



**UNIVERSIDAD DE GUAYAQUIL**

**FACULTAD DE CIENCIAS MATEMÁTICAS Y FÍSICAS**

**CARRERA DE INGENIERÍA DE SOFTWARE**

**DOCUMENTACIÓN DE CÓDIGO**

**PROYECTO DE AEROLINEAS**

**ASIGNATURA:** CONSTRUCCIÓN DE SOFTWARE

**DOCENTE:** ING. JOHANNA ZUMBA GAMBOA

**GRUPO:** #8

**INTEGRANTES:**

- FIGUEROA SALAZAR ODALYS DAYANNA
- ZAPATA LEÓN JOSYMAR
- PINO CRESPO EDGAR
- SUAREZ ORTEGA AARON

**REPOSITORIO DEL PROYECTO**

<https://github.com/SuarezAAa/TravelConnect.git>

# **Documentación de código fuente del Proyecto**

## ÍNDICE

Introducción.....	6
Objetivo .....	6
Distribución de carpetas.....	6
ASSETS.....	7
CONFIG.....	7
Clase Conexion.....	8
Controller .....	8
ClientesController.php .....	8
Método view_clientes() .....	9
Método new_cliente().....	10
Método Lista_cliente() .....	11
HomeController .....	12
HotelController .....	12
Método new_hoteles().....	12
Método register_hotel().....	13
Método view_Editar() .....	14
Método actualizar_hotel() .....	14
Método view_hoteles() .....	15
Método Lista_hoteles() .....	16
Método Eliminar_hotel().....	17
Indexcontroller .....	18
LoginController .....	18
Método login() .....	18
Método Agente() .....	18
Método salirLogin() .....	20
PaquetesController.....	20
Método Lista_paquetes() .....	20
Método register_paquete().....	21
Método actualizar_paquete() .....	23
Método Eliminar_paquete().....	24
ReservasController .....	25
Metodo new_reservaH() .....	25
Metodo new_reservaV() .....	26
Metodo new_reservaP().....	27

VueloController .....	29
Método Lista_aerolinea() .....	29
Método Lista_vuelos() .....	30
Método register_vuelo() .....	31
Método Eliminar_vuelo() .....	32
MODEL .....	34
CARPETA DTO .....	34
ClientesModel.php .....	35
Clase ClientesModel .....	35
Método get_cliente() .....	35
Método get_cliente_by_cedula(\$cedula) .....	35
Función insert_cliente(\$cliente_DAO) .....	36
ConsultaSelectModel.php .....	37
Método consultarCiudad() .....	38
Método consultarAerolinea() .....	38
Método consultarHotel() .....	38
Método consultarVuelo() .....	39
HotelModel.php .....	39
Método get_hotel() .....	39
función registrar(\$hotel_model) .....	40
función selectOne(\$id) .....	41
función update_hotel(\$hotel_model) .....	42
función delete_hotel(\$id) .....	43
LoginModel.php .....	43
Método Agente_list(\$correo, \$pass) .....	43
Método LoginCerrar() .....	44
PaquetesModel.php .....	44
Método get_paquetes() .....	45
función registrar(\$paquete_model) .....	45
Función selectOne(\$id) .....	46
función update_paquete(\$paquete_model) .....	47
función delete_paquete(\$id) .....	47
ReservasModel.php .....	48
Método selectOne(\$valoridc) .....	49
Método selectOneVuelo(\$valoridc) .....	50
Método selectOnePaquete(\$valoridc) .....	50

Método get_reservas_vuelo()	51
Método get_reservas_paquete()	52
Método get_cliente()	52
Método get_cliente_by_cedula(\$cedula)	53
Método get_cliente_id_by_cedula(\$cedula)	54
Método insert_reservaH(\$reservahotel)	54
El método insert_reservaV(\$reservavuelo)	55
VuelosModel.php	56
Función get_aerolinea()	56
Función registrar(\$vuelo_model)	57
Función selectOne(\$id)	58
Función update_vuelo(\$vuelo_model)	58
Función delete_vuelo(\$id)	59
VIEW	59
Clientes	61
Home	62
Hotel	63
Panel	65
Paquetes	67
Listado	67
Reservas	68
Vuelo	69
Login.php	70

## Introducción

El proyecto es una aplicación web desarrollada utilizando la arquitectura Modelo-Vista-Controlador (MVC). Esta arquitectura es ampliamente utilizada en el desarrollo de aplicaciones web debido a su capacidad para separar la lógica de negocio, la presentación y el flujo de control en componentes independientes y reutilizables. La aplicación se ha estructurado de manera modular y organizada, lo que facilita la escalabilidad y el mantenimiento.

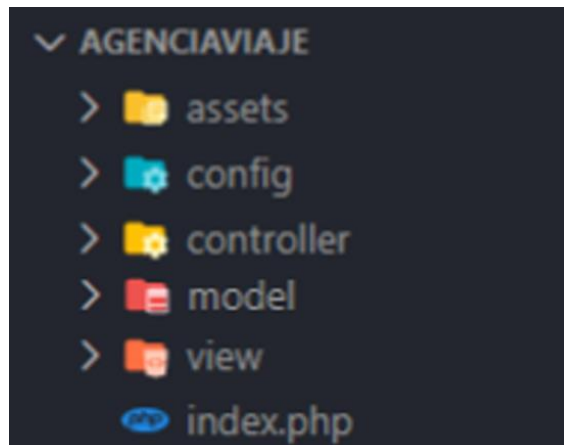
## Objetivo

El objetivo principal de este proyecto es desarrollar una aplicación web que permita gestionar y mostrar información relacionada con vuelos. A través de la arquitectura MVC, buscamos lograr los siguientes objetivos:

- Separación de responsabilidades: La arquitectura MVC permitirá separar la lógica de negocio (modelo), la presentación (vista) y el control del flujo (controlador) en componentes distintos, lo que facilitará la comprensión y el mantenimiento del código.
- Reutilización de código: Al dividir el proyecto en módulos independientes, podremos reutilizar componentes y funcionalidades en diferentes partes de la aplicación, lo que mejorará la eficiencia y la coherencia del código.
- Escalabilidad: La organización del proyecto en carpetas y la separación de responsabilidades nos permitirá añadir nuevas funcionalidades y características de manera sencilla, sin afectar otras partes del sistema.

## Distribución de carpetas

El proyecto se organizará siguiendo la arquitectura Modelo-Vista-Controlador. A continuación, se describe la distribución de carpetas y su función:

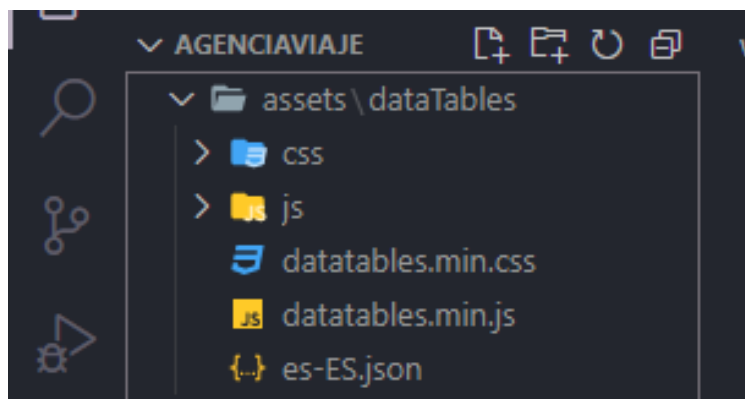


- **ASSETS:** Esta carpeta contendrá los recursos estáticos utilizados en la aplicación, como archivos CSS, JavaScript, imágenes y fuentes. Los archivos de DataTables, mencionados anteriormente, también se pueden ubicar aquí.
- **CONFIG:** En esta carpeta se alojarán archivos relacionados con la configuración de la aplicación, como el archivo conexion.php, que contiene la clase Conexion para conectar con la base de datos.

- **CONTROLLER:** Aquí se ubicarán los controladores de la aplicación. Los controladores son responsables de recibir las solicitudes del usuario, procesar la lógica de negocio y devolver la respuesta adecuada.
- **MODEL:** En esta carpeta estarán los modelos de la aplicación. Los modelos representan la estructura y la lógica de datos del sistema.
- **VIEW:** Aquí se alojarán las vistas de la aplicación. Las vistas son responsables de mostrar la información al usuario, utilizando plantillas y componentes para renderizar el contenido.
- **index.php:** Este archivo será el punto de entrada de la aplicación. Será responsable de iniciar la ejecución del proyecto.

## ASSETS

La carpeta "assets" contiene todos los recursos necesarios para la implementación de DataTables en el proyecto. Esta carpeta incluye subcarpetas para los archivos CSS y JavaScript, así como un archivo .json que proporciona el lenguaje en español para DataTables.



- **Carpetas CSS y JS:** Documenta el contenido de las carpetas "css" y "js". Entre ellos se encuentran los archivos complementarios como estilos de diseño de estilos de proyecto en general, estilos bootstrap y entre otras, en cada carpeta y en la carpeta de js (JavaScript) se describe brevemente su función en la implementación de DataTables.
- **Archivos específicos de DataTables:** Documenta el propósito y la funcionalidad de los archivos CSS y JS de DataTables. Explica cómo estos archivos están relacionados con la apariencia y el comportamiento de las tablas. Si es posible, proporciona enlaces a la documentación oficial de DataTables o cualquier recurso relevante para que los usuarios puedan obtener más información sobre cómo personalizar y usar DataTables.
- **Archivo de lenguaje .json:** Explica que este archivo contiene la traducción de DataTables al español. Describe cómo este archivo se utiliza para cambiar el idioma de la tabla y cómo los usuarios pueden personalizarlo o agregar soporte para otros idiomas si es necesario.

## CONFIG

El archivo conexion.php contiene la clase Conexion, que se encarga de proporcionar una conexión a la base de datos utilizando la extensión PDO (PHP Data Objects). Esta clase está diseñada para conectar a una base de datos MySQL con una configuración predefinida.



```
1 <?php
2
3 class Conexion {
4     public static function getConnection(){
5         $dsn = 'mysql:host=localhost;port=3306;dbname=' . "vuelosbd";
6         $conexion = null;
7
8         try{
9             $conexion = new PDO($dsn, "root", "",array(PDO::MYSQL_ATTR_INIT_COMMAND=>"SET NAMES utf8mb4"));
10        } catch (Exception $e){
11            echo $e;
12            die("ERROR:" . $e->getMessage());
13        }
14
15        return $conexion;
16    }
17
18 }
```

## Clase Conexion

La clase Conexion ofrece un método estático getConnection() que devuelve una instancia de la conexión PDO si la conexión es exitosa. En caso de que ocurra un error durante la conexión, se lanzará una excepción con un mensaje de error descriptivo.

- **Método getConnection():** Es responsable de configurar y establecer la conexión con la base de datos. Utiliza la configuración definida en la variable \$dsn, que especifica la ubicación del servidor, el puerto y el nombre de la base de datos.
  - Si la conexión es exitosa, el método devuelve una instancia de la clase PDO configurada para utilizar la codificación utf8mb4, lo que asegura un manejo adecuado de caracteres multibyte.
  - Si ocurre un error durante la conexión, se captura la excepción y se imprime el mensaje de error utilizando echo. A continuación, la ejecución del programa se detiene con die(), lo que muestra el mensaje de error al usuario.
- **Uso de la clase Conexión:** Para utilizar la clase Conexion, se puede llamar al método estático getConnection() desde cualquier parte del código para obtener una instancia de la conexión PDO. Esta instancia puede utilizarse para realizar operaciones de base de datos, como consultas y actualizaciones.

## Controller

La carpeta "Controller" contiene todos los recursos necesarios para la implementación de los controlares en el proyecto. Esta carpeta incluye archivos PHP

### CientesController.php

En las primeras líneas, se incluyen los archivos necesarios con require\_once para poder utilizar las clases ClientesModel, cliente, ReservasModel y reserva. Esto indica que el código depende de esos archivos para funcionar correctamente.

Se define la clase ClientesController, que será responsable de manejar las acciones relacionadas con los clientes en el sistema.



```

controller > CientesController.php
1 <?php
2     require_once 'model/CientesModel.php';
3     require_once 'model/Dto/cliente.php';
4     require_once 'model/ReservasModel.php';
5     require_once 'model/Dto/reserva.php';
6
7
8     class CientesController
9     {
10         private $model;
11         private $modelR;
12         public function __construct()
13         {
14             {
15                 $this->model = new CientesModel();
16                 $this->modelR = new ReservasModel();
17             }
18
19         public function view_clientes()
20         {
21             require_once 'view/Cientes/Cientes.php';
22         }
23

```

La clase tiene dos propiedades privadas \$model y \$modelR. Estas propiedades se utilizan para crear instancias de las clases CientesModel y ReservasModel, respectivamente, que se utilizarán para interactuar con la base de datos y realizar operaciones relacionadas con clientes y reservas.

El constructor \_\_construct() de la clase se ejecutará cada vez que se cree una instancia de CientesController. Dentro del constructor, se crean las instancias de CientesModel y ReservasModel, lo que permite acceder a sus métodos y datos en los métodos de la clase.

### Método view\_clientes()

se define para mostrar la vista de clientes. Dentro de este método, se utiliza require\_once para cargar el archivo de la vista Cientes.php. La vista es responsable de mostrar los datos de los clientes en el sistema.

### Método new\_cliente()

este método se encarga de procesar los datos de un formulario para crear un nuevo registro de cliente. Configura las propiedades del cliente, intenta insertar el registro en una base de datos y muestra mensajes e íconos apropiados según el resultado. Si la inserción es exitosa, redirige al usuario a una página para gestionar las reservas relacionadas con ese cliente.

Se crea una nueva instancia de la clase 'cliente' (probablemente un objeto de acceso a datos) utilizando la línea: `$cliente_DAO = new cliente();`.

Se establecen varias propiedades del objeto 'cliente' basadas en los datos POST recibidos de un formulario.

```
public function new_cliente(){
    $cliente_DAO = new cliente();
    $cliente_DAO->setNombre($_POST['nombre']);
    $cliente_DAO->setApellido($_POST['Apellido']);
    $cliente_DAO->setCorreo($_POST['Correo']);
    $cliente_DAO->setCedula($_POST['cedula']);
    $cliente_DAO->setTelefono($_POST['telefono']);
    $cliente_DAO->setDireccion($_POST['Direccion']);
    $cliente_DAO->setCiudadFk($_POST['Ciudad_FK']);

    $exito = $this->model->insert_cliente($cliente_DAO);
    if (!$exito) {
        $msj = "Ingrese los datos correctamente";
        $icon = 'error';
    }

    else{
        $id=$this->model->get_cliente_id_by_cedula($_POST['cedula']);
        $msj = 'Guardado exitosamente';
        $icon = 'success';
        //crear reserva

        header('Location:index.php?c=Reservas&a=view_servicios_reservas&idC='.$id);
    }

    $_SESSION['m_crear_usuario'] = $msj;
    $_SESSION['m_icon_interesado'] = $icon;
}
```

### Método Lista\_cliente()

se encarga de obtener una lista de clientes y formatearla en un formato JSON adecuado para su posterior uso en una tabla en una interfaz web.

```
public function Lista_cliente(){  
  
    $datos=$this->model->get_cliente();  
  
    $data= Array();  
    foreach($datos as $row){  
        $sub_array = array();  
        $sub_array[] = $row["nombre"];  
        $sub_array[] = $row["Apellido"];  
        $sub_array[] = $row["Correo"];  
        $sub_array[] = $row["cedula"];  
        $sub_array[] = $row["Direccion"];  
        $sub_array[] = $row["telefono"];  
        $sub_array[] = $row["nombreCiudad"];  
  
        $data[]=$sub_array;  
    }  
    $results = array(  
        "sEcho"=>1,  
        "iTotalRecords"=>count($data),  
        "iTotalDisplayRecords"=>count($data),  
        "aaData"=>$data);  
    /* var_dump($results); */  
    echo $json = json_encode($results);  
}
```

1. El método Lista\_cliente() es llamado para obtener la lista de clientes desde el modelo (posiblemente a través del método get\_cliente()).
2. Se crea un arreglo vacío llamado \$data que se utilizará para almacenar los datos de los clientes.
3. Se recorre cada registro de cliente obtenido en el paso 1 usando un bucle foreach. Para cada registro, se crea un sub-arreglo llamado \$sub\_array que contiene los valores de las columnas relevantes, como el nombre, apellido, correo, cédula, dirección, teléfono y nombre de la ciudad.
4. El sub-arreglo \$sub\_array se agrega al arreglo principal \$data para acumular los datos de todos los clientes.
5. Al final del bucle, se construye un arreglo de resultados llamado \$results, que contiene información necesaria para el procesamiento en el lado del cliente. Esto incluye un contador de registros totales y registros a mostrar (iTotalRecords e iTotalDisplayRecords), así como el contenido real de los datos en el formato adecuado para la tabla (aaData).
6. Se utiliza la función json\_encode() para convertir el arreglo \$results a formato JSON.
7. Finalmente, se imprime el JSON generado utilizando echo, lo que enviará la respuesta

## HomeController

**session\_start();**:: Inicia la sesión de PHP para permitir el uso de variables de sesión en la aplicación.

```
HomeController.php
controller > HomeController.php
1  <?php
2      session_start();
3
4  class HomeController {
5      private $model;
6
7      public function mostrarHome() {
8          require_once 'view/Home/Home.php';
9      }
10 }
11
```

HomeController tiene un método `mostrarHome()` que carga y muestra el contenido de la página de inicio. Es común que los controladores sean responsables de manejar la interacción entre las vistas y los modelos en una arquitectura de software basada en el patrón MVC (Modelo-Vista-Controlador).

## HotelController

### Método `new_hoteles()`

El propósito principal de esta función es mostrar un formulario para añadir nuevos hoteles. Aquí hay un resumen de lo que hace:

1. La función **`new_hoteles()`** es llamada para mostrar el formulario de creación de hoteles.
2. Se crea una instancia de **`ConsultaSelectModel`**, que parece ser un modelo para realizar consultas de selección (probablemente desde una base de datos) relacionadas con las ciudades.
3. Se llama al método **`consultarCiudad()`** del modelo para obtener datos sobre las ciudades. Esto posiblemente se utilice para llenar una lista desplegable en el formulario de creación de hoteles.
4. Se requiere el archivo **`view/Hotel/Hotel_new.php`**, que se encarga de mostrar el formulario de creación de hoteles en la interfaz de usuario.

```
public function new_hoteles()
{
    //comunicarse con el modelo
    $modeloTipoCiudad = new ConsultaSelectModel();
    $ciudad = $modeloTipoCiudad->consultarCiudad();
    require_once 'view/Hotel/Hotel_new.php';
}
```

### Método register\_hotel()

este código controla el proceso de registro de un nuevo hotel, desde la obtención de datos del formulario hasta el almacenamiento y el manejo de mensajes de éxito o error, además de redirigir al usuario a una vista donde puede ver la lista de hoteles.

