Desarrollo de un CRUD usando HTML + Php (mysqli) + MySQL + POO + MVC

Continuamos con la Quinta entrega de las guías sobre base de datos, php, MySQL, en esta ocasión le damos el turno a MVC(Modelo - Vista - Controlador), en esta sesión tendremos el mismo CRUD, solamente que introduciremos MVC, separándolo en tres capas por decirlo de una manera, de igual manera no podemos dejar atrás guías las cuales son [1] Como crear una base de datos en MySQL usando PhpMyAdmin, [2] Formularios HTML y procesamiento usando php, [3] - Desarrollo de un CRUD usando HTML + Php (mysqli) + MySQL), [4] - Desarrollo de un CRUD usando HTML + Php (mysqli) + MySQL + POO, ya que esta es continuación de las antes mencionadas.

Asumiendo que ya esas dos guías fueron estudiadas, comprendidas y practicadas procedemos por definir ciertos conceptos que necesitamos tener en claro, el cual es el tema que corresponde a este guía.

Nota: la siguiente estructura es una manera de muchas de desarrollar con MVC + POO, dicha estructura es parte tomada de internet, parte tomada de la ayuda de php, parte tomada de experiencia personal, por lo cual NO, es restrictivo su uso tal cual, solo se trata de plasmar una forma sencilla de entender el paradigma de la POO.

Algunos conceptos básicos para poder entender el código.

Definamos:

Modelo Vista Controlador (MVC) es un estilo de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos.

Se trata de un modelo muy maduro y que ha demostrado su validez a lo largo de los años en todo tipo de aplicaciones, y sobre multitud de lenguajes y plataformas de desarrollo.

El **Modelo** que contiene una representación de los datos que maneja el sistema, su lógica de negocio, y sus mecanismos de persistencia.

La **Vista**, o interfaz de usuario, que compone la información que se envía al cliente y los mecanismos interacción con éste.

El **Controlador**, que actúa como intermediario entre el Modelo y la Vista, gestionando el flujo de información entre ellos y las transformaciones para adaptar los datos a las necesidades de cada uno.

El modelo es el responsable de:

- Acceder a la capa de almacenamiento de datos. Lo ideal es que el modelo sea independiente del sistema de almacenamiento.
- Define las reglas de negocio (la funcionalidad del sistema). Un ejemplo de regla puede ser: "Si la mercancía pedida no está en el almacén, consultar el tiempo de entrega estándar del proveedor".
- Lleva un registro de las vistas y controladores del sistema.
- Si estamos ante un modelo activo, notificará a las vistas los cambios que en los datos pueda producir un agente externo (por ejemplo, un fichero por lotes que actualiza los datos, un temporizador que desencadena una inserción, entre otros).

El controlador es responsable de:

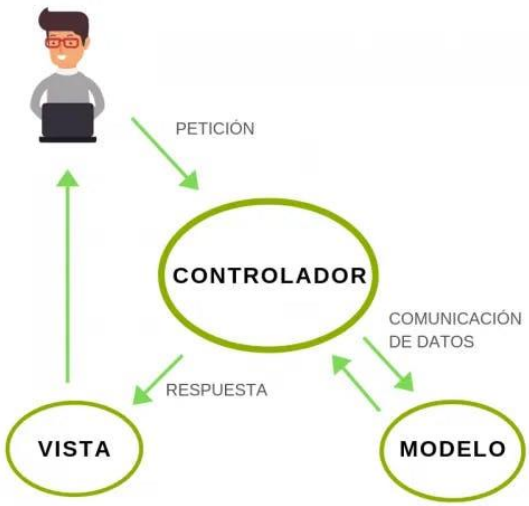
- Recibe los eventos de entrada (un clic, un cambio en un campo de texto, etc.).
- Contiene reglas de gestión de eventos, del tipo "SI Evento Z, entonces Acción W". Estas acciones pueden suponer peticiones al modelo o a las vistas. Una de estas peticiones a las vistas puede ser una llamada al

método "agregarEst(), utilizado en esta guía". Una petición al modelo puede ser "Obtener el registro cuya cedula es 17.256.256 (Consulta)".

Las vistas son responsables de:

- Recibir datos del modelo y los muestra al usuario.
- Tienen un registro de su controlador asociado (normalmente porque además lo instancia).
- Pueden dar el servicio de "Actualización()", para que sea invocado por el controlador o por el modelo (cuando es un modelo activo que informa de los cambios en los datos producidos por otros agentes).

Ejemplo de una petición de un Usuario



- El usuario realiza una petición. En nuestro caso desea modificar el nombre del estudiante.
- El controlador recibe dicha petición (de la interfaz gráfica) y procede a ejecutar la acción pertinente enviando los datos al Modelo.
- El Modelo realiza las modificaciones con la base de datos de MariaDB.
- Luego, devuelve al controlador los cambios solicitados.
- Cuando el Controlador recibe todos los datos del cambio, envía una respuesta a la Vista.
- La Vista aplica la modificación en la interfaz del navegador mostrando al usuario.

Ventajas de MVC

Implementar dicho patrón en nuestros proyectos puede beneficiarnos en los siguientes puntos:

- Organización modular. Podrás dividir la lógica del programa haciendo la aplicación más escalable.
- Puedes hacer abstracción de datos para facilitar la realización de consultas a la base de datos. Esto se da en el caso de los Framework como Ruby on Rails, ASP NET, etc.
- Código más organizado y legible. Siendo ideal si trabajas en equipo o para continuar el trabajo de otro programador.

- Este patrón se implementa hace muchos años y lo utilizan grandes empresas debido a su correcto diseño y extensibilidad.

MVC y bases de datos

Muchos sistemas [informáticos] utilizan un sistema de gestión de base de datos el cual gestiona los datos que debe utilizar la aplicación; en líneas generales del MVC dicha gestión corresponde al modelo. La unión entre capa de presentación y capa de negocio conocido en el paradigma de la Programación por capas representaría la integración entre la Vista y su correspondiente Controlador de eventos y acceso a datos, MVC no pretende discriminar entre capa de negocio y capa de presentación pero sí pretende separar la capa visual gráfica de su correspondiente programación y acceso a datos, algo que mejora el desarrollo y mantenimiento de la Vista y el Controlador en paralelo, ya que ambos cumplen ciclos de vida muy distintos entre sí.

Uso en aplicaciones Web

Aunque originalmente MVC fue desarrollado para aplicaciones de escritorio, ha sido ampliamente adaptado como arquitectura para diseñar e implementar aplicaciones web en los principales lenguajes de programación. Se han desarrollado multitud de frameworks, comerciales y no comerciales, que implementan este patrón este frameworks se diferencian básicamente en la interpretación de como las funciones MVC se dividen entre cliente y servidor.

Los primeros frameworks MVC para desarrollo web planteaban un enfoque de cliente ligero en el que casi todas las funciones, tanto de la vista, el modelo y el controlador recaían en el servidor. En este enfoque, el cliente manda la petición de cualquier hiperenlace o formulario al controlador y después recibe de la vista una página completa y actualizada (u otro documento); tanto el modelo como el controlador (y buena parte de la vista) están completamente alojados en el servidor. Como las tecnologías web han madurado, ahora existen frameworks como JavaScriptMVC, Backbone o jQuery14 que permiten que ciertos componentes MVC se ejecuten parcial o totalmente en el cliente (véase AJAX).

Referencias Bibliográficas

- https://nicobobb.com/mvc
- https://si.ua.es/es/documentacion/asp-net-mvc-3/1-dia/modelo-vista-controlador-mvc.html
- https://es.wikipedia.org/wiki/Modelo%E2%80%93vista%E2%80%93controlador

Contenido de la Carpeta guiaCrudMVC-POO

Para esta guía la estructura del CRUD, cambia drásticamente es por ello por lo que aparecen ciertas carpetas, entre ella, **Modelo**, donde almacenaremos la clase que contiene toda la lógica del negocio, **Vista**, donde encontraremos las vistas del CRUD (Listado estudiantes, Agregar Estudiante, Editar Estudiante, Confirmar la eliminación del Registro), **Controlador**, aquí encontraremos el controlador de la aplicación o CRUD, Conexión esta seguirá teniendo los archivos config.php y basedatos.php.

Tenemos la presencia de un archivo **index.php**, el cual es el archivo principal de la aplicación (Controlador principal) este se encargará según la petición del usuario (acción) decidir qué proceso, método o controlador que se ejecutará, a su vez este controlador llamado ejecutará el/los métodos de la clase.

Este archivo **index.php** se encargará de encadenar o unir cada uno de los diferentes archivos, controlador, modelo y vista, al final se obtendrá un solo archivo HTML o php que empezará por el encabezado y culminará con el pie.

La manera del llamado a este archivo es la manera convencional, para hacerlo más profesional, se puede hacer uso de los archivos .httaccess, el cual permite brindar una experiencia más cómoda para el usuario final.

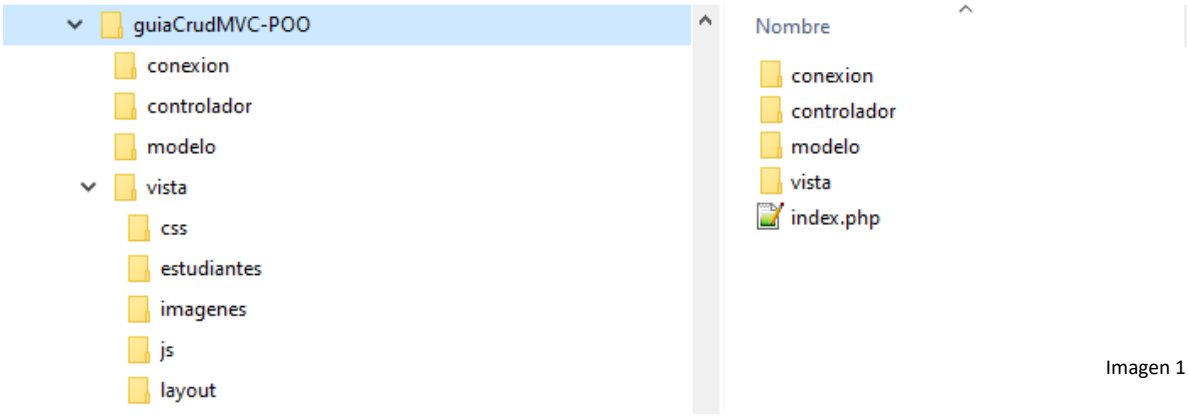
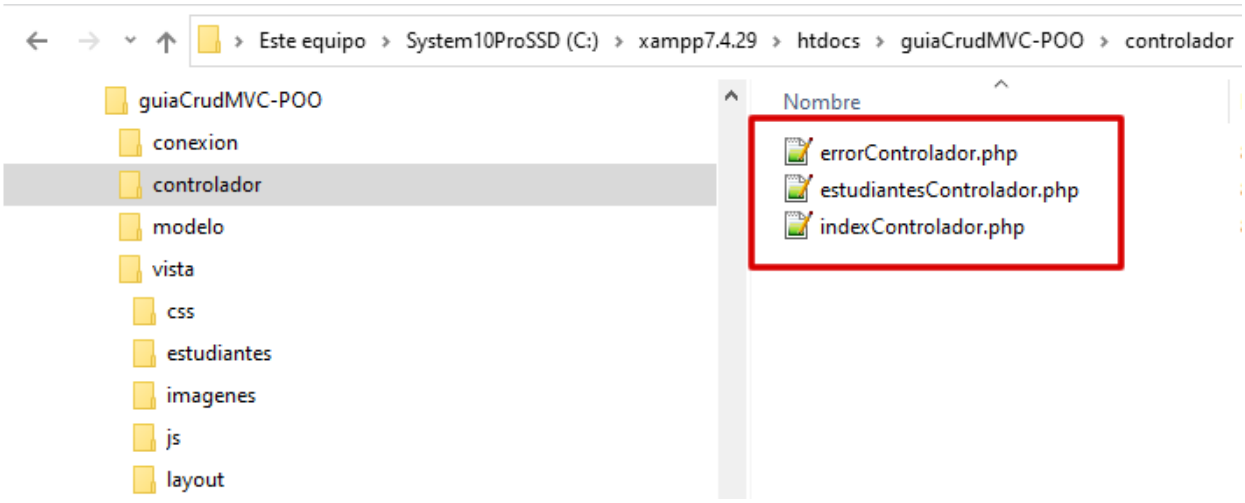


