**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

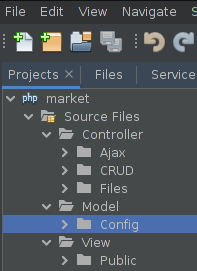
Controller/Ajax:.. archivos PHP con respuesta a las peticiones AJAX que se realizan desde la capa de presentación

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();**

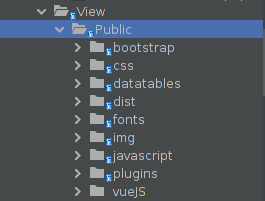
**}**

**}**

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

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