56
DESVENTAJAS DE RBAC	57
IMPLEMENTACIÓN DE RBAC	57
COMPARACIÓN CON OTROS MODELOS DE CONTROL DE ACCESO.....	58
BIBLIOGRAFIA	59

<i>Ilustración 1 Código del archivo register.html</i>	7
<i>Ilustración 2 Funciones del código del archivo script.js</i>	8
<i>Ilustración 3 Código para mostrar la fuerza de las contraseñas y de que no coincidan</i>	8
<i>Ilustración 4 Código del archivo admin.html</i>	9
<i>Ilustración 5 Código del archivo login.html</i>	10
<i>Ilustración 6 Código del archivo profile.html</i>	12
<i>Ilustración 7 Instalación de npm</i>	13
<i>Ilustración 8 Ejecución del comando de node app.js</i>	13
<i>Ilustración 9 Página de registro</i>	13
<i>Ilustración 10 Fuerza de contraseña débil</i>	14
<i>Ilustración 11 Fuerza de contraseña moderada</i>	14
<i>Ilustración 12 Fuerza de contraseña fuerte</i>	15
<i>Ilustración 13 Selección de registro como usuario</i>	15
<i>Ilustración 14 Página de inicio de sesión</i>	15
<i>Ilustración 15 Inicio de sesión con usuario y contraseña</i>	16
<i>Ilustración 16 página del usuario</i>	16
<i>Ilustración 17 Registro como administrador</i>	17
<i>Ilustración 18 Inicio de sesión como administrador</i>	17
<i>Ilustración 19 Página del administrador</i>	18
<i>Ilustración 20 Configuración de archivos estáticos</i>	19
<i>Ilustración 21 Configuración de inicio de sesion</i>	19
<i>Ilustración 22 Rutas de autenticación</i>	19
<i>Ilustración 23 Verificación de usuario autenticado</i>	20
<i>Ilustración 24 Código de acceso a administradores</i>	20
<i>Ilustración 25 Código de registro de nuevos usuarios</i>	20
<i>Ilustración 26 Login deminicio de sesion</i>	21
<i>Ilustración 27 Código de protección de rutas</i>	21
<i>Ilustración 28 inicio de sesión y registro se configuraron</i>	21
<i>Ilustración 29 Página de inicio de sesión</i>	22
<i>Ilustración 30 Mensaje denegado</i>	22
<i>Ilustración 31 Código de enrutador de express</i>	23
<i>Ilustración 32 Creación de arreglo</i>	23
<i>Ilustración 33 Validación de contraseñas</i>	23
<i>Ilustración 34 Código de verificación de credenciales</i>	24
<i>Ilustración 35 Página de registro de usuarios</i>	24
<i>Ilustración 36 código de verificación de usuario autenticado</i>	24
<i>Ilustración 37 verificación de usuario autenticado como administrador</i>	25
<i>Ilustración 38 Código para cerrar sesion</i>	25
<i>Ilustración 39 Nombre, versión y descripción del proyecto</i>	25
<i>Ilustración 40 Script para iniciar la aplicación</i>	26
<i>Ilustración 41 Dependencias para el proyecto</i>	26
<i>Ilustración 42 Script de inactividad</i>	29
<i>Ilustración 43 Identificador de temporizador</i>	29
<i>Ilustración 44 Ruta de cierre de sesión</i>	29
<i>Ilustración 45 Función a ejecutar cuando el temporizador alcance su límite</i>	29
<i>Ilustración 46 Función para el comportamiento de inactividad</i>	30

INTRODUCCIÓN

La práctica titulada "Autorización y Autenticación" aborda los conceptos fundamentales de seguridad en aplicaciones web, enfocándose en los mecanismos que garantizan el acceso controlado a las funcionalidades y recursos de la aplicación. Durante el desarrollo de esta práctica, se realizaron una serie de ejercicios utilizando Visual Studio Code como entorno de desarrollo, con el propósito de implementar funcionalidades clave para el manejo de usuarios y roles dentro de una aplicación web.

Los ejercicios realizados incluyen la creación de una aplicación web que permite a los usuarios registrarse, iniciar sesión y acceder a diferentes páginas dependiendo de su nivel de autorización. Además, se implementaron medidas adicionales para validar la seguridad de las contraseñas y mecanismos para gestionar el cierre de sesión en casos de inactividad prolongada. Estos ejercicios no solo permitieron profundizar en el concepto de autenticación, donde se valida la identidad de los usuarios, sino también en la autorización, encargada de otorgar permisos según el rol asignado, como en el caso de un administrador.

En este reporte, se detallarán los pasos seguidos en la implementación de las funcionalidades, destacando cómo se manejaron las rutas protegidas y la validación de los roles para asegurar un acceso controlado y seguro.

EJERCICIO 1. Crea una aplicación [web|mobile|escritorio] que permita loguearse con un usuario y contraseña.

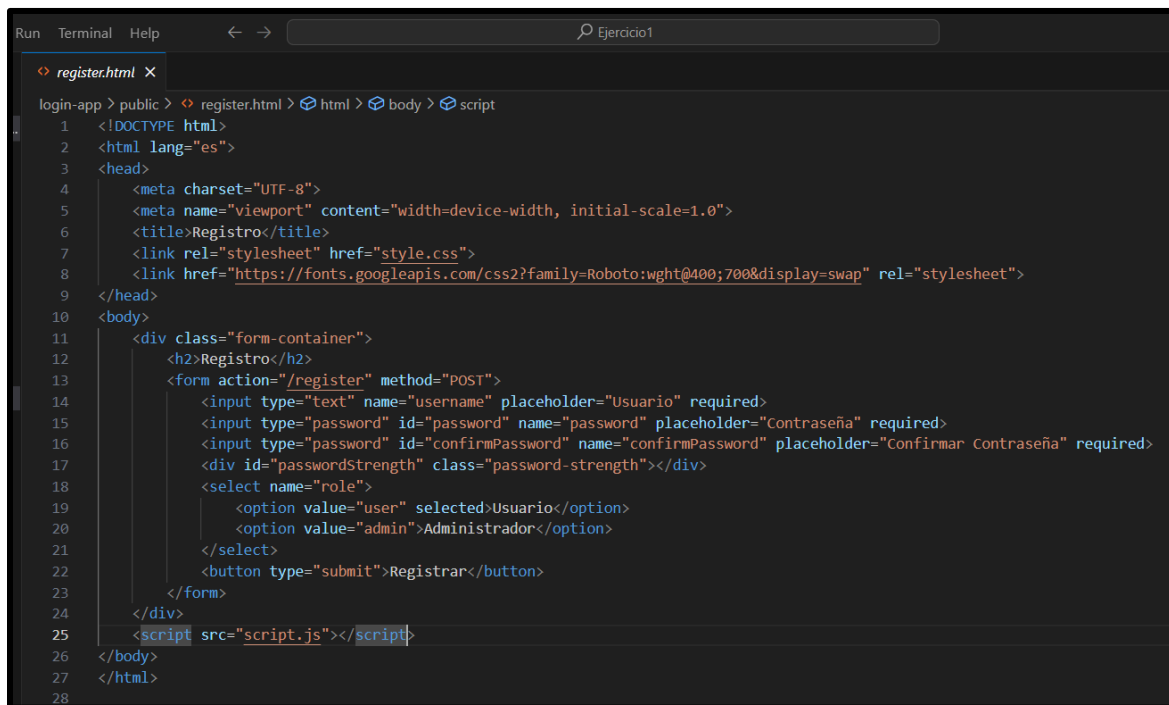
- La aplicación debe tener un formulario de login con los campos de usuario y contraseña.
- La aplicación debe tener un formulario de registro con los campos de usuario, contraseña y confirmación de contraseña.
- La aplicación debe decirme si la contraseña es segura o no (extra).
- La aplicación debe tener una página de inicio que sea accesible para cualquier usuario.
- La aplicación debe tener una página de perfil que solo sea accesible si el usuario ha iniciado sesión.
- La aplicación debe tener una página de administración que solo sea accesible si el usuario ha iniciado sesión y tiene un rol de administrador.

Paso 1. Archivo register.html:

Lo primero que realizamos fue crear un archivo llamado "register.html". Este archivo define la estructura y el diseño de la página de registro.

Dentro del archivo contiene lo siguiente:

- **Formulario:** Se usa un formulario (<form>) con los campos username, password, y confirmPassword para que el usuario ingrese sus datos. También tiene un campo <select> para elegir el rol del usuario (usuario o administrador).
- **Contraseña segura:** Se añade un <div> con id="passwordStrength", que es el contenedor donde se mostrará la fuerza de la contraseña en tiempo real.
- **Script externo:** Se carga el archivo script.js al final del cuerpo para que el código JavaScript maneje la validación y verificación de la contraseña.



```
1 <!DOCTYPE html>
2 <html lang="es">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Registro</title>
7   <link rel="stylesheet" href="style.css">
8   <link href="https://fonts.googleapis.com/css2?family=Roboto:wght@400;700&display=swap" rel="stylesheet">
9 </head>
10 <body>
11   <div class="form-container">
12     <h2>Registro</h2>
13     <form action="/register" method="POST">
14       <input type="text" name="username" placeholder="Usuario" required>
15       <input type="password" id="password" name="password" placeholder="Contraseña" required>
16       <input type="password" id="confirmPassword" name="confirmPassword" placeholder="Confirmar Contraseña" required>
17       <div id="passwordStrength" class="password-strength"></div>
18       <select name="role">
19         <option value="user" selected>Usuario</option>
20         <option value="admin">Administrador</option>
21       </select>
22       <button type="submit">Registrar</button>
23     </form>
24   </div>
25   <script src="script.js"></script>
26 </body>
27 </html>
28
```

Ilustración 1 Código del archivo register.html

Paso 2. Creamos un archivo js:

Como segundo paso creamos un archivo llamado script.js.

Este archivo contiene el código que valida la fortaleza de la contraseña y la coincidencia entre las contraseñas.

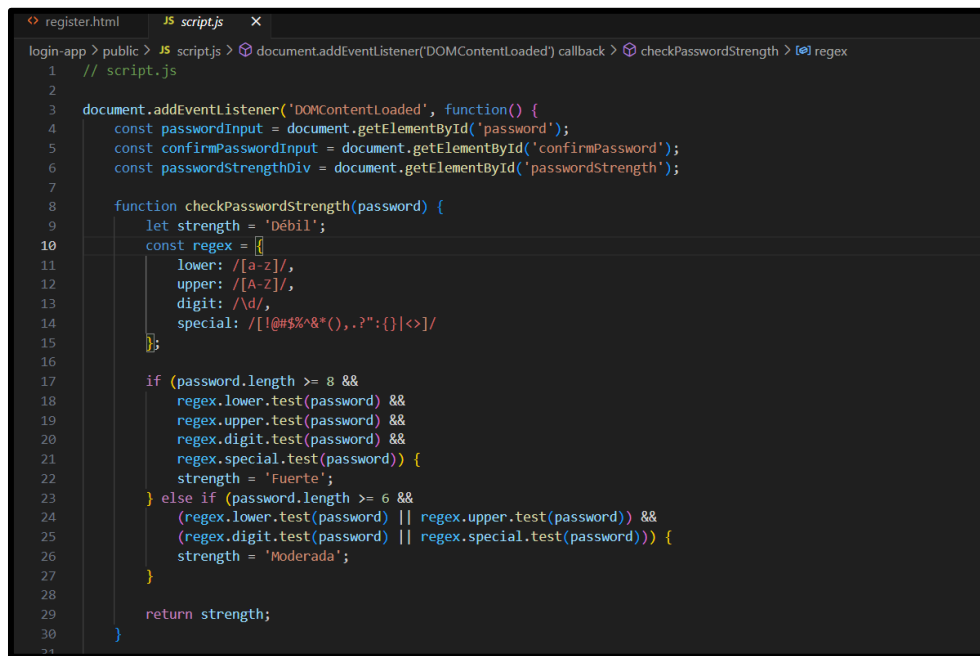
Dentro de este archivo .js realizamos lo siguiente:

Creamos una función llamada: **checkPasswordStrength()**, la cual esta función toma la contraseña como argumento y la verifica usando expresiones regulares (regex). Las expresiones regulares buscan la presencia de letras mayúsculas, minúsculas, números y caracteres especiales.

Si la contraseña tiene al menos 8 caracteres y cumple con todas las reglas (mayúsculas, minúsculas, números y caracteres especiales), se considera "Fuerte".

Si tiene al menos 6 caracteres y cumple con algunas de estas reglas, se considera "Moderada".