Imagen 1

Para este ejemplo de un CRUD con MVC tendremos la siguiente estructura

Conexión: esta carpera contiene el archivo de **config.php**, el cual es una clase llamada **Cconfig** el cual contiene los atributos de conexión y los métodos Getter de dichos atributos, el archivo **basedatos.php** el cual es una clase el cual es un **extends** de la clase **mysqli** de php, aquí tenemos un solo método el cual es el **__construct()** dicho método se ejecuta cuando se hace una instanciación a dicha clase, es decir cuando la misma sea instanciada se realiza la conexión con la base de datos. Mantiene la estructura y el contenido tal cual al anterior.

Controlador: aquí tendremos el controlador o los controladores de la aplicación denotado o nombrado de la siguiente manera **procesoControlador.php**, por ejemplo, para el caso de este ejemplo **estudiantesControlador.php**, se debe respetar dicha estructura ya que el index.php hará el llamado el controlador según el proceso o la acción que se quiera ejecutar.



Podemos observar 3 controladores (Para esta aplicación puede haber muchos según el tamaño de esta).

- **errorControlador.php**: se ejecutará en el caso de que el proceso o archivo llamados no se encuentre en el directorio.

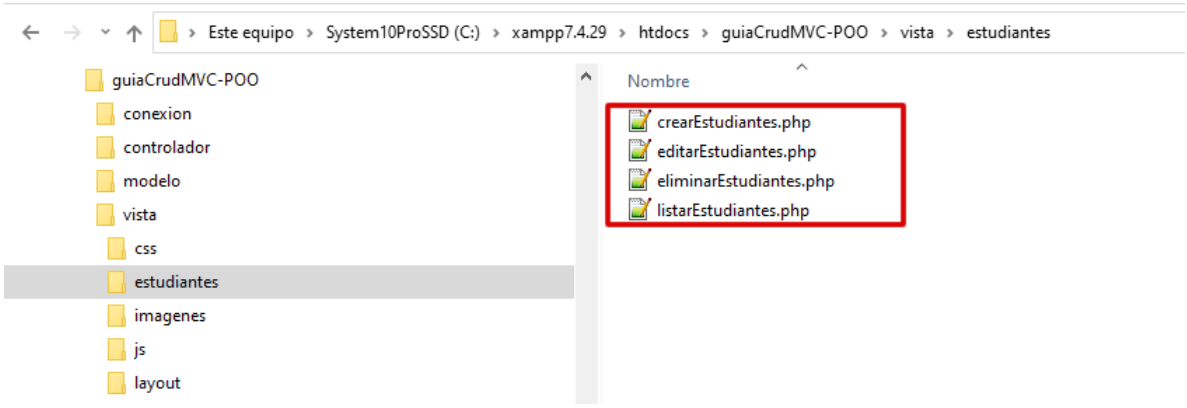
- **indexControlador.php**: se ejecutará cuando se la primera vez que se cargue la aplicación.
- **EstudiantesControlador.php**: el controlador del ejemplo actual.

Modelo: aquí tendremos la clase estudiantes, pero con otro nombre para este caso **estudianteModelo.php**, con un pequeño cambio el cual utilizaremos instrucciones SQL preparadas esto para darle seguridad a nuestra aplicación y así evitar las inyecciones SQL. Véase https://www.php.net/manual/es/security.database.sql-injection.php , de la ayuda oficial de php.

Véase: Pueden verificar el video de mi autoría de cómo realizar una inyección SQL.
https://www.youtube.com/watch?v=VixXzpgrwus
Tiempo: En el minuto 11:55, Inyecciones SQL. https://youtu.be/VixXzpgrwus?t=716

Vista: contiene las diferentes carpetas que contiene la vista por ejemplo:

Css: donde se almacenan los archivos css (No usada).
Imágenes: donde se almacenan la imagen (No usada).
Js: donde se almacenarán los archivos js (javascript).
layout: donde tendremos la plantilla que compone la aplicación, encabezado.html, pie.html, también podemos agregar Menú de la aplicación u otra sección que permita darle mejor vistosidad a la aplicación.
estudiantes: donde tendremos las diferentes vistas de estudiantes.



Recordar : que estas vistas utilizan php embebido en HTML, lo cual no es una buena técnica programación para ello se puede hacer uso de JavaScript, JQuery, Ajax, usar un framework especial para realizar Vistas (Extjs, Vuejs, Angular, entre otras cientos existentes) estas librerías permiten proporcionar al usuario un entorno agradable a la vista, fácil de manejar y atractivo, en su mayoría usan JavaScript, y como estamos trabajando con MVC podemos hacer las vistas en dichos framework y usar el mismo controlador y el mismo Modelo sin alterar el codigo.

Nota : La siguiente explicación es una copia fiel y exacta del artículo [3] - Desarrollo de un CRUD usando HTML + Php (mysqli) + MySql), ya que el uso para el usuario final es el mismo el que cambia es el código que se empleó para el desarrollo.

lista_estudiantes.php (Imagen 2): es el primer archivo que se debe ejecutar y contiene la lista que permite visualizar los diferentes registros existentes en la tabla permitiendo manipularlo mostrando las opciones de Agregar, Modificar y Eliminar respectivamente. A nivel de programación se sigue la secuencia de consulta a la tabla con un SELECT, extraer los registros a la memoria del PC y luego son mostrados ordenados en una tabla y se colocan los botones de acción.

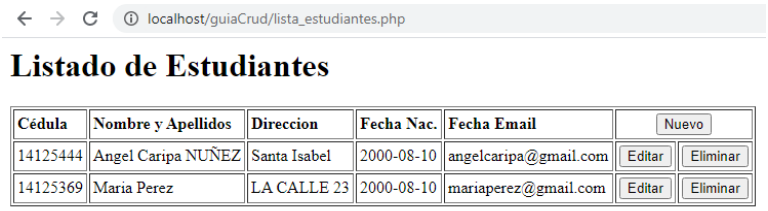


Imagen 2

Agregar Nuevo Registro

Presionemos el botón nuevo y nos llevará al formulario de estudiantes.html, el cual permite capturar los datos para poder registrar un nuevo estudiante que será anexado a la lista, el formulario vacío (Imagen 3), procedemos a editar los datos del mismo (imagen 4) y al presionar el botón de enviar (Imagen 4), mostrara el mensaje de Guardado con Exito y un link para regresar al listado (Imagen 5). A nivel de programación la secuencia seria estudiante.html (Formulario que captura los datos) este envía los datos a un archivo llamado guardar_estudiante.php y este conecta con la base de datos y guarda el registro.

Imagen 3

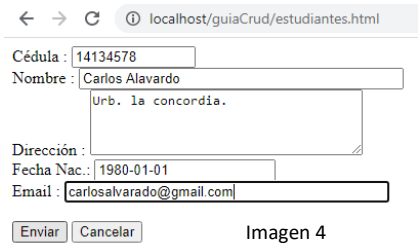


Imagen 4

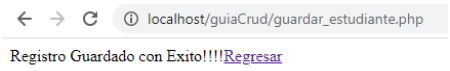


Imagen 5

Cuando se regresa a la lista podemos visualizar el nuevo registro de Carlos Alvarado (imagen 6).

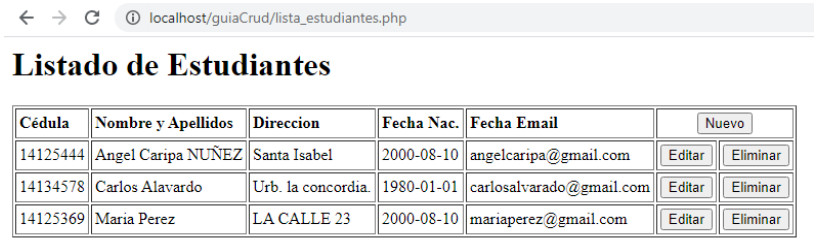


Imagen 6

Editar Registro

Si elegimos editar un Registro seleccionamos el que deseamos haciendo Click en la fila correspondiente al registro, aparecerá un mensaje indicando si se desea editar el registro (Imagen 7, Algo opcional, pero hacemos uso del lenguaje JavaScript embebido en HTML).

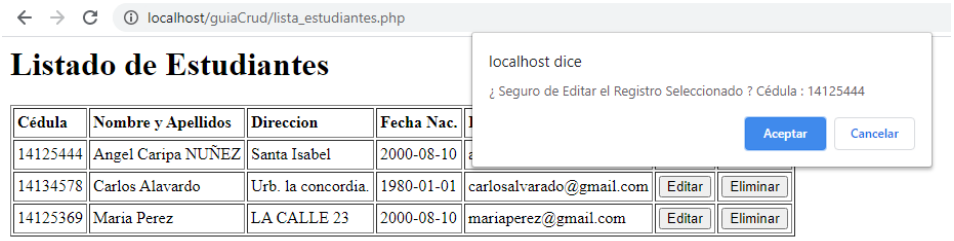


Imagen 7

Presionamos Aceptar y nos permitirá realizar la edición de este. A nivel de programación la secuencia es , al seleccionar el botón de editar se envía la cedula(campo clave) via GET(Observe la URL en la barra de dirección los datos son enviados a través de esta, Imagen 8), se realiza una consulta a la tabla contenida en la base de datos con SELECT se almacenan los datos en la memoria y se muestran en las cajas de textos con la opción de Editar, modificar o cambiar, véase imagen 9 donde se cambian el nombre se borra **NUÑEZ** y se le agregar **la Playa** en dirección, al presionar Enviar en la imagen 9 se llama **actualizar_estudiante.php** este hace la conexión con la base de datos y actualiza el registro y muestra el mensaje (Imagen 10), al presionar el link de Regresar se muestra el listado de estudiantes actualizado (Imagen 11).

