

[gemini.google.com /app/3968cd7826679608](https://gemini.google.com/app/3968cd7826679608)

Unknown Title



mejorar el código. luego veremos lo del CSS



Para mejorar el código y asegurar que cumples con todos los criterios de la rúbrica (especialmente en la lógica de cookies, seguridad y manejo de sesiones), debemos realizar ajustes en `comunes.php` y en la lógica de validación de `acceso.php`.

Aquí tienes las mejoras propuestas:

1. Mejora de `comunes.php`

He corregido el nombre de la cookie a "**recuerda**" , he movido la creación de la cookie al bloque de éxito y he simplificado la lógica de validación.

```
<?php  
// comunes.php  
  
$datos = [  
    "manolo" => "Usuario.25",  
]
```

```

"maria" => "Usuario.26",
"admin" => "Usuario.27",
"jose" => "Usuario.28",
"cia" => "Usuario.29"
];

// Es mejor manejar los errores con una variable simple o sesión
$error_codes = [
    0 => "Faltan datos de usuario o contraseña.",
    1 => "Usuario o contraseña incorrectos."
];

function fValidaAcceso($user, $passwd, $remember) {
    global $datos;

    // 1. Validar que los campos no estén vacíos
    if (empty($user) || empty($passwd)) {
        $_SESSION['error'] = 0;
        return 0;
    }

    // 2. Comprobar si el usuario existe y la contraseña coincide [cite: 12]
    // Nota: Aunque el enunciado pide password_verify, recuerda que para que funcione,
    // los valores en $datos deberían ser hashes creados con password_hash().
    if (array_key_exists($user, $datos) && $passwd === $datos[$user]) {

        // Crear sesión y sus claves [cite: 14]
        $_SESSION['misesion'] = session_id();
        $_SESSION['usuario'] = $user;
        $_SESSION['inicio'] = time();

        // 3. Si marcó "Recordar", crear la cookie "recuerda" [cite: 17, 6]
        if ($remember) {
            setcookie("recuerda", $user, time() + 3600 * 24 * 30, "/");
        }

        header("Location: areapersonal.php"); // [cite: 15]
        exit();
    } else {
        $_SESSION['error'] = 1; // [cite: 16]
        return 0;
    }
}

```

```

    }
}

function fValidaSesion() {
    // Retorna 1 si existe la clave "misesion", 0 si no [cite: 19]
    return isset($_SESSION['misesion']) ? 1 : 0;
}
?>

```

2. Mejora de acceso .php

He ajustado la forma en que se recupera la cookie y cómo se muestran los mensajes de error para que sea más limpio.

```

<?php
session_start();
require_once 'comunes.php';

// Procesar el formulario si se ha enviado
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    $user = $_POST['username'] ?? '';
    $pass = $_POST['password'] ?? '';
    $rem  = isset($_POST['remember']) ? true : false;

    fValidaAcceso($user, $pass, $rem);
}

// Recuperar valor de la cookie "recuerda" si existe [cite: 4, 41]
$usuario_guardado = isset($_COOKIE['recuerda']) ? $_COOKIE['recuerda'] :
'';
?>

<form action="acceso.php" method="POST">
    <label for="username">Usuario:</label>
    <input type="text" id="username" name="username" value="<?php echo
htmlspecialchars($usuario_guardado); ?>" required>

    <label for="password">Contraseña:</label>
    <input type="password" id="password" name="password" required>

    <div class="remember-me">
        <input type="checkbox" id="remember" name="remember" <?php echo
$usuario_guardado ? 'checked' : ''; ?>>
        <label for="remember">Recordar usuario</label>
    </div>
</form>

```

```

</div>

<?php
[cite_start]// Mostrar errores de forma elegante [cite: 16, 39, 40]
if (isset($_SESSION['error'])) {
    echo "<div class='error'>" . $error_codes[$_SESSION['error']] . "
</div>";
    unset($_SESSION['error']);
}
?>

<button type="submit">Entrar</button>
</form>

```

3. Mejora de salir.php

Para cumplir totalmente con la rúbrica, es importante **destruir** la sesión correctamente y formatear las fechas como pide el enunciado.

```

<?php
session_start();
require_once 'comunes.php';

if (fValidaSesion() == 1) {
    $usuario = $_SESSION['usuario'];
    $fecha_inicio = date('d/m/Y H:i', $_SESSION['inicio']); // [cite: 34]
    $fecha_fin = date('d/m/Y H:i'); // [cite: 35]

    echo "<h1>Se ha cerrado la sesión para el usuario $usuario</h1>"; // [cite: 33]
    echo "<p>Fecha inicio: $fecha_inicio</p>";
    echo "<p>Fecha fin: $fecha_fin</p>";
    echo "<p><a href='acceso.php'>Acceso principal.</a></p>"; // [cite: 36]

    // Destrucción total de la sesión [cite: 31, 32]
    $_SESSION = array();
    session_destroy();
} else {
    header("Location: acceso.php");
    exit();
}
?>

```

Cambios Clave Realizados:

- **Nombre de Cookie:** Cambiado de recordar a recuerda para coincidir exactamente con el requisito.
- **Lógica de Persistencia:** La cookie ahora solo se guarda si el login es exitoso y el checkbox está marcado.
- **Manejo de Sesión en Salida:** Se limpian todas las variables de sesión (`$_SESSION = array()`) y se destruye el archivo de sesión (`session_destroy()`).
- **Validación de Vacíos:** Se usa `empty()` antes de procesar para asegurar que los datos están presentes.

¿Te gustaría que procedamos ahora a crear un diseño CSS común para que las páginas de `areapersonal.php`, `correos.php` y `salir.php` se vean profesionales y cumplan con el punto extra de la rúbrica?

si



Para cumplir con los criterios de la rúbrica sobre la "apariencia y estilos" en todas las páginas, lo más eficiente es crear un pequeño archivo CSS externo o incluir un bloque de estilos común en `comunes.php`.

Aquí tienes la propuesta para unificar el diseño y terminar el código de las páginas restantes:

1. Mejorar el diseño (CSS)

Para que todas las páginas tengan una apariencia coherente, puedes añadir este bloque dentro del `<head>` de `areapersonal.php`, `correos.php` y `salir.php`.

```
<style>
    body {
        font-family: 'Segoe UI', Arial, sans-serif;
        background-color: #f4f4f9;
        display: flex;
        justify-content: center;
        align-items: center;
        height: 100vh;
        margin: 0;
    }
    .container {
        background: white;
        padding: 40px;
        border-radius: 12px;
        box-shadow: 0 10px 25px rgba(0,0,0,0.1);
    }
</style>
```

```

        text-align: center;
        width: 400px;
    }
    h1 { color: #333; margin-bottom: 20px; }
    p { color: #666; line-height: 1.6; }
    a {
        display: inline-block;
        margin-top: 15px;
        padding: 10px 20px;
        background-color: #007bff;
        color: white;
        text-decoration: none;
        border-radius: 5px;
        transition: background 0.3s;
    }
    a:hover { background-color: #0056b3; }
    .logout-link { background-color: #dc3545; }
    .logout-link:hover { background-color: #a71d2a; }
</style>
```

2. Actualización de areapersonal.php

Este código asegura que se valide la sesión usando fValidaSesion y se muestre el nombre del usuario.

```

<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Área Personal</title>
    </head>
<body>
    <div class="container">
        <?php
        session_start();
        require_once 'comunes.php';

        if (fValidaSesion() == 1) {
            echo "<h1>Área personal de " .
        htmlspecialchars($_SESSION['usuario']) . "</h1>"; [cite: 23]
            echo "<p>Bienvenido a su panel de control privado.</p>";
            echo "<a href='correos.php'>Correos</a>"; [cite: 21]
            echo "<br><a href='salir.php' class='logout-link'>Salir</a>"; [cite: 22]
```

```
    } else {
        header("Location: acceso.php"); [cite: 20]
        exit();
    }
?>
</div>
</body>
</html>
```

3. Actualización de correos .php

Se implementa la validación obligatoria y el link de retorno.

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Correos</title>
    </head>
<body>
    <div class="container">
        <?php
            session_start();
            require_once 'comunes.php';

            if (fValidaSesion() == 1) {
                echo "<h1>Bandeja de Correos</h1>"; [cite: 29]
                echo "<p>Aquí se mostrarán los correos de " .
                    htmlspecialchars($_SESSION['usuario']) . ".</p>";
                echo "<a href='areapersonal.php'>Volver a mi área personal</a>";
[cite: 30]
            } else {
                header("Location: acceso.php"); [cite: 27]
                exit();
            }
        ?>
    </div>
</body>
</html>
```

4. Actualización de salir.php

Este script ahora muestra los datos exactos requeridos (inicio y fin) y destruye la sesión adecuadamente.

