



# UNIVERSIDAD DE GRANADA

## El Gran Hotel

---

**Autor:**

Juan José Martínez Águila  
Alberto Siles Molina



GRADO EN INGENIERÍA INFORMÁTICA

—  
Curso 2023 - 2024

# Índice

<b>1. Estructuras de carpetas</b>	<b>2</b>
<b>2. FrontEnd</b>	<b>2</b>
2.1. Header . . . . .	3
2.1.1. Header usuario anónimo . . . . .	3
2.1.2. Header usuario cliente . . . . .	3
2.1.3. Header usuario administrador . . . . .	4
2.1.4. Header usuario recepcionista . . . . .	4
2.1.5. Header móvil . . . . .	4
2.2. Footer . . . . .	5
2.3. Index . . . . .	5
2.4. Formularios . . . . .	6
2.5. Mostrar las reservas, habitaciones y usuarios . . . . .	7
2.6. Mostrar información de usuario . . . . .	8
2.7. Aviso después de hacer una reserva . . . . .	8
2.8. Mostrar habitación . . . . .	10
<b>3. Backend</b>	<b>10</b>
3.1. Reservas . . . . .	10
3.1.1. MostrarReservas . . . . .	10
3.1.2. ComprobarReserva . . . . .	11
3.1.3. EliminarReserva . . . . .	11
3.1.4. ModificarReserva . . . . .	12
3.1.5. FiltrarReservas . . . . .	12
3.2. ConfirmarReserva . . . . .	13
3.3. InsertReserva . . . . .	13
3.4. Habitaciones . . . . .	13
3.4.1. mostrarInfoHabitacion . . . . .	13
3.4.2. InsertarHabitación . . . . .	14
3.4.3. filtrarHabitaciones . . . . .	14
3.4.4. MostrarHabitaciones . . . . .	14
3.4.5. ObtenerFoto . . . . .	14
3.4.6. EditarHabitacion . . . . .	15
3.5. Usuarios . . . . .	15
3.5.1. MostrarUsuario y mostrarClientes . . . . .	15
3.5.2. RegistrarUsuario . . . . .	15
3.5.3. modificarUsuario . . . . .	15
3.6. CreacionTablas . . . . .	16
3.7. ConexionBD y credenciales . . . . .	16
3.8. FuncionesLogin . . . . .	16
3.9. Header . . . . .	17
3.10. Navs . . . . .	17
3.11. Logs y Backups . . . . .	17

## 1. Estructuras de carpetas

Como se puede observar en la siguiente imagen, el proyecto estará compuesto por:

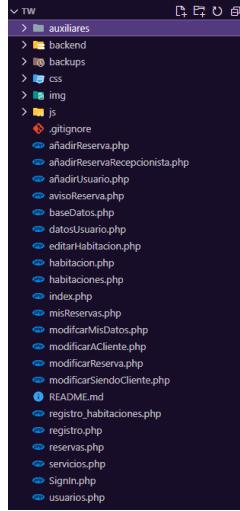


Figura 1: Estructura de carpetas

- **auxiliares:** Aquí estarán guardados todos los archivos .php que se usarán en todos los demás archivos .php como el **footer** o el **header**, también estará el archivo **navs.php** que contiene todas las funciones que muestran el código HTML de los distintos navs.
- **Backend:** Aquí guardaremos todos los ficheros que ejecutan la lógica de negocio de nuestro sitio web, como son las operaciones CRUD de los usuarios, reservas, habitaciones,... También se guardará el fichero que contiene el código para iniciar sesiones y cerrarlas además de las credenciales y la conexión para la base de datos.
- **backups:** Aquí estará todo lo relacionado con los backups y los logs de la base de datos.
- **css:** usamos tailwind pero tenemos ficheros CSS para guardar clases creadas por nosotros que Tailwind no trae implementadas por ejemplo la paleta de colores que vamos usar para nuestro sitio web.
- **img:** Aquí guardaremos las imágenes usadas en nuestra página web como el logo y las imágenes de las distintas habitaciones.
- **js:** Aquí guardaremos el código de javaScript que hará más dinámica nuestra página por ejemplo para el menú en dispositivos móviles y tables.
- **Páginas visitables:** Al mismo nivel que estos directorios estarán todo los archivos .php que serán visitables en nuestra página como puede ser el index o los distintos formularios.

## 2. FrontEnd

Para hacer el frontend de la página hemos utilizado el framework [tailwind CSS](#) que se puede obtener desde la página oficial. Hay distintas maneras de descargarlo y usarlo pero nosotros hemos decidido usarlo con el CDN debido a su facilidad. En un proyecto real es mejor descargarlo debido a que mejora la rapidez de la página web. Aclarar también que toda la página web tiene un diseño responsive, se puede navegar sin problema desde móviles y tablets. Esto ha sido gracias a breakpoints que proporciona tailwind y usando sobretodo las propiedades grid y flex. También quitando elementos o posicionándolos en resoluciones bajas unos debajo de otros en vez de horizontalmente gracias a las dos propiedades mencionadas anteriormente.

