

# DOCUMENTACIÓN APP HOTEL

## Herramientas Utilizadas:

- IDE: NetBeans V19
- Servidor web utilizado: Apache/2.4.56
- Lenguajes utilizados: HTML, PHP, CSS.
- Base de Datos: phpmyadmin.
- Frameworks: Bootstrap v5.3.
- Github : <https://github.com/joelortiiz/App-Funcional-Hoteles->

## Requisitos de Aplicación:

Aplicación web de gestión de reservas de hoteles

En este ejercicio, deberás desarrollar una aplicación web de gestión de reservas de hoteles utilizando PHP, Objetos y el patrón de arquitectura MVC. La aplicación deberá contar con las siguientes funcionalidades:

### 1. Autenticación de usuarios:

La aplicación deberá tener un sistema de autenticación de usuarios que permita acceder a la aplicación únicamente a usuarios registrados.

Deberás utilizar sesiones para controlar el estado de autenticación del usuario.

Deberás utilizar al menos una cookie con información de la última visita del usuario

### 2. Gestión de hoteles y habitaciones:

La aplicación deberá permitir a los usuarios ver la lista completa de hoteles disponibles y sus habitaciones.

Deberás utilizar una vista para mostrar la lista de hoteles y otra para mostrar los detalles de un hotel concreto, incluyendo la lista de habitaciones disponibles.

### 3. Gestión de reservas:

Los usuarios deberán poder realizar reservas de habitaciones a través de la aplicación.

Deberás utilizar una tabla adicional para almacenar las reservas realizadas por los usuarios.

Deberás utilizar una vista para mostrar la lista de reservas realizadas por un usuario concreto y otra para mostrar los detalles de una reserva concreta.

## Diseño de Aplicación:

Para el diseño de la vista de esta aplicación me he ayudado de una plantilla o “template” de Bootstrap v5.3

<https://bootstrapped.com/sailor-free-bootstrap-theme/>

El diseño u organización del proyecto se divide en varias partes:

Nombre	Fecha de modificación	Tipo	Tamaño
.git	29/01/2024 18:05	Carpeta de archivos	
assets	25/01/2024 17:22	Carpeta de archivos	
config	19/01/2024 16:05	Carpeta de archivos	
controllers	24/01/2024 17:43	Carpeta de archivos	
db	11/01/2024 17:27	Carpeta de archivos	
lib	18/01/2024 19:12	Carpeta de archivos	
nbproject	11/01/2024 16:35	Carpeta de archivos	
views	27/01/2024 18:12	Carpeta de archivos	
frontcontroller.php	25/01/2024 12:06	Archivo de origen ...	3 KB
index.php	25/01/2024 16:33	Archivo de origen ...	1 KB

- Assets: contiene archivos adicionales para nuestro proyecto como las imágenes, estilos, javascript, etc...
- Config: Tiene las credenciales de la conexión a la base de datos. Al ahora de crear una conexión llamamos a dicho archivo. Esto sirve para que en un caso de error o cambios de credenciales en la conexión, solo sea necesario modificar este archivo.
- Controllers: Los controllers son componentes encargados de manejar las solicitudes del usuario, procesar la lógica de la aplicación y coordinar la interacción entre el modelo y la vista.
- Db: Contiene la clase de la base de datos
- Lib: librerías adicionales necesarias para el funcionamiento del proyecto, dentro está:

Nombre	Fecha de modificación	Tipo	Tamaño
class	15/01/2024 17:52	Carpeta de archivos	
functions	18/01/2024 19:12	Carpeta de archivos	
models	25/01/2024 10:55	Carpeta de archivos	

El "Model" (Modelo) se refiere a la parte de la aplicación que se ocupa de la representación y manipulación de los datos.

- View: Se encarga de la presentación y visualización de los datos al usuario.

### Implementación y Pruebas de la aplicación:

La aplicación al estar creada con el modelo frontcontroller, únicamente habrá que incluir dicho fichero en el **index.php**

```
<?php
// Incluimos el archivo frontcontroller.php
include './frontcontroller.php';
?>
```

En **frontcontroller.php** incluimos los diferentes controladores a utilizar.

```
<?php

include './controllers/usuariosController.php';
include './controllers/HotelesController.php';
include './controllers/ReservasController.php';
include './controllers/HabitacionesController.php';
```

Definimos nuestra acción por defecto que será lo que se ejecutará al iniciar la aplicación

```
// Define la acción por defecto
define('ACCION_DEFECTO', 'mostrarUsuarios');

// Define el controlador por defecto
define('CONTROLADOR_DEFECTO', 'usuarios');
```

La acción por defecto llama, como podemos ver, a la acción mostrar usuarios del controlador de Usuarios, donde se llama a la función de la vista para que el usuario pueda mandarnos la información necesaria para sacar los datos correspondientes de la Base de datos.

