Maximo Escobar

Carpeta de campo

Grupo Derma

Gestor de ventas

Fecha de entrega: Febrero

En la jornada de hoy, me complace presentar el emocionante inicio del proyecto denominado "Grupo Derma". En respuesta a la consigna proporcionada, me embarco en la creación de un completo gestor de ventas diseñado para satisfacer las diversas necesidades de los usuarios. Este ambicioso proyecto abarcará una amplia gama de funcionalidades, entre las cuales se destacan el control de stock, facturación, gestión de clientes, control de pedidos y seguimiento de materia prima, entre otros aspectos esenciales para una operación comercial eficiente.

La concepción de este gestor de ventas no solo se centra en la eficacia operativa, sino también en la creación de una experiencia integral para los usuarios. La implementación de un sistema de control de stock permitirá una gestión precisa y oportuna de los productos disponibles, asegurando la disponibilidad de productos en todo momento y evitando pérdidas innecesarias.

En el ámbito de la facturación, se busca diseñar un sistema que simplifique y agilice el proceso de emisión de facturas, brindando a los usuarios una herramienta intuitiva y eficaz para gestionar las transacciones comerciales de manera efectiva.

La gestión de clientes se presenta como un pilar fundamental, con la creación de un sistema que permita registrar y mantener información detallada de los clientes, fomentando relaciones sólidas y facilitando la comunicación personalizada.

El control de pedidos se convierte en un componente crucial para optimizar la logística y garantizar la entrega puntual de productos, mientras que el seguimiento de la materia prima proporcionará una visión integral del ciclo de producción, contribuyendo así a una planificación eficiente.

Este proyecto, bajo la denominación "Grupo Derma", representa un compromiso con la excelencia y la satisfacción del usuario. A medida que avanzo en esta emocionante empresa, mi objetivo es no solo cumplir con las expectativas de la consigna, sino superarlas, creando un gestor de ventas que no solo resuelva desafíos operativos, sino que también impulse el crecimiento y éxito continuo del grupo. ¡Estoy entusiasmado por los avances futuros y por el potencial impacto positivo que este gestor de ventas puede tener en el entorno empresarial

El planteamiento inicial es como iniciar, para ello evaluaremos la situación presentada y buscaremos de iniciar con los diagramas.

Diagrama de secuencia

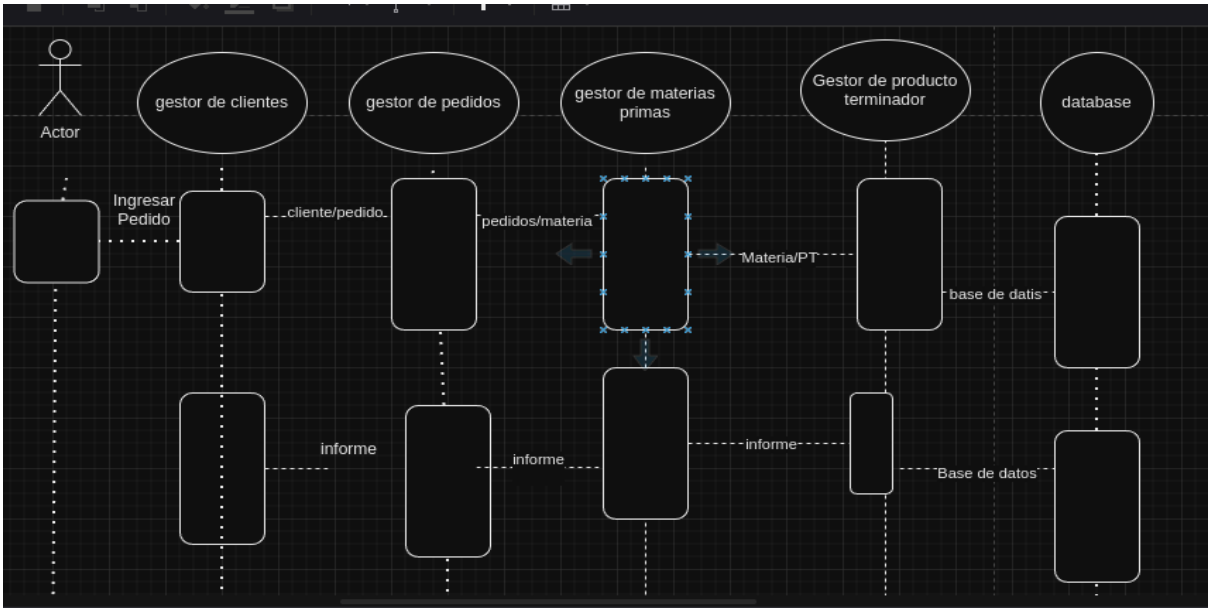


Diagrama de flujo

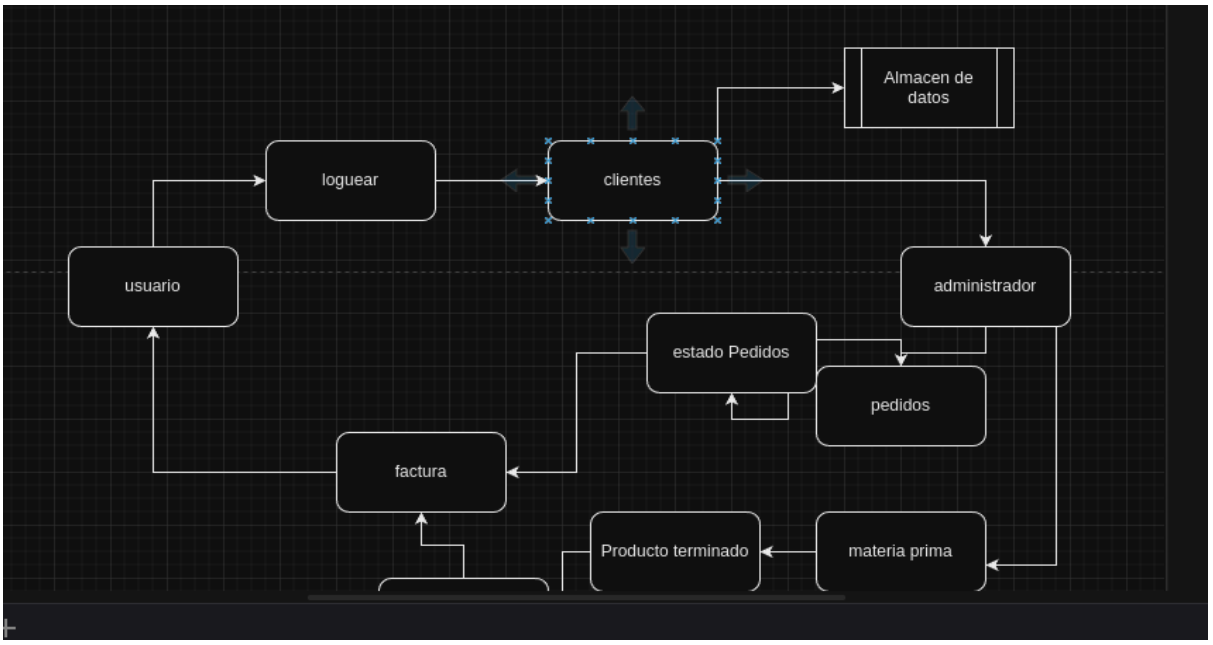
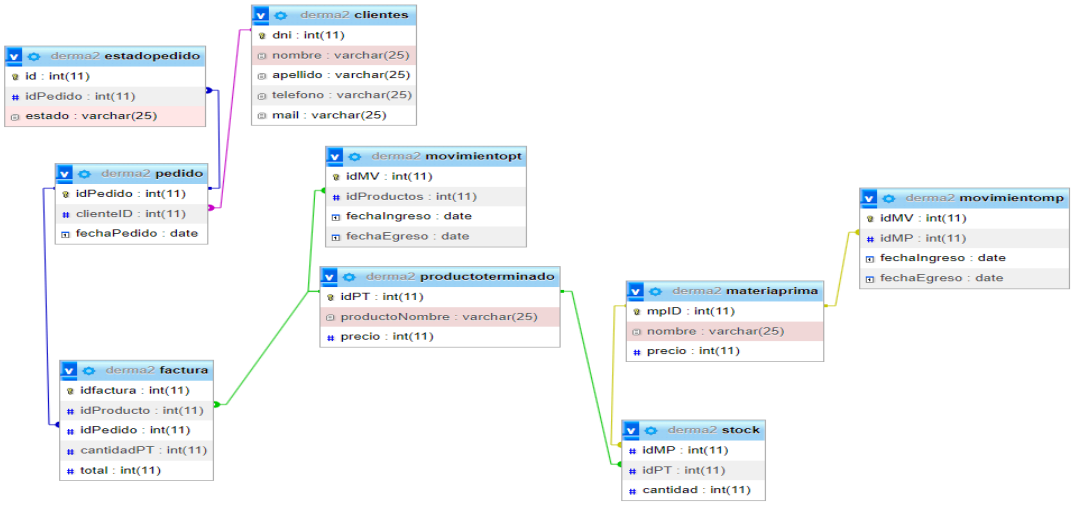


