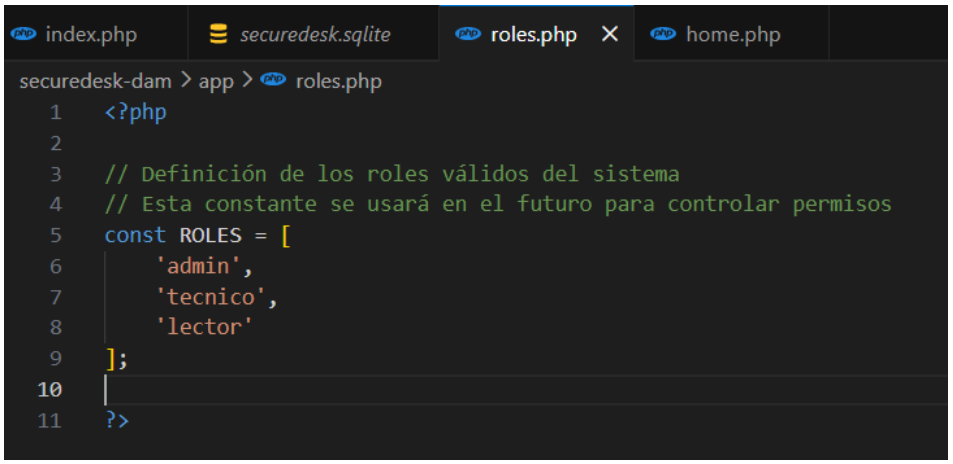
SEMANA 2 — LOGIN + ROLES + SESIONES

ÍNDICE

Tarea 1 — Completar la tabla users y crear usuarios iniciales

Lo primero que se pide en la tarea es asegurarnos que los campos de la tabla users “id, username, password_hash, role, created_at” han sido creados, efectivamente en la anterior semana fueron creados y es tan fácil de comprobar como ver que la tabla y ver que los campos han sido creados.

Lo siguiente que se nos pide es definir los roles de técnico, admin y lector, para ello yo he creado un nuevo archivo llamado **roles.php** en la carpeta **app/**, de esta forma dejará claro cuales son los roles predeterminados para que a la hora de escribir los roles las futuras veces no hayan errores. Estos roles se utilizan para controlar el acceso a las distintas funcionalidades de la aplicación y se almacenan en la base de datos como texto.



Tras definir los roles que usaremos en un archivo, debemos de crear un usuario para cada rol, para ello crearemos un archivo aparte con nombre **db_seed_users.php**, para añadir los usuarios de prueba, no se debe de mezclar en el init.php, por que de esta manera podemos separar responsabilidades además de que si lo juntamos todo, cada vez que ejecutemos el código puede darse el caso de que se dupliquen los usuarios, cosa que no queremos que pase.