1. Se instancia un objeto de la clase **hotel** para representar y almacenar los datos del nuevo hotel.
2. Los datos del formulario enviado mediante POST se recogen y se asignan a las propiedades correspondientes del objeto **hotel**. Estos datos incluyen el nombre del hotel, la valoración, la dirección, el precio por noche y la ciudad de destino.
3. Se llama al método **registrar()** del modelo correspondiente para llevar a cabo el registro del hotel en una base de datos.
4. Se verifica si el registro fue exitoso o no. Si no fue exitoso, se establece un mensaje de error y se asigna un icono correspondiente.
5. Si el registro fue exitoso, se establece un mensaje de éxito y se asigna un icono.
6. Se almacenan los mensajes y los iconos en variables de sesión para su posterior visualización.
7. Se redirige al usuario a una página donde puede ver la lista de hoteles, utilizando la función **header()**.

```
public function register_hotel() {
    $hotel_model = new hotel(); // llamar a las entidades de la clase usuarios
    // lectura de parametros
    $hotel_model->setnombre_hotel(htmlspecialchars($_POST['nombre_hotel']));
    $hotel_model->setvaloracion(htmlentities($_POST['valoracion']));
    $hotel_model->setdireccion(htmlspecialchars($_POST['direccion']));
    $hotel_model->setprecioNoche(htmlspecialchars($_POST['precio_hotel']));
    $hotel_model->setciudadhotel(htmlentities($_POST['ciudad_destino']));

    //llamar al modelo
    $exito = $this->model->registrar($hotel_model);

    if (!$exito) {
        $msj = "Hotel ha sido registrado, número de cédula existente";
        $icon = 'error';
    }

    else{
        $msj = 'Guardado exitosamente';
        $icon = 'success';
    }

    $_SESSION['m_crear_usuario'] = $msj;
    $_SESSION['m_icon_interesado'] = $icon;
    header('Location:index.php?c=Hotel&a=view_hoteles');
}
```

### Método view\_Editar()

se encarga de cargar los detalles de un hotel específico que se va a editar, así como la información de las ciudades. Luego, carga la vista correspondiente para mostrar el formulario de edición con los datos relevantes del hotel.

```
public function view_Editar(){  
  
    $this->user = $_GET['hotel_edit'];  
    $hotel_edit = $this->model->selectOne( $this->user);  
    //comunicarse con el modelo  
    $modeloTipoCiudad = new ConsultaSelectModel();  
    $ciudad = $modeloTipoCiudad->consultarCiudad();  
    require_once 'view/Hotel/Hotel_edit.php';  
}
```

### Método actualizar\_hotel()

esta función maneja la actualización de los detalles de un hotel en la base de datos, muestra mensajes de éxito o error y redirige al usuario de regreso a la lista de hoteles después de la actualización.

```
public function actualizar_hotel() {  
    $hotel_model = new hotel(); // llamar a las entidades de la clase usuarios  
    // lectura de parametros  
    $hotel_model->sethotel_id(htmlspecialchars($_POST['id']));  
    $hotel_model->setnombre_hotel(htmlspecialchars($_POST['nombre_hotel']));  
    $hotel_model->setvaloracion(htmlentities($_POST['valoracion']));  
    $hotel_model->setdireccion(htmlspecialchars($_POST['direccion']));  
    $hotel_model->setprecioNoche(htmlspecialchars($_POST['precio_hotel']));  
    $hotel_model->setciudadhotel(htmlentities($_POST['ciudad_destino']));  
  
    //llamar al modelo  
    $exito = $this->model->update_hotel($hotel_model);  
  
    if (!$exito) {  
        $msj = "No se ha actualizado correctamente";  
        $icon = 'error';  
    }  
  
    else{  
        $msj = 'Actualizado exitosamente';  
        $icon = 'success';  
    }  
  
    $_SESSION['m_crear_usuario'] = $msj;  
    $_SESSION['m_icon_interesado'] = $icon;  
    header('Location:index.php?c=Hotel&a=view_hoteles');  
}
```

1. Se instancia un objeto de la clase **hotel** para representar los datos del hotel que se actualizará.
2. Los datos del formulario enviado mediante POST se recogen y se asignan a las propiedades correspondientes del objeto **hotel**. Estos datos incluyen el ID del hotel, el nombre, la valoración, la dirección, el precio por noche y la ciudad de destino.
3. Se llama al método **update\_hotel()** del modelo correspondiente para llevar a cabo la actualización del hotel en la base de datos.
4. Se verifica si la actualización fue exitosa o no. Si no fue exitosa, se establece un mensaje de error y se asigna un icono correspondiente.

### Método `view_hoteles()`

Esta función PHP, llamada `view_hoteles()`, se encarga de cargar y mostrar una vista que presenta una lista de hoteles. Explicación breve de lo que hace:

```
public function view_hoteles()
{
    require_once 'view/Hotel/Hotel_list.php';
}
```

`require_once 'view/Hotel/Hotel_list.php';`: Esta línea carga el archivo `Hotel_list.php` desde el directorio de vistas. Este archivo probablemente contiene el código HTML y PHP necesario para mostrar la lista de hoteles en la interfaz de usuario.

## Método Lista\_hoteles()

obtiene una lista de hoteles desde un modelo, formatea los datos en formato JSON y los envía al cliente

```
public function Lista_hoteles()  
{  
    $datos=$this->model->get_hotel();  
  
    $data= Array();  
    foreach($datos as $row){  
        $sub_array = array();  
        $sub_array[] = $row["nombre_hotel"];  
        $sub_array[] = $row["valoracion"];  
        $sub_array[] = $row["direccion"];  
        $sub_array[] = $row["precioNoche"];  
        $sub_array[] = $row["nombreCiudad"];  
        $sub_array[] =  
'<div class="btn-group btn-group-sm">  
        <a class="btn btn-alt-primary"  
        href="index.php?c=Hotel&a=view_Editar&hotel_edit='.$row["hotel_id"].'">  
        <i class="fa fa-edit mr-5"></i>&nbsp;Editar  
        </a>  
        <a class="btn btn-alt-danger mr-5 mb-5 eliminar"  
        href="index.php?c=Hotel&a=Eliminar_hotel&id='.$row["hotel_id"].'" data-id="'.$row["hotel_id"].'">  
        <i class="fa fa-remove mr-5"></i>&nbsp;Borrar  
        </a>  
        </div>';  
        $data[]=$sub_array;  
    }  
    $results = array(  
        "sEcho">1,  
        "iTotalRecords">count($data),  
        "iTotalDisplayRecords">count($data),  
        "aaData">$data);  
    /* var_dump($results); */  
    echo $json = json_encode($results);  
}
```

se encarga de obtener una lista de hoteles desde el modelo y formatearla en un formato JSON que es utilizado para llenar una tabla en la interfaz de usuario. Explicación general de lo que hace:

1. **public function Lista\_hoteles() { ... }**: Define una función pública llamada **Lista\_hoteles()**.
2. **\$datos=\$this->model->get\_hotel();**: Se llama al método **get\_hotel()** del modelo para obtener datos sobre los hoteles. Estos datos son recuperados y almacenados en la variable **\$datos**.
3. Se crea un arreglo vacío llamado **\$data** que se utilizará para almacenar los datos formateados de los hoteles.
4. Se utiliza un bucle **foreach** para recorrer cada registro de hotel en el arreglo **\$datos**.
  - Dentro del bucle, se crea un sub-arreglo llamado **\$sub\_array** que contiene los valores de las columnas relevantes del hotel, como el nombre, la valoración, la dirección, el precio por noche y el nombre de la ciudad.
  - Se crea una cadena HTML que representa dos botones en una div. Uno de los botones es para editar el hotel y el otro es para eliminarlo. Los enlaces de estos botones contienen parámetros relevantes, como el ID del hotel, que se pasan como parte de la URL.
  - El sub-arreglo **\$sub\_array** se agrega al arreglo principal **\$data**.



5. Se construye un arreglo de resultados llamado **\$results**, que contiene información necesaria para el procesamiento en el lado del cliente. Esto incluye un contador de registros totales y registros a mostrar (**iTotalRecords** e **iTotalDisplayRecords**), así como el contenido real de los datos en el formato adecuado para la tabla (**aaData**).
6. Se utiliza la función **json\_encode()** para convertir el arreglo **\$results** a formato JSON.

### Método Eliminar\_hotel()

se encarga de eliminar un registro de hotel de la base de datos. Luego, devuelve una respuesta en formato JSON indicando si la eliminación fue exitosa o no. Explicación general de lo que hace:

```
public function Eliminar_hotel(){
    $id = $_GET['id'];

    // Eliminar el registro de la base de datos
    $resultado = $this->model->delete_hotel($id);

    // Verificar si la eliminación fue exitosa
    if ($resultado) {
        // La eliminación fue exitosa
        $response = array('exito' => true);
    } else {
        // La eliminación no fue exitosa
        $response = array('exito' => false);
    }

    // Devolver la respuesta en formato JSON
    echo json_encode($response);
}
```

1. **public function Eliminar\_hotel() { ... }:** Define una función pública llamada **Eliminar\_hotel()**.
2. Se obtiene el valor del parámetro "id" de la URL utilizando el método GET y se almacena en la variable **\$id**. Este valor probablemente representa el ID del hotel que se desea eliminar.
3. Se llama al método **delete\_hotel()** del modelo para eliminar el registro del hotel con el ID proporcionado. El resultado de esta operación se almacena en la variable **\$resultado**.
4. Se verifica si la eliminación fue exitosa.
  - Si la eliminación fue exitosa, se crea un arreglo asociativo **\$response** con la clave 'exito' establecida como **true**.
  - Si la eliminación no fue exitosa, se crea un arreglo asociativo **\$response** con la clave 'exito' establecida como **false**.
5. Se utiliza la función **json\_encode()** para convertir el arreglo **\$response** a formato JSON.

## Indexcontroller

**class IndexController { ... }:** Define una clase llamada **IndexController**.

- **public function index() { ... }:** Define un método público llamado **index()**. Este método se encarga de cargar y mostrar la vista de inicio de sesión.
- **require\_once 'view/login.php';:** Carga el archivo **login.php** desde el directorio de vistas. Este archivo probablemente contiene el código HTML y PHP necesario para mostrar el formulario de inicio de sesión en la interfaz de usuario.



```
IndexController.php X
controller > IndexController.php
1  <?php
2
3  class IndexController {
4
5
6      public function index(){
7          require_once 'view/login.php';
8      }
9
10
11 }
```

## LoginController

### Método login()

se encarga de cargar y mostrar la vista de inicio de sesión en la interfaz de usuario. Carga el archivo **login.php** desde el directorio de vistas

```
public function login(){
    require_once 'view/login.php';
}
```

### Método Agente()

la función Agente() maneja el proceso de autenticación de un agente en el sistema. Verifica las credenciales ingresadas, establece variables de sesión y redirige al agente a la página de inicio si las credenciales son válidas, o muestra mensajes de error y carga la vista de inicio de sesión en caso contrario.

```

public function Agente(){
    $errores = '';
    if(!empty($_POST['login-username']) && !empty($_POST['login-password'])){

        $correo = htmlentities($_POST['login-username']);
        $pass = htmlentities($_POST['login-password']);
        $agente = $this->model->Agente_list($correo, $pass);
        if(isset($agente)){
            $_SESSION['user'] = $correo;
            $_SESSION['agente'] = $agente['nombre'];
            $_SESSION['apellido'] = $agente['Apellido'];
            $_SESSION['id'] = $agente['agente_id'];
            $_SESSION['login'] = true;
            /* header("Location:index.php?c=Home&a=mostrarHome"); */
            require_once 'view/Home/Home.php';
        }
        else {
            $errores .= 'La contraseña o correo son incorrecto';
            $_SESSION['login'] = false;
            include 'view/login.php';
        }
    }
    else{
        $errores .= 'No puede quedar datos vacios...';
        $_SESSION['login'] = false;
        include 'view/login.php';
    }
}

```

1. **\$errores = ''**; Se declara una variable **\$errores** inicialmente vacía. Se utilizará para almacenar mensajes de error si ocurrieran durante el proceso de autenticación.
2. Se verifica si se han enviado los campos de nombre de usuario (**login-username**) y contraseña (**login-password**) mediante POST.
  - Si ambos campos tienen valores:
    - Se obtienen y se almacenan los valores de correo electrónico y contraseña de forma segura en las variables **\$correo** y **\$pass**.
    - Se llama al método **Agente\_list()** del modelo con el correo y la contraseña proporcionados para verificar la autenticidad del agente.
    - Si se encuentra un agente válido:
      - Se establecen varias variables de sesión para almacenar información relevante del agente.
      - Se redirige al agente a la página de inicio (se muestra la vista **Home.php**).

- Si no se encuentra un agente válido:
    - Se agrega un mensaje de error a la variable **\$errores**.
    - Se establece la variable de sesión **login** como **false**.
    - Se carga la vista de inicio de sesión (**login.php**) nuevamente.
  - Si algún campo está vacío:
    - Se agrega un mensaje de error a la variable **\$errores**.
    - Se establece la variable de sesión **login** como **false**.
    - Se carga la vista de inicio de sesión (**login.php**) nuevamente.
3. Si no se cumple ninguna de las condiciones anteriores (es decir, si no se enviaron los campos POST):
- Se agrega un mensaje de error a la variable **\$errores**.
  - Se establece la variable de sesión **login** como **false**.
  - Se carga la vista de inicio de sesión (**login.php**) nuevamente.

### Método salirLogin()

este fragmento de código cierra la sesión de usuario (eliminando las variables de sesión) y redirige al usuario de regreso a la página de inicio de sesión.

```
public function salirLogin(){
    $this->model->LoginCerrar();

    header('Location:index.php?c=login&a=login');
}
```

## PaquetesController

### Método Lista\_paquetes()

obtiene una lista de paquetes desde un modelo, formatea los datos en formato JSON y los envía al cliente

```

public function Lista_paquetes(){

    $datos=$this->model->get_paquetes();

    $data= Array();
    foreach($datos as $row){
        $sub_array = array();
        $sub_array[] = $row["nombreP"];
        $sub_array[] = $row["nombre"];
        $sub_array[] = $row["nombre_hotel"];
        $sub_array[] = $row["precio"];
        $sub_array[] =
        '<div class="btn-group btn-group-sm">

            <a class="btn btn-alt-danger mr-5 mb-5 eliminar"
            href="index.php?c=Paquetes&a=Eliminar_paquete&id='.$row['paquete_id'].'" data-id="'.$row['paquete_id'].'">
                <i class="fa fa-remove mr-5"></i>&nbsp;Borrar
            </a>
        </div>';
        $data[]=$sub_array;
    }
    $results = array([
        "sEcho">1,
        "iTotalRecords">count($data),
        "iTotalDisplayRecords">count($data),
        "aaData">$data]);
    /* var_dump($results); */
    echo $json = json_encode($results);
}

```

## Método register\_paquete()

recoge los datos del formulario de registro de paquetes, los procesa, intenta registrar el paquete en la base de datos y luego redirige al usuario a la página de visualización de paquetes con un mensaje de éxito o error.

```

public function register_paquete() {
    $paquete_model = new paquete(); // llamar a las entidades de la clase usuarios
    // lectura de parametros
    $paquete_model->setnombre(htmlspecialchars($_POST['nombre']));
    $paquete_model->setPhotel_fk(htmlentities($_POST['vuelo']));
    $paquete_model->setPvuelo_fk(htmlspecialchars($_POST['hotel']));
    $paquete_model->setprecio(htmlspecialchars($_POST['precio']));

    //llamar al modelo
    $exito = $this->model->registrar($paquete_model);

    if (!$exito) {
        $msj = "Hotel ha sido registrado, número de cédula existente";
        $icon = 'error';
    }

    else{
        $msj = 'Guardado exitosamente';
        $icon = 'success';
    }

    $_SESSION['m_crear_usuario'] = $msj;
    $_SESSION['m_icon_interesado'] = $icon;
    header('Location:index.php?c=Paquetes&a=view_paquetes');
}

```

1. **public function register\_paquete() { ... }**: Define una función pública llamada **register\_paquete()**.
2. Se crea una instancia del modelo **paquete** llamada **\$paquete\_model** para representar los datos del paquete que se va a registrar.
3. Se leen y se sanetizan los parámetros del formulario:
  - El nombre del paquete se obtiene y se almacena en el modelo después de ser saneado.
  - Los IDs de vuelo y hotel se obtienen y se almacenan en el modelo después de ser saneados.
  - El precio del paquete se obtiene y se almacena en el modelo después de ser saneado.
4. Se llama al método **registrar()** del modelo pasando el objeto **\$paquete\_model** para intentar registrar el paquete en la base de datos.
5. Si el registro fue exitoso:
  - Se establece el mensaje de éxito en la variable **\$msj**.
  - Se establece el ícono de éxito en la variable **\$icon**.
6. Si el registro no fue exitoso:

- Se establece un mensaje de error en la variable **\$msj**.
  - Se establece el ícono de error en la variable **\$icon**.
7. Se establecen variables de sesión para almacenar el mensaje y el ícono correspondientes al resultado del registro.
  8. Se redirige al usuario a la página de visualización de paquetes (**view\_paquetes**) usando el encabezado **header()**.

### Método actualizar\_paquete()

```
public function actualizar_paquete() {
    $paquete_model = new paquete(); // llamar a las entidades de la clase usuarios
    // lectura de parametros
    $paquete_model->setpaquete_id(htmlspecialchars($_POST['id']));
    $paquete_model->setnombre(htmlspecialchars($_POST['nombre']));
    $paquete_model->setPhotel_fk(htmlentities($_POST['vuelo']));
    $paquete_model->setPvuelo_fk(htmlspecialchars($_POST['hotel']));
    $paquete_model->setprecio(htmlspecialchars($_POST['precio']));

    //llamar al modelo
    $exito = $this->model->update_paquete($paquete_model);

    if (!$exito) {
        $msj = "Hotel ha sido registrado, número de cédula existente";
        $icon = 'error';
    }

    else{
        $msj = 'Guardado exitosamente';
        $icon = 'success';
    }

    $_SESSION['m_crear_usuario'] = $msj;
    $_SESSION['m_icon_interesado'] = $icon;
    header('Location:index.php?c=Paquetes&a=view_paquetes');
}
```

1. **public function actualizar\_paquete() { ... }**: Define una función pública llamada **actualizar\_paquete()**.
2. Se crea una instancia del modelo **paquete** llamada **\$paquete\_model** para representar los datos del paquete que se va a actualizar.
3. Se leen y se sanetizan los parámetros del formulario:
  - El ID del paquete se obtiene y se almacena en el modelo después de ser saneado.

