**Lenguaje de programación PHP7x**

retomo mi lenguaje de programación por exelencia.

¿Porqué PHP es mi favotiro para la Web?.... Beuno, pues porque fué el que me abrió las puertas a mis primeros empleos DECENTES.

Por Otro lado, trabajar con PHP es Genial. Uno de sus puntos fuertes es que fué creado específicamente para la Web. es un lenguaje Nativo De internet por lo que No necesitas más que su interprete y jugar con Bases de Datos y HTML5 para hacer magia en la Web de manera rápita y relativamente sencilla.

Es un lenguaje que enamora porque cuando se implementa de la manera correcta (cosa que suelo hacer) se convierte en una tecnología Robusta y muy poderosa. Esto suele suceder cuando implementamos de la mejor manera la división en Capas y la programación orientada a Objetos.

**Proyecto de referencia**

El proyecto se llama “market” y como su nombre lo indica, es una tienda de compra y posterior venta de productos. en Python utilizamos al poderoso IDE Eclipse. Para PHP de momento utilizaremos Netbeans IDE porque su compatibilidad con PHP y su autocompletado es fascinante. Para iniciar con nuestro proyecto de referencia Creamos el proyecto en Netbeans, referenciando la copia de archivos en el directorio que utilizaremos para nuestro repositorio de GitHub.

Para el diseño y posterior creación de la BD utilicé el programa PGModeler 0.7 y la versión 10 del SGBD PostgreSql

Implementamos inicialmente la Base de la división en Capas. Que en este caso nos basamos en el Model-View-Controller. Donde:...

MODEL: se encarga de conectarse al SGB PostgreSql, extraer los datos y/o ingresarlos al Gestor

CONTROLLER: Es la Capa del proyecto que se encarga de “jugar” con los datos que provienen, ya sea de la capa de presentación “View” o del Modelo. En esta capa están por lo general los archivos PHP con las Clases que escriben las consultas, los procediminetos que realizan operaciones lógico-matemáticas y Los archivos PHP de AJAX para funcionalidades asincronas del sistema

VIEW: Es la capa que presenta los resultados finales por medio de HTML5. en esta capa están estrictamente los archivos con extención HTML, plantillas, librerías y frameworks de CSS y HTML como Bootstrap(ojalá tenga la posibilidad de cambiarlo por Bulma), los archivos estáticos de Javascript(.js, la librería JQuery, y otras libreías y plugins de javascript que utilizará la vista de nuestro proyecto) y poco a poco iremos reemplazando a JQery por el Framework VueJs.

Desde luego el proyecto se divide en más capas a medida que va escalando(por ejemplo, dentro del View hay subcapas o directorios internos como public y otros), pero básicamente el MVC es nuestro referente.

Creamos entonces el sistema de directorios del modelo básico. ...

Controller/CRUD:... directorio que contiene de manera general, directorios y archivos con Clases para realizar el “CRUD” de la Base de datos

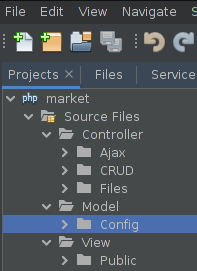
Controller/Ajax:.. archivos PHP con respuesta a las peticiones AJAX que se realizan desde la capa de presentación

View:.. Contiene subdirectorios y archivos con interfáces HTML y archivos .js para consultas dinámicas (AJAX) y realización de reportes.

View/Public:... contiene archivos estáticos del sistema, como Jojas de Estilo, archivos Javascript, Imágenes, Fuentes, Plugins y librerías propias y de terceros.

Model/Config:... Archivos de Configuración del funcionamiento del sistema, Clases con códigos de Conexión al SGBD

Controller/Files:... Archivos subidos desde la capa de presentación al sistema(PDF o archivos multimedia)



Bien. ahora vamos a crear la configuración para poder conectar el proyecto con la BD en PostgreSql...

1. Creamos un archivo en el directorio Model/Config que va a contener una clase con atributos-constantes que guardarán los datos esenciales para acceder al SGBD... ConnectData.php

**class ConnectData**

**{**

**const server\_system = "localhost";**

**const user\_system = "Demonscript";**

**const user\_pass = "developer";**

**const db\_system = "market";**

**const db\_charset = "UTF8";**

**const name\_project = "Market";**

**}**

1. Ahora en el mismo directorio creamos la clase de conexión a PostgreSql. Utilizaremos el método de conexión **PDO** y para conectarnos con PostgreSql tenemos que abrir como administrador el archivo que se encuentra en el la partición del sistema **etc/php/7.0/apache2/php.ini** y habilitar la opción...

**extension=php\_pdo\_pgsql.dll**

y de paso habilité..

**extension=php\_pgsql.dll**

1. Acto seguido, instalamos el driver requerido para que funcione la interacción entre PHP y PostgreSql, de manera que, en la terminal ejecutamos:...

**sudo apt install php7.0-pgsql**

y despues reiniciamos el servicio de Apache con...

**sudo /etc/init.d/apache2 restart**

**NOTA:..** Recordemos estos comando útiles para iniciar, frenar y reiniciar el servicio de apache:...

**sudo /etc/init.d/apache2 start**

**sudo /etc/init.d/apache2 stop**

**sudo /etc/init.d/apache2 restart**

1. creamos entonces la clase **ConnectionDB** en el directorio Model/config. Clase que hereda de ConnectData y en su constructor vamos a disparar la conexión al SGBD, también vamos a implementar en esta Clase, los métodos de ejecución de consultas SQL y los métodos para obtenes respuesta de dichas consultas...