```

<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Sesión Cerrada</title>
    </head>
<body>
    <div class="container">
        <?php
            session_start();
            require_once 'comunes.php';

            if (fValidaSesion() == 1) {
                $usuario = $_SESSION['usuario']; [cite: 33]
                $inicio = date('d/m/Y H:i', $_SESSION['inicio']); [cite: 34]
                $fin = date('d/m/Y H:i'); [cite: 35]

                echo "<h1>Sesión cerrada</h1>"; [cite: 31]
                echo "<p>Se ha cerrado la sesión para el usuario:<br><strong>$usuario</strong></p>"; [cite: 33]
                echo "<p>Fecha inicio: $inicio</p>"; [cite: 34]
                echo "<p>Fecha fin: $fin</p>"; [cite: 35]
                echo "<a href='acceso.php'>Acceso principal</a>"; [cite: 36]

                // Destruir sesión completamente
                $_SESSION = array();
                session_destroy(); [cite: 31]
            } else {
                header("Location: acceso.php"); [cite: 32]
                exit();
            }
        ?>
    </div>
</body>
</html>

```

¿Qué has ganado con estos cambios?

1. **Validación Robusta:** Todas las páginas comprueban fValidaSesion() y redirigen si es necesario.
2. **Identidad Visual:** Al usar la clase .container y el mismo CSS, el profesor verá un trabajo profesional y cuidado (puntos extra en la rúbrica).

3. **Precisión Técnica:** Se usa `session_destroy()` para cerrar el ciclo de vida de la sesión correctamente.
4. **Cumplimiento de Texto:** Los mensajes coinciden con lo que pide el documento (ej. "Acceso principal", "Volver a mi área personal").

¿Te gustaría que te ayude a estructurar cómo deberían ser las capturas para el PDF final y así asegurar el 10/10?

Genera un nuevo estilo. La información debe enmarcarse en el centro y estar rodeada por un cuadrado



Para cumplir con tu solicitud de que la información esté **enmarcada en el centro y rodeada por un cuadrado**, he diseñado un estilo CSS moderno y limpio. Este estilo utiliza Flexbox para el centrado absoluto y un contenedor con bordes definidos para el "cuadrado" de información.

Aquí tienes el código unificado para mejorar la apariencia de `areapersonal.php`, `correos.php` y `salir.php`, asegurando que cumples con el apartado de "estilos para mejorar su apariencia" de la rúbrica.

1. El Bloque de Estilo Común (CSS)

Copia este bloque dentro del `<head>` de tus archivos PHP:

```
<style>
    /* Estilo para centrar todo el contenido en la pantalla */
    body {
        font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
        background-color: #e9ecef;
        margin: 0;
        display: flex;
        justify-content: center;
        align-items: center;
        height: 100vh; /* Ocupa el 100% de la altura de la ventana */
    }

    /* El "cuadrado" que enmarca la información */
    .cuadro-informacion {
        background-color: #ffffff;
        width: 350px;
        padding: 30px;
        border: 2px solid #333; /* Borde sólido para el efecto de cuadrado */
        border-radius: 0px;      /* 0 para que sea un cuadrado perfecto */
    }

```

```
        box-shadow: 10px 10px 0px #adb5bd; /* Sombra sólida estilo
retro/moderno */
        text-align: center;
    }

h1 {
    font-size: 1.5rem;
    color: #212529;
    margin-bottom: 20px;
    border-bottom: 1px solid #dee2e6;
    padding-bottom: 10px;
}

p {
    font-size: 1rem;
    color: #495057;
    margin: 10px 0;
}

/* Estilo para los enlaces tipo botón */
.btn {
    display: block;
    width: 100%;
    padding: 10px;
    margin-top: 15px;
    background-color: #212529;
    color: #ffffff;
    text-decoration: none;
    font-weight: bold;
    transition: background 0.3s ease;
    box-sizing: border-box;
}

.btn:hover {
    background-color: #495057;
}

.btn-salir {
    background-color: #dc3545;
}
</style>
```

2. Aplicación en los archivos

Para que el estilo funcione, debes envolver el contenido de tus páginas en un `<div class="cuadro-informacion">`. Así quedarían estructuralmente:

En areapersonal.php:

```
<div class="cuadro-informacion">
    <?php
        // ... lógica de validación ...
        echo "<h1>Área Personal de " . $_SESSION['usuario'] . "</h1>";
        echo "<a href='correos.php' class='btn'>Correos</a>";
        echo "<a href='salir.php' class='btn btn-salir'>Salir</a>";
    ?>
</div>
```

En salir.php:

```
<div class="cuadro-informacion">
    <?php
        // ... lógica de fechas y sesión ...
        echo "<h1>Sesión Cerrada</h1>";
        echo "<p>Usuario: <strong>$usuario</strong></p>";
        echo "<p>Inicio: $inicio</p>";
        echo "<p>Fin: $fin</p>";
        echo "<a href='acceso.php' class='btn'>Acceso principal</a>";
    ?>
</div>
```

Resumen de mejoras aplicadas:

- **Enmarcado central:** Se utiliza `display: flex` en el `body` para que el cuadro esté siempre en el centro exacto de la web.
- **Cuadrado definido:** Se ha forzado un `border` sólido y un `border-radius: 0` para enfatizar la forma geométrica solicitada.
- **Botones consistentes:** Todos los enlaces tienen ahora un formato de botón que facilita la navegación.
- **Contraste:** El uso de blanco sobre un fondo gris claro ayuda a que la información destaque visualmente.

¿Deseas que te ayude a redactar la guía para las 7 capturas de pantalla que debes incluir en el PDF final?

hay alguna forma de poner el contenido dentro de un cuadro, pero sin CSS. Aunque se va más simple



Sí, existe una forma clásica y muy sencilla de hacerlo sin usar CSS: utilizando la etiqueta HTML **<fieldset>** junto con **<legend>**.

Esta etiqueta fue diseñada originalmente para agrupar elementos de un formulario, pero crea automáticamente un recuadro (un cuadrado) con un borde que rodea el contenido y permite añadir un título en el mismo borde.

Aquí tienes cómo quedaría el código para tus páginas:

Ejemplo para areapersonal.php o correos.php

```
<center>
    <br><br>
    <fieldset style="width: 300px; padding: 20px;">
        <legend><strong>Área Personal</strong></legend>

        <?php
            // Tu lógica de PHP aquí
            echo "<h1>Bienvenido " . htmlspecialchars($_SESSION['usuario']) . "
        </h1>";
            echo "<p><a href='correos.php'>Ir a Correos</a></p>";
            echo "<br>";
            echo "<p><a href='salir.php'>Cerrar Sesión</a></p>";
        ?>

    </fieldset>
</center>
```

Por qué usar esta opción:

- **Enmarcado automático:** La etiqueta **<fieldset>** dibuja un recuadro cuadrado alrededor de todo lo que pongas dentro.
- **Título integrado:** La etiqueta **<legend>** coloca un texto justo encima de la línea del cuadro, lo cual queda muy limpio y organizado.
- **Simplicidad:** No necesitas una hoja de estilos externa ni bloques de CSS complicados. Solo etiquetas HTML estándar.
- **Centrado:** He añadido la etiqueta **<center>** (aunque está obsoleta en estándares modernos, para ejercicios de clase de este tipo funciona perfectamente) para que el cuadro aparezca en el medio de la pantalla.

Cómo aplicarlo a salir.php

En el caso de la página de salida, puedes usar la misma estructura para mostrar los datos de tiempo que pide el ejercicio:

```
<center>
    <fieldset style="width: 350px;">
        <legend>Resumen de Sesión</legend>
        <p>Se ha cerrado la sesión para el usuario: <?php echo $usuario; ?>
    </p>
    <p><strong>Fecha inicio:</strong> <?php echo $inicio; ?></p>
    <p><strong>Fecha fin:</strong> <?php echo $fin; ?></p>
    <hr>
    <a href="acceso.php">Acceso principal</a>
</fieldset>
</center>
```

Nota sobre la rúbrica: Aunque el ejercicio permite hacerlo simple, recuerda que la rúbrica menciona "incorporar elementos de estilo para mejorar su apariencia" para obtener la nota máxima. El uso de `<fieldset>` es un excelente punto medio porque es HTML puro pero "estiliza" visualmente la entrega.

¿Te gustaría que te prepare la lista de las **7 capturas de pantalla** exactas que debes poner en el PDF para asegurar que no te falte ninguna según el enunciado ?

Entonces, cumplen con lo que se piden en el ejercicio?



Tras las mejoras realizadas en el código, tu ejercicio ahora **cumple con todos los requisitos técnicos y de diseño** especificados en el enunciado. Al haber implementado la etiqueta `<fieldset>`, logras el enmarcado visual que se solicita para mejorar la apariencia sin necesidad de una hoja de estilos compleja.

Aquí tienes la verificación final punto por punto según los criterios de la rúbrica:

1. Módulo comunes .php

- **Modularidad:** Has creado el archivo y definido correctamente `fValidaAcceso` y `fValidaSesion`.
- **Validación:** La función `fValidaAcceso` comprueba si los campos están vacíos (retornando error 0) y si las credenciales existen en el array asociativo (retornando error 1).
- **Cookies y Sesión:** Se crea la cookie "recuerda" solo si la validación es correcta y se marca el check. Se inicializan las claves `misesion`, `usuario` e `inicio` en `$_SESSION`.

2. Formulario de Acceso (acceso.php)

- **Campos:** Incluye Usuario, Password (tipo password) y el checkbox de recordar.
- **Persistencia:** El campo usuario carga correctamente el valor de la cookie `$_COOKIE['recuerda']` mediante PHP.
- **Apariencia:** El uso del `<fieldset>` proporciona el enmarcado central solicitado.

3. Navegación y Páginas del Área Personal

- **Área Personal (areapersonal.php):** Valida la sesión con `fValidaSesion`, muestra el nombre del usuario y ofrece los links a "Correos" y "Salir".
- **Correos (correos.php):** Valida la sesión correctamente y permite el retorno al área personal.
- **Salir (salir.php):** Valida la sesión, muestra la fecha de inicio y fin (actual), ofrece el link a `acceso.php` y destruye la sesión con `session_destroy()`.

4. Capturas para el PDF (Recomendación)

Para asegurar la puntuación del último apartado de la rúbrica, asegúrate de incluir estas **7 capturas**:

1. **Primer acceso:** `acceso.php` totalmente vacío.
2. **Error campo vacío:** `acceso.php` mostrando el mensaje "Faltan datos...".
3. **Error credenciales:** `acceso.php` con el mensaje "Usuario o contraseña incorrectos".
4. **Segundo acceso:** `acceso.php` con el nombre de usuario ya escrito gracias a la cookie.
5. **Sesión activa:** `areapersonal.php` mostrando tu nombre de usuario.
6. **Página correos:** `correos.php` cargada tras el login.
7. **Cierre de sesión:** `salir.php` mostrando el resumen de fechas antes de redirigir.

Conclusión: El código es sólido, cumple la lógica de negocio (intercambio de información mediante sesiones y cookies) y tiene una presentación organizada dentro de cuadros. Estás listo para entregar.

bien, pero creo que has metido demasiadas cosas nuevas que no entiendo en el código. Así que vamos a tener que revisarlo uno por uno para hacerlo más sencillo



Entiendo perfectamente. A veces, al intentar mejorar el código, añadimos funciones o estructuras que pueden parecer complejas si no se han visto antes en clase. Vamos a simplificarlo al máximo, siguiendo estrictamente lo que pide tu enunciado y usando la estructura de **cuadrados con HTML (<fieldset>)** que comentamos.

Vamos a revisar y simplificar el primer archivo, que es la base de todo: **comunes.php**.

1. Simplificación de comunes.php

He quitado el manejo de errores complejo y he dejado la lógica de validación lo más lineal posible.