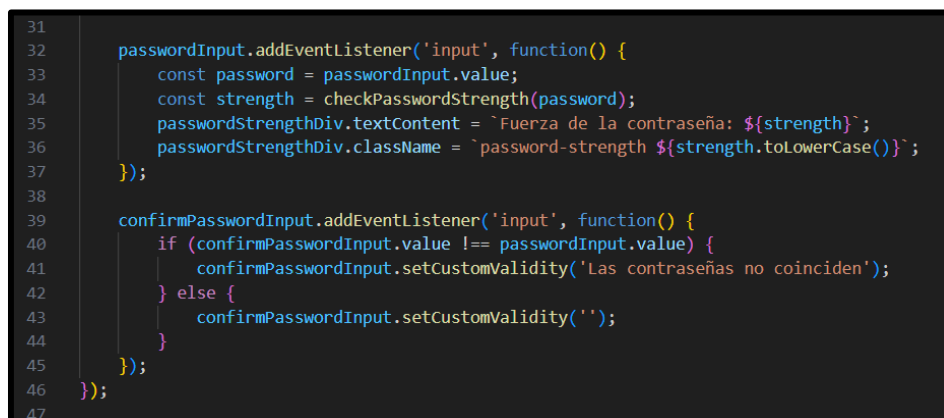
De lo contrario, es "Débil".



```
1 // script.js
2
3 document.addEventListener('DOMContentLoaded', function() {
4   const passwordInput = document.getElementById('password');
5   const confirmPasswordInput = document.getElementById('confirmPassword');
6   const passwordStrengthDiv = document.getElementById('passwordStrength');
7
8   function checkPasswordStrength(password) {
9     let strength = 'Débil';
10    const regex = {
11      lower: /[a-z]/,
12      upper: /[A-Z]/,
13      digit: /\d/,
14      special: /[!@#%&*().,?":{}|<>]/
15    };
16
17    if (password.length >= 8 &&
18        regex.lower.test(password) &&
19        regex.upper.test(password) &&
20        regex.digit.test(password) &&
21        regex.special.test(password)) {
22      strength = 'Fuerte';
23    } else if (password.length >= 6 &&
24        (regex.lower.test(password) || regex.upper.test(password)) &&
25        (regex.digit.test(password) || regex.special.test(password))) {
26      strength = 'Moderada';
27    }
28
29    return strength;
30  }
31 }
```

Ilustración 2 Funciones del código del archivo script.js

- **passwordInput.addEventListener('input')**: Cada vez que el usuario escribe en el campo de la contraseña, se llama a `checkPasswordStrength()`, y el resultado se muestra en el div de fortaleza de contraseña.
- **confirmPasswordInput.addEventListener('input')**: Este evento verifica si las contraseñas coinciden y muestra un mensaje si no lo hacen usando `setCustomValidity()`.



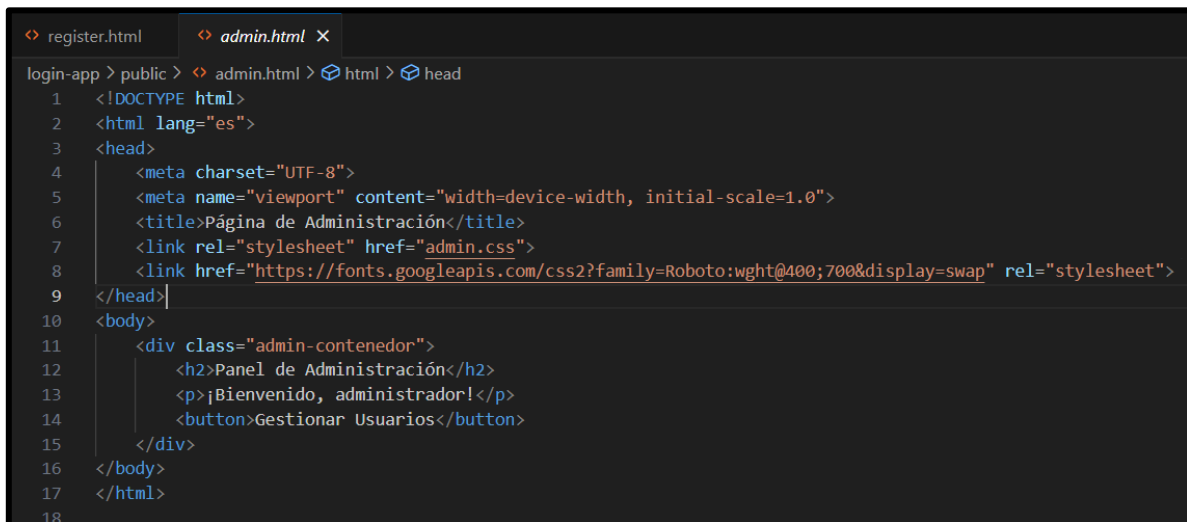
```
31
32 passwordInput.addEventListener('input', function() {
33   const password = passwordInput.value;
34   const strength = checkPasswordStrength(password);
35   passwordStrengthDiv.textContent = `Fuerza de la contraseña: ${strength}`;
36   passwordStrengthDiv.className = `password-strength ${strength.toLowerCase()}`;
37 });
38
39 confirmPasswordInput.addEventListener('input', function() {
40   if (confirmPasswordInput.value !== passwordInput.value) {
41     confirmPasswordInput.setCustomValidity('Las contraseñas no coinciden');
42   } else {
43     confirmPasswordInput.setCustomValidity('');
44   }
45 });
46
47 }
```

Ilustración 3 Código para mostrar la fuerza de las contraseñas y de que no coincidan

Paso 3. Creamos un archivo llamado admin.html:

El archivo "admin.html" es la página de administración que solo los usuarios con el rol de "Administrador" pueden acceder. Este archivo contiene lo siguiente:

- **<div class="admin-contenedor">**: Este div contiene todos los elementos principales de la página de administración. Es el contenedor central donde se encuentra el título, el mensaje de bienvenida y el botón.
- **<h2>**: Aquí se muestra el título "Panel de Administración", indicando que esta es una página especial para el administrador.
- **<p>**: Un mensaje simple de bienvenida: "¡Bienvenido, administrador!". Este mensaje refuerza que la persona que ha accedido a esta página tiene permisos administrativos.
- **<button>**: Un botón con la etiqueta "Gestionar Usuarios". Este botón probablemente estará asociado a alguna funcionalidad futura para administrar a los usuarios del sistema (aunque en este HTML no tiene una acción asignada aún).

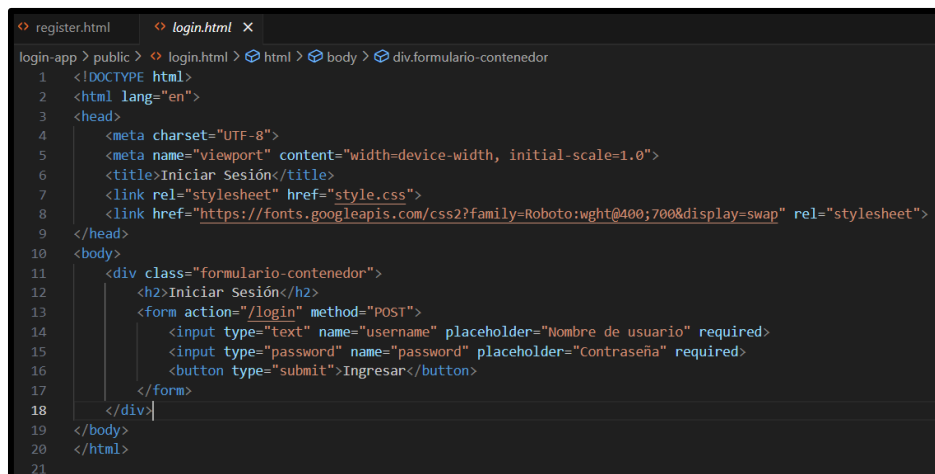


```
login-app > public > admin.html > html > head
1 <!DOCTYPE html>
2 <html lang="es">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Página de Administración</title>
7   <link rel="stylesheet" href="admin.css">
8   <link href="https://fonts.googleapis.com/css2?family=Roboto:wght@400;700&display=swap" rel="stylesheet">
9 </head>
10 <body>
11   <div class="admin-contenedor">
12     <h2>Panel de Administración</h2>
13     <p>¡Bienvenido, administrador!</p>
14     <button>Gestionar Usuarios</button>
15   </div>
16 </body>
17 </html>
18
```

Ilustración 4 Código del archivo admin.html

Paso 4. Creamos un archivo llamado login.html que es una página de inicio de sesión para que los usuarios puedan acceder a su cuenta en la aplicación. En el cuerpo del documento pusimos lo siguiente:

- **<div class="formulario-contenedor">**: Este div envuelve todo el formulario de inicio de sesión y se utiliza para aplicar estilos. Este contenedor sirve para organizar y centrar el formulario en la página.
- **action="/login" method="POST"**: Indica que los datos del formulario se enviarán al servidor a la ruta /login usando el método POST. Este método envía datos de forma segura (como las contraseñas) y permite al servidor procesarlos.
- **<input type="text" name="username" placeholder="Nombre de usuario" required>**: Es un campo de texto donde el usuario debe ingresar su nombre de usuario. El atributo required asegura que el campo debe estar lleno antes de enviar el formulario.
- **<input type="password" name="password" placeholder="Contraseña" required>**: Es un campo de tipo "password", que oculta el texto mientras se escribe, para la contraseña del usuario. También incluye required.
- **<button type="submit">Ingresar</button>**: Es el botón de envío del formulario. Al hacer clic en este botón, el formulario se envía al servidor para verificar las credenciales ingresadas.



```
login-app > public > login.html > html > body > div.formulario-contenedor
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4    <meta charset="UTF-8">
5    <meta name="viewport" content="width=device-width, initial-scale=1.0">
6    <title>Iniciar Sesión</title>
7    <link rel="stylesheet" href="style.css">
8    <link href="https://fonts.googleapis.com/css2?family=Roboto:wght@400;700&display=swap" rel="stylesheet">
9  </head>
10 <body>
11   <div class="formulario-contenedor">
12     <h2>Iniciar Sesión</h2>
13     <form action="/login" method="POST">
14       <input type="text" name="username" placeholder="Nombre de usuario" required>
15       <input type="password" name="password" placeholder="Contraseña" required>
16       <button type="submit">Ingresar</button>
17     </form>
18   </div>
19 </body>
20 </html>
21
```

Ilustración 5 Código del archivo login.html

Paso 5. Creamos un archivo llamado `profile` que es una página que muestra el perfil del usuario una vez que ha iniciado sesión en la aplicación.

En la sección del `body` se hizo lo siguiente:

- **`<div class="perfil-contenedor">`**: Este `div` sirve como contenedor principal para toda la información y los botones del perfil. Es útil para aplicar estilos de forma conjunta y organizar el contenido en un bloque centrado en la página.
- **`<h1>Bienvenido a tu Perfil</h1>`**: Es un encabezado principal que da la bienvenida al usuario a su página de perfil. El `h1` se utiliza para el título más importante de la página.
- **`<p>Esta es tu página de perfil personal.</p>`**: Es un párrafo que describe la página al usuario. Sirve como una pequeña introducción para saber dónde se encuentra.
- **`<div class="perfil-opciones">`**: Este `div` contiene los botones para las opciones relacionadas con el perfil. Aquí se incluyen botones que no tienen funcionalidad por el momento, pero podrías enlazarlos a futuras páginas:
- **`<button>Editar Perfil</button>`**: Muestra un botón para editar el perfil del usuario.
- **`<button>Configuración</button>`**: Otro botón que podría llevar a una página de configuración de la cuenta.
- **`<form action="/logout" method="POST">`**: Este formulario es responsable de cerrar la sesión del usuario. Al hacer clic en el botón, el formulario envía una solicitud `POST` a la ruta `/logout` para cerrar la sesión.
- **`<button type="submit">Cerrar Sesión</button>`**: Es un botón que al ser presionado enviará el formulario, lo que permite cerrar la sesión y redirigir al usuario fuera de su perfil.

```

login-app > public > <> profile.html > html > body > div.perfil-contenedor > form
1  <!DOCTYPE html>
2  <html lang="es">
3  <head>
4    <meta charset="UTF-8">
5    <meta name="viewport" content="width=device-width, initial-scale=1.0">
6    <title>Perfil de Usuario</title>
7    <link rel="stylesheet" href="styleprofile.css">
8  </head>
9  <body>
10   <div class="perfil-contenedor">
11     <h1>Bienvenido a tu Perfil</h1>
12     <p>Esta es tu página de perfil personal.</p>
13     <div class="perfil-opciones">
14       <button>Editar Perfil</button>
15       <button>Configuración</button>
16     </div>
17     <form action="/logout" method="POST">
18       <button type="submit">Cerrar Sesión</button>
19     </form>
20   </div>
21 </body>
22 </html>

```

Ilustración 6 Código del archivo profile.html

Paso 6. Cada página tiene su propio archivo CSS para darle estilo:

style.css para el formulario de registro e inicio de sesión.

styleprofile.css para la página de perfil.

admin.css para la página de administración.

Se utiliza una imagen de fondo para mejorar la apariencia visual y se han agregado efectos como transparencia a los formularios para que el fondo sea visible pero no interrumpa la lectura.