A continuación se encuentra el código donde se crean dichos usuarios

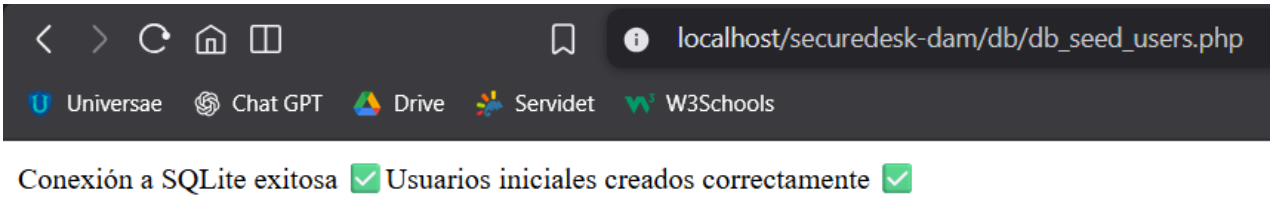
db_seed_users.php

securedesk-dam > db > db_seed_users.php

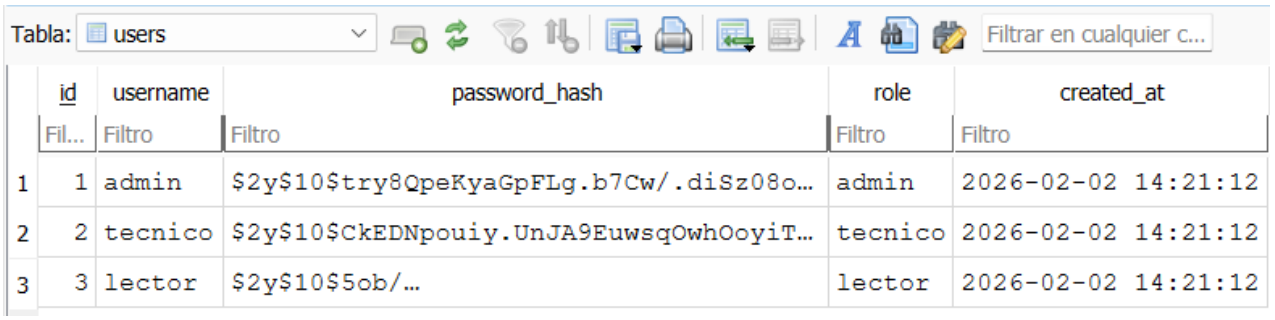
```php
<?php

// Cargamos la configuración de la base de datos
require_once __DIR__ . '/../app/config.php';

// (Opcional pero recomendable) Cargamos los roles definidos
require_once __DIR__ . '/../app/roles.php';

try {
    // Definimos los usuarios iniciales del sistema
    $users = [
        [
            'username' => 'admin',
            'password' => 'admin123',
            'role' => 'admin'
        ],
        [
            'username' => 'tecnico',
            'password' => 'tecnico123',
            'role' => 'tecnico'
        ],
        [
            'username' => 'lector',
            'password' => 'lector123',
            'role' => 'lector'
        ]
    ];
    // Preparamos la consulta SQL para insertar usuarios
    $stmt = $db->prepare("
        INSERT INTO users (username, password_hash, role, created_at)
        VALUES (:username, :password_hash, :role, :created_at)
    ");

    // Insertamos cada usuario en la base de datos
    foreach ($users as $user) {
        $stmt->execute([
            ':username' => $user['username'],
            ':password_hash' => password_hash($user['password'], PASSWORD_DEFAULT),
            ':role' => $user['role'],
            ':created_at' => date('Y-m-d H:i:s')
        ]);
    }

    echo "Usuarios iniciales creados correctamente ✅";

} catch (PDOException $e) {
    echo "Error al crear usuarios: " . $e->getMessage();
}

```

Para ejecutar el código, encenderemos el servidor con XAMPP, y entramos en la ruta: **http://localhost/securedesk-dam/db/db_seed_users.php**

Si todo ha salido bien por pantalla saldrá que han sido creados los usuarios correctamente.



Para comprobar que efectivamente se ha creado como dice, abriremos un visualizador para tablas sqlite, la otra vez use una extensión, pero en este caso he descargado "**DB Browser for SQLite**", en el cual abriremos nuestro archivo sqlite donde se encuentran las tablas, que tiene el nombre de "**securedesk.sqlite**" y nos metemos en la tabla users, click derecho y mostrar datos. A continuación se puede ver que se ha creado uno con cada rol.



	id	username	password_hash	role	created_at
	Fil...	Filtro	Filtro	Filtro	Filtro
1	1	admin	$2y$10$try8QpeKyaGpFLg.b7Cw/.diSz08o…	admin	2026-02-02 14:21:12
2	2	tecnico	$2y$10$CkEDNpouiy.UnJA9EuwsqOwhOoyiT…	tecnico	2026-02-02 14:21:12
3	3	lector	$2y$10$5ob/…	lector	2026-02-02 14:21:12

Con esto concluye la primera tarea de esta semana.

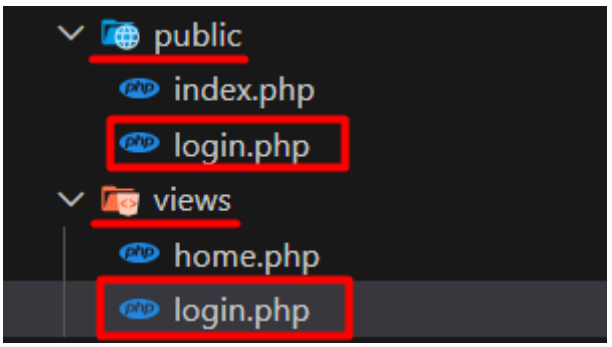
Tarea 2 — Implementar login (validación + hash de contraseña)

En esta tarea se ha implementado el sistema de acceso al SecureDesk mediante usuario y contraseña. El objetivo principal es que el acceso a la aplicación deje de ser público y solo puedan entrar usuarios registrados en la base de datos.

Para ello, se ha creado un formulario de login y la lógica necesaria para validar las credenciales introducidas por el usuario contra la base de datos SQLite, utilizando contraseñas cifradas para garantizar la seguridad.

Para implementar el login se han utilizado dos archivos diferentes, siguiendo la estructura del proyecto y separando la lógica de la vista.

Por un lado, se ha creado el archivo **login.php** dentro de la carpeta **public/**, que se encarga de gestionar la lógica del login. Por otro lado, se ha creado una vista dentro de la carpeta **views/**, que contiene únicamente el formulario de acceso.



Esta separación permite que el código sea más claro, organizado y fácil de mantener.

Comenzando con el **views/login.php** es decir, la pantalla que se mostrará, este tiene el código HTML encargado de mostrar el formulario de acceso al usuario. En esta vista se muestra un título identificando la pantalla de login y un formulario con dos campos obligatorios: nombre de usuario y contraseña.