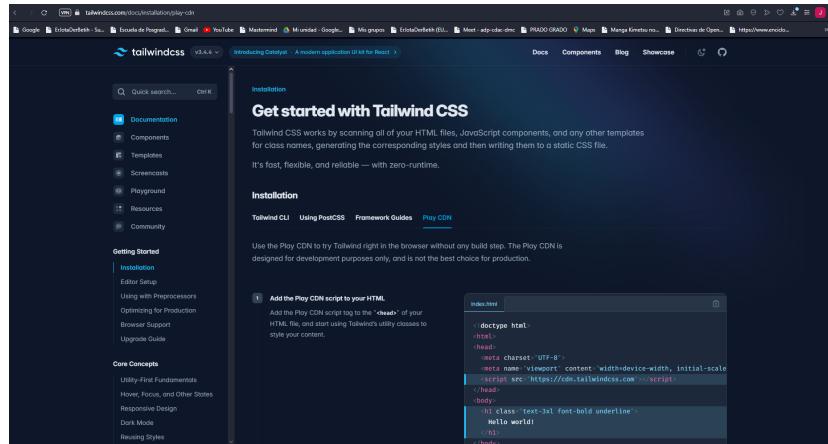


Figura 2: Página de TailwindCSS

## 2.1. Header

Tendremos distintas funciones del navegador dependiendo del tipo de usuario que seas, todas tendrán el mismo maquetado y mismos estilos.

La idea para el header ha sido poner el logo del hotel a la izquierda y el nav a la derecha usando flexbox y space-between con los respectivos padding o margin. Para el dispositivo móvil utilizando los breakpoints que te proporciona tailwind de manera muy sencilla. Además la idea de Tailwind es el principio de mobile First por lo que primero se pone el estilo para el móvil y luego los breakpoints para dispositivos más grandes gracias a md: o lg:. La idea para dispositivos móviles y tables es ocultar el nav y poner el ícono de las tres barras. Con la ayuda de JavaScript cuando se clickea en las barras se muestra el menú del móvil. Con tailwind se hace muy fácil porque lo único que tenemos que hacer con JavaScript es que cuando se cliquea a las tres barras se le añada la clase al menú de navegación que ocupa el resto de la pantalla menos el header **-translate-x-0** y quitarle la clase **translate-x-full**.

### 2.1.1. Header usuario anónimo



Figura 3: Header usuario anónimo

### 2.1.2. Header usuario cliente

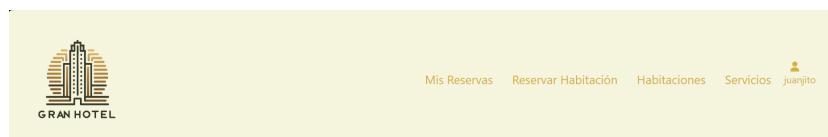


Figura 4: Header usuario cliente

### 2.1.3. Header usuario administrador



Figura 5: Header usuario administrador

### 2.1.4. Header usuario recepcionista



Figura 6: Header usuario recepcionista

### 2.1.5. Header móvil



Figura 7: Header móvil 1/2



Figura 8: Header móvil 2/2

## 2.2. Footer

Para el footer utilizaremos la propiedad grid para tenerlo en 3 columnas que se irán reduciendo según la pantalla del dispositivo que renderice la página. Cada section dentro del footer tendrá la propiedad flex para poder alinear a los elementos dentro de este fácilmente además de conseguir que sea responsive.



Figura 9: Diseño del footer

## 2.3. Index

El Index estará compuesto por una foto de la fachada del Gran hotel y una pequeña descripción, está también estará alineada con la foto con flex-box, siempre se utilizarán las propiedades flex y grid aprovechando su potencialidad para distribuir los distintos elementos en la pantalla y que son responsive. La otra parte del index serán 3 habitaciones que están dentro de un section grid.

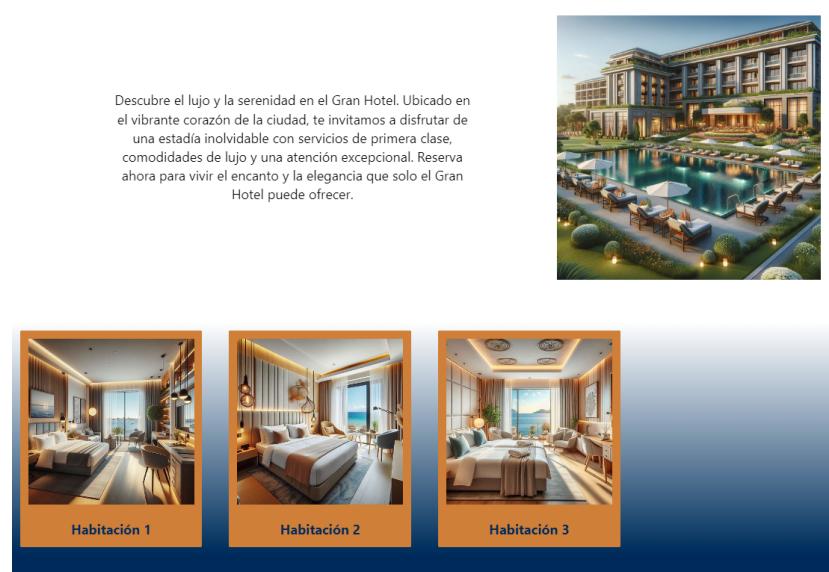


Figura 10: Maquetado del index

Mencionar que para hacerlo responsive utilizamos las propiedades col-span que sirven para determinar cuantas columnas dentro de la maya grid ocupa cada contenedor. Por ejemplo en dispositivos móviles col-span-1.