**class ConnectionDB extends ConnectData**

**{**

**//Los aributos privados estáticos solo serán accedidos desde dentro de la clase**

**static public $connex;**

**static private $responseSQL;**

**static private $failure;**

**/\*\***

**\* Connect database using the constructor**

**\* Conectarse a la BD mediante el constructor**

**\*/**

**public function \_\_construct()**

**{**

**//echo 'jjjj <br>';**

**try**

**{**

**self::$connex = new PDO('pgsql:dbname='.parent::db\_system.';**

**host='.parent::server\_system.';**

**user='.parent::user\_system.';**

**password='.parent::user\_pass**

**);**

**//Pejecución para capturar caracteres especiales...**

**self::$connex->exec("SET NAMES ".parent::db\_charset);**

**/\***

**\* // La siguiente es otra conexión que funciona y tiene nuna sintaxis diferente...**

**\* self::$connex = new PDO(**

**"pgsql:dbname=".parent::db\_system.**

**";host=".parent::server\_system,**

**parent::user\_system,**

**parent::user\_pass**

**);**

**\*/**

**//echo "Congratulations Successful Connection!!!";**

**}**

**catch (Exception $failure)**

**{**

**//echo "error de acceso a PostgreSql...<br> ".$failure->getMessage();**

**header('Location: ../View/error.php?message='.$failure->getMessage());**

**exit();**

**}**

**}**

**/\*\***

**\* SQL procedure that returns a complete content**

**\* Procedimiento SQL que retorna un contenido Completo**

**\* @param type $sql\_code**

**\* @return None**

**\*/**

**protected function general\_query($sql\_code)**

**{**

**$ejecutar = self::$connex->query($sql\_code);**

**//echo 'ejecutada SQL<br>';**

**self::$responseSQL = $ejecutar;**

**//echo 'consulta '.self::$connex->setAttribute(PDO::ATTR\_ERRMODE, PDO::ERRMODE\_EXCEPTION);**

**}**

**/\*\***

**\* Este método posiblemente desaparezca porque en Postgresql se obtiene el ultimo**

**\* ID en la misma consulta de insertar o de seleccionar.**

**\* @param type $sql\_code**

**\* @param type $id**

**\*/**

**protected function simple\_query($sql\_code,$id)**

**{**

**$query = self::$connex->query($sql\_code);**

**{**

**$query = false;**

**}**

**}**

**/\*\***

**\* procedure that returns SQL response as an object**

**\* Procedimiento que devuelve la respuesta de SQL Como un Objeto**

**\* @return string, Object or boolean**

**\*/**

**protected function get\_response()**

**{**

**if(self::$responseSQL === FALSE)**

**{**

**self::$failure = self::$connex->errorInfo();**

**}**

**return self::$responseSQL;**

**}**

**/\*\***

**\* Procedure that returns an SQL Error**

**\* Procedimiento que retorna un Error de SQL**

**\* @return string**

**\*/**

**protected function get\_error()**

**{**

**return self::$failure;**

**}**

**protected function break\_connection()**

**{**

**unset(self::$connex);**

**}**

**}**

1. Hecho eso, ahora creamos la Clase preliminar(seguramente le añadiremos más funcionalidades) de las consultas que podemos reutilizar desde todos los ámbitos de nuestro sistema, el archivo y clase CommonSQLQueries.php en el directorio Model...

La clase CommonSQLQueries hereda de la Clase ConnectionDB para poderse comunicar con el SGBD PostgreSql. Su código será inicialmente el sig...

**require\_once './Config/ConnectionDB.php';**

**class CommonSQLQueries extends ConnectionDB**

**{**

**public function \_\_construct()**

**{**

**parent::\_\_construct();**

**}**

**/\*\***

**\* Procedure To select all Records in a table**

**\* Procedimiento para seleccionar todos los registros de una tabla**

**\* @param string $table\_name**

**\* @return None**

**\*/**

**public function select\_all($table\_name)**

**{**

**$query = "SELECT \* FROM $table\_name ORDER BY id DESC";**

**parent::general\_query($query);**

**}**

**/\*\***

**\* procedure to select a record from a table**

**\* Procedimiento para seleccionar un registro de una tabla**

**\* @param string $table\_name**

**\* @param string $field\_name**

**\* @param int $field\_id**

**\* @return None**

**\*/**

**public function select\_by\_ID($table\_name,$field\_name,$field\_id)**

**{**

**$query = "SELECT \* FROM $table\_name WHERE $field\_name = $field\_id";**

**parent::general\_query($query);**

**}**

**/\*\***

**\* Procedure to select all records in a table according to their status**

**\* Procedimiento para seleccionar registros de una tabla según su estado**

**\* @param String $table\_name**

**\* @param Bool $state**

**\* @return None**

**\*/**

**public function select\_by\_state($table\_name,$state)**

**{**

**$query = "SELECT \* FROM $table\_name WHERE status = $state";**

**parent::general\_query($query);**

**}**

**/\*\***

**\* Procedure to Update a Record Status**

**\* Procedimiento para actualizar el estado de un registro**

**\* @param string $table\_name**

**\* @param string $name\_fieldID**

**\* @param int $fieldID**

**\* @param bool $state**

**\* @return None**

**\*/**

**public function update\_state($table\_name,$name\_fieldID,$fieldID,$state)**

**{**

**$query="UPDATE $table\_name SET status='$state' WHERE $name\_fieldID ='$fieldID'";**

**parent::general\_query($query);**

**}**

**/\*\***

**\* \* procedure that gets an response**

**\* Procedimiento para obtener una respuesta**

**\* @return object, string or boolean.**

**\*/**

**public function get\_response()**

**{**

**return parent::get\_response();**

**}**

**/\*\***

**\* Procedure that gets an error**

**\* Procedimineto para obtener un error**

**\* @return string**

**\*/**

**public function get\_error()**

**{**

**return parent::get\_error();**

**}**

**/\*\***

**\* Procedure that request to break the connection with the DBMS**

**\* Procedimiento que solicita romper la conexión con el SGBD**

**\* @return Null**

**\*/**

**public function break\_connection()**

**{**

**parent::break\_connection();**

**return null;**

**}**

**}**

Este código funciona en su totalidad. ha sido probado de manera interna.

1. Creamos también el primer archivo o clase independiente, para consultas referentes a la tabla “category”. La clase Category en el directorio Model...