- El nombre del paquete se obtiene y se almacena en el modelo después de ser saneado.
  - Los IDs de vuelo y hotel se obtienen y se almacenan en el modelo después de ser saneados.
  - El precio del paquete se obtiene y se almacena en el modelo después de ser saneado.
4. Se llama al método **update\_paquete()** del modelo pasando el objeto **\$paquete\_model** para intentar actualizar la información del paquete en la base de datos.
  5. Si la actualización fue exitosa:
    - Se establece el mensaje de éxito en la variable **\$msj**.
    - Se establece el ícono de éxito en la variable **\$icon**.
  6. Si la actualización no fue exitosa:
    - Se establece un mensaje de error en la variable **\$msj**.
    - Se establece el ícono de error en la variable **\$icon**.
  7. Se establecen variables de sesión para almacenar el mensaje y el ícono correspondientes al resultado de la actualización.
  8. Se redirige al usuario a la página de visualización de paquetes (**view\_paquetes**) usando el encabezado **header()**.

### Método Eliminar\_paquete()

se encarga de eliminar un paquete de la base de datos a través de una solicitud AJAX y luego devuelve una respuesta en formato JSON indicando si la eliminación fue exitosa.

```
public function Eliminar_paquete(){
    $id = $_GET['id'];

    // Eliminar el registro de la base de datos
    $resultado = $this->model->delete_paquete($id);

    // Verificar si la eliminación fue exitosa
    if ($resultado) {
        // La eliminación fue exitosa
        $response = array('exito' => true);
    } else {
        // La eliminación no fue exitosa
        $response = array('exito' => false);
    }

    // Devolver la respuesta en formato JSON
    echo json_encode($response);
}
```



## ReservasController

### Metodo new\_reservaH()

se encarga de crear una nueva reserva de hotel en la base de datos. La reserva está asociada a un cliente y a un hotel específico.

```
public function new_reservaH()
{
    $reservahotel = new reservashotel();
    $valoridc = $_SESSION['idCliente'];
    $reservahotel ->setClienteFK($valoridc);
    $reservahotel ->setRHhotelFk($_GET['hid']); // ID hotel */

    $exito = $this->model->insert_reservaH($reservahotel);
    $view_cliente = $this->model->selectOne($valoridc); //ID para el cliente y consultar para factura

    if (!$exito) {
        $msj = "Ingrese los datos correctamente";
        $icon = 'error';
    }

    else{
        $msj = 'Guardado exitosamente';
        $icon = 'success';
        require_once 'view/Reservas/Reservas_factura.php';
    }

    $_SESSION['m_crear_usuario'] = $msj;
    $_SESSION['m_icon_interesado'] = $icon;
}
```

1. **public function new\_reservaH() { ... }:** Define una función pública llamada **new\_reservaH()**.
2. Se crea una instancia de la clase **reservashotel** llamada **\$reservahotel** que representa los datos de la reserva de hotel que se va a crear.
3. Se obtiene el ID del cliente de la variable de sesión **\$\_SESSION['idCliente']** y se almacena en la variable **\$valoridc**.
4. Se establecen los valores en el objeto **\$reservahotel**:
  - Se establece el ID del cliente (**ClienteFK**) en base al valor obtenido de **\$\_SESSION['idCliente']**.
  - Se establece el ID del hotel (**RHhotelFk**) en base al valor obtenido de **\$\_GET['hid']**, que generalmente se pasa como un parámetro en la URL.
5. Se llama al método **insert\_reservaH()** del modelo pasando el objeto **\$reservahotel** para intentar insertar la reserva de hotel en la base de datos.
6. Si la inserción fue exitosa:

- Se establece el mensaje de éxito en la variable **\$msj**.
  - Se establece el ícono de éxito en la variable **\$icon**.
7. Si la inserción no fue exitosa:
- Se establece un mensaje de error en la variable **\$msj**.
  - Se establece el ícono de error en la variable **\$icon**.
8. Se llama al método **selectOne()** del modelo para obtener los datos del cliente en base a su ID (**\$valoridc**). Esto se hace para consultar los datos del cliente para la generación de la factura.
9. Si la inserción fue exitosa, se redirige al usuario a la página de generación de factura de reservas (**Reservas\_factura.php**).

### Metodo new\_reservaV()

crea una nueva reserva de vuelo asociada a un cliente específico en la base de datos y redirige al usuario a la página de generación de factura de vuelo en caso de éxito. También maneja los mensajes y los íconos de éxito/error utilizando variables de sesión.

```
public function new_reservaV()
{
    $reservavuelo = new reservavuelo();
    $valoridc = $_SESSION['idCliente'];
    $reservavuelo ->setClienteFK($valoridc);
    $reservavuelo ->setVueloFK($_GET['idv']); // ID vuelo */

    $exito = $this->model->insert_reservaV($reservavuelo);
    $view_cliente_vuelo = $this->model->selectOneVuelo($valoridc); //ID para el cliente y consultar para factura

    if (!$exito) {
        $msj = "Ingrese los datos correctamente";
        $icon = 'error';
    }
    else{
        $msj = 'Guardado exitosamente';
        $icon = 'success';
        require_once 'view/Reservas/Reservas_facturaVuelo.php';
    }

    $_SESSION['m_crear_usuario'] = $msj;
    $_SESSION['m_icon_interesado'] = $icon;
}
```

1. **public function new\_reservaV() { ... }**: Define una función pública llamada **new\_reservaV()**.
2. Se crea una instancia de la clase **reservavuelo** llamada **\$reservavuelo** que representa los datos de la reserva de vuelo que se va a crear.
3. Se obtiene el ID del cliente de la variable de sesión **\$\_SESSION['idCliente']** y se almacena en la variable **\$valoridc**.
4. Se establecen los valores en el objeto **\$reservavuelo**:
  - Se establece el ID del cliente (**ClienteFK**) en base al valor obtenido de **\$\_SESSION['idCliente']**.

- Se establece el ID del vuelo (**VueloFK**) en base al valor obtenido de **\$\_GET['idv']**, que generalmente se pasa como un parámetro en la URL.
5. Se llama al método **insert\_reservaV()** del modelo pasando el objeto **\$reservavuelo** para intentar insertar la reserva de vuelo en la base de datos.
  6. Si la inserción fue exitosa:
    - Se establece el mensaje de éxito en la variable **\$msj**.
    - Se establece el ícono de éxito en la variable **\$icon**.
  7. Si la inserción no fue exitosa:
    - Se establece un mensaje de error en la variable **\$msj**.
    - Se establece el ícono de error en la variable **\$icon**.
  8. Se llama al método **selectOneVuelo()** del modelo para obtener los datos del cliente en base a su ID (**\$valoridc**). Esto se hace para consultar los datos del cliente para la generación de la factura de vuelo.
  9. Si la inserción fue exitosa, se redirige al usuario a la página de generación de factura de reservas de vuelo (**Reservas\_facturaVuelo.php**).

### Metodo new\_reservaP()

crea una nueva reserva de paquete que puede incluir vuelos y hoteles, asociada a un cliente específico en la base de datos, y redirige al usuario a la página de generación de factura en caso de éxito. También maneja los mensajes y los íconos de éxito/error utilizando variables de sesión.

```
public function new_reservaP()
{
    $valoridc = $_SESSION['idCliente'];

    $reservap = new reservapaquete();
    $reservap->setclienteFK($valoridc);
    $reservap->setPaqueteFK($_GET['idP']); // ID paquete
    $exito = $this->model->insert_reservaP($reservap);

    $view_cliente_paquete = $this->model->selectOnePaquete($valoridc); //ID para el cliente y consultar para factura

    //Crear objeto reserva hotel
    $exito = $this->model->insert_reservaV($reservap);
    if (!$exito) {
        $msj = "Ingrese los datos correctamente";
        $icon = 'error';
    }
    else{
        $msj = 'Guardado exitosamente';
        $icon = 'success';
        require_once 'view/Reservas/Reservas_facturaPaquete.php';
    }
    $_SESSION['m_crear_usuario'] = $msj;
    $_SESSION['m_icon_interesado'] = $icon;
}
```

1. **public function new\_reservaP()** { ... }: Define una función pública llamada `new_reservaP()`.
2. Se obtiene el ID del cliente de la variable de sesión `$_SESSION['idCliente']` y se almacena en la variable `$valoridc`.
3. Se crea una instancia de la clase `reservapaquete` llamada `$reservap` para representar los datos de la reserva de paquete que se va a crear.
4. Se establecen los valores en el objeto `$reservap`:
  - Se establece el ID del cliente (`clienteFK`) en base al valor obtenido de `$_SESSION['idCliente']`.
  - Se establece el ID del paquete (`PaqueteFK`) en base al valor obtenido de `$_GET['idP']`, que generalmente se pasa como un parámetro en la URL.
5. Se llama al método `insert_reservaP()` del modelo pasando el objeto `$reservap` para intentar insertar la reserva de paquete en la base de datos.
6. Se llama al método `selectOnePaquete()` del modelo para obtener los datos del cliente en base a su ID (`$valoridc`). Esto se hace para consultar los datos del cliente para la generación de la factura de paquete.
7. Se crea un objeto de reserva de vuelo y se intenta insertar esta reserva utilizando el método `insert_reservaV()` del modelo. Sin embargo, en el código proporcionado, se está reutilizando la variable `$exito` que se usó para la inserción de la reserva de paquete, lo que podría ser un error.
8. Si la inserción fue exitosa:
  - Se establece el mensaje de éxito en la variable `$msj`.
  - Se establece el ícono de éxito en la variable `$icon`.
9. Si la inserción no fue exitosa:
  - Se establece un mensaje de error en la variable `$msj`.
  - Se establece el ícono de error en la variable `$icon`.
10. Se redirige al usuario a la página de generación de factura de reservas de paquete (`Reservas_facturaPaquete.php`) en caso de éxito.

## VueloController

### Método Lista\_aerolinea()

obtiene los datos de aerolíneas desde la base de datos, los organiza en un formato adecuado para DataTables y los envía como una respuesta JSON para su visualización en una tabla en la página web.

```
public function Lista_aerolinea()  
{  
    $datos=$this->model->get_aerolinea();  
  
    $data= Array();  
    foreach($datos as $row){  
        $sub_array = array();  
        $sub_array[] = $row["aerolinea_id"];  
        $sub_array[] = $row["nombre"];  
        $sub_array[] = $row["aeropuerto"];  
  
        $data[]=$sub_array;  
    }  
    $results = array(  
        "sEcho"=>1,  
        "iTotalRecords"=>count($data),  
        "iTotalDisplayRecords"=>count($data),  
        "aaData"=>$data);  
    /* var_dump($results); */  
    echo $json = json_encode($results);  
}
```

## Método Lista\_vuelos()

obtiene los datos de vuelos desde la base de datos, los organiza en un formato adecuado para DataTables y los envía como una respuesta JSON para su visualización en una tabla en la página web, incluyendo los botones de edición y eliminación.

```
public function Lista_vuelos()  
{  
    $datos=$this->model->get_vuelo();  
  
    $data= Array();  
    foreach($datos as $row){  
        $sub_array = array();  
        $sub_array[] = $row["nombre"];  
        $sub_array[] = $row["origen"];  
        $sub_array[] = $row["nombreCiudad"];  
        $sub_array[] = $row["precio"];  
        $sub_array[] = $row["fecha"];  
        $sub_array[] =  
        '<div class="btn-group btn-group-sm">  
            <a class="btn btn-alt-primary"  
                href="index.php?c=Vuelo&a=view_Editar&vuelo_edit='.$row['vuelo_id'].'">  
                <i class="fa fa-edit mr-5"></i>&nbsp;Editar  
            </a>  
            <a class="btn btn-alt-danger mr-5 mb-5 eliminar"  
                href="index.php?c=Vuelo&a=Eliminar_vuelo&id='.$row['vuelo_id'].' " data-id="'.$row['vuelo_id'].'">  
                <i class="fa fa-remove mr-5"></i>&nbsp;Borrar  
            </a>  
        </div>';  
  
        $data[]=$sub_array;  
    }  
    $results = array(  
        "sEcho"=>1,  
        "iTotalRecords"=>count($data),  
        "iTotalDisplayRecords"=>count($data),  
        "aaData"=>$data;  
        /* var_dump($results); */  
    );  
    echo $json = json_encode($results);  
}
```

1. **public function Lista\_vuelos() { ... }**: Define una función pública llamada **Lista\_vuelos()**.
2. Se obtienen los datos de vuelos utilizando el método **get\_vuelo()** del modelo.
3. Se crea un array llamado **\$data** para almacenar los datos de cada fila de vuelo.
4. Se recorren los datos de vuelos obtenidos en un bucle **foreach**:
  - Se crea un array llamado **\$sub\_array** para almacenar los datos de una fila específica de vuelo.
  - Se agrega el nombre del vuelo, su origen, el nombre de la ciudad, el precio y la fecha al array **\$sub\_array**.
  - Se construye un conjunto de botones de edición y eliminación en la última columna del array **\$sub\_array**.
  - El array **\$sub\_array** se agrega al array **\$data**.

- Se construye un array llamado **\$results** que contiene la información necesaria para DataTables:
  - "sEcho"**: Número de secuencia de la respuesta (generalmente 1).
  - "iTotalRecords"**: Total de registros disponibles en la base de datos.
  - "iTotalDisplayRecords"**: Total de registros que se muestran después de aplicar los filtros.
  - "aData"**: Array que contiene los datos de vuelos con sus botones de edición y eliminación.
- Se codifica el array **\$results** en formato JSON utilizando **json\_encode()**.
- Se imprime el JSON resultante, que será utilizado por la página web para mostrar los datos en una tabla DataTables, incluyendo los botones de edición y eliminación.

### Método register\_vuelo()

registra un nuevo vuelo en la base de datos, establece mensajes de éxito o error en las variables de sesión y redirecciona al usuario a la página de visualización de vuelos después de realizar la operación.

```
public function register_vuelo() {
    $vuelo_model = new vuelo(); // llamar a las entidades de la clase usuarios
    // lectura de parametros
    $vuelo_model->setorigen(htmlspecialchars($_POST['origen']));
    $vuelo_model->setaerolinea_fk(htmlentities($_POST['aerolinea']));
    $vuelo_model->setfecha(htmlspecialchars($_POST['fecha_p']));
    $vuelo_model->setprecio(htmlspecialchars($_POST['precio_vuelo']));
    $vuelo_model->setdestino(htmlentities($_POST['destino_vuelo']));

    //llamar al modelo
    $exito = $this->model->registrar($vuelo_model);

    if (!$exito) {
        $msj = "Hotel ha sido registrado, número de cédula existente";
        $icon = 'error';
    }

    else{
        $msj = 'Guardado exitosamente';
        $icon = 'success';
    }

    $_SESSION['m_crear_usuario'] = $msj;
    $_SESSION['m_icon_interesado'] = $icon;
    header('Location:index.php?c=Vuelo&a=view_vuelos');
}
```

- public function register\_vuelo() { ... }**: Define una función pública llamada **register\_vuelo()**.

2. Se crea una instancia del modelo de vuelo llamada **\$vuelo\_model**.
3. Se leen y sanitizan los parámetros del formulario:
  - Se utiliza **htmlspecialchars()** para evitar ataques XSS en el campo de origen del vuelo.
  - Se utiliza **htmlentities()** para evitar ataques XSS en el campo de aerolínea del vuelo y destino del vuelo.
  - Se utiliza **htmlspecialchars()** para evitar ataques XSS en el campo de fecha del vuelo.
  - Se utiliza **htmlspecialchars()** para evitar ataques XSS en el campo de precio del vuelo.
4. Se configuran los valores de las propiedades del objeto **\$vuelo\_model** con los datos sanitizados.
5. Se llama al método **registrar(\$vuelo\_model)** del modelo para insertar el nuevo registro de vuelo en la base de datos.
6. Se verifica si el registro fue exitoso:
  - Si el registro no fue exitoso, se establece un mensaje de error y un ícono de error en las variables de sesión.
  - Si el registro fue exitoso, se establece un mensaje de éxito y un ícono de éxito en las variables de sesión.
7. Se redirecciona al usuario a la página de visualización de vuelos utilizando **header('Location:index.php?c=Vuelo&a=view\_vuelos')**.

### Método Eliminar\_vuelo()

elimina un registro de vuelo de la base de datos y devuelve una respuesta JSON indicando si la eliminación fue exitosa o no, lo cual puede ser utilizado por la página web para mostrar mensajes al usuario sobre el resultado de la operación.



```

public function Eliminar_vuelo()  

{  

    $id = $_GET['id'];  

    // Eliminar el registro de la base de datos  

    $resultado = $this->model->delete_vuelo($id);  

    // Verificar si la eliminación fue exitosa  

    if ($resultado) {  

        // La eliminación fue exitosa  

        $response = array('exito' => true);  

    } else {  

        // La eliminación no fue exitosa  

        $response = array('exito' => false);  

    }  

    // Devolver la respuesta en formato JSON  

    echo json_encode($response);  

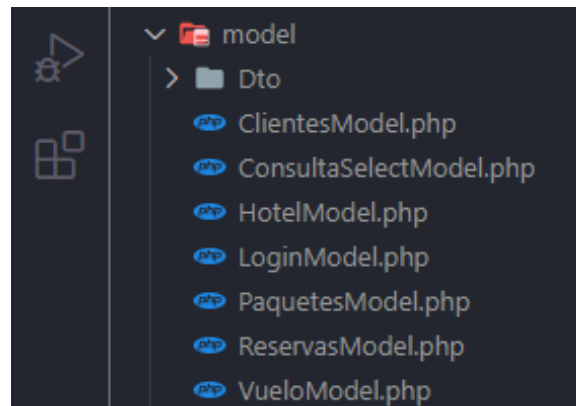
}

```

1. **public function Eliminar\_vuelo() { ... }:** Define una función pública llamada **Eliminar\_vuelo()**.
2. Se obtiene el ID del vuelo a eliminar a través del parámetro **id** en la URL (mediante **\$\_GET['id']**).
3. Se utiliza el método **delete\_vuelo(\$id)** del modelo para eliminar el registro de vuelo correspondiente en la base de datos.
4. Se verifica si la eliminación fue exitosa:
  - Si la eliminación fue exitosa, se crea un array llamado **\$response** con el valor **'exito'** establecido como **true**.
  - Si la eliminación no fue exitosa, se crea un array llamado **\$response** con el valor **'exito'** establecido como **false**.
5. Se codifica el array **\$response** en formato JSON utilizando **json\_encode()**.
6. Se imprime el JSON resultante, que será utilizado por la página web para recibir la respuesta sobre si la eliminación del vuelo fue exitosa o no.