El formulario utiliza el método **POST**, ya que es el método adecuado para enviar datos sensibles como contraseñas sin que aparezcan en la URL.

Además, esta vista está preparada para mostrar un mensaje de error en caso de que las credenciales introducidas no sean correctas. Dicho mensaje se muestra

únicamente cuando existe una variable llamada **$error**, la cual es definida desde el archivo de lógica.
La vista no realiza ninguna validación ni conexión con la base de datos, ya que su única función es mostrar la interfaz al usuario. A continuación el código de la vista.

login.php ...\views × login.php ...\public

securedesk-dam > views > login.php

```php
<!DOCTYPE html>
<html lang="es">
<head>
    <!-- Definimos la codificación de caracteres -->
    <meta charset="UTF-8">

    <!-- Título que se mostrará en la pestaña del navegador -->
    <title>Login | SecureDesk</title>
</head>
<body>

    <!-- Título principal de la página -->
    <h1>Acceso a SecureDesk</h1>

    <?php
    // Comprobamos si existe la variable $error
    // Esta variable se define en el archivo de lógica cuando las credenciales son incorrectas
    if (isset($error)):
    ?>
        <!-- Mostramos el mensaje de error en color rojo -->
        <p style="color:red;">
            <?= $error ?>
        </p>
    <?php endif; ?>
```

```php
    <!--
        Formulario de login
        Usamos el método POST para no enviar datos sensibles por la URL
    -->
    <form method="POST">

        <!-- Campo para introducir el nombre de usuario -->
        <label>
            Usuario:
            <input type="text" name="username" required>
        </label>

        <br><br>

        <!-- Campo para introducir la contraseña -->
        <label>
            Contraseña:
            <input type="password" name="password" required>
        </label>

        <br><br>

        <!-- Botón para enviar el formulario -->
        <button type="submit">Entrar</button>
    </form>

</body>
</html>
```

El archivo **public/login.php** es el encargado de procesar el formulario de login y validar las credenciales introducidas. En primer lugar, se carga el archivo **config.php**, el cual establece la conexión con la base de datos SQLite. Esta conexión es necesaria para poder consultar la tabla de usuarios.

A continuación, se inicializa una variable llamada **$error**, que se utilizará para almacenar un mensaje de error en caso de que el usuario introduzca credenciales incorrectas.

El archivo comprueba si el formulario ha sido enviado mediante el método **POST**. Si es así, se recogen los datos introducidos por el usuario (nombre de usuario y contraseña).

Posteriormente, se realiza una consulta a la base de datos para buscar un usuario cuyo nombre coincida con el introducido en el formulario. Para esta consulta se utilizan sentencias preparadas, lo que evita posibles ataques de inyección SQL.

Si el usuario existe, se compara la contraseña introducida con el hash almacenado en la base de datos utilizando la función **password_verify**. Esta función permite comprobar de forma segura si la contraseña es correcta. Si las credenciales son válidas, el sistema considera el login como correcto y muestra un mensaje de confirmación. En esta fase del proyecto aún no se gestionan sesiones, por lo que el acceso no se mantiene entre páginas.

En caso de que el usuario no exista o la contraseña no sea correcta, se asigna un mensaje de error a la variable **$error**, que posteriormente será mostrado en la vista del formulario.

Finalmente, se carga la vista del login, que se encargará de mostrar el formulario y, en caso necesario, el mensaje de error correspondiente.

