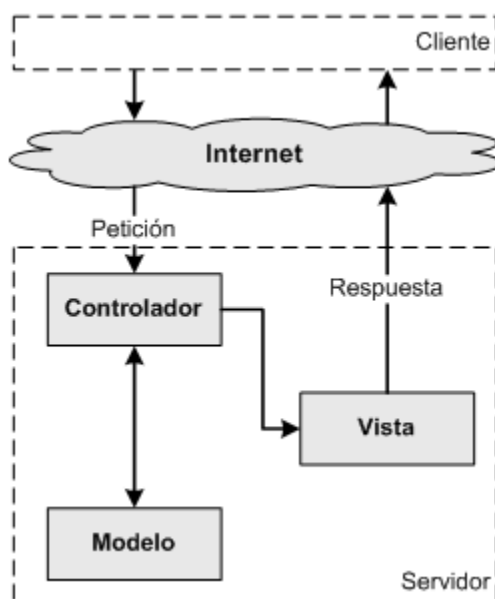


Autor: Fredy Cortés

Arquitectura Modelo-Vista-Controlador

Es un patrón arquitectónico que fue inventado en el contexto de SmallTalk para realizar una separación entre la interfaz gráfica y el código del funcionamiento de una aplicación. Esta idea teórica afectó de forma importante, a gran parte del código de SmallTalk y fue posteriormente aplicada a los lenguajes de programación que están orientados a objetos.

Las entradas del usuario, los modelos del mundo exterior y la retroalimentación visual, son explícitamente separados y manejados por tres tipos de objetos llamados Controladores, Modelos y Vistas respectivamente. Cada uno especializado para un conjunto de tareas específicas.



Este patrón ya tiene un buen tiempo desde su creación, no obstante, en los últimos años ha ganado mucha fuerza y seguidores, gracias a la aparición de muchos frameworks de desarrollo web que lo utilizan para la construcción de aplicaciones.

A continuación, podemos visualizar algunos frameworks que implementan el patrón arquitectónico MVC.



ACREDITACIÓN INSTITUCIONAL
Avanzamos... ¡Es nuestro objetivo!



Para entender este patrón como debe ser, hablaremos de cada una de las capas que maneja.

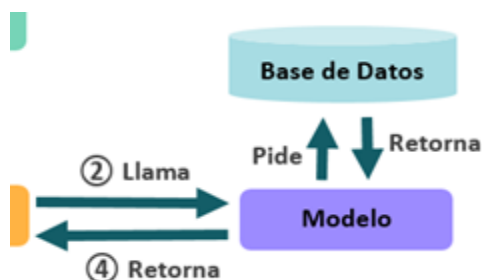
MODELO

En la capa Modelo encontraremos siempre una representación de los **datos del dominio**, es decir, aquellas entidades que nos servirán para almacenar información del sistema que estamos desarrollando. Por ejemplo, si estamos desarrollando una aplicación de facturación, en el modelo existirán las clases **Factura**, **Cliente** o **Proveedor**, entre otras.

Si nuestra aplicación forma parte de un sistema distribuido, es decir, consume servicios prestados por otros sistemas, en el Modelo encontraremos **las clases de transferencia de datos** (DTO, *Data Transfer Objects*) que nos permitirán intercambiar información con ellos.

Así mismo, encontraremos la **lógica de negocio** de la aplicación, es decir, la implementación de las reglas, acciones y restricciones que nos permiten gestionar las entidades del dominio. Será por tanto el responsable de que el sistema se encuentre siempre en un estado consistente e íntegro.

Por último, el Modelo será también el encargado de **gestionar el almacenamiento y recuperación de datos y entidades del dominio**, es decir, incluirá mecanismos de persistencia o será capaz de interactuar con ellos. Dado que habitualmente la persistencia se delega a un motor de bases de datos, es muy frecuente encontrar en el Modelo la implementación de componentes tipo DAL (*Data Access Layer*, o Capa de Acceso a Datos) y ORMs.