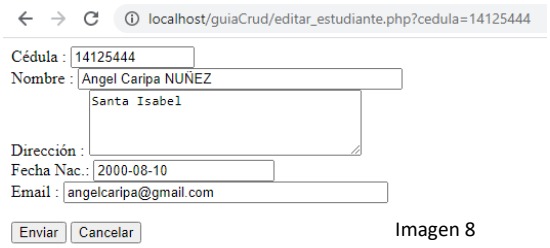


Imagen 8

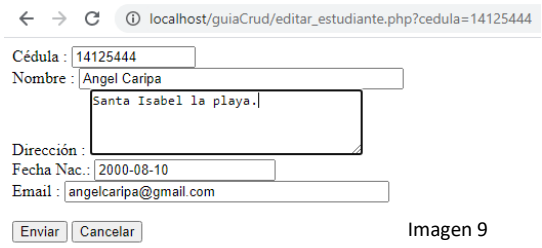


Imagen 9

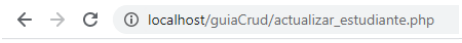


Imagen 10

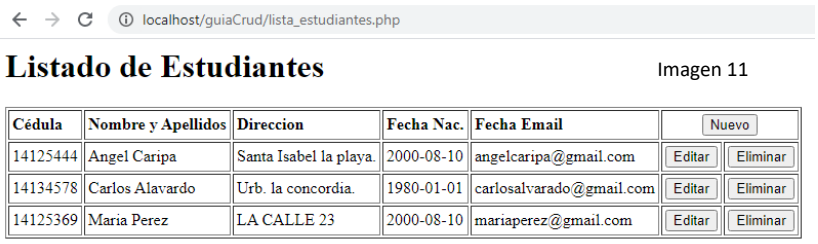


Imagen 11

Editar Registro

Si elegimos eliminar un Registro seleccionamos el que deseamos haciendo Click en la fila correspondiente al registro, aparecerá un mensaje indicando si se desea eliminar el registro (Imagen 12, este si no es opcional ya que voy a eliminar el mismo la idea sería consultar el usuario si realmente está de acuerdo en eliminar el registro, de igual manera hacemos uso de código JavaScript embebido en html (Imagen 12).

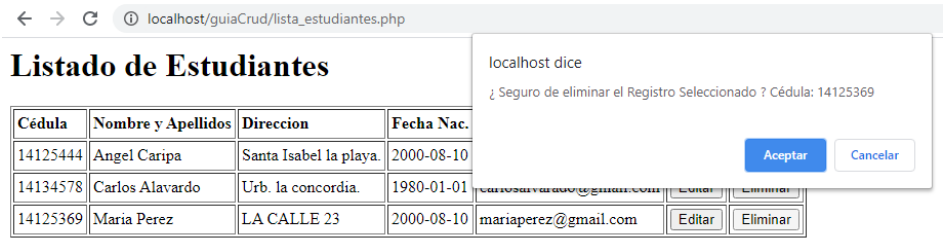


Imagen 12

Presionamos Aceptar y nos permitirá eliminar el registro, nos mostrará el menaje de Registro Eliminado (Imagen 13). A nivel de programación la secuencia es, al seleccionar el botón de eliminar se envía la cedula(campo clave) via GET(Observe la URL en la barra de dirección los datos son enviados a través de esta, Imagen 13), se realiza un DELETE en la tabla correspondiente y se elimina el mismo, al regresar de nuevo al listado vemos que el mismo esta actualizado con 1 registro menos (Imagen 14).

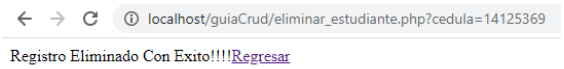


Imagen 13

Imagen 14

Nota: una aclaración importante con respecto a la eliminación de registros, ya que esto es algo delicado ya que una eliminación con DELETE permite eliminar físicamente el registro y el mismo no se puede recuperar, muchos autores o programadores utilizan la eliminación lógica la cual consiste en marcar o ocultar los registros para que los mismos no aparezcan en las consultas y se puedan recuperar en cualquier momento, otros utilizan una tabla o base de datos de histórico, otros utilizan una auditoria o bitácora de eventos ocurridos en los registros para asi tener un seguimiento a los que le sucedió a un registro. Cualquier técnica que utilice es valida siempre y cuando pueda permitir tener un control, integridad y seguridad en los datos.

Esta explicación es a nivel de usuario Final, es decir lo que hace el usuario en las vistas de la aplicación, y por supuestos todos estos procesos inciden en la base de datos actualizando la misma, ahora analicemos el código contenido en cada uno de los archivos de la aplicación.

Nota: la explicación de como el usuario maneja la aplicación se le conoce como manual de Usuario final.

Código de los archivos (Idéntico a la anterior guía)

conexión/config.php : en este archivo se encuentra la clase de **Cconfig** la cual contiene los datos de conexión con el servidor de base de datos, nótese que los atributos son de tipo privado y para poder acceder a ellos se debe crear los métodos Getter(Permiten leer el valor de los atributos) el cual son de tipo público y son los que están presentes o se pueden acceder cuando se hace un instanciación a dicha clase.


```php
<?php
class Cconfig
{
    /** Atributos de la Clase */
    /** Datos de conexion al servidor de Base de Datos. */
    private $baseDato       = "academia";
    private $servidor       = "localhost";
    private $usuarioBd      = "root";
    private $claveBd        = "";
    private $puerto         = "3306";

    /**
     * Get the value of baseDato
     */
    public function getBaseDato()
    {
        return $this->baseDato;
    }

    /**
     * Get the value of servidor
     */
    public function getServidor()
    {
        return $this->servidor;
    }

    /**
     * Get the value of usuarioBd
     */
    public function getUsuarioBd()
    {
        return $this->usuarioBd;
    }

    /**
     * Get the value of claveBd
     */
    public function getClaveBd()
    {
        return $this->claveBd;
    }

    /**
     * Get the value of tipoBaseDato
     */
    public function getTipoBaseDato()
    {
        return $this->tipoBaseDato;
    }

    /**
     * Get the value of puerto
     */
    public function getPuerto()
    {
        return $this->puerto;
    }
}
```

conexion/basedatos.php : (Idéntico al anterior)en este archivo encontramos una clase **CBasedatos** la cual realiza un **extends** a la clase mysqli de php el cual están contenido los métodos para acceder a base de datos de MySQL y MariaDB. Al final de la guía pueden encontrar una Lista de los Métodos con sus parámetros y la descripción de cada uno de ellos (en Ingles).

C:\xampp7.4.29\htdocs\guiaCrudPOO\conexion\basedatos.php - Notepad++

Archivo Editar Buscar Vista Codificación Lenguaje Configuración Herramientas Macro Ejecutar Plugins Ventana ?

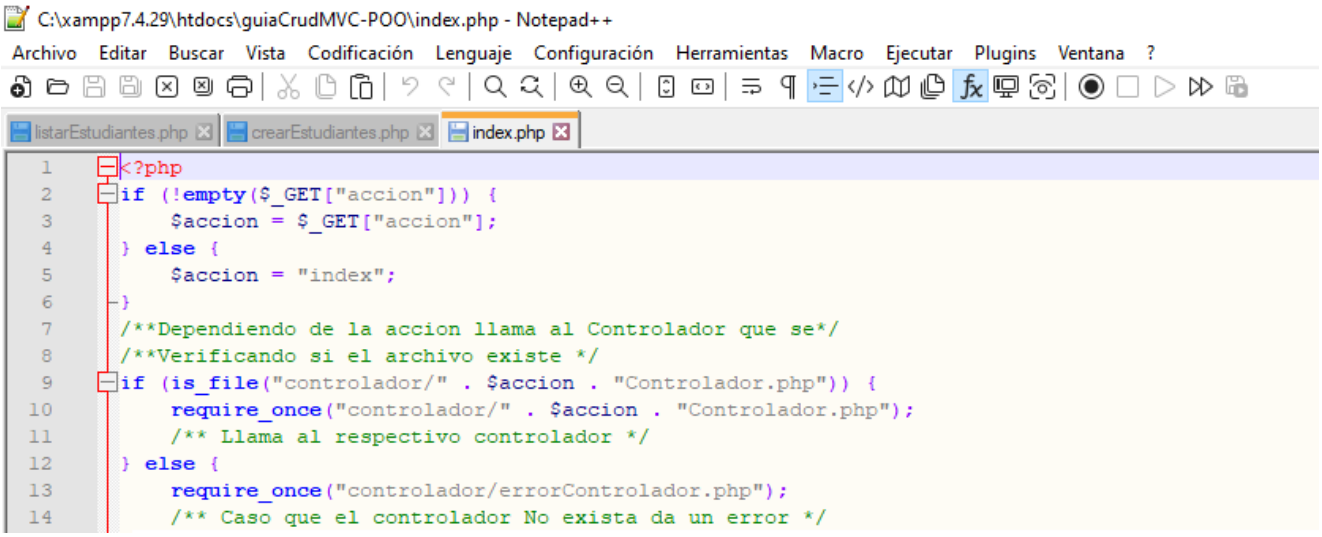
config.php basedatos.php

```php
<?php
/**Se requiere la utilizacion de la clase Cconfig contenida en el archivo config.php*/
require_once('config.php');

class CBaseDato extends mysqli
{
    private $dbh;
    public function __construct()
    {
        try {
            /**Instanciacion a la Clase config donde se encuentran todos los parametros de la misma*/
            $Oconfig   = new CConfig();
            $this->dbh = parent::__construct($Oconfig->getServidor(),
            $Oconfig->getUsuarioBd(), $Oconfig->getClaveBd(), $Oconfig->getBaseDato());
            if (mysqli_connect_error()) {
                die('Error de Conexión (' . mysqli_connect_errno() . ') '
                        . mysqli_connect_error());
            }
        } catch (Exception $e) {
            echo "Error : ".$e->getCode().":".$e->getMessage();
        }
    }

    /* cerrarConexion(): permite cerrar la conexion con el servidor de Base de Datos */
    public function cerrarConexion()
    {
        $this->conn->close();
        $this->dbh=null;
    }
}
```

PHP Hypertext Preprocessor file | length : 1.051 lines : 30 | Ln : 30 Col : 2 Pos :

Index.php (Nuevo)

Este archivo se encuentra en la raíz del sitio y es el responsable de capturar cual es la acción que se va a ejecutar, cuando hacemos referencia a la acción no es mas el modulo que llamara en cuestión. Este archivo tiene algo particular y es que el es un archivo el cual crea dinámicamente el nombre del controlador a ejecutar, ya que el mismo se encarga de capturar que acción se va a ejecutar y arma el nombre del archivo el cual es el que va a ejecutar de allí el nombre de los archivos indexControlador.php, estudiantesControlador.php, errorControlador.php.



En la línea numero 2: observamos que se verifica la existencia de la variable acción la cual es enviada via $_GET, es decir a través de la URL, **empty()** es una función de php que verifica si existe o no la variable.