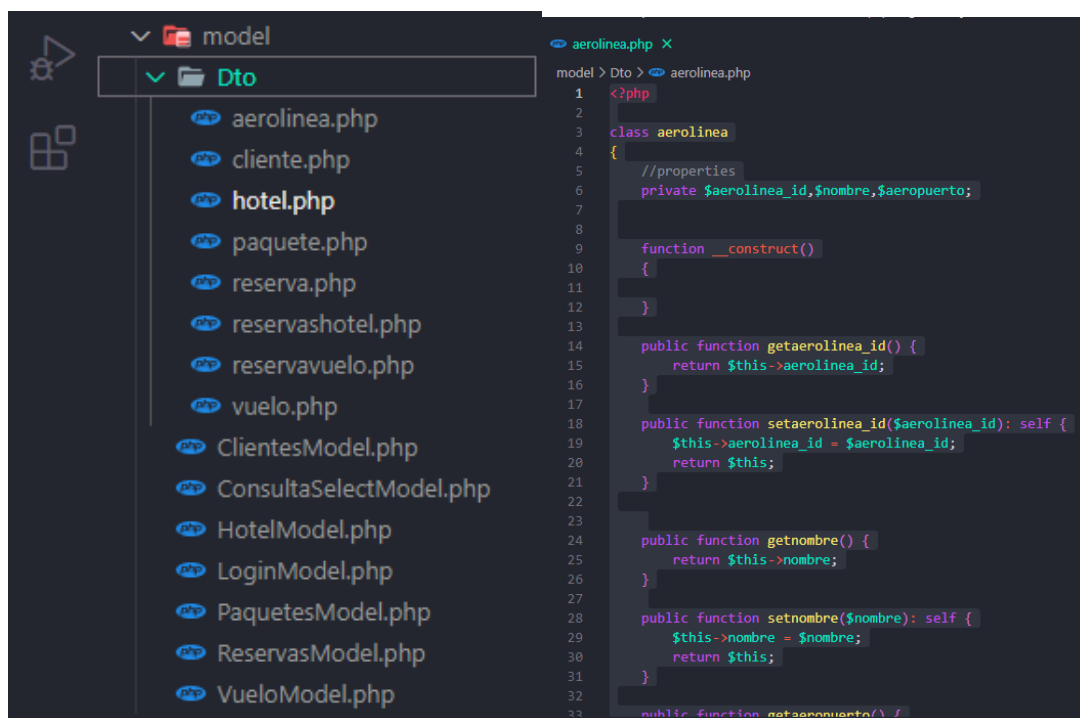
## MODEL

La carpeta "model" contendrá archivos PHP que implementan la lógica y la interacción con la base de datos mediante métodos y funciones. Aquí te muestro cómo podría ser la estructura de esta carpeta.



## CARPETA DTO

En la carpeta "DTO", tenemos los siguientes archivos: aerolinea.php, cliente.php, hotel.php, paquete.php, reserva.php, reservashotel.php, reservavuelo.php. Cada uno se representa como objeto, en este ejemplo mostraremos "aerolínea" y tiene propiedades para el "aerolinea\_id", "nombre" y "aeropuerto". También tiene métodos "get" y "set" para acceder y asignar valores a estas propiedades.



## CientesModel.php

El archivo "ClientesModel.php" contiene la clase ClientesModel, que es responsable de manejar las operaciones relacionadas con la tabla "cliente" en la base de datos. Proporciona dos métodos: get\_cliente() para obtener todos los registros de clientes y get\_cliente\_by\_cedula() para obtener un cliente específico por su número de cédula.

```
ClientesModel.php
require_once 'config/Conexion.php';
class ClientesModel
{
    private $con;
    public function __construct()
    { //constructor
        $this->con = Conexion::getConexion(); //operador :: llamando a un metodo estatico
    }
}
```

### Clase ClientesModel

La clase ClientesModel tiene un constructor que establece la conexión con la base de datos utilizando la clase Conexion del archivo "conexion.php" ubicado en la carpeta "config".

### Método get\_cliente()

El método get\_cliente() consulta la base de datos para obtener todos los registros de la tabla "cliente" junto con la información de la tabla "ciudades" mediante un INNER JOIN. Luego, ejecuta la sentencia SQL y recupera los resultados utilizando fetchAll() para obtener un array con los datos de todos los clientes.

```
}
public function get_cliente(){
    $sql = "SELECT * FROM cliente
    INNER JOIN ciudades ON ciudades.ciudades_id = cliente.Ciudad_FK ";
    // preparar la sentencia
    $stmt = $this->con->prepare($sql);
    // ejecutar la sentencia
    $stmt->execute();
    //recuperar resultados
    $resultado = $stmt->fetchAll(PDO::FETCH_ASSOC);
    //retornar resultados
    return $resultado;
}
```

### Método get\_cliente\_by\_cedula(\$cedula)

La función get\_cliente\_id\_by\_cedula(\$cedula) se utiliza para buscar el cliente\_id correspondiente a una cédula específica en la tabla "cliente" de la base de datos. Toma el número de cédula como parámetro de búsqueda y devuelve el cliente\_id encontrado o false si no se encuentra la cédula.

```

public function get_cliente_id_by_cedula($cedula) {
    $sql = "SELECT cliente_id FROM cliente WHERE cedula = :cedula";

    // Preparar la sentencia
    $stmt = $this->con->prepare($sql);

    // Asignar el valor de la cédula al parámetro :cedula
    $stmt->bindParam(':cedula', $cedula);

    // Ejecutar la sentencia
    $stmt->execute();

    // Recuperar el resultado
    $resultado = $stmt->fetchColumn();

    // Retornar el resultado
    return $resultado;
}

```

#### Parámetro:

- \$cedula: El número de cédula del cliente que se desea buscar en la base de datos.

#### Retorno

- Retorna el cliente\_id (int) si se encuentra la cédula en la base de datos.
- Retorna false si no se encuentra la cédula en la base de datos.

#### Flujo de la función

1. Se construye la sentencia SQL para seleccionar el cliente\_id de la tabla "cliente" donde la cédula sea igual al valor del parámetro :cedula.
2. Se prepara la sentencia SQL utilizando prepare() para evitar posibles ataques de inyección de SQL.
3. Se asigna el valor de la cédula al parámetro :cedula utilizando bindParam().
4. Se ejecuta la sentencia SQL utilizando execute().
5. Se recupera el resultado del cliente\_id utilizando fetchColumn().
6. Se retorna el cliente\_id encontrado (convertido a tipo entero) o false si no se encuentra la cédula en la base de datos.

#### Función insert\_cliente(\$cliente\_DAO)

se utiliza para insertar un nuevo cliente en la tabla "cliente" de la base de datos. Toma un objeto \$cliente\_DAO como parámetro, el cual contiene los datos del cliente que se desea insertar.

```

CientesModel.php

public function insert_cliente($cliente_DAO) {
    try{
        $sql = "INSERT INTO `cliente` (`nombre`, `Apellido`, `Correo`, `cedula`, `telefono`, `Direccion`, `Ciudad_I
            VALUES (:nombre, :apellido, :correo, :cedula, :telefono, :direccion, :ciudad_fk)";
        // Preparar la sentencia
        $stmt = $this->con->prepare($sql);
        $data = [
            'nombre' => $cliente_DAO->getNombre(),
            'apellido' => $cliente_DAO->getApellido(),
            'correo' => $cliente_DAO->getCorreo(),
            'cedula' => $cliente_DAO->getCedula(),
            'telefono' => $cliente_DAO->getTelefono(),
            'direccion' => $cliente_DAO->getDireccion(),
            'ciudad_fk' => $cliente_DAO->getCiudadFk(),
        ];
        // Ejecutar la sentencia
        $stmt->execute($data);

        if ($stmt->rowCount() <= 0) { // verificar si se inserto
            //rowCount permite obtener el numero de filas afectadas
            return false;
        }
    } catch (Exception $e){
        //echo $e->getMessage();
        header('Location:index.php?c=Reservas&a=new_reservas');
        return false;
    }

    return true;
}

```

### Parámetro

- \$cliente\_DAO: Un objeto cliente\_DAO que contiene los datos del cliente que se va a insertar en la base de datos.

### Retorno

- Retorna true si la inserción es exitosa.
- Retorna false en caso de error.

### Flujo de la función

1. Se construye la sentencia SQL para realizar la inserción de un nuevo cliente en la tabla "cliente" con los respectivos campos y marcadores de posición.
2. Se prepara la sentencia SQL utilizando prepare() para evitar posibles ataques de inyección de SQL.
3. Se obtienen los datos del objeto \$cliente\_DAO mediante los métodos "get" y se almacenan en un array llamado \$data, donde las claves corresponden a los nombres de los marcadores de posición en la sentencia SQL.
4. Se ejecuta la sentencia SQL utilizando execute(\$data) para insertar los datos en la base de datos.
5. Si la inserción es exitosa, se retorna true. Si ocurre un error durante la inserción, se captura la excepción, se muestra el mensaje de error utilizando echo y se retorna false.

### ConsultaSelectModel.php

La clase ConsultaSelectModel se utiliza para realizar consultas SELECT en las tablas ciudades, aerolineas, hotel y vuelos de la base de datos y obtener información relacionada con las informacion de cada tabla.

### Método consultarCiudad()

Realiza una consulta SELECT en la tabla "ciudades" para obtener todos los registros. Utiliza la conexión establecida en el constructor para preparar y ejecutar la consulta. Los resultados de la consulta se obtienen en forma de objetos utilizando fetchAll(PDO::FETCH\_OBJ) y se almacenan en un array. Finalmente, se retorna el array con los objetos que representan los datos de las ciudades.

```
12 public function consultarCiudad(){
13     //prepare
14     $sql="SELECT * FROM ciudades";
15     $sentencia = $this->con->prepare($sql);
16     //binding parameters
17     //execute
18     $sentencia->execute();
19     //retornar resultados
20     $resultados = $sentencia->fetchAll(PDO::FETCH_OBJ);
21
22     return $resultados;
23 }
24
25
```

### Método consultarAerolinea()

En este método se usa SELECT en la tabla "aerolinea" para obtener todos los registros. Utiliza la conexión establecida en el constructor para preparar y ejecutar la consulta. Los resultados de la consulta se obtienen en forma de objetos utilizando fetchAll(PDO::FETCH\_OBJ) y se almacenan en un array. Finalmente, se retorna el array con los objetos que representan los datos de las aerolíneas.

```
public function consultarAerolinea(){
    //prepare
    $sql="SELECT * FROM aerolinea";
    $sentencia = $this->con->prepare($sql);
    //binding parameters
    //execute
    $sentencia->execute();
    //retornar resultados
    $resultados = $sentencia->fetchAll(PDO::FETCH_OBJ);

    return $resultados;
}
```

### Método consultarHotel()

El método consultarHotel() realiza una consulta SELECT en la tabla "hotel" para obtener todos los registros. Utiliza la conexión establecida en el constructor para preparar y ejecutar la consulta. Los resultados de la consulta se obtienen en forma de objetos utilizando fetchAll(PDO::FETCH\_OBJ) y se almacenan en un array. Finalmente, se retorna el array con los objetos que representan los datos de los hoteles.

```

public function consultarHotel(){
    //prepare
    $sql="SELECT * FROM hotel";
    $sentencia = $this->con->prepare($sql);
    //binding parameters
    //execute
    $sentencia->execute();
    //retornar resultados
    $resultados = $sentencia->fetchAll(PDO::FETCH_OBJ);

    return $resultados;
}

```

### Método consultarVuelo()

El método consultarVuelo() realiza una consulta SELECT en la tabla "vuelo" y utiliza un INNER JOIN con la tabla "aerolinea" para obtener información relacionada con la aerolínea de cada vuelo. Utiliza la conexión establecida en el constructor para preparar y ejecutar la consulta. Los resultados de la consulta se obtienen en forma de objetos utilizando fetchAll(PDO::FETCH\_OBJ) y se almacenan en un array. Finalmente, se retorna el array con los objetos que representan los datos de los vuelos, incluyendo la información de la aerolínea.

```

53
54 public function consultarVuelo(){
55     //prepare
56     $sql="SELECT * FROM vuelo
57     INNER JOIN aerolinea ON aerolinea.aerolinea_id = vuelo.aerolinea_fk";
58     $sentencia = $this->con->prepare($sql);
59     //binding parameters
60     //execute
61     $sentencia->execute();
62     //retornar resultados
63     $resultados = $sentencia->fetchAll(PDO::FETCH_OBJ);
64
65     return $resultados;
66
67 }
68

```

## HotelModel.php

La clase HotelModel se utiliza para realizar consultas SELECT en la tabla "hotel" de la base de datos y obtener información detallada sobre los hoteles, incluyendo información de la ciudad a la que pertenece cada hotel.

### Método get\_hotel()

El método get\_hotel() realiza una consulta SELECT en la tabla "hotel" y utiliza un INNER JOIN con la tabla "ciudades" para obtener información de la ciudad correspondiente a cada hotel.

- Utiliza la conexión establecida en el constructor para preparar y ejecutar la consulta.
- Los resultados de la consulta se obtienen en forma de array asociativo utilizando fetchAll(PDO::FETCH\_ASSOC) y se almacenan en un array.
- Finalmente, se retorna el array con los arrays asociativos que representan los datos de los hoteles, incluyendo información de la ciudad.

```

HotelModel.php X
model > HotelModel.php
6
7
8     private $con;
9
10    public function __construct()
11    { //constructor
12        $this->con = Conexion::getConexion(); //operador :: llamando a un metodo estatico
13    }
14
15
16    public function get_hotel(){
17        $sql = "SELECT * FROM hotel INNER JOIN ciudades ON ciudades.ciudades_id = hotel.ciudadhotel ";
18        // preparar la sentencia
19        $stmt = $this->con->prepare($sql);
20
21        // ejecutar la sentencia
22        $stmt->execute();
23        //recuperar resultados
24        $resultado = $stmt->fetchAll(PDO::FETCH_ASSOC);
25        //retornar resultados
26        return $resultado;
27    }

```

### función registrar(\$hotel\_model)

La función registrar(\$hotel\_model) en la clase HotelModel se utiliza para registrar un nuevo hotel en la tabla "hotel" de la base de datos. Toma un objeto \$hotel\_model como parámetro, el cual contiene los datos del hotel que se desea registrar.

#### Flujo de la función

1. Se construye la sentencia SQL para realizar la inserción de un nuevo hotel en la tabla "hotel" con los respectivos campos y marcadores de posición.
2. Se prepara la sentencia SQL utilizando prepare() para evitar posibles ataques de inyección de SQL.
3. Se obtienen los datos del objeto \$hotel\_model mediante los métodos "get" y se almacenan en un array llamado \$data, donde las claves corresponden a los nombres de los marcadores de posición en la sentencia SQL.
4. Se ejecuta la sentencia SQL utilizando execute(\$data) para insertar los datos en la base de datos.



```

HotelModel.php
model > HotelModel.php
25 //retornar resultados
26 return $resultado;
27 }
28 public function registrar($hotel_model){
29     try{
30         $sql = "INSERT INTO `hotel` (`nombre_hotel`,`valoracion`,`direccion`,`precioNoche`,`ciudadhotel`)
31             VALUES (:nom_h,:va,:dir,:pre,:ciu)";
32
33         $sentencia = $this->con->prepare($sql);
34
35         $data = [
36             'nom_h' => $hotel_model->getnombre_hotel(),
37             'va' => $hotel_model->getvaloracion(),
38             'dir' => $hotel_model->getdireccion(),
39             'pre' => $hotel_model->getprecioNoche(),
40             'ciu' => $hotel_model->getciudadhotel(),
41         ];
42         //execute
43         $sentencia->execute($data);
44         //retornar resultados,
45         if ($sentencia->rowCount() <= 0) { // verificar si se inserto
46             //rowCount permite obtener el numero de filas afectadas
47             return false;
48         }
49     }catch(Exception $e){
50         echo $e->getMessage();
51         return false;
52     }
53     return true;
54 }
55

```

### función selectOne(\$id)

La función selectOne(\$id) en la clase HotelModel se utiliza para obtener los datos de un hotel específico en la tabla "hotel" de la base de datos. Toma el ID del hotel que se desea obtener como parámetro.

#### Flujo de la función

1. Se construye la sentencia SQL para realizar la consulta SELECT en la tabla "hotel" con INNER JOIN a la tabla "ciudades" utilizando el ID proporcionado.
2. Se prepara la sentencia SQL utilizando prepare() para evitar posibles ataques de inyección de SQL.
3. Se crea un array llamado \$data, que contiene el valor del ID del hotel proporcionado, y se utiliza para ejecutar la sentencia con execute(\$data).
4. Se ejecuta la sentencia SQL y se recupera el primer registro encontrado en forma de objeto utilizando fetch(PDO::FETCH\_OBJ).
5. Se retorna el objeto con los datos del hotel encontrado o NULL si no se encuentra ningún registro con el ID proporcionado.

```

public function selectOne($id) {
    $sql = "SELECT * FROM hotel
        INNER JOIN ciudades ON hotel.ciudadhotel= ciudades.ciudades_id
        where hotel_id=:id";
    // preparar la sentencia
    $stmt = $this->con->prepare($sql);
    $data = ['id' => $id];
    // ejecutar la sentencia
    $stmt->execute($data);
    // recuperar los datos (en caso de select)
    $user = $stmt->fetch(PDO::FETCH_OBJ);
    /* $user = $stmt->fetchAll(PDO::FETCH_ASSOC); */ // fetch retorna el primer registro
    // retornar resultados
    return $user;
}

```

### función update\_hotel(\$hotel\_model)

La función update\_hotel(\$hotel\_model) en la clase HotelModel se utiliza para actualizar los datos de un hotel existente en la tabla "hotel" de la base de datos. Toma un objeto \$hotel\_model como parámetro, el cual contiene los nuevos datos del hotel que se desea actualizar.

#### Flujo de la función

1. Se construye la sentencia SQL para realizar la actualización de los datos del hotel en la tabla "hotel" utilizando los respectivos campos y marcadores de posición, y especificando el hotel a actualizar por su ID.
2. Se prepara la sentencia SQL utilizando prepare() para evitar posibles ataques de inyección de SQL.
3. Se obtienen los nuevos datos del objeto \$hotel\_model mediante los métodos "get" y se almacenan en un array llamado \$data, donde las claves corresponden a los nombres de los marcadores de posición en la sentencia SQL.
4. Se ejecuta la sentencia SQL utilizando execute(\$data) para actualizar los datos en la base de datos.
5. Se verifica si se actualizó algún registro utilizando rowCount(). Si no se actualizó ningún registro, se retorna false.
6. En caso de error durante la actualización, se captura la excepción, se muestra el mensaje de error utilizando echo y se retorna false.
7. Si la actualización es exitosa, se retorna true.

```
HotelModel.php

public function update_hotel($hotel_model){
    try{

        $sql = "UPDATE `hotel` SET nombre_hotel=:nom_h, valoracion=:va,
        direccion=:dir, precioNoche=:pre, ciudadhotel=:ciu
        Where hotel_id =:id";

        $sentencia = $this->con->prepare($sql);

        $data = [
            'id' => $hotel_model->gethotel_id(),
            'nom_h' => $hotel_model->getnombre_hotel(),
            'va' => $hotel_model->getvaloracion(),
            'dir' => $hotel_model->getdireccion(),
            'pre' => $hotel_model->getprecioNoche(),
            'ciu' => $hotel_model->getciudadhotel(),
        ];

        //execute
        $sentencia->execute($data);
        //retornar resultados,

        if ($sentencia->rowCount() <= 0) { // verificar si se inserto
            //rowCount permite obtener el numero de filas afectadas
            return false;
        }
    } catch (Exception $e){
        echo $e->getMessage();
        return false;
    }

    return true;
}
```

### función delete\_hotel(\$id)

La función delete\_hotel(\$id) en la clase HotelModel se utiliza para eliminar un hotel específico de la tabla "hotel" de la base de datos. Toma el ID del hotel que se desea eliminar como parámetro.