SC-CER96940

"Formando líderes para la construcción de un nuevo país en paz"

Universidad de Pamplona
Pamplona - Norte de Santander - Colombia
Tels: (7) 5685303 - 5685304 - 5685305 - Fax: 5682750



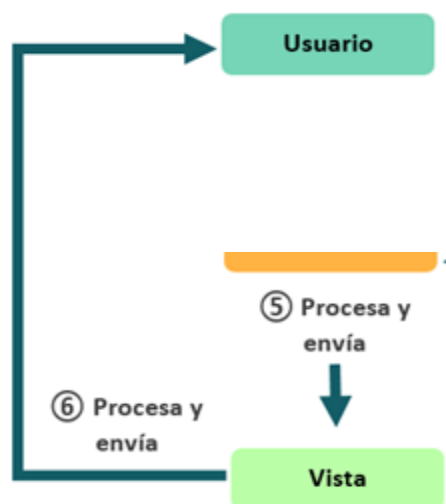
VISTA

Los componentes de la Vista son los responsables de **generar la interfaz** de nuestra aplicación, es decir, de componer las pantallas, páginas, o cualquier tipo de resultado utilizable por el usuario o cliente del sistema. De hecho, suele decirse que **la vista es una representación del estado del Modelo** en un momento concreto y en el contexto de una acción determinada.

Por ejemplo, si un usuario está consultando una factura a través de una aplicación web, la Vista se encargará de representar visualmente el estado actual de la misma en forma de página visualizable en su navegador. Si en un contexto B2B el cliente de nuestro sistema es a su vez otro sistema, la vista podría ser un archivo **XML o JSON** con la información solicitada. En ambos casos se trataría de la misma factura, es decir, la misma entidad del Modelo, pero su representación es distinta en función de los requisitos.

Cuando las vistas componen la interfaz de usuario de una aplicación, deberán contener los **elementos de interacción** que permitan al usuario enviar información e invocar acciones en el sistema, como botones, cuadros de edición o cualquier otro tipo de elemento, convenientemente adaptados a la tecnología del cliente.

En el caso de las aplicaciones para la Web, normalmente en la Vista se encontrarán los componentes capaces de **generar el lenguaje de marcado** de la página que será enviada al usuario.





ACREDITACIÓN INSTITUCIONAL
Avanzamos... ¡Es nuestro objetivo!



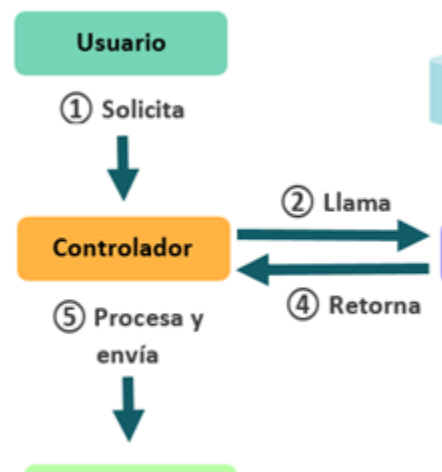
CONTROLADOR

La misión principal de los componentes incluidos en el Controlador es actuar como **intermediarios entre el usuario y el sistema**. Serán capaces de capturar las acciones de este sobre la Vista, como puede ser la pulsación de un botón o la selección de una opción de menú, interpretarlas y actuar en función de ellas. Por ejemplo, retornando al usuario una nueva vista que represente el estado actual del sistema, o invocando a acciones definidas en el Modelo para consultar o actualizar información.

Realizarán también tareas de **transformación de datos** para hacer que los componentes de la Vista y el Modelo se entiendan. Así, traducirán la información enviada desde la interfaz, por ejemplo, los valores de campos de un formulario recibidos mediante el protocolo HTTP, a objetos que puedan ser comprendidos por el Modelo, como pueden las clases o las entidades del dominio.

Y de la misma forma, el Controlador tomará la información procedente del Modelo y la adaptará a formatos o estructuras de datos que la Vista sea capaz de manejar.

Por todo ello, podríamos considerar **el Controlador como un coordinador general del sistema**, que regula la navegación y el flujo de información con el usuario, ejerciendo también como intermediario entre la capa de Vista y el Modelo.



SC-CER96940



"Formando líderes para la construcción de un nuevo país en paz"

Universidad de Pamplona
Pamplona - Norte de Santander - Colombia
Tels: (7) 5685303 - 5685304 - 5685305 - Fax: 5682750



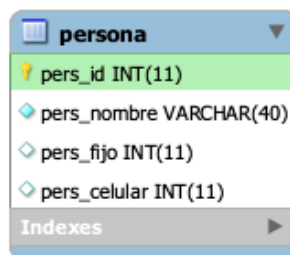
ACREDITACIÓN INSTITUCIONAL
Avanzamos... ¡Es nuestro objetivo!