## 2.4. Formularios

Figura 11: Formulario registro

Este formulario esta hecho con flex y grid para separar por columnas los inputs de los label, ademas aunque no se aprecie cada section contiene dos section uno con el input y el label y otro con un p que muestra un mensaje de error cuando lo hay gracias a javaScript que cuando no matchea por ejemplo el email con una expresión regular de javaScript sse le quita la clase hidden y se muestra el error. También no se puede enviar el formulario hasta que todos los campos matcheen con las distintas expresiones

regulares que hay en el archivo registro.js. Una vez se envía el formulario se envía primero al mismo archivo registro.php pero ahora como es un post se muestran los valores en los inputs con el atributo readonly y el mensaje del botón cambia a confirmar datos. Cuando se pulsa se redirigirá al index y se produce el registro y el login automático además de recargarse la página para evitar que cuando le demos al f5 se envíe otra vez el POST. Básicamente está será la idea de los distintos formularios solo que por ejemplo el de registro de reserva te enviará a reservas en vez de al index y recargará otra vez. Además todos los demás formularios tendrán el mismo maquetado solo que cambiarán los datos que recojamos.

## 2.5. Mostrar las reservas, habitaciones y usuarios

Para estas páginas he utilizado un section con la propiedad grid y luego dentro de este otro más que contendrá la información de los distintos datos que tendrá cada registro, habitación o usuario. En esta página se podrá ver todas las habitaciones, reservas y usuarios siempre que el tipo de usuario tenga permiso de acceder a estas páginas. Si un cliente intenta acceder por ejemplo a la página donde se muestran los usuarios este será redirigido al index mediante esta linea de código `meta http-equiv="refresh" content="0;url=index.php"`.

Cada section dentro del grid tiene la propiedad flex, asignando sus colores mediante las clases que hemos predefinido en el archivo ejercicio14.css y con los padding y margin necesarios ademas de que dentro de estos section están puestos de manera vertical mediante la propiedad flex-col.

Si en estas páginas está metido un recepcionista se verán dos iconos en cada usuario, registro o habitación de edición y para borrar cogidos de [fontawesome](#). Los aside de estas páginas son fixed quiere decir que se desplazan a la vez que el scroll.

The screenshot displays a user management interface. On the left, there is a sidebar titled "Añadir Usuario". The main area contains a grid of six user profiles, each enclosed in a dark blue box with white text. Each profile includes a small edit icon and a trash icon. The profiles are as follows:

- martaGod**: Apellidos: gil, DNI: 12891231E, Email: albertoMaricon@gmail.com, Tarjeta de credito: 1234123412341234, Rol: cliente
- primopepe**: Apellidos: holamamor, DNI: 23232323L, Email: primopepe@gmail.com, Tarjeta de credito: 1234123412341234, Rol: cliente
- russel**: Apellidos: Westbrook, DNI: 23232323P, Email: russel@westbrook.com, Tarjeta de credito: 1234123412341234, Rol: cliente
- juanjito**: Apellidos: martagui, DNI: 23423427P, Email: juanjito@gmail.com, Tarjeta de credito: 1234123412341234, Rol: cliente
- hola mi amor**: Apellidos: tengo que hablar contigo, DNI: 35142787R, Email: carmela@outlook.es, Tarjeta de credito: 1234123412341234
- Lucia**: Apellidos: Bellido, DNI: 23498756M, Email: luciaBellido@gmail.com, Tarjeta de credito: 1234123412341234, Rol: cliente

Figura 12: Gestión de usuarios

Para colocar los aside al lado del section que contiene todos los usuarios por ejemplo, el contendor que contiene a estos dos tendrá la propiedad flex y como por defecto se pone a flex-row ya estaría y así en las otras páginas. Para dispositivos móviles se pone a flex-col. Para mostrar la información se utiliza un listado con ul y li y aplicamos estilos como padding, font-bold para poner el texto en negrita o por ejemplo en el nombre de los usuarios border-bottom.

## 2.6. Mostrar información de usuario



Figura 13: Mostrar datos de usuario

Se sigue la idea del aside que en la página que muestra todos los usuarios. Pero en esta esta compuesta por dos section dentro de otro donde tiene el atributo flex y cuando es en resoluciones más pequeñas con la dirección flex-col-reverse que sería que el section donde está el email y la tarjeta se pone por encima del otro, esto es para que no se junten los fondos blancos del aside con el de este ya que quedaría menos visual la página, recordar que esto se hace con los breakpoints que proporciona TailwindCSS como md: o lg: que quiere decir que las propiedades después de esos puntos se aplican cuando llegan a esas resoluciones.

## 2.7. Aviso después de hacer una reserva

Esta muestra los datos de la reserva recién realizada y dos botones dentro de dos form abajo que pone confirmar o rechazar. Después de todo esto si pasan más de 30 segundos se ejecutará un código JavaScript que te redirigirá a la función para borrar la reserva y esta borrará la reservá recién creada y te devolverá al index.