#### Flujo de la función

1. Se construye la sentencia SQL para realizar la eliminación del hotel en la tabla "hotel" utilizando el ID proporcionado.
2. Se prepara la sentencia SQL utilizando prepare() para evitar posibles ataques de inyección de SQL.
3. Se crea un array llamado \$data, que contiene el valor del ID del hotel proporcionado, y se utiliza para ejecutar la sentencia con execute(\$data).
4. Se ejecuta la sentencia SQL y se verifica si se eliminó algún registro utilizando rowCount(). Si no se eliminó ningún registro, se retorna false.
5. En caso de error durante la eliminación, se captura la excepción, se muestra el mensaje de error utilizando echo y se retorna false.
6. Si la eliminación es exitosa, se retorna true.

```
104
105 public function delete_hotel($id){
106     //prepare
107     $sql = "DELETE FROM hotel WHERE hotel_id = :id";
108
109     $sentencia = $this->con->prepare($sql);
110     $data = ['id' => $id];
111     //execute
112     $sentencia->execute($data);
113     //retornar resultados,
114     if ($sentencia->rowCount() <= 0) { // verificar si se inserto
115         //rowCount permite obtener el numero de filas afectadas
116         return false;
117     }
118     return true;
119 }
120
121
```

### LoginModel.php

La clase LoginModel se encarga de manejar las operaciones de inicio de sesión y cierre de sesión de los agentes en el sistema.

```
el > LoginModel.php
<?php
require_once 'config/Conexion.php';
class LoginModel
{
    private $con;
    public function __construct()
    {
        $this->con = Conexion::getConexion();
    }
}
```

### Método Agente\_list(\$correo, \$pass)

Este método se utiliza para realizar el inicio de sesión de un agente en el sistema. Toma como parámetros el correo electrónico y la contraseña del agente. Retorna un array asociativo con los datos del agente si el inicio de sesión es exitoso o NULL si las credenciales son incorrectas.

```

public function Agente_list($correo, $pass)
{
    $sql = "SELECT agente_id,nombre, Apellido
    FROM agentes where Correo=:correo and password=:pass";
    $stmt = $this->con->prepare($sql);
    $data = ['correo' => $correo, 'pass' => $pass,];
    $stmt->execute($data); //ejecuto la sentencia
    $resultado = $stmt->fetch(PDO::FETCH_ASSOC);
    if ($stmt->rowCount() <= 0) { // verificar si se inserto
        return $resultado = null;
    }
    return $resultado;
}

```

### Método LoginCerrar()

Este método se utiliza para cerrar la sesión activa del agente. Elimina la variable de sesión \$\_SESSION['login'] y destruye la sesión.

```

public function LoginCerrar (){
    unset($_SESSION['login']);
    session_destroy();
}

```

### Flujo de la clase

1. El constructor de la clase establece la conexión con la base de datos utilizando la clase Conexion.
2. El método Agente\_list(\$correo, \$pass) prepara la sentencia SQL para obtener los datos del agente que coincidan con el correo electrónico y la contraseña proporcionados.
3. Se ejecuta la sentencia SQL utilizando execute(\$data) y se recupera el resultado utilizando fetch(PDO::FETCH\_ASSOC).
4. Si no se encuentra ningún registro con las credenciales proporcionadas (rowCount() es menor o igual a 0), se retorna NULL.
5. Si el inicio de sesión es exitoso (rowCount() es mayor a 0), se retorna un array asociativo con los datos del agente.
6. El método LoginCerrar() se utiliza para cerrar la sesión activa del agente. Esto se logra eliminando la variable de sesión \$\_SESSION['login'] y destruyendo la sesión con session\_destroy().

### PaquetesModel.php

La clase PaquetesModel se encarga de realizar operaciones relacionadas con la obtención de datos de paquetes turísticos desde la base de datos. Contiene una función principal llamada get\_paquetes() que se utiliza para obtener la información completa de los paquetes turísticos, incluyendo detalles de la aerolínea, vuelo y hotel asociados a cada paquete.

```

> PaquetesModel.php
<?php

require_once 'config/Conexion.php';

class PaquetesModel
{
    private $con;

    public function __construct()
    { //constructor
        $this->con = Conexion::getConexion(); //operador :: llamando a un metodo estatico
    }
}

```

### Método `get_paquetes()`

Este método se utiliza para obtener la información completa de todos los paquetes turísticos desde la base de datos. Retorna un array con la información de cada paquete, incluyendo detalles de la aerolínea, vuelo y hotel asociados a cada paquete.

#### Flujo de la clase

1. El constructor de la clase establece la conexión con la base de datos utilizando la clase `Conexion`.
2. El método `get_paquetes()` construye la sentencia SQL para obtener la información completa de los paquetes turísticos, incluyendo los datos de las tablas "paquetes", "aerolinea", "vuelo" y "hotel" mediante `INNER JOIN`.
3. Se prepara la sentencia SQL utilizando `prepare()` para evitar posibles ataques de inyección de SQL.
4. Se ejecuta la sentencia SQL utilizando `execute()`.
5. Se recuperan los resultados en forma de array asociativo utilizando `fetchAll(PDO::FETCH_ASSOC)`.
6. Se retorna el array con la información completa de los paquetes turísticos.

```
public function get_paquetes(){
    $sql = "SELECT * FROM paquetes
    INNER JOIN aerolinea ON aerolinea.aerolinea_id = paquetes.Pvuelo_fk
    INNER JOIN vuelo ON vuelo.vuelo_id = paquetes.Pvuelo_fk
    INNER JOIN hotel ON hotel.hotel_id = paquetes.Photel_fk ";
    // preparar la sentencia
    $stmt = $this->con->prepare($sql);

    // ejecutar la sentencia
    $stmt->execute();
    //recuperar resultados
    $resultado = $stmt->fetchAll(PDO::FETCH_ASSOC);
    //retornar resultados
    return $resultado;
}
```

### función `registrar($paquete_model)`

La función `registrar($paquete_model)` en la clase `PaquetesModel` se utiliza para insertar un nuevo paquete turístico en la base de datos. Toma como parámetro un objeto de la clase `PaqueteModel` que contiene los datos del paquete que se desea registrar.

#### Flujo de la función

1. Se construye la sentencia SQL para realizar el registro del paquete en la tabla "paquetes" utilizando los datos proporcionados por el objeto `PaqueteModel`.
2. Se prepara la sentencia SQL utilizando `prepare()` para evitar posibles ataques de inyección de SQL.
3. Se crea un array llamado `$data`, que contiene los valores de los campos del paquete a registrar, obtenidos del objeto `PaqueteModel`.
4. Se ejecuta la sentencia SQL y se verifica si se insertó algún registro utilizando `rowCount()`. Si no se insertó ningún registro, se retorna `false`.
5. En caso de error durante el registro, se captura la excepción, se muestra el mensaje de error utilizando `echo` y se retorna `false`.
6. Si el registro es exitoso, se retorna `true`.

```

el > PaquetesModel.php
}

public function registrar($paquete_model){

    try{

        $sql = "INSERT INTO `paquetes` (`nombreP`,`Pvuelo_fk`,`Photel_fk`, `precio`)
        VALUES (:nom,:vu,:ho,:pre)";

        $sentencia = $this->con->prepare($sql);

        $data = [
            'nom' => $paquete_model->getnombre(),
            'vu' => $paquete_model->getPvuelo_fk(),
            'ho' => $paquete_model->getPhotel_fk(),
            'pre' => $paquete_model->getprecio(),
        ];

        //execute
        $sentencia->execute($data);
        //retornar resultados,

        if ($sentencia->rowCount() <= 0) { // verificar si se inserto
            //rowCount permite obtener el numero de filas afectadas
            return false;
        }
    } catch (Exception $e){
        echo $e->getMessage();
        return false;
    }

    return true;
}

```

### Función selectOne(\$id)

La función selectOne(\$id) en la clase PaquetesModel se utiliza para obtener la información completa de un paquete turístico específico, identificado por su paquete\_id.

```

public function selectOne($id) {

    $sql ="SELECT * FROM paquetes
    INNER JOIN aerolinea ON aerolinea.aerolinea_id = paquetes.Pvuelo_fk
    INNER JOIN vuelo ON vuelo.vuelo_id = paquetes.Pvuelo_fk
    INNER JOIN hotel ON hotel.hotel_id = paquetes.Photel_fk where paquete_id=:id";
    // preparar la sentencia
    $stmt = $this->con->prepare($sql);
    $data = ['id' => $id];
    // ejecutar la sentencia
    $stmt->execute($data);
    // recuperar los datos (en caso de select)
    $user = $stmt->fetch(PDO::FETCH_OBJ);
    /* $user = $stmt->fetchAll(PDO::FETCH_ASSOC); */// fetch retorna el primer registro
    // retornar resultados
    return $user;
}

```

### Flujo de la función

1. Se construye la sentencia SQL para obtener la información completa del paquete específico utilizando los datos de las tablas "paquetes", "aerolinea", "vuelo" y "hotel" mediante INNER JOIN. Se utiliza un WHERE para filtrar por el paquete\_id proporcionado.
2. Se prepara la sentencia SQL utilizando prepare() para evitar posibles ataques de inyección de SQL.
3. Se crea un array llamado \$data, que contiene el valor del ID del paquete proporcionado.
4. Se ejecuta la sentencia SQL con el ID del paquete utilizando execute(\$data).
5. Se recupera el resultado en forma de objeto utilizando fetch(PDO::FETCH\_OBJ).
6. Se retorna el objeto con la información completa del paquete.

### función update\_paquete(\$paquete\_model)

La función update\_paquete(\$paquete\_model) en la clase PaquetesModel se utiliza para actualizar la información de un paquete turístico existente en la base de datos.

```
model > PaquetesModel.php
//
78     }
79
80     public function update_paquete($paquete_model){
81         try{
82
83             $sql = "UPDATE `paquetes` SET nombreP=:nom, Pvuelo_fk=:vu,
84             Photo1_fk=:ho, precio=:pre
85             Where paquete_id =:id";
86
87             $sentencia = $this->con->prepare($sql);
88
89             $data = [
90                 'id' => $paquete_model->getpaquete_id(),
91                 'nom' => $paquete_model->getnombre(),
92                 'vu' => $paquete_model->getPvuelo_fk(),
93                 'ho' => $paquete_model->getPhoto1_fk(),
94                 'pre' => $paquete_model->getprecio(),
95             ];
96             //execute
97             $sentencia->execute($data);
98             //retornar resultados,
99
100             if ($sentencia->rowCount() <= 0) { // verificar si se inserto
101                 //rowCount permite obtener el numero de filas afectadas
102                 return false;
103             }
104         }catch(Exception $e){
105             echo $e->getMessage();
106             return false;
107         }
108         return true;
109     }
```

### Flujo de la función

1. Se construye la sentencia SQL para realizar la actualización de la información del paquete en la tabla "paquetes" utilizando los datos proporcionados por el objeto PaqueteModel.
2. Se prepara la sentencia SQL utilizando prepare() para evitar posibles ataques de inyección de SQL.
3. Se crea un array llamado \$data, que contiene los nuevos valores de los campos del paquete actualizado, obtenidos del objeto PaqueteModel.
4. Se ejecuta la sentencia SQL con los datos actualizados utilizando execute(\$data).
5. Se verifica si se actualizó algún registro utilizando rowCount(). Si no se actualizó ningún registro, se retorna false.
6. En caso de error durante la actualización, se captura la excepción, se muestra el mensaje de error utilizando echo y se retorna false.
7. Si la actualización es exitosa, se retorna true.

### función delete\_paquete(\$id)

La función delete\_paquete(\$id) en la clase PaquetesModel se utiliza para eliminar un paquete turístico existente de la base de datos.

```

110
111 public function delete_paquete($id){
112     //prepare
113     $sql = "DELETE FROM paquetes WHERE paquete_id = :id";
114
115     $sentencia = $this->con->prepare($sql);
116     $data = ['id' => $id];
117     //execute
118     $sentencia->execute($data);
119     //retornar resultados,
120     if ($sentencia->rowCount() <= 0) { // verificar si se inserto
121         //rowCount permite obtener el numero de filas afectadas
122         return false;
123     }
124     return true;
125
126 }
127

```

### Flujo de la función

1. Se construye la sentencia SQL para eliminar el paquete de la tabla "paquetes" utilizando el ID proporcionado.
2. Se prepara la sentencia SQL utilizando prepare() para evitar posibles ataques de inyección de SQL.
3. Se crea un array llamado \$data, que contiene el valor del ID del paquete que se desea eliminar.
4. Se ejecuta la sentencia SQL con el ID del paquete utilizando execute(\$data).
5. Se verifica si se eliminó algún registro utilizando rowCount(). Si no se eliminó ningún registro, se retorna false.
6. En caso de error durante la eliminación, se captura la excepción, se muestra el mensaje de error utilizando echo y se retorna false.
7. Si la eliminación es exitosa, se retorna true.

### ReservasModel.php

El archivo ReservasModel.php contiene la clase ReservasModel, la cual se encarga de realizar operaciones relacionadas con las reservas de hoteles en la base de datos. A continuación, te proporciono la documentación para esta clase y su método get\_reservas\_hotel():

### Flujo del método

1. Se construye la sentencia SQL para obtener el listado de reservas de hoteles, realizando un INNER JOIN con las tablas "hotel", "cliente" y "ciudades" para obtener la información completa de cada reserva.
2. Se prepara la sentencia SQL utilizando prepare() para evitar posibles ataques de inyección de SQL.
3. Se ejecuta la sentencia SQL utilizando execute().
4. Se recuperan los resultados de la consulta utilizando fetchAll(PDO::FETCH\_ASSOC), lo que devuelve un array asociativo con la información de todas las reservas de hoteles.
5. Se retorna el array con el listado de reservas de hoteles.



```

model > ReservasModel.php
1  <?php
2
3  require_once 'config/Conexion.php';
4
5  class ReservasModel
6  {
7
8      private $con;
9
10     public function __construct()
11     { //constructor
12         $this->con = Conexion::getConexion(); //operador :: llamando a un metodo estatico
13     }
14
15     /* Listado de reservas */
16     public function get_reservas_hotel(){
17         $sql = "SELECT * FROM `reservas_hotel`
18             INNER JOIN hotel ON hotel.hotel_id = reservas_hotel.RHhotel_fk
19             INNER JOIN cliente ON cliente.cliente_id = reservas_hotel.cliente_fk
20             INNER JOIN ciudades ON ciudades.ciudades_id = hotel.ciudadhotel";
21         // preparar la sentencia
22         $stmt = $this->con->prepare($sql);
23
24         // ejecutar la sentencia
25         $stmt->execute();
26         //recuperar resultados
27         $resultado = $stmt->fetchAll(PDO::FETCH_ASSOC);
28         //retornar resultados
29         return $resultado;
30     }
31

```

### Método selectOne(\$valoridc)

El método selectOne(\$valoridc) en la clase ReservasModel se utiliza para obtener la información de una reserva de hotel específica asociada a un cliente dado. Toma el valor del cliente\_id como parámetro y devuelve un objeto con la información de la reserva encontrada.

```

public function selectOne($valoridc) {

    $sql = "SELECT *
    FROM reservas_hotel
    INNER JOIN cliente ON cliente.cliente_id = reservas_hotel.cliente_fk
    INNER JOIN hotel ON hotel.hotel_id = reservas_hotel.RHhotel_fk
    INNER JOIN ciudades ON ciudades.ciudades_id = hotel.ciudadhotel
    WHERE cliente_id = :valoridc";

    // preparar la sentencia
    $stmt = $this->con->prepare($sql);
    $data = ['valoridc' => $valoridc];
    // ejecutar la sentencia
    $stmt->execute($data);
    // recuperar los datos (en caso de select)
    $view_cliente = $stmt->fetch(PDO::FETCH_OBJ);
    /* $user = $stmt->fetchAll(PDO::FETCH_ASSOC); */// fetch retorna el primer registro
    // retornar resultados
    return $view_cliente;
}

```

### Flujo del método

1. Se construye la sentencia SQL para obtener la información de la reserva de hotel asociada al cliente específico, realizando INNER JOIN con las tablas "cliente", "hotel" y "ciudades" para obtener la información completa de la reserva.
2. Se prepara la sentencia SQL utilizando prepare() para evitar posibles ataques de inyección de SQL.
3. Se crea un array llamado \$data, que contiene el valor del ID del cliente para filtrar las reservas asociadas a ese cliente.
4. Se ejecuta la sentencia SQL con el ID del cliente utilizando execute(\$data).
5. Se recupera la información de la reserva encontrada utilizando fetch(PDO::FETCH\_OBJ), lo que devuelve un objeto con la información de la reserva asociada al cliente.
6. Se retorna el objeto con la información de la reserva encontrada.