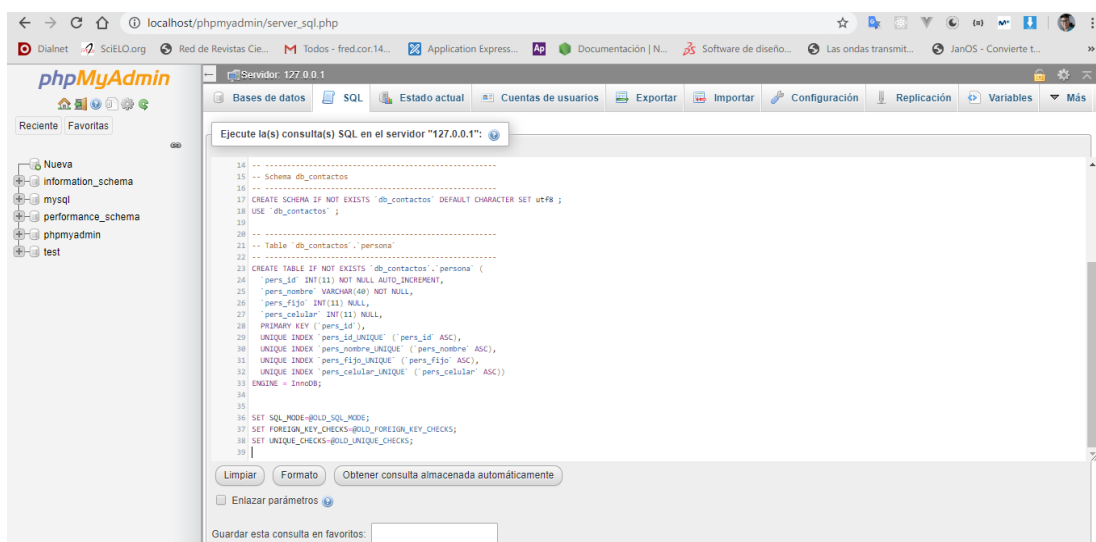
Implementación

Para esta práctica o implementación de la arquitectura Modelo-Vista-Controlador utilizaremos puro código PHP sin utilizar ningún framework en específico, con el fin de entender en profundidad este patrón arquitectónico. También utilizaremos el gestor de base de datos MySQL que viene integrado en la herramienta de Xampp.

Construiremos un mini proyecto de agenda personal. Es decir, crearemos un sitio web para registrar números de contacto personales. Para empezar, realizaremos el modelo relacional el cual nos dará a entender la lógica del software, para este caso realizamos dicho modelo utilizando la herramienta de ORACLE llamada MySQL Workbench en su versión 8.0.



Este software también nos ofrece la posibilidad de generar el código SQL para ejecutarlo dentro de nuestra interfaz “phpmyadmin” quien se encargará de crear la base de datos. La llamaremos “db_contactos”.