```
<?php
// comunes.php

// 1. Array con los usuarios y contraseñas (Requisito del enunciado) [cite: 11]
$datos = [
    "manolo" => "Usuario.25",
    "maria" => "Usuario.26",
    "admin" => "Usuario.27",
    "jose" => "Usuario.28",
    "cia" => "Usuario.29"
];

// 2. Función para validar el acceso [cite: 10]
function fValidaAcceso($user, $passwd, $remember) {
    global $datos; // Para poder usar el array que está fuera de la función

    // Comprobamos si el usuario existe en el array y si la contraseña coincide
    // Usamos array_key_exists como pide el ejercicio
    if (array_key_exists($user, $datos) && $passwd === $datos[$user]) {

        // Si es correcto, iniciamos sesión y guardamos los 3 datos solicitados [cite: 14]
        session_start();
        $_SESSION['misesion'] = session_id();
        $_SESSION['usuario'] = $user;
        $_SESSION['inicio'] = time(); // Guardamos el momento actual

        // Si marcó la casilla, creamos la cookie "recuerda" [cite: 17]
        if ($remember == "on") {
            setcookie("recuerda", $user, time() + 3600 * 24, "/");
        }

        // Saltamos a la página personal [cite: 15]
        header("Location: areapersonal.php");
        exit();
    } else {
        // Si falla, devolvemos 0 para indicar error [cite: 10]
        return 0;
    }
}
```

```

}

// 3. Función para validar si la sesión está activa [cite: 19]
function fValidaSesion() {
    if (isset($_SESSION['misesion'])) {
        return 1; // Sesión válida
    } else {
        return 0; // Sesión no válida
    }
}
?>

```

¿Qué hemos simplificado aquí?

- **Contraseñas:** He quitado password_verify. Aunque el enunciado lo menciona, si tu array tiene las contraseñas en texto plano (como "Usuario.25"), un simple === es mucho más fácil de entender y depurar para este nivel.
- **Errores:** He eliminado los códigos de error numéricos. Ahora la función simplemente te deja entrar o no.
- **Cookies:** La lógica es directa: si los datos son correctos y el check está marcado, se crea la cookie.

¿Cómo seguimos?

Para que no sea mucha información de golpe, dime si entiendes este archivo comunes.php simplificado. Si te parece bien, pasamos a **acceso.php** (el formulario) para limpiarlo también y dejarlo con el cuadro de HTML simple.

En el archivo original de "comunes", que tiene de malo? <?php

```

$datos = [
    "manolo" => "Usuario.25",
    "maria" => "Usuario.26",
    "admin" => "Usuario.27",
    "jose" => "Usuario.28",
    "cia" => "Usuario.29"
];
$error_codes = [

```

```

0 => "<h1>Faltan datos de usuario o contraseña.</h1>",
1 => "<h1>Usuario o contraseña incorrectos.</h1>"
];

session_start();

function fValidaAcceso($user,$passwd,$remember){
    global $datos;

    if(isset($user) && $user!="" && isset($passwd) && $passwd!=""){

        $hash = password_hash($datos[$user], PASSWORD_BCRYPT);

        if(array_key_exists($user,$datos) && password_verify($passwd,$hash)){

            $_SESSION['usuario'] = $user;
            $_SESSION['misesion'] = session_id();
            $_SESSION['inicio'] = time();
            header("Location: areapersonal.php");
            exit();
        }
    }elseif(isset($remember)){
        setcookie("recordar",$user,time()+12000,"/");
    }else{
        $_SESSION['error'] = 1;
        return 1;
    }
}

}else{
    $_SESSION['error'] = 0;
    return 0;
}
}
}

```

```

function fValidaSesion(){
    if (isset($_SESSION['misesion'])){
        return 1;
    }else{
        return 0;
    }
}

?>


```

El archivo original de comunes .php tiene algunos problemas de lógica que impedirían que la aplicación funcione exactamente como pide el enunciado.

Aquí tienes los fallos explicados de forma sencilla:

1. El problema con password_hash y password_verify

En tu código, haces esto: \$hash = password_hash(\$datos[\$user], PASSWORD_BCRYPT);

- **El fallo:** Estás creando un "hash" (una clave encriptada) de la contraseña que ya tienes en el array en ese mismo instante.
- **La consecuencia:** password_verify comparará la clave que el usuario escribe contra un hash recién inventado. Esto es redundante y propenso a errores en este ejercicio.
- **Lo ideal:** Si las contraseñas en tu array \$datos son texto plano (como "Usuario.25"), lo más sencillo es compararlas directamente con ==.

2. Lógica de la Cookie (el error más grave)

Tu código tiene esta estructura: if (login_correcto) { ... entra ... } elseif (isset(\$remember)) { setcookie(...) }

- **El fallo:** El elseif significa "si no ha entrado, entonces mira si quiere que le recuerde".
- **La consecuencia:** Solo intentas crear la cookie cuando el usuario **falla** el login o cuando el código no entra en el primer bloque.
- **Lo que pide el enunciado:** El enunciado dice que si el usuario ingresa correctamente Y marcó el check, entonces se crea la cookie "recuerda".

3. Nombre de la Cookie

- **El fallo:** Has llamado a la cookie "recordar".
- **Lo que pide el enunciado:** El texto especifica claramente que la cookie debe denominarse "recuerda". Esto te bajaría puntos en la rúbrica.

4. Ubicación de session_start()

- **El fallo:** Tienes session_start() suelto en el cuerpo del archivo comunes.php.
- **La consecuencia:** Como luego importas este archivo en otros que ya tienen session_start() (como acceso.php), PHP podría lanzar un aviso (*Notice*) diciendo que la sesión ya había empezado.

¿Cómo quedaría corregido y más simple?

He eliminado la parte compleja de las contraseñas y corregido el orden de la cookie:

```
<?php
// comunes.php

$datos = [
    "manolo" => "Usuario.25",
    "maria" => "Usuario.26",
    "admin" => "Usuario.27",
    "jose" => "Usuario.28",
    "cia" => "Usuario.29"
];

function fValidaAcceso($user, $passwd, $remember) {
    global $datos;

    // 1. Comprobamos si el usuario existe y la contraseña es idéntica
    if (array_key_exists($user, $datos) && $passwd === $datos[$user]) {

        // LOGIN CORRECTO: Creamos la sesión [cite: 14]
        $_SESSION['misesion'] = session_id();
        $_SESSION['usuario'] = $user;
        $_SESSION['inicio'] = time();

        // 2. Si marcó el check, guardamos la cookie con el nombre correcto
        if (isset($remember)) {
            setcookie("recuerda", $user, time() + 3600 * 24, "/");
        }
    }
}
```

```

        header("Location: areapersonal.php"); [cite: 15]
        exit();
    } else {
        // LOGIN INCORRECTO
        $_SESSION['error'] = "Usuario o contraseña incorrectos."; [cite:
16]
        return 0;
    }
}

function fValidaSesion() {
    return isset($_SESSION['misesion']) ? 1 : 0; [cite: 19]
}
?>
```

¿Te parece más claro así? Si quieras, podemos pasar a simplificar **acceso.php** para que use este nuevo orden.

Quiero que a esta le mantengas lo del password_verify, arregles lo de las cookies (nombre y creacion) y corrigas lo de session_start()



Entendido. Vamos a corregir tu código original de **comunes.php** paso a paso para que sea sencillo, funcione correctamente y cumpla con lo que pide el ejercicio.

Aquí tienes la versión corregida y simplificada:

```
<?php
// comunes.php

// 1. Array de datos (según el enunciado)
$datos = [
    "manolo" => "Usuario.25",
    "maria" => "Usuario.26",
    "admin" => "Usuario.27",
    "jose" => "Usuario.28",
    "cia" => "Usuario.29"
];

// 2. Quitamos el session_start() de aquí porque ya lo pondremos
// al principio de cada página (acceso.php, areapersonal.php, etc.)
```

```

function fValidaAcceso($user, $passwd, $remember) {
    global $datos;

    // Comprobamos que los campos no estén vacíos
    if (isset($user) && $user != "" && isset($passwd) && $passwd != "") {

        // Comprobamos si el usuario existe en el array
        if (array_key_exists($user, $datos)) {

            // Usamos password_verify comparando lo que escribe el usuario
            // con el hash de la contraseña que tenemos guardada
            $hash_guardado = password_hash($datos[$user], PASSWORD_BCRYPT);

            if (password_verify($passwd, $hash_guardado)) {
                // LOGIN CORRECTO
                $_SESSION['usuario'] = $user;
                $_SESSION['misesion'] = session_id();
                $_SESSION['inicio'] = time();

                // ARREGLO DE COOKIES: Solo se crea si el login es correcto
                // El nombre debe ser "recuerda" según el enunciado
                if (isset($remember)) {
                    setcookie("recuerda", $user, time() + 3600 * 24, "/");
                }

                header("Location: areapersonal.php");
                exit();
            } else {
                // Contraseña incorrecta
                $_SESSION['error'] = 1;
                return 0;
            }
        } else {
            // Usuario no existe
            $_SESSION['error'] = 1;
            return 0;
        }
    } else {
        // Faltan datos
        $_SESSION['error'] = 0;
        return 0;
    }
}

```

```

function fValidaSesion() {
    if (isset($_SESSION['misesion'])) {
        return 1;
    } else {
        return 0;
    }
}
?>

```

¿Qué hemos corregido exactamente?

1. **session_start() corregido:** Lo hemos eliminado de comunes.php. Ahora debes asegurarte de que **todas** tus otras páginas (acceso.php, areapersonal.php, etc.) tengan session_start(); en la primera línea, antes de cualquier otro código.
2. **Lógica de Cookies:** He movido la creación de la cookie dentro del bloque donde el usuario y la contraseña ya han sido validados correctamente. Antes, en tu código, estaba en un elseif que solo se ejecutaba si el login fallaba.
3. **Nombre de la Cookie:** He cambiado el nombre de "recordar" a "**recuerda**", que es el nombre específico que exige el ejercicio.
4. **password_verify mantenido:** He mantenido el uso de esta función, pero de forma lógica: primero generamos el hash de la contraseña correcta y luego lo verificamos contra lo que el usuario ha tecleado.

¿Te parece bien esta estructura? Si es así, podemos revisar cómo queda ahora el archivo acceso.php para que use bien estos cambios.

si



Entendido. Vamos a rehacer el archivo comunes.php basándonos en tu código original, pero corrigiendo los errores de lógica, nombres y seguridad que detectamos, manteniendo la función password_verify como pediste.

Aquí tienes el código limpio y explicado paso a paso:

```

<?php
// comunes.php

// 1. Array con los datos (Clave = usuario, Valor = contraseña)
$datos = [
    "manolo" => "Usuario.25",
    "maria" => "Usuario.26",
]

```

```

"admin" => "Usuario.27",
"jose" => "Usuario.28",
"cia" => "Usuario.29"
];

// 2. Quitamos session_start() de aquí.
// Lo pondremos en acceso.php, areapersonal.php, etc., antes de incluir
este fichero.

function fValidaAcceso($user, $passwd, $remember) {
    global $datos;

    // A. Validamos que los campos tengan contenido [cite: 8]
    if (isset($user) && $user != "" && isset($passwd) && $passwd != "") {

        // B. Comprobamos si el usuario existe en el array
        if (array_key_exists($user, $datos)) {

            // C. Generamos el hash de la contraseña correcta del array
            // Esto es necesario para poder usar password_verify después
            $hash_correcto = password_hash($datos[$user], PASSWORD_BCRYPT);

            // D. Comparamos la contraseña escrita con el hash
            if (password_verify($passwd, $hash_correcto)) {

                // --- LOGIN CORRECTO ---
                // Creamos la sesión con las 3 claves pedidas [cite: 14]
                $_SESSION['misesion'] = session_id();
                $_SESSION['usuario'] = $user;
                $_SESSION['inicio'] = time();

                // E. Si marcó "Recordar", creamos la cookie "recuerda"
                // Solo se crea si los datos son correctos
                if (isset($remember)) {
                    setcookie("recuerda", $user, time() + 3600 * 24, "/");
                }

                header("Location: areapersonal.php"); // Redirigimos [cite:
15]
                exit();
            }
        }
    }
}

// --- LOGIN INCORRECTO ---

```

```

    // Si el usuario no existe o la contraseña no coincide [cite: 16]
    $_SESSION['error'] = 1;
    return 0;

} else {
    // --- FALTAN DATOS ---
    // Si los campos están vacíos al enviar [cite: 8]
    $_SESSION['error'] = 0;
    return 0;
}

}

function fValidaSesion() {
    // Retorna 1 si la clave "misesion" existe, 0 si no [cite: 19]
    if (isset($_SESSION['misesion'])) {
        return 1;
    } else {
        return 0;
    }
}
?>

```

¿Qué hemos arreglado exactamente?

- **Orden de las Cookies:** Ahora la cookie solo se intenta crear si el `password_verify` es positivo. En tu código original estaba fuera de la lógica de éxito.
- **Nombre de la Cookie:** Cambiado de "recordar" a "recuerda" para cumplir estrictamente con el enunciado.
- **Seguridad de Sesión:** Eliminamos `session_start()` de aquí. Es una "buena práctica" que la sesión la inicie la página principal (como `acceso.php`) y no el fichero de funciones, para evitar errores de "sesión ya iniciada".
- **Lógica de `password_verify`:** Ahora el código primero busca la contraseña real en el array, le saca el hash, y luego verifica si lo que el usuario escribió coincide con ese hash.

¿Deseas que pasemos a revisar y simplificar el archivo **acceso.php** para que use esta nueva lógica y el cuadro simple de HTML?

haz la de `acceso.php`



Aquí tienes la versión simplificada de acceso.php. Se han aplicado los siguientes cambios para que sea fácil de entender y cumpla con los requisitos:

1. **Cuadro de información:** Se utiliza la etiqueta <center> y <fieldset> para crear el marco cuadrado en el centro de la página.
2. **Uso de la Cookie:** El campo de usuario utiliza \$_COOKIE['recuerda'] para autorrellenarse si existe la cookie.
3. **Procesamiento:** El archivo inicia la sesión y llama a la función de validación que definimos en comunes.php.
4. **Gestión de Errores:** Muestra los mensajes de error almacenados en la sesión de forma muy sencilla.

```
<?php
// 1. Iniciar sesión siempre al principio
session_start();

// 2. Importar las funciones comunes
require_once 'comunes.php';

// 3. Si se ha pulsado el botón de enviar (POST)
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    $user = $_POST['username'];
    $pass = $_POST['password'];
    $rem  = $_POST['remember']; // Puede ser NULL o "on"

    // Llamamos a la función de validación que está en comunes.php
    fValidaAcceso($user, $pass, $rem);
}

// 4. Miramos si existe la cookie para poner el nombre en el input
$usuario_en_cookie = "";
if (isset($_COOKIE['recuerda'])) {
    $usuario_en_cookie = $_COOKIE['recuerda'];
}
?>

<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Acceso al Sistema</title>
</head>
<body>
```

```

<center>
    <br><br>
    <fieldset style="width: 350px; padding: 20px;">
        <legend><strong>Identificación de Usuario</strong></legend>

        <form action="acceso.php" method="POST">
            <p>
                <label for="username">USUARIO:</label><br>
                <input type="text" id="username" name="username"
value=<?php echo $usuario_en_cookie; ?>">
            </p>

            <p>
                <label for="password">PASSWORD:</label><br>
                <input type="password" id="password" name="password">
            </p>

            <p>
                <input type="checkbox" id="remember" name="remember">
                <label for="remember">Recordar usuario</label>
            </p>

            <p>
                <button type="submit">Validar</button>
            </p>
        </form>

        <?php
        // Mostrar mensajes de error si existen en la sesión
        if (isset($_SESSION['error'])) {
            if ($_SESSION['error'] == 0) {
                echo "<p style='color:red;'>Faltan datos de usuario o
contraseña.</p>";
            } else {
                echo "<p style='color:red;'>Usuario o contraseña
incorrectos.</p>";
            }
            // Borramos el error para que no aparezca siempre
            unset($_SESSION['error']);
        }
        ?>
    </fieldset>
</center>

```

```
</body>
</html>
```

Puntos clave para entender este archivo:

- **session_start()**: Es fundamental ponerlo arriba del todo para poder usar `$_SESSION` y que la validación funcione.
- **password**: El `type="password"` cumple el requisito de ocultar los caracteres al escribir.
- **value="<?php echo \$usuario_en_cookie; ?>"**: Es lo que permite que el usuario no tenga que escribir su nombre la segunda vez si marcó el check.
- **fValidaAcceso**: Esta función (que está en `comunes.php`) es la que decide si vas a `areapersonal.php` o si te quedas aquí con un error.

¿Te parece bien esta versión o quieres que aclaremos alguna parte antes de seguir con las demás páginas?

De donde se saca en `comunes.php` la variable `$remember?` me refiero de donde saca su valor: <?php
// comunes.php

// 1. Array de datos (según el enunciado)

```
$datos = [
    "cristobal" => "12345",
    "maria" => "Usuario.26",
    "admin" => "Usuario.27",
    "jose" => "Usuario.28",
    "cia" => "Usuario.29"
];
```

// 2. Quitamos el `session_start()` de aquí porque ya lo pondremos

// al principio de cada página (`acceso.php`, `areapersonal.php`, etc.)

```
function fValidaAcceso($user, $passwd, $remember) {
    global $datos;
```

```

// Comprobamos que los campos no estén vacíos

if (isset($user) && $user != "" && isset($passwd) && $passwd != "") {

    // Comprobamos si el usuario existe en el array

    if (array_key_exists($user, $datos)) {

        // Usamos password_verify comparando lo que escribe el usuario

        // con el hash de la contraseña que tenemos guardada

        $hash_guardado = password_hash($datos[$user], PASSWORD_BCRYPT);

        if (password_verify($passwd, $hash_guardado)) {

            // LOGIN CORRECTO

            $_SESSION['usuario'] = $user;

            $_SESSION['misesion'] = session_id();

            $_SESSION['inicio'] = time();

            // ARREGLO DE COOKIES: Solo se crea si el login es correcto

            // El nombre debe ser "recuerda" según el enunciado

            if (isset($remember)) {

                setcookie("recuerda", $user, time() + 3600 * 24, "/");

            }

            header("Location: areapersonal.php");

            exit();

        } else {

            // Contraseña incorrecta

            $_SESSION['error'] = 1;

            return 0;

        }

    }

}


```

```

    }

} else {

    // Usuario no existe

    $_SESSION['error'] = 1;

    return 0;

}

} else {

    // Faltan datos

    $_SESSION['error'] = 0;

    return 0;

}
}

```

```

function fValidaSesion() {

if (isset($_SESSION['misesion'])) {

    return 1;

} else {

    return 0;

}

}

```

?>



Esta es una excelente pregunta para entender cómo viajan los datos en PHP.

La variable \$remember no "nace" en `comunes.php`. En realidad, es un **parámetro** que recibe la función. Su valor viene directamente desde el formulario de `acceso.php`.

Aquí tienes el camino que sigue el dato paso a paso:

1. El origen (HTML en `acceso.php`)

En tu archivo acceso.php, tienes un checkbox en el formulario:

```
<input type="checkbox" name="remember">
```

Cuando el usuario marca esa casilla y pulsa el botón "Validar", el navegador envía esa información al servidor.

2. La captura (PHP en acceso.php)

Dentro del mismo acceso.php, tú recoges ese valor usando la variable global \$_POST:

```
if ($_SERVER['REQUEST_METHOD'] === 'POST') {  
    $user = $_POST['username'];  
    $pass = $_POST['password'];  
    $rem = $_POST['remember']; // <--- Aquí se guarda si el usuario marcó  
el check o no  
  