Para llamar a la función de la vista, creamos un nuevo objeto de la clase de la misma.

Mismo procedimiento para el objeto modal.

```
<?php

include './lib/models/UsuarioModel.php';
include './views/usuariosView.php';

class UsuariosController {

    private $model;
    private $view;

    public function __construct() {
        $this->model = new UsuarioModel();
        $this->view = new usuariosView();
    }

    public function mostrarUsuarios() {

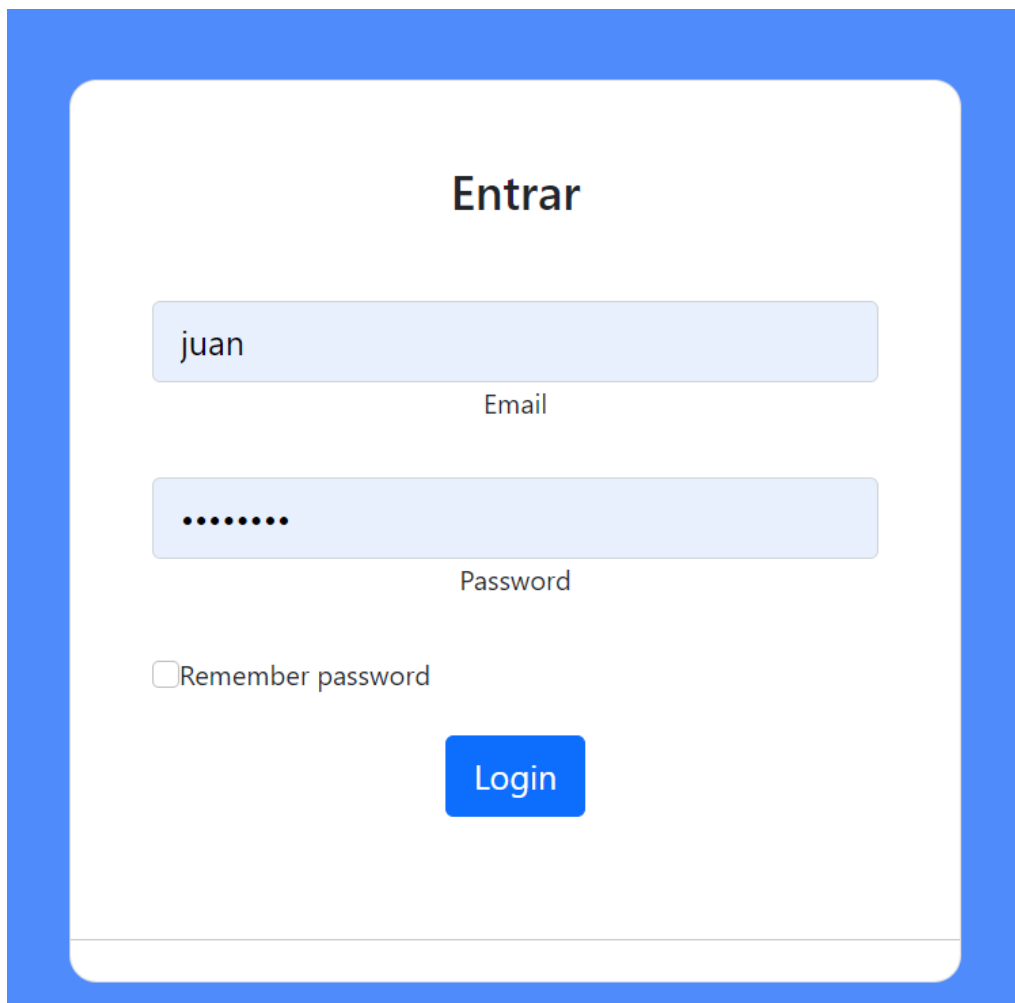
        $this->view->mostrarFormulario();
    }
}
```

```
// Obtiene una instancia del modelo y de la
class usuariosView {

    // Muestra la lista de tareas
    public function mostrarFormulario() {

        ?>
<!DOCTYPE html>
<html>
    <head>
        <title>Mi aplicación de hoteles</ti
```

-Usuarios View:



Entrar

juan

Email

.....

Password

☐ Remember password

Login

El usuario debe introducir su nombre y contraseña. Si esta coincide con alguno de los usuarios y contraseña de la BBDD, se mostrarán los Hoteles.