**class Category extends ConnectionDB**

**{**

**public function \_\_construct()**

**{**

**parent::\_\_construct();**

**}**

**/\*\***

**\* Procedure to create a New category**

**\* @param string $name**

**\* @param string $description**

**\* @return None**

**\*/**

**public function new\_cat($name,$description)**

**{**

**$query="INSERT INTO category (name, description) VALUES ('$name', '$description');";**

**parent::general\_query($query);**

**}**

**/\*\***

**\* Procedure to update caregory**

**\* Procedimiento para actualizar categoría**

**\* @param int $id\_ctgry**

**\* @param string $nm**

**\* @param string $dscrptn**

**\*/**

**public function update\_cat($id\_ctgry,$nm,$dscrptn)**

**{**

**$query = "UPDATE category SET name ='$nm', description = '$dscrptn' WHERE id ='$id\_ctgry'";**

**parent::general\_query($query);**

**}**

**/\*\***

**\* procedure that gets an response**

**\* Procedimiento para obtener una respuesta**

**\* @return array, Object or Boolean**

**\*/**

**public function get\_response()**

**{**

**return parent::get\_response();**

**}**

**/\*\***

**\*\* Procedure that gets an error**

**\* @return string**

**\*/**

**public function get\_error()**

**{**

**return parent::get\_error();**

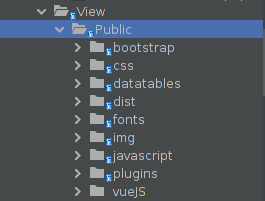
**}**

**}**

Código probado de manera interna. Funciona.

ya tenemos nuestra estructura básica del Modelo M-V-C y algunas clases ya implementadas para ensayar la conexión con PostgreSql, que de momento funciona muy bien. Ahora vamos a implementar los archivos estáticos como , Plantillas HTML, el Framework Bootstrap, Librerías de Javascript, etc...

1. en primera instancia me dirijo al directorio View y creo el sig sistema de archivos...



1. Ahora, ubico los archivos de mi platilla que corresponden a cada directorio y los pego.
2. acto seguido ponemos el archivo listado.html directamente en el directorio View
3. vamos a editar el archivo listado.html para que tome los archivos estaticos. Por ej.

**<link rel="stylesheet" href="Public/css/bootstrap.min.css">**

**<!-- Font Awesome -->**

**<link rel="stylesheet" href="Public/css/font-awesome.css">**

**<!-- Theme style -->**

**<link rel="stylesheet" href="Public/css/AdminLTE.min.css">**

Y hago lo mismo con el resto de llamados, tanto en el Header como el la estructura del footer.

Por ultimo modifico algunos textos como en la etiqueta Title y otros, y cambio algunos iconos.

por alguna razón que desconozco, no se carga una ventanita al dar clic en el icono del lado derecho. Esto se soluciona eliminando la etiqueta ...

**<div class="wrapper">**

y eliminamos también su respectivo cierre **</div>**

Ahora verifico en el navegador que la página listado.html carga todos los archivos estáticos de manera que se vea toda la interfaz grafica correctamente.

Ahora vamos a organizar y dividir el contenido estático de que será reutilizable...

1. en el directorio View creamos dos archivos PHP que son header y footer
2. Tomamos el contenido del archivo listado.html desde su inicio hasta la etiqueta de cierre que está antes antes del comentario **<!--Contenido-->** y pegamos ese código en el archivo **header.php** después de cerrar su etiqueta PHP
3. ahora copiamos el código del archivo listado.html a partir de la etiqueta **<footer>** hasta el final del archivo y pegamos dicho código en el archivo **footer.php** después de cerrar su etiqueta php
4. ahora vamos acrear un archivo de ensayo en el dir View, el cual llamaremos listar.php

al principio de este archivo hacemos un requre al header..

**require\_once './header.php';**

cerramos las etiquetas de PHP

reabrimos las etiquetas de PHP e incluimos dentro el require hacia el footer...

**<?php**

**require\_once './footer.php';**

1. Ahora tomamos el código de listado.html que pertenece al contenido y pegamos dicho código en medio del esquema de etiquetas PHP que acabamos de implementar en el archivo listar.php...

**<!--Contenido-->**

**<!-- Content Wrapper. Contains page content -->**

**<div class="content-wrapper">**

**....**

**</div><!-- /.content-wrapper -->**

**<!--Fin-Contenido-->**

Lanzamos el archivo listar.php y vemos que, al igual que listado.html, se carga correctamente todo el contenido Web.

Personalizamos el sitio con nuestras imégenes y eliminamos las imágenes sobrantes, corto las imágenes de public/dist/img

y las pego en el dir public/img

Luego, eliminamos public/dist/img Porque no veo necesario ese directorio. Y modifico los llamados del antiguo dir en header.php y footer.php a dichas imágenes

Ahora estamos listos para la Magia de AJAX y JQuery...

1. En el directorio Controller/Ajax creamos el archivo Category.php para implementar el sig código...

$view\_data = $\_POST;

if(empty($view\_data))

{

exit('No hay Datos para procesar');

}

//capturar el valor de la clave "option"

//capture the value of the "option" key

$option = $view\_data["option"];

//Condicionar el contenido de la var "option" para obtener la herencia requerida

//Condition the content of the "option" variable to obtain the required inheritance

/\*\*#@+

\* Bloque condicional de la variable $opción para heredar de la clase CommonSQLSQueries

\* Conditional block of variable $option to inherit from CommonSQLSQueries class

\*/

if($option=='select\_all' || $option=='by\_id' || $option=='by\_state' || $option=='update\_state' || $option=='repeat')

{

require\_once '../Model/CommonSQLQueries.php';

/\*

\* Clase que hereda de CommonSQLQueries

\* class inherit to CommonSQLQueries

\*/

class AjaxCategory extends CommonSQLQueries

{

/\*\*

\* Datos de la capa View destinados a la Clase CommonSQLQueries

\*/

static private $data\_process;

/\*\*

\* recibir los datos requeridos para el proceso, mediante el constructor

\* receive the data required for the process, through the constructor...