in.php ×

edesk-dam > public > login.php

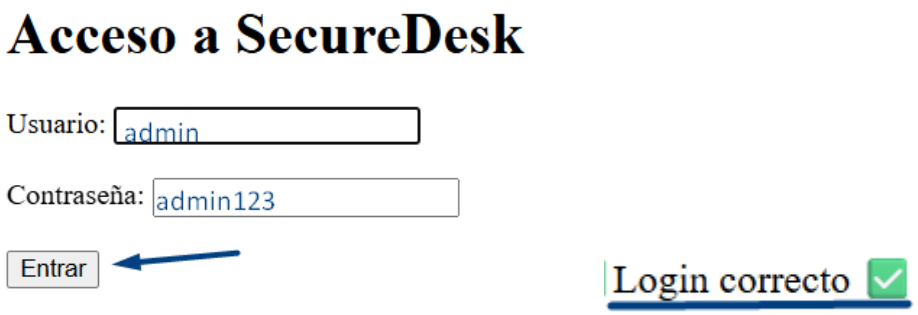
```php
<?php

// Cargamos el archivo de configuración
// Este archivo contiene la conexión a la base de datos SQLite
require_once __DIR__ . '/../app/config.php';

// Inicializamos la variable de error como null
// Se usará para mostrar mensajes en la vista si algo falla
$error = null;

// Comprobamos si el formulario se ha enviado usando el método POST
if ($_SERVER['REQUEST_METHOD'] === 'POST') {

    // Recogemos el nombre de usuario enviado por el formulario
    // Usamos ?? para evitar errores si el campo no existe
    $username = $_POST['username'] ?? '';

    // Recogemos la contraseña enviada por el formulario
    $password = $_POST['password'] ?? '';

    // Preparamos una consulta SQL para buscar al usuario en la base de datos
    // Usamos una consulta preparada para evitar inyecciones SQL
    $stmt = $db->prepare("
        SELECT * FROM users
        WHERE username = :username
        LIMIT 1
    ");

    // Ejecutamos la consulta pasando el nombre de usuario como parámetro
    $stmt->execute([
        ':username' => $username
    ]);
```

```php
    // Obtenemos el resultado de la consulta como un array asociativo
    $user = $stmt->fetch(PDO::FETCH_ASSOC);

    // Comprobamos:
    // 1. Que el usuario exista
    // 2. Que la contraseña introducida coincida con el hash almacenado
    if ($user && password_verify($password, $user['password_hash'])) {

        // Si las credenciales son correctas, el login es válido
        // De momento solo mostramos un mensaje de éxito
        echo "Login correcto ✅";

        // Detenemos la ejecución del script
        exit;

    } else {
        // Si el usuario no existe o la contraseña no es correcta
        // Definimos un mensaje de error
        $error = 'Usuario o contraseña incorrectos';
    }
}

// Cargamos la vista del formulario de login
// Si existe un error, la vista lo mostrará automáticamente
require_once __DIR__ . '/../views/login.php';
?>
```

Las contraseñas de los usuarios no se almacenan en texto plano en la base de datos. Durante la creación de los usuarios se utilizan contraseñas cifradas mediante la función **password_hash**, y durante el login se validan utilizando **password_verify**.

Este método garantiza un nivel de seguridad adecuado y sigue las buenas prácticas recomendadas en el desarrollo de aplicaciones web con PHP.

A continuación, haremos un intento para ver si efectivamente todo lo creado funciona, como siempre, ejecutamos **XAMPP** y para ejecutar el login debemos entrar en http://localhost/securedesk-dam/public/login.php se hace desde la carpeta public ya que es donde se encuentra la lógica del login y nos visualiza el código como hemos explicado con anterioridad.



Meteremos como en la imagen anterior se ve, meteremos estos datos, le damos a entrar y nos saltará un pequeño mensaje de **login correcto**. Para comprobar que no ha funcionado el login, metemos unos datos que sean erróneos y nos debe de saltar este mensaje.

Con esto concluye la segunda tarea de la segunda semana.

Tarea 3 — Implementar sesión y logout

En esta tarea se ha implementado el sistema de sesiones de usuario, con el objetivo de mantener al usuario autenticado mientras navega por la aplicación SecureDesk y permitir cerrar sesión de forma segura.
Gracias a este sistema, la aplicación deja de ser pública y sólo permite el acceso a usuarios que hayan iniciado sesión correctamente.

Para poder mantener la información del usuario entre distintas páginas, se utiliza el sistema de sesiones que ofrece PHP mediante la variable global **$_SESSION**.

Para ello, es obligatorio llamar a la función **session_start()** al inicio de cualquier archivo que vaya a utilizar o comprobar datos de sesión. Esta llamada permite iniciar o recuperar una sesión existente asociada al usuario.

El archivo login.php ha sido modificado para que, además de validar las credenciales del usuario, guarde sus datos en la sesión cuando el login es correcto.

Cuando el usuario introduce correctamente su nombre de usuario y contraseña:

- Se comprueba la contraseña mediante **password_verify**
- Se almacenan en la sesión datos como el identificador del usuario, su nombre y su rol
- Se redirige al usuario a la página principal de la aplicación

De esta forma, el sistema recuerda qué usuario ha iniciado sesión. A continuación, en código modificado:

```php
<?php

// Iniciamos la sesión para poder guardar datos del usuario
// Debe ir siempre al principio del archivo
session_start();

// Cargamos la configuración y la conexión a la base de datos
require_once __DIR__ . '/../app/config.php';

// Inicializamos la variable de error
// Se utilizará para mostrar mensajes en la vista si el login falla
$error = null;

// Comprobamos si el formulario se ha enviado mediante POST
if ($_SERVER['REQUEST_METHOD'] === 'POST') {

    // Recogemos el nombre de usuario del formulario
    $username = $_POST['username'] ?? '';

    // Recogemos la contraseña del formulario
    $password = $_POST['password'] ?? '';

    // Preparamos la consulta SQL para buscar al usuario por su username
    // Se utiliza una consulta preparada para evitar inyecciones SQL
    $stmt = $db->prepare("
        SELECT * FROM users
        WHERE username = :username
        LIMIT 1
    ");

    // Ejecutamos la consulta pasando el nombre de usuario como parámetro
    $stmt->execute([
        ':username' => $username
    ]);

    // Obtenemos el usuario encontrado (si existe) como array asociativo
    $user = $stmt->fetch(PDO::FETCH_ASSOC);

    // Comprobamos que el usuario exista y que la contraseña sea correcta
    // password_verify compara la contraseña introducida con el hash guardado
    if ($user && password_verify($password, $user['password_hash'])) {

        // Guardamos los datos del usuario en la sesión
        // Esto permite mantener al usuario autenticado
        $_SESSION['user_id'] = $user['id'];
        $_SESSION['username'] = $user['username'];
        $_SESSION['role'] = $user['role'];

        // Redirigimos al usuario a la página principal
        header('Location: index.php');
        exit;

    } else {
        // Si las credenciales no son correctas, definimos el mensaje de error
        $error = 'Usuario o contraseña incorrectos';
    }
}

// Cargamos la vista del formulario de login
// La vista mostrará el formulario y el error si existe
require_once __DIR__ . '/../views/login.php';
?>
```