Ejemplo de un llamado al index.php

- http://localhost/guiaCrudMVC-POO : nótese que no se envía ningún parámetro por ser la primera ejecución, como no existe ningún parámetro, la variable acción toma el valor de índex, luego veamos en la línea numero 9, armamos el nombre de un archivo usando una constante controlador + el contenido de la variable acción + la constante Controlador.php, esto formaría **controlador/indexControlador.php**, este valor es pasado a **is_file()** función que determina si un archivo existe en el directorio mencionado, si existe es llamado o se incluye, caso contrario envía el mensaje de error.
- http://localhost/guiaCrudMVC-POO/?accion=estudiantes: nótese que se envía un parámetro de acción=estudiantes, quiere decir que el archivo armará un nombre de archivo con el siguiente **controlador/estudiantesControlador.php,** si este existe lo incluirá o llamará caso contrario mostrar el error. Caso de existir el archivo el controlador se encargará de la demás ejecución.
- http://localhost/guiaCrudMVC-POO/?accion=estudiantes&operacion=leer : nótese que se envían dos parámetros acción=estudiantes y operación = leer, aquí igualmente se armara el archivo estudiantesControlador.php y además tenemos una variable de operación la cual será la que se verifique en el controlador y ejecutara los método correspondiente para cada operación, sean leer, nuevo, buscar, actualizar, eliminar entre otros método contenidos en el **estudiantesModelo.php** (La misma clase_estudiantes.php de la guía anterior pero ahora la llamaremos de ese nombre).

Resumen:

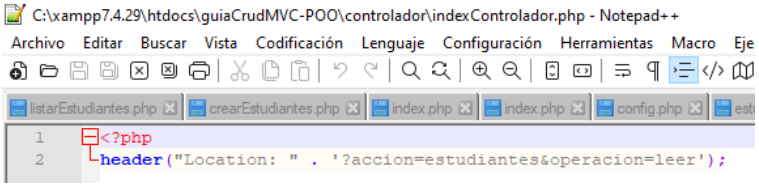
index.php es el archivo base para el llamado a los controladores es decir si creo un CRUD u otro modulo llamado por ejemplo **materias**, entonces quiere decir que debo tener un **materiasControlador.php**, **materiasModelo.php**, en la carpeta vista una carpeta llamada materias y dentro **crearMaterias.php, editarMaterias.php, eliminarMaterias.php, listarMaterias.php**, el cual son los archivos de las diferentes vistas que se usan para el caso de materias y el llamado para ponerlo en ejecución serian:

- http://localhost/guiaCrudMVC-POO/?accion=materias
- http://localhost/guiaCrudMVC-POO/?accion=materias&operacion=leer
- http://localhost/guiaCrudMVC-POO/?accion=materias&operacion=nuevo
- http://localhost/guiaCrudMVC-POO/?accion=materias&operacion=guardar
- http://localhost/guiaCrudMVC-POO/?accion=materias&operacion=buscarEst
- http://localhost/guiaCrudMVC-POO/?accion=materias&operacion=editar
- http://localhost/guiaCrudMVC-POO/?accion=materias&operacion=actualizar
- http://localhost/guiaCrudMVC-POO/?accion=materias&operacion=confirmarEli
- http://localhost/guiaCrudMVC-POO/?accion=materias&operacion=eliminar

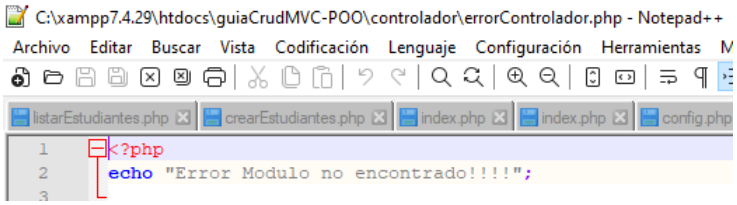
La variable acción sería el módulo, o el controlador a llamar y la variable operación indicará que método debe ejecutarse dentro de la clase que esta definida en el modelo.

Contenido de la Carpeta Controlador

indexControlador.php (Nuevo): solo contine una línea de código si es ejecutada solo se llama al index.php pero se le envía los parámetros de estudiantes (acción o modulo) y operación leer el cual muestra el listado. Aquí se podría poner por ejemplo el llamado a un Inicio de Sesión donde se pida Usuario y Password.



errorControlador.php (Nuevo): se muestra una sola línea indicando que ocurrió un error, aquí se podría poner un mensaje más elaborado.



controladorEstudiantes.php (Nuevo): este archivo contenido dentro de la carpeta controlador el mismo es el que proporciona el control a la aplicación cuando es invocado la acción estudiante, es decir controla que método se debe llamar cuando el usuario desde la vista haga un evento sobre los botones, o eventos implícitos con el leer el cual ocurre cuando se carga la aplicación por primera vez. Para el caso de este controlador no está hecho en POO, pero se podría realizar una clase para su uso, sin embargo, para lo didáctico de la guía y fácil entendimiento lo hacemos a la vieja escuela, en una próxima guía se podría realizar de esa manera.

C:\xampp7.4.29\htdocs\guiaCrudMVC-POO\controlador\estudiantesControlador.php - Notepad++

Archivo Editar Buscar Vista Codificación Lenguaje Configuración Herramientas Macro Ejecutar Plugins Ventana ?

listarEstudiantes.php | crearEstudiantes.php | index.php | index.php | config.php | estudiantesControlador.php

```php
<?php
/**La url de llamado deberia ser  :
 * http://localhost/guiaCrudMVC-POO/?accion=estudiantes&operacion=leer
 * Notese que la misma trae o envia los parametros de accion y operacion
 * */
if (isset($_REQUEST['operacion'])) {
    $operacion = $_REQUEST['operacion'];
    /**Extraigo todas las variables que vienen del formulario enviado y se convierten en
     * variables que pueden ser utilizadas sin usar el $_REQUEST, $_GET, $_POST, con esto
     * me ahorro la creacion de las variables y la asiganacion de la misma.
     * El seteo se hace directamente solo se debe tener en cuenta que el nombre de la variable
     * se crea con el mismo nombre que se le asigno al campo del input
     * <input type="text" name="cedula" id="cedula" size="12" maxlength="12" required />
     * para el ejemplo anterior la variable creada sera $cedula
     */
    extract($_REQUEST);
    /**Llama la clase estudiantes */
    include('modelo/estudiantesModelo.php');
    /**Realiza la instanciacion de la clase usando la variable de tipo Objeto $Oestudiantes */
    $Oestudiantes = new Cestudiantes();
    /**Caso que se recibio operacion = lerr */
    if ($operacion == 'leer') {
        /**Invoca el metodo publico llamado leerEst(): leer estudiantes almacena el resultado
         * en la variable de tipo arreglo $datos*/
        $datos = $Oestudiantes->leerEst();
        /**Cuenta la Cantidad de Registros para mandarselo a la vista la misma
         * verificara para mostrar
         * */
        $cant = count($datos);
        require('vista/estudiantes/listarEstudiantes.php');
        return;
    }
    /**Caso Nuevo estudiante->Solo mostraria el formulario operacion = nuevo*/
    if ($operacion == 'nuevo') {
        require('vista/estudiantes/crearEstudiantes.php');
        return;
    }
    /**Caso guardar ocurre al presionar el boton de Enviar del formulario
     * crearEstudiante.php operacion=guardar
     **/
    if ($operacion == 'guardar') {
        /**Seteo los atributos con los valores obtenidos del formulario */
        $Oestudiantes->setCedula($cedula);
        $Oestudiantes->setNombre($nombre);
        $Oestudiantes->setDireccion($direccion);
        $Oestudiantes->setFechanac($fechanac);
        $Oestudiantes->setEmail($email);
        /**Llama el ametodo agergarEst() el cual guarda el estudiante */
        $Oestudiantes->agregarEst();
        /**Muesta el mensaje del resultado del metosdo y un boton de Regresar */
        echo "<a href='?accion=estudiantes&operacion=leer'>Regresar</a>";
        return;
    }
    /**Caso operacion = eliminar */
    if ($operacion == "eliminar") {
        /** Seteo solo la cedula ya que es el campo clave el cual necesito para eliminar */
        $Oestudiantes->setCedula($cedula);
        /** Llamo el metodo de eliminarEst(), el cual elimina el estudiante. */
        $Oestudiantes->eliminarEst();
        /** Muestra rsultado del metodo actual y muestra el boton de regresar*/
        echo "<a href='?accion=estudiantes&operacion=leer'>Regresar</a>";
    }
    /**Caso operacion confirmarEl, muestra el formulario con los datos para estar
     * seguro de la eliminacion.
     */
    if ($operacion == "confirmarEli") {
        /**Seteo el valor de la cedula el cual es el campo clave por donde se realiza
         * la busqueda
         * */
        $Oestudiantes->setCedula($cedula);
        /**Llamo al metodo que busca los datos */
        $datos = $Oestudiantes->buscarEst();
        /**Se incluye la vista de eliminarEstudiantes.php la cual muestra los datos antes obtenidos */
        require('vista/estudiantes/eliminarEstudiantes.php');
        return;
    }
    /**Caso operacion = editar */
    if ($operacion == "editar") {
        /**Seteo el valor de la cedula el cual es el campo clave por donde se realiza
         * la busqueda
         * */
        $Oestudiantes->setCedula($cedula);
        /**Llamo al metodo que busca los datos hago un reuso del metodo*/
        $datos = $Oestudiantes->buscarEst();
        /**Se incluye la vista de editarEstudiantes.php la cual muestra los datos antes obtenidos */
        require('vista/estudiantes/editarEstudiantes.php');
        return;
    }
    /**Caso operacion = actualizar o guardar los datos editados en el formulario editarEstudiantes.php */
    if ($operacion == "actualizar") {
        /**Seteo los atributos con los valores obtenidos del formulario */
        $Oestudiantes->setCedula($cedula);
        $Oestudiantes->setNombre($nombre);
        $Oestudiantes->setDireccion($direccion);
        $Oestudiantes->setFechanac($fechanac);
        $Oestudiantes->setEmail($email);
        $Oestudiantes->modificarEst();
        echo "<a href='?accion=estudiantes&operacion=leer'>Regresar</a>";
    }
} else {
    /**Caso de que el parametro o el llamado sea de la siguiente manera
     * http://localhost/guiaCrudMVC-POO  : sin especificar la accion.
     * http://localhost/guiaCrudMVC-POO/?accion=estudiantes : especificando solo la accion sin la operacion
     * */
    header("Location: " . '?accion=estudiantes&operacion=leer');
}
```

PHP Hypertext Preprocessor file | length : 5.195 lines : 107 | Ln : 5 Col : 6 Pos : 198 | Windows (CR LF)

Dicho controlador no es mas que un archivo el cual tiene la función de consultar que operación se esta ejecutando recordemos que ya el **index.php** (controlador principal de la aplicación), nos indico que se debería de ejecutar gracias a la acción que recibió, ahora este verificará que operación debe ejecutar para ello utiliza un simple if (condición) para verificar que método se lleve llamar. Podemos encontrar, por ejemplo:

```php
<?php
/**La url de llamado deberia ser  :
 * http://localhost/guiaCrudMVC-POO/?accion=estudiantes&operacion=leer
 * Notese que la misma trae o envia los parametros de accion y operacion
 * */
if (isset($_REQUEST['operacion'])) {
    $operacion = $_REQUEST['operacion'];
    /**Extraigo todas las variables que vienen del formulario enviado y se convierten en
     * variables que pueden ser utilizadas sin usar el $_REQUEST, $_GET, $_POST, con esto
     * me ahorro la creacion de las variables y la asiganacion de la misma.
     * El seteo se hace directamente solo se debe tener en cuenta que el nombre de la variable
     * se crea con el mismo nombre que se le asigno al campo del input
     * <input type="text" name="cedula" id="cedula" size="12" maxlength="12" required />
     * para el ejemplo anterior la variable creada sera $cedula
     */
    extract($_REQUEST);
    /**Llama la clase estudiantes */
    include('modelo/estudiantesModelo.php');
    /**Realiza la instanciacion de la clase usando la variable de tipo Objeto $Oestudiantes */
    $Oestudiantes = new Cestudiantes();
    /**Caso que se recibio operacion = lerr */
    if ($operacion == 'leer') {
        /**Invoca el metodo publico llamado leerEst(): leer estudiantes almacena el resultado
         * en la variable de tipo arreglo $datos*/
        $datos = $Oestudiantes->leerEst();
        /**Cuenta la Cantidad de Registros para mandarselo a la vista la misma
         * verificara para mostrar
         * */
        $cant = count($datos);
        require('vista/estudiantes/listarEstudiantes.php');
        return;
    }
    /**Caso Nuevo estudiante->Solo mostraria el formulario operacion = nuevo*/
    if ($operacion == 'nuevo') {
        require('vista/estudiantes/crearEstudiantes.php');
        return;
    }
```