\* posteriormente, se lanza la conexión con el SGBD

\* subsequently, the connection with the DBMS is launched

\* finalmente, los datos requeridos se asignan al atributo $view\_data

\* finally, the required data is assigned to the $view\_data attribute

\* @param array() $view\_data

\*/

public function \_\_construct($view\_data)

{

parent::\_\_construct();

self::$data\_process = $view\_data;

}

/\*\*

\* Método para enviar datos al Modelo

\* Method to send data to Model

\*/

public function process()

{

/\*

\* capturar los datos del formulario en variables independientes

\* capture form data into independent variables...

\* @var string option

\* @var string search\_words

\* @var string search\_field\_name

\* @var int id\_cat

\* @var bool actual\_state

\* @var bool new\_state

\*/

$option = self::$data\_process["option"];

$search\_words = self::$data\_process["words"];

$search\_field\_name = self::$data\_process["search\_field\_name"];

$id\_cat = self::$data\_process["id\_cat"];

$actual\_state = self::$data\_process["actual\_state"];

$new\_state = self::$data\_process["new\_state"];

$field\_PK = "id";

switch ($option)

{

case "select\_all":

parent::select\_all('category');

break;

case "by\_id":

parent::select\_by\_ID('category', $search\_field\_name, $id\_cat);

break;

case "repeat":

parent::count\_by\_field('category', $search\_field\_name, $search\_words);

break;

case "by\_state":

parent::select\_by\_state('category', $actual\_state);

break;

default:

parent::update\_state('category', $field\_PK, $id\_cat, $new\_state);

break;

}

}

/\*\*

\* procedimiento relacionado con la clase CommonSQLQueries para obtener una respuesta

\* procedure related to the CommonSQLQueries class to get a response

\* @return object or @return array() or @return string or @return boolean

\*/

public function get\_response()

{

return parent::get\_response();

}

/\*\*

\* procedimiento relacionado con la clase CommonSQLQueries para obtener un error

\* procedure related to CommonSQLQueries class to get error

\* @return string

\*/

public function get\_error()

{

return parent::get\_error();

}

/\*\*

\* procedimiento relacionado con la clase CommonSQLQueries que solicita romper la conexión con el SGBD

\* procedure related to CommonSQLQueries class that requests to break the connection with the DBMS

\*/

public function break\_connection()

{

parent::break\_connection();

}

// Porqué el método anterios al utilizar autocmpletado me salía así???...

// public function break\_connection(): \Null {

// parent::break\_connection();

// }

} //end Class

//Creación del objeto para ejecutar la solicitud del form...

$ObjectCat = new AjaxCategory($view\_data);

$ObjectCat->process();

$response = $ObjectCat->get\_response();

if($response==FALSE || $response==NULL)

{

$response = $ObjectCat->get\_error();

}

if($option=='by\_id' || $option=='repeat')

{

//utilizamos JSon para codificar mediante clave-Valor un registro

$response = json\_encode($response);

}

/\*\*#@+

\* obtener un array de datos a partir de un objeto

\* get an array of data from an object

\*/

if ($option=='select\_all' || $option=='by\_state')

{

$dataset = Array();

while($data\_row=$response->fetchobject())

{

//echo '<br>'.$data\_row->name;

if($data\_row->status=='1')

{

$status = 1;

}

else

{

//echo '0';

$status = 0;

}

$dataset[] = array

(

//implemento una bifurcación IF:...

// "0"=>($status)?:

// De manera que lo correspondiente a true va a continuación del signo de pregunta ?

// y el código que va después de los dos puntos : es para cuando la condición no se cumple:...

"0"=>$status,

"1"=>$data\_row->name,

"2"=>$data\_row->description,

"3"=>$data\_row->status

);

}//end while

// Configuramos la información para el DataTable.

$DataTable = array

(

"sEcho"=>1,

"iTotalRecords"=> count($dataset),//num de registros obtenidos

"iTotalDisplayRecords"=> count($dataset),//num de registros a mostrar

"aaData"=>$dataset

);

$response = json\_encode($DataTable);

}

/\*#@+

\* end of elseif

\*/

$ObjectCat->break\_connection();

echo $response;

}

/\*\*#@-

\* Fin del Condicional superior

\*/

/\*\*#@+

\* Bloque condicional de la variable $opción para heredar de la clase Category

\* Conditional block of variable $option to inherit from Category class

\*/

else

{

require\_once '../../Model/Category.php';

class AjaxCategory extends Category

{

/\*\*

\* Datos de la capa View destinados a la Clase Category

\*/

static private $data\_process;

/\*\*

\* recibir los datos requeridos para el proceso, mediante el constructor

\* receive the data required for the process, through the constructor...

\* posteriormente, se lanza la conexión con el SGBD

\* subsequently, the connection with the DBMS is launched

\* finalmente, los datos requeridos se asignan al atributo $view\_data

\* finally, the required data is assigned to the $view\_data attribute

\* @param array() $view\_data

\*/

public function \_\_construct($view\_data)

{

parent::\_\_construct();

self::$data\_process = $view\_data;

}

/\*\*

\* Método para enviar datos al Modelo

\* Method to send data to Model

\*/

public function process()

{

/\*

\* capturar los datos del formulario en variables independientes

\* capture form data into independent variables...