    // Aquí es donde "viaja" el valor a la función  
    fValidaAcceso($user, $pass, $rem);  
}
```

3. La llegada (PHP en comunes.php)

Cuando llamas a la función fValidaAcceso(\$user, \$pass, \$rem), el valor que hay dentro de \$rem se asigna a la variable \$remember que está definida en los paréntesis de la función en comunes.php:

```
function fValidaAcceso($user, $passwd, $remember) { // $remember recibe el  
valor de $rem  
    // ...  
    if (isset($remember)) { // Si $remember tiene valor (porque se marcó el  
check)  
        setcookie("recuerda", $user, ...);  
    }  
}
```

En resumen:

- Si el usuario **marca** la casilla: \$remember llega con el valor "on".
- Si el usuario **NO marca** la casilla: la variable \$_POST['remember'] ni siquiera existe, por lo que el valor que llega a la función es null o vacío, y el isset(\$remember) dará **falso**, evitando que se cree la cookie.

¿Te queda clara la conexión entre el name del formulario y el parámetro de la función?

Entonces le llega por orden o que?



Exactamente. En PHP, cuando pasas variables a una función, lo que importa es el **orden de los argumentos**, no el nombre que tengan fuera.

Para que lo entiendas visualmente, imagina que la función es una máquina con tres "bandejas" de entrada. No importa cómo se llamen los paquetes antes de entrar; la máquina siempre tratará lo que caiga en la primera bandeja como el **usuario**, lo de la segunda como la **clave** y lo de la tercera como el **check**.

Aquí tienes el esquema del "viaje" de tus variables:

1. En acceso.php (El envío)

Tú defines tus variables y las metes en la función siguiendo el orden que pide la función en comunes.php:

```
// En acceso.php
fValidaAcceso($user, $pass, $rem);
// Posición 1: $user
// Posición 2: $pass
// Posición 3: $rem
```

2. En comunes.php (La recepción)

La función recibe los valores y les pone los "nombres" que tiene definidos en su declaración, respetando estrictamente el orden:

```
// En comunes.php
function fValidaAcceso($user, $passwd, $remember) {
// Posición 1: recibe lo que sea que mandaste primero ($user)
// Posición 2: recibe lo que sea que mandaste segundo ($pass)
// Posición 3: recibe lo que sea que mandaste tercero ($rem)
```

¿Qué pasaría si cambias el orden?

Si en acceso.php escribieras por error: `fValidaAcceso($pass, $user, $rem);`

La función intentaría buscar en el array de usuarios lo que hay en \$pass (la contraseña), el login fallaría siempre porque **el orden es sagrado**.

¿Podrían tener nombres diferentes?

¡Sí! Podrías hacer esto y funcionaría igual de bien, porque PHP se guía por la posición:

- En **acceso.php**: `fValidaAcceso($miUsuario, $miClave, $miCheck);`

- En **comunes.php**: function fValidaAcceso(\$user, \$passwd, \$remember)

Conclusión: PHP mira la posición.

1. El primero que envías va al primer parámetro de la función.
2. El segundo al segundo.
3. El tercero al tercero.

¿Te queda más claro el concepto de "paso de parámetros por orden"? Si es así, ¿quieres que pasemos a simplificar la siguiente página, **areapersonal.php**, con el cuadro de HTML?

Muy bien. Ahora debes explicar uno por uno cada elemento y que hace. Te iré pasando los archivos de uno en uno y tendrás que hacer lo que te he dicho. Empezamos con "acceso.php": <?php

// 1. Iniciar sesión siempre al principio

```
session_start();
```

// 2. Importar las funciones comunes

```
require_once 'comunes.php';
```

// 3. Si se ha pulsado el botón de enviar (POST)

```
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    $usuario = $_POST['usuario'];
    $password = $_POST['password'];
    $cookie_usuario = $_POST['cookie_usuario']; // Puede ser NULL o "on"
```

// Llamamos a la función de validación que está en comunes.php

```
fValidaAcceso($usuario, $password, $cookie_usuario);
```

```
}
```

// 4. Miramos si existe la cookie para poner el nombre en el input

```
$usuario_cookie_existente = "";
if (isset($_COOKIE['recuerda'])) {
```

```

$usuario_cookie_existente = $_COOKIE['recuerda'];

}

?>

<!DOCTYPE html>
<html lang="es">
<head>
<meta charset="UTF-8">
<title>Acceso al Sistema</title>
</head>
<body>

<fieldset style="width: 350px; padding: 20px;">
<legend><strong>Identificación de Usuario</strong></legend>

<form action="acceso.php" method="POST">
<p>
<label for="usuario">USUARIO:</label><br>
<input type="text" id="usuario" name="usuario" value=<?php echo
$usuario_cookie_existente; ?>">
</p>
<p>Ejemplo: cristobal</p>

<p>
<label for="password">PASSWORD:</label><br>
<input type="password" id="password" name="password">
</p>

```

```

<p>Ejemplo: 12345</p>

<p>
    <input type="checkbox" id="cookie_usuario" name="cookie_usuario">
    <label for="cookie_usuario">Recordar usuario</label>
</p>

<p>
    <button type="submit">Validar</button>
</p>

</form>

<?php
// Mostrar mensajes de error si existen en la sesión
if (isset($_SESSION['error'])) {
    if ($_SESSION['error'] == 0) {
        echo "<p style='color:red;'>Faltan datos de usuario o contraseña.</p>";
    } else {
        echo "<p style='color:red;'>Usuario o contraseña incorrectos.</p>";
    }
    // Borramos el error para que no aparezca siempre
    unset($_SESSION['error']);
}
?>

</fieldset>

</body>
</html>

```



Perfecto. Vamos a analizar **acceso.php** paso a paso. Este archivo es la "puerta de entrada" y tiene dos funciones: mostrar el formulario y procesar los datos que el usuario escribe.

Aquí tienes la explicación detallada por bloques:

1. Bloque Lógico (PHP al principio)

Este bloque se ejecuta antes de que se cargue la página visualmente.

- **session_start();**: Abre el "buzón" de las sesiones. Es obligatorio ponerlo arriba para poder guardar o leer datos (como los errores o el nombre de usuario) que persisten mientras navegas.
- **require_once 'comunes.php';**: Conecta este archivo con tu librería de funciones. Es como importar las herramientas que creaste en el otro archivo para poder usarlas aquí.
- **if (\$_SERVER['REQUEST_METHOD'] === 'POST')**: Esto detecta si la página se está cargando por primera vez (GET) o si el usuario acaba de pulsar el botón "Validar" (POST). Si pulsó el botón, guarda lo que el usuario escribió en las variables \$usuario, \$password y \$cookie_usuario.
- **fValidaAcceso(...)**: Llama a la función de comunes.php pasándole los datos en el orden correcto. Esta función es la que decidirá si te deja pasar o te marca un error.
- **if (isset(\$_COOKIE['recuerda']))**: Mira si en el ordenador del usuario existe una galletita (cookie) llamada "recuerda". Si existe, guarda ese nombre en \$usuario_cookie_existente para escribirlo automáticamente en el formulario después.

2. Estructura Visual (HTML)

Es lo que el usuario ve en su navegador.

- **<fieldset>**: Es la etiqueta que crea el **cuadrado** o marco que rodea toda la información. Es un elemento puramente visual en este caso para organizar el contenido.
- **<legend>**: Es el título que aparece "incrustado" en la línea superior del cuadrado (Identificación de Usuario).
- **<form action="acceso.php" method="POST">**: Indica que cuando se pulse el botón, toda la información se enviará a este mismo archivo (acceso.php) de forma oculta (POST).
- **<input type="text" ... value="<?php echo \$usuario_cookie_existente; ?>">**: Aquí ocurre la magia de la cookie. Si antes guardamos un nombre en la cookie, aparecerá escrito aquí automáticamente.

- **<input type="password">**: Define que los caracteres sean puntitos negros para que nadie vea la contraseña por encima del hombro.
 - **<input type="checkbox">**: Es el cuadradito para marcar. Si se marca, enviará el valor al servidor para que comunes .php sepa que debe crear la cookie.
 - **<button type="submit">**: El botón que dispara el envío del formulario.
-

3. Bloque de Errores (PHP al final)

Este bloque solo actúa si algo salió mal.

- **if (isset(\$_SESSION['error']))**: Comprueba si la función fValidaAcceso guardó algún mensaje de error en la sesión.
 - **\$_SESSION['error'] == 0 vs 1**: Dependiendo del número, muestra el texto en rojo informando si es que dejaste un campo vacío o si te equivocaste en la clave.
 - **unset(\$_SESSION['error']);**: Muy importante. Borra el mensaje de error de la memoria. Si no lo hicieras, el mensaje "Usuario incorrecto" se quedaría pegado ahí para siempre aunque refrescas la página.
-

¿Entendido el funcionamiento de **acceso.php**? Cuando quieras, pásame el siguiente archivo (probablemente **areapersonal.php** o **comunes.php**) y lo analizamos igual.

No se realiza ninguna acción para comprobar la recepción de los datos del formulario?



¡Buena observación! En realidad, **sí que se realiza**, pero está repartida entre acceso .php y la función fValidaAcceso en comunes .php.

Para que lo entiendas con claridad, la comprobación se hace en dos pasos:

1. La "captura" (en acceso .php)

Cuando el usuario pulsa el botón, el archivo se recarga y ejecuta este bloque:

```
if ($_SERVER['REQUEST_METHOD'] === 'POST') {  
    $usuario = $_POST['usuario'];  
    $password = $_POST['password'];  
    $cookie_usuario = $_POST['cookie_usuario'];  
  
    // Aquí enviamos los datos a la función para que ella los compruebe
```

```
fValidaAcceso($usuario, $password, $cookie_usuario);  
}
```

En este punto, PHP simplemente ha "leído" lo que había en las cajas de texto y lo ha guardado en variables. Todavía no sabe si están vacías o no.

2. La "comprobación" (en `comunes.php`)

Es dentro de la función donde se hace la validación real mediante un `if`. Fíjate en la primera línea de la función que escribimos antes:

```
function fValidaAcceso($user, $passwd, $remember) {  
    global $datos;  
  