Paso 7. Para correr la aplicación, ejecutamos desde la terminal los siguientes comandos:

“npm install”, “npm install express” y “npm list express”.

```

Server is running on port 3000
PS C:\Users\luzar\OneDrive\Documentos\TECNOLOGICO\SEPTIMO SEMESTRE\SEGURIDAD Y VIRTUALIZACIÓN\Practica2\Ejercicio1\login-app> npm install
up to date, audited 203 packages in 3s
28 packages are looking for funding
  run `npm fund` for details
found 0 vulnerabilities
PS C:\Users\luzar\OneDrive\Documentos\TECNOLOGICO\SEPTIMO SEMESTRE\SEGURIDAD Y VIRTUALIZACIÓN\Practica2\Ejercicio1\login-app> npm install express
up to date, audited 203 packages in 5s
28 packages are looking for funding
  run `npm fund` for details
found 0 vulnerabilities
PS C:\Users\luzar\OneDrive\Documentos\TECNOLOGICO\SEPTIMO SEMESTRE\SEGURIDAD Y VIRTUALIZACIÓN\Practica2\Ejercicio1\login-app> npm list express
luzar@ C:\Users\luzar
└─ express@4.19.2
PS C:\Users\luzar\OneDrive\Documentos\TECNOLOGICO\SEPTIMO SEMESTRE\SEGURIDAD Y VIRTUALIZACIÓN\Practica2\Ejercicio1\login-app>

```

Ilustración 7 Instalación de npm

Paso 8. Una vez hecho lo anterior ejecutamos el siguiente comando: `node app.js`.

Donde nos dice que el servidor se encuentra corriendo en el puerto 3000.

```

PS C:\Users\luzar\OneDrive\Documentos\TECNOLOGICO\SEPTIMO SEMESTRE\SEGURIDAD Y VIRTUALIZACIÓN\Practica2\Ejercicio1\login-app> node app.js
Server is running on port 3000

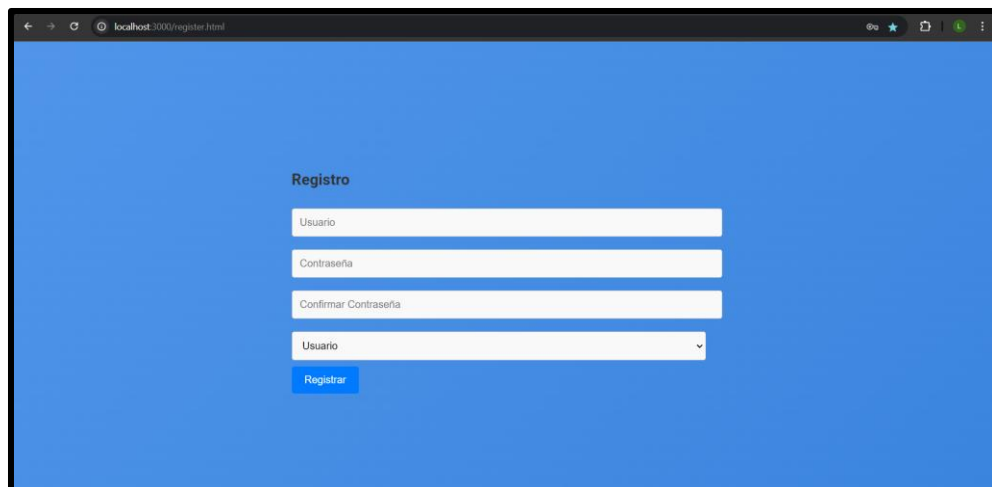
```

Ilustración 8 Ejecución del comando de node app.js

Paso 9. Desde el navegador ponemos la siguiente dirección:

<http://localhost:3000/register.html>

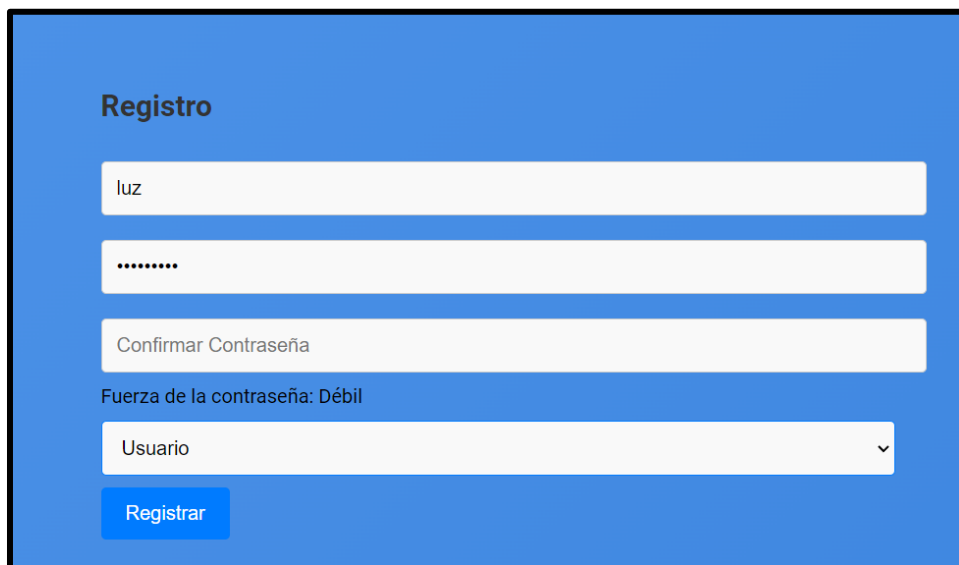
El cual nos mostrará la interfaz de registro:



The screenshot shows a web browser window with the address bar displaying 'localhost:3000/register.html'. The page has a solid blue background. In the center, there is a registration form titled 'Registro'. The form contains four input fields: 'Usuario', 'Contraseña', 'Confirmar Contraseña', and a dropdown menu labeled 'Usuario'. Below these fields is a blue button with the text 'Registrar'.

Ilustración 9 Página de registro

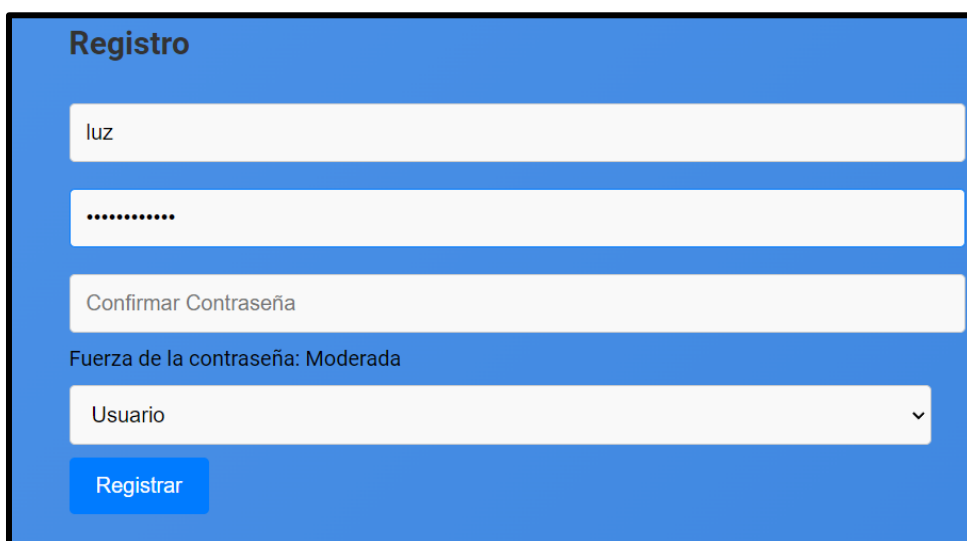
Paso 10. Ingreso el usuario y la contraseña, en donde en las siguientes capturas de pantalla se puede visualizar que conforme voy ingresando una contraseña me indica si la contraseña es débil, moderada o fuerte. También puedo seleccionar si quiero registrarme como usuario o como administrador, en este caso seleccioné la opción de usuario. Una vez dándole en la opción de registrar me enviará a la ventana de inicio de sesión.



The screenshot shows a registration form titled "Registro" on a blue background. It contains the following elements:

- A text input field with the value "luz".
- A password input field with masked characters "*****".
- A text input field with the placeholder "Confirmar Contraseña".
- A feedback message: "Fuerza de la contraseña: Débil".
- A dropdown menu with the selected option "Usuario".
- A blue button labeled "Registrar".

Ilustración 10 Fuerza de contraseña débil



The screenshot shows the same registration form as above, but with the following changes:

- The password input field now contains "*****" (the same as before).
- The feedback message has changed to: "Fuerza de la contraseña: Moderada".
- The dropdown menu still shows "Usuario".
- The "Registrar" button remains visible.

Ilustración 11 Fuerza de contraseña moderada

Registro

luz

.....

Confirmar Contraseña

Fuerza de la contraseña: Fuerte

Usuario ▾

Registrar

This registration form is set against a blue background. It includes a title 'Registro' at the top. Below it are four input fields: the first contains 'luz', the second is masked with '.....', the third is labeled 'Confirmar Contraseña', and the fourth is a dropdown menu labeled 'Usuario'. A feedback message 'Fuerza de la contraseña: Fuerte' is displayed above the dropdown. A blue 'Registrar' button is at the bottom.

Ilustración 12 Fuerza de contraseña fuerte

Registro

luz

.....

.....

Fuerza de la contraseña: Fuerte

Usuario ▾

Usuario

Administrador

The registration form is identical to the previous one, but the 'Usuario' dropdown menu is open, showing two options: 'Usuario' and 'Administrador'.

Ilustración 13 Selección de registro como usuario

Iniciar Sesión

Nombre de usuario


Contraseña

Ingresar

The login form is centered on a blue background. It has a title 'Iniciar Sesión' at the top. Below the title are two input fields: 'Nombre de usuario' and 'Contraseña'. A blue 'Ingresar' button is positioned below the second field.

Ilustración 14 Página de inicio de sesión

Paso 11. Estando en la ventana de inicio de sesión ingreso el usuario y contraseña con el que me registré anteriormente y posteriormente le doy en la opción de ingresar. Al darle en el botón de ingresar me enviará a la interfaz del usuario.



The image shows a login interface with a blue background. In the center is a white rounded rectangle containing the title "Iniciar Sesión". Below the title are two input fields: the first contains the text "luz", and the second contains a series of dots representing a password. Below these fields is a blue button with the text "Ingresar".

Ilustración 15 Inicio de sesión con usuario y contraseña

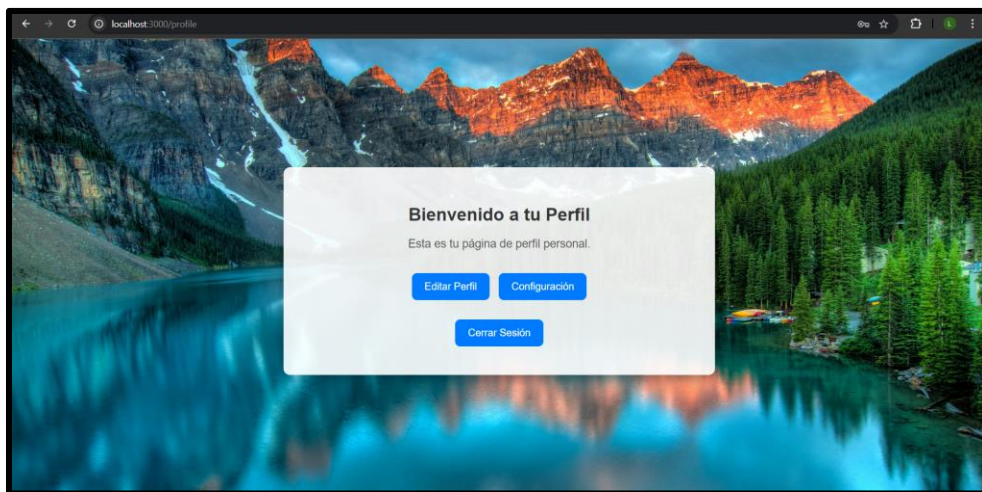


Ilustración 16 página del usuario

Paso 12. También puedo registrarme como administrador, por lo que si me dirijo a la página de registro y relleno los campos correspondientes y selecciono la opción como administrador, me enviará a la página de inicio de sesión.



Registro

maria

.....

.....

Fuerza de la contraseña: Fuerte

Administrador

Registrar

This is a registration form titled 'Registro' on a blue background. It contains four input fields: a text field with 'maria', a password field with masked characters, another password field with masked characters, and a dropdown menu currently showing 'Administrador'. Below these fields is a blue button labeled 'Registrar'. A label 'Fuerza de la contraseña: Fuerte' is positioned above the dropdown menu.

Ilustración 17 Registro como administrador



Iniciar Sesión

maria

.....

Ingresar

This is a login form titled 'Iniciar Sesión' on a blue background. It contains two input fields: a text field with 'maria' and a password field with masked characters. Below these fields is a blue button labeled 'Ingresar'.

Ilustración 18 Inicio de sesión como administrador

Una vez iniciando sesión me enviará a la página del administrador.