Diagrama de clases



En el día de hoy, nos disponemos a formular el principal problema que enfrentamos en nuestro proyecto. Previamente, contaba con cierto conocimiento sobre las tecnologías que íbamos a emplear, tales como PHP y HTML. En el transcurso, desarrollé un modelo MVC básico con el objetivo de obtener una comprensión inicial de la aplicación y poder continuar trabajando en ella de manera progresiva en los días subsiguientes. Este enfoque nos permitió establecer una estructura sólida y facilitó el seguimiento del progreso a medida que avanzábamos en el desarrollo.

Iniciamos el proceso de programación incorporando el framework de CakePHP, procediendo a su instalación. Sin embargo, nos encontramos con un inconveniente significativo, ya que al ejecutarlo, se presentó un error en el servidor que impidió su correcto funcionamiento. Este contratiempo generó ligeros retrasos en la fase de codificación, ya que fue necesario abordar y resolver el problema técnico antes de poder avanzar de manera efectiva en el desarrollo del proyecto.

Y que es cakephp?

CakePHP es un framework de desarrollo web de código abierto que sigue el patrón de diseño Modelo-Vista-Controlador (MVC). Fue creado para facilitar y agilizar el desarrollo de aplicaciones web, proporcionando una estructura organizada y convenciones predefinidas que permiten a los desarrolladores construir aplicaciones de manera eficiente.

Algunas características clave de CakePHP son:

Patrón MVC: Sigue el patrón de diseño Modelo-Vista-Controlador, que ayuda a separar la lógica de negocios, la presentación y la manipulación de datos en componentes distintos.

Convenciones sobre configuración: CakePHP utiliza un enfoque de "convención sobre configuración", lo que significa que siguiendo ciertas convenciones en la nomenclatura de archivos y carpetas, los desarrolladores pueden reducir la cantidad de configuración manual que necesitan realizar.

Generación de código automático: CakePHP incluye herramientas que permiten la generación automática de código CRUD (Crear, Leer, Actualizar, Eliminar) para agilizar el desarrollo de aplicaciones y reducir la cantidad de código repetitivo.

Seguridad integrada: Incorpora medidas de seguridad predefinidas para ayudar a prevenir ataques comunes, como inyecciones SQL y ataques Cross-Site Scripting (XSS).

Componentes y Helpers: CakePHP ofrece una serie de componentes y helpers que facilitan tareas comunes, como manejo de formularios, autenticación de usuarios, y manipulación de sesiones.

Manejo de bases de datos: Proporciona un ORM (Object-Relational Mapping) que simplifica la interacción con bases de datos, permitiendo a los desarrolladores trabajar con datos de manera orientada a objetos.

Extensibilidad: Es altamente extensible y permite a los desarrolladores incorporar sus propios componentes, helpers y plugins para extender la funcionalidad base del framework.

Ante esta situación, decidí explorar otras opciones y ofrecer una oportunidad a diferentes frameworks, como Symfony y Laravel. En un intento por superar los desafíos anteriores, creé estructuras de carpetas ligeramente diferentes para cada uno de ellos. No obstante, al final, ambas tentativas resultaron infructuosas, ya que

me percaté de que carecía de un profundo entendimiento de estas tecnologías específicas.

Este contratiempo me llevó a reflexionar sobre la importancia de adquirir un conocimiento más exhaustivo de los frameworks seleccionados antes de implementarlos en el proyecto. En consecuencia, he tomado la decisión de dedicar tiempo adicional al estudio y comprensión detallada de Symfony y Laravel, con el propósito de superar las barreras técnicas que han surgido hasta el momento. Este enfoque más profundo no solo busca solventar los problemas actuales, sino también garantizar una implementación más efectiva y eficiente en el futuro del desarrollo de la aplicación.

En el transcurso de la jornada actual, logré conseguir que el servidor de CakePHP iniciara sin contratiempos, aunque me encontré con un inconveniente al intentar ejecutarlo en el localhost, ya que se presentaron algunas dependencias no instaladas. Para abordar este problema, fue necesario instalar el xml-php en mi sistema. Posteriormente, ejecuté el composer como administrador para instalar las dependencias faltantes en una carpeta específica, permitiendo así resolver las complicaciones surgidas en la configuración del entorno.

Después de completar la instalación, intenté escribir código utilizando el entorno visual. Sin embargo, al reiniciar el servidor de CakePHP, experimenté dificultades para que reconociera los cambios realizados. Ante esta situación, me vi obligado a posponer el desarrollo y buscar soluciones consultando diversos videos

de instalación. Esta investigación se llevó a cabo con el objetivo de obtener perspectivas y orientación sobre cómo superar los obstáculos técnicos y asegurar un funcionamiento fluido y consistente del entorno de desarrollo.

Este proceso de aprendizaje continuo y resolución de problemas demuestra el compromiso persistente para comprender y dominar cada aspecto del entorno de desarrollo, garantizando así un progreso significativo en el proyecto. La exploración de recursos adicionales, como videos instructivos, refuerza mi determinación de abordar los desafíos de manera integral y adquirir un conocimiento más profundo para optimizar el desarrollo de la aplicación en CakePHP.

Debido a las dificultades que enfrenté al intentar hacer funcionar el framework CakePHP, tomé la decisión de prescindir de cualquier framework y optar por trabajar exclusivamente con AJAX y PHP.

El contratiempo derivado de los problemas en la implementación del framework me llevó a replantear la estructura de carpetas, ya que la gestión de archivos en el modelo MVC junto con AJAX resultaba complicada, especialmente por errores constantes en el enrutamiento que generaban retrasos significativos en los tiempos de desarrollo. En mi intento de simplificar, creé distribuciones separadas para la API y las vistas, inspirándome en una división de componentes que me resultara más cómoda. Sin embargo, los desafíos persistentes con el enrutamiento AJAX me llevaron a optar por un enfoque más básico que facilitara el enrutamiento sin generar tantos problemas.