## -Usuarios View:

Para llegar a hotelesView, este ha sido el proceso que ha seguido la aplicación:

```
public function comprobarLogin() {  
  
    if (isset($_POST['usuario']) && isset($_POST['password'])) {  
        //Guardamos los datos recibidos por el formulario.  
        $userPost = $_POST['usuario'];  
        $passPost = $_POST['password'];  
        //comprobamos que el usuario y contraseña sean correctos  
        $user = $this->model->comprobarUsuarioDB($userPost, $passPost);  
        //Si es correcto:  
        if ($user != false) {  
            $_SESSION['id'] = $user->getId();  
            //Llamamos a la función que se encarga de redirigir al usuario a ver los hoteles.  
            $this->correctoLogin($user);  
            //Si el usuario no es correcto  
        } else {  
            //función que hace que evita al cliente intruso o avisa que los datos no son correctos.  
            $this->errorLogin();  
        }  
    }  
}
```

Comprobamos que se han recibido los datos a través del formulario y luego mandamos dichos datos a la función del modelo de usuarios que nos devolverá un objeto del usuario o false en función a que la consulta a la base de datos sea correcta o no.

```
if ($stmt->rowCount()>0) {  
    foreach ($stmt as $user) {  
        $userObject = new Usuario($user['id'], $user['nombre'], $user['password']);  
    }  
  
    // $userObject = $stmt->fetch();  
  
    $_SESSION['id'] = $userObject->getId();  
    //He tenido problemas para almacenar la id del usuario en la sesión  
    setcookie("id", $userObject->getId(), time() + 20 * 2400);  
  
    $this->bd->cerrarBD();  
  
    return $userObject;  
}
```

Al ser correcto, se llama a la siguiente función

```
function correctoLogin($user) {  
    $this->iniciarSesionUsuario($user);  
    header('Location:' . $_SERVER['PHP_SELF'] . '?controller=Hoteles&action=mostrarHoteles');  
}
```

Que redirige y llama al controlador de Hoteles

Hoteles controller creamos objetos con la clase del modelo y la vista.

```
public function __construct() {  
    $this->model = new HotelModel();  
    $this->view = new hotelesView();  
}
```

Si la sesión está creada, continua el código, si no es redirigido al inicio

```
public function mostrarHoteles() {
    session_start();
    if(!$_SESSION['user']) {
        header('Location: ./index.php');
    }
}
```

Después se declara una variable que tendrá como valor la respuesta de la consulta a la base de datos donde requerimos de todos los hoteles. Luego mandamos dicha variable con todos los hoteles a la vista para iterar los datos

```
$hoteles = $this->model->getHoteles();
$this->view->mostrarHoteles($hoteles);
```

Consulta preparada y hacemos fetch class con la clase Hotel \*

```
$sql = "SELECT * from hoteles";
$hoteles = $this->pdo->prepare($sql);
$hoteles->execute();
$hoteles->setFetchMode(PDO::FETCH_CLASS, 'Hotel');

if ($hoteles) {
    $hotelesObject = $hoteles->fetchAll();
    // print_r($hotelesObject);
    // echo $hotelesObject[0]->getNombre();
    $this->bd->cerrarBD();
    return $hotelesObject;
}
```

Importante cerrar la conexión a la BD.

En la vista recibimos el array de objetos de la clase hotel y mostramos sus datos

```
public function mostrarHoteles($hoteles) {
    ?>
    <!DOCTYPE html>
```

Iteramos

```
<?php
foreach ($hoteles as $hotel) {
```

Al ser objetos de una clase llamamos a su getter para recibir el dato:

```
<?php echo $hotel->getNombre();
```

En el caso de la imagen, esta línea de código HTML se utiliza para mostrar una imagen directamente en una página web, sin necesidad de cargarla desde un archivo externo. La imagen está utilizando la codificación base64.

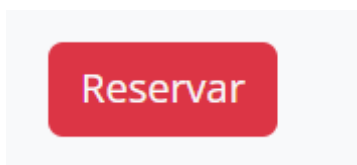
```
getId() ?>">
<input type="hidden" name="nombre_hotel" value="<?php echo $hotel->getNombre() ?>">
<button class="btn bg-danger text-light" type="submit">Ver habitaciones</button>
```

En **habitaciones controller** seguimos un proceso muy similar al controlador de hoteles, con la diferencia de que llamamos a las funciones que devuelven los datos enviándoles parámetros para concretar la consulta

```
public function mostrarHabitacionesHotel() {
    session_start();
    if (!$SESSION['user']) {
        header('Location: ../index.php');
    }
    if (isset($_POST['id_hotel']) && isset($_POST['nombre_hotel'])) {
        //echo $_POST["nombre_hotel"];
        $id = $_POST['id_hotel'];
        $nombre = $_POST['nombre_hotel'];
        $habitaciones = $this->model->getHabitacionesDeHotel($id, $nombre);
        $this->view->mostrarHabitaciones($habitaciones);
    }
}
```

Para la vista igual. Recorremos el array de objetos de la clase Habitación para imprimir los datos de cada una.