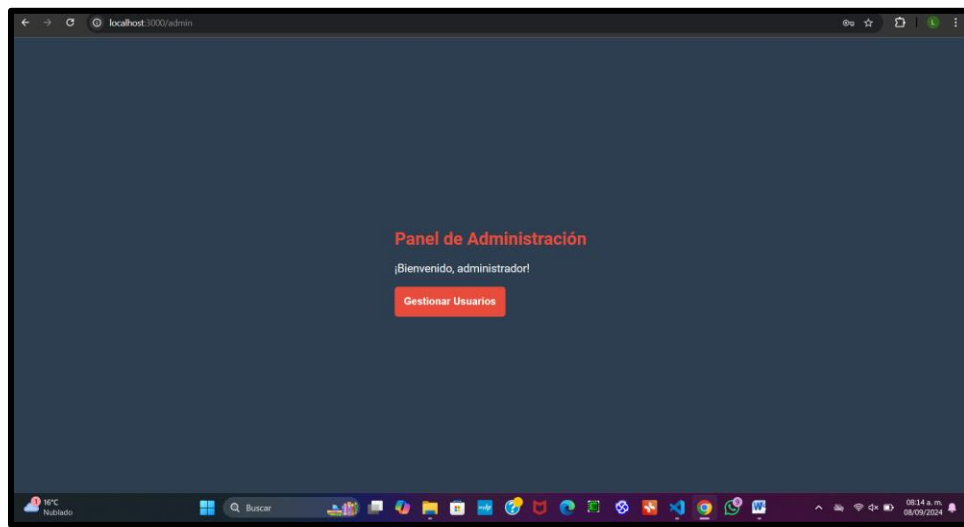


Ilustración 19 Página del administrador

EJERCICIO 2. Implementa un mecanismo de autorización que permita o deniegue el acceso a ciertas rutas de la aplicación, en este caso la página de perfil y la página de administración si en dado caso el usuario no ha iniciado sesión o no tiene el rol de administrador.

Paso 1. Configuramos los archivos estáticos donde se configuró Express para servir archivos estáticos desde la carpeta public.

```
app.use(express.static(path.join(__dirname, 'public'))  
app.use(express.urlencoded({ extended: true })));
```

Ilustración 20 Configuración de archivos estáticos

Paso 2. Después configuramos el inicio de sesión añadiendo middleware para manejar sesiones, utilizando express-session.

```
// Configuración de sesión  
app.use(session({  
  secret: 'mySecret',  
  resave: false,  
  saveUninitialized: true  
}));
```

Ilustración 21 Configuración de inicio de sesion

Paso 3. Integramos rutas de autenticación donde se añadieron las rutas de autenticación definidas en el archivo routes/auth.js.

```
// Rutas de autenticación  
const authRoutes = require('./routes/auth');  
app.use('/', authRoutes);
```

Ilustración 22 Rutas de autenticación

Paso 4. Nos dirigimos al siguiente archivo con el nombre auth.js creando middleware isAuthenticated para verificar si un usuario está autenticado.

```

Tabnine: Edit | Test | Explain | Document | Ask
function isAuthenticated(req, res, next) {
  if (req.session.user) {
    return next();
  }
  res.redirect('/login');
}

```

Ilustración 23 Verificación de usuario autenticado

Paso 5. Creamos un middleware isAdmin para permitir el acceso solo a administradores.

```

function isAdmin(req, res, next) {
  if (req.session.user && req.session.role === 'admin') {
    return next();
  }
  res.send('Acceso Denegado. Solo administradores.');
```

Ilustración 24 Código de acceso a administradores

Paso 6. Agregamos una ruta POST /register para manejar el registro de nuevos usuarios. Incluye verificación de contraseñas y validación de existencia de usuario.

```

router.post('/register', (req, res) => {
  const { username, password, confirmPassword, role } = req.body;

  if (password !== confirmPassword) {
    return res.send('Las contraseñas no coinciden');
  }

  const userExists = users.find(user => user.username === username);
  if (userExists) {
    return res.send('El usuario ya existe');
  }

  const newUser = { username, password, role };
  users.push(newUser);

  res.redirect('/login');
});

```

Ilustración 25 Código de registro de nuevos usuarios

Paso 7. Después agregamos la ruta POST /login para manejar el inicio de sesión y redirigir al usuario basado en su rol.

```

router.post('/login', (req, res) => {
  const { username, password } = req.body;

  const user = users.find(user => user.username === username && user.password === password);

  if (user) {
    req.session.user = user.username;
    req.session.role = user.role;

    if (user.role === 'admin') {
      res.redirect('/admin');
    } else {
      res.redirect('/profile');
    }
  } else {
    res.send('Nombre de usuario o contraseña inválidos');
  }
});

```

Ilustración 26 Login de inicio de sesión

Paso 8. Realizamos la protección de las rutas GET /admin y GET /profile utilizando los middlewares isAuthenticated y isAdmin.

```

router.get('/admin', isAuthenticated, isAdmin, (req, res) => {
  res.sendFile(path.join(__dirname, '../public/admin.html'));
});

router.get('/profile', isAuthenticated, (req, res) => {
  res.sendFile(path.join(__dirname, '../public/profile.html'));
});

```

Ilustración 27 Código de protección de rutas

Paso 9. Agregamos las rutas para la página de inicio, inicio de sesión y registro se configuraron para servir archivos HTML estáticos.

```

router.get('/', (req, res) => {
  res.sendFile(path.join(__dirname, '../public/index.html'));
});

router.get('/login', (req, res) => {
  res.sendFile(path.join(__dirname, '../public/login.html'));
});

router.get('/register', (req, res) => {
  res.sendFile(path.join(__dirname, '../public/register.html'));
});

```

Ilustración 28 inicio de sesión y registro se configuraron

Enlaces para Pruebas.

El primer enlace proporcionado <http://localhost:3000/profile> es para mostrarnos el perfil del usuario. En este caso denegará el acceso al usuario y lo redijirá al inicio de sesión para pueda acceder.

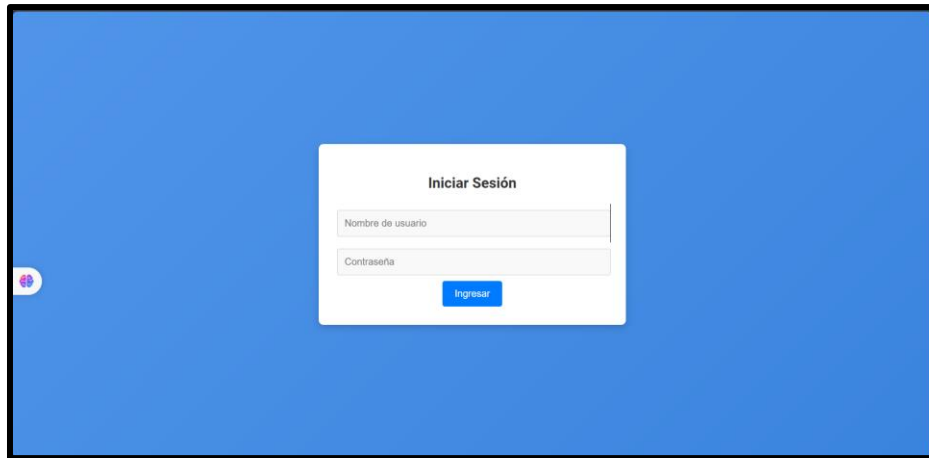


Ilustración 29 Página de inicio de sesión

El segundo enlace proporcionado <http://localhost:3000/admin> es para mostrarnos el perfil del administrador. De igual forma el acceso se negará lo que mostrará un mensaje de acceso denegado.

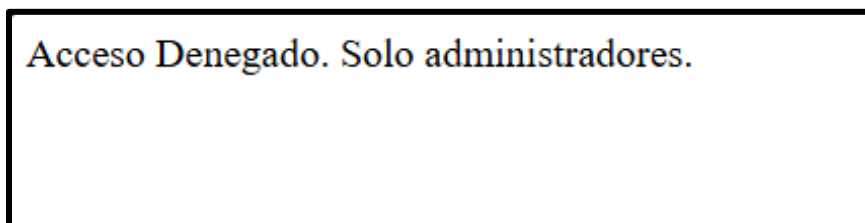


Ilustración 30 Mensaje denegado

EJERCICIO 3. Implementa un mecanismo de autenticación que permita a los usuarios registrarse, iniciar sesión y cerrar sesión.

Cambio realizado: Manejo de rutas para el registro, inicio de sesión, así como el cierre de sesión.

Paso 1. Importamos los módulos necesarios y se crea un enrutador de Express.

```
const express = require('express');
const path = require('path');
const router = express.Router();
```

Ilustración 31 Código de enrutador de express

Paso 2. Creamos un arreglo para simular una base de datos de usuarios.

```
const users = [];
```

Ilustración 32 Creación de arreglo

Paso 3. Validamos que las contraseñas coincidan, verificando si el usuario ya existe, agregamos un nuevo usuario a la base de datos simulada y redirige al usuario a la página de inicio de sesión.

```
Tabnine: Edit | Test | Explain | Document | Ask
router.post('/register', (req, res) => {
  const { username, password, confirmPassword, role } = req.body;

  if (password !== confirmPassword) {
    | return res.send('Las contraseñas no coinciden');
  }

  const userExists = users.find(user => user.username === username);
  if (userExists) {
    | return res.send('El usuario ya existe');
  }

  const newUser = { username, password, role };
  users.push(newUser);

  res.redirect('/login');
});
```

Ilustración 33 Validación de contraseñas

Paso 4. Verificamos las credenciales del usuario y, si son correctas, establece la sesión del usuario y redirige según el rol (administrador o usuario regular). Si las credenciales son incorrectas, muestra un mensaje de error.

```
Tabnine: Edit | Test | Explain | Document | Ask
router.post('/login', (req, res) => {
  const { username, password } = req.body;

  const user = users.find(user => user.username === username && user.password === password);

  if (user) {
    req.session.user = user.username;
    req.session.role = user.role;

    if (user.role === 'admin') {
      res.redirect('/admin');
    } else {
      res.redirect('/profile');
    }
  } else {
    res.send('Nombre de usuario o contraseña inválidos');
  }
});
```

Ilustración 34 Código de verificación de credenciales

Paso 5. Página de registro para los usuarios, para que puedan registrarse.

```
Tabnine: Edit | Test | Explain | Document | Ask
router.get('/register', (req, res) => {
  res.sendFile(path.join(__dirname, '../public/register.html'));
});
Tabnine: Edit | Test | Explain | Document | Ask
```

Ilustración 35 Página de registro de usuarios

Paso 6. Si el usuario está autenticado, envía el archivo profile.html. De lo contrario, redirige al usuario a la página de inicio de sesión.

```
Tabnine: Edit | Test | Explain | Document | Ask
router.get('/profile', (req, res) => {
  if (req.session.user) {
    res.sendFile(path.join(__dirname, '../public/profile.html'));
  } else {
    res.redirect('/login.html');
  }
});
```

Ilustración 36 código de verificación de usuario autenticado

Paso 7. Verificamos si el usuario está autenticado y tiene el rol de administrador. Si es así, envía el archivo admin.html; de lo contrario, muestra un mensaje de acceso denegado.

```
router.get('/admin', (req, res) => {
  if (req.session.user && req.session.role === 'admin') {
    res.sendFile(path.join(__dirname, '../public/admin.html'));
  } else {
    res.send('Acceso denegado: necesitas privilegios de administrador');
  }
});
```

Ilustración 37 verificación de usuario autenticado como administrador

Paso 8. Si el usuario cierra sesión nos redigirá a la página de inicio.

```
router.post('/logout', (req, res) => {
  req.session.destroy();
  res.redirect('/');
});

module.exports = router;
```

Ilustración 38 Código para cerrar sesión

Creación de un archivo package.json

Paso 10. Declaramos el nombre, versión y descripción del proyecto. Indica app.js como el archivo principal del proyecto.

```
{
  "name": "my-authentication-app",
  "version": "1.0.0",
  "description": "Sistema de autenticación simple con Node.js",
  "main": "app.js",
```

Ilustración 39 Nombre, versión y descripción del proyecto

Paso 11. Definimos un script para iniciar la aplicación usando Node.js. Se puede ejecutar con el comando npm start.

```

"scripts": {
  | "start": "node app.js"
  },

```

Ilustración 40 Script para iniciar la aplicación

Paso 12. Especificamos las dependencias necesarias para el proyecto: express para el servidor web y express-session para manejar sesiones.

```

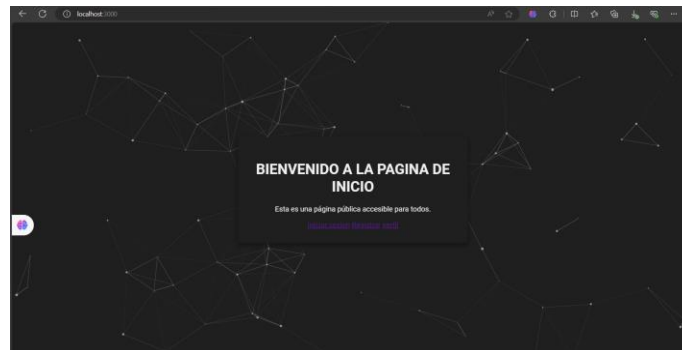
"dependencies": {
  | "express": "^4.18.2",
  | "express-session": "^1.17.3"
  }
}