# Datos de su reserva

**Numero de la reserva:** 63

**Dia de llegada:** 2024-06-07

**Dia de salida:** 2024-06-08

**Número de personas:** 3



Figura 14: Aviso que muestra todos los datos de la reserva

setTimeout es una función en JavaScript que lo que hace es que cuando pasé el tiempo que le pasemos por parámetro se ejecuta el código que hay dentro, por lo tanto en nuestro caso se ejecutó lo explicado anteriormente y encima muestra un alert que avisa de lo que acaba de pasar.

```
You, 20 hours ago | I author (You)
let idReserva = document.getElementById('id_reserva');
setTimeout(function(){
    window.location.href = 'reservas.php?id_reserva=' + idReserva.innerHTML;
    window.alert("Reserva cancelada, No ha sido aceptada en menos de 30 segundos")
}, 30000);
```

Figura 15: Código JavaScript para los 30 segundos

## 2.8. Mostrar habitación

Esta página es la que muestra toda la información necesaria que hay en una habitación. Esta compuesta con un slider que muestra las fotos y los datos que en resoluciones grandes el slider y el section que contiene los datos de la habitación están puestos horizontalmente.

Este slider se ha conseguido gracias a JavaScript y a la propiedad flex-shrink-0 que hace que solo se muestre la imagen que nos interesa encogiendo los flex que tengan esta propiedad. Para los botones del sliders se han colocado poniéndole un position absolute, centrando los y un right-0 y un left-0. y el contenedor de las imágenes se ha puesto como relativo y con JavaScript vamos transladando las imágenes según pulsamos los botones del slider. También si vemos que un contenedor ocupa más ancho o alto de lo que esperamos podemos utilizar w-5/6 o h-5/6 y así con los distintos tamaños.

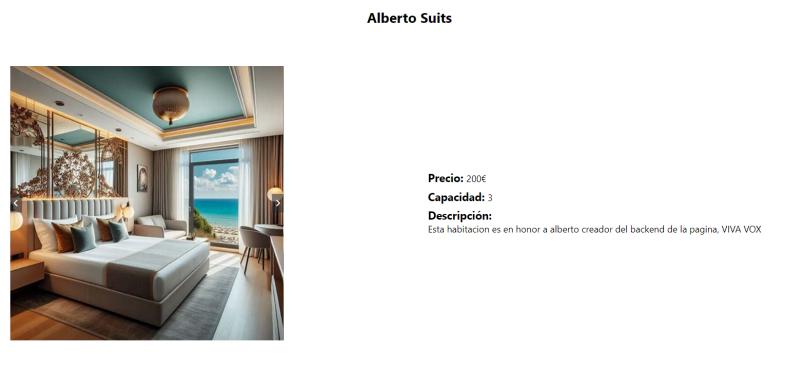


Figura 16: Habitación

## 3. Backend

### 3.1. Reservas

Para esta parte en la parte del servidor, hemos querido modularizar las funciones en un archivo de extensión PHP llamado CrudReservas.php

#### 3.1.1. MostrarReservas

En esta parte tenemos una función que recibe por parámetro el id del usuario que quiere obtener los datos, ya que dentro del propio método para poder reutilizarlo y no crear funciones de más, si el usuario que manda la petición es cliente mostraremos unas reservas (las suyas) y si es recepcionistas tendrá la opción de un botón para poder añadir una reserva y también de listar todas las reservas de todos los clientes. Dentro del PHP generamos el HTML con CSS para poder mostrar las habitaciones.

```

function mostrarReservas($id_usuario){
    require 'conexionBD.php';

    if(esRepcionista()){
        $query = "SELECT id_reserva,id_habitacion,id_cliente,num_pax,dia_entrada,dia_salida,comentario,estado FROM reservas";
    }else{
        $query = "SELECT id_reserva,id_habitacion,id_cliente,num_pax,dia_entrada,dia_salida,comentario,estado FROM reservas WHERE id_cliente=:id";
    }
    $stmt= $conn->prepare($query);
    if(!esRepcionista()){
        $stmt->bindParam(':id', $id_usuario);
    }
    $stmt->execute();

    $resultados = $stmt->fetchAll(PDO::FETCH_ASSOC);

    foreach($resultados as $reserva){

        $query_hab = 'SELECT nombre FROM habitaciones WHERE id_habitacion=:id';
        $statement = $conn->prepare($query_hab);

        $statement->bindParam(':id', $reserva['id_habitacion']);
        $statement->execute();

        $query_cliente = 'SELECT nombre FROM usuarios WHERE id_usuario=:id';
        $stmt2 = $conn->prepare($query_cliente);

        $stmt2->bindParam(':id', $reserva['id_cliente']);
    }
}

```

Figura 17: Funcion para mostrar las reservas

### 3.1.2. ComprobarReserva

Esta función es una función auxiliar que basicamente nos da la habitación en caso de que se pueda para los parametros que le decimos a la reserva, en caso de que no sea posible devolvería false.

```

function comprobarReserva($pax,$fecha_entrada,$fecha_salida){

    require 'conexionBD.php';

    $query = "
        SELECT h.id_habitacion
        FROM habitaciones h
        WHERE h.estado = 'LIBRE'
        AND h.capacidad >= :pax
        AND NOT EXISTS (
            SELECT * FROM reservas r
            WHERE r.id_habitacion = h.id_habitacion
            AND (
                (r.dia_entrada <= :fecha_entrada AND r.dia_salida >= :fecha_entrada)
                OR (r.dia_entrada <= :fecha_salida AND r.dia_salida >= :fecha_salida)
                OR (r.dia_entrada >= :fecha_entrada AND r.dia_salida <= :fecha_salida)
            )
        ) LIMIT 1
    ";

    $stmt= $conn->prepare($query);

    $stmt->bindParam(':pax', $pax);
    $stmt->bindParam(':fecha_entrada', $fecha_entrada);
    $stmt->bindParam(':fecha_salida', $fecha_salida);

    $stmt->execute();

    if($stmt->rowCount()>0){
        return $habitacion = $stmt->fetchAll(PDO::FETCH_ASSOC);
    }
    return false;
}