Incluimos un botón que manda el formulario que nos lleva a reservar dicha habitación



```
<form class="form" action="<?=$_SERVER['PHP_SELF'] . '?controller=Reservas&action=mostrarReservas&idHotel=' . $habitacion->
<input type="hidden" name="id_hotel" value="<?php echo $habitacion->getId() ?>">
<input type="hidden" name="nombre_hotel" value="<?php echo $habitacion->getNum_habitacion() ?>">
<button class="btn bg-danger text-light" type="submit">Reservar</button>
```

**Reservas Controller:**

En el controlador de reserva tenemos las siguientes funciones:

Mostrar Reservas empieza comprobando que la sesión esté iniciada para que no puedan entrar usuarios que no hayan iniciado sesión.

Recibimos los valores necesarios de la vista de habitaciones para poder realizar una reserva de la habitación concreta.

También guardamos con una función del modelo, las reservas que ya tiene realizadas el usuario el cual ha iniciado sesión.

```
public function mostrarReservas() {  
    $id_hotel = $_GET['idHotel'];  
    $id_habitacion = $_GET['idHabitacion'];  
    session_start();  
    $idUser = $_COOKIE['id'];  
    if (!$_SESSION['user']) {  
        header('Location: ./index.php');  
    }  
  
    $reservas = $this->model->getReserva($id_hotel, $id_habitacion);  
    $reservasUser = $this->model->getReservasUser($idUser);  
    $this->view->mostrarReservas($reservas, $reservasUser, null);  
}
```

#### Get Reserva:

Reciben el id de hotel y el id de habitación para sacar los datos de la habitación la cual el usuario quiere reservar

```
public function getReserva($id_hotel, $id_habitacion) {  
    try {  
        // Consulta SQL para seleccionar todas las reservas de una habitación en un hotel  
        $sql = "SELECT * FROM habitaciones WHERE id_hotel = ? AND id = ?";  
  
        // Preparar y ejecutar la consulta con los parámetros proporcionados  
        $reservas = $this->pdo->prepare($sql);  
        $reservas->execute(array($id_hotel, $id_habitacion));  
  
        // Obtener todas las reservas como un array asociativo  
    }  
}
```

#### Get reservas user:

Mandando el id de usuario que, en este caso lo guardo en una cookie ya que al guardarla en la sesión como pedía el ejercicio, aparecía un error que no conseguía solucionar.

Hacemos la consulta y la ejecutamos con la variable recibida en el parámetro el cual es el id del usuario que ha iniciado sesión.



```
public function getReservasUser($id) {
    try {
        // Consulta SQL para seleccionar todas las reservas de un usuario
        $sql = "SELECT * FROM reservas WHERE id_usuario = ?";

        // Preparar y ejecutar la consulta con el parámetro proporcionado
        $reservasUser = $this->pdo->prepare($sql);
        $reservasUser->execute(array($id));

        // Obtener todas las reservas como un array asociativo
        $todasReservas = $reservasUser->fetchAll();

        // Retornar el array de reservas
        return $todasReservas;
    }
}
```

Retornamos el array que contiene todas las reservas que tiene el usuario.

Llamamos a la vista mandando los resultados de las consultas y el último parámetro como null ya que no queremos mandar ningún estado de la aplicación.

```
$this->view->mostrarReservas($reservas, $reservasUser, null);
```

Mostramos la vista

```
class reservasView {
    function mostrarReservas($reservas, $reservasUser, $resultadoReserva) {
```

En caso de recibir variables que muestran el estado final de una inserción de reserva

```
if ($resultadoReserva == 'fallida') {
    echo '<div class="alert alert-danger alert-dismissible" >';
    echo 'Reserva no disponible o errónea. Cambia la fecha';
    echo '<button type="button" class="btn-close" data-bs-dismiss="alert" >';
    echo '</div>';
}

if (isset($_GET['success'])) {
    echo '<div class="alert alert-success alert-dismissible" >';
    echo 'La Reserva ha sido realizada con éxito.';
    echo '<button type="button" class="btn-close" data-bs-dismiss="alert" >';
}
```

Mostramos los datos de la habitación que quiere reservar el cliente

```
foreach ($reservas as $value) {
    echo '<p>Tipo de habitación: ' . $value['tipo'] . '</p>';
    echo '<p>Precio Final: ' . $value['precio'] . '€</p>';
    echo '<p>Nº Habitaciones: ' . $value['num_habitacion'] . '</p>';
    echo '<p>Desc: ' . $value['descripcion'] . '</p>';
    ?>
```

Equivale a esto

## ZONA DE RESERVA

Detalles de la reserva

Tipo de habitación: individual

Precio Final: 50€

Nº Habitaciones: 1

Desc: Habitación individual con vistas a la ciudad

Si el usuario quiere hacer una reserva, debe seleccionar fecha de entrada y de salida:


Este formulario manda las fechas elegidas por el usuario y por “detrás” manda los id de usuario, hotel y habitación.

```
<h3>Confirma tu reserva</h3>
<form action="index.php?controller=Reservas&action=comprobarReserva" class="mt-4" method="post">
  <?php ?>
  <input type="hidden" name="id_usuario" value="<?php echo $_COOKIE['id'] ?>" >
  <input type="hidden" name="id_hotel" value="<?php echo $value['id_hotel'] ?>" >
  <input type="hidden" name="id_habitacion" value="<?php echo $value['id'] ?>" >
  <div class="mb-3">
    <label for="fecha_entrada" class="form-label">Fecha de Entrada:</label>
    <input type="date" id="fecha_entrada" name="fecha_entrada" class="form-control" >
  </div>
  <div class="mb-3">
    <label for="fecha_salida" class="form-label">Fecha de Salida:</label>
    <input type="date" id="fecha_salida" name="fecha_salida" class="form-control" >
  </div>
  <button type="submit" class="btn btn-danger">Enviar Reserva</button>
</form>
```


En la vista, se muestra así:

### Confirma tu reserva

Fecha de Entrada:

dd/mm/aaaa 

Fecha de Salida:

dd/mm/aaaa 

Enviar Reserva

Más abajo podemos ver que se muestran las reservas que tenga cada clientes almacenadas o registradas y junto a un botón desplegable que muestra los detalles más concretos de dicha reserva

```
foreach ($reservasUser as $reserva) {  
    // Cabecera de la tabla
```

```
<div class="dropdown-menu" aria-labelledby="dropdownMenuButton">  
    <a class="dropdown-item" href="#"><?php echo "ID Reserva: " . $reserva['id']; ?></a>  
    <a class="dropdown-item" href="#"><?php echo "ID Usuario: " . $reserva['id_usuario']; ?></a>  
    <a class="dropdown-item" href="#"><?php echo "ID Habitación: " . $reserva['id_habitacion']; ?></a>  
</div>
```

En la vista se muestra así:

Tus Reservas, Joel

Detalles	Fecha Entrada	Fecha Salida
Mostrar ▾	2024-01-10	2024-01-11
ID Reserva: 6		
ID Usuario: 3		
ID Habitación: 1		

Al pulsar en enviar reserva, se realiza la siguiente acción:

```
<form action="index.php?controller=Reservas&action=comprobarReserva" class="mt-4" method="post">
```

Llama al controlador de Reservas y a la acción comprobar reserva que consta en recibir todos los datos de la habitación a reservar y primero comprueba con la función **comprobarReservaCorrecta** comprueba que la solicitud de la reserva no exista ya.

```
try {  
    // Consulta SQL para contar las reservas que se superponen con la nueva reserva  
    $sql = "SELECT COUNT(*) FROM reservas WHERE id_hotel = "  
        . ":id_hotel AND id_habitacion = :id_habitacion AND NOT (fecha_entrada >= :fecha_salida OR fecha_salida <= :fecha_entrada)";  
  
    // Preparar y ejecutar la consulta con los parámetros proporcionados  
    $reservasExistentes = $this->pdo->prepare($sql);  
    $reservasExistentes->execute(array('id_hotel' => $id_hotel, 'id_habitacion' => $id_habitacion, 'fecha_entrada' => $fecha_entrada, 'fecha_salida' => $fecha_salida));  
  
    // Obtener el resultado de la consulta (número de reservas que se superponen)  
    $reservasExistentes = $reservasExistentes->fetchColumn();
```

Después revisa que la fecha de entrada introducida no sea menor a la de salida y si cumple todas las condiciones, se devuelve true

```
// Si no hay superposición, verificar la validez de las fechas  
if ($this->comprobarFechaReserva($fecha_entrada, $fecha_salida)) {  
    return true;  
} else {  
    return false;
```

Si es true, se ejecuta la inserción de la reserva con los datos recibidos. Si es false se devuelve a la vista con el mensaje de inserción fallida

```
if ($reservaCorrecta == true) {  
    $this->model->insertarReserva($id_habitacion, $id_hotel, $fecha_entrada, $fecha_salida);  
} else {  
    $resultadoReserva = "fallida";  
    $reservas = $this->model->getReserva($id_hotel, $id_habitacion);  
    $reservasUser = $this->model->getReservasUser($id_usuario);  
  
    $this->view->mostrarReservas($reservas, $reservasUser, $resultadoReserva);
```

Para **insertar la reserva** primero seleccionamos el id máximo de las reservas ya existentes.

```
$sqlId = "SELECT MAX(id) FROM reservas";
$idbd = $this->pdo->prepare($sqlId);
$idbd->execute();
```