```

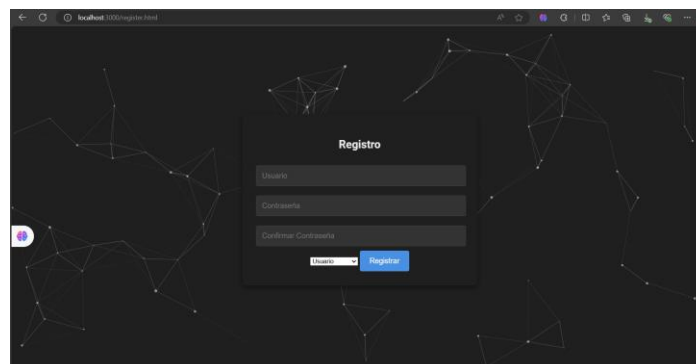
Ilustración 41 Dependencias para el proyecto

Resultados

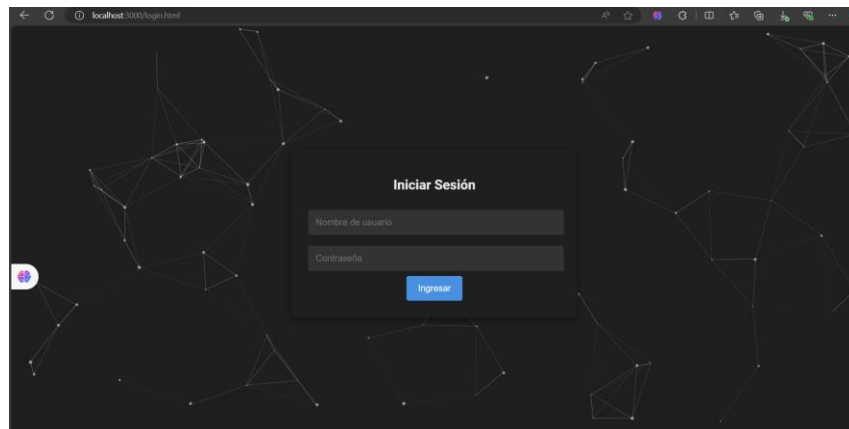
Página de inicio:



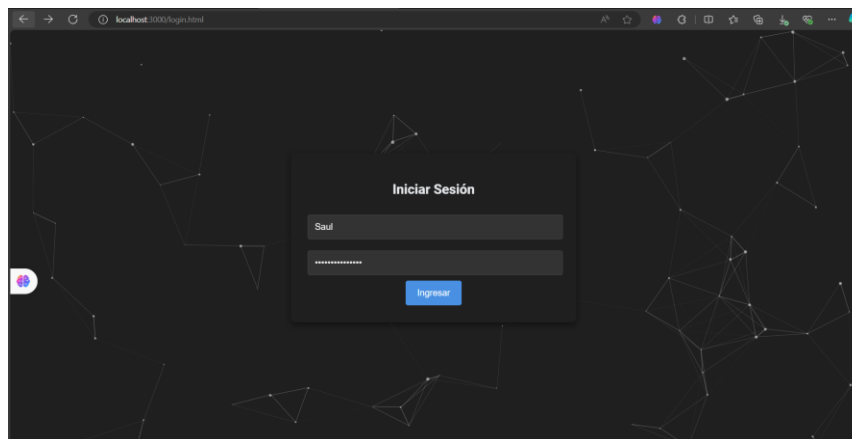
Registros de usuarios:



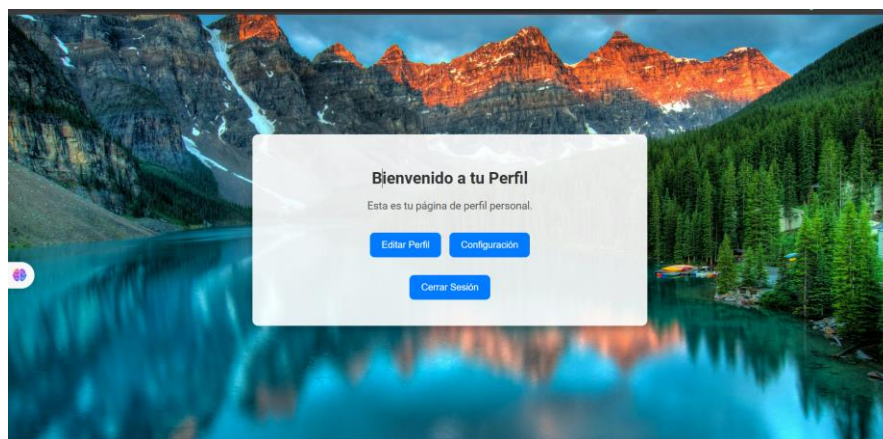
Inicio de sesión:



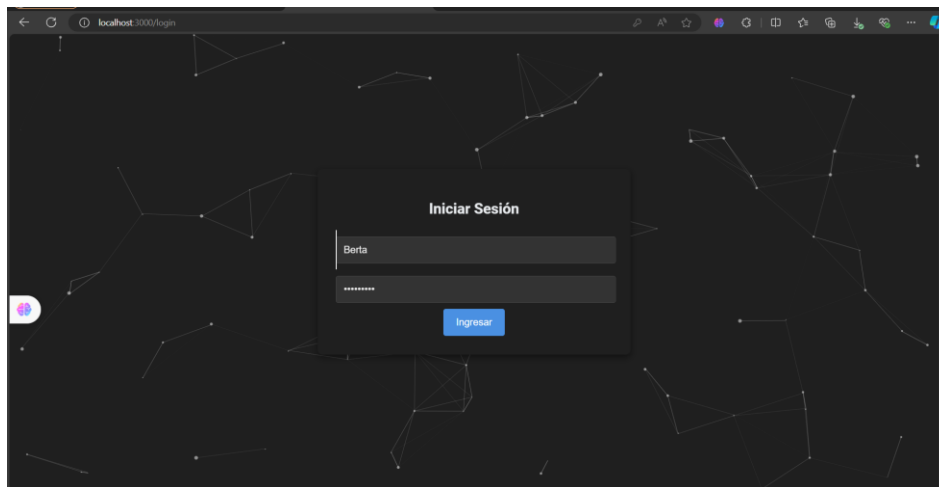
Inicio de sesión del usuario:



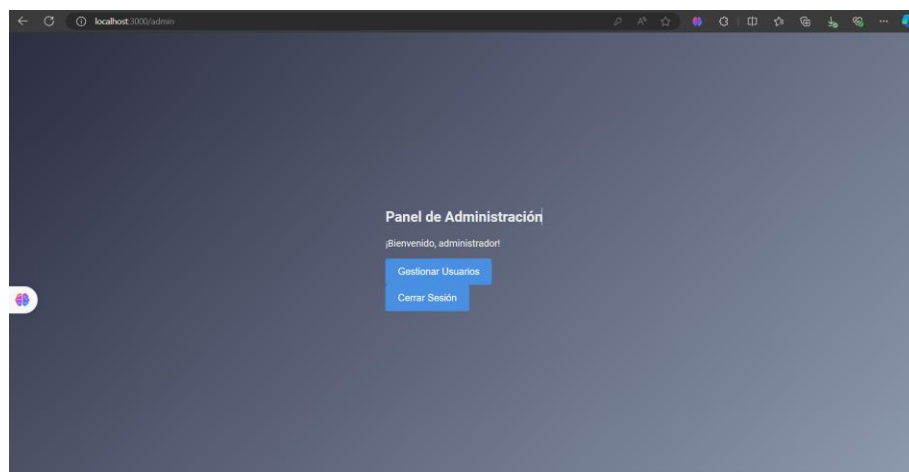
Perfil del usuario:



Inicio sesión administrador:



Perfil administrador:



EJERCICIO 4. Implementa un mecanismo para cerrar sesión de un usuario si ha pasado un tiempo determinado sin actividad de 5 mins.

Tanto lo que se realizó en el archivo prefile.html también se realizó lo mismo en el archivo admin.html.

Paso 1. Creamos un nuevo script que va a contener la inactividad de uso de la página.

```
let inactivityTime = function () {
```

Ilustración 42 Script de inactividad

Paso 2: Definimos una variable time el cual es un identificador de temporizador que se usará para almacenar el resultado de setTimeout, lo que permitirá reiniciar o cancelar el temporizador cuando el usuario interactúe con la página.

```
let time;
```

Ilustración 43 Identificador de temporizador

Paso 3. Esta línea define la URL a la que el usuario será redirigido cuando se considere inactivo, en este caso, /login. Esta URL corresponde a la página o ruta donde se manejará la lógica de cierre de sesión.

```
const logoutUrl = '/login';
```

Ilustración 44 Ruta de cierre de sesión

Paso 4. Agregamos la función handleLogout el cual se va a ejecutar cuando el temporizador alcanza su límite (cuando el usuario no interactúa con la página). Usa window.location.href para redirigir al usuario a la URL de cierre de sesión definida anteriormente (/login).

```
const handleLogout = () => {  
  window.location.href = logoutUrl;  
};
```

Ilustración 45 Función a ejecutar cuando el temporizador alcance su límite

Paso 5. Agregamos la función `resetTimer`, el cual será el que detectará el comportamiento de la inactividad. Aquí están los detalles de su funcionamiento:

1. `clearTimeout(time)`: Si existe un temporizador activo, lo cancela, asegurando que no se ejecute antes de tiempo.
2. `setTimeout(handleLogout, 5 * 60 * 1000)`: Luego, establece un nuevo temporizador que ejecutará la función `handleLogout` después de 5 minutos ($5 * 60 * 1000$ milisegundos). Esto significa que si el usuario no realiza ninguna acción en esos 5 minutos, se cerrará la sesión.

```
const resetTimer = () => {  
  clearTimeout(time);  
  time = setTimeout(handleLogout, 5 * 60 * 1000);  
};
```

Ilustración 46 Función para el comportamiento de inactividad

CONCLUSION

En conclusión, la práctica "Autorización y Autenticación" permitió poner en práctica los conceptos clave relacionados con la seguridad en aplicaciones web, específicamente en cuanto a la autenticación de usuarios y la autorización de acceso a rutas y recursos. A lo largo de los ejercicios realizados, se pudo observar la importancia de implementar mecanismos robustos que aseguren tanto la verificación de identidad como el control de permisos basados en roles.

La creación de un sistema de login y registro, la protección de páginas según el nivel de autorización, y la implementación de un cierre de sesión automático tras un período de inactividad son medidas esenciales para garantizar la seguridad de una aplicación. Además, la verificación de la seguridad de contraseñas añade una capa adicional de protección, lo que refuerza la fiabilidad del sistema en términos de confidencialidad y acceso.

En conjunto, esta práctica brindó una comprensión sólida de cómo aplicar la autenticación y autorización de manera efectiva en una aplicación web, fortaleciendo la seguridad y mejorando la experiencia del usuario mediante la gestión adecuada de roles y accesos.

INVESTIGACION SERVICIOS DE AUTENTICACIÓN

LDAP (LIGHTWEIGHT DIRECTORY ACCESS PROTOCOL)

Es un protocolo ligero utilizado para acceder, gestionar y mantener servicios de información de directorio. Un directorio es una base de datos optimizada para realizar búsquedas rápidas y eficientes, que contiene información estructurada de manera jerárquica. LDAP fue diseñado como una versión simplificada del estándar de directorios X.500, con el objetivo de ser más ligero y adecuado para operar sobre redes TCP/IP.

PRINCIPIOS CLAVE DE LDAP

Jerarquía de Directorios: LDAP organiza la información en una estructura jerárquica similar a un árbol, lo que se conoce como el Árbol de Información de Directorios (DIT, Directory Information Tree). Los nodos de este árbol pueden representar entradas como usuarios, grupos, dispositivos, impresoras, servidores, y más. Cada entrada en el directorio tiene un DN (Distinguished Name), que es su identificador único dentro del árbol.

Por ejemplo, una entrada para un usuario podría estar organizada como:

dn: uid=jdoe, ou=People, dc=example, dc=com

Esquema de LDAP: LDAP utiliza un esquema que define qué tipo de objetos y atributos puede contener el directorio. Esto incluye clases de objetos (como “person” para un usuario) y los atributos que pueden tener (como “cn” para nombre común o mail para la dirección de correo electrónico). El esquema asegura que las entradas tengan una estructura coherente.

Operaciones Básicas: LDAP admite varias operaciones para interactuar con el directorio, las más comunes son:

- Bind (enlazar): Autentica al cliente con el servidor LDAP.
- Search (buscar): Realiza búsquedas en el directorio con base en ciertos criterios.

- **Compare (comparar):** Verifica si un atributo específico de una entrada tiene un valor determinado.
- **Modify (modificar):** Permite actualizar los atributos de una entrada.
- **Add (añadir) y Delete (borrar):** Añade o elimina entradas en el directorio.
- **Unbind (desenlazar):** Termina la sesión del cliente con el servidor.

Autenticación: LDAP puede ser utilizado como un servicio de autenticación centralizado, donde las aplicaciones y los sistemas consultan al servidor LDAP para verificar las credenciales de usuario (nombre de usuario y contraseña). Una vez autenticado, LDAP puede proporcionar información adicional sobre el usuario, como roles o permisos.