```

Figura 18: Función para comprobar reservas

### 3.1.3. EliminarReserva

Esta función es una función que elimina una reserva pasandole el ID, es este caso usamos GET. Se ha de tener en cuenta que previamente para eliminar la reserva se pulsa un boton que solo esta visible para el recepcionista, con un if de PHP.

```

function eliminarReserva($id){
    require 'conexionBD.php';
    You, 29 minutes ago • Uncommitted changes
    $query = 'DELETE FROM reservas WHERE id_reserva=:id';

    $stmt= $conn->prepare($query);

    $stmt->bindParam(':id', $id);

    $stmt->execute();

    $query_logs = 'INSERT INTO logs (accion) VALUES (:query);';

    $stmt = $conn->prepare($query_logs);

    $stmt->bindParam(":query",$query);
    $stmt->execute();
}

```

Figura 19: Función para eliminar reserva

### 3.1.4. ModificarReserva

Esta función se encarga de modificar el comentario de una reserva tal y como pone el guión.

### 3.1.5. FiltrarReservas

Esta función filtra las reservas según el estado. Recibe por parametro el estado que quieras mostrar por medio de la variable get. Tambien dentro se lanza otra query para obtener los datos de la persona que ha reservado, ya que al lanzar el select en la tabla reservas, por como hemos creado las tablas obtenemos el id del usuario y asi sacamos el nombre y apellidos, dentro de la función añadimos texto html para poder mostrarlo y dejar mas limpios la página reservas.php .

```

function filtrarReservas($estado){
    require 'conexionBD.php';

    $query = 'SELECT * FROM reservas WHERE estado=:estado';

    $stmt= $conn->prepare($query);

    $stmt->bindParam(':estado', $estado);

    $stmt->execute();

    $resultados = $stmt->fetchAll(PDO::FETCH_ASSOC);

    foreach($resultados as $reserva){
        $query = 'SELECT nombre FROM habitaciones WHERE id_habitacion=:id';

        $stmt = $conn->prepare($query);

        $stmt->bindParam(':id', $reserva['id_habitacion']);

        $stmt->execute();

        $query_cliente = 'SELECT nombre FROM usuarios WHERE id_usuario=:id';

        $stmt2 = $conn->prepare($query_cliente);

        $stmt2->bindParam(':id', $reserva['id_cliente']);

        $stmt2->execute();
    }
}

```

Figura 20: Función para filtrar reservas

### 3.2. ConfirmarReserva

Función para modificar el estado de una reserva justo después de aceptar el mensaje de aviso, se le pasa por parametro el id de la reserva que dentro de la página por el parámetro GET.

### 3.3. InsertReserva

Esta función se encarga de crear una reserva de un cliente ya sea mediante mediación del recepcionista o del usuario. Para ahorrar código hemos creado varios if que reciben por ejemplo por la variable POST que se ha enviado la petición por el formulario del cliente ejecutara una query diferente. Previamente antes de poder crear la query llamamos a comprobarReserva para ver si hay disponibilidad, y en caso de que no avisar al cliente o recepcionista. La función devolverá el objeto sql que contiene la reserva para así mostrarlo en el aviso de reserva

```
function insertReservaPrevia($num_pax, $dia_entrada, $dia_salida, $comentario) {
    require 'conexionBD.php';

    $habitacion = comprobarReserva($num_pax,$dia_entrada,$dia_salida);
    if($habitacion !== FALSE){
        try {
            $query = "INSERT INTO reservas (id_cliente, id_habitacion, dia_entrada, dia_salida, num_pax, comentario, estado)
                      VALUES (:id_cliente, :id_habitacion, :dia_entrada, :dia_salida, :num_pax, :comentario, 'PENDIENTE')";
            $stmt = $conn->prepare($query);

            if(isset($_POST['add_reserva_recepcion'])){
                $query_id = 'SELECT id_usuario FROM usuarios WHERE email=:email';
                $statement = $conn->prepare($query_id);
                $statement->bindParam(':email',$_POST['email']);
                $statement->execute();

                $id_cliente=$statement->fetch(PDO::FETCH_ASSOC);
                $stmt->bindParam(':id_cliente', $id_cliente['id_usuario']);
            }

            if(!esRepcionista()){
                $stmt->bindParam(':id_cliente', $_SESSION['datosUsuario']['id_usuario']);
            }

            $stmt->bindParam(':id_habitacion', $habitacion[0]['id_habitacion']);
            $stmt->bindParam(':dia_entrada',$dia_entrada );
            $stmt->bindParam(':dia_salida',$dia_salida );
            $stmt->bindParam(':num_pax', $num_pax );
            $stmt->bindParam(':comentario', $comentario);

            $stmt->execute();
        }
    }
}
```