Después creamos la consulta SQL y la ejecutamos mandando un array de las variables con los datos a introducir en la inserción

```
// Consulta para insertar la nueva reserva
$sql = "INSERT INTO reservas (id, id_usuario, id_hotel, id_habitacion, fecha_entrada, fecha_salida) "
      . "VALUES(:id, :id_usuario, :id_hotel, :id_habitacion,"
      . " :fecha_entrada, :fecha_salida)";

// Preparar y ejecutar la consulta con los parámetros proporcionados
$insertReserva = $this->pdo->prepare($sql);
$insertReserva->execute(array('id' => $idnuevo, 'id_usuario' => $_COOKIE['id'], 'id_hotel' => $id_hotel, 'id_habitacion' => $id_habitacion, 'fecha_entrada' => $fecha_entrada, 'fecha_salida' => $fecha_salida));
```

Redirigimos con el mensaje de que la inserción se ha realizado satisfactoriamente si es que no salta una excepción del try.

```
header('Location: index.php?controller=Reservas&action=mostrarReservas&idHotel=' . $id_hotel . '&idHabitacion=' . $id_habitacion);
throw new Exception("Reserva incorrecta");
```

En la vista, el mensaje de inserción correcta se muestra así

[Home](#) [About](#) [Services](#) [Portfolio](#) [Pri](#)

# ZONA DE RESERVA

La Reserva ha sido realizada con éxito.

## Detalles de la reserva

Título de la reserva: Reserva de hotel

Y la podremos ver en la misma página ya añadida

Tus Reservas, Joel

Detalles	Fecha Entrada	Fecha Salida
Mostrar	2024-01-10	2024-01-11
Mostrar	2024-01-01	2024-01-02

### Cerrar Sesión:

En todas las vistas (a excepción del Login) tendremos arriba a la derecha un botón para cerrar la sesión

Cerrar Sesión

En el código se traduce a esto:

Un enlace que llama al controlador de usuarios y a su acción de cerrar las sesiones además de mandar una variable que será recibida por get.

```
<a href="./index.php?controller=usuarios&action=cerrarSesionUsuario&error" class="getstarted">Cerrar Sesión</a></li>
```

La acción llamada consiste en convertir la variable global de sesión en un array para así vaciar todas las sesiones que tenga dentro incluidas. Al mismo tiempo se redirige llamando al controlador del usuario con la acción que muestra el formulario

```
function cerrarSesionUsuario() {
    session_start();
    //Destruimos las sesiones

    $_SESSION = array();
    session_destroy();

    //Redirigimos al inicio con todas las sesiones destruidas
    header('Location: ./index.php?controller=usuarios&action=mostrarErrorForm&logout'
    );
}
```

```
if (isset($_GET["error"])) {
    echo "<p class='text-danger'>Datos incorrectos</p>";
}
if (isset($_GET["logout"])) {
    echo "<p class='text-success'>Sesión cerrada correctamente </p>";
}
?>
```

A nivel de vista se muestra así:

Entrar

Sesión cerrada correctamente

joel

Email

....

Password

☐ Remember password

Login

Última conexión:

Se nos requiere crear una cookie que guarde la última conexión del usuario.

Esta debemos crearla una vez se haya confirmado la conexión correcta y que el login sea satisfactorio. Nos iremos al **controlador de usuarios**, donde al ya haber recibido la llamada a la función del Login correcto, llamamos a una función que inicia la sesión global y crea una cookie con la fecha actual antes de redirigir al controlador que recoge los hoteles de la bbdd

```
function correctoLogin($user) {  
    $this->iniciarSesionUsuario($user);  
    header('Location:' . $_SERVER['PHP_SELF']  
}
```

Llamamos a la función que crea la sesión y luego llama a la función que crea las cookies.

```
function iniciarSesionUsuario($user) {  
    session_start();  
  
    $_SESSION['user'] = $user->getNombre();  
    $this->createSessionCookie($_SESSION['user']);  
}
```

Aquí vemos que si no está declarada la cookie, la crea, y si ya está creada, la elimina para volver a crearla con la nueva fecha de conexión.