En la línea numero 6: encontramos la verificación de la existencia de la variable operación, si la misma no se encuentra es decir no fue enviada desde la url, entonces se ejecuta el **else** el cual me envía nuevamente el llamado al controlador estudiantes, pero con el parámetro de leer, Véase líneas numero 100 al 105.

```php
} else {
    /**Caso de que el parametro o el llamado sea de la siguiente manera
     * http://localhost/guiaCrudMVC-POO  : sin especificar la accion.
     * http://localhost/guiaCrudMVC-POO/?accion=estudiantes : especificando solo la accion
     * */
    header("Location: " . '?accion=estudiantes&operacion=leer');
}
```

PHP Hypertext Preprocessor file — length : 5.195 lines : 107

En la línea numero 16: encontramos algo nuevo que ayuda a la captura de las variables evitándome crear una variable y asignar el valor del formulario que viene por $_REQUEST[], $_GET[] o $_POST[], la función extrac(). Útil, pero php.net nos indica una advertencia de seguridad.

En la línea numero 18: incluimos el archivo estudiantesModelo.php, tenga en cuenta la ruta, con esto tenemos acceso a la clase Cestudiantes, la cual debemos instanciar para poder tener acceso a sus métodos y atributos.

En la línea numero 20: instanciamos la clase creando para ello una variable llamada $Oestudiante, la O mayúscula indicara que es un Objeto.

Listar Estudiantes o Mostrar Listado

Antes de llamar a cada método se consulta el valor de la variable operación, si la misma leer entonces llamaremos al método leerEst() en este caso particular el método devuelve un arreglo, el cual es guardado en una variable $datos (Línea 25).

En la línea numero 29: encontramos la variable **$cant** la cual toma el valor devuelto de **count()**, función la cual se encarga de indicar si el arreglo o variable datos tiene información o no, esta variable es pasada a la vista la cual se encargará de emitir el mensaje en caso de no haber registros o mostrar el listado de registros.

En la línea numero 30: incluimos el archivo listarEstudiantes.php, que está o pertenece a la vista, la cual se encargara de mostrar el listado, es por ello por lo que en la vista (listarEstudiantes.php), no se muestra el código del llamado a el método leerEst() o el código de conexión a la Base de datos.

Nuevo Estudiante

En la línea numero 34: observamos el llamado o inclusión de la vista **crearEstudiante.php** esta no llama ningún método ya que solo mostrara el formulario para la captura de datos.

```
38      /**Caso guardar ocurre al presionar el boton de Enviar del formulario
39       * crearEstudiante.php operacion=guardar
40       **/
41      if ($operacion == 'guardar') {
42          /**Seteo los atributos con los valores obtenidos del formulario */
43          $Oestudiantes->setCedula($cedula);
44          $Oestudiantes->setNombre($nombre);
45          $Oestudiantes->setDireccion($direccion);
46          $Oestudiantes->setFechanac($fechanac);
47          $Oestudiantes->setEmail($email);
48          /**Llama el ametodo agergarEst() el cual guarda el estudiante */
49          $Oestudiantes->agregarEst();
50          /**Muesta el mensaje del resultado del metosdo y un boton de Regresar */
51          echo "<a href='?accion=estudiantes&operacion=leer'>Regresar</a>";
52          return;
53      }
```

En la línea número 41: observamos la verificación de la operación guardar, esta es enviada cuando se hace click en el botón de Enviar un botón del tipo Submit (Véase la vista crearEstudiante.php), en este caso el llamado seria: ?accion=estudiantes&operacion=guardar, la misma ubicada en el action del formulario (Véase la guía 2 - Formularios HTML y procesamiento usando php)

En la línea Numero 43-47: Aquí nos encontramos con el seteo de los atributos de la clase como estos atributos son de tipo privados es decir no están accesibles desde una instanciación se deben utilizar los metodos Setter para setear o asignar los valores de las variables devueltas desde el formulario, recuerde que para definir las variables usamos la función extrac de php (Línea número 16).

setCedula() : será el método que se encargara de asignarle el contenido de la variable **$cedula**, la cual es enviado como parámetro, dicha variable **$cedula** fue enviada por el formulario cuyo valor **name='cedula'**, es por ello que se crea una variable llamada **$cedula**, gracias a la función extrac(), así análogamente se hace el llamado a los demás metodos set, estos método deben estar creados en la clase Cestudiantes la cual está contenida dentro de **estudiantesModelo.php**.

En la Línea numero 49: hacemos el llamado al método agregarEst(), el cual se encarga de ejecutar la instrucción SQL parametrizada, algo nuevo en esta guía, al finalizar la ejecución de dicho método se muestra un mensaje y posteriormente un botón de Regresar.

Eliminar estudiantes.

Para eliminar estudiantes, primero debemos de realizar una búsqueda del estudiantes a eliminar para luego mostrarlo y pedir una confirmación de eliminación, el controlador observamos el llamado a dichos metodos y vistas respectivas.

```
54      /**Caso operacion = eliminar */
55      if ($operacion == "eliminar") {
56          /** Seteo solo la cedula ya que es el campo clave el cual necesito para eliminar */
57          $Oestudiantes->setCedula($cedula);
58          /** Llamo el metodo de eliminarEst(), el cual elimina el estudiante. */
59          $Oestudiantes->eliminarEst();
60          /** Muestra rsultado del metodo actual y muestra el boton de regresar*/
61          echo "<a href='?accion=estudiantes&operacion=leer'>Regresar</a>";
62      }
63      /**Caso operacion confirmarEl, muestra el formulario con los datos para estar
64       * seguro de la eliminacion.
65       */
66      if ($operacion == "confirmarEli") {
67          /**Seteo el valor de la cedula el cual es el campo clave por donde se realiza
68           * la busqueda
69           * */
70          $Oestudiantes->setCedula($cedula);
71          /**Llamo al metodo que busca los datos */
72          $datos = $Oestudiantes->buscarEst();
73          /**Se incluye la vista de eliminarEstudiantes.php la cual muestra los datos antes obtenidos */
74          require('vista/estudiantes/eliminarEstudiantes.php');
75          return;
76      }
```

En la línea numero 66: observamos la verificación de la operación confirmarEli, el cual me permite buscar el registro que se va a eliminar (Línea 72) y luego este es mostrado en la vista, cuyo nombres es **eliminarEstudiantes.php (Línea 74),** en esta línea se incluye la vista para mostrar el registro y pedir la confirmación. En esta vista al hacer click en el botón de Enviar (Submit) se hace el llamado a eliminar, el cual es el método que si elimina el registro.

En la línea numero 55: observamos la consulta de la operación de eliminar, donde enviamos la cedula nuestro campo clave y mandamos a ejecutar el método **eliminarEst()**, el cual mostrar el mensaje de eliminado o error al eliminar.

Editar estudiantes

Para editar el estudiante en un caso muy parecido a eliminar solo que aquí se muestran los datos con la intención de que los mismos sean editados y luego se actualizan en la base de datos.

```php
/**Caso operacion = editar */
if ($operacion == "editar") {
    /**Seteo el valor de la cedula el cual es el campo clave por donde se realiza
     * la busqueda
     * */
    $Oestudiantes->setCedula($cedula);
    /**Llamo al metodo que busca los datos hago un reuso del metodo*/
    $datos = $Oestudiantes->buscarEst();
    /**Se incluye la vista de editarEstudiantes.php la cual muestra los datos antes obtenidos */
    require('vista/estudiantes/editarEstudiantes.php');
    return;
}
/**Caso operacion = actualizar o guardar los datos editados en el formulario editarEstudiantes.php */
if ($operacion == "actualizar") {
    /**Seteo los atributos con los valores obtenidos del formulario */
    $Oestudiantes->setCedula($cedula);
    $Oestudiantes->setNombre($nombre);
    $Oestudiantes->setDireccion($direccion);
    $Oestudiantes->setFechanac($fechanac);
    $Oestudiantes->setEmail($email);
    $Oestudiantes->modificarEst();
    echo "<a href='?accion=estudiantes&operacion=leer'>Regresar</a>";
}
```

En la línea numero 78: se realiza la consulta si la operación es editar, y si es correcta seteamos la cedula(Línea 82).

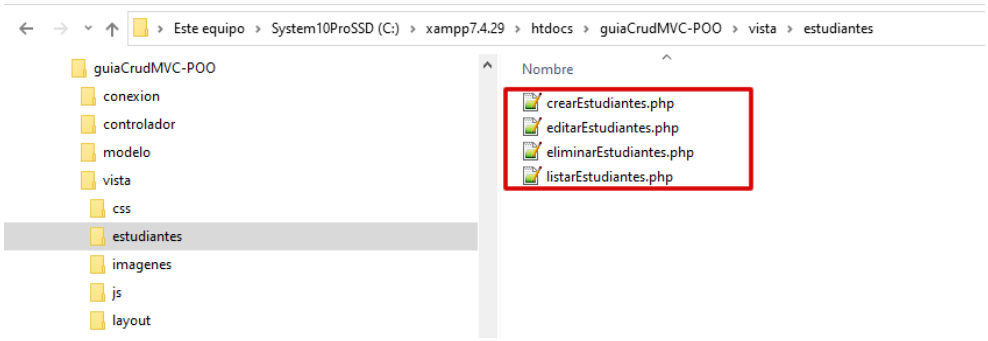
En la línea numero 84: cargamos la variable $datos el cual devuelve un arreglo de 1 registro.

En la línea numero 86: incluimos en archivo editarEstudiantes.php el cual mostrar los datos del registro con opción a guardar o actualizar.

En la línea numero 90: observamos la verificación de si es actualizar, operación que es enviado desde la vista de editar, la cual el llamado es ?accion=estudiantes&operacion=actualizar, si el mismo es correcto se setean las variables, y se llama al método actualizar este emitirá un mensaje si se completó la operación o si ocurrió un error.

Contenido de la carpeta Vista

Aquí tendremos una carpeta llamada estudiantes en el cual están las 4 vistas que se usan en un CRUD, el cual seria listado, crear, editar, eliminar estas solo muestran información al usuario para que el mismo interactúe y le envié las solicitudes al controlador.