Hoy, retomé la programación y me enfoqué en el desarrollo del lado del cliente, específicamente en la funcionalidad de añadir clientes. Para respaldar esta acción, creé una base de datos con tablas fundamentales como Clientes, Pedidos, Estado Pedidos, mP, ProductoTerminado, mvMP, MvPT, Factura, Informe Cliente y Stock. Aunque esta estructura es básica, se diseñó para simplificar la escritura del código.

La distribucion de las carpetas sera la siguiente:

Durante el desarrollo de la funcionalidad de añadir clientes y pedidos, implementamos una estructura de carpetas en la aplicación con el objetivo de organizar eficientemente el código y facilitar su mantenimiento y escalabilidad. En este esquema, se incluye la carpeta api, que sirve como el directorio principal que organiza y contiene los distintos componentes de la aplicación.

Dentro de api, se encuentra la carpeta model, la cual alberga los modelos correspondientes a cada tabla de la base de datos. Los modelos encapsulan la lógica de negocio y las interacciones con la base de datos, proporcionando una capa de abstracción que facilita la manipulación de datos.

Además, la carpeta table está diseñada para manejar la organización y manipulación de la información de cada entidad en la aplicación. Aquí, se puede encontrar la lógica específica relacionada con la recuperación y procesamiento de datos, contribuyendo así a una mayor modularidad y claridad en la estructura del código.

La carpeta controller juega un papel crucial en la organización, ya que contiene toda la lógica de programación que utiliza los modelos. Aquí se definen las rutas de la aplicación, se procesan las solicitudes del usuario y se coordina la interacción con los modelos para llevar a cabo la lógica de negocio específica de cada entidad.

Finalmente, la carpeta vistas es el lugar donde se almacena la presentación de la aplicación web. Aquí encontramos archivos que definen cómo se mostrará la información al usuario. Estos archivos de vistas son esenciales para lograr una experiencia de usuario intuitiva y amigable.

Esta estructura organizativa modular y escalonada se ha implementado con el propósito de fomentar una arquitectura clara y sostenible. Cada carpeta cumple una función específica, lo que simplifica la comprensión y modificación del código a medida que la aplicación evoluciona. En resumen, esta organización estructurada contribuye a un desarrollo más eficiente y a un mantenimiento sencillo a lo largo del ciclo de vida de la aplicación.

Desarrollamos el codigo para cada archivo, he aqui un ejemplo e los controladores

```
namespace App\Controller;
use App\Controller\AppController
class ClientesController extends AppController {
```

```
        public function index() {
            $clientes = $this->Clientes->find('all');
            $this->set(compact('clientes'));
        }
        public function add() {
            $cliente = $this->Clientes->newEmptyEntity();
            if ($this->request->is('post')) {
                $cliente =
$this->Clientes->patchEntity($cliente,
$this->request->getData());
                if ($this->Clientes->save($cliente)) {
                    $this->Flash->success(__('Cliente
guardado.'));
                    return $this->redirect(['action' =>
'index']);
                }
                $this->Flash->error(__('No se pudo guardar el
cliente. Por favor, inténtelo de nuevo.'));
            }
            $this->set(compact('cliente'));
        }
    }
```

Este código es parte de un controlador en el framework CakePHP diseñado para manejar las operaciones relacionadas con la entidad "Clientes". En resumen, el controlador ClientesController tiene dos métodos principales: index y add.

El método index se encarga de mostrar una lista de clientes obtenida a través de la consulta al modelo Clientes. Este resultado se pasa a la vista para su presentación.

El método add maneja la lógica para agregar un nuevo cliente. Primero, crea una entidad de cliente vacía y luego, si la solicitud es de tipo POST (indicando que se envió un formulario), llena la entidad con los datos proporcionados en la solicitud. Posteriormente, intenta guardar la entidad en la base de datos. Si la operación es exitosa, redirige al usuario a la lista de clientes y muestra un mensaje de éxito. En caso de un fallo, muestra un mensaje de error.

En resumen, este controlador organiza y coordina las acciones relacionadas con la entidad "Clientes", siguiendo las convenciones de CakePHP, que facilitan el desarrollo y mantenimiento del código.

Otro codigo es el de models:

```
namespace App\Model\Table;

use Cake\ORM\Table;

class ClientesTable extends Table {
```

```
        public function initialize(array $config): void {
            parent::initialize($config);

            $this->setTable('clientes'); // Nombre de la
tabla en la base de datos
            $this->setPrimaryKey('id'); // Clave primaria de
la tabla

            $this->addBehavior('Timestamp'); // Añade campos
de timestamp (created, modified)
        }
    }
```

El código del modelo ClientesTable.php en CakePHP está diseñado para definir cómo la aplicación interactúa con la tabla de la base de datos asociada a la entidad "Clientes". Aquí está una explicación detallada:

Namespace:

Explicación: El namespace (namespace App\Model\Table;) proporciona un espacio único para la clase ClientesTable. Sirve para organizar y evitar conflictos de nombres con otras clases en la aplicación.

Uso de la Clase Base Table:

Explicación: La línea use Cake\ORM\Table; importa la clase Table del ORM de CakePHP. La clase Table proporciona funcionalidades esenciales para interactuar con las tablas de la base de datos.

Declaración de la Clase:

Explicación: class ClientesTable extends Table { define la clase ClientesTable, que hereda de la clase base Table. Esta clase representa el modelo asociado a la tabla de clientes en la base de datos.

Método Initialize:

Explicación: public function initialize(array $config): void { inicia el método initialize, que se llama automáticamente al crear una instancia de la clase. Dentro de este método, se configuran propiedades esenciales del modelo.

Llamada a parent::initialize($config);:

Explicación: parent::initialize($config); llama al método initialize de la clase base Table. Esto es una buena práctica para asegurarse de que cualquier configuración predeterminada se aplique correctamente antes de agregar configuraciones adicionales.

Establecimiento de la Tabla:

Explicación: $this->setTable('clientes'); define el nombre de la tabla asociada al modelo. En este caso, se asume que la tabla se llama "clientes". Es esencial especificar el nombre de la tabla para que CakePHP pueda realizar consultas correctamente.

Definición de la Clave Primaria:

Explicación: $this->setPrimaryKey('id'); establece la clave primaria de la tabla. Aquí, se asume que la clave primaria es "id". Es crucial especificar la clave primaria para que CakePHP pueda identificar y manipular correctamente los registros en la tabla.

Añadir Comportamiento de Timestamp:

Explicación: $this->addBehavior('Timestamp'); agrega el comportamiento de timestamp. Este comportamiento automatiza el seguimiento de las fechas y horas de creación y modificación de los registros. Facilita el control de la temporalidad de los datos.

Cierre del Método initialize:

Explicación: } marca el final del método initialize y, por ende, de la clase ClientesTable.

En resumen, este código configura el modelo ClientesTable para interactuar eficientemente con la tabla de clientes en la base de datos. Define la tabla asociada, la clave primaria, y agrega un comportamiento para el seguimiento automático de las fechas de creación y modificación. Estas configuraciones son fundamentales para el correcto funcionamiento de las operaciones de la base de datos en el contexto de CakePHP.