TIPOS DE AUTENTICACIÓN EN LDAP

Anonymous Bind: Autenticación anónima (sin credenciales).

Simple Bind: Autenticación básica mediante el envío de credenciales en texto plano (generalmente usado con LDAP sobre SSL/TLS para seguridad).

SASL Bind: Autenticación más segura mediante el uso de mecanismos SASL (como Kerberos).

Seguridad: De forma predeterminada, LDAP no cifra las comunicaciones, lo que puede suponer un riesgo de seguridad al enviar contraseñas y datos sensibles. Para mitigar este riesgo, es común utilizar LDAPS (LDAP Secure), que implementa LDAP sobre una conexión SSL/TLS, cifrando la comunicación entre el cliente y el servidor.

Además, LDAP puede implementarse junto con otros mecanismos de seguridad, como Kerberos, para proporcionar autenticación segura sin el intercambio directo de credenciales.

Integración con Otros Servicios: LDAP es ampliamente utilizado en muchas implementaciones de directorios, como:

Microsoft Active Directory (AD): Aunque Active Directory tiene algunas extensiones propietarias, es compatible con LDAP para la autenticación y acceso a la información de usuarios.

OpenLDAP: Una implementación de código abierto de LDAP, ampliamente utilizada en entornos Unix/Linux.

Red Hat Directory Server, ApacheDS, Oracle Directory Server, entre otros.

EJEMPLO DE CASOS DE USO

Autenticación Centralizada: En una organización grande, LDAP puede actuar como un servicio de directorio centralizado donde se almacena toda la información sobre los usuarios, incluidas sus credenciales, roles y permisos. Los sistemas de correo electrónico, aplicaciones web y otros servicios pueden consultar este directorio para autenticar a los usuarios y proporcionar acceso controlado.

Consulta de Información de Usuarios: Muchas aplicaciones empresariales, como los sistemas CRM o sistemas de administración de redes, pueden realizar búsquedas en un servidor LDAP para recuperar información de usuarios, grupos o dispositivos, y así integrar la información del directorio con otras aplicaciones.

Gestión de Dispositivos: LDAP no solo almacena información sobre usuarios, sino también sobre dispositivos como impresoras, servidores, y más. Esto facilita la administración centralizada y permite que diferentes sistemas accedan a un único repositorio de información.

VENTAJAS DE LDAP

Escalabilidad: Puede manejar grandes volúmenes de entradas en la base de datos de directorios, y es adecuado para grandes organizaciones.

Compatibilidad Multiplataforma: LDAP es compatible con diferentes sistemas operativos y aplicaciones, lo que lo hace versátil en entornos heterogéneos.

Centralización: Permite administrar todos los usuarios y recursos desde un único punto, simplificando la administración de identidades y accesos.

DESVENTAJAS DE LDAP

Seguridad Inherente Limitada: LDAP no cifra la información por defecto, lo que puede hacer vulnerables las credenciales si no se utiliza LDAPS o TLS.

Complejidad: La implementación y configuración de un servidor LDAP puede ser complicada, especialmente cuando se combina con otros protocolos o servicios de autenticación como Kerberos.

Dependencia de la Infraestructura: LDAP depende de la integridad de la red y los servidores para funcionar correctamente; si el servidor LDAP falla, los servicios que dependen de él también pueden verse afectados.

EJEMPLO DE UNA BÚSQUEDA LDAP

Si queremos buscar a un usuario en el servidor LDAP cuyo nombre sea "Juan Pérez", podemos realizar una búsqueda con un cliente LDAP. El siguiente ejemplo de consulta LDAP podría ejecutarse:

```
ldapsearch -x -LLL -H ldap://servidorldap.com -b "dc=example,dc=com" "(cn=Juan Pérez)"
```

Esto busca dentro de la base de datos de LDAP cualquier entrada que tenga un nombre común (cn) igual a "Juan Pérez" en la base dc=example,dc=com.

RADIUS (REMOTE AUTHENTICATION DIAL-IN USER SERVICE)

Es un protocolo de red que proporciona servicios de autenticación, autorización y contabilidad (AAA) para usuarios que acceden a una red o servicio remoto. Originalmente desarrollado por Livingston Enterprises en 1991 para gestionar el acceso remoto a través de redes de acceso telefónico, se ha convertido en un estándar ampliamente utilizado para la gestión del acceso a redes, especialmente en entornos de redes empresariales y proveedores de servicios de Internet (ISP).

FUNCIONES PRINCIPALES DE RADIUS:

Autenticación (Authentication): RADIUS verifica la identidad de los usuarios que intentan acceder a la red, asegurándose de que sus credenciales sean válidas. Esto puede implicar la validación de un nombre de usuario y contraseña, o bien otros factores de autenticación, como certificados digitales, tokens de hardware o autenticación multifactorial (MFA).

Autorización (Authorization): Después de autenticar a un usuario, RADIUS determina los permisos o los derechos que ese usuario tiene en la red. Por ejemplo, puede decidir si el usuario tiene acceso a ciertos recursos o qué nivel de servicio se les debe otorgar (como acceso a internet limitado o acceso completo).

Contabilidad (Accounting): RADIUS también realiza un seguimiento de las actividades de los usuarios mientras están conectados. Esto incluye detalles como la duración de la sesión, los recursos utilizados, y otra información de uso que puede ser útil para la auditoría, facturación o gestión de la red.

ARQUITECTURA DE RADIUS:

RADIUS utiliza una arquitectura cliente-servidor. Los elementos principales son:

Cliente RADIUS: El cliente RADIUS generalmente es un NAS (Network Access Server), que puede ser un dispositivo de red como un router, un switch, un controlador de acceso inalámbrico o un servidor VPN. Este cliente recibe las credenciales de los usuarios y envía una solicitud al servidor RADIUS para autenticar y autorizar la conexión.

Servidor RADIUS: El servidor RADIUS recibe las solicitudes del cliente, valida las credenciales del usuario (comparándolas con una base de datos de autenticación, como un directorio LDAP, Active Directory, o una base de datos local), y toma una decisión sobre si se permite o no el acceso.

El servidor puede estar conectado a varias bases de datos de usuarios para verificar las credenciales.

Base de Datos de Usuarios: El servidor RADIUS puede consultar diversas fuentes de datos para validar la identidad de los usuarios. Estas bases pueden incluir archivos locales, directorios LDAP, bases de datos SQL o sistemas de autenticación como Active Directory.

PROCESO DE AUTENTICACIÓN DE RADIUS

El proceso básico de autenticación con RADIUS se desarrolla de la siguiente manera:

Solicitud de Acceso: El usuario intenta acceder a la red proporcionando sus credenciales (por ejemplo, un nombre de usuario y una contraseña). Estas credenciales son recogidas por el dispositivo cliente RADIUS, como un router, switch, o controlador inalámbrico.

Envío de Solicitud al Servidor RADIUS: El cliente RADIUS (el NAS) envía una solicitud al servidor RADIUS. Esta solicitud incluye las credenciales del usuario y

otra información relevante, como la dirección IP del cliente, el puerto desde el que se conecta, y el tipo de conexión (VPN, inalámbrica, etc.).

Validación en el Servidor RADIUS: El servidor RADIUS recibe la solicitud y verifica las credenciales comparándolas con su base de datos de usuarios. Si las credenciales son correctas, el servidor responde con un mensaje de Access-Accept (acceso permitido). Si las credenciales son incorrectas, el servidor envía un mensaje de Access-Reject (acceso denegado).

Autorización y Contabilidad: Si el acceso es aceptado, el servidor RADIUS también puede proporcionar parámetros adicionales de autorización, como el tiempo máximo de sesión o el ancho de banda permitido. Además, RADIUS puede iniciar el proceso de contabilidad, donde registra detalles de la sesión como el tiempo de conexión y el uso de recursos.

Finalización de la Sesión: Cuando el usuario finaliza la sesión o es desconectado, el cliente RADIUS envía un mensaje de contabilidad de cierre de sesión al servidor RADIUS para registrar el tiempo de conexión y otros detalles.

PROTOCOLOS Y SEGURIDAD

RADIUS utiliza el protocolo UDP (User Datagram Protocol) para la comunicación entre el cliente y el servidor. Los mensajes de autenticación y autorización se transmiten a través de los puertos 1812 (por defecto) o 1645 en algunas implementaciones más antiguas. La contabilidad se maneja a través del puerto 1813 o 1646.

En cuanto a la seguridad, RADIUS cifra solo las contraseñas del usuario, pero no cifra el resto del paquete, lo que podría exponer información sensible si se interceptan los datos. Por ello, muchas implementaciones de RADIUS se complementan con IPsec o TLS para proteger la comunicación entre el cliente y el servidor.

CARACTERÍSTICAS DE RADIUS

Escalabilidad: RADIUS es muy escalable y es utilizado en redes de cualquier tamaño, desde pequeñas oficinas hasta grandes redes de telecomunicaciones. Su arquitectura cliente-servidor permite que múltiples dispositivos de red consulten un único servidor RADIUS centralizado, lo que simplifica la gestión de autenticación y autorización.

Interoperabilidad: Al ser un estándar ampliamente adoptado, RADIUS es compatible con una amplia gama de dispositivos y software de diferentes proveedores. Es comúnmente utilizado en redes inalámbricas (Wi-Fi), redes de acceso remoto (VPN), y redes de proveedores de servicios (ISP).

Compatibilidad con Diferentes Métodos de Autenticación: RADIUS soporta una variedad de mecanismos de autenticación, incluyendo:

- Autenticación por contraseña (PAP, CHAP, MS-CHAP).
- Certificados digitales para autenticación más fuerte.
- Autenticación multifactorial (MFA), que puede requerir una combinación de contraseñas, tokens y otros factores.

Autorización Basada en Políticas: A través de RADIUS, las organizaciones pueden aplicar políticas de autorización. Esto significa que diferentes usuarios pueden tener diferentes niveles de acceso o privilegios basados en su rol o grupo. Por ejemplo, un administrador de red podría tener acceso completo, mientras que un usuario regular podría estar restringido a ciertas partes de la red.

Contabilidad para Auditoría y Facturación: Las funciones de contabilidad permiten a las organizaciones llevar un registro detallado de la actividad de los usuarios, lo que es útil para auditorías de seguridad, facturación basada en el uso o para realizar análisis de capacidad de la red.

VENTAJAS DE RADIUS

Centralización de la Autenticación: Facilita la gestión de usuarios en grandes redes al centralizar la autenticación y autorización, eliminando la necesidad de gestionar credenciales en cada dispositivo individual.

Soporte para Diversos Métodos de Acceso: RADIUS puede autenticar usuarios que acceden a través de diferentes tecnologías, como redes inalámbricas (Wi-Fi), VPNs, conexiones por cable, o acceso remoto.

Amplia Adopción: Es un protocolo estándar y ampliamente soportado por una gran cantidad de dispositivos y proveedores de software, lo que facilita su implementación en diversas infraestructuras.

Contabilidad y Control de Sesiones: Permite registrar información detallada sobre las sesiones de los usuarios, lo que puede ser útil tanto para propósitos de seguridad como para la facturación en servicios de internet o redes privadas.

DESVENTAJAS DE RADIUS

Cifrado Limitado: Aunque las contraseñas están cifradas, la mayor parte del paquete RADIUS se transmite en texto claro. Esto puede ser un problema de seguridad si la comunicación no está protegida adecuadamente, por ejemplo, con IPsec o TLS.

Uso de UDP: Al usar UDP como protocolo de transporte, RADIUS es susceptible a la pérdida de paquetes, lo que puede afectar la fiabilidad en redes no confiables o congestionadas. Otros protocolos como TACACS+ usan TCP, que es más confiable.

Escalabilidad Limitada en Redes Masivas: Aunque RADIUS es escalable, en redes extremadamente grandes y complejas (por ejemplo, proveedores globales de internet), puede haber dificultades en la administración eficiente si no se implementa correctamente.

EJEMPLOS DE USO DE RADIUS

Wi-Fi Empresarial: En muchas organizaciones, los puntos de acceso inalámbrico utilizan RADIUS para autenticar a los usuarios antes de permitirles conectarse a la red. El servidor RADIUS consulta una base de datos de usuarios (como Active Directory) para verificar las credenciales.

VPN (Redes Privadas Virtuales): Muchas VPNs utilizan RADIUS para autenticar a los usuarios remotos. Cuando un usuario intenta conectarse a la VPN, el servidor VPN envía una solicitud de autenticación a un servidor RADIUS, que valida las credenciales.

ISP (Proveedores de Servicios de Internet): Los ISP a menudo utilizan RADIUS para autenticar y realizar un seguimiento de los usuarios de sus servicios de acceso a internet. Esto les permite registrar el uso de datos y facturar a los clientes de manera adecuada.

TACACS+ (TERMINAL ACCESS CONTROLLER ACCESS-CONTROL SYSTEM PLUS)

TACACS+ es un protocolo desarrollado por Cisco Systems que proporciona servicios de autenticación, autorización y contabilidad (AAA), principalmente utilizado en entornos de redes para la gestión y control del acceso a dispositivos de red (como routers, switches y firewalls).