\*

\*/

$name\_cat = self::$data\_process["name\_cat"];

$description = self::$data\_process["description"];

$id\_cat = self::$data\_process["id\_cat"];

if($id\_cat=="")

{

parent::new\_cat($name\_cat, $desciption);

}

else

{

parent::update\_cat($id\_cat, $name\_cat, $description);

}

}

/\*\*

\* procedimiento relacionado con la clase Category para obtener una respuesta

\* procedure related to the Category class to get a response

\* @return object or @return array() or @return string or @return boolean

\*/

public function get\_response()

{

return parent::get\_response();

}

/\*\*

\* procedimiento relacionado con la clase Category para obtener un error

\* procedure related to Categoryry class to get error

\* @return string

\*/

public function get\_error()

{

parent::get\_error();

}

/\*\*

\* procedimiento relacionado con la clase Category que solicita romper la conexión con el SGBD

\* procedure related to Category class that requests to break the connection with the DBMS

\*/

public function break\_connection()

{

parent::break\_connection();

}

}

$ObjectCat = new AjaxCategory($view\_data);

$ObjectCat->process();

$response = $ObjectCat->get\_response();

if($response==false || $response==null)

{

$response = $ObjectCat->get\_error();

}

else

{

$response="Successful";

}

$ObjectCat->break\_connection();

echo $response;

}

El código es un poco extenso, pero está bien documentado y Comentado, basta con pasarlo al IDE e identarlo correctamente, para analizarlo con dedicación. Desde luego será modificado muy pronto, es solo la Base Ajax inicial para la tabla category de nuestra DB

Podemos verificar su funcionamiento, cambiando la var **$view\_data = $\_POST** por **$view\_data = $\_GET**

Nos dirigimos al Navegador Firefox y visitamos la dirección...

**http://localhost/market/Controller/Category.php?option=select\_all**

Lo que nos debe mostrar en una matrix todo el contenido de la Tabla category

OJO CON EL SISTEMA DE ORGANIZACION DE LOS ARCHIVOS para el tema de implementación del require\_once Recuerda que todas las rutas deben quedar igual inicialmente. De lo contrario no funciona el llamado.

Ahora vamos acrear el primer archivo para la interacción AJAX. En el directorio **View/javascript** creamos el archivo **category.js** ...

1. en este archivo implementaremos el código Javascript que realizará las Peticiones al servidor utilizando AJAx...

explicamos a continuación y de manera general el código del archivo category.js y al final mostramos dicho código

Inicialmente declaramos una variable que alamcenará los datos que obtenemos en el dataTables.

después implementamos la FX “**initial**” que se va a ejecutar al cargar la página, dicha FX llamaŕa a una FX que oculta el formulario de ingresar categorías, y después llama a una FX que carga y muestra el total de las categorías existentes en la BD.

acto seguirdo, implementamos la FX “**clean\_fields**” la cual llamará a los objetos HTML del formulario y limpiará su contenido.

Implementamos después la FX “**show\_form**”, función que, inicialmente limpiará los campos del formulario llamando a “**clean\_fields**” y después, de acuerdo al valor del parámetro que recibe (true o false) mostrará el formulario o la lista completa de las categorías, esto lo hace llamando al ID contenedor de dichos elementos.

Ahora implementamos la FX “cancel\_form” para cancelar el proceso de insertar nueva categoría. Esta FX limpia los campos llamando a “**clean\_fields**” y oculta el formulario, llamando a “**show\_form(false)**”

implementamos ahora la FX “**show\_records**” que realiza la primera solictud AJAX al controlador de category.php ...

utilizamos dntro de esta FX la variable **table** a la cual le indicamos que hay un objeto html con Id record\_table al cual le asignamos unos valores mediante DataTable...

En la FX “datatable” definimos los parámetros necesarios para cargar la información requerida y la metodología AJAX utilizada para el procedimiento.

Al final de todo el código llamamos a la FX inicial.

aquí el código respectivo...

var table;

/\*\*

\* función que se ejecuta con la carga inicial del archivo PHP

\* function that is executed with the initial load of the PHP file

\* @returns {undefined}

\*/

function initial()

{

show\_form(false);

show\_records();

}

/\*\*

\* Función que deja los campos de los elementos HTML en blanco mediante JQuery...

\* Function that clears HTML element fields using JQuery

\* @returns {none}

\*/

function clean\_fields()

{

//llamamos a los objetos HTML por su id...

$("#name").val("");

$("#id\_cat").val("");

$("#description").val("");

}

/\*\*

\* función que muestra el formulario o la lista de datos

\* function that displays the form or data list

\* @param {boolean} signal

\* @returns {none}

\*/

function show\_form(signal)

{

//limpiar contenido de los objetos HTML...

clean\_fields();

if(signal===true)

{

//Ocultar registros listados

$("#list\_records").hide();

//mostrar formulario de ingreso

$("#new\_record").show();

//PENDIENTE este botón los podemo activar de forma interactiva con las cajitas de texto:...

$("#save").prop("disabled",false);

}

else

{

//mostrar registros listados

$("#list\_records").show();

//Ocultar formulario de ingreso

$("#new\_record").hide();

}

}

/\*\*

\* cancelar envío de datos

\* cancel sending data

\* @returns {none}

\*/

function cancel\_form()

{

clean\_fields();

show\_form(false);

}

/\*\*

\* función para obtener registros mediante AJAX

\* function to get records using AJAX

\* @returns {none}

\*/

function show\_records()

{

table=$("#record\_table").dataTable

(

{

"aProcessing":true, //prcesamiento de datos activo

"aServerSide":true,//paginación y filtrado activo desde el servidor

dom: 'Bfrtip',//definir elementos de control de la tabla

buttons:// definnimos botones de control para exportar registors a formatos predefinidos..