### Método selectOneVuelo(\$valoridc)

En la clase ReservasModel se utiliza para obtener la información de una reserva de vuelo específica asociada a un cliente dado. Toma el valor del cliente\_id como parámetro y devuelve un objeto con la información de la reserva de vuelo encontrada.

```
52
53     public function selectOneVuelo($valoridc) {
54
55         $sql = "SELECT *, origenes.nombreCiudad AS vuelo_o, destinos.nombreCiudad AS Vuelo_d,
56             aerolinea.nombre AS nombre_A, cliente.nombre AS nombre_C
57         FROM reservas_vuelo
58         INNER JOIN cliente ON cliente.cliente_id = reservas_vuelo.cliente_fk
59         INNER JOIN vuelo ON vuelo.vuelo_id = reservas_vuelo.vuelo_fk
60         INNER JOIN aerolinea ON aerolinea.aerolinea_id = vuelo.aerolinea_fk
61         INNER JOIN ciudades AS origenes ON origenes.ciudades_id = vuelo.origen
62         INNER JOIN ciudades AS destinos ON destinos.ciudades_id = vuelo.destino
63         WHERE cliente_id = :valoridc";
64
65         // preparar la sentencia
66         $stmt = $this->con->prepare($sql);
67         $data = ['valoridc' => $valoridc];
68         // ejecutar la sentencia
69         $stmt->execute($data);
70         // recuperar los datos (en caso de select)
71         $view_cliente = $stmt->fetch(PDO::FETCH_OBJ);
72         /* $user = $stmt->fetchAll(PDO::FETCH_ASSOC); */ // fetch retorna el primer registro
73         // retornar resultados
74         return $view_cliente;
75     }
76
```

#### Flujo del método

1. Se construye la sentencia SQL para obtener la información de la reserva de vuelo asociada al cliente específico, realizando INNER JOIN con las tablas "cliente", "vuelo", "aerolinea" y dos veces con "ciudades" para obtener la información completa de la reserva.
2. Se prepara la sentencia SQL utilizando prepare() para evitar posibles ataques de inyección de SQL.
3. Se crea un array llamado \$data, que contiene el valor del ID del cliente para filtrar las reservas de vuelos asociadas a ese cliente.
4. Se ejecuta la sentencia SQL con el ID del cliente utilizando execute(\$data).
5. Se recupera la información de la reserva de vuelo encontrada utilizando fetch(PDO::FETCH\_OBJ), lo que devuelve un objeto con la información de la reserva asociada al cliente.
6. Se retorna el objeto con la información de la reserva de vuelo encontrada.

### Método selectOnePaquete(\$valoridc)

En la clase ReservasModel se utiliza para obtener la información de una reserva de paquete turístico específica asociada a un cliente dado. Toma el valor del cliente\_id como parámetro y devuelve un objeto con la información de la reserva de paquete turístico encontrada.

#### Flujo del método

1. Se construye la sentencia SQL para obtener la información de la reserva de paquete turístico asociada al cliente específico, realizando INNER JOIN con las tablas "cliente", "paquetes", "vuelo", "hotel", "aerolinea" y dos veces con "ciudades" para obtener la información completa de la reserva.
2. Se prepara la sentencia SQL utilizando prepare() para evitar posibles ataques de inyección de SQL.
3. Se crea un array llamado \$data, que contiene el valor del ID del cliente para filtrar las reservas de paquetes turísticos asociadas a ese cliente.

4. Se ejecuta la sentencia SQL con el ID del cliente utilizando `execute($data)`.
5. Se recupera la información de la reserva de paquete turístico encontrada utilizando `fetch(PDO::FETCH_OBJ)`, lo que devuelve un objeto con la información de la reserva asociada al cliente.
6. Se retorna el objeto con la información de la reserva de paquete turístico encontrada.

```
public function selectOnePaquete($valoridc) {

    $sql = "SELECT *, origenes.nombreCiudad AS vuelo_o, destinos.nombreCiudad AS Vuelo_d,
aerolinea.nombre AS nombre_A, cliente.nombre AS nombre_C, hotel.direccion AS ubicacion
FROM reservas_paquete
INNER JOIN cliente ON cliente.cliente_id = reservas_paquete.clientefk
INNER JOIN paquetes On paquetes.paquete_id = reservas_paquete.paquetefk
INNER JOIN vuelo ON vuelo.vuelo_id = paquetes.pvuelo_fk
INNER JOIN hotel ON hotel.hotel_id = paquetes.photel_fk
INNER JOIN aerolinea ON aerolinea.aerolinea_id = vuelo.aerolinea_fk
INNER JOIN ciudades AS origenes ON origenes.ciudades_id = vuelo.origen
INNER JOIN ciudades AS destinos ON destinos.ciudades_id = vuelo.destino
WHERE cliente_id = :valoridc";

    // preparar la sentencia
    $stmt = $this->con->prepare($sql);
    $data = ['valoridc' => $valoridc];
    // ejecutar la sentencia
    $stmt->execute($data);
    // recuperar los datos (en caso de select)
    $view_cliente = $stmt->fetch(PDO::FETCH_OBJ);
    /* $user = $stmt->fetchAll(PDO::FETCH_ASSOC); */// fetch retorna el primer registro
    // retornar resultados
    return $view_cliente;
}
```

### Método `get_reservas_vuelo()`

En la clase `ReservasModel` se utiliza para obtener una lista de reservas de vuelos junto con la información adicional relacionada, como el nombre de la ciudad de origen, el nombre de la ciudad de destino y el nombre del cliente asociado a cada reserva.

```
public function get_reservas_vuelo(){
    $sql ="SELECT *, origenes.nombreCiudad AS vOrigen, destinos.nombreCiudad AS vDestino,
    Clientes.nombre AS nCliente
    FROM `reservas_vuelo`
    INNER JOIN vuelo ON vuelo.vuelo_id = reservas_vuelo.vuelo_fk
    INNER JOIN cliente AS clientes ON clientes.cliente_id = reservas_vuelo.cliente_FK
    INNER JOIN aerolinea ON aerolinea.aerolinea_id = vuelo.aerolinea_fk
    INNER JOIN ciudades AS origenes ON origenes.ciudades_id = vuelo.origen
    INNER JOIN ciudades AS destinos ON destinos.ciudades_id = vuelo.destino";
    // preparar la sentencia
    $stmt = $this->con->prepare($sql);

    // ejecutar la sentencia
    $stmt->execute();
    //recuperar resultados
    $resultado = $stmt->fetchAll(PDO::FETCH_ASSOC);
    //retornar resultados
    return $resultado;
}
```

### Flujo del método

1. Se construye la sentencia SQL para obtener la información de la reserva de paquete turístico asociada al cliente específico, realizando INNER JOIN con las tablas "cliente", "paquetes", "vuelo", "hotel", "aerolinea" y dos veces con "ciudades" para obtener la información completa de la reserva.
2. Se prepara la sentencia SQL utilizando `prepare()` para evitar posibles ataques de inyección de SQL.
3. Se crea un array llamado `$data`, que contiene el valor del ID del cliente para filtrar las reservas de paquetes turísticos asociadas a ese cliente.

4. Se ejecuta la sentencia SQL con el ID del cliente utilizando `execute($data)`.
5. Se recupera la información de la reserva de paquete turístico encontrada utilizando `fetch(PDO::FETCH_OBJ)`, lo que devuelve un objeto con la información de la reserva asociada al cliente.
6. Se retorna el objeto con la información de la reserva de paquete turístico encontrada.

### Método `get_reservas_paquete()`

En la clase `ReservasModel` se utiliza para obtener una lista de reservas de paquetes turísticos junto con la información adicional relacionada.

```
public function get_reservas_paquete(){
    $sql="SELECT *, origenes.nombreCiudad AS vOrigen, destinos.nombreCiudad AS vDestino,
    Clientes.nombre AS nCliente
    FROM `reservas_paquete`
    INNER JOIN paquetes ON paquetes.paquete_id = reservas_paquete.paquetefk
    INNER JOIN hotel ON hotel.hotel_id = paquetes.photel_fk
    INNER JOIN vuelo ON vuelo.vuelo_id = paquetes.pvuelo_fk
    INNER JOIN cliente AS clientes ON clientes.cliente_id = reservas_paquete.clientefk
    INNER JOIN aerolinea ON aerolinea.aerolinea_id = vuelo.aerolinea_fk

    INNER JOIN ciudades AS origenes ON origenes.ciudades_id = vuelo.origen
    INNER JOIN ciudades AS destinos ON destinos.ciudades_id = vuelo.destino";
    // preparar la sentencia
    $stmt = $this->con->prepare($sql);

    // ejecutar la sentencia
    $stmt->execute();
    //recuperar resultados
    $resultado = $stmt->fetchAll(PDO::FETCH_ASSOC);
    //retornar resultados
    return $resultado;
}
```

### Flujo del método

1. Se construye la sentencia SQL para obtener la lista de reservas de paquetes turísticos junto con la información adicional relacionada, como el nombre de la ciudad de origen, el nombre de la ciudad de destino, el nombre del cliente, el nombre de la aerolínea y la ubicación del hotel asociados a cada reserva de paquete turístico.
2. Se prepara la sentencia SQL utilizando `prepare()` para evitar posibles ataques de inyección de SQL.
3. Se ejecuta la sentencia SQL utilizando `execute()`.
4. Se recupera la lista de resultados utilizando `fetchAll(PDO::FETCH_ASSOC)`, lo que devuelve un array con las reservas de paquetes turísticos y la información adicional asociada a cada reserva.
5. Se retorna el array con la lista de reservas de paquetes turísticos y la información adicional.

### Método `get_cliente()`

En la clase `ClientesModel` se utiliza para obtener una lista de clientes junto con la información de la ciudad a la que pertenecen.

### Flujo del método

1. Se construye la sentencia SQL para obtener la lista de clientes junto con la información de la ciudad a la que pertenecen.
2. Se prepara la sentencia SQL utilizando `prepare()` para evitar posibles ataques de inyección de SQL.

3. Se ejecuta la sentencia SQL utilizando `execute()`.
4. Se recupera la lista de resultados utilizando `fetchAll(PDO::FETCH_ASSOC)`, lo que devuelve un array con los clientes y la información de la ciudad a la que pertenecen asociada a cada cliente.
5. Se retorna el array con la lista de clientes y la información de la ciudad a la que pertenecen.

```
public function get_cliente(){
    $sql = "SELECT * FROM cliente
    INNER JOIN ciudades ON ciudades.ciudades_id = cliente.Ciudad_FK ";
    // preparar la sentencia
    $stmt = $this->con->prepare($sql);

    // ejecutar la sentencia
    $stmt->execute();
    //recuperar resultados
    $resultado = $stmt->fetchAll(PDO::FETCH_ASSOC);
    //retornar resultados
    return $resultado;
}
```

### Método `get_cliente_by_cedula($cedula)`

En la clase `ClientesModel` se utiliza para obtener la información de un cliente específico basado en su número de cédula.

```
ReservasModel.php
model > ReservasModel.php
156     $resultado = $stmt->fetchAll(PDO::FETCH_ASSOC);
157     //retornar resultados
158     return $resultado;
159 }
160
161 public function get_cliente_by_cedula($cedula) {
162     $sql = "SELECT * FROM cliente
163           INNER JOIN ciudades ON ciudades.ciudades_id = cliente.Ciudad_FK
164           WHERE cliente.cedula = :cedula";
165
166     // Preparar la sentencia
167     $stmt = $this->con->prepare($sql);
168
169     // Asignar el valor de la cédula al parámetro :cedula
170     $stmt->bindParam(':cedula', $cedula);
171
172     // Ejecutar la sentencia
173     $stmt->execute();
174
175     // Recuperar el resultado
176     $resultado = $stmt->fetch(PDO::FETCH_ASSOC);
177
178     // Retornar el resultado
179     return $resultado;
180 }
181
```

### Flujo del método

1. Se construye la sentencia SQL para obtener la información del cliente específico basado en su número de cédula.
2. Se prepara la sentencia SQL utilizando `prepare()` para evitar posibles ataques de inyección de SQL.
3. Se asigna el valor de la cédula al parámetro `:cedula` utilizando `bindParam()` para evitar posibles ataques de inyección de SQL.

4. Se ejecuta la sentencia SQL utilizando `execute()`.
5. Se recupera la información del cliente utilizando `fetch(PDO::FETCH_ASSOC)`, lo que devuelve un array con la información del cliente.
6. Se retorna el array con la información del cliente encontrado o null si no se encuentra ningún cliente con la cédula especificada.

### Método `get_cliente_id_by_cedula($cedula)`

En la clase `CientesModel` se utiliza para obtener el ID de un cliente específico basado en su número de cédula.

```
model > ReservasModel.php
180     }
181
182     public function get_cliente_id_by_cedula($cedula) {
183         $sql = "SELECT cliente_id FROM cliente WHERE cedula = :cedula";
184
185         // Preparar la sentencia
186         $stmt = $this->con->prepare($sql);
187
188         // Asignar el valor de la cédula al parámetro :cedula
189         $stmt->bindParam(':cedula', $cedula);
190
191         // Ejecutar la sentencia
192         $stmt->execute();
193
194         // Recuperar el resultado
195         $resultado = $stmt->fetchColumn();
196
197         // Retornar el resultado
198         return $resultado;
199     }
200 }
```

### Flujo del método

1. Se construye la sentencia SQL para obtener el ID del cliente específico basado en su número de cédula.
2. Se prepara la sentencia SQL utilizando `prepare()` para evitar posibles ataques de inyección de SQL.
3. Se asigna el valor de la cédula al parámetro `:cedula` utilizando `bindParam()` para evitar posibles ataques de inyección de SQL.
4. Se ejecuta la sentencia SQL utilizando `execute()`.
5. Se recupera el resultado utilizando `fetchColumn()`, lo que devuelve el valor de la primera columna del primer resultado de la consulta.
6. Se retorna el ID del cliente encontrado o false si no se encuentra ningún cliente con la cédula especificada.

### Método `insert_reservaH($reservahotel)`

En la clase `ReservasModel` se utiliza para insertar una nueva reserva de hotel en la base de datos. Realiza una inserción en la tabla "reservas\_hotel" con los datos proporcionados en el objeto `$reservahotel`.

### Flujo del método

1. Se construye la sentencia SQL para insertar una nueva reserva de hotel en la tabla "reservas\_hotel" con los campos `cliente_fk` y `RHhotel_fk`.

2. Se prepara la sentencia SQL utilizando prepare() para evitar posibles ataques de inyección de SQL.
3. Se obtienen los datos necesarios del objeto \$reservahotel y se asignan a los parámetros correspondientes en la sentencia SQL.
4. Se ejecuta la sentencia SQL utilizando execute().
5. Se verifica si la inserción fue exitosa comparando el número de filas afectadas utilizando rowCount(). Si no se insertó ninguna fila, se retorna false indicando que ocurrió algún error.
6. Si todo fue exitoso y se insertó la reserva correctamente, se retorna true.

```
public function insert_reservaH($reservahotel) {
    try{
        $sql = "INSERT INTO `reservas_hotel` (`cliente_FK`,`RHhotel_fk`) VALUES(:cfk,:hfk)";

        // Preparar la sentencia
        $stmt = $this->con->prepare($sql);
        $data = [
            'cfk' => $reservahotel->getClientefk(),
            'hfk' => $reservahotel->getRHhotelFK()
        ];
        // Ejecutar la sentencia
        $stmt->execute($data);

        if ($stmt->rowCount() <= 0) { // verificar si se inserto
            //rowCount permite obtener el numero de filas afectadas
            return false;
        }
    } catch (Exception $e){
        echo $e->getMessage();
        return false;
    }
    return true;
}
```

### El método insert\_reservaV(\$reservavuelo)

En la clase ReservasModel se utiliza para insertar una nueva reserva de vuelo en la base de datos. Realiza una inserción en la tabla "reservas\_vuelo" con los datos proporcionados en el objeto \$reservavuelo

```
public function insert_reservaV($reservavuelo) {
    try{
        $sql = "INSERT INTO `reservas_vuelo` (`vuelo_fk`,`cliente_FK`)
        VALUES(:vfk,:cfk)";

        // Preparar la sentencia
        $stmt = $this->con->prepare($sql);
        $data = [
            'cfk' => $reservavuelo->getClientefk(),
            'vfk' => $reservavuelo->getVueloFk()
        ];
        // Ejecutar la sentencia
        $stmt->execute($data);

        if ($stmt->rowCount() <= 0) { // verificar si se inserto
            //rowCount permite obtener el numero de filas afectadas
            return false;
        }
    } catch (Exception $e){
        echo $e->getMessage();
        return false;
    }
    return true;
}
```

### Flujo del método

1. Se construye la sentencia SQL para insertar una nueva reserva de vuelo en la tabla "reservas\_vuelo" con los campos vuelo\_fk y cliente\_fk.
2. Se prepara la sentencia SQL utilizando prepare() para evitar posibles ataques de inyección de SQL.
3. Se obtienen los datos necesarios del objeto \$reservavuelo y se asignan a los parámetros correspondientes en la sentencia SQL.
4. Se ejecuta la sentencia SQL utilizando execute().
5. Se verifica si la inserción fue exitosa comparando el número de filas afectadas utilizando rowCount(). Si no se insertó ninguna fila, se retorna false indicando que ocurrió algún error.
6. Si todo fue exitoso y se insertó la reserva correctamente, se retorna true.

## VuelosModel.php

El archivo VueloModel.php contiene la clase VueloModel, que se encarga de realizar operaciones relacionadas con la tabla de vuelos en la base de datos.

### Método de get\_vuelo()

- Retorna un array con todos los vuelos de la base de datos. Cada elemento del array es un array asociativo con la información de un vuelo y sus detalles (aerolínea, ciudad de origen y ciudad de destino).

```

model > VueloModel.php
1
2
3
4
5 class VueloModel
6
7
8     private $con;
9
10    public function __construct()
11    { //constructor
12        $this->con = Conexion::getConexion(); //operador :: llamando a un metodo estatico
13    }
14
15
16    public function get_vuelo(){
17        $sql = "SELECT * FROM vuelo
18            INNER JOIN aerolinea ON aerolinea.aerolinea_id = vuelo.aerolinea_fk
19            INNER JOIN ciudades AS origen_ciudad ON origen_ciudad.ciudades_id = vuelo.origen
20            INNER JOIN ciudades AS destino_ciudad ON destino_ciudad.ciudades_id = vuelo.destino";
21
22        // preparar la sentencia
23        $stmt = $this->con->prepare($sql);
24
25        // ejecutar la sentencia
26        $stmt->execute();
27        //recuperar resultados
28        $resultado = $stmt->fetchAll(PDO::FETCH_ASSOC);
29        //retornar resultados
30        return $resultado;
31    }
32

```

### Función get\_aerolinea()

Se encarga de realizar una consulta SQL para obtener todos los registros de la tabla "aerolinea" de la base de datos. Luego, recupera los resultados de la consulta utilizando fetchAll(PDO::FETCH\_ASSOC) para obtener un array asociativo con la información de todas las aerolíneas.



```

32
33     public function get_aerolinea(){
34         $sql = "SELECT * FROM aerolinea";
35
36         // preparar la sentencia
37         $stmt = $this->con->prepare($sql);
38
39         // ejecutar la sentencia
40         $stmt->execute();
41         //recuperar resultados
42         $resultado = $stmt->fetchAll(PDO::FETCH_ASSOC);
43         //retornar resultados
44         return $resultado;
45     }
46
47

```

## Retorno

- La función retorna un array con todas las aerolíneas de la base de datos. Cada elemento del array es un array asociativo que contiene la información de una aerolínea, donde las claves son los nombres de las columnas en la tabla "aerolinea".
- Este método puede ser utilizado para obtener una lista de todas las aerolíneas registradas en la base de datos y luego mostrarlas en una interfaz de usuario o realizar cualquier otra operación necesaria.