Figura 21: Funcion para insertar reservas

```
$stmt->execute();
$query_id = 'SELECT * FROM reservas
WHERE id_habitacion = :id_hab AND id_cliente = :id_cliente
AND dia_entrada = :dia_entrada AND dia_salida = :dia_salida
AND num_pax = :num_pax AND comentario = :comentario';

$statement = $conn->prepare($query_id);
if(!esRepcionista()){
    $statement->bindParam(':id_cliente', $_SESSION['datosUsuario']['id_usuario']);
}
else{
    $statement->bindParam(':id_cliente', $id_cliente['id_usuario']);
}

$statement->bindParam(':id_hab', $habitacion[0]['id_habitacion'] );
$statement->bindParam(':dia_entrada',$dia_entrada );
$statement->bindParam(':dia_salida',$dia_salida );
$statement->bindParam(':num_pax', $num_pax );
$statement->bindParam(':comentario', $comentario);

$statement->execute();

$reserva = $statement->fetchAll(PDO::FETCH_ASSOC);

return $reserva;
```

Figura 22: Funcion para insertar reservas

### 3.4. Habitaciones

#### 3.4.1. mostrarInfoHabitacion

Esta función se encarga de mostrar la información completa de la habitación, recibe por parametros GET la id de la habitación y dentro de la función php metemos texto html con el css para no sobrecargar la página donde se llamará a la función.

### 3.4.2. InsertarHabitación

Esta función inserta una habitación en la tabla pasandole los parametros por referencia, tambien se encarga de en la tabla fotos\_habitaciones que la tenemos como relación entre las fotos y las habitaciones, cree una nueva tupla que relacione las fotos que introducimos con la habitación. En la tabla foto\_habitación almacenamos la ruta donde se almacena la foto.

```
function insertar_habitacion($nombre, $precio, $capacidad, $descripcion, $estado, $num_fotos) {  
    require 'conexionBD.php';  
    try {  
        $conn->beginTransaction();  
  
        $query_habitacion = "INSERT INTO habitaciones (nombre, precio, capacidad, descripcion, estado, fotos)  
                            VALUES (:nombre, :precio, :capacidad, :descripcion, :estado, :num_fotos);"  
        $stmt_habitacion = $conn->prepare($query_habitacion);  
        $stmt_habitacion->bindParam(':nombre', $nombre);  
        $stmt_habitacion->bindParam(':precio', $precio);  
        $stmt_habitacion->bindParam(':capacidad', $capacidad);  
        $stmt_habitacion->bindParam(':descripcion', $descripcion);  
        $stmt_habitacion->bindParam(':estado', $estado);  
        $stmt_habitacion->bindParam(':num_fotos', $num_fotos);  
        $stmt_habitacion->execute();  
  
        $id_habitacion = $conn->lastInsertId();  
  
        for ($i = 0; $i < $num_fotos; $i++) {  
            $foto_nombre = $_FILES['filestoUpload']['name'][$i];  
            $foto_temporal = $_FILES['filestoUpload']['tmp_name'][$i];  
            $ruta_foto = './img/granHotel/habitaciones/' . $foto_nombre;  
  
            $query_foto = "INSERT INTO fotos_habitaciones (id_habitacion, foto) VALUES (:id_habitacion, :ruta_foto);"  
            $stmt_foto = $conn->prepare($query_foto);  
            $stmt_foto->bindParam(':id_habitacion', $id_habitacion);  
            $stmt_foto->bindParam(':ruta_foto', $ruta_foto);  
            $stmt_foto->execute();  
        }  
    }  
}
```

Figura 23: Enter Caption

### 3.4.3. filtrarHabitaciones

Esta función es igual que la función de filtrarReservas solo que filtra por numero de personas que caben en la habitación. Recibe por parámetro el número de personas.

```
function filtrarHabitaciones($pax){  
    require 'conexionBD.php';  
    $query = "SELECT * FROM habitaciones WHERE capacidad=:pax";  
  
    $stmt = $conn->prepare($query);  
    $stmt->bindParam(':pax', $pax);  
    $stmt->execute();  
  
    $resultados = $stmt->fetchAll(PDO::FETCH_ASSOC);  
    foreach($resultados as $hab){  
        $fotos = obtenerFotos($hab['id_habitacion']);  
        echo '';  
        echo ' ';  
        echo ' <figcaption class="p-3 color-azul-marino font-bold text-xl"><a href="habitacion.php?id='.$hab['id_habitacion'].'">'.$hab['nombre'].':</a>';  
        echo ' </figcaption>';  
    }  
}
```

Figura 24: Función para filtrar

### 3.4.4. MostrarHabitaciones

Para esta función como solo puede tener acceso el recepcionista y el administrador, hemos diferenciado en el código html generado hacemos unos if para que el administrador no tenga acceso a la función que no puede acceder. La función hace un select \* from habitaciones y muestra por código html todas las habitaciones.