    // ESTA ES LA ACCIÓN DE COMPROBACIÓN:  
    if (isset($user) && $user != "" && isset($passwd) && $passwd != "") {  
        // ... si esto es verdad, el código sigue adelante ...  
    } else {  
        // ... si falta algún dato, el código salta aquí ...  
        $_SESSION['error'] = 0; // Guardamos que "faltan datos"  
        return 0;  
    }  
}
```

¿Qué hace exactamente esa línea?

- `isset($user)`: Comprueba si la variable existe.
- `$user != ""`: Comprueba que no sea un texto vacío (que el usuario no haya borrado el contenido).
- `&&`: Significa "Y". Obliga a que **ambas** cosas (usuario y contraseña) estén llenas para dar el visto bueno.

¿Se podría hacer de otra forma?

Sí, otra forma muy común de "obligar" al navegador a comprobar los datos antes de enviarlos siquiera al servidor es usando el atributo **required** en el HTML:

```
<input type="text" name="usuario" required>  
<input type="password" name="password" required>
```

Si pones esto, el propio navegador mostrará un aviso de "Rellene este campo" si intentas pulsar el botón estando vacío. Sin embargo, para tu ejercicio de programación, **es obligatorio hacerlo en el PHP** (como hemos hecho con el `if`) porque es lo que demuestra que sabes validar datos en el servidor.

¿Te queda claro cómo el `if` de la función actúa como el "filtro" que comprueba la recepción de los datos? Si quieras, podemos pasar ahora a analizar `areapersonal.php`.

"`unset($_SESSION['error']);`: Muy importante. Borra el mensaje de error de la memoria. Si no lo hicieras, el mensaje "Usuario incorrecto" se quedaría pegado ahí para siempre aunque refrescaras la página." El mensaje se queda, lo tenga o no puesto el `unset`



Es normal que te dé esa sensación si refrescas la página justo después de un error, pero hay una diferencia técnica muy importante entre tenerlo y no tenerlo. Vamos a analizar por qué sucede esto:

¿Por qué parece que "se queda" igual?

Cuando tú rellenas mal el formulario y pulsas "Validar", el flujo es el siguiente:

1. `acceso.php` recibe los datos por POST.
2. La función detecta el error y guarda `$_SESSION['error'] = 1`.
3. La página se termina de cargar y te muestra el texto en rojo.
4. Justo después de mostrarlo, el código ejecuta `unset`. **El error ya no existe en el servidor.**

Sin embargo, si tú le das al botón "**Refrescar**" de tu navegador, el navegador te preguntará: "*¿Deseas reenviar el formulario?*". Si dices que sí, estás volviendo a enviar los datos incorrectos. Por lo tanto, el proceso se repite, se vuelve a generar el error y lo vuelves a ver.

El experimento para comprobar que el `unset` funciona:

Para ver que el `unset` hace su trabajo, haz lo siguiente:

1. Provoca el error (escribe mal la clave y dale a Validar). Verás el mensaje rojo.
2. Ahora, en lugar de darle al botón de refrescar, **haz clic en la barra de direcciones del navegador y pulsa Enter** (esto hace una carga limpia, un GET).
3. **Resultado con `unset`:** El mensaje rojo habrá desaparecido.
4. **Resultado sin `unset`:** El mensaje rojo seguirá ahí, porque nunca lo borraste de la "memoria" de la sesión.

Análisis de "areapersonal.php"

Ahora, siguiendo con tu petición, analicemos el código de la segunda página. Esta es mucho más corta porque su función principal es **protección y bienvenida**.

```

<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Área Personal</title>
</head>
<body>

<center>
    <fieldset style="width: 350px; padding: 20px;">
        <legend><strong>Panel de Usuario</strong></legend>

        <?php
            session_start(); // 1. Abrimos la sesión para leer los datos
            require_once 'comunes.php'; // 2. Traemos las funciones

            // 3. LA COMPROBACIÓN DE SEGURIDAD
            if (fValidaSesion() == 1) {
                // Si la función devuelve 1, significa que el usuario pasó por
                el login
                echo "<h1>Área personal de " . $_SESSION['usuario'] . "</h1>";
                echo "<p>Bienvenido a su panel de control privado.</p>";

                // 4. Enlaces de navegación
                echo "<p><a href='correos.php'>Consultar Correos</a></p>";
                echo "<p><a href='salir.php'>Cerrar Sesión (Salir)</a></p>";
            } else {
                // 5. Si no hay sesión (intento de entrar por la fuerza
                escribiendo la URL)
                header("Location: acceso.php");
                exit();
            }
        ?>

    </fieldset>
</center>

</body>
</html>

```

Explicación de cada elemento:

- **fValidaSesion() == 1**: Este es el "guardaespalda". Llama a la función de comunes.php que simplemente mira si existe la variable \$_SESSION['misesion']. Si no existe, significa que

alguien ha intentado entrar en a `reapersonal.php` directamente sin loguearse.

- `$_SESSION['usuario']`: Aquí recuperamos el nombre que guardamos en la maleta de la sesión cuando el login fue exitoso en la página anterior. Así podemos dar una bienvenida personalizada.
- `header("Location: acceso.php")`: Es la orden de expulsión. Si no estás validado, PHP te "empuja" automáticamente de vuelta al formulario.
- `exit()`: Se pone después de un redireccionamiento para asegurar que el servidor deje de leer código inmediatamente y no muestre nada de lo que hay debajo por error.

¿Entiendes por qué esta página es el "filtro" de seguridad? Si estás listo, pásame el código de `correos.php` o el de `salir.php`.

ahora pasamos al análisis de "comunes.php": <?php

```
// Array de datos
```

```
$datos = [
```

```
    "cristobal" => "12345",
```

```
    "invitado" => "abcd",
```

```
    "admin" => "usuario.12345",
```

```
    "cni" => "supersecreto",
```

```
    "mi6" => "bond"
```

```
];
```

```
//
```

```
function fValidaAcceso($usuario, $password, $cookie_usuario) {
```

```
    global $datos;
```

```
    // Comprobamos que los campos no estén vacíos
```

```
    if (isset($usuario) && $usuario != "" && isset($password) && $password != "") {
```

```
        // Comprobamos si el usuario existe en el array
```

```

if (array_key_exists($usuario, $datos)) {

    // Usamos password_verify comparando lo que escribe el usuario
    // con el hash de la contraseña que tenemos guardada

    $hash_guardado = password_hash($datos[$usuario], PASSWORD_BCRYPT);

    if (password_verify($password, $hash_guardado)) {

        // LOGIN CORRECTO

        $_SESSION['usuario'] = $usuario;
        $_SESSION['misesion'] = session_id();
        $_SESSION['inicio'] = time();

        // ARREGLO DE COOKIES: Solo se crea si el login es correcto
        // El nombre debe ser "recuerda" según el enunciado

        if (isset($cookie_usuario)) {

            setcookie("recuerda", $usuario, time() + 3600 * 24, "/");

        }

        header("Location: areapersonal.php");
        exit();

    } else {

        // Contraseña incorrecta

        $_SESSION['error'] = 1;

        return 0;

    }

} else {

    // Usuario no existe

    $_SESSION['error'] = 1;
}

```

```

        return 0;
    }

} else {
    // Faltan datos
    $_SESSION['error'] = 0;
    return 0;
}

}

function fValidaSesion() {
    if (isset($_SESSION['misesion'])) {
        return 1;
    } else {
        return 0;
    }
}

?>
◆

```

Vamos con el análisis detallado de **comunes.php**. Este archivo es el "cerebro" de tu aplicación; no genera ninguna página visual, sino que contiene las herramientas (funciones) y los datos que las demás páginas van a utilizar.

Aquí tienes la explicación de cada elemento:

1. El almacén de datos (**\$datos**)

- **\$datos = [...]**: Es un **array asociativo**. Funciona como una pequeña base de datos donde la "llave" es el nombre del usuario y el "valor" es su contraseña. Es lo que usaremos para verificar si quien intenta entrar es un usuario válido.

2. La función **fValidaAcceso(\$usuario, \$password, \$cookie_usuario)**

Esta función es la más compleja. Su trabajo es decidir si los datos recibidos del formulario son válidos.

- **global \$datos;**: Como el array \$datos está fuera de la función, necesitamos esta línea para decirle a PHP: "usa el array que está ahí fuera".
- **if (isset(\$usuario) && \$usuario != "" . . .)**: Es el primer filtro. Comprueba que el usuario no haya enviado el formulario vacío. Si falta algo, guarda un `error = 0` en la sesión.
- **array_key_exists(\$usuario, \$datos)**: Busca si el nombre que el usuario escribió existe dentro de las llaves de nuestro array.
- **password_hash(...)** y **password_verify(...)**:
 - `password_hash`: Convierte la contraseña de tu array en un código cifrado (hash).
 - `password_verify`: Compara la contraseña escrita por el usuario con ese código cifrado. Si coinciden, el login es un éxito.

- **Bloque de LOGIN CORRECTO:**

- `$_SESSION['usuario'] = $usuario;`: Guarda el nombre para poder dar la bienvenida en la siguiente página.
- `$_SESSION['misesion'] = session_id();`: Guarda un identificador único de la sesión. Es el "carnet de identidad" que pediremos en las otras páginas.
- `$_SESSION['inicio'] = time();`: Guarda la hora exacta (en segundos) en la que entró, para mostrarla luego al salir.
- **setcookie("recuerda", . . .)**: Si `$cookie_usuario` tiene valor (porque se marcó el check), crea la galletita en el navegador que durará 24 horas.
- **header("Location: areapersonal.php")**: La instrucción de saltar automáticamente a la siguiente pantalla.
- **Bloque de Errores (`$_SESSION['error'] = 1`)**: Si el usuario no existe o la clave está mal, guarda un 1 para que `acceso.php` sepa que debe decir "Usuario o contraseña incorrectos".

3. La función `fValidaSesion()`

Es una función mucho más sencilla que actúa como un "puesto de control".

- **isset(\$_SESSION['misesion'])**: Solo hace una pregunta: "¿Tiene este usuario un carnet de identidad válido?".
 - Si la respuesta es **sí**, devuelve 1.
 - Si la respuesta es **no** (porque no ha hecho login o la sesión caducó), devuelve 0.

Resumen del flujo: `acceso.php` envía los datos -> `fValidaAcceso` comprueba todo -> Si todo está OK, guarda los datos en la "maleta" de la `$_SESSION` y te manda al área personal.

¿Ha quedado claro cómo funciona "el cerebro" del ejercicio? Cuando quieras, pásame el código de **areapersonal.php** o **correos.php**.

Explica mejor esta parte. Porque se usan esos códigos? Por qué los dos primeros usan 1?
header("Location: areapersonal.php");

```
exit();  
  
} else {  
  
    // return de contraseña incorrecta.  
  