Bootstrap para el diseño y me esforzé por sanear el código en el backend. Sin embargo, surgieron problemas en la programación, especialmente en la interacción de AJAX con el método POST para pasar y recibir parámetros. Constantemente, se presentaban errores o, en ocasiones, el servidor no recibía nada. La resolución de este inconveniente llevó algún tiempo, pero finalmente se logró un código funcional que permitió la correcta transmisión de datos a través de AJAX y el método POST.

Este proceso de ajuste y resolución de problemas refleja mi compromiso continuo con la mejora y adaptación, demostrando la habilidad para superar obstáculos y avanzar en el desarrollo de la aplicación de manera eficiente

esta para a tener las carpetas informes clientes stocks y movimientos, la carpeta clientes se encargara de los clientes, la carpeta movimientos dle control de la materia prima y prodcutos terminados y sus movimientos correspondientes, la carpeta de pedidos de adminitra los pedidos y la carpeta informes, que contendra el stock las facturas las proyecciones de ventas etc

A continuacion codigo a manera de explicación

```
function agregarCliente() {
    let datosCliente = {
        dni: $('#dni').val(),
        nombre: $('#nombre').val(),
        direccion: $('#direccion').val(),
        telefono: $('#telefono').val(),
        email: $('#email').val()
    };
```

```
    $.ajax({
        url: '../../Api/clientes/add.php',
        type: 'POST',
        data: datosCliente,
        success: function(response) {
            alert(response);

        },
        error: function(error) {
            console.log(error);
        }
    });
}
```

Este fragmento de código en JavaScript define una función llamada agregarCliente que utiliza jQuery para realizar una solicitud AJAX (Asynchronous JavaScript and XML) al servidor. La función recopila datos de diferentes campos de formulario y envía estos datos al servidor mediante una petición POST. Aquí tienes una explicación detallada:

Creación del Objeto datosCliente:

Explicación: Se crea un objeto llamado datosCliente que contiene propiedades correspondientes a los campos de un formulario (dni, nombre, direccion, telefono, email). Los valores de estas propiedades se obtienen mediante el uso de los selectores de jQuery para recuperar el contenido de los elementos de entrada con los IDs correspondientes.

Solicitud AJAX con jQuery:

Explicación: Se utiliza $.ajax() para realizar una solicitud asíncrona al servidor. Los parámetros de esta función incluyen:

url: Especifica la URL a la que se enviará la solicitud. En este caso, se espera que sea un archivo PHP ubicado en la ruta relativa '../../Api/clientes/add.php'.
type: Especifica el tipo de solicitud HTTP. Aquí se utiliza 'POST'.
data: Contiene los datos que se enviarán al servidor. En este caso, son los datos recopilados del formulario almacenados en el objeto datosCliente.
success: Define una función que se ejecutará si la solicitud tiene éxito. En este caso, simplemente muestra una alerta con la respuesta del servidor.
error: Define una función que se ejecutará si hay algún error en la solicitud. En este caso, se imprime el error en la consola del navegador.
Manejo de la Respuesta del Servidor:

Explicación: La función success maneja la respuesta del servidor mostrando una alerta con el contenido de la respuesta. Este es un enfoque básico; normalmente, se realizarían acciones más específicas según la respuesta del servidor, como actualizar la interfaz de usuario o redirigir a otra página.
Manejo de Errores:

Explicación: La función error maneja cualquier error que pueda ocurrir durante la solicitud AJAX. En este caso, simplemente imprime el error en la consola del navegador para fines de depuración.
En resumen, la función agregarCliente facilita la interacción asíncrona con el servidor para agregar clientes mediante una solicitud AJAX. Este tipo de enfoque es común en aplicaciones web modernas para mejorar la experiencia del usuario al evitar recargas de página completas.

PAra el dia de hoy seguiremos codeando la funcion pedido:

```
function agregarPedido() {
    let idProductos = $('#idProductos').val();
    let fechaPedido = $('#fechaPedido').val();


    let fechaFormateada = new Date(fechaPedido);
    let fechaMySQL = fechaFormateada.toISOString().split('T')[0];

    let datosPedido = {
        idProductos: idProductos,
        fechaPedido: fechaMySQL
    };

    $.ajax({
        url: '../../Api/pedidos/adding.php',
        type: 'POST',
        data: datosPedido,
        success: function(response) {
            alert(response);
        },
        error: function(error) {
            console.log(error);
        }
    });
}
```

Este bloque de código en JavaScript define la función agregarPedido, que utiliza jQuery para realizar una solicitud AJAX al servidor. La función recopila datos de ciertos elementos del formulario, realiza algunas manipulaciones con fechas y luego envía estos datos al servidor mediante una petición POST. Aquí tienes una explicación detallada:

Recopilación de Datos del Formulario:

Explicación: Se obtienen los valores de dos campos del formulario (idProductos y fechaPedido) utilizando los selectores de jQuery y se almacenan en las variables correspondientes (let idProductos = $('#idProductos').val(); y let fechaPedido = $('#fechaPedido').val();).

Formateo de la Fecha:

Explicación: La fecha obtenida del formulario (fechaPedido) se convierte en un objeto de fecha de JavaScript (new Date(fechaPedido)). Luego, se formatea esa fecha en un formato compatible con MySQL utilizando el método toISOString() y se elimina la parte de la hora utilizando split('T')[0]. El resultado se almacena en la variable fechaMySQL.

Creación del Objeto datosPedido:

Explicación: Se crea un objeto llamado datosPedido que contiene las propiedades idProductos y fechaPedido con los valores correspondientes recopilados del formulario.

Solicitud AJAX con jQuery:

Explicación: Se utiliza $.ajax() para realizar una solicitud asíncrona al servidor. Los parámetros de esta función incluyen:

url: Especifica la URL a la que se enviará la solicitud. En este caso, se espera que sea un archivo PHP ubicado en la ruta relativa '../../Api/pedidos/adding.php'.

type: Especifica el tipo de solicitud HTTP. Aquí se utiliza 'POST'.

data: Contiene los datos que se enviarán al servidor. En este caso, son los datos recopilados del formulario almacenados en el objeto datosPedido.

success: Define una función que se ejecutará si la solicitud tiene éxito. En este caso, simplemente muestra una alerta con la respuesta del servidor.

error: Define una función que se ejecutará si hay algún error en la solicitud AJAX. En este caso, se imprime el error en la consola del navegador.

Manejo de la Respuesta del Servidor:

Explicación: La función success maneja la respuesta del servidor mostrando una alerta con el contenido de la respuesta. Este es un enfoque básico; normalmente, se realizarían acciones más específicas según la respuesta del servidor, como actualizar la interfaz de usuario o redirigir a otra página.

Manejo de Errores:

Explicación: La función error maneja cualquier error que pueda ocurrir durante la solicitud AJAX. En este caso, simplemente imprime el error en la consola del navegador para fines de depuración.

En resumen, la función agregarPedido facilita la interacción asíncrona con el servidor para agregar pedidos mediante una solicitud AJAX. Esta estrategia es común en aplicaciones web modernas para mejorar la experiencia del usuario y evitar recargas de página completas.

Hoy me encontré con un nuevo desafío que ha estado causando demoras significativas en el avance del proyecto, específicamente en las funciones de

actualizar y mostrar clientes. La complejidad principal se presenta al mostrar datos, ya que la información está almacenada, pero no es visible para el usuario. Este obstáculo afecta considerablemente el progreso general del proyecto, ya que la capacidad de visualizar los datos es esencial para la interacción efectiva con la aplicación.