Pueden utilizar un archivo, para colocar todas las vistas esto permite tener menos cantidad de archivos y de igual manera el controlador se encargaría de mostrar la vista respectiva según el proceso a ejecutar.

Recordemos que esta guía es una manera de realizar MVC + POO, cada desarrollador lo ve desde su perspectiva y le integra conocimientos, metodos de desarrollo que a la final llevan al mismo resultado, para muestra observen los 3 CRUD, que se han realizado la vista al usuario o el producto final es el mismo pero detrás de él están tres formas de codificación diferentes.

Lista_estudiantes.php(Nuevo)

C:\xampp7.4.29\htdocs\guiaCrudMVC-POO\vista\estudiantes\listarEstudiantes.php - Notepad++

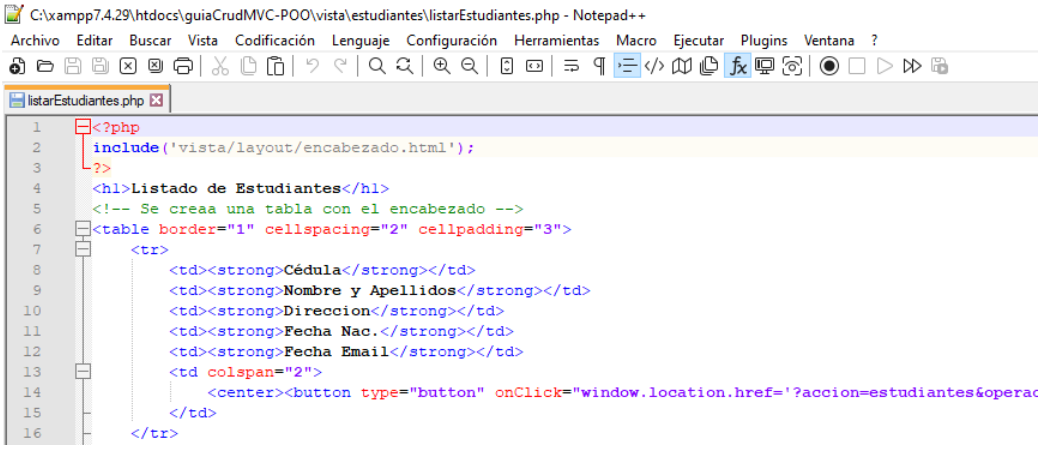
Archivo Editar Buscar Vista Codificación Lenguaje Configuración Herramientas Macro Ejecutar Plugins Ventana ?

listarEstudiantes.php

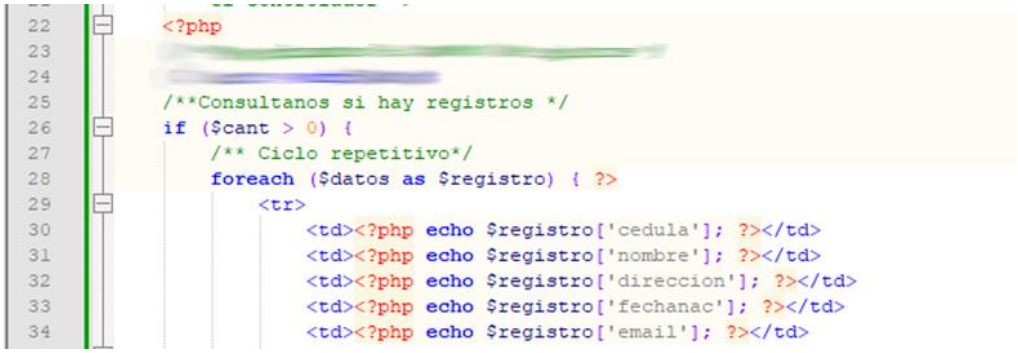
```php
<?php
include('vista/layout/encabezado.html');
?>
<h1>Listado de Estudiantes</h1>
<!-- Se creaa una tabla con el encabezado -->
<table border="1" cellspacing="2" cellpadding="3">
    <tr>
        <td><strong>Cédula</strong></td>
        <td><strong>Nombre y Apellidos</strong></td>
        <td><strong>Direccion</strong></td>
        <td><strong>Fecha Nac.</strong></td>
        <td><strong>Fecha Email</strong></td>
        <td colspan="2">
            <center><button type="button" onClick="window.location.href='?accion=estudiantes&operacion=nuevo'">Nuevo</button></center>
        </td>
    </tr>
    <!--Se inserta codigo php embebido en html se pregunta si hay registros entonces
        hace un ciclo repetitivo(foreach) para mostrar cada uno de los datos de la tabla.
        para cada registro se coloca un boton de eliminar y editar mismo codigo anterior
        pero aqui no vemos donde se hace el llamado a leerEst() el mismo es llamado desde
        el controlador-->
    <?php

    /**Consultanos si hay registros */
    if ($cant > 0) {
        /** Ciclo repetitivo*/
        foreach ($datos as $registro) { ?>
            <tr>
                <td><?php echo $registro['cedula']; ?></td>
                <td><?php echo $registro['nombre']; ?></td>
                <td><?php echo $registro['direccion']; ?></td>
                <td><?php echo $registro['fechanac']; ?></td>
                <td><?php echo $registro['email']; ?></td>
                <!-- Botones para Editar y Eliminar respectivamente
                    AQUI INTEGRAMOS JAVASCRIPT YA QUE NI HTML NI PHP TIENEN ESAS OPCIONES DE MOSTRAR
                    VENTANAS PARA REALIZAR CONFIRMACIONES.
                    los mismo llaman a una funcion de javascript la cual consulta al usuario
                    si esta seguro de realizar la operacion, estas funciones son codigo javascript
                    el cual se colocan al final de la pagina despues de </body> y antes del </html>-->
                <td>
                    <center><button type="button" onClick="javascript:editarRegistro('<?php echo $registro['cedula']; ?>')">Editar</but
                    <!-- editarRegistro() es la funcion la misma recibe como parametro la cedula-->
                </td>
                <td>
                    <center><button type="button" onClick="javascript:eliminaRegistro('<?php echo $registro['cedula']; ?>')">Eliminar</
                    <!-- eliminarRegistro() es la funcion la misma recibe como parametro la cedula-->
                </td>
            </tr>
    <?php };
        /** Fin del ciclo pertenece al do (do {} while(condicion))*/
    } ?>
</table>
<!-- Caso de que no haya Registro se muestra un mesaje al Usuario -->
<?php if ($cant == 0) {
    echo 'No Hay Registros para mostrar!!!!';
}
?>
</div>

<!--Finaliza el Body-->
<!--javaScrip para eliminar registros-->
<script language="Javascript">
    /*Crea la funcion javascritp*/
    function eliminaRegistro(parametro) {
        /** Solicita o hace la consulta al usuario si desea eliminar el registro */
        /** confirm: funcion de javascript que muestra el cuadro de dialogo de Si No */
        eliminar = confirm("¿ Seguro de eliminar el Registro Seleccionado ? Cédula: " + parametro);
        if (eliminar) /**Si se presiono Si*/
            /** Hace el llamado a el script de php para eliminar y le envia  como parametro la cedula recibida*/
            window.location.href = "?accion=estudiantes&operacion=confirmarEli&cedula=" + parametro;
        else
            /** Si el Usuario hace click en No*/
            alert('El Registro no ha sido Eliminado')
    }
    /*JavaScrip para Pedir confirmacion de Editar*/
    function editarRegistro(parametro) {
        /** Solicita o hace la consulta al usuario si desea editar el registro */
        /** confirm: funcion de javascript que muestra el cuadro de dialogo de Si No */
        editar = confirm("¿ Seguro de Editar el Registro Seleccionado ? Cédula : " + parametro);
        if (editar)
            /** Hace el llamado a el script de php para  */
            /** editar(Formulario para mostrar registro y envia  como parametro la cedula recibida*/
            window.location.href = "?accion=estudiantes&operacion=editar&cedula=" + parametro;
    }
</script>

<?php
include('vista/layout/pie.html');
?>
```

Para el código de **listado_estudiantes.php** el cual permite mostrar una tabla con los datos de los estudiantes, hay que hacer énfasis en tres secciones, la primera es el principio del archivo ya que en ningún momento se hace el llamado al código que leer los datos de los estudiantes, el ciclo repetitivo sigue siendo el mismo y en la última sección donde se hace llamado al pie.html. Este archivo usa lo que se llama código php embebido en HTML. Aquí se conservan parte del código de guiaCrud.

Primeras líneas del archivo.



En la línea numero 2: comienza por un include, y hace referencia a un archivo **encabezado.html**, el cual el objetivo es establecer un encabezado para toda la aplicación es decir este archivo debe ser el mismo para todas las pantallas entonces para ahorra trabajo se crear un archivo separado y al modificarse este se modifica en todos los archivos donde se utiliza, así como se tiene el encabezado al principio del archivo se usa uno al final el cual es el **pie.html**, preste especial atención la ruta del include el mismo hace referencia **vista/layout/encabezado.html**, esto se debe a que el archivo que desencadena el llamado a los demás archivos es el **index.php** y él está en la raíz del sitio entonces siempre los llamados a los archivo se harán desde la raíz, es decir para llamar a un controlador debería llamarse **controlador/estudiantesControlador.php**, para llamar el modelo seria **modelo/estudiantesModelo.php**, para una imagen seria **vista/imágenes/logo.png**, y de manera similar para otros llamados a archivos. Debe también prestar atención a que no se hace el llamado a **leerEst()**, esto sucede porque quien llama a ese método es el controlador ya que este es el que se conecta con el modelo carga el arreglo y la vista(**listarEstudiante.php**), solo se encarga de mostrar el resultado o los registros.



En la línea numero 26: realizamos la verificación de la variable $cant si la misma es mayor a 0, es porque el método leerEst(), devolvió una cantidad de registros y los mismos se pueden listar o mostrar.

En la línea numero 28: realizamos el ciclo repetitivo, en este caso utilizamos un foreach(), el cual permite recorrer arreglos asociativos.

En la línea número 30,31,32,33,34: se muestran los valores de los campos, para ello se utiliza la variable $registro[‘campo’], dicha variable está contenida en el foreach() y es la que contiene los campos de cada registro.