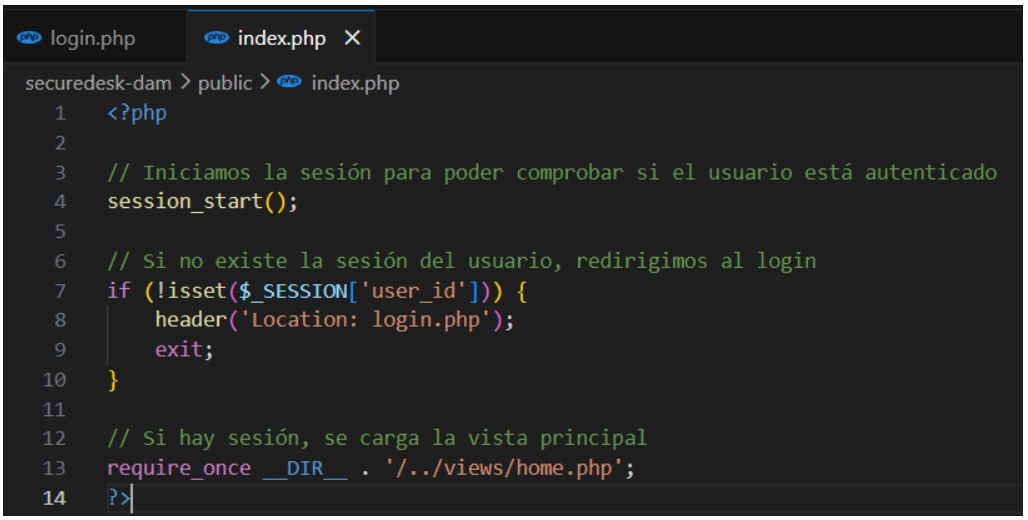
En caso de que las credenciales introducidas no sean correctas, el sistema define un mensaje de error que se muestra en la vista del formulario de login.
Esto permite informar al usuario sin comprometer la seguridad del sistema, ya que no se especifica si el error es el usuario o la contraseña, el mismo error que salía anteriormente.

Una vez implementada la sesión, es necesario proteger las páginas privadas de la aplicación para evitar accesos no autorizados.

Para ello, en archivos como **index.php**, se comprueba si existe una sesión activa del usuario. Si no existe, el sistema redirige automáticamente al usuario a la pantalla de login.

Esto asegura que:

- Un usuario no autenticado no puede acceder directamente mediante la URL.
- El acceso a la aplicación está completamente controlado.

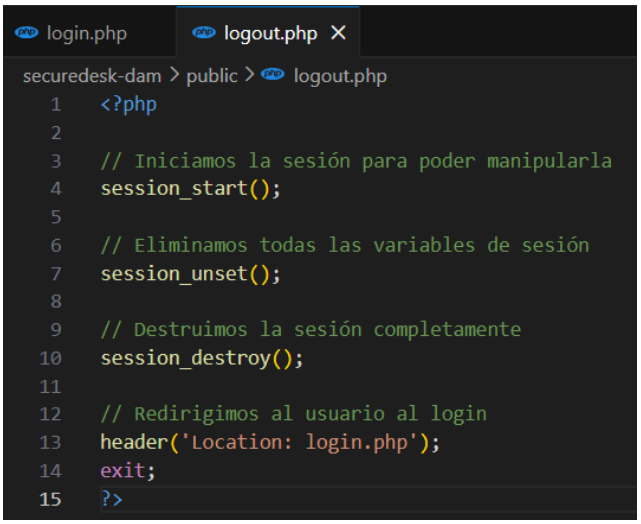


Para permitir que el usuario cierre sesión de forma segura, se ha creado un archivo específico llamado **logout.php**.