Para abordar esta situación, estoy en búsqueda de una solución viable que permita actualizar y mostrar los clientes de manera eficiente. Actualmente, estoy explorando diversas opciones, pero hasta ahora, no he encontrado una solución que satisfaga completamente los requisitos del proyecto.

El código en AJAX que se ha estado utilizando es crucial para estas funcionalidades, pero su eficacia depende de cómo se implementa y se conecta con el backend. Estoy revisando y ajustando este código para garantizar una interacción fluida entre el frontend y el backend, permitiendo así la correcta visualización y actualización de la información del cliente.

Este desafío resalta la importancia de la interfaz de usuario en la experiencia general del usuario. La capacidad de mostrar datos de manera clara y actualizada es esencial para la funcionalidad completa de la aplicación. Continuaré trabajando en encontrar una solución efectiva y garantizar un progreso constante en el proyecto.

```
$(document).ready(function() {
    $.ajax({
        url: '../../Api/informes/movimientoPT.php',
        type: 'GET',
```

```
        dataType: 'json',
        success: function(data) {

            actualizarTabla(data);
        },
        error: function(error) {
            console.log(error);
        }
    });
});


function actualizarTabla(datos) {

    var tabla = $('#miTabla tbody');


    tabla.empty();

     datos.forEach(function(fila) {

        var nuevaFila = $('<tr>');
        nuevaFila.append('<td>' + fila.idProductos + '</td>');
        nuevaFila.append('<td>' + fila.fecha_ingreso + '</td>');
        nuevaFila.append('<td>' + fila.fecha_egreso + '</td>');

        tabla.append(nuevaFila);
    });
}
```

Este bloque de código JavaScript define una función llamada actualizarTabla, que tiene como objetivo actualizar el contenido de una tabla HTML con datos proporcionados. La función asume que se le pasa un array de datos y los utiliza para construir filas de una tabla, actualizando así la presentación visual en la interfaz de usuario.

A continuación, se ofrece una explicación detallada del código:

Definición de la Función actualizarTabla:

Explicación: Esta función toma un parámetro datos, que se espera que sea un array de objetos.

Selección de la Tabla:

Explicación: Se selecciona la tabla HTML con el id miTabla y se accede a su cuerpo (tbody). Este paso es necesario para poder manipular el contenido de la tabla.

Vaciamiento de la Tabla Existente:

Explicación: Se utiliza tabla.empty(); para vaciar el contenido actual del cuerpo de la tabla. Este paso es esencial para evitar duplicaciones y garantizar que la tabla se actualice con los nuevos datos de manera limpia.

Iteración a través de los Datos:

Explicación: Se utiliza un bucle forEach para iterar sobre cada objeto en el array datos. Dentro de este bucle, se realiza la construcción de nuevas filas para la tabla.

Creación de Nuevas Filas:

Explicación: Para cada objeto en datos, se crea una nueva fila (<tr>) utilizando jQuery. Luego, se agregan celdas (<td>) a la fila con los valores correspondientes de las propiedades del objeto (por ejemplo, fila.idProductos, fila.fecha_ingreso, fila.fecha_egreso).

Inserción de la Nueva Fila en la Tabla:

Explicación: Después de construir la fila con los datos, se agrega al cuerpo de la tabla (tabla.append(nuevaFila);). Este paso se repite para cada objeto en el array datos.

En resumen, la función actualizarTabla se encarga de recibir un array de datos y actualizar dinámicamente el contenido de una tabla HTML en la interfaz de usuario. Este enfoque es común en aplicaciones web que necesitan mostrar información de manera dinámica y en tiempo real. La función utiliza jQuery para facilitar la manipulación del DOM y la construcción de nuevas filas de tabla.

Hoy, me enfrenté con la tarea de dar seguimiento al código PHP de todas las demás páginas de mi proyecto. La razón principal fue asegurarme de que cualquier solución que encuentre pueda ser implementada en todas las secciones del programa. Durante las últimas semanas, he experimentado persistentes problemas en el funcionamiento del software, lo que ha dificultado considerablemente mi progreso. Además, estuve sin computadora durante aproximadamente 20 días, lo que complicó aún más la gestión general del trabajo.

Para abordar una de las áreas específicas del programa, trabajé en un código para gestionar de manera más efectiva el stock de productos terminados. En este proceso, utilicé una analogía básica en PHP como parte de la solución. La idea detrás de esta implementación es simplificar la gestión del inventario y proporcionar una base sólida para futuras mejoras en la funcionalidad del programa. A pesar de los desafíos previos, mi objetivo es superar estos obstáculos y avanzar de manera más eficiente en el desarrollo del proyecto.

```php
<?php
error_reporting(E_ALL);
ini_set('display_errors', 1);

$servername = "localhost";
$username = "root";
$password = "";
$database = "derma2";
```

creamos la conexión a la base de datos

```php
try {
   $conn = new mysqli($servername, $username, $password, $database);

   if ($conn->connect_error) {
       throw new Exception("Error de conexión a la base de datos: " .
$conn->connect_error);
   }

   $idPT = isset($_POST['idPT']) ? $_POST['idPT'] : 0;
   $nombre = isset($_POST['productoNombre']) ? $_POST['productoNombre']
: '';


   $insertQuery = "INSERT INTO productoTerminado (idProducto, nombre)
VALUES (?, ?)";
   $insertStmt = $conn->prepare($insertQuery);
```

Este bloque de código en PHP se encarga de realizar una conexión a una base de datos MySQL utilizando MySQLi (interfaz mejorada de MySQL) y luego prepara una consulta de inserción para añadir registros a una tabla llamada productoTerminado. Aquí está una explicación detallada:

Creación de la Conexión a la Base de Datos:

Explicación: Se intenta establecer una conexión a la base de datos utilizando el constructor new mysqli(). Se pasan como parámetros el nombre del servidor ($servername), el nombre de usuario ($username), la contraseña ($password), y el nombre de la base de datos ($database).

Verificación de la Conexión:

Explicación: Se utiliza una estructura if para verificar si la conexión se estableció correctamente ($conn->connect_error). Si la conexión falla, se lanza una excepción con un mensaje de error detallado.

Inicialización de Variables:

Explicación: Se inicializan dos variables, $idPT y $nombre, con valores predeterminados. Estas variables capturan los datos recibidos a través de la solicitud POST, específicamente los valores de idPT y productoNombre. Si estos valores no están presentes, se asignan valores predeterminados de 0 y una cadena vacía, respectivamente.

Preparación de la Consulta de Inserción:

Explicación: Se construye la consulta de inserción SQL ($insertQuery) que insertará valores en la tabla productoTerminado. Luego, se prepara la consulta utilizando $conn->prepare($insertQuery). Este paso es esencial para prevenir ataques de inyección SQL y garantizar una ejecución segura de la consulta.

Este código se centra principalmente en la configuración de la conexión a la base de datos y la preparación de una consulta de inserción. Es una estructura común en PHP para realizar operaciones de base de datos, asegurándose de

manejar posibles errores y garantizar la seguridad en la manipulación de datos. La captura y manejo de excepciones son prácticas recomendadas para obtener información detallada sobre posibles problemas durante la ejecución del código.

```
if ($insertStmt === false) {
        throw new Exception("Error en la preparación de la consulta de inserción: " . $conn->error);
    }

    $insertStmt->bind_param("is", $idPT, $nombre);
    $insertStmt->execute();


    if ($insertStmt->affected_rows > 0) {
        echo json_encode(['success' => 'Producto Terminado agregado correctamente']);
    } else {
        echo json_encode(['error' => 'Error al agregar Producto Terminado']);
    }
```

Este fragmento de código en PHP se encarga de ejecutar una consulta de inserción en una base de datos MySQL utilizando MySQLi (interfaz mejorada de MySQL) y gestionar el resultado de dicha operación. A continuación, se ofrece una explicación detallada:

Verificación de la Preparación de la Consulta:

Explicación: El código comienza con una condición que verifica si la preparación de la consulta de inserción fue exitosa o no. Si la preparación ($insertStmt === false) falla, se lanza una excepción con un mensaje de error que incluye detalles específicos sobre el error.