## Función registrar(\$vuelo\_model)

La función registrar(\$vuelo\_model) recibe un objeto de tipo VueloModel que contiene la información del vuelo a registrar, incluyendo la aerolínea, origen, destino, precio y fecha del vuelo. Luego, realiza una consulta SQL de tipo INSERT para insertar los datos del vuelo en la tabla "vuelo" de la base de datos.

```

model > VueloModel.php
48
49     public function registrar($vuelo_model){
50
51         try{
52
53             $sql = "INSERT INTO `vuelo` (`aerolinea_fk`,`origen`,`destino`,`precio`,`fecha`)
54             VALUES (:aer,:ori,:dest,:pre,:fech)";
55
56             $sentencia = $this->con->prepare($sql);
57
58             $data = [
59                 'aer' => $vuelo_model->getaerolinea_fk(),
60                 'ori' => $vuelo_model->getorigen(),
61                 'dest' => $vuelo_model->getdestino(),
62                 'pre' => $vuelo_model->getprecio(),
63                 'fech' => $vuelo_model->getfecha(),
64             ];
65             //execute
66             $sentencia->execute($data);
67             //retornar resultados,
68
69             if ($sentencia->rowCount() <= 0) { // verificar si se inserto
70                 //rowCount permite obtener el numero de filas afectadas
71                 return false;
72             }
73         }catch(Exception $e){
74             echo $e->getMessage();
75             return false;
76         }
77         return true;
78     }
79

```

## Retorno

- La función retorna true si el registro fue exitoso, lo que significa que el vuelo se insertó correctamente en la base de datos. En caso contrario, si el registro no pudo ser insertado, la función retorna false.

- Este método puede ser utilizado para registrar un nuevo vuelo en la base de datos, tomando como entrada un objeto que contenga la información del vuelo a registrar.

### Función selectOne(\$id)

Recibe un parámetro \$id que representa el ID del vuelo que se desea obtener. Luego, realiza una consulta SQL de tipo SELECT para buscar el vuelo con el ID especificado en la tabla "vuelo" de la base de datos. La consulta incluye las tablas "aerolinea" y "ciudades" para obtener información adicional sobre la aerolínea, el origen y el destino del vuelo.

```
public function selectOne($id) {
    $sql = "SELECT * FROM vuelo
    INNER JOIN aerolinea ON aerolinea.aerolinea_id = vuelo.aerolinea_fk
    INNER JOIN ciudades AS origen_ciudad ON origen_ciudad.ciudades_id = vuelo.origen
    INNER JOIN ciudades AS destino_ciudad ON destino_ciudad.ciudades_id = vuelo.destino
    where vuelo_id=:id";
    // preparar la sentencia
    $stmt = $this->con->prepare($sql);
    $data = ['id' => $id];
    // ejecutar la sentencia
    $stmt->execute($data);
    // recuperar los datos (en caso de select)
    $user = $stmt->fetch(PDO::FETCH_OBJ);
    /* $user = $stmt->fetchAll(PDO::FETCH_ASSOC); */// fetch retorna el primer registro
    // retornar resultados
    return $user;
}
```

### Retorno

- La función retorna un objeto con los datos del vuelo si se encuentra en la base de datos. Si no existe un vuelo con el ID proporcionado, la función retorna null.
- Este método puede ser utilizado para obtener los detalles de un vuelo específico de la base de datos, tomando como entrada su ID.

### Función update\_vuelo(\$vuelo\_model)

Recibe como parámetro \$vuelo\_model, que es un objeto que contiene los datos del vuelo que se desea actualizar en la base de datos. Luego, realiza una consulta SQL de tipo UPDATE para modificar los campos del vuelo con el ID especificado en la tabla "vuelo".

### Retorno

- La función retorna true si la actualización se realizó correctamente, lo que significa que se modificó al menos una fila en la base de datos. Si la actualización no tuvo éxito (por ejemplo, si no se encontró el vuelo con el ID especificado), la función retorna false.
- Este método puede ser utilizado para actualizar los datos de un vuelo existente en la base de datos, tomando como entrada un objeto con la nueva información del vuelo.

```

model > VueloModel.php
99
100 public function update_vuelo($vuelo_model){
101     try{
102         $sql = "UPDATE `vuelo` SET aerolinea_fk=:aer, origen=:ori,
103             destino=:dest, precio=:pre, fecha=:fec
104             Where vuelo_id =:id";
105
106         $sentencia = $this->con->prepare($sql);
107
108         $data = [
109             'id' => $vuelo_model->getvuelo_id(),
110             'aer' => $vuelo_model->getaerolinea_fk(),
111             'ori' => $vuelo_model->getorigen(),
112             'dest' => $vuelo_model->getdestino(),
113             'pre' => $vuelo_model->getprecio(),
114             'fec' => $vuelo_model->getfecha(),
115         ];
116         //execute
117         $sentencia->execute($data);
118         //retornar resultados,
119
120         if ($sentencia->rowCount() <= 0) { // verificar si se inserto
121             //rowCount permite obtener el numero de filas afectadas
122             return false;
123         }
124     } catch (Exception $e){
125         echo $e->getMessage();
126         return false;
127     }
128     return true;
129 }

```

### Función delete\_vuelo(\$id)

Recibe como parámetro \$id, que es el ID del vuelo que se desea eliminar de la base de datos. Luego, realiza una consulta SQL de tipo DELETE para eliminar el registro correspondiente al vuelo con el ID especificado en la tabla "vuelo".

```

130
131 public function delete_vuelo($id){
132     //prepare
133     $sql = "DELETE FROM vuelo WHERE vuelo_id = :id";
134
135     $sentencia = $this->con->prepare($sql);
136     $data = ['id' => $id];
137     //execute
138     $sentencia->execute($data);
139     //retornar resultados,
140     if ($sentencia->rowCount() <= 0) { // verificar si se inserto
141         //rowCount permite obtener el numero de filas afectadas
142         return false;
143     }
144     return true;
145
146 }
147

```

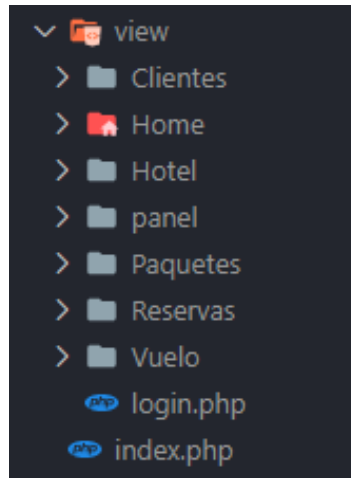
### Retorno

- La función retorna true si la eliminación se realizó correctamente, lo que significa que se eliminó al menos una fila en la base de datos. Si la eliminación no tuvo éxito (por ejemplo, si no se encontró el vuelo con el ID especificado), la función retorna false.
- Este método puede ser utilizado para eliminar un vuelo existente de la base de datos, tomando como entrada el ID del vuelo que se desea eliminar.

## VIEW

En la arquitectura MVC (Modelo-Vista-Controlador), las "views" (vistas) son la parte de la aplicación que se encarga de mostrar los datos al usuario y de interactuar con él. La vista es

responsable de la presentación visual y la interfaz con la que el usuario interactúa para ver y manipular los datos.



- **clientes:** Esta vista podría contener páginas relacionadas con la gestión de clientes, como mostrar la lista de clientes, agregar nuevos clientes, editar detalles de clientes existentes, etc.
- **home:** En esta vista, normalmente encontrarás la página principal o de inicio de la aplicación, que puede contener información general sobre la aplicación o enlaces a otras secciones importantes.
- **panel:** Esta vista podría representar el panel de control (dashboard) de la aplicación, donde se muestra una visión general de diferentes aspectos y estadísticas relevantes.
- **paquetes:** Aquí estarían las páginas relacionadas con la gestión de paquetes turísticos, como la lista de paquetes disponibles, agregar nuevos paquetes, editar paquetes, etc.
- **reservas:** En esta vista, podrías mostrar las páginas relacionadas con las reservas de hoteles, vuelos o paquetes, como mostrar las reservas existentes, agregar nuevas reservas, editar detalles de reservas, etc.
- **vuelos:** Esta vista contendría las páginas relacionadas con la gestión de vuelos, como la lista de vuelos disponibles, agregar nuevos vuelos, editar detalles de vuelos existentes, etc.
- **login.php:** Esta es la página de inicio de sesión, donde los usuarios pueden ingresar sus credenciales para acceder a la aplicación.

Cada vista generalmente contiene archivos HTML, CSS, JavaScript y PHP, y se encarga de mostrar la información requerida y responder a las interacciones del usuario. Los datos necesarios para mostrar la información generalmente se obtienen del modelo y se envían a la vista a través del controlador.

## Cientes

utiliza DataTables para crear una tabla de datos interactiva y flexible en una página web. Los usuarios pueden ordenar, buscar, paginar y exportar los datos de la tabla utilizando los controles proporcionados por DataTables

```
51 $(document).ready(function(){
52     var table = $('#example').DataTable({
53         dom: 'Bfrtip',
54         buttons: [
55             'copy', 'excel', 'pdf'
56         ],
57         "aProcessing": true, //Activamos el procesamiento del datatables
58         "aServerSide": true, //Paginación y filtrado realizados por el servidor
59
60         "ajax": {
61             url: 'index.php?c=Clientes&a=Lista_cliente',
62             type: "get",
63         },
64         "bDestroy": true,
65         "responsive": true,
66         "bInfo": true,
67         "iDisplayLength": 5, //Por cada 10 registros hace una paginación
68         "lengthMenu": [ 5, 10, 15, 20, 25 ],
69         "order": [ [ 0, "asc" ] ], //Ordenar (columna,orden)
70         "language": {
71             "sProcessing": "Procesando...",
72             "sLengthMenu": "Mostrar _MENU_ registros",
73             "sZeroRecords": "No se encontraron resultados",
74             "sEmptyTable": "Ningún dato disponible en esta tabla",
75             "sInfo": "Mostrando un total de _TOTAL_ registros",
76             "sInfoEmpty": "Mostrando un total de 0 registros",
77             "sInfoFiltered": "(filtrado de un total de _MAX_ registros)",
78             "sInfoPostFix": "",
79             "sSearch": "Buscar:",
80             "sUrl": "",
81             "sInfoThousands": ",",
82             "sLoadingRecords": "Cargando...",
83             "oPaginate": {
84                 "sFirst": "Primero",
85                 "sLast": "Último",
86                 "sNext": "Siguiente",
87                 "sPrevious": "Anterior"
88             },
89             "oAria": {
90                 "sSortAscending": ": Activar para ordenar la columna de manera ascendente",
91                 "sSortDescending": ": Activar para ordenar la columna de manera descendente"
92             }
93         },
94     });
95 }
96 );
97 </script>
```

Aquí tienes una descripción general de lo que hace este script:

1. **\$(document).ready(function(){ ... });**: Encierra el código JavaScript en una función que se ejecutará una vez que el documento HTML esté completamente cargado.
2. **var table = \$('#example').DataTable({ ... });**: Inicializa una instancia de DataTables en el elemento HTML con el ID "example". Esto crea una tabla interactiva y aplica configuraciones específicas.
3. **dom: 'Bfrtip'**: Define la estructura de los controles de la tabla. En este caso, se utilizan botones para copiar, exportar a Excel y exportar a PDF.

4. **"ajax": { ... },**: Configura la carga de datos de forma asíncrona utilizando Ajax. La URL especificada en "url" (en este caso, 'index.php?c=Clientes&a=Lista\_cliente') se utilizará para obtener los datos de la tabla.
5. Diversas configuraciones como paginación, ordenamiento, búsqueda y visualización de registros por página están definidas en esta sección para personalizar el comportamiento y apariencia de la tabla DataTables.
6. **"language": { ... },**: Define el texto y mensajes que se muestran en la interfaz de la tabla, como textos de paginación, búsqueda y mensajes de información.

## Home

Esta compuesto de código HTML que organiza los elementos de la pagina principal .

```

view > Home > Home.php
1 <?php require_once('view/panel/header.php'); >
2
3 <main id="main-container">
4     <!-- Hero -->
5     <div class="bg-image bg-image-bottom" style="background-image: url('public/assets/img/hero_panel.jpg');">
6         <div class="bg-primary-dark-op">
7             <div class="content content-top text-center overflow-hidden">
8                 <div class="pt-50 pb-20">
9                     <h1 class="font-w700 text-white mb-10 invisible" data-toggle="appear" data-class="animated fadeInUp">Panel de control</h1>
10                    <h2 class="h4 font-w400 text-white-op invisible" data-toggle="appear" data-class="animated fadeInUp">¡Bienvenido a tu panel personalizado!</h2>
11                </div>
12            </div>
13        </div>
14    </div>
15    <!-- END Hero -->
16
17    <!-- Page Content -->
18    <div class="content">
19        <div class="row invisible" data-toggle="appear">
20            <!-- Row #1 -->
21            <div class="col-6 col-xl-3">
22                <a class="block block-link-pop text-right bg-primary" href="index.php?c=Reservas&a=opcion_reservas">
23                    <div class="block-content block-content-full clearfix border-black-op-b border-3x">
24                        <div class="float-left mt-10 d-none d-sm-block">
25                            <i class="si si-bar-chart fa-3x text-primary-light"></i>
26                        </div>
27                        <!-- <div class="font-size-h3 font-w600 text-white" data-toggle="countTo" data-speed="1000" data-to="8900">0</div> -->
28                        <div class="font-size-sm font-w600 text-uppercase text-white-op">Nueva reserva</div>
29                    </div>
30                </a>
31            </div>
32            <div class="col-6 col-xl-3">
33                <a class="block block-link-pop text-right bg-earth" href="index.php?c=Hotel&a=new_hotels">
34                    <div class="block-content block-content-full clearfix border-black-op-b border-3x">
35                        <div class="float-left mt-10 d-none d-sm-block">
36                            <i class="si si-trophy fa-3x text-earth-light"></i>
37                        </div>
38                        <div class="font-size-sm font-w600 text-uppercase text-white-op">Agregar Hotel</div>
39                    </div>
40                </a>
41            </div>
42            <div class="col-6 col-xl-3">
43                <a class="block block-link-pop text-right bg-elegance" href="index.php?c=Vuelo&a=new_vuelos">
44                    <div class="block-content block-content-full clearfix border-black-op-b border-3x">
45                        <div class="float-left mt-10 d-none d-sm-block">
46                            <i class="si si-envelope-letter fa-3x text-elegance-light"></i>
47                        </div>

```

## Hotel

crea una tabla interactiva utilizando DataTables y agrega una funcionalidad de eliminación de registros con confirmación utilizando SweetAlert. Cuando un usuario elimina un registro, se realiza una solicitud AJAX para eliminarlo y se actualiza la tabla sin recargar la página.

```
Hotel_list.php X
view > Hotel > Hotel_list.php
46
47
48
49 <script>
50 $(document).ready(function(){
51
52     var table = $('#S_side').DataTable({
53         "aProcessing": true, //Activamos el procesamiento del datatables
54         "aServerSide": true, //Paginación y filtrado realizados por el servidor
55
56         "ajax": {
57             url: 'index.php?c=Hotel&a=Lista_hoteles',
58             type: "get",
59         },
60         "bDestroy": true,
61         "responsive": true,
62         "bInfo": true,
63         "iDisplayLength": 5, //Por cada 10 registros hace una paginación
64         "lengthMenu": [ 5, 10, 15, 20, 25 ],
65         "order": [ [ 0, "asc" ] ], //Ordenar (columna,orden)
66         "language": {
67             "sProcessing": "Procesando...",
68             "sLengthMenu": "Mostrar _MENU_ registros",
69             "sZeroRecords": "No se encontraron resultados",
70             "sEmptyTable": "Ningún dato disponible en esta tabla",
71             "sInfo": "Mostrando un total de _TOTAL_ registros",
72             "sInfoEmpty": "Mostrando un total de 0 registros",
73             "sInfoFiltered": "(filtrado de un total de _MAX_ registros)",
74             "sInfoPostFix": "",
75             "sSearch": "Buscar:",
76             "sUrl": "",
77             "sInfoThousands": ",",
78             "sLoadingRecords": "Cargando...",
79             "oPaginate": {
80                 "sFirst": "Primero",
81                 "sLast": "Último",
82                 "sNext": "Siguiente",
83                 "sPrevious": "Anterior"
84             },
85             "oAria": {
86                 "sSortAscending": ": Activar para ordenar la columna de manera ascendente",
87                 "sSortDescending": ": Activar para ordenar la columna de manera descendente"
88             }
89         },
90
91     });
92
93 }
```

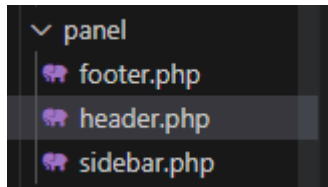
1. **`$(document).ready(function(){ ... });`**: Encierra el código JavaScript en una función que se ejecutará una vez que el documento HTML esté completamente cargado.
2. **`var table = $('#S_side').DataTable({ ... });`**: Inicializa una instancia de DataTables en el elemento HTML con el ID "S\_side". Esto crea una tabla interactiva y aplica configuraciones específicas.
3. La sección de configuración es similar a la que explicaste anteriormente. Define opciones para paginación, ordenamiento, búsqueda y visualización de registros por página, junto con el idioma personalizado.
4. **`$(document).on('click', 'eliminar', function(e) { ... });`**: Define un evento de clic para los elementos con la clase "eliminar". Esto se aplica a los botones de eliminación en la tabla.
  - Dentro de la función de clic:

- Se evita el comportamiento predeterminado del enlace.
  - Se obtiene el ID del registro a eliminar y la URL del enlace.
  - Se muestra un cuadro de diálogo de confirmación utilizando SweetAlert, pidiendo al usuario que confirme la eliminación.
  - Si el usuario confirma, se realiza una solicitud AJAX para eliminar el registro utilizando la URL 'index.php?c=Hotel&a=Eliminar\_hotel'.
  - Si la eliminación es exitosa, se muestra un mensaje de éxito utilizando SweetAlert y se actualiza la tabla DataTables sin recargar la página.
  - Si la eliminación no tiene éxito, se muestra un mensaje de error.
5. Al final del script, el evento "click" está dentro de la función **\$(document).ready()**, lo que asegura que se active una vez que la página se haya cargado completamente.