[

'copyHtml5',

'excelHtml5',

'csvHtml5',

'pdf'

],

"ajax":

{

url:'../Controller/Category.php',

type:"POST",

dataType:"json",

data:{"option":'select\_all'},

error:function(e)// en caso de presentar error

{

console.log(e.responseText);

}

},

"bDestroy":true,

"iDisplayLength":5,//paginación limitada a 5 registros

"order":[[0,"asc"]]//organizar registros de manera ascendente por el campo id

}

).DataTable();

}

initial(); //llamamos a initial en la carga del archivo

1. Ahora vamos a implementar la interfaz gráfica para hacer el CRUD de la tabla category, utilizando Bootstrap, la librería JQuery, la librería DataTable y otras librerías auxiliares con los archivos que ya implementamos para conectarnos mediante AJAX (**view/javascript/category.js** y **controller/category.php** que a su vez está conectada a **CommonSQLQueries.php** y a **Category.php** que están en la capa Model)
2. verificamos que en el hader tenemos el llamado a los archivos CSS requeridos para la correcta funcionalidad de DataTables
3. también verificamos que el llamado a los archivos JS de dtaTables están siendo llamados desde el footer
4. tenemos el archivo listar.php el cual vamos a renombrar a category.php porque la utilizaremos para hacer el crud a dicha tabla
5. cambiamos el texto de la etiqueta h1 de “Texto” a “Categorías”. Podemos ver que seguido al h1 está el botón que llama mediante el evento onclick a la FX que muestra el formulario, de manera que digitamos la respectiva Fx “show\_form(true)” y cambiamos su texto de “Agregar” por “Nueva”
6. buscamos mas abajo el elemento html que tiene como id “listar registros” y cambiamos su nombre de id a “list\_records” como hemos designado en el archivo category.js, tambien eliminamos el style porque la tabla con DataTables tiene sus medidas justas y será responsive
7. duplicamos el anterior elemento y en su id digitamos “new\_record”, y en su class eliminamos la clase “table-responsive” porque no vamos a utilizar tabla en el form.

Podemos ver el resultado en el navegador, agregando texto para cada uno de los div, así...

<div class="panel-body table-responsive" id="list\_records">

**listado**

</div>

<div class="panel-body" style="height: 400px;" id="new\_record">

**el form**

</div>

Podemos ver en firefox que al cargar la página se visualiza el texto “listado”, pero al presionar el botón “Nueva” se oculta “listado” y se muestra “el form” señal de que la FX están funcionando.

1. vamos ahora a implementar la estructura html (una tabla) para mostrar los registros de la tabla category. Esto lo hacemos dentro del elemento html con id=“list\_records” donde implementamos...

la etiqueta <table> con id =”record\_table” Y algunas clases que modifican mediante CSS su apariencia, y dentro de ella la etiqueta <thead> donde implementaremos los <th> correspondientes, uno para cara fila de nuestra tabla, a saber:.. Option, Name, Description, Status

fuera del <thead> implementamos <tbody> y dejamos su implementación sin contenido porque ese contenido lo mostraremos mediante DataTable.

finalmente y fuera del <tbody> implementamos <tfooter> y como contenido implementamos los mismos <th> de la cabecera, con filas para Option, Name, Description, Status. El código HTML será...

<table id="record\_table" class="table table-striped table-bordered table-condensed table-hover">

<thead>

<th>Options</th><th>Name</th><th>Description</th><th>Status</th>

</thead>

<tbody>

</tbody>

<tfoot>

<th>Options</th><th>Name</th><th>Description</th><th>Status</th>

</tfoot>

</table>

Con el código anterior podemos ejecutar el archivo View/category.php y vemos el resultado parcial (el listado de los registros que se genera mediante AJAX).

Posteriormente modificaremos este archivo para aplicarle funcionalidades, pero de momento vemos que funciona perfectamente la solicitud AJAX mediante JQuery y DtaTables.

Además podemos verificar que el filtro de busqueda que nos aporta DataTables también funciona, así como la paginación de los registros y los controles para ordenar por Name, Description, etc. Otro detalle importante y que funciona es la exportación a diferentes formatos a partir de los botones que están en la parte superior del resultado de registros :)

Ahora vamos a crear el formulario y utilizaremos nuestro archivo category.js para insertar una nueva categoría mediante JQery y AJAX...

1. nos ubicamos dentro del div con id=”new\_record”, precisamente donde tenemos el texto “el form”. Para implementar el código HTML que va a generar nuestro formulario...

ingresamos la etiqueta <form> con id y name “insert\_record” y method=”POST”

1. Dentro del formulario creamos un <div> con class= “form-group col-lg-6 col-md-6 col-sm-12” (el col-lg-6 es para hecrlo responsive, recordemos que la grilla es hasta de 12 columnas de manera que le indicamos que cuando sea pequeño, utilice las 12 celdas... el tamaño total de la pantalla)
2. dentro del div implementamos las etiquetas y los campos de texto correspondientes para cada dato, ya sean visibles o en el caso necesario.. ocultos.

Las etiquetas correspondientes a name y description tienen un maxlength de 50 y 256 respectivamente porque así está especifoicado en la BD. Aquí el código...

los iconos son generados por font awesome. Podemos obtener la sintaxis de cada icono buscando en google por “font awesome”.

aquí el código HTML..

**<form name="insert\_record" id="insert\_record" method="POST">**

**<div class="form-group col-lg-6 col-md-6 col-sm-12">**

**<label>Nombre:</label>**

**<i class="fa fa-angle-double-down"></i>**

**<input type="text" class="form-control" id="name" name="name" maxlength="50" placeholder="Nombre de la categoría" required>**

**<input type="hidden" id="id\_cat" name="id\_cat" value="">**

**<input type="hidden" id="id" name="id" value="">**

**</div>**

**<div class="form-group col-lg-6 col-md-6 col-sm-6 col-xs-12">**

**<label>Descripción:...</label><i class="fa fa-angle-double-down"></i>**

**<input type="text" class="form-control" id="description" name="description" maxlength="256" placeholder="Descripción de la Categoría">**

**</div>**

**<div class="form-group col-lg-12 col-md-12 col-sm-12 col-xs-12">**

**<button class="btn btn-success" id="save" type="submit">**

**<i class="fa fa-cloud-upload"></i> Insertar**

**</button>**

**<button class="btn btn-danger" id="cancel" type="button" onclick="cancel\_form()">**

**<i class="fa fa-thumbs-down"></i> Cancelar**

**</button>**

**</div>**

**</form>**

Importante: recordemos que en el archivo .js generalmente trabajamos con los **id** de cada elemento. Pero el archivo php de la capa Controller reconoce es el **name** de cada elemento. Asi, es recomendable poner en el código HTML el mismo valor en **id** tanto como en **name**

antes de iniciar con el código javascript del formulario, faltaba pegar en View/public/css el archivo **jquery.toast.css** el cual estaba en el antiguo proyecto y sirve junto a **jquery.toast.js** para generar los Alert bonitos :)

ahora agregamos, al header el llamado a **jquery.toast.css** y en el footer el llamado a **jquery.toast.js**

también puedo utilizar bootbox como alert, pero me gusta mas toast

Vamos a digitar el código Javascript para controlar el formulario de insertar nueva categoría.