TACACS+ es la versión más reciente de los protocolos TACACS y XTACACS, que fueron originalmente diseñados en los años 80 para gestionar el acceso de usuarios a sistemas de red. A diferencia de sus predecesores, TACACS+ ofrece más funcionalidades y una mayor flexibilidad, lo que lo ha hecho ampliamente adoptado, especialmente en dispositivos Cisco.

Aunque TACACS+ y RADIUS proporcionan servicios similares (AAA), existen diferencias importantes que hacen que TACACS+ sea preferido en ciertos entornos, especialmente para la gestión de dispositivos de red:

Protocolo de Transporte: TACACS+ usa TCP (puerto 49), que es un protocolo orientado a la conexión. Esto significa que es más confiable en términos de garantizar la entrega de paquetes, a diferencia de RADIUS, que utiliza UDP (protocolo sin conexión). TCP garantiza que los paquetes se reenvíen si se pierden, lo que lo hace más adecuado en entornos críticos de red.

Separación de Autenticación, Autorización y Contabilidad: TACACS+ separa claramente las funciones de autenticación, autorización y contabilidad, lo que permite un control más granular sobre el acceso de los usuarios y sus acciones. Esto es diferente a RADIUS, donde la autenticación y autorización están combinadas, lo que limita la flexibilidad de los permisos y acciones que se pueden controlar.

Cifrado Completo: TACACS+ cifra todo el contenido del paquete durante la transmisión, mientras que RADIUS solo cifra las contraseñas. Este nivel de cifrado más alto hace que TACACS+ sea más seguro, especialmente en redes sensibles

donde la información que se intercambia entre el cliente y el servidor podría incluir configuraciones y comandos críticos de red.

Aplicación: TACACS+ se utiliza principalmente para la gestión de dispositivos de red como routers, switches y firewalls, lo que lo hace ideal para entornos empresariales o de proveedores de servicios que requieren control granular sobre el acceso de los administradores de red. RADIUS, por otro lado, se utiliza más comúnmente para autenticar a usuarios finales que acceden a la red a través de VPN, Wi-Fi o servicios de acceso remoto.

FUNCIONES DE TACACS+

Autenticación (Authentication): TACACS+ permite que un servidor central verifique las credenciales de los usuarios que intentan acceder a dispositivos de red. A diferencia de RADIUS, que combina autenticación y autorización en una sola etapa, TACACS+ separa estos procesos, permitiendo un control más detallado y flexible. Esta autenticación puede realizarse mediante varios métodos, incluyendo contraseñas simples, tokens, o certificados.

Autorización (Authorization): Después de autenticar a un usuario, TACACS+ determina qué acciones puede realizar ese usuario en el dispositivo de red. Esto significa que, una vez que el usuario ha sido autenticado, TACACS+ decide los privilegios que tendrá, como qué comandos pueden ejecutar o qué configuraciones pueden modificar. Esta capacidad de granularidad es crucial en redes donde diferentes usuarios necesitan diferentes niveles de acceso.

Contabilidad (Accounting): Al igual que en RADIUS, TACACS+ registra las actividades de los usuarios, incluyendo detalles como el tiempo de conexión, los comandos ejecutados, y otras acciones importantes que pueden ser útiles para auditorías, cumplimiento de políticas de seguridad, o resolución de problemas.

PROCESO DE AUTENTICACIÓN TACACS+

Solicitud de Autenticación: Un administrador de red intenta iniciar sesión en un dispositivo de red (por ejemplo, un switch o router). El dispositivo envía una solicitud de autenticación al servidor TACACS+, proporcionando las credenciales del usuario (nombre de usuario y contraseña).

Verificación de Credenciales: El servidor TACACS+ recibe la solicitud, verifica las credenciales del usuario consultando una base de datos (por ejemplo, un servidor LDAP, Active Directory o una base de datos local), y devuelve una respuesta de autenticación exitosa o fallida.

Autorización de Comandos: Si la autenticación es exitosa, TACACS+ verifica qué comandos o acciones está autorizado a realizar el usuario. Esto se basa en las políticas definidas en el servidor TACACS+. Por ejemplo, un administrador de red con permisos de "lectura" podría solo tener acceso a ciertos comandos, mientras que un administrador con permisos de "escritura" podría modificar configuraciones críticas.

Contabilidad de Sesiones: TACACS+ registra la sesión del usuario, incluyendo detalles sobre los comandos ejecutados, el tiempo de conexión, y otros eventos significativos. Esta información puede ser utilizada para auditorías de seguridad o cumplimiento normativo.

VENTAJAS DE TACACS+

Seguridad Mejorada: Dado que TACACS+ cifra todo el contenido del paquete, ofrece una mayor protección contra ataques de interceptación (como el sniffing) en comparación con RADIUS, que solo cifra las contraseñas.

Control Granular: TACACS+ permite un control mucho más detallado sobre las acciones que los usuarios pueden realizar en los dispositivos de red. Los administradores pueden especificar qué comandos están permitidos o prohibidos

para cada usuario, lo que proporciona una flexibilidad que no está disponible en RADIUS.

Fiabilidad con TCP: Al utilizar TCP, TACACS+ es más confiable en entornos donde la entrega de paquetes es crítica. TCP garantiza que los paquetes lleguen de manera segura y en orden, lo que es esencial en la gestión de dispositivos de red.

Modularidad: La separación clara entre autenticación, autorización y contabilidad permite a los administradores gestionar estas funciones de manera independiente, lo que resulta en una gestión más flexible y eficiente del acceso a los dispositivos de red.

DESVENTAJAS DE TACACS+

Propietario de Cisco: Aunque TACACS+ es ampliamente utilizado, es un protocolo propietario de Cisco, lo que significa que está principalmente optimizado para dispositivos Cisco. Esto puede ser una limitación si se utilizan dispositivos de red de múltiples proveedores en una infraestructura. Sin embargo, algunos fabricantes han implementado compatibilidad con TACACS+ en sus dispositivos.

Mayor Complejidad: La flexibilidad y control granular que ofrece TACACS+ también puede introducir más complejidad en la configuración y gestión, en comparación con RADIUS, que es más simple en su implementación.

Mayor Sobrecarga: Debido a que TACACS+ utiliza TCP y cifra todos los paquetes, puede haber una ligera sobrecarga en el rendimiento en comparación con RADIUS, que utiliza UDP y no cifra todo el paquete. Esto puede no ser un problema significativo en redes bien dimensionadas, pero es un factor a considerar en redes muy grandes o congestionadas.

KERBEROS

Es un protocolo de autenticación de red diseñado para proporcionar una forma segura de verificar la identidad de los usuarios en redes distribuidas. Fue desarrollado por el MIT (Massachusetts Institute of Technology) como parte de su Proyecto Athena en los años 80. Kerberos se basa en criptografía de clave simétrica y tiene como objetivo proteger la comunicación entre los clientes y los servidores dentro de una red, asegurando que la identidad de ambas partes esté verificada antes de que se permita el acceso.

Kerberos utiliza un modelo cliente-servidor basado en tickets para autenticar usuarios y servicios. Su diseño sigue una arquitectura de "tercero confiable", lo que significa que un servicio centralizado (denominado Key Distribution Center, o KDC) actúa como intermediario entre el cliente y el servidor, generando y distribuyendo los tickets de autenticación.

COMPONENTES CLAVE:

KDC (Key Distribution Center): El KDC es el componente central del sistema Kerberos, que contiene dos funciones esenciales:

AS (Authentication Server): Realiza la autenticación inicial del usuario y genera un ticket conocido como Ticket Granting Ticket (TGT).

TGS (Ticket Granting Server): Se encarga de generar tickets de servicio específicos basados en el TGT.

Ticket Granting Ticket (TGT): El TGT es un ticket de autenticación temporal que permite al cliente obtener acceso a diferentes servicios dentro de la red sin tener que volver a autenticarse repetidamente. El TGT tiene una duración limitada (normalmente algunas horas), lo que mejora la seguridad.

Tickets de Servicio: Después de obtener un TGT, el cliente puede solicitar al TGS un ticket de servicio específico para acceder a un recurso determinado, como un servidor o una aplicación. El ticket de servicio permite la comunicación segura entre el cliente y el servidor.

PROCESO DE AUTENTICACIÓN DE KERBEROS

Autenticación Inicial (AS Exchange): El usuario inicia sesión y envía una solicitud al Authentication Server (AS) del KDC, proporcionando su nombre de usuario. El AS verifica al usuario y devuelve un TGT cifrado con la clave secreta del usuario, que generalmente está relacionada con la contraseña.

Obtención de un Ticket de Servicio (TGS Exchange): Una vez que el cliente tiene el TGT, puede usarlo para solicitar acceso a otros servicios. El cliente envía el TGT al Ticket Granting Server (TGS) junto con una solicitud para un ticket de servicio. Si la solicitud es válida, el TGS emite un ticket de servicio para el recurso solicitado.

Acceso a Servicios (Client/Server Exchange): El cliente presenta el ticket de servicio al servidor de destino (por ejemplo, una base de datos o un servidor de archivos). El servidor verifica la autenticidad del ticket y, si es válido, permite al cliente acceder al recurso solicitado. A partir de este momento, toda la comunicación entre el cliente y el servidor estará cifrada para proteger la integridad y confidencialidad de los datos.

CARACTERÍSTICAS CLAVE DE KERBEROS

Autenticación Mutua: Kerberos proporciona autenticación mutua, lo que significa que tanto el cliente como el servidor se autentican entre sí. Esto protege contra ataques como la suplantación de servidores, ya que el cliente puede verificar que está interactuando con el servidor correcto.

Uso de Tickets Temporales: Los tickets generados por Kerberos tienen una duración limitada, lo que reduce el riesgo de que un ticket sea reutilizado por un atacante. Una vez que expira el ticket, el usuario deberá volver a autenticarse para obtener uno nuevo.

Cifrado para la Comunicación Segura: Toda la comunicación que involucra el intercambio de tickets y claves se cifra, protegiendo la confidencialidad de las credenciales del usuario y garantizando la integridad de los datos transmitidos.

Delegación: Kerberos admite la delegación, lo que permite a un servidor actuar en nombre de un cliente para acceder a otros servicios dentro de la red. Esto es útil en escenarios donde los servidores necesitan autenticarse en nombre de los usuarios para realizar tareas en otros recursos.

Escalabilidad: Kerberos es altamente escalable, lo que lo hace adecuado para redes grandes y complejas. Su enfoque de autenticación centralizada a través del KDC permite manejar múltiples servicios y usuarios de manera eficiente.

VENTAJAS DE KERBEROS

Seguridad Mejorada: Kerberos utiliza cifrado robusto (normalmente, algoritmos como AES) para proteger las credenciales de los usuarios y las comunicaciones entre clientes y servidores. Además, el uso de tickets temporales limita la posibilidad de reutilización de credenciales robadas.

Autenticación Transparente: Una vez que el usuario ha obtenido un TGT, puede acceder a múltiples servicios dentro de la red sin necesidad de volver a ingresar sus credenciales repetidamente. Esto facilita la autenticación en redes grandes, eliminando la fricción en el proceso de acceso.

Reducción de la Exposición de Contraseñas: En Kerberos, las contraseñas no se transmiten directamente por la red. En su lugar, el KDC utiliza la contraseña del usuario para cifrar el TGT. Esto reduce el riesgo de que un atacante intercepte las credenciales de usuario.

Autenticación Centralizada: Al centralizar la autenticación a través del KDC, Kerberos simplifica la administración de usuarios y servicios. Esto también facilita la implementación de políticas de seguridad y control de acceso en redes empresariales.

Compatibilidad y Adopción Amplia: Kerberos es un estándar abierto que ha sido adoptado por muchas plataformas y sistemas operativos, incluyendo Windows, Linux y macOS. Es compatible con una amplia gama de aplicaciones y servicios de red.

DESVENTAJAS DE KERBEROS:

Punto Único de Fallo (KDC): Dado que Kerberos depende de un KDC centralizado para gestionar la autenticación, este se convierte en un punto único de fallo. Si el KDC deja de funcionar, los usuarios no podrán autenticarse y acceder a los servicios de red.

Complejidad en la Configuración: Implementar y mantener un sistema Kerberos puede ser complejo, especialmente en redes grandes con muchos servicios. Se requiere una configuración adecuada y una infraestructura de red confiable para evitar problemas de rendimiento y seguridad.

Dependencia de la Sincronización de Relojes: Kerberos utiliza marcas de tiempo para evitar ataques de repetición. Por lo tanto, es esencial que todos los dispositivos en la red mantengan relojes sincronizados con precisión. Cualquier desincronización entre el cliente, el servidor y el KDC puede resultar en fallos de autenticación.

Costo Computacional: El cifrado y descifrado de tickets en cada paso del proceso de autenticación implica una sobrecarga computacional, especialmente en redes grandes con muchos usuarios concurrentes.