## Panel

Componente grafico usado por todo el sistema que contempla en su estructura los componentes del footer , header y sidebar de toda la pagina web desarrollados en HTML.



```
footer.php X
view > panel > footer.php
1  <!-- Footer -->
2  <footer id="page-footer" class="opacity-0">
3      <div class="content py-20 font-size-xs clearfix">
4
5
6      </div>
7  </footer>
8  <!-- END Footer -->
9  </div>
10 <!-- END Page Container -->
11
12 <!-- Codebase Core JS -->
13 <script src="public/assets/js/core/jquery.min.js"></script>
14 <script src="public/assets/js/core/popper.min.js"></script>
15 <script src="public/assets/js/core/bootstrap.min.js"></script>
16 <script src="public/assets/js/core/jquery.slimscroll.min.js"></script>
17 <script src="public/assets/js/core/jquery.scrolllock.min.js"></script>
18 <script src="public/assets/js/core/jquery.appear.min.js"></script>
19 <script src="public/assets/js/core/jquery.countTo.min.js"></script>
20 <script src="public/assets/js/core/js.cookie.min.js"></script>
21 <script src="public/assets/js/codebase.js"></script>
22
23 <!-- Page JS Plugins -->
24 <!-- <script src="public/assets/js/plugins/datatables/jquery.dataTables.min.js"></script>
25 <script src="public/assets/js/plugins/datatables/dataTables.bootstrap4.min.js"></script> -->
26
27 <script src="https://cdn.datatables.net/1.13.4/js/jquery.dataTables.min.js"></script>
28 <!-- Page JS Plugins -->
29 <script src="public/assets/js/plugins/bootstrap-wizard/jquery.bootstrap.wizard.js"></script>
30 <script src="public/assets/js/plugins/jquery-validation/jquery.validate.min.js"></script>
31 <script src="public/assets/js/plugins/jquery-validation/additional-methods.min.js"></script>
32
33 <!-- Page JS Code -->
34 <!-- <script src="public/assets/js/pages/be_forms_wizard.js"></script> -->
35
36 <!-- Page JS Code -->
37 <!-- <script src="public/assets/js/pages/be_tables_datatables.js"></script> -->
38
39
40
41 </body>
42 </html>
```

```

view > panel > header.php
1
2 <?php
3     if (!isset($_SESSION)) {
4         session_start();
5     }
6
7 <?php if(isset($_SESSION['login']) && $_SESSION['login'] == true) {
8
9     $rol = (isset($_SESSION['rol_id']));
10
11 }
12
13
14 <!doctype html>
15 <html lang="en" class="no-focus">
16 <head>
17     <meta charset="utf-8">
18     <meta name="viewport" content="width=device-width, initial-scale=1.0, user-scalable=0">
19
20     <title>Travel connect </title>
21
22     <meta name="description" content="Codebase - Bootstrap 4 Admin Template & UI Framework created by pixelcave and published on Themeforest">
23     <meta name="author" content="pixelcave">
24     <meta name="robots" content="noindex, nofollow">
25
26     <!-- Open Graph Meta -->
27     <meta property="og:title" content="Codebase - Bootstrap 4 Admin Template & UI Framework">
28     <meta property="og:site_name" content="Codebase">
29     <meta property="og:description" content="Codebase - Bootstrap 4 Admin Template & UI Framework created by pixelcave and published on Themeforest">
30     <meta property="og:type" content="website">
31     <meta property="og:url" content="">
32     <meta property="og:image" content="">
33
34
35
36 <link href="DataTables/datatables.min.css" rel="stylesheet"/>
37
38 <script src="DataTables/datatables.min.js"></script>
39
40
41
42 <!-- Icons -->
43 <!-- The following icons can be replaced with your own, they are used by desktop and mobile browsers -->
44 <link rel="shortcut icon" href="public/assets/img/favicons/favicon.png">
45 <link rel="icon" type="image/png" sizes="192x192" href="public/assets/img/favicons/favicon-192x192.png">
46 <link rel="apple-touch-icon" sizes="180x180" href="public/assets/img/favicons/apple-touch-icon-180x180.png">

```

```

view > panel > sidebar.php
1 <nav id="sidebar">
2
3     <!-- Sidebar Scroll Container -->
4     <div id="sidebar-scroll">
5         <!-- Sidebar Content -->
6         <div class="sidebar-content">
7             <!-- Side Header -->
8             <div class="content-header content-header-fullrow px-15">
9                 <!-- Mini Mode -->
10                 <div class="content-header-section sidebar-mini-visible-b">
11                     <!-- Logo -->
12                     <span class="content-header-item font-w700 font-size-xl float-left animated fadeIn">
13                         <span class="text-dual-primary-dark"></span><span class="text-primary">b</span></span>
14                     <!-- END Logo -->
15                 </div>
16                 <!-- END Mini Mode -->
17
18                 <!-- Normal Mode -->
19                 <div class="content-header-section text-center align-parent sidebar-mini-hidden">
20                     <!-- Close Sidebar, Visible only on mobile screens -->
21                     <!-- Layout API: functionality initialized in Codebase() -> uiAplLayout() -->
22                     <button type="button" class="btn btn-circle btn-dual-secondary d-lg-none align-v-n" data-toggle="layout" data-action="sidebar_close">
23                         <i class="fa fa-times text-danger"></i>
24                     </button>
25                     <!-- END Close Sidebar -->
26
27                     <!-- Logo -->
28                     <div class="content-header-item">
29                         <a class="link-effect font-w700 href="index.php?c=Home&a=mostrarHome">
30                             <i class="si si-plane text-primary"></i>
31                             <span class="font-size-xl text-dual-primary-dark">Travel</span><span class="font-size-xl text-primary">connect</span>
32                         </a>
33                     </div>
34                     <!-- END Logo -->
35                 </div>
36                 <!-- END Normal Mode -->
37             </div>
38             <!-- END Side Header -->
39
40             <!-- Side User -->
41             <div class="content-side content-side-full content-side-user px-10 align-parent">
42                 <!-- Visible only in mini mode -->
43                 <div class="sidebar-mini-visible-b align-v animated fadeIn">
44                     
45                 </div>
46                 <!-- END Visible only in mini mode -->
47
48                 <!-- Visible only in normal mode -->

```

## Paquetes

En la carpeta "paquetes" podrías organizar las vistas relacionadas con la gestión de paquetes turísticos de la siguiente manera:

### Listado

Esta vista mostraría una lista de los paquetes turísticos disponibles. En esta página, los usuarios podrían ver los detalles de cada paquete y realizar acciones como editar paquetes.

```
$(document).ready(function(){
var table = $('#example').DataTable({
  "aProcessing": true, //Activamos el procesamiento del datatables
  "aServerSide": true, //Paginación y filtrado realizados por el servidor
  "ajax":{
    url: 'index.php?c=Paquetes&a=Lista_paquetes',
    type : "get",
  },
  "bDestroy": true,
  "responsive": true,
  "bInfo": true,
  "order": [[ 0, "asc" ]], //Ordenar (columna,orden)
  "language": {
    "sProcessing": "Procesando...",
    "sLengthMenu": "Mostrar _MENU_ registros",
    "sZeroRecords": "No se encontraron resultados",
    "sEmptyTable": "Ningún dato disponible en esta tabla",
    "sInfo": "Mostrando un total de _TOTAL_ registros",
    "sInfoEmpty": "Mostrando un total de 0 registros",
    "sInfoFiltered": "(filtrado de un total de _MAX_ registros)",
    "sInfoPostFix": "",
    "sSearch": "Buscar:",
    "sUrl": "",
    "sInfoThousands": ",",
    "sLoadingRecords": "Cargando...",
    "oPaginate": {
      "sFirst": "Primero",
      "sLast": "Último",
      "sNext": "Siguiente",
      "sPrevious": "Anterior"
    },
    "oAria": {
```

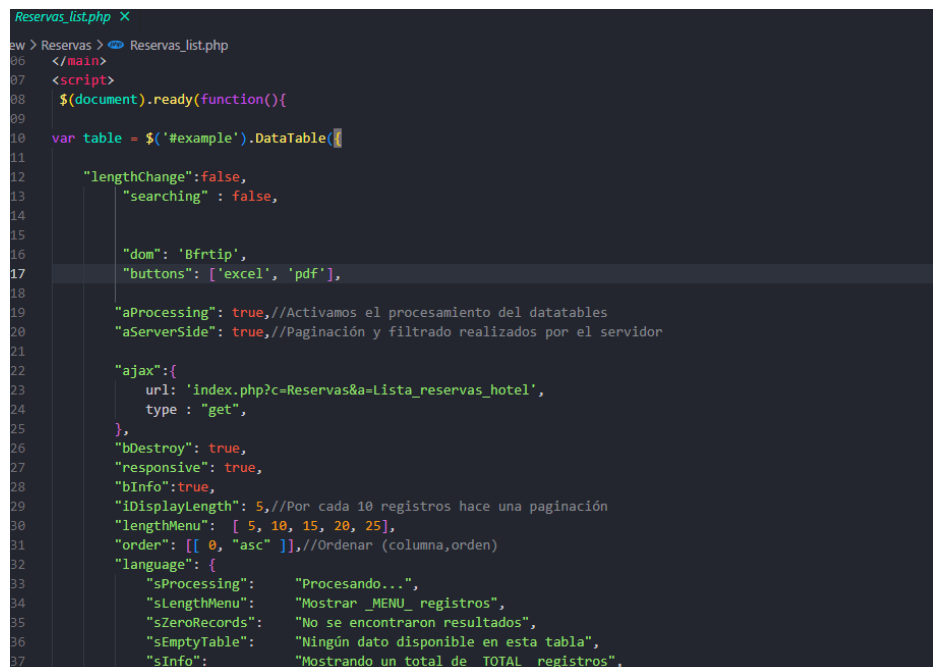
- Configuración del DataTables: El objeto DataTable se inicializa con varias opciones para personalizar su comportamiento. Algunas de las opciones utilizadas incluyen:
- aProcessing y aServerSide: Estas opciones activan el procesamiento y paginación realizados por el servidor, lo que significa que los datos se obtienen a través de una solicitud Ajax al servidor.
- ajax: Aquí se especifica la URL del servidor que se utilizará para obtener los datos de los paquetes turísticos. Esto se realiza mediante la solicitud HTTP GET al controlador, usando los parámetros "c" y "a" para identificar el controlador y el método de acción respectivamente (ejemplo: index.php?c=Paquetes&a=Lista\_paquetes).
- bDestroy: Esta opción permite destruir y recrear la tabla cada vez que se actualiza, lo cual es útil cuando los datos cambian dinámicamente.
- responsive: Esta opción permite que la tabla sea responsiva, es decir, se adapte al tamaño de la pantalla en dispositivos móviles.
- bInfo: Esta opción muestra información sobre la paginación y el número total de registros.
- iDisplayLength y lengthMenu: Estas opciones controlan la cantidad de registros mostrados por página y las opciones de longitud disponibles para el usuario.
- order: Aquí se especifica la columna y el tipo de ordenación inicial de la tabla (ejemplo: orden ascendente en la columna 0).

- **language:** Esta opción permite personalizar el texto y los mensajes que se muestran en la tabla.
- **Document Ready:** Todo el código está envuelto en la función `$(document).ready(function() { ... });`, que asegura que el código se ejecute una vez que la página ha terminado de cargarse.

En general, este código configura DataTables para que haga una solicitud Ajax al controlador para obtener los datos de los paquetes turísticos y los muestre en una tabla con paginación, búsqueda y ordenación. Esto permite que la vista "listado" de paquetes turísticos sea más interactiva y fácil de usar para los usuarios.

## Reservas

El código que proporcionaste configura el DataTables para mostrar la lista de reservas de hotel en la vista "listado" de reservas de hotel. Explicaré algunas de las opciones adicionales utilizadas en esta configuración:



```

Reservas_list.php X
1  </main>
2  <script>
3    $(document).ready(function(){
4
5      var table = $('#example').DataTable({
6
7        "lengthChange":false,
8        "searching" : false,
9
10       "dom": 'Bfrtip',
11       "buttons": ['excel', 'pdf'],
12
13       "aProcessing": true,//Activamos el procesamiento del datatables
14       "aServerSide": true,//Paginación y filtrado realizados por el servidor
15
16       "ajax":{
17         url: 'index.php?c=Reservas&a=Lista_reservas_hotel',
18         type : "get",
19       },
20       "bDestroy": true,
21       "responsive": true,
22       "bInfo":true,
23       "iDisplayLength": 5,//Por cada 10 registros hace una paginación
24       "lengthMenu": [ 5, 10, 15, 20, 25],
25       "order": [[ 0, "asc" ]],//Ordenar (columna,orden)
26       "language": {
27         "sProcessing": "Procesando...",
28         "sLengthMenu": "Mostrar _MENU_ registros",
29         "sZeroRecords": "No se encontraron resultados",
30         "sEmptyTable": "Ningún dato disponible en esta tabla",
31         "sInfo": "Mostrando un total de _TOTAL_ registros",
32       }
33     });
34   });
35 </script>
36
37

```

- **lengthChange:** Esta opción desactiva la selección del número de registros mostrados por página por parte del usuario, lo que significa que no se mostrará el cuadro para seleccionar el número de registros por página.
- **searching:** Esta opción desactiva la funcionalidad de búsqueda en la tabla, lo que significa que no se mostrará el cuadro de búsqueda.
- **dom y buttons:** Estas opciones se utilizan para agregar botones de exportación a la tabla. En este caso, se agregan los botones "Excel" y "PDF" para permitir a los usuarios descargar los datos de la tabla en formato Excel y PDF, respectivamente.
- **ajax:** Esta opción especifica la URL del servidor que se utilizará para obtener los datos de las reservas de hotel. Al igual que en el ejemplo anterior, se utiliza una solicitud Ajax al controlador para obtener los datos.
- El resto de las opciones son similares al ejemplo anterior y se utilizan para controlar el comportamiento y la apariencia de la tabla, como la paginación, el número de registros mostrados por página, el orden inicial y los mensajes de idioma.

## Vuelo

Este código configura el DataTables para mostrar la lista de aerolíneas en la vista "Lista Aerolínea". Vamos a explicar algunas de las opciones utilizadas en esta configuración:

```
aw > Vuelo > aerolinea_list.php
33 </main>
34 <script>
35 $(document).ready(function() {
36
37     var table = $('#example').DataTable({
38         "aProcessing": true, //Activamos el procesamiento del datatables
39         "aServerSide": true, //Paginación y filtrado realizados por el servidor
40
41         "ajax": {
42             url: 'index.php?c=Vuelo&a=Lista_aerolinea',
43             type: "get",
44         },
45         "bDestroy": true,
46         "responsive": true,
47         "bInfo": true,
48         "iDisplayLength": 5, //Por cada 10 registros hace una paginación
49         "lengthMenu": [ 5, 10, 15, 20, 25 ],
50         "order": [[ 0, "asc" ]], //Ordenar (columna,orden)
51         "language": {
52             "sProcessing": "Procesando...",
53             "sLengthMenu": "Mostrar _MENU_ registros",
54             "sZeroRecords": "No se encontraron resultados",
55             "sEmptyTable": "Ningún dato disponible en esta tabla",
56             "sInfo": "Mostrando un total de _TOTAL_ registros",
57             "sInfoEmpty": "Mostrando un total de 0 registros",
58             "sInfoFiltered": "(filtrado de un total de _MAX_ registros)",
59             "sInfoPostFix": "",
60             "sSearch": "Buscar:",
61             "sUrl": "",
62             "sInfoThousands": ",",
63             "sLoadingRecords": "Cargando...",
64             "oPaginate": {
```

- **ajax:** La opción "ajax" se utiliza para cargar los datos de la tabla mediante una solicitud Ajax al servidor. La URL especificada es "index.php?c=Vuelo&a=Lista\_aerolinea", lo que indica que se debe enviar una solicitud GET al controlador "Vuelo" con la acción "Lista\_aerolinea" para obtener los datos de la lista de aerolíneas.
- **bDestroy:** La opción "bDestroy" se establece en "true" para destruir y reinstanciar la tabla cada vez que se realiza una nueva solicitud Ajax, asegurándose de que los datos se actualicen correctamente.
- **responsive:** La opción "responsive" se establece en "true" para habilitar el diseño sensible en la tabla, lo que permitirá que la tabla se adapte a diferentes tamaños de pantalla.
- **bInfo:** La opción "bInfo" se establece en "true" para mostrar la información sobre el número total de registros y el número de registros mostrados en la tabla.
- **iDisplayLength y lengthMenu:** Estas opciones se utilizan para controlar la paginación y el número de registros mostrados por página. En este caso, se muestran 5 registros por página y se proporciona un menú para que el usuario pueda seleccionar diferentes cantidades de registros por página.
- **order:** La opción "order" se utiliza para establecer el orden inicial de la tabla. En este caso, se ordena por la columna 0 (primera columna) en orden ascendente.

## Login.php

Este es el código de un formulario de inicio de sesión para el controlador "Login" con la acción "Agente". Vamos a explicar el código paso a paso:

```
view > login.php
56 <h2 class="h5 font-w400 text-muted mb-0">Inicio sesión</h2>
57 </div>
58
59
60 <form class="js-validation-signin px-30" action="index.php?c=Login&a=Agente" method="post">
61 <div class="form-group row">
62 <div class="col-12">
63 <div class="form-material floating">
64 <input type="text" class="form-control" id="login-username" name="login-username">
65 <label for="login-username">Usuario</label>
66 </div>
67 </div>
68 </div>
69 <div class="form-group row">
70 <div class="col-12">
71 <div class="form-material floating">
72 <input type="password" class="form-control" id="login-password" name="login-password">
73 <label for="login-password">Contraseña</label>
74 </div>
75 </div>
76 </div>
77
78 <div class="form-group">
79 <button type="submit" class="btn btn-sm btn-hero btn-alt-primary">
80 <i class="si si-login mr-10"></i> Ingresar
81 </button>
82 </div>
83 </form>
84
85 </div>
```

1. El formulario tiene una clase "js-validation-signin", que generalmente se utiliza para habilitar la validación de JavaScript del formulario.
2. El atributo "action" del formulario está configurado para enviar la solicitud POST al controlador "Login" con la acción "Agente". Esto significa que cuando el usuario envíe el formulario, se enviarán los datos de inicio de sesión al controlador para su procesamiento.
3. Los campos de usuario y contraseña se definen con las etiquetas "input" y el atributo "type" configurado como "text" y "password", respectivamente. Estos campos recopilarán el nombre de usuario y la contraseña del usuario.
4. Los campos de usuario y contraseña tienen la clase "form-control", que es una clase de Bootstrap que proporciona estilos predefinidos para los campos de entrada.
5. Las etiquetas "label" asociadas con los campos de usuario y contraseña tienen la clase "form-material floating". Esto es una característica de Bootstrap que crea un efecto flotante para las etiquetas de los campos de entrada cuando el usuario ingresa datos.
6. El botón "Ingresar" tiene una clase "btn btn-sm btn-hero btn-alt-primary", que es una clase de Bootstrap que proporciona estilos predefinidos para los botones.
7. El botón está configurado con el atributo "type" como "submit", lo que indica que al hacer clic en el botón, se enviará el formulario.