    $_SESSION['error'] = 1;  
  
    return 0;  
  
}  
  
} else {  
  
    // return de usuario incorrecto  
  
    $_SESSION['error'] = 1;  
  
    return 0;  
  
}  
  
} else {  
  
    // return de faltan datos  
  
    $_SESSION['error'] = 0;  
  
    return 0;
```



Esta parte del código es la que gestiona la **comunicación** entre la función (que está oculta en el "cerebro" comunes .php) y la parte visual (acceso .php).

Se usan códigos numéricos (0 y 1) porque para un ordenador es más fácil y rápido comparar números que leer frases largas.

1. ¿Por qué los dos primeros usan el código 1?

En tu código, el número **1** significa: "**Las cajas están rellenas, pero los datos no son válidos**".

Fíjate que usas el 1 en dos situaciones distintas:

- **Caso A:** El usuario existe, pero la contraseña está mal.
- **Caso B:** El usuario ni siquiera existe en el array.

¿Por qué el mismo número para ambos? Por seguridad. En la web moderna, no es bueno decirle a un hacker: "Oye, el usuario sí existe, pero te has equivocado en la clave". Es mejor dar un mensaje genérico: "Usuario o contraseña incorrectos". Al usar el mismo código 1, la página acceso.php mostrará ese único mensaje general para ambos errores.

2. ¿Por qué el último usa el código 0?

El número 0 significa: "Ni siquiera he podido empezar a validar porque hay cajas vacías".

Si el usuario no escribe nada y pulsa enviar, no tiene sentido buscar "nada" en el array de contraseñas. Por eso, el código salta directamente al final y guarda un 0. Así, acceso.php sabe que debe mostrar un mensaje distinto: "Faltan datos de usuario o contraseña".

3. Diferencia entre `$_SESSION['error']` y `return`

Es muy importante no confundir estas dos órdenes, aunque vayan juntas:

- `$_SESSION['error'] = X;`: Esto es como **escribir una nota y guardarla en la maleta** (la sesión). Se queda ahí guardada para que cuando lleguemos a acceso.php, podamos leerla y saber qué mensaje mostrar en rojo.
- `return 0;`: Esto es la **orden de parada**. Cuando la función lee un return, deja de trabajar inmediatamente y le devuelve ese valor a quien la llamó. Es como decir: "He terminado, el resultado de mi trabajo es 0".

4. ¿Qué hacen header y exit?

Estas dos líneas solo se ejecutan si **TODO ha salido bien**:

- `header("Location: areapersonal.php");`: Es un "teletransporte". Le dice al navegador: "No te quedes aquí, vete inmediatamente a la página del área personal".
- `exit();`: Es el cerrojo. Le dice a PHP: "Deja de leer este archivo ahora mismo". Se pone por seguridad, para asegurar que si el "teletransporte" tarda un milisegundo en actuar, no se ejecute nada de código que haya debajo por error.

¿Entiendes ahora por qué clasificamos los errores con 0 y 1? Es simplemente para que acceso.php sepa qué frase elegir del "menú de errores". Cuando estés listo, dime y analizamos `areapersonal.php`.

entiendo. Y ahora, para hacer que se identifique los tres problemas de manera individual y poder imprimirlos aquí:

```
if (isset($_SESSION['error'])) {
```

```
    if ($_SESSION['error'] == 0) {
```

```

echo "<p style='color:red;'>Faltan datos de usuario o contraseña.</p>";
} else {
    echo "<p style='color:red;'>Usuario o contraseña incorrectos.</p>";
}
// Borramos el error para que no aparezca siempre. Dale a INTRO desde la URL,
// no desde el botón de "recargar" del navegador.
unset($_SESSION['error']);
}
?>

```



Para que puedas identificar los tres problemas de forma individual (campos vacíos, usuario inexistente o contraseña mal) y mostrar un mensaje específico para cada uno, tenemos que hacer un pequeño cambio en la función de **comunes.php** y luego en el visual de **acceso.php**.

Vamos a asignar un número único a cada situación.

1. Cambio en comunes.php (La lógica)

Modificamos la función para que use tres códigos diferentes en lugar de solo dos:

- **Código 0:** Faltan datos (vacío).
- **Código 1:** El usuario no existe.
- **Código 2:** La contraseña es incorrecta.

```

// comunes.php (dentro de fValidaAcceso)

if (isset($usuario) && $usuario != "" && isset($password) && $password != "") {
    if (array_key_exists($usuario, $datos)) {
        $hash_guardado = password_hash($datos[$usuario], PASSWORD_BCRYPT);

        if (password_verify($password, $hash_guardado)) {
            // ... LOGIN CORRECTO ... (lo mismo de antes)
        } else {
            // ERROR TIPO 2: Usuario bien, pero clave mal
            $_SESSION['error'] = 2;
        }
    }
    return 0;
}

```

```

        }
    } else {
        // ERROR TIPO 1: El usuario no está en el array
        $_SESSION['error'] = 1;
        return 0;
    }
} else {
    // ERROR TIPO 0: Al menos una caja está vacía
    $_SESSION['error'] = 0;
    return 0;
}

```

2. Cambio en acceso.php (El visual)

Ahora que la sesión puede guardar 0, 1 o 2, actualizamos los `if` para mostrar los tres mensajes distintos:

```

<?php
// acceso.php (en la parte donde imprimes el error)

if (isset($_SESSION['error'])) {

    if ($_SESSION['error'] == 0) {
        echo "<p style='color:red;'>¡Atención! Debes llenar todos los
campos.</p>";
    }
    elseif ($_SESSION['error'] == 1) {
        echo "<p style='color:red;'>El usuario introducido no existe en el
sistema.</p>";
    }
    elseif ($_SESSION['error'] == 2) {
        echo "<p style='color:red;'>La contraseña es incorrecta. Inténtalo
de nuevo.</p>";
    }

    // Borramos el error de la "maleta" (sesión)
    unset($_SESSION['error']);
}
?>

```

¿Por qué hacerlo así?

- 1. Claridad para el usuario:** Es mucho más amable decirle al usuario exactamente qué ha fallado.

2. Depuración: Para ti, como programador, es más fácil saber si el problema está en que el array no lee bien las llaves (Error 1) o si es que `password_verify` está fallando (Error 2).

Nota de seguridad: Recuerda que en entornos de trabajo reales, se prefiere el mensaje genérico (el que tenías antes) para no dar pistas a atacantes, pero **para este ejercicio escolar**, identificar los tres errores demuestra que dominas perfectamente los condicionales anidados (`if` / `elseif` / `else`).

¿Te parece bien este cambio o prefieres que los errores 1 y 2 sigan agrupados? Si estás de acuerdo, podemos pasar a explicar `correos.php`.

y es necesario usar el "return 0;"?



Es una buena pregunta. En este caso específico, el `return 0;` tiene una función técnica, aunque no es la que hace que el mensaje aparezca.

Aquí te explico por qué se pone y qué pasaría si lo quitas:

1. ¿Para qué sirve realmente?

El `return 0;` le dice a la función: "**Detente aquí y devuelve un resultado falso (fallido)**".

Cuando tú en `acceso.php` llamas a la función así: `fValidaAcceso($usuario, $password, $cookie_usuario);`

Realmente no estás guardando lo que la función devuelve (el 0 o el 1). Sin embargo, es una **buenas práctica de programación** por dos razones:

- **Corta la ejecución:** Al leer `return`, PHP deja de leer el resto de la función. Si no estuviera, PHP podría intentar seguir ejecutando líneas de código que no corresponden a un error.
- **Aclara el código:** Indica visualmente a cualquier programador que en ese punto la validación ha fallado.

2. ¿Es estrictamente necesario para que funcione tu ejercicio?

Técnicamente, no. Lo que hace que el mensaje de error se vea en la pantalla es la línea:

```
$_SESSION['error'] = X;
```

Si quitas el `return 0;`, el error se seguiría guardando en la sesión y `acceso.php` lo seguiría mostrando.

3. El riesgo de quitarlo

Si quitas los `return` y tu código es muy largo, podrías tener problemas de lógica. Por ejemplo:

```

if (clave_mal) {
    $_SESSION['error'] = 2;
    // Si no hay return aquí...
}

// ... PHP sigue leyendo y podría acabar ejecutando el código de login
correcto por error
header("Location: areapersonal.php");

```

Conclusión para tu examen/ejercicio:

Déjalos puestos. El profesor valorará que la función sea "formal", es decir, que devuelva un valor (1 si todo fue bien, 0 si algo falló). Es la forma correcta de escribir funciones en cualquier lenguaje de programación.

Análisis de "areapersonal.php"

Siguiendo con el orden, analicemos este archivo:

```

<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Área Personal</title>
</head>
<body>

<center>
    <fieldset style="width: 350px; padding: 20px;">
        <legend><strong>Panel de Usuario</strong></legend>

        <?php
        session_start();
        require_once 'comunes.php';

        // 1. EL GUARDAESPALDAS
        if (fValidaSesion() == 1) {
            // Si el carnet es válido, mostramos el contenido
            echo "<h1>Área personal de: " . $_SESSION['usuario'] . "</h1>";
            echo "<p><a href='correos.php'>Ir a mis Correos</a></p>";
            echo "<p><a href='salir.php'>Cerrar Sesión</a></p>";
        } else {
            // 2. LA EXPULSIÓN
            header("Location: acceso.php");
        }
    </fieldset>
</center>

```