1. Nos dirigimos al archivo category.js y añadimos la sig Fx...

function save\_edit(e)

{

e.preventDefault();//desactriva la ejecución normal del evento del botón

$("#save").prop("disabled",true);

var form\_data = new FormData($("#insert\_record")[0]);

//petición AJAX:..

$.ajax(

{

url:'../Controller/Category.php?option=save\_edit',

type:"POST",

data:form\_data,

contentType: false,

processData: false,

//cuando todo se ejecuta correctamente, recibo la respuesta del servidor...

success: function (response)

{

if(response === 'Successful')

{

$.toast({

heading: 'Succes',

text: 'Proceso Exitoso!!!...',

textAlign: 'center',

icon: 'success',

loader: true, // Change it to false to disable loader

position: 'mid-center',

showHideTransition: 'slide',

hideAfter: 1500

//loaderBg: '#9EC600' // To change the background

});

//bootbox.alert("Proceso exitoso!!!...");

}

else

{

//$.toast('Error!!!... '+response);

$.toast({

heading: 'Warning',

text: response,

textAlign: 'center',

icon: 'error',

loader: true, // Change it to false to disable loader

position: 'mid-center',

showHideTransition: 'slide',

hideAfter: 6000

//loaderBg: '#9EC600' // To change the background

});

//bootbox.alert(response); //muestro la respuesta

}

show\_form(false);

table.ajax.reload();

}

}

);

clean\_fields()(); //limpiar los campos del formulario

}

bootbox y toast son las librerías que utilizamos (en este caso Toast) para los alert.

adicional a esto, agregamos un evento que muestra un alert cuando se ingresa un nombre de categoría que ya existe...

Al código html del formulario en el control cin id=”name” agregamos...

<input type="text" class="form-control" id="name" name="name" maxlength="50" placeholder="Nombre de la categoría" **onblur=message(this.value) required**>

1. Y en el archivo category.js ingresamos la sig Fx...

function message(nm\_cat)

{

$.post(

"../Controller/Category.php",

{option:'repeat',word:nm\_cat,search\_field:'name'},

function (data,status)

//data es el valor obtenido del servidor mediante POST

//contiene el estado de la solicitudstatus("success", "notmodified", "error", "timeout", or "parsererror")

{

if(data>= 1)

{

$.toast({

heading: 'Warning',

text: 'Esta Categoría ya existe !!!...',

textAlign: 'center',

icon: 'error',

loader: true, // Change it to false to disable loader

position: 'mid-center',

showHideTransition: 'slide',

hideAfter: 2000

//loaderBg: '#9EC600' // To change the background

});

}

}

);

}

Desde luego que se puede mejorar el código. Pero de momento lo dejamos así.

1. Aplicamos algunos cambios en el código Controller/category.php...

$view\_data = $\_POST;

if(empty($view\_data))

{

exit('No hay Datos para procesar');

}

//capturar el valor de la clave "option"

//capture the value of the "option" key

$option = $view\_data["option"];

//Condicionar el contenido de la var "option" para obtener la herencia requerida

//Condition the content of the "option" variable to obtain the required inheritance

/\*\*#@+

\* Bloque condicional de la variable $opción para heredar de la clase CommonSQLSQueries

\* Conditional block of variable $option to inherit from CommonSQLSQueries class

\*/

if($option=='select\_all' || $option=='by\_id' || $option=='by\_state' || $option=='update\_state' || $option=='repeat')

{

require\_once '../Model/CommonSQLQueries.php';

/\*

\* Clase que hereda de CommonSQLQueries

\* class inherit to CommonSQLQueries

\*/

class AjaxCategory extends CommonSQLQueries

{

/\*\*

\* Datos de la capa View destinados a la Clase CommonSQLQueries

\*/

static private $data\_process;

/\*\*

\* recibir los datos requeridos para el proceso, mediante el constructor

\* receive the data required for the process, through the constructor...

\* posteriormente, se lanza la conexión con el SGBD

\* subsequently, the connection with the DBMS is launched

\* finalmente, los datos requeridos se asignan al atributo $view\_data

\* finally, the required data is assigned to the $view\_data attribute

\* @param array() $view\_data

\*/

public function \_\_construct($view\_data)

{

parent::\_\_construct();

self::$data\_process = $view\_data;

}

/\*\*

\* Método para enviar datos al Modelo

\* Method to send data to Model

\*/

public function process()

{

/\*

\* capturar los datos del formulario en variables independientes

\* capture form data into independent variables...