Este archivo se encarga de:

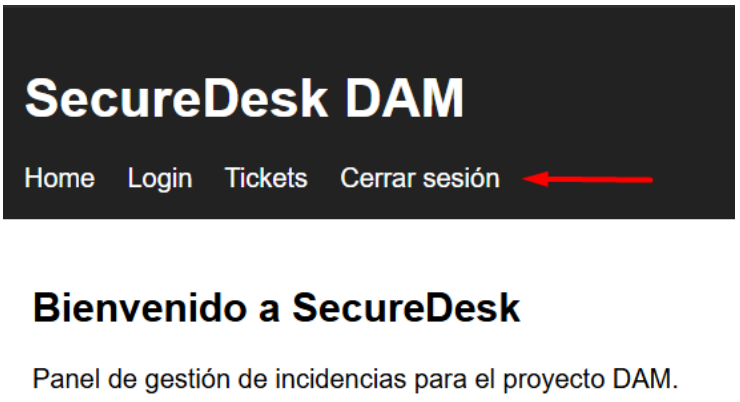
- Vaciar las variables de sesión
- Destruir completamente la sesión
- Redirigir al usuario de nuevo a la pantalla de login

De esta forma, el usuario deja de estar autenticado y no puede acceder a páginas privadas sin volver a iniciar sesión.



Tras eso desde **views/home.php**, crearemos un enlace que haga un llamamiento a **logout.php** a continuación el código.

Tras entrar con un usuario como hemos hecho otras veces, nos redirige a la página principal, en la cual ahora en la barra de menú existirá un nuevo botón para cerrar sesión, a continuación la muestra.



Tras tocar el botón nos redirige a la página de inicio se sesión, también se ha realizado la prueba de recargar la página en el menú principal y efectivamente mantiene la sesión iniciada sin problema.

Tarea 4 — Control de acceso por rol (middleware/guard simple)

En esta tarea se ha implementado un sistema de control de acceso por roles, con el objetivo de que cada usuario solo pueda acceder a las funcionalidades que le corresponden dentro de la aplicación SecureDesk.

De esta forma, se mejora la seguridad y se evita que usuarios sin permiso accedan a zonas restringidas de la aplicación.

La aplicación SecureDesk cuenta con tres tipos de usuario, cada uno con permisos distintos:

- Admin: Tiene acceso total a la aplicación, incluyendo la gestión de usuarios y tickets.
- Técnico: Puede visualizar, crear y editar tickets, pero no tiene acceso a la gestión de usuarios.
- Lector: Solo puede visualizar los tickets, sin posibilidad de crear o editar.

El rol de cada usuario se guarda en la base de datos y se carga en la sesión al iniciar sesión correctamente.

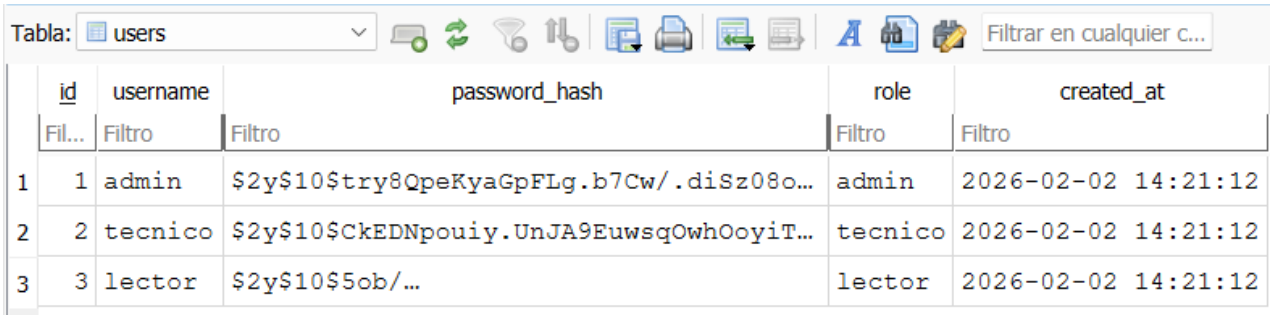


Tabla: users

Filtrar en cualquier c...

	id	username	password_hash	role	created_at
	Fil...	Filtro	Filtro	Filtro	Filtro
1	1	admin	$2y$10$try8QpeKyaGpFLg.b7Cw/.diSz08o…	admin	2026-02-02 14:21:12
2	2	tecnico	$2y$10$CkEDNpouiy.UnJA9EuwsqOwhOoyiT…	tecnico	2026-02-02 14:21:12
3	3	lector	$2y$10$5ob/…	lector	2026-02-02 14:21:12

Para centralizar la lógica de seguridad y evitar repetir código en todas las páginas, se ha creado un archivo encargado de comprobar si el usuario está autenticado y si su rol tiene permiso para acceder a una página concreta.