### 3.4.5. ObtenerFoto

Esta función es una función auxiliar donde obtenemos las fotos de la tabla de fotos de una habitación, previamente obtenemos el id de la habitación por la variable GET.

```

function obtenerFotos($id){

    require 'conexionBD.php';
    $query= 'SELECT * FROM fotos_habitaciones WHERE id_habitacion=:id;';
    $stmt = $conn->prepare($query);
    $stmt->bindParam(':id', $id);
    $stmt->execute();
    $resultados = $stmt->fetchAll(PDO::FETCH_ASSOC);

    return $resultados;

}

```

Figura 25: Función para obtener fotos

### 3.4.6. EditarHabitacion

Esta función se encarga de editar los valores de una habitación, recibe por parametro el id de la habitacion a editar y los parametros nuevos de la habitacion los recibe por la variable post que provienen de un formulario

## 3.5. Usuarios

### 3.5.1. MostrarUsuario y mostrarClientes

Esta función es como las demás de mostrar con la diferencia que mostrarClientes esta hecha para los recepcionistas para que solo vean los clientes y la otra función para los administrados que muestren todos los usuarios. La función genera texto html y css para mostrar los usuarios.

```

function mostrarUsuarios(){
    require 'conexionBD.php';

    $query_select = 'SELECT * FROM usuarios';

    $stmt = $conn->prepare($query_select);

    $stmt->execute();

    $resultados = $stmt->fetchAll(PDO::FETCH_ASSOC);

    foreach ($resultados as $fila) {
        echo "<section class='flex flex-col p-6 bg-color-azul-marino border-solid border-4 border-cobre color-bronce-metalico text-xl lg:text-lg'>
            <div class='w-full flex justify-between'>
                <a class='transition-transform duration-100 hover:scale-105' href='modificarACliente.php?id_usuario_recepcionista=".$fila['id_usuario']."'>$fila['nombre']</a>
                <a class='transition-transform duration-100 hover:scale-105' href='usuarios.php?eliminar_id_usuario=".$fila['id_usuario']."'><i class='fas fa-trash'></i></a>
            </div>
            <section class='grid grid-cols-1'>
                <section class='flex justify-start items-center h-full'>
                    <ul class='pt-6'>
                        <li class='break-all overflow-hidden'><span class='font-bold'>Apellidos: </span>". $fila["apellidos"] . "</li>
                        <li class='pt-6 break-all overflow-hidden'><span class='font-bold'>DNI: </span>". $fila["dni"] . "</li>
                    </ul>
                </section>
                <section class='flex justify-start items-center h-full'>
                    <ul class='pt-6'>
                        <li class='break-all overflow-hidden'><span class='font-bold'>Email: </span>". $fila["email"] . "</li>
                        <li class='pt-6 break-all overflow-hidden'><span class='font-bold'>Tarjeta de crédito: </span>". $fila["tarjeta_credito"] . "</li>
                        <li class='pt-6 break-all overflow-hidden'><span class='font-bold'>Rol: </span>". $fila["rol"] . "</li>
                    </ul>
                </section>
            </section>
        </section>";
    }
}

```

Figura 26: MostrarUsuario

### 3.5.2. RegistrarUsuario

Esta función se encarga de crear un usuario en la página de tipo cliente (viene predefinido por la tabla), se encarga de añadir los valores al cliente y de encriptar la contraseña con la función password\_hash.

### 3.5.3. modificarUsuario

Esta función esta hecha de forma general debido a que dependiendo de quien modifique el usuario, ya sea un cliente recepcionista o administrador. Dependiendo del rol que sea obtenemos una query o otra.

```

function modificarUsuario($id_usuario){
    require 'conexionBD.php';
    try {
        if(esRepcionista()){
            $query_update = "UPDATE usuarios SET nombre = :nombre, apellidos = :apellidos, dni = :dni, email = :email, tarjeta_credito = :tarjeta_credito WHERE id_usuario = :id_usuario";
            $statement = $conn->prepare($query_update);
            $statement->bindParam( ':nombre', $_POST['nombre']);
            $statement->bindParam( ':apellidos', $_POST['apellidos']);
            $statement->bindParam( ':email', $_POST['email']);
            $statement->bindParam( ':dni', $_POST['dni']);
            $statement->bindParam( ':tarjeta_credito', $_POST['tarjeta_credito']);
            $statement->bindParam( ':id_usuario', $id_usuario);
            $statement->execute();
        }else if(esCliente()){
            $query_update = "UPDATE usuarios SET contrasena=:contrasena, email = :email, tarjeta_credito = :tarjeta_credito WHERE id_usuario = :id_usuario";
            $statement = $conn->prepare($query_update);
            $statement->bindParam( ':email', $_POST['email']);
            $statement->bindParam( ':dni', $_POST['dni']);
            $statement->bindParam( ':id_usuario', $id_usuario);
            $statement->execute();
        }else{
            $query_update = "UPDATE usuarios SET nombre = :nombre, apellidos = :apellidos, dni = :dni, email = :email, tarjeta_credito = :tarjeta_credito WHERE id_usuario = :id_usuario";
            $statement = $conn->prepare($query_update);
            $statement->bindParam( ':nombre', $_POST['nombre']);
            $statement->bindParam( ':apellidos', $_POST['apellidos']);
            $statement->bindParam( ':email', $_POST['email']);
            $statement->bindParam( ':dni', $_POST['dni']);
            $statement->bindParam( ':tarjeta_credito', $_POST['tarjeta_credito']);
            $statement->bindParam( ':rol', $_POST['rol']);
            $statement->bindParam( ':id_usuario', $id_usuario);
            $statement->execute();
        }
    }
}

```

Figura 27: Función para modificar Usuario

### 3.6. CreacionTablas

Este fichero contiene el sql necesario para levantar la base de datos de nuestra web y las diferentes tablas.