```

        exit();
    }
?>

    </fieldset>
</center>

</body>
</html>

```

Explicación de los elementos:

- **fValidaSesion() == 1**: Esta es la parte más importante. Llama a la función de **comunes.php** para ver si el usuario tiene permiso para estar aquí. Es lo que impide que alguien vea tu área personal solo conociendo la URL.
- **\$_SESSION['usuario']**: Recuperamos el nombre que guardamos en la página de acceso para que el usuario vea que la web lo reconoce.
- **header("Location: acceso.php")**: Si la sesión no es válida (por ejemplo, porque ha caducado o porque el usuario intenta entrar "por la cara"), esta línea lo manda de vuelta al inicio.

¿Entendido el papel de "seguridad" de esta página? Si quieras, pasamos a **correos.php**, que es casi idéntica.

ahora pasamos a "areapersonal.php": <!DOCTYPE html>

```

<html lang="es">

<head>

    <meta charset="UTF-8">

    <title>Área Personal</title>

</head>

<body>

<fieldset style="width: 300px; padding: 20px;">

    <div class="container">

        <?php

        session_start();

        require_once 'comunes.php';

```

```

if (fValidaSesion() == 1) {

    echo "<h1>El área personal de " . $_SESSION['usuario'] . "</h1>";

    echo "<a href='correos.php'>Correos</a>";

    echo "<br><a href='salir.php' class='logout-link'>Salir</a>";

} else {

    header("Location: acceso.php");

    exit();

}

?>

</div>

</fieldset>

</body>

</html>

```



Vamos con el análisis de **areapersonal.php**. Este archivo es la "zona privada". Su función principal no es procesar datos, sino **proteger** el contenido y mostrar opciones al usuario que ya se ha identificado.

Aquí tienes la explicación elemento por elemento:

1. El marco visual (HTML)

- **<fieldset style="...>**: Como en la página anterior, esto dibuja el **cuadrado** que rodea la información. Al mantener el mismo estilo (width: 300px), consigues que toda tu aplicación tenga una apariencia coherente y ordenada.
- **<div class="container">**: Es una caja invisible que sirve para agrupar los elementos de PHP y poder aplicarles estilos en el futuro si quisieras.

2. La lógica de control (PHP)

Este bloque es el "vigilante" de la página:

- **session_start();**: Abre la sesión. Es fundamental porque aquí es donde vamos a mirar si el usuario tiene el "carnet" (la sesión activa) que le dimos en **acceso.php**.
- **require_once 'comunes.php';**: Importa las funciones. Sin esto, la página no sabría qué es **fValidaSesion()**.
- **if (fValidaSesion() == 1)**: Aquí llamamos a la función de seguridad.
 - Si devuelve **1**: El usuario está autorizado.
 - Si devuelve **0**: Significa que alguien ha intentado entrar escribiendo la dirección **areapersonal.php** directamente en el navegador sin poner su contraseña.

3. El contenido para el usuario identificado

Si la validación es correcta (el **if** es verdadero), se ejecuta lo siguiente:

- **\$_SESSION['usuario']**: PHP va a la memoria de la sesión y saca el nombre que guardamos antes. Así la página dice: "*El área personal de cristobal*".
- **Los Enlaces (<a>):**
 - El primero te lleva a **correos.php** (otra página protegida).
 - El segundo te lleva a **salir.php**, que es el archivo encargado de "romper" la sesión y borrar los datos de la memoria.

4. La expulsión (El **else**)

- **header("Location: acceso.php")**: Si **fValidaSesion** dio 0, esta línea redirige automáticamente al usuario a la página de inicio. Es la forma de decir: "*No tienes permiso, vuelve al principio e identifícate*".
- **exit()**: Detiene el código por completo para que no se envíe nada más al navegador.

Resumen: Esta página es un filtro. Si tienes sesión, te saluda y te da opciones; si no la tienes, te echa.

¿Está todo claro en este archivo? Cuando quieras, pásame **correos.php** (que verás que es casi igual) o **salir.php**.

correos.php: <!DOCTYPE html>

```
<html lang="es">
```

```
<head>
```

```
  <meta charset="UTF-8">
```

```
  <title>Correos</title>
```

```

</head>

<body>

<fieldset style="width: 300px; padding: 20px;">

<?php

session_start();

require_once 'comunes.php';

if (fValidaSesion() == 1) {

    echo "<h1>Bandeja de Correos</h1>";

    echo "<p>Los correos de " . $_SESSION['usuario'] . ".</p>";

    echo "<a href='areapersonal.php'>Volver a mi área personal</a>";

} else {

    header("Location: acceso.php");

    exit();

}

?>

</fieldset>

</body>

</html>

```

 Analicemos ahora **correos.php**. Como habrás notado, su estructura es casi idéntica a la de `areapersonal.php`. Esto es porque ambas son "páginas protegidas" que forman parte de la zona privada de tu web.

Aquí tienes la explicación detallada:

1. El Marco de Seguridad (fieldset)

- `<fieldset style="width: 300px; padding: 20px;">`: Mantenemos el mismo diseño visual. Esto es importante para que el usuario sienta que no ha salido de la aplicación y que todo está bajo el mismo "sistema".

2. El Control de Acceso (PHP)

- `session_start();`: Volvemos a abrir la sesión. Recuerda que la sesión es como una conversación: cada vez que cambias de página, hay que decir `session_start()` para continuar la charla donde la dejamos.
- `fValidaSesion() == 1`: Esta es la comprobación clave.
 - Si alguien intenta entrar a `correo.php` poniendo la URL directamente en el navegador, esta función dirá "No tienes carnet" (devolverá 0) y el código saltará al `else`.
 - `header("Location: acceso.php");`: La expulsión inmediata. Nadie que no esté logueado puede ver esta página.

3. Personalización del Contenido

- `$_SESSION['usuario']`: Al igual que en el área personal, usamos el nombre guardado en la sesión. Así, aunque sea una página genérica de "Correos", el usuario siente que es su bandeja de entrada personal.
- `echo "<p>Los correos de " . $_SESSION['usuario'] . ".</p>"`: Aquí simplemente estamos imprimiendo texto, pero el hecho de usar la variable de sesión demuestra que el servidor sabe quién eres en todo momento.

4. Navegación Interna

- `Volver...`: Es fundamental dar siempre una salida al usuario hacia atrás. En las aplicaciones web, no debemos obligar al usuario a usar las flechas del navegador; debemos darle botones de navegación propios.

Diferencia clave con las anteriores: A diferencia de `acceso.php` (que crea datos) o `salir.php` (que los borra), esta página es meramente de **lectura**. Solo lee la sesión para confirmar que puedes estar ahí y lee el nombre para mostrarlo.

¿Todo claro hasta aquí? Ya solo nos queda el último archivo, que es el que cierra el ciclo: **salir.php**. ¿Me lo pasas para terminar el análisis?

`salir.php: <!DOCTYPE html>`

```
<html lang="es">
```

```
<head>
```

```
<meta charset="UTF-8">

<title>Sesión Cerrada</title>

</head>

<body>

<fieldset style="width: 300px; padding: 20px;">

<?php

session_start();

require_once 'comunes.php';

if (fValidaSesion() == 1) {

$usuario = $_SESSION['usuario'];

$fecha_inicio_sesion = date('d/m/Y H:i', $_SESSION['inicio']);

$fecha_fin_sesion = date('d/m/Y H:i');

echo "<h1>Sesión cerrada</h1>";

echo "<p>Se ha cerrado la sesión para el usuario: <strong>$usuario</strong></p>";

echo "<p>Fecha inicio: $fecha_inicio_sesion</p>";

echo "<p>Fecha fin: $fecha_fin_sesion</p>";

echo "<a href='acceso.php'>Acceso principal</a>";

// Destruir sesión completamente

$_SESSION = array();

session_destroy();

} else {

header("Location: acceso.php");
}
```

```
exit();
```

```
}
```

```
?>
```

```
</fieldset>
```

```
</body>
```

```
</html>
```



Finalmente, analizamos **salir.php**. Este archivo es el encargado de hacer la "limpieza" final. Su objetivo es mostrar un resumen de la actividad del usuario y asegurarse de que la sesión se cierre de forma segura para que nadie más pueda entrar.

Aquí tienes la explicación detallada:

1. El Control de Seguridad (Nuevamente)

- **if (fValidaSesion() == 1):** Al igual que en las páginas anteriores, primero comprobamos si hay una sesión activa. Si alguien intenta entrar a `salir.php` sin haberse logueado, el código lo expulsa a `acceso.php`. Esto evita que se generen errores al intentar leer datos que no existen.

2. Preparación del Resumen

Antes de borrar los datos, los leemos para mostrarlos por última vez:

- **\$usuario = \$_SESSION['usuario'];**: Recuperamos el nombre para despedirnos.
- **date('d/m/Y H:i', \$_SESSION['inicio'])**: Aquí ocurre algo interesante. Transformamos el número de segundos que guardamos al principio (el *timestamp*) en un formato que los humanos entendemos (día/mes/año hora:minuto).
- **\$fecha_fin_sesion = date('d/m/Y H:i');**: Creamos una fecha nueva con el momento exacto actual (el momento en que el usuario pulsa "Salir").

3. El Proceso de Destrucción (Cierre de Sesión)

Esta es la parte técnica más importante para que el sistema sea seguro:

- **\$_SESSION = array();**: Esta línea vacía la "maleta". Borra todas las variables que habíamos guardado (usuario, inicio, etc.) dentro del servidor.
- **session_destroy();**: Esta orden rompe la conexión entre el navegador y el servidor. Es como tirar el carnet de identidad a la basura; a partir de este momento, si el usuario intenta volver atrás al área personal, el sistema le denegará el acceso.

4. El Enlace de Retorno

- Acceso principal: Es el único camino que dejamos abierto. Una vez cerrada la sesión, el usuario solo puede volver a empezar desde el principio.
-

Resumen del ciclo completo:

1. **comunes.php**: Define las reglas y guarda las contraseñas.
2. **acceso.php**: Pide los datos y crea la sesión y la cookie.
3. **areapersonal.php** y **correos.php**: Comprueban que la sesión exista y muestran contenido.
4. **salir.php**: Muestra el resumen de tiempos y destruye la sesión.

¡Con esto hemos analizado todo tu código! Todo cumple con los requisitos: usas sesiones para la seguridad, cookies para recordar al usuario, password_verify para la clave y fieldset para el diseño. ¿Tienes alguna duda más sobre algún archivo o alguna parte específica del código?

C