Este archivo actúa como un **guard o middleware simple**, ejecutándose antes de mostrar el contenido de las páginas protegidas.
Archivo: **app/auth.php**

Este archivo contiene funciones que comprueban si el usuario ha iniciado sesión y verifican si el rol del usuario está autorizado.

auth.php

securedesk-dam > app > auth.php

```php
<?php

// Este archivo contiene funciones de seguridad relacionadas con autenticación y roles

// Comprueba si el usuario ha iniciado sesión
function requireLogin()
{
    if (!isset($_SESSION['user_id'])) {
        header('Location: login.php');
        exit;
    }
}

// Comprueba si el usuario tiene uno de los roles permitidos
function requireRole(array $roles)
{
    if (!isset($_SESSION['role']) || !in_array($_SESSION['role'], $roles)) {
        header('Location: index.php');
        exit;
    }
}
?>
```

Para aplicar el control de acceso, se han creado páginas de prueba que representan las distintas zonas de la aplicación.
Estas páginas no contienen todavía la funcionalidad completa, sino que sirven para comprobar que los roles funcionan correctamente.

Antes de mostrar el contenido de cada página, se comprueba:

- Que el usuario haya iniciado sesión
- Que su rol tenga permiso para acceder

La **página de gestión de tickets** sólo es accesible para usuarios con rol **admin o técnico**. Este archivo ha sido llamado **public/tickets.php**

securedesk-dam > public > tickets.php

```php
<?php

session_start();
require_once __DIR__ . '/../app/auth.php';

// Solo usuarios logueados
requireLogin();

// Solo admin y técnico
requireRole(['admin', 'tecnico']);

echo "<h1>Zona de Tickets</h1>";
echo "<p>Solo admin y técnico pueden acceder.</p>";
?>
```

He entrado como admin en gestión de usuarios y de tickets y funcionan sin ningún tipo de problema.

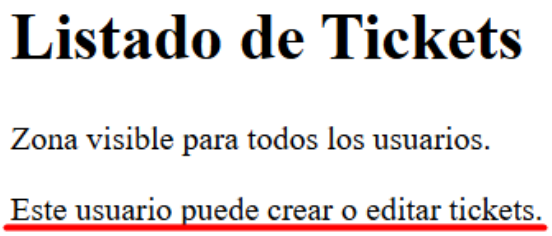
En cambio, si entro con el usuario lector, simplemente no me deja acceder a ninguna de estas dos.

La **página de visualización de tickets** puede ser accedida por todos los roles, aunque las acciones disponibles dependen del tipo de usuario. Este archivo ha sido llamado **public/tickets_view.php**

En esta página:

- Todos los usuarios pueden ver los tickets
- Solo admin y técnico pueden ver opciones de creación o edición

```
securedesk-dam > public > tickets_view.php
1   <?php
2
3   session_start();
4   require_once __DIR__ . '/../app/auth.php';
5
6   requireLogin();
7
8   // Todos los roles
9   requireRole(['admin', 'tecnico', 'lector']);
10
11  echo "<h1>Listado de Tickets</h1>";
12  echo "<p>Zona visible para todos los usuarios.</p>";
13
14  // Control visual de acciones
15  if ($_SESSION['role'] !== 'lector') {
16      echo "<p>Este usuario puede crear o editar tickets.</p>";
17  }
```



La frase subrayada de rojo, en caso de entrar como usuario lector, no aparece.

La página de gestión de usuarios es exclusiva para el **rol admin**.
Cualquier otro rol que intente acceder será bloqueado automáticamente. Este archivo ha sido llamado **public/users.php.**

securedesk-dam > public > users.php

```php
<?php
session_start();
require_once __DIR__ . '/../app/auth.php';

// Obligamos a que el usuario esté logueado
requireLogin();

// Solo el rol admin puede acceder
requireRole(['admin']);

echo "<h1>Zona de gestión de usuarios</h1>";
echo "<p>Zona exclusiva para administradores.</p>";
?>
```

A esta pestaña como se ha explicado antes, solo se puede entrar desde admin, no se puede entrar con lector o técnico, al presionar sobre la pestaña de usuarios te redirige a Home.

Tarea 5 — Menú dinámico según rol + página “Mi cuenta”

En esta tarea se ha mejorado la experiencia de usuario de la aplicación SecureDesk mediante la implementación de un **menú dinámico adaptado al rol del usuario** y la creación de una página personal denominada “**Mi cuenta**”.
Además, se ha añadido una redirección automática tras el login en función del rol, haciendo la navegación más intuitiva.

Hasta este punto, todos los usuarios visualizaban las mismas opciones en el menú, independientemente de su rol.
Para mejorar la usabilidad y evitar accesos innecesarios, se ha implementado un menú que cambia dinámicamente según el rol del usuario autenticado.