Asociación de Parámetros y Ejecución de la Consulta de Inserción:

Explicación: Después de la preparación exitosa, se procede a asociar los parámetros a la consulta preparada mediante $insertStmt->bind_param("is", $idPT, $nombre);. Esta línea indica que se esperan dos parámetros en la consulta: un entero (i) y una cadena (s). Luego, se ejecuta la consulta de inserción mediante $insertStmt->execute();.

Verificación del Número de Filas Afectadas:

Explicación: Posteriormente, se evalúa el número de filas afectadas por la ejecución de la consulta utilizando $insertStmt->affected_rows. Si se inserta al menos una fila en la base de datos, el código envía una respuesta JSON indicando el éxito de la operación. De lo contrario, si no se afecta ninguna fila, se envía un mensaje de error.

Respuestas JSON:

Explicación: Dependiendo del resultado de la inserción, se emite una respuesta JSON al cliente. Si la inserción fue exitosa ($insertStmt->affected_rows > 0), se envía un mensaje de éxito. En caso contrario, se envía un mensaje de error indicando que la inserción del "Producto Terminado" no pudo completarse.

Este código se encarga de manejar la inserción de datos en la base de datos, asegurándose de que la consulta esté correctamente preparada, ejecutada y proporcionando información detallada sobre el resultado de la operación. La inclusión de excepciones permite un manejo de errores más robusto, y los mensajes

de respuesta JSON facilitan la comunicación entre el servidor y el cliente a través de AJAX.

```
error_reporting(E_ALL);
ini_set('display_errors', 1);
```

Esa linea de codigo permite mostrar todos los errores y acciones que corre el programa desde el incio, facilitando un poco la busqueda de errores .

el dia de hoy hice un prototipo de actualizacion para clientes

```
if (isset($_POST['dniCliente'], $_POST['nombre'], $_POST['apellido'], $_POST['telefono'], $_POST['email'])) {
   $dniCliente = $_POST['dniCliente'];
   $nombre = $_POST['nombre'];
   $apellido = $_POST['apellido'];
   $telefono = $_POST['telefono'];
   $email = $_POST['email'];

   $query = "UPDATE clientes SET nombre=?, apellido=?, telefono=?, email=? WHERE dniCliente=?";
   $stmt = mysqli_prepare($conn, $query);

   if ($stmt) {
```

Este fragmento de código en PHP verifica la existencia de ciertos datos en la solicitud POST antes de procesarlos. Si los datos necesarios están presentes, se asignan a variables correspondientes. Luego, se construye una consulta SQL de actualización para modificar un registro en la tabla de clientes en una base de datos MySQL. El código utiliza funciones de MySQLi para preparar la consulta y realizar la actualización.

A continuación, una explicación detallada:

Verificación de Datos en la Solicitud POST:

Explicación: El código verifica si ciertos campos (dniCliente, nombre, apellido, telefono, email) están presentes en la solicitud POST utilizando la función isset(). Si todos los campos están presentes, se procede a asignar los valores a variables individuales.

Asignación de Variables:

Explicación: Los valores correspondientes a dniCliente, nombre, apellido, telefono, y email se asignan a variables del mismo nombre para su posterior uso.

Construcción de la Consulta de Actualización:

Explicación: Se construye una consulta SQL de actualización que actualizará la información del cliente en la tabla de clientes. La consulta utiliza parámetros de marcadores de posición (?) para valores que se asignarán más adelante.

Preparación de la Consulta:

Explicación: Se utiliza mysqli_prepare() para preparar la consulta SQL. Este paso es crucial para prevenir ataques de inyección SQL y garantizar una ejecución segura de la consulta.

Verificación de la Preparación de la Consulta:

Explicación: Se verifica si la preparación de la consulta fue exitosa (if ($stmt)). Si es exitosa, se procede con la ejecución de la consulta; de lo contrario, se manejaría el error de manera adecuada.

Este código se encarga de validar y procesar datos provenientes de la solicitud POST, preparar una consulta SQL de actualización, y verificar la correcta preparación de la consulta antes de ejecutarla. Este tipo de estructura es común al realizar operaciones de base de datos en PHP, con el objetivo de garantizar la seguridad y la integridad de los datos manipulados.

ahora dentro del html

Anteriormente hable sobre bootstrap

```
<script src="./app.js"></script>
   <script src="https://code.jquery.com/jquery-3.6.4.min.js"></script>
   <script src="
https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.m
in.js"></script>
```

Aqui podemos observar que no solo usamos bootstrap, también bootswatch que es una herramienta basada en el código base de bootstrap para poder recrear los componentes con estilos ya definidos

En el transcurso de mi tiempo de trabajo, he enfrentado desafíos considerables derivados de las limitaciones de rendimiento de mi computadora. A pesar de estas dificultades, me he centrado en optimizar al máximo los códigos de

PHP y AJAX, buscando no solo pulir su estructura sino también abordar cualquier posible inconveniente que pudieran presentar. Esta dedicación se ha convertido en una prioridad fundamental con el objetivo de garantizar el funcionamiento más eficiente posible, incluso en condiciones adversas.

Algunos de los codigo que he diseñado hasta ahora son los siguientes:

Movimiento de materia prima

```
<?php
error_reporting(E_ALL);
ini_set('display_errors', 1);

$servername = "localhost";
$username = "root";
$password = "";
$database = "derma2";

$conn = new mysqli($servername, $username, $password, $database);

if ($conn->connect_error) {
    die("Error de conexión a la base de datos: " . $conn->connect_error);
}

$mpID = isset($_POST['mpID']) ? intval($_POST['mpID']) : 0;
```

```
    $fechaIngreso = isset($_POST['fechaIngreso']) ?
$_POST['fechaIngreso'] : '';
    $fechaEgreso = isset($_POST['fechaEgreso']) ?
$_POST['fechaEgreso'] : '';
    if (empty($fechaIngreso) || empty($fechaEgreso) ||
!validarFormatoFecha($fechaIngreso) ||
!validarFormatoFecha($fechaEgreso)) {
        echo json_encode(['error' => 'Error: Las fechas no
son válidas']);
        exit;
    }
    $query = "INSERT INTO movimientosMP (idMP, fechaIngreso,
fechaEgreso) VALUES (?, ?, ?)";
    $stmt = $conn->prepare($query);
    if ($stmt) {
        $stmt->bind_param("iss", $mpID, $fechaIngreso,
$fechaEgreso);
        $stmt->execute();

        if ($stmt->affected_rows > 0) {
            echo json_encode(['success' => 'Movimiento
agregado correctamente']);
        } else {
            echo json_encode(['error' => 'Error al agregar
movimiento: ' . $stmt->error]);
        }
```

```
        $stmt->close();
    } else {
        echo json_encode(['error' => 'Error en la preparación
de la consulta']);
    }
    mysqli_close($conn);

    function validarFormatoFecha($fecha) {
        $date = DateTime::createFromFormat('Y-m-d', $fecha);
        return $date && $date->format('Y-m-d') === $fecha;
    }
    ?>
```