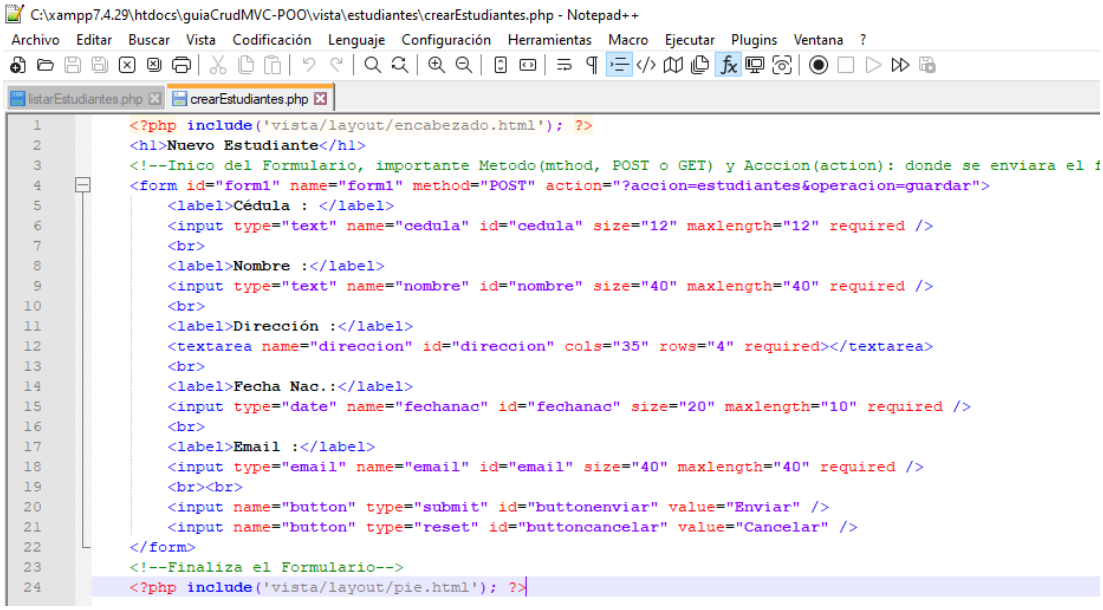
```
49                  </tr>
50              <?php };
51                  /** Fin del ciclo pertenece al do (do {} while(condicion))*/
52              } ?>
53          </table>
54          <!-- Caso de que no haya Registro se muestra un mesaje al Usuario -->
55          <?php if ($cant == 0) {
56              echo 'No Hay Registros para mostrar!!!!';
57          }
58          ?>
59          </div>
60
```

En la línea numero 50: en esta línea tenemos el cierre del foreach().

En la línea numero 55: en esta línea realizamos la verificación nuevamente de la variable $cant si la misma es igual a 0, se emite un mensaje indicando la no existencia de registros, también se puede usar **else** para indicar la no existencia de registros.

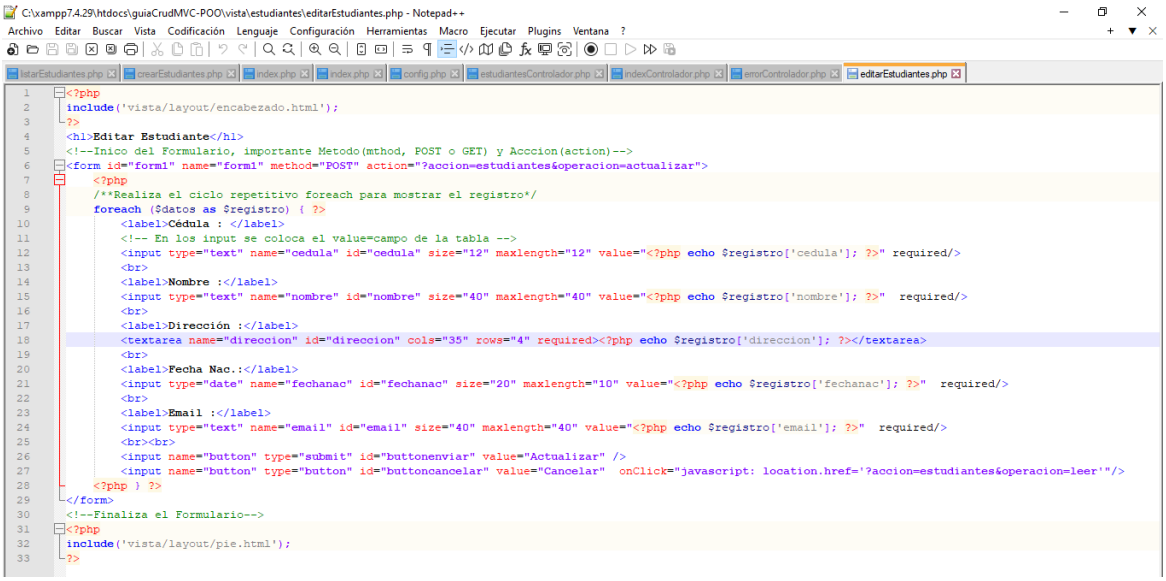
Crear Estudiantes

~~**estudiantes.php**~~ **(Formulario d Captura de Datos):** ahora llamado **crearEstudiantes.php** el mismo formulario, pero ahora está dentro de la carpeta **vista/estudiantes/crearEstudiantes.php** se incluyen encabezado.html y pie.html, el cual están dentro de vista/layout/encabezado.html tenga en cuenta la ruta del archivo, véase líneas 1 y 24 respectivamente, preste especial atención en el action del form, el cual indica ?accion=estudiantes&operacion=guardar



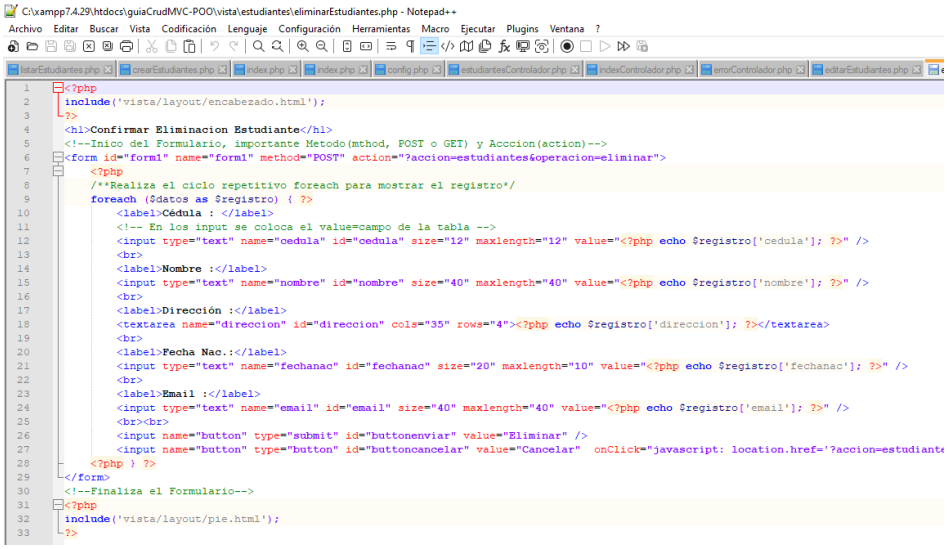
Editar Estudiante

~~**editar_estudiante.php**~~ (Formulario para editar los registros): ahora llamado **editarEstudiantes.php**, el mismo formulario, pero ahora está dentro de la carpeta **vistas/estudiantes/editarEstudiantes.php**, se incluyen encabezado.html y pie.html, el cual están dentro de vista/layout/encabezado.html tenga en cuenta la ruta del archivo, véase líneas 2 y 32 respectivamente, preste especial atención es el action del form ?accion=estudiantes&operacion=actualizar



Eliminar Estudiantes

~~**eliminar_estudiante.php**~~ (Formulario para pedir la confirmación de eliminar registros): ahora llamado **eliminarEstudiantes.php**, el mismo formulario, pero ahora está dentro de la carpeta **vistas/estudiantes/eliminarEstudiantes.php**, se incluyen encabezado.html y pie.html, el cual están dentro de **vista/layout/encabezado.html** tenga en cuenta la ruta del archivo, véase líneas 2 y 32 respectivamente, preste especial atención es el action del form ?accion=estudiantes&operacion=eliminar



Los siguientes archivos fueron eliminados pero su contenido fue movido al controlador es decir estudiantesCcontrolador.php y desde allí son ejecutados.

guardar_estidiante.php (Ya el archivo no existe es eliminado).
editar_estudiante.php (Ya el archivo no existe es eliminado).
actualizar_estudiante.php (Ya el archivo no existe es eliminado).
eliminar_registro.php (Ya el archivo no existe es eliminado).

Contenido de la Clase ~~class_estudiantes.php~~ (Nuevo): ahora contenida en el archivo **estudiantesModelo.php**, la misma contiene la misma clase, mismos atributos, mismos metodos, con un ligero cambio en el uso de las sentencias preparadas es decir las instrucciones SQL del ejemplo anterior (guiaCrudPOO), se le agregan parámetros para evitar la inyecciones SQL.

Cambios realizados en los metodos para el uso de sentencias preparadas.

C:\xampp7.4.29\htdocs\guiaCrudMVC-POO\modelo\estudiantesModelo.php - Notepad++

Archivo Editar Buscar Vista Codificación Lenguaje Configuración Herramientas Macro Ejecutar Plugins Ventana ?

crearEstudiantes.php | index.php | index.php | config.php | estudiantesControlador.php | indexControlador.php | errorControlador.php | editarEstudiante

```
30      public function agregarEst()
31      {
32          /** Guardar el Registro usando INSERT, prepare y bind_param, esto para corregir el error de
33           *  seguirdad que existia en las guia anterior, esto evita ataques de tipo inyeccion de
34           * codigo SQL.
35           * Véase:  Pueden verificar el video de mi autoría de cómo realizar una inyección SQL.
36           * https://www.youtube.com/watch?v=VixXzpgrwus
37           * Tiempo: En el minuto 11:55, Inyecciones SQL. https://youtu.be/VixXzpgrwus?t=716
38           */
39          /* Preparo la instruccion para luego enviarle los parametros*/
40          $sql = $this->conn->prepare("INSERT INTO estudiantes(cedula,nombre,direccion,fechanac,email)
41                                       VALUE(?,?,?,?,?)");
42          /**Envio los parametros a la SQL antes preparada
43           *   Especificación del tipo de caracteres
44           *   Carácter   Descripción
45           *      i       la variable correspondiente es de tipo entero
46           *      d       la variable correspondiente es de tipo double
47           *      s       la variable correspondiente es de tipo string
48           *      b       la variable correspondiente es un blob y se envía en paquetes
49           */
50          $sql->bind_param("sssss",$this->cedula,$this->nombre,$this->direccion,$this->fechanac,$this->email
51          );
52          /** Ejecuto la intruccion SQL ya ya con los parametros asignados */
53          if ($sql->execute()) {
54              /** Si la ejecucion anterior devuelve 1 es pq fue correcta la insercion */
55              /** Mensaje de correcto */
56              echo 'Registro Guardado con Exito!!!!';
57          } else {
58              /** Mensaje de error */
59              echo "Error al Guardar el Registro!!!!";
60          }
61          $sql->close();
62          $this->CerrarConexion();
63          return;
64      }
65
```

PHP Hypertext Preprocessor file | length : 6.535 lines : 202 | Ln : 43 Col : 51 Pos