De esta forma:

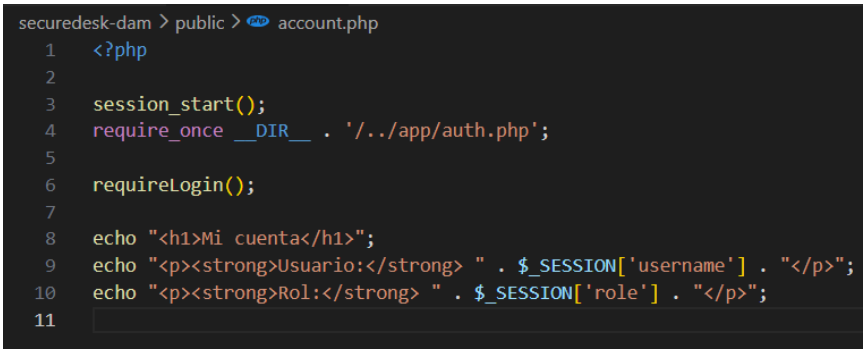
- El **administrador** ve todas las opciones disponibles
- El **técnico** solo ve las opciones relacionadas con los tickets
- El **lector** únicamente ve las opciones de visualización

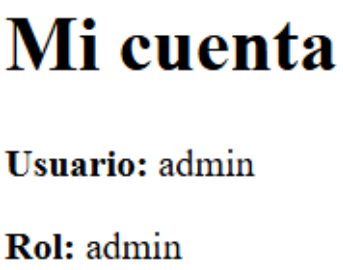
Este control se realiza utilizando la información del rol almacenada en la sesión del usuario.

```
<header>
    <h1>SecureDesk DAM</h1>
    <nav>
        <ul>
            <li><a href="index.php">Home</a></li>
            <li><a href="tickets_view.php">Ver tickets</a></li>

            <?php if ($_SESSION['role'] !== 'lector'): ?>
                <li><a href="tickets.php">Gestionar tickets</a></li>
            <?php endif; ?>

            <?php if ($_SESSION['role'] === 'admin'): ?>
                <li><a href="users.php">Usuarios</a></li>
            <?php endif; ?>

            <li><a href="account.php">Mi cuenta</a></li>
            <li><a href="logout.php">Cerrar sesión</a></li>
        </ul>
    </nav>
</header>
```

Se ha creado una página denominada “**Mi cuenta**”, accesible para todos los usuarios autenticados. Esta página muestra información básica del usuario que ha iniciado sesión, como su nombre de usuario y su rol dentro de la aplicación.

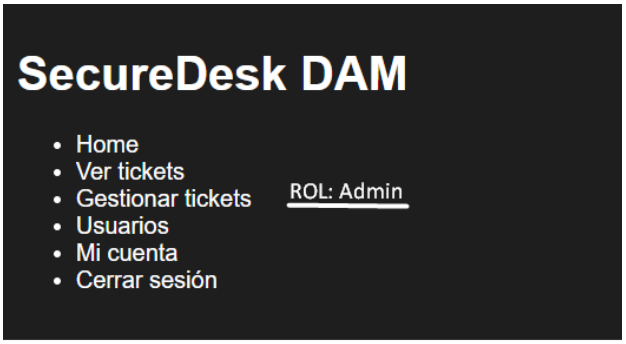
Los datos mostrados se obtienen directamente de la sesión, sin necesidad de realizar consultas adicionales a la base de datos.

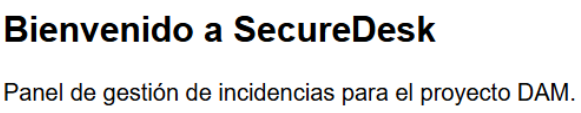




Para mejorar la navegación, se ha modificado el proceso de login para que, una vez autenticado el usuario, sea redirigido automáticamente a una página acorde a su rol.
La redirección se realiza de la siguiente manera:
Admin → Página principal (Home)
Técnico → Gestión de tickets
Lector → Visualización de tickets

A continuación los diferentes pantallas al entrar dependiendo del rol





Esta funcionalidad evita que todos los usuarios comiencen en la misma página y mejora la experiencia de uso.

Por último he añadido una barra de navegación en las pestañas de gestión de usuarios, tickets... Para mejorar la navegación de la web y que puedan volver a el home sin problema y también haré cambios para mejorar la parte visual de la web un poco, por el momento la barra ha quedado así:

Conclusión

Durante esta semana se ha conseguido transformar SecureDesk en una aplicación web funcional y segura, implementando un sistema completo de autenticación, gestión de sesiones y control de acceso por roles.

Este trabajo ha permitido reforzar conceptos clave de desarrollo web con PHP, como el uso de sesiones, la seguridad en el manejo de contraseñas y la separación de responsabilidades dentro de la aplicación.
Además, la implementación de un menú dinámico y una página de usuario mejora notablemente la experiencia de uso y acerca el proyecto a un entorno real de trabajo.

En conjunto, esta semana ha sido fundamental para sentar las bases de seguridad y navegación del proyecto, dejando la aplicación preparada para ampliar funcionalidades en las siguientes semanas.

El repositorio de Github con todos los códigos a continuación:

https://github.com/elgancho06/Practicas-DAM.git