Codigo de productosTerminados

```
    <?php
    error_reporting(E_ALL);
    ini_set('display_errors', 1);

    $servername = "localhost";
    $username = "root";
    $password = "";
    $database = "derma2";
    try {
        $conn = new mysqli($servername, $username, $password,
$database);
```

```
        if ($conn->connect_error) {
            throw new Exception("Error de conexión a la base
de datos: " . $conn->connect_error);
        }

        $idPT = isset($_POST['idPT']) ? $_POST['idPT'] : 0;
        $nombre = isset($_POST['productoNombre']) ?
$_POST['productoNombre'] : '';
        $insertQuery = "INSERT INTO productoTerminado
(idProducto, nombre) VALUES (?, ?)";
        $insertStmt = $conn->prepare($insertQuery);
        if ($insertStmt === false) {
            throw new Exception("Error en la preparación de
la consulta de inserción: " . $conn->error);
        }
        $insertStmt->bind_param("is", $idPT, $nombre);
        $insertStmt->execute();

        if ($insertStmt->affected_rows > 0) {
            echo json_encode(['success' => 'Producto
Terminado agregado correctamente']);
        } else {
            echo json_encode(['error' => 'Error al agregar
Producto Terminado']);
        }
        $insertStmt->close();
```

```
    } catch (Exception $e) {
        echo json_encode(['error' => 'Excepción capturada: '
. $e->getMessage()]);
    } finally {
        if (isset($conn)) {
            $conn->close();
        }
    }
    ?>
```

codigo de factura

```
<?php
error_reporting(E_ALL);
ini_set('display_errors', 1);

$servername = "localhost";
$username = "root";
$password = "";
$database = "derma2";

$conn = new mysqli($servername, $username, $password,
$database);

if ($conn->connect_error) {
```

```
    echo json_encode(['error' => 'Error de conexión a la base
de datos: ' . $conn->connect_error]);
    exit;
}

$dniCliente = isset($_POST['dniCliente']) ?
intval($_POST['dniCliente']) : 0;
$idProducto = isset($_POST['idProducto']) ?
mysqli_real_escape_string($conn, $_POST['idProducto']) : 0;
$cantidadPT = isset($_POST['cantidadPT']) ?
mysqli_real_escape_string($conn, $_POST['cantidadPT']) : 0;
$total = isset($_POST['total']) ?
mysqli_real_escape_string($conn, $_POST['total']) : 0;
$query = "INSERT INTO factura (dniCliente, idProducto,
cantidadPT,  Total) VALUES (?, ?, ?, ?)";
$stmt = $conn->prepare($query);
if ($stmt === false) {
    echo json_encode(['error' => 'Error en la preparación de
la consulta: ' . $conn->error]);
    exit;
}
$stmt->bind_param("isss", $dniCliente, $idProducto,
$cantidadPT,  $total);
$result = $stmt->execute();
if ($result) {
```

```
    echo json_encode(['success' => 'Cliente agregado
correctamente']);
} else {
    echo json_encode(['error' => 'Error al agregar cliente: '
. $stmt->error]);
}
$stmt->close();
$conn->close();
?>
```

Este script en PHP realiza la inserción de datos en una tabla llamada "factura" en una base de datos MySQL. Aquí tienes una explicación detallada del código:

Configuración de Informes de Errores:

Explicación: Se establece la configuración para informar sobre todos los errores y mostrarlos en pantalla. Esto es útil durante el desarrollo para identificar y corregir problemas.

Conexión a la Base de Datos:

Explicación: Se crea una conexión a la base de datos utilizando la clase mysqli. Se utiliza la información proporcionada como nombre del servidor ($servername), nombre de usuario ($username), contraseña ($password), y nombre de la base de datos ($database).

Verificación de la Conexión:

Explicación: Se verifica si la conexión a la base de datos fue exitosa. Si hay un error de conexión, se devuelve un mensaje de error en formato JSON y se termina la ejecución del script.

Recopilación de Datos del Formulario:

Explicación: Se obtienen los datos del formulario a través de la solicitud POST. Se valida y limpia cada variable ($dniCliente, $idProducto, $cantidadPT, $total) para evitar inyecciones SQL y asegurar que los datos sean del tipo esperado.

Preparación de la Consulta SQL:

Explicación: Se construye la consulta SQL de inserción utilizando marcadores de posición (?). Luego, se prepara la consulta mediante el método prepare().

Verificación de la Preparación de la Consulta:

Explicación: Se verifica si la preparación de la consulta fue exitosa. Si hay un error, se devuelve un mensaje de error en formato JSON y se termina la ejecución del script.

Enlace de Parámetros y Ejecución de la Consulta:

Explicación: Los parámetros son enlazados a la consulta utilizando el método bind_param(). Luego, se ejecuta la consulta con el método execute().

Verificación del Resultado de la Inserción:

Explicación: Se verifica si la inserción fue exitosa. Si es así, se devuelve un mensaje de éxito en formato JSON; de lo contrario, se devuelve un mensaje de error.

Cierre de Recursos:

Explicación: Se cierran los recursos (la consulta preparada y la conexión a la base de datos) para liberar memoria y evitar posibles problemas.

En resumen, este script PHP maneja la inserción de datos de facturación en una base de datos MySQL. Se toman precauciones para evitar errores y se proporciona retroalimentación en formato JSON sobre el resultado de la operación.

En el ultimo dia de entrega con ayuda se logró hacer funcionar el codigo de mostrar datos, lo que permite hacer un método get para traer los datos desde la base de datos y todo se debió al siguiente cambio:

```
$query = "SELECT * FROM movimientopt";
      $result = $conn->query($query);
      echo '<table class="table table-borderless">
      <thead>
          <tr>
            <th scope="col">ID</th>
            <th scope="col">Fecha Ingreso</th>
            <th scope="col">Fecha Egreso</th>
          </tr>
        </thead>
        ';
      if ($result) {
```

```
            // Construir la respuesta HTML
            $htmlResponse = '<tbody>';
            while ($row = $result->fetch_assoc()) {
                $htmlResponse .= '<tr>';
                $htmlResponse .= '<th scope="row">' .
$row['idProductos'] . '</th>';
                $htmlResponse .= '<td>' .
$row['fechaIngreso'] . '</td>';
                $htmlResponse .= '<td>' . $row['fechaEgreso']
. '</td>';
                $htmlResponse .= '</tr>';
            }
            $htmlResponse .= '</tbody>';

            // Devolver la respuesta HTML
            echo $htmlResponse;
        } else {
            echo json_encode(['error' => 'Error al obtener
los datos de la base de datos']);
        }
        echo "</table>";
        mysqli_close($conn);
        ?>
    </body>
    </html>
```

Realiza una consulta SELECT a la base de datos para obtener datos de la tabla "movimientopt". Luego, construye y devuelve una tabla HTML con los resultados obtenidos. Aquí está una explicación detallada del código:

Realización de la Consulta SELECT:

Se ejecuta una consulta SELECT para obtener todos los registros de la tabla "movimientopt". El resultado se almacena en la variable $result.

Construcción de la Estructura de la Tabla HTML:

Se comienza construyendo la estructura básica de la tabla HTML con un encabezado que incluye las columnas "ID", "Fecha Ingreso" y "Fecha Egreso".

Verificación de Resultados de la Consulta:

Se verifica si la ejecución de la consulta fue exitosa (if ($result)).