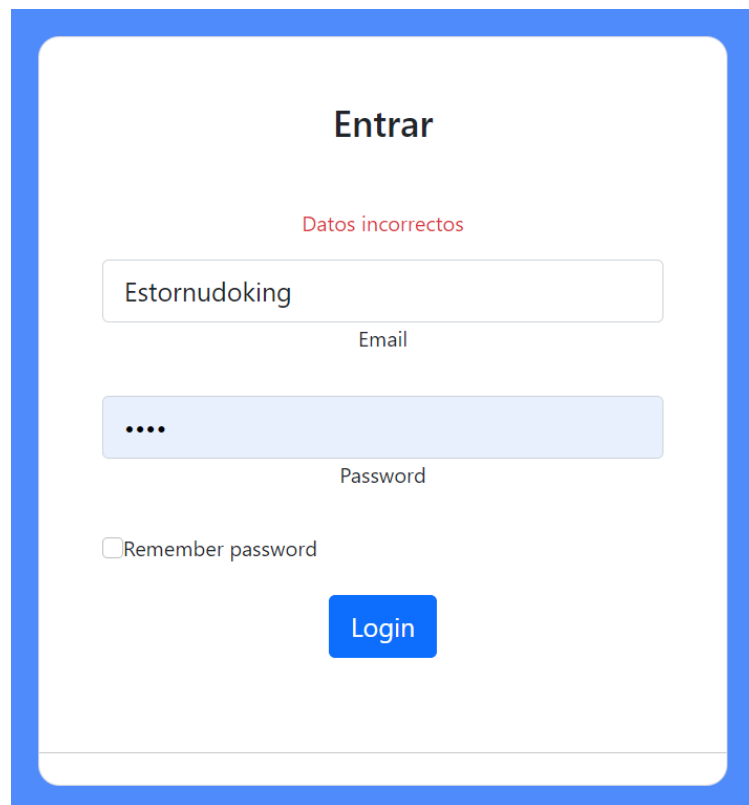
```
public function createSessionCookie($user) {  
    if (!isset($_COOKIE["conexion"])) {  
        $fecha_actual = date("d/m H:i");  
        setcookie("conexion", $fecha_actual, time() + 3600 * 24, "/");  
    } else {  
        setcookie("conexion", $fecha_actual, time() - 60, "/");  
        $fecha_actual = date("d/m H:i");  
        setcookie("conexion", $fecha_actual, time() + 3600 * 24, "/");  
    }  
    setcookie(hash('sha256', $user), 'sessionCookie', time() + 2400, '/');  
}
```

Creamos una segunda cookie para almacenar el usuario para evitar el problema circunstancial mencionado anteriormente.

### Parte de Pruebas:

#### PRUEBAS LOGIN:

- Entrar con un usuario no existente y contraseña SÍ existente en la BBD



Entrar

Datos incorrectos

Estornudoking

Email

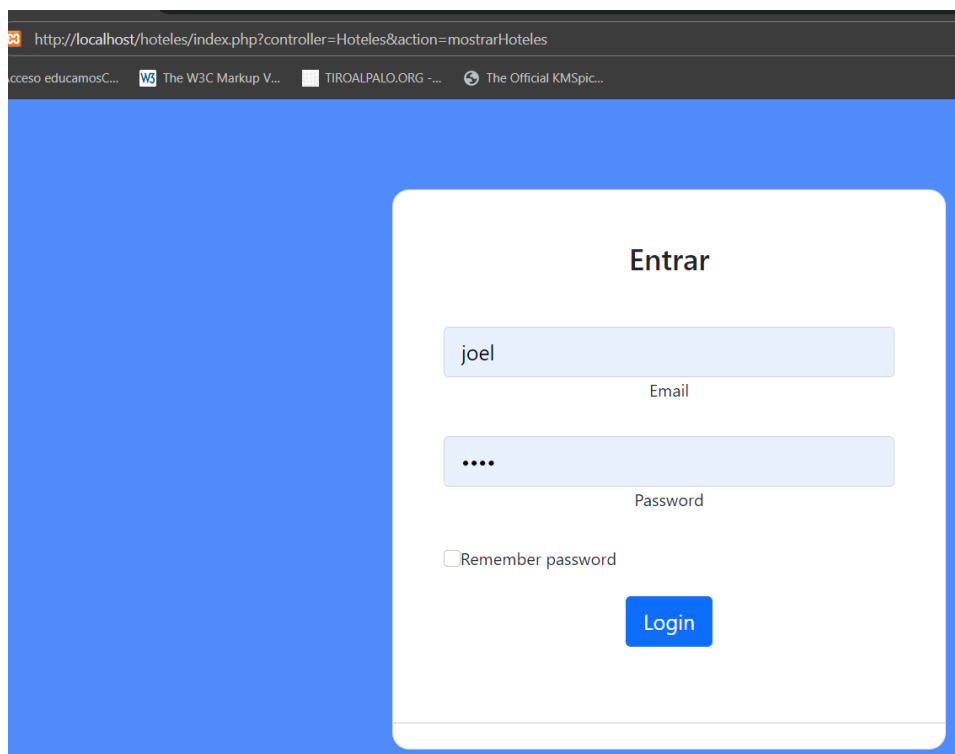
....

Password

☐ Remember password

Login

- Entrar directamente mediante el enlace sin haber iniciado sesión:



http://localhost/hoteles/index.php?controller=Hoteles&action=mostrarHoteles

acceso educamosC... W3 The W3C Markup V... TIROALPALO.ORG -... The Official KMSpic...

Entrar

joel

Email

....