\* @var string option

\* @var string search\_words

\* @var string search\_field\_name

\* @var int id\_cat

\* @var bool actual\_state

\* @var bool new\_state

\*/

$option = self::$data\_process["option"];

$search\_words = self::$data\_process["word"];

$search\_field\_name = self::$data\_process["search\_field"];

$id\_cat = self::$data\_process["id\_cat"];

$actual\_state = self::$data\_process["actual\_state"];

$new\_state = self::$data\_process["new\_state"];

$field\_PK = "id";

switch ($option)

{

case "select\_all":

parent::select\_all('category');

break;

case "by\_id":

parent::select\_by\_ID('category', $search\_field\_name, $id\_cat);

break;

case "repeat":

parent::count\_by\_field('category', $search\_field\_name, $search\_words);

break;

case "by\_state":

parent::select\_by\_state('category', $actual\_state);

break;

default:

parent::update\_state('category', $field\_PK, $id\_cat, $new\_state);

break;

}

}

/\*\*

\* procedimiento relacionado con la clase CommonSQLQueries para obtener una respuesta

\* procedure related to the CommonSQLQueries class to get a response

\* @return object or @return array() or @return string or @return boolean

\*/

public function get\_response()

{

return parent::get\_response();

}

/\*\*

\* procedimiento relacionado con la clase CommonSQLQueries para obtener un error

\* procedure related to CommonSQLQueries class to get error

\* @return string

\*/

public function get\_error()

{

return parent::get\_error();

}

/\*\*

\* procedimiento relacionado con la clase CommonSQLQueries que solicita romper la conexión con el SGBD

\* procedure related to CommonSQLQueries class that requests to break the connection with the DBMS

\*/

public function break\_connection()

{

parent::break\_connection();

}

// Porqué el método anterios al utilizar autocmpletado me salía así???...

// public function break\_connection(): \Null {

// parent::break\_connection();

// }

} //end Class

//Creación del objeto para ejecutar la solicitud del form...

$ObjectCat = new AjaxCategory($view\_data);

$ObjectCat->process();

$response = $ObjectCat->get\_response();

if($response==FALSE || $response==NULL)

{

$response = $ObjectCat->get\_error();

}

**if($option=='by\_id')**

**{**

**//utilizamos JSon para codificar mediante clave-Valor un registro**

**//$response = json\_encode($response);**

**$response = $response->fetchobject();**

**}**

**if($option=='repeat')**

**{**

**//utilizamos JSon para codificar mediante clave-Valor un registro**

**//$response = json\_encode($response);**

**$number = $response->fetchobject();**

**$response = $number->number;**

**}**

/\*\*#@+

\* obtener un array de datos a partir de un objeto

\* get an array of data from an object

\*/

if ($option=='select\_all' || $option=='by\_state')

{

$dataset = Array();

while($data\_row=$response->fetchobject())

{

//echo '<br>'.$data\_row->name;

if($data\_row->status=='1')

{

$status = 1;

}

else

{

//echo '0';

$status = 0;

}

$dataset[] = array

(

//implemento una bifurcación IF:...

// "0"=>($status)?:

// De manera que lo correspondiente a true va a continuación del signo de pregunta ?

// y el código que va después de los dos puntos : es para cuando la condición no se cumple:...

"0"=>$status,

"1"=>$data\_row->name,

"2"=>$data\_row->description,

"3"=>$data\_row->status

);

}//end while

// Configuramos la información para el DataTable.

$DataTable = array

(

"sEcho"=>1,

"iTotalRecords"=> count($dataset),//num de registros obtenidos

"iTotalDisplayRecords"=> count($dataset),//num de registros a mostrar

"aaData"=>$dataset

);

$response = json\_encode($DataTable);

}

/\*#@+

\* end of elseif

\*/

$ObjectCat->break\_connection();

echo $response;

}

/\*\*#@-

\* Fin del Condicional superior

\*/

/\*\*#@+

\* Bloque condicional de la variable $opción para heredar de la clase Category

\* Conditional block of variable $option to inherit from Category class

\*/

else

{

require\_once '../Model/Category.php';

class AjaxCategory extends Category

{

/\*\*

\* Datos de la capa View destinados a la Clase Category

\*/

static private $data\_process;

/\*\*

\* recibir los datos requeridos para el proceso, mediante el constructor

\* receive the data required for the process, through the constructor...

\* posteriormente, se lanza la conexión con el SGBD

\* subsequently, the connection with the DBMS is launched

\* finalmente, los datos requeridos se asignan al atributo $view\_data

\* finally, the required data is assigned to the $view\_data attribute

\* @param array() $view\_data

\*/

public function \_\_construct($view\_data)

{

parent::\_\_construct();

self::$data\_process = $view\_data;

}

/\*\*

\* Método para enviar datos al Modelo

\* Method to send data to Model

\*/

public function process()

{

/\*

\* capturar los datos del formulario en variables independientes

\* capture form data into independent variables...

\*

\*/

$name\_cat = self::$data\_process["name"];

$description = self::$data\_process["description"];

$id\_cat = self::$data\_process["id\_cat"];

if($id\_cat=="")

{

parent::new\_cat($name\_cat, $description);

}

else

{

parent::update\_cat($id\_cat, $name\_cat, $description);

}

}

/\*\*

\* procedimiento relacionado con la clase Category para obtener una respuesta

\* procedure related to the Category class to get a response

\* @return object or @return array() or @return string or @return boolean

\*/

public function get\_response()

{

return parent::get\_response();

}

/\*\*

\* procedimiento relacionado con la clase Category para obtener un error

\* procedure related to Categoryry class to get error

\* @return string

\*/

public function get\_error()

{

return parent::get\_error();

}

/\*\*

\* procedimiento relacionado con la clase Category que solicita romper la conexión con el SGBD

\* procedure related to Category class that requests to break the connection with the DBMS

\*/

public function break\_connection()

{

parent::break\_connection();

}

}

$ObjectCat = new AjaxCategory($view\_data);

$ObjectCat->process();

$response = $ObjectCat->get\_response();

if($response==false || $response==null)

{

$error = $ObjectCat->get\_error();

$response = $error[2];

}

else

{

$response="Successful";

}

$ObjectCat->break\_connection();

echo $response;

}

En negrita el código modificado. Recordemos que PostgreSql trata casi todo como objetos. Y eso es lo que obtenemos de casi todas las consultas.

Con lo anterior terminamos el proceso de Ingresar una nueva categoría, verificando e informando al usuario si la categoría a ingresar ya existe. En caso contrario la ingresa en la DB y muestra el respectivo mensaje, recargando finalmente la lista de categorías existentes

Ahora vamos a modificar el código de cada archivo necesario para listar mejor las categorías y habilitar la edicion de cada una de ellas...