Construcción de las Filas de la Tabla:

Si hay resultados, se inicia un bucle while para recorrer cada fila del resultado (while ($row = $result->fetch_assoc())). Se construye una fila HTML para cada registro con las columnas "ID", "Fecha Ingreso" y "Fecha Egreso".

Construcción de la Respuesta HTML:

Se construye una respuesta HTML que incluye las filas de la tabla. Esta respuesta se almacena en la variable $htmlResponse.

Devolver la Respuesta HTML:

Se imprime la respuesta HTML en el punto adecuado dentro del documento, específicamente entre las etiquetas <tbody> de la tabla.

Manejo de Errores:

Si hay algún problema al obtener los datos de la base de datos, se imprime un mensaje de error JSON.

Cierre de la Conexión:

Se cierra la conexión a la base de datos utilizando mysqli_close($conn).

En resumen, este script PHP se utiliza para obtener datos de la base de datos y presentarlos en una tabla HTML. La construcción cuidadosa de la respuesta HTML asegura que la información se presente de manera estructurada y legible en el lado del cliente. La gestión de errores también está incorporada para manejar situaciones donde la consulta no es exitosa.

Algo que no se explico anteriormente es el login que el código es el siguiente:

```
session_start();

if (isset($_POST['login'])) {
```

```
        $mail = $conexion->real_escape_string($_POST['mail']);
        $contrasena =
$conexion->real_escape_string($_POST['contrasena']);
        if ($contrasena == "" || strlen($contrasena) < 4) {
            echo "La contraseña no puede estar vacía y debe tener
al menos 4 caracteres.";
        } elseif ($mail == "" || !filter_var($mail,
FILTER_VALIDATE_EMAIL)) {
            echo "Ingrese un correo electrónico válido.";
        } else {

            $sql = " SELECT * FROM signup where mail= '$mail' and
'$contrasena' ";

            $resultado = mysqli_query($conexion, $sql);



            $filas = mysqli_num_rows($resultado);

            // Verificar si el usuario existe
            if ($filas != 0) {
                header("Location: index.php");
                exit();
            } else {

                echo "Usuario inválido, verifique sus datos. Si
no tiene una cuenta, le invitamos a registrarse.";
                echo "Error: " . mysqli_error($conexion);
```

```
            }
        }
    }
    mysqli_close($conexion);
    ?>
```

Inicio de Sesión:

session_start(); inicializa o reanuda una sesión de PHP. Esto se utiliza para mantener el estado de la sesión entre las distintas páginas del sitio.

Comprobación de Datos del Formulario:

if (isset($_POST['login'])) verifica si se ha enviado el formulario de inicio de sesión. Esto significa que el código que sigue a continuación se ejecutará solo cuando se envíe el formulario.

Obtención de Datos del Formulario:

$mail y $contrasena se obtienen a través de $_POST y se limpian usando real_escape_string para prevenir ataques de inyección SQL.

Validación de Contraseña y Correo Electrónico:

Se realizan verificaciones para asegurarse de que la contraseña tenga al menos 4 caracteres y que el correo electrónico tenga un formato válido.

Consulta SQL:

Se construye una consulta SQL para verificar si existe un registro en la tabla "signup" con el correo electrónico y la contraseña proporcionados.

Ejecución de la Consulta:

$resultado = mysqli_query($conexion, $sql); ejecuta la consulta SQL en la base de datos.

Conteo de Filas Resultantes:

$filas = mysqli_num_rows($resultado); cuenta el número de filas resultantes de la consulta.

Verificación de Existencia del Usuario:

Si $filas es diferente de cero, significa que se encontró al menos un usuario que coincide con las credenciales proporcionadas. En este caso, se redirige al usuario a la página "index.php" utilizando header("Location: index.php");.

Manejo de Usuario No Válido:

Si no se encuentra ningún usuario que coincida con las credenciales, se emite un mensaje de error indicando que los datos son inválidos.

Y el codigo de registro

```
if (isset($_POST['registro'])) {
        $nombre = $_POST['nombre'];
        $apellido = $_POST['apellido'];
        $mail = $_POST['mail'];
        $contrasena = $_POST['contrasena'];
        $confcontrasena = $_POST['confcontrasena'];
```

```
        if (empty($nombre) || empty($apellido) ||
empty($mail) || empty($contrasena) || empty($confcontrasena))
{
            echo "Complete todos los campos.";
        } else {

            if ($contrasena == $confcontrasena) {
                $hashContrasena = password_hash($contrasena,
PASSWORD_DEFAULT);
                $sql = "INSERT INTO signup(nombre, apellido,
mail, contrasena) VALUES (?, ?, ?, ?)";
                $stmt = mysqli_stmt_init($conexion);
                if (mysqli_stmt_prepare($stmt, $sql)) {

                    mysqli_stmt_bind_param($stmt, 'ssss',
$nombre, $apellido, $mail, $contrasena);
                $resultado = mysqli_stmt_execute($stmt);
                    if ($resultado) {
                        echo "Registro exitoso. ¡Bienvenido,
$nombre!";
                    } else {
                        echo "Verifique los datos
insertados.<br>";
```

```
                        echo "Error: " .
mysqli_error($conexion);
                    }
                    mysqli_stmt_close($stmt);
                }
            } else {

                echo "Las contraseñas no coinciden. Por
favor, inténtelo de nuevo.";
            }
        }
    }
```

Comprobación del Envío del Formulario:

if (isset($_POST['registro'])) verifica si el formulario de registro fue enviado. Esto significa que el código siguiente se ejecutará solo cuando se envíe el formulario.

Obtención de Datos del Formulario:

Se obtienen los valores del formulario para nombre, apellido, mail, contrasena, y confcontrasena usando $_POST.

Verificación de Campos Vacíos:

Se verifica si algún campo está vacío utilizando empty(). Si algún campo está vacío, se emite un mensaje indicando que todos los campos deben completarse.

Verificación de Coincidencia de Contraseñas:

Se verifica si las contraseñas proporcionadas (contrasena y confcontrasena) coinciden. Si no coinciden, se emite un mensaje indicando que las contraseñas no coinciden y se le pide al usuario que lo intente de nuevo.

Consulta SQL Preparada:

Se utiliza una consulta SQL preparada para insertar los datos del usuario en la base de datos. Esto ayuda a prevenir ataques de inyección SQL.

Hashing de Contraseña:

La contraseña se hashea utilizando la función password_hash() antes de almacenarla en la base de datos. Esto mejora la seguridad almacenando contraseñas seguras y evita el almacenamiento directo de contraseñas en texto plano.

Inicialización de la Consulta Preparada:

mysqli_stmt_init($conexion) inicializa una consulta preparada.

Preparación de la Consulta Preparada:

mysqli_stmt_prepare($stmt, $sql) prepara la consulta preparada con la conexión a la base de datos y la consulta SQL.

Vinculación de Parámetros:

mysqli_stmt_bind_param($stmt, 'ssss', $nombre, $apellido, $mail, $hashContrasena) vincula los parámetros de la consulta preparada con los valores proporcionados por el usuario, incluyendo la contraseña hasheada.

Ejecución de la Consulta Preparada:

mysqli_stmt_execute($stmt) ejecuta la consulta preparada.

Manejo de Resultados:

Se verifica si la ejecución de la consulta preparada fue exitosa ($resultado). Se emite un mensaje de éxito si es así, de lo contrario, se imprime un mensaje de error y se proporciona información detallada sobre el error.