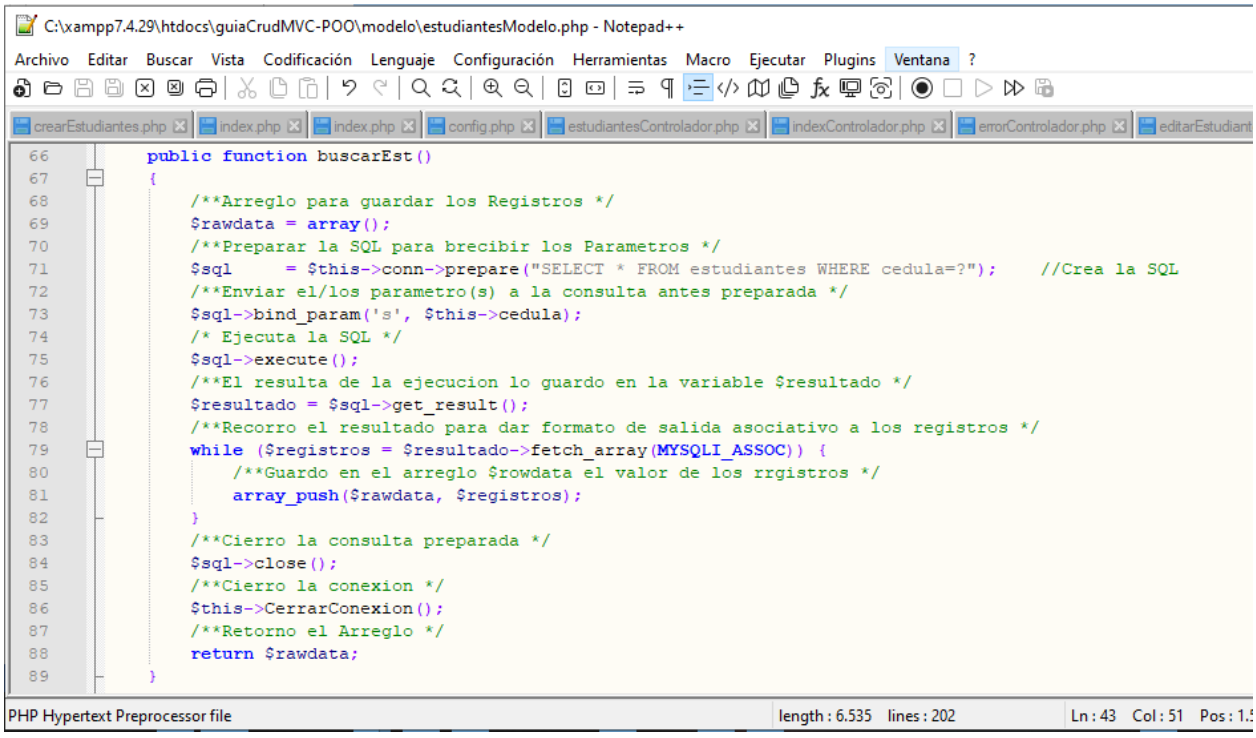
En la línea numero 40: tenemos lo referente a las sentencias preparadas, aquí se indica que se va a preparar la instrucción SQL, para posteriormente enviarles los parámetros. La instrucción es casi igual solo que en el apartado

de VALUE (?,?,?,?,?) colocamos símbolos de interrogación esto indica que esos valores van a ser asignados via parámetros.

En la línea 42 a 49: podemos observar algo referente al tipo de parámetros que recibe el método **bind_param().**

En la línea numero 50: tenemos la asignación de los parámetros usando el método bind_param("tipo de datos", valor1, valor2, valorn), donde tipo de datos, es una carácter que indica el tipo de datos según la tabla mencionada en líneas anteriores. Valor es el valor de los atributos el cual fueron seteados en el controlador.

En la línea numero 53: tenemos la ejecución de la instrucción para ello usamos el método **excute()**, el cual se encarga de ejecutar la instrucción SQL previamente preparada y asignada sus valores. Este método **excute()** devuelve un valor lógico si fue correcto muestra el respectivo mensaje de no ser correcto devuelve error, estos errores se pueden producir por errores de sintaxis, falta de comas, puntos, errores en el tipo de datos, errores de variables no definidas, errores de valores nulos o vacíos.



De igual manera para cada instrucción SQL o cada método de la clase(agergarEst (), buscarEst(), actualizarEst(), eliminarEst()), se utilizan sentencias preparadas, como lo dije anteriormente la clase es la misma solo que se le agrega el prepare(), bind_param(), y el execute(),esto con el fin de evitar inyecciones SQL.

Contenido de la clase mysqli (PHP 5, PHP 7, PHP 8)

```
class mysqli {
        /* Properties */
        public readonly int|string $affected_rows;
        public readonly string $client_info;
        public readonly int $client_version;
        public readonly int $connect_errno;
        public readonly ?string $connect_error;
        public readonly int $errno;
        public readonly string $error;
        public readonly array $error_list;
        public readonly int $field_count;
        public readonly string $host_info;
        public readonly ?string $info;
        public readonly int|string $insert_id;
        public readonly string $server_info;
        public readonly int $server_version;
        public readonly string $sqlstate;
        public readonly int $protocol_version;
        public readonly int $thread_id;
        public readonly int $warning_count;
        /* Methods */
        public __construct(
            string $hostname = ini_get("mysqli.default_host"),
            string $username = ini_get("mysqli.default_user"),
            string $password = ini_get("mysqli.default_pw"),
            string $database = "",
            int $port = ini_get("mysqli.default_port"),
            string $socket = ini_get("mysqli.default_socket")
        )
        public autocommit(bool $enable): bool
        public begin_transaction(int $flags = 0, ?string $name = null): bool
        public change_user(string $username, string $password, ?string $database): bool
        public character_set_name(): string
        public close(): bool
        public commit(int $flags = 0, ?string $name = null): bool
        public connect(
            string $hostname = ini_get("mysqli.default_host"),
            string $username = ini_get("mysqli.default_user"),
            string $password = ini_get("mysqli.default_pw"),
            string $database = "",
            int $port = ini_get("mysqli.default_port"),
            string $socket = ini_get("mysqli.default_socket")
        ): void
        public debug(string $options): bool
        public dump_debug_info(): bool
        public get_charset(): ?object
        public get_client_info(): string
        public get_connection_stats(): array
        public get_server_info(): string
        public get_warnings(): mysqli_warning|false
        public kill(int $process_id): bool
        public more_results(): bool
        public multi_query(string $query): bool
        public next_result(): bool
        public options(int $option, string|int $value): bool
        public ping(): bool
        public static poll(
            ?array &$read,
            ?array &$error,
            array &$reject,
```

```
            int $seconds,
            int $microseconds = 0
        ): int|false
        public prepare(string $query): mysqli_stmt|false
        public query(string $query, int $result_mode = MYSQLI_STORE_RESULT): mysqli_result|bool
        public real_connect(
            string $host = ?,
            string $username = ?,
            string $passwd = ?,
            string $dbname = ?,
            int $port = ?,
            string $socket = ?,
            int $flags = ?
        ): bool
        public real_escape_string(string $string): string
        public real_query(string $query): bool
        public reap_async_query(): mysqli_result|bool
        public refresh(int $flags): bool
        public release_savepoint(string $name): bool
        public rollback(int $flags = 0, ?string $name = null): bool
        public savepoint(string $name): bool
        public select_db(string $database): bool
        public set_charset(string $charset): bool
        public ssl_set(
            ?string $key,
            ?string $certificate,
            ?string $ca_certificate,
            ?string $ca_path,
            ?string $cipher_algos
        ): bool
        public stat(): string|false
        public stmt_init(): mysqli_stmt|false
        public store_result(int $mode = 0): mysqli_result|false
        public thread_safe(): bool
        public use_result(): mysqli_result|false
}
```

Descripción

- mysqli::$affected_rows — Gets the number of affected rows in a previous MySQL operation
- mysqli::autocommit — Turns on or off auto-committing database modifications
- mysqli::begin_transaction — Starts a transaction
- mysqli::change_user — Changes the user of the specified database connection
- mysqli::character_set_name — Returns the current character set of the database connection
- mysqli::close — Closes a previously opened database connection
- mysqli::commit — Commits the current transaction
- mysqli::$connect_errno — Returns the error code from last connect call
- mysqli::$connect_error — Returns a description of the last connection error
- mysqli::__construct — Open a new connection to the MySQL server
- mysqli::debug — Performs debugging operations
- mysqli::dump_debug_info — Dump debugging information into the log
- mysqli::$errno — Returns the error code for the most recent function call
- mysqli::$error_list — Returns a list of errors from the last command executed
- mysqli::$error — Returns a string description of the last error
- mysqli::$field_count — Returns the number of columns for the most recent query
- mysqli::get_charset — Returns a character set object
- mysqli::$client_info — Get MySQL client info
- mysqli::$client_version — Returns the MySQL client version as an integer

- mysqli::get_connection_stats — Returns statistics about the client connection
- mysqli::$host_info — Returns a string representing the type of connection used
- mysqli::$protocol_version — Returns the version of the MySQL protocol used
- mysqli::$server_info — Returns the version of the MySQL server
- mysqli::$server_version — Returns the version of the MySQL server as an integer
- mysqli::get_warnings — Get result of SHOW WARNINGS
- mysqli::$info — Retrieves information about the most recently executed query
- mysqli::init — Initializes MySQLi and returns an object for use with mysqli_real_connect()
- mysqli::$insert_id — Returns the value generated for an AUTO_INCREMENT column by the last query
- mysqli::kill — Asks the server to kill a MySQL thread
- mysqli::more_results — Check if there are any more query results from a multi query
- mysqli::multi_query — Performs one or more queries on the database
- mysqli::next_result — Prepare next result from multi_query
- mysqli::options — Set options
- mysqli::ping — Pings a server connection, or tries to reconnect if the connection has gone down
- mysqli::poll — Poll connections
- mysqli::prepare — Prepares an SQL statement for execution
- mysqli::query — Performs a query on the database
- mysqli::real_connect — Opens a connection to a mysql server
- mysqli::real_escape_string — Escapes special characters in a string for use in an SQL statement, taking into account the current charset of the connection
- mysqli::real_query — Execute an SQL query
- mysqli::reap_async_query — Get result from async query
- mysqli::refresh — Refreshes
- mysqli::release_savepoint — Removes the named savepoint from the set of savepoints of the current transaction
- mysqli::rollback — Rolls back current transaction
- mysqli::savepoint — Set a named transaction savepoint
- mysqli::select_db — Selects the default database for database queries
- mysqli::set_charset — Sets the client character set
- mysqli::$sqlstate — Returns the SQLSTATE error from previous MySQL operation
- mysqli::ssl_set — Used for establishing secure connections using SSL
- mysqli::stat — Gets the current system status
- mysqli::stmt_init — Initializes a statement and returns an object for use with mysqli_stmt_prepare
- mysqli::store_result — Transfers a result set from the last query
- mysqli::$thread_id — Returns the thread ID for the current connection
- mysqli::thread_safe — Returns whether thread safety is given or not
- mysqli::use_result — Initiate a result set retrieval
- mysqli::$warning_count — Returns the number of warnings from the last query for the given link

✓ Contenido de la clase mysqli, fue extraído de Zeal(Zeal is an offline documentation browser for software developers.), puedes descargar desde: https://zealdocs.org/

✓ Descarga el archivo guiaCrudMVC-POO desde: https://www.dropbox.com/s/ps8v9mhza23nd5i/guiaCrudPOO.rar?dl=0

Ing. Alexis Uranga
+58(424)520.65.39 / +58(416)451.52.24
Desarrollo de Portales Web, Aplicaciones Móviles, Cableado Estructurado, Redes en General Cámaras de Seguridad, Consultoría en el Área de Informática, Docente Universitario.
Empresa: mastertradeca.com | **Email:** info@mastertradeca.com
Web : alexisuranga.com.ve | **Twitter** : @alexisuranga | **Facebook** : alexis.uranga
Instagram : alexisuranga.com.ve | **Linkedin** : alexis-uranga