### 3.7. ConexionBD y credenciales

Son dos ficheros diferente en credenciales tenemos los valores de la base de datos para hacer la conexión. Y conexionBD se encarga de crear un objeto para instanciar la conexión de la base de datos.

### 3.8. FuncionesLogin

Este apartado son funciones auxiliares que nos ayuda a acortar código que se repetía mucho, para comentarlas por encima, son varias funciones para comprobar el rol la variable SESSION, la función isLoggedIn() que comprueba si esta o no logueado el usuario, la función cerrrar\_sesion se encarga de destruir la sesión actual también esta la función que se encarga de loggear al usuario y darle valor a la variable SESSION y de usar password\_verify para validar la contraseña.

```

function login(){
    require 'conexionBD.php';

    if($_SERVER["REQUEST_METHOD"]=="POST"){

        $email=$_POST['email'];
        $contrasena=$_POST['contrasena'];

        $query="SELECT * FROM usuarios WHERE email=:email AND contrasena=:contrasena";

        $stament=$conn->prepare($query);
        $stament->bindParam(":email",$email);
        $stament->bindParam(":contrasena", $contrasena);

        $stament->execute();

        if($stament->rowCount()==1){
            $usuario_log=$stament->fetch(PDO::FETCH_ASSOC);
            $_SESSION['datosUsuario']=$usuario_log;
            $query_logs = 'INSERT INTO logs (accion) VALUES (:query);';

            $stament=$conn->prepare($query_logs);

            $stament->bindParam(":query",$query);
            $stament->execute();

            return true;
        }
        return false;
    }
}

```

Figura 28: Función login

### 3.9. Header

Este fichero contiene contenido html fijo que siempre va en todas las páginas y tambien llama a las funciones mencionadas previamente para que no tengamos que tener mucho código repetido, además se encarga de importar los demás ficheros y de llamar a la función crearTablas.

```

<?php
session_start();
require_once '/backend/crearTablas.php';
require_once '/backend/CRUDusuarios.php';
require_once '/Backend/FuncionesLogin.php';
require_once '/Backend/CRUDReservas.php';
require_once '/Backend/CrudHabitaciones.php';
require_once 'navs.php';
crearTablas();

if(isset($_GET['cerrar_sesion'])){
    cerrarSession();
}
if (isset($_POST['sign_in'])) {
    login();
} elseif (isset($_POST['Registrarse'])) {
    registrarUsuario();
    login();
    echo '<meta http-equiv="refresh" content="0;url=index.php">';
}

if(isLogged()){
    if(erRepcionista()){
        navRepcionistaMovil();
        navRepcionista();
    }else if(esCliente()){
        navClienteMovil();
        navCliente();
    }else{
        navAdministrador();      juanjomrtz19, 15 hours ago + Bug nav solucionado y formulario registro compl.
    }
}
if(!isLogged()){
    navAnonimoMovil();
}

```

Figura 29: Fichero header

### 3.10. Navs

Este fichero llama a funciones auxiliares de funcionesLogin y se encarga de mostrar un nav o otro dependiendo del rol del usuario.

### 3.11. Logs y Backups

Esta parte se centra en el administrador que basa en realizar los backups y mostrar los logs además de borrar la base de datos es decir vaciar las tablas.

```

function vaciarTablas(){
    require 'conexionBD.php';

    $tablas = array("reservas", "fotos_habitaciones", "habitaciones", "usuarios");

    foreach ($tablas as $tabla) {
        $query = "DELETE FROM $tabla";
        $stmt = $conn->prepare($query);
        $stmt->execute();
    }
}

function hacerBackup($backup_path) {
    require 'credenciales.php';

    $command = "mysqldump -h$host -u$username -p$password $database > $backup_path";

    // Ejecutar el comando
    exec($command, $output, $exitCode);

    // Verificar si la ejecución fue exitosa
    if ($exitCode === 0) {
        echo "Backup realizado correctamente en $rutaBackup";
    } else {
        echo "Error al realizar el backup";
    }
}

```

Figura 30: Funciones de Admin

```

function restaurarBackup($rutaBackup) {
    require 'conexionBD.php';

    try {
        $sql = file_get_contents($rutaBackup);
        $conn->exec($sql);
        echo "Backup restaurado correctamente desde $rutaBackup";
    } catch (PDOException $e) {
        echo "Error al restaurar el backup: " . $e->getMessage();
    }
}

function mostrarLogs(){
    require 'conexionBD.php';
    try{
        $query = 'SELECT * FROM logs';
        $stmt = $conn->prepare($query);
        $stmt->execute();
        $logs = $stmt->fetchAll(PDO::FETCH_ASSOC);

        echo '<ul style="list-style-type: none;">';
        foreach ($logs as $log) {
            echo '<li>';
            foreach ($log as $key => $value) {
                echo htmlspecialchars("$key: $value ", ENT_QUOTES, 'UTF-8');
            }
            echo '</li>';
        }
        echo '</ul>';

    }catch(Exception $e){
        echo "Error mostrar los logs: " . $e->getMessage();
    }
}

```

Figura 31: Funciones de admin