Password

☐ Remember password

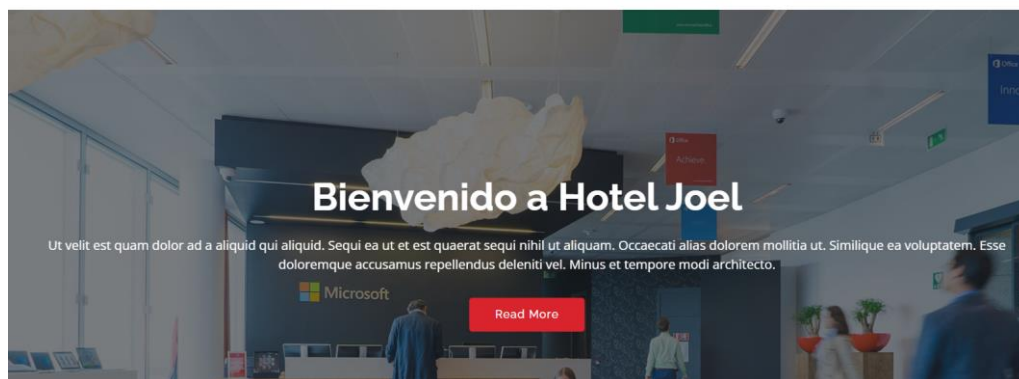
Login

(El usuario vuelve a ser redirigido al formulario de login)

- Entrar con un usuario y contraseña correctos:

JOEL

Home About Services Portfolio Pricing Blog Contact Cerrar Sesión



Se muestra 'hotelesView' y los hoteles disponibles en la base de datos.

**-Entrar a habitaciones sin entrar a hoteles:**

**-Entrar en reservas sin entrar a habitaciones**



## RESERVAS

- Insertar reserva vacía:

### ZONA DE RESERVA

Reserva no disponible o errónea. Cambia la fecha.

×

Detalles de la reserva

- Insertar reserva ya existente

### ZONA DE RESERVA

Reserva no disponible o errónea. Cambia la fecha.

×

Detalles de la reserva

- Insertar reserva con fecha de salida inferior a la fecha de entrada:
















### ZONA DE RESERVA

Reserva no disponible o errónea. Cambia la fecha.

×

Detalles de la reserva

- Resultado:

					id	id_usuario	id_hotel	id_habitacion	fecha_entrada	fecha_salida
<input type="checkbox"/>					1	1	1	1	2022-01-10	2022-01-15
<input type="checkbox"/>					2	1	2	3	2022-01-20	2022-01-25
<input type="checkbox"/>					3	2	1	2	2022-02-01	2022-02-05
<input type="checkbox"/>					4	1	1	1	2024-01-30	2024-01-31
<input type="checkbox"/>					5	1	1	1	2024-03-13	2024-03-15

Ninguna reserva del usuario con ID 3 ( con el que se han hecho las pruebas.





- Insertar reserva con fecha de salida superior a la fecha de entrada:

### ZONA DE RESERVA

La Reserva ha sido realizada con éxito.

×

- Resultado:

<input type="checkbox"/>					5	1	1	1	2024-03-13	2024-03-15
<input type="checkbox"/>					6	3	1	1	2024-01-31	2024-02-02

Ius Reservas, Joel

Detalles	Fecha Entrada	Fecha Salida
Mostrar ▾	2024-01-31	2024-02-02

#### VALIDACIÓN DE PÁGINAS:

- Usuarios View:

Document checking completed. No errors or warnings to show.

0 - - - - -

- Hoteles View:

Document checking completed. No errors or warnings to show.

- Habitaciones View:

Document checking completed. No errors or warnings to show.

- Reservas View:

Document checking completed. No errors or warnings to show.

### COMENTARIO PERSONAL:

Acerca de esta práctica, considero que me ha resultado más que efectiva para aprender de manera muy clara el Modelo Vista Controlador además de la aplicación del Front Controller a esta, que ha hecho todo mucho más rápido, sencillo y mejor organizado.

Al igual que siempre en mi caso, he tenido que realizar la aplicación con un tiempo muy limitado, principalmente por mi culpa por haber tardado en entender como funcionaba dicho modelo y el front controller, pero ahora que lo he terminado podría haberle añadido muchos más detalles y funcionalidades que mejorarían la organización y efectividad de la aplicación y sobre todo una mayor claridad y seguridad del código.

Considero que lo único que me ha resultado dificultoso fue aplicar el MVC y el frontcontroller y después organizar las vistas para mostrar los datos y demás, donde no conseguía tener imaginación y cuando no veo claramente como organizar y mostrar todo, me estanco mucho mucho en un punto.

También pienso que debo repasar ciertas condiciones de las consultas de la base de datos, cosa que los últimos meses apenas he visto consultas relativamente complejas y debo refrescar dichos conocimientos del año pasado.

***Se recuerda que se incluye un documento How To Use en la misma carpeta donde ha encontrado este documento.***