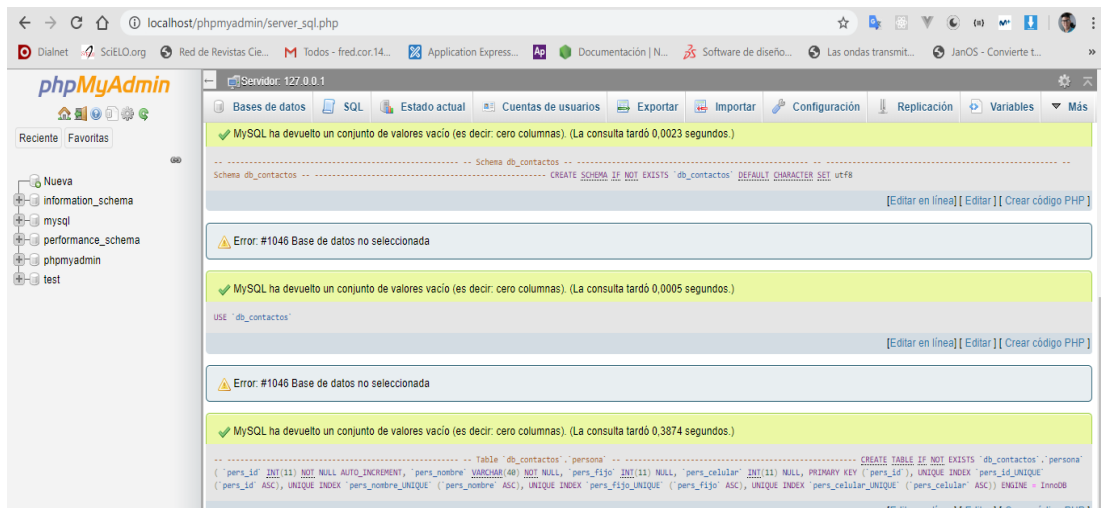
SC-CER96940



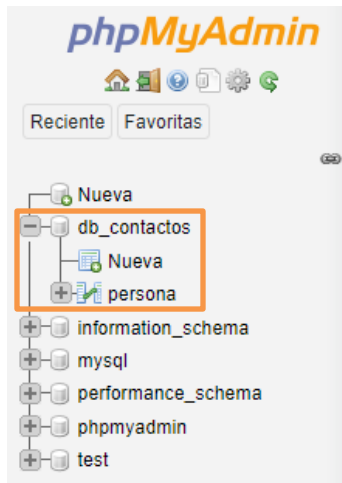
“Formando líderes para la construcción de un nuevo país en paz”

Universidad de Pamplona
Pamplona - Norte de Santander - Colombia
Tels: (7) 5685303 - 5685304 - 5685305 - Fax: 5682750

Cuando ejecutemos nuestro script de base de datos nos debería devolver una respuesta como la que se muestra a continuación.



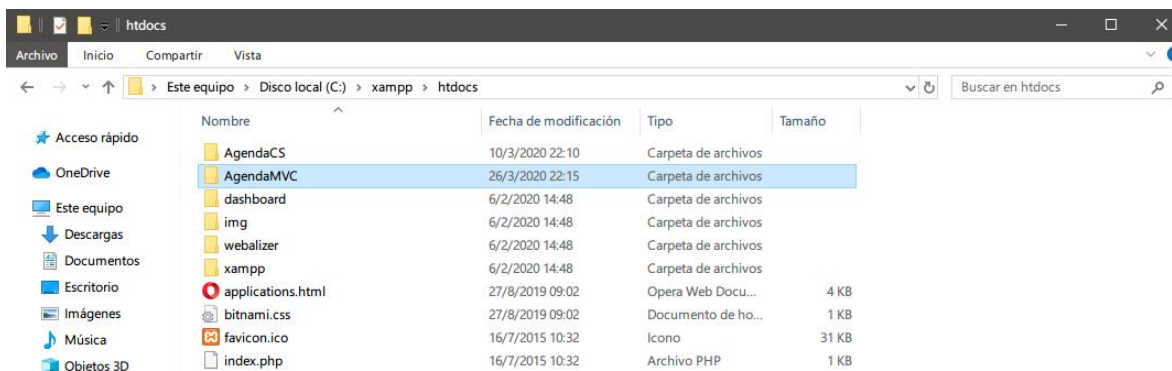
Luego de que su ejecución haya sido exitosa, procedemos a recargar el servidor de “phpmyadmin”, para verificar que nuestra base de datos “db_contactos” ha sido creada correctamente y que contiene la tabla que hemos definido.



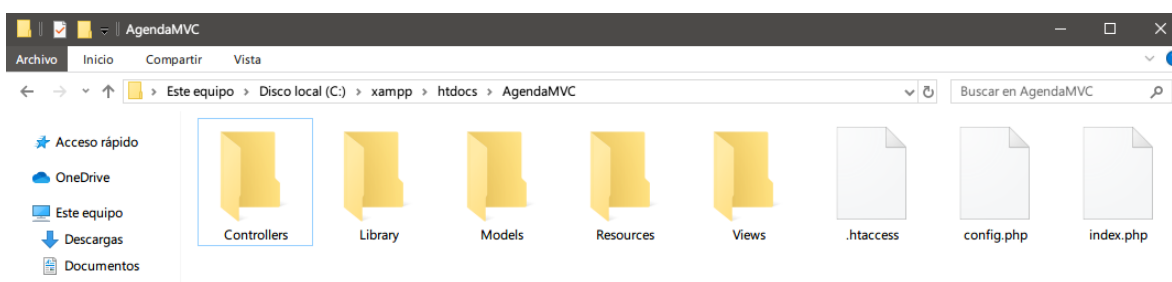
Continuaremos creando nuestro proyecto dentro de la carpeta “htdocs” la cuál es la que está enlazada directamente a nuestro servidor local “localhost” o “127.0.0.1”, que se encuentra dentro de la carpeta principal “xampp”.



ACREDITACIÓN INSTITUCIONAL
Avanzamos... ¡Es nuestro objetivo!



Ahora procederemos a construir las siguientes carpetas las cuales definirán la estructura general de nuestro patrón arquitectónico MVC que implementaremos.



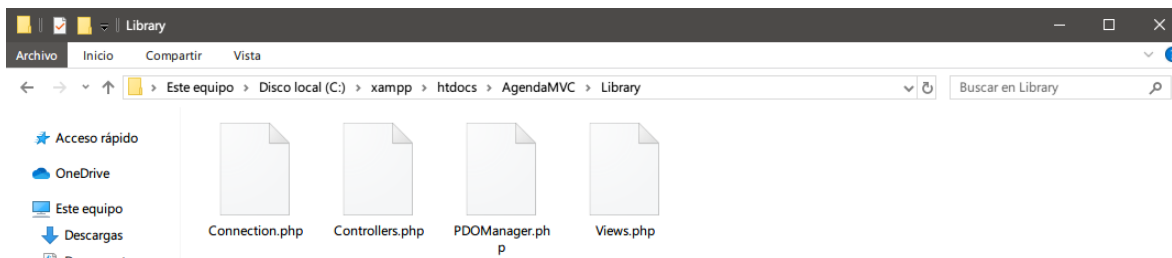
A continuación, se muestra la codificación del archivo “config.php” el cual simplemente maneja constantes de cadenas de texto que hacen referencia a algunas carpetas en específico que necesitaremos utilizar en otros archivos más adelante.

```
<?php
const LB = "Library/";
const VW = "Views/";
const DF = "Default/";
const RS = "Resources/";
const URL = "http://localhost/AgendaMVC"
;
```

Dentro de nuestra carpeta Library crearemos los siguientes archivos PHP.



ACREDITACIÓN INSTITUCIONAL
Avanzamos... ¡Es nuestro objetivo!



Estos archivos PHP son los más importantes, ya que con ellos se logrará modelar el patrón arquitectónico MVC que queremos implementar. Por lo tanto, empezaremos a codificar estos archivos que finalmente terminarán siendo clases de PHP.

Empezaremos codificando el archivo “Views.php”, el cual se encargará de gestionar la lógica de renderización de las vistas, es decir la respuesta que finalmente se le da al cliente.

```
<?php
class Views {
    function render($controller, $view) {
        // Obtenemos el nombre de la clase que nos llega como parametro
        $controller = get_class($controller);
        // Luego requerimos el archivo html para que la vista responda
        require VW . $controller . "/" . $view . ".html";
    }
}
```

Luego codificaremos el archivo “Controllers.php”, el cual se encargará de crear la comunicación tanto con el modelo como con la vista, lo cual vimos en la parte teórica.

Crearemos una función que llamaremos loadClassModel la cual entabla la conexión de un determinado controlador con su respectivo modelo, parametrizando una estructura determinada en los nombres de los archivos del modelo.

Por otra parte creamos una instancia de la clase Views codificada anteriormente, para de igual manera obtener el enlace y poder comunicarse con ella.



SC-CER96940



“Formando líderes para la construcción de un nuevo país en paz”

Universidad de Pamplona
Pamplona - Norte de Santander - Colombia
Tels: (7) 5685303 - 5685304 - 5685305 - Fax: 5682750


```
<?php
class Controllers {
    function __construct() {
        // creamos una instancia de la Views para
        // poder acceder al metodo render
        $this->view = new Views();
        $this->loadClassModels();
    }

    private function loadClassModels() {
        // obtenemos el nombre de la clase controladora
        // determinando que el modelo perteneciente a este
        // controlador siempre debera tener al final la palabra "Model"
        $model = get_class($this) . "Model";
        // creamos una cadena de texto con la ruta en la cual debe
        // existir la clase modelo perteneciente a dicho controlador
        $path = "Models/" . $model . ".php";
        if(file_exists($path)) { // verificamos que exista el archivo
            require $path; // para poder requerirlo
            $this->model = new $model(); // y crear una instancia de el
        }
    }
}
```

Ahora procederemos a codificar nuestro archivo “PDOManager.php”, el cual nos permitirá crear la lógica de manipulación y consulta de datos, por tanto, tendremos métodos básicos como consultas, inserciones, actualizaciones, eliminaciones, entre otras.

El nombre de la clase indica que utilizaremos la biblioteca PDO que viene integrada en PHP, también existen alternativas como mysqli, la cual es bastante utilizada. Pero te recomendamos utilizar la biblioteca PDO, la cual ofrece un mayor o más amplio ecosistema para la conexión de diferentes gestores de bases de datos, lo cual esto nos permite desarrollar una aplicación con un grado de escalabilidad mayor. Esto es muy importante en la ingeniería del software, ya que estás manejando uno de los tópicos bastante estudiados en este campo, que es la gestión del cambio.

A continuación, se muestra la codificación de nuestra clase PDOManager.



ACREDITACIÓN INSTITUCIONAL

Avanzamos... ¡Es nuestro objetivo!



```
<?php
class PDOManager {
    private $pdo; // creamos la variable que va a contener la instancia de la clase PDO
    function __construct($user, $password, $dbname) {
        try {
            // creamos la instancia utilizando la variable $pdo
            $this->pdo = new PDO(
                'mysql:host=localhost;dbname=' . $dbname . ';charset=utf8',
                $user,
                $password,
                [
                    PDO::ATTR_EMULATE_PREPARES => FALSE,
                    PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION
                ]
            );
        } catch (PDOException $e) { // si algo llega a salir mal
            print "Error!: " . $e->getMessage();
            die(); // matamos o cerramos la conexión
        }
    }

    function select($columns, $tables, $where, $params) {
        try {
            if ($where == "")
                $query = "SELECT " . $columns . " FROM " . $tables;
            else
                $query = "SELECT " . $columns . " FROM " . $tables . " WHERE " . $where;

            $sth = $this->pdo->prepare($query);

            if ($where == "")
                $sth->execute();
            else
                $sth->execute($params);

            $response = $sth->fetchAll(PDO::FETCH_ASSOC);
            return array('results' => $response);
        } catch (PDOException $e) {
            return $e->getMessage();
        }
        $pdo = null;
    }

    function insert($table, $values, $params) {
        try {
            $query = "INSERT INTO " . $table . " VALUES";
            $sth = $this->pdo->prepare($query);
            $sth->execute($params);
            return true;
        } catch (PDOException $e) {
            return $e->getMessage();
        }
        $pdo = null;
    }

    function update($table, $columns, $newValues, $where, $params) {
        try {
            $query = "UPDATE " . $table . " SET ";
            for ($i = 0; $i < count($columns); $i++) {
                $query .= $columns[$i] . " = " . $newValues[$i] . " ";
                if ((count($columns) - 1) != $i) $query .= ", ";
            }
            $query .= " WHERE " . $where;
            $sth = $this->pdo->prepare($query);
            $sth->execute($params);
            return true;
        } catch (PDOException $e) {
            return $e->getMessage();
        }
        $pdo = null;
    }

    function delete($table, $where, $params) {
        try {
            $query = "DELETE FROM " . $table . " WHERE " . $where;
            $sth = $this->pdo->prepare($query);
            $sth->execute($params);
            return true;
        } catch (PDOException $e) {
            return $e->getMessage();
        }
        $pdo = null;
    }
}
```



SC-CER96940



"Formando líderes para la construcción de un nuevo país en paz"

Universidad de Pamplona
Pamplona - Norte de Santander - Colombia
Tels: (7) 5685303 - 5685304 - 5685305 - Fax: 5682750



Por otra parte, la clase Connection se creará con el fin de crear una instancia de la clase PDOManager y lograr enviarle los parámetros de conexión a nuestra base de datos. Esta clase Connection será únicamente accedida por los modelos, ya que cada modelo existente en nuestra aplicación heredará de dicha clase, para poder conectarnos o establecer la comunicación con el gestor o manipulador de los datos.

Continuaremos mostrando la implementación de la clase Connection dentro de nuestro archivo “Connection.php”.