Compatibilidad Limitada con Proveedores de Servicios No Kerberos: Aunque Kerberos es un estándar ampliamente adoptado, algunos servicios y sistemas antiguos o propietarios pueden no ser compatibles. Esto puede limitar su aplicabilidad en ciertos entornos.

CASOS DE USO DE KERBEROS:

Autenticación en Redes Empresariales: Kerberos se utiliza comúnmente en grandes redes empresariales para proporcionar autenticación centralizada y segura a servicios internos, como bases de datos, servidores de archivos y aplicaciones de misión crítica.

Windows Active Directory: Microsoft adoptó Kerberos como el protocolo de autenticación principal para su sistema Active Directory (AD), utilizado en entornos corporativos para gestionar usuarios y recursos de red.

Autenticación en Aplicaciones Web: Kerberos también puede integrarse con aplicaciones web para autenticar usuarios de forma segura, especialmente en entornos donde la privacidad y la seguridad son prioritarias, como en instituciones financieras y gubernamentales.

Infraestructuras Cloud y Virtualizadas: En entornos de nube y virtualización, Kerberos se utiliza para autenticar el acceso a servicios y recursos distribuidos, permitiendo que los usuarios y sistemas de diferentes ubicaciones geográficas accedan a los recursos de forma segura.

INVESTIGACIÓN DE LOS SERVICIOS DE AUTORIZACIÓN

ACL (ACCESS CONTROL LIST)

Es uno de los métodos más básicos y comunes para gestionar la autorización en sistemas de información. Su objetivo principal es definir qué usuarios o grupos de usuarios pueden acceder a ciertos recursos y qué tipo de acciones pueden realizar sobre esos recursos. Es un enfoque relativamente simple, pero efectivo para sistemas donde el control detallado y directo del acceso es necesario.

Una Access Control List (ACL) es esencialmente una lista que está asociada a un recurso (como un archivo, carpeta, dispositivo de red, etc.), y contiene una serie de entradas que definen los permisos específicos para usuarios o grupos de usuarios. Cada entrada en la lista generalmente especifica:

El identificador del usuario o grupo (por ejemplo, el nombre de usuario o el ID del grupo).

Los permisos asociados con ese usuario o grupo (por ejemplo, permisos de lectura, escritura, ejecución).

Cuando un usuario intenta acceder a un recurso, el sistema consulta la ACL correspondiente para determinar si se le debe permitir realizar la acción solicitada. Si el usuario tiene los permisos adecuados definidos en la ACL, se le otorga acceso. Si no, el acceso es denegado.

TIPOS DE PERMISOS EN UNA ACL

Las ACL pueden especificar varios tipos de permisos, que generalmente se agrupan en tres categorías principales:

1. **Lectura:** Permite que el usuario lea o vea el contenido del recurso.
2. **Escritura:** Permite que el usuario modifique o cambie el contenido del recurso.
3. **Ejecución:** Permite que el usuario ejecute el recurso, que es especialmente relevante en el caso de archivos ejecutables o scripts.
4. En algunos sistemas, puede haber permisos más específicos, como la capacidad de eliminar un recurso o cambiar su propietario.

TIPOS DE ACL

Existen dos tipos principales de ACL que se aplican en diferentes contextos:

ACL Estándar: Define reglas simples basadas solo en la dirección IP o el ID del usuario. Es más común en redes para controlar el tráfico.

ACL Extendida: Proporciona reglas más avanzadas que permiten especificar múltiples condiciones, como la dirección IP de origen y destino, el tipo de tráfico, el puerto, y más. Esto es más útil para un control más preciso en dispositivos de red como routers y firewalls.

En sistemas de archivos, se suelen manejar solo ACL estándar, donde se define el acceso en función del nombre de usuario o grupo.

VENTAJAS DE ACL

Control granular: ACL permite controlar el acceso de manera detallada a nivel de usuario o grupo para cada recurso. Esto permite una personalización precisa del control de acceso.

Simplicidad: El modelo es relativamente fácil de implementar y entender, especialmente en sistemas pequeños donde el número de usuarios y recursos es limitado.

Compatibilidad: Las ACL han sido ampliamente adoptadas y son compatibles con la mayoría de los sistemas operativos y dispositivos de red, lo que las hace versátiles.

DESVENTAJAS DE ACL

Escalabilidad limitada: En entornos grandes con miles de usuarios y recursos, las ACL pueden volverse difíciles de gestionar, ya que cada recurso debe tener su propia lista de control de acceso.

Falta de flexibilidad: Las ACL tienden a ser estáticas y no consideran atributos dinámicos como la ubicación, el tiempo o el dispositivo desde el cual un usuario está accediendo. Esto limita su utilidad en entornos modernos como la nube.

Complejidad en la administración: Si no se administran adecuadamente, las ACL pueden volverse confusas y difíciles de mantener. Es fácil perder el rastro de quién tiene acceso a qué, lo que puede llevar a fallas en la seguridad.

USO DE ACL EN DIFERENTES CONTEXTOS

Sistemas de archivos: En sistemas operativos como Linux y Windows, las ACL se utilizan para controlar el acceso a archivos y carpetas. Los administradores del sistema pueden definir quién puede leer, escribir o ejecutar archivos específicos.

Redes: En el contexto de las redes, las ACL se utilizan en routers y firewalls para controlar el tráfico que pasa a través de una red. Se pueden crear reglas para permitir o bloquear paquetes basados en direcciones IP, puertos, y otros parámetros.

Bases de datos: Las bases de datos también pueden utilizar ACL para controlar qué usuarios pueden realizar consultas, insertar datos o modificar tablas.

EJEMPLO PRÁCTICO DE ACL EN UN SISTEMA DE ARCHIVOS

Imaginemos un sistema de archivos en un servidor que contiene un archivo llamado report.pdf. La ACL para este archivo podría verse así:

- Usuario: Juan – Permisos: Leer, Escribir

- Usuario: María – Permisos: Leer

- Grupo: Administración – Permisos: Leer, Escribir, Ejecutar

En este caso:

Juan puede leer y modificar el archivo.

María solo puede leer el archivo.

Los miembros del grupo Administración pueden leer, modificar y ejecutar el archivo.

ACL EN DISPOSITIVOS DE RED

En routers o firewalls, las ACL son usadas para controlar qué tráfico puede pasar a través del dispositivo. Por ejemplo, una ACL puede permitir todo el tráfico HTTP entrante, pero bloquear el tráfico SSH. Un ejemplo de una regla ACL en un router podría ser:

- Permitir: Tráfico HTTP desde 192.168.1.0/24 a 10.0.0.0/24

- Denegar: Tráfico SSH desde cualquier origen a cualquier destino

Aquí, el tráfico HTTP entre ciertas redes está permitido, mientras que todo el tráfico SSH está bloqueado, independientemente de su origen y destino.

RBAC (ROLE-BASED ACCESS CONTROL)

Es un modelo de autorización ampliamente utilizado en sistemas de gestión de accesos. En RBAC, el acceso a los recursos y la ejecución de acciones están basados en los roles asignados a los usuarios, en lugar de asignar permisos directamente a los usuarios individuales. Este enfoque simplifica la gestión de permisos en organizaciones grandes y complejas al agrupar permisos en roles predefinidos.

En el modelo RBAC, los permisos no se asignan directamente a los usuarios, sino a roles, y luego esos roles se asignan a los usuarios. Los roles representan conjuntos de permisos relacionados con funciones específicas dentro de una organización. Cuando un usuario recibe un rol, hereda todos los permisos asociados con ese rol.

COMPONENTES CLAVE DE RBAC:

Roles: Un rol es un conjunto de permisos que define lo que los usuarios pueden hacer en el sistema. Ejemplos de roles pueden ser "Administrador", "Gerente" o "Empleado". Cada rol tiene permisos específicos, como acceso a ciertas aplicaciones, archivos, o recursos del sistema.

Permisos: Los permisos definen las acciones que pueden ser realizadas sobre los recursos. Por ejemplo, un permiso puede permitir leer un archivo, modificar una base de datos, o ejecutar un programa. Los permisos están agrupados en roles.

Usuarios: Los usuarios son las personas que interactúan con el sistema. Cada usuario puede ser asignado a uno o más roles, y así hereda los permisos asociados con esos roles.

Asignaciones de roles: Los usuarios se asignan a roles específicos, y cada rol tiene un conjunto de permisos predefinidos. La asignación de roles es un proceso clave para gestionar el acceso en RBAC.

Ejemplo:

En una empresa, los roles podrían ser:

- **Administrador:** Tiene permisos completos sobre todos los recursos.
- **Gerente:** Tiene permisos para acceder y modificar informes financieros, pero no puede cambiar configuraciones del sistema.
- **Empleado:** Solo tiene permisos para ver documentos relacionados con su trabajo.

Un usuario llamado Juan puede ser asignado al rol de Gerente, y, por lo tanto, podrá acceder y modificar los informes financieros según los permisos del rol de Gerente. María, que tiene el rol de Empleado, solo podrá ver documentos y no tendrá permisos para modificar o administrar otros recursos.

VENTAJAS DE RBAC

Simplificación de la Gestión de Permisos: Al agrupar permisos en roles, se reduce la complejidad en la gestión de accesos. Los administradores pueden asignar permisos a roles y luego asignar esos roles a usuarios, simplificando la administración.

Escalabilidad: RBAC es escalable en organizaciones grandes porque los permisos se gestionan a nivel de roles, no de usuarios individuales. Esto hace que sea más fácil manejar una gran cantidad de usuarios y permisos.

Facilidad de Auditoría y Cumplimiento: La estructura basada en roles facilita la auditoría y el cumplimiento normativo al permitir una revisión clara de qué permisos están asociados con cada rol y quién tiene asignados esos roles.

Reducción de Errores: Al gestionar permisos a nivel de roles, se minimiza el riesgo de errores que pueden ocurrir al asignar permisos directamente a los usuarios.

DESVENTAJAS DE RBAC

Rigidez: RBAC puede ser rígido y difícil de ajustar a cambios rápidos en el entorno de trabajo. Los roles pueden no adaptarse bien a las necesidades cambiantes de los usuarios o de la organización.

Sobrecarga de Roles: En entornos complejos, puede haber una proliferación de roles para cubrir todas las combinaciones de permisos necesarios, lo que puede llevar a una administración y mantenimiento complicados.

No considera atributos dinámicos: RBAC no toma en cuenta factores contextuales o dinámicos (como la ubicación o el momento del acceso), lo que puede limitar su capacidad para manejar escenarios complejos que requieren decisiones de acceso basadas en múltiples factores.

IMPLEMENTACIÓN DE RBAC

Para implementar RBAC en un sistema, se siguen generalmente los siguientes pasos:

Definir Roles: Identificar y definir roles basados en las funciones y responsabilidades dentro de la organización.

Asignar Permisos a Roles: Determinar qué permisos son necesarios para cada rol y asignarlos en consecuencia.

Asignar Roles a Usuarios: Asignar a los usuarios los roles apropiados basados en su función en la organización.

Gestionar Cambios en Roles y Permisos: Actualizar los roles y permisos según sea necesario para reflejar cambios en la organización o en las políticas de acceso.

COMPARACIÓN CON OTROS MODELOS DE CONTROL DE ACCESO

RBAC frente a ACL:

- RBAC se basa en roles y es más eficiente en grandes organizaciones con muchos usuarios y recursos.
- ACL se basa en permisos específicos para cada usuario o grupo, lo que puede ser más detallado pero menos escalable.

RBAC frente a ABAC:

- RBAC es más sencillo y se enfoca en roles estáticos.
- ABAC ofrece mayor flexibilidad al basarse en atributos dinámicos y contextuales, lo que puede adaptarse mejor a entornos cambiantes.

RBAC frente a PBAC:

- RBAC usa roles predefinidos para asignar permisos.
- PBAC utiliza políticas que pueden incorporar roles, atributos, y otros factores, ofreciendo una solución más flexible y adaptable.

BIBLIOGRAFIA

Apache Software Foundation. (n.d.). Apache Directory - LDAP documentation.

Wikipedia. (n.d.). Lightweight Directory Access Protocol.

Microsoft. (n.d.). What is LDAP?

FreeRADIUS. (n.d.). FreeRADIUS documentation.

Wikipedia. (n.d.). RADIUS.

Cisco. (n.d.). What is RADIUS?

Cisco. (n.d.). TACACS+ and RADIUS comparison.

Wikipedia. (n.d.). TACACS.

MIT. (n.d.). MIT Kerberos.

Wikipedia. (n.d.). Kerberos (protocol).

Microsoft. (n.d.). Kerberos authentication overview.

Cisco Systems. (n.d.). Access Control Lists (ACLs).

TechTarget. (n.d.). Access Control List (ACL).

National Institute of Standards and Technology. (n.d.). Role-Based Access Control.

Microsoft. (n.d.). Role-based access control (RBAC) for Azure resources.

NIST. (n.d.). Guide to Attribute-Based Access Control (ABAC) Definition and Considerations.

Oracle. (n.d.). Attribute-Based Access Control (ABAC).

IBM. (n.d.). Policy-Based Access Control.

Red Hat. (n.d.). Policy-Based Access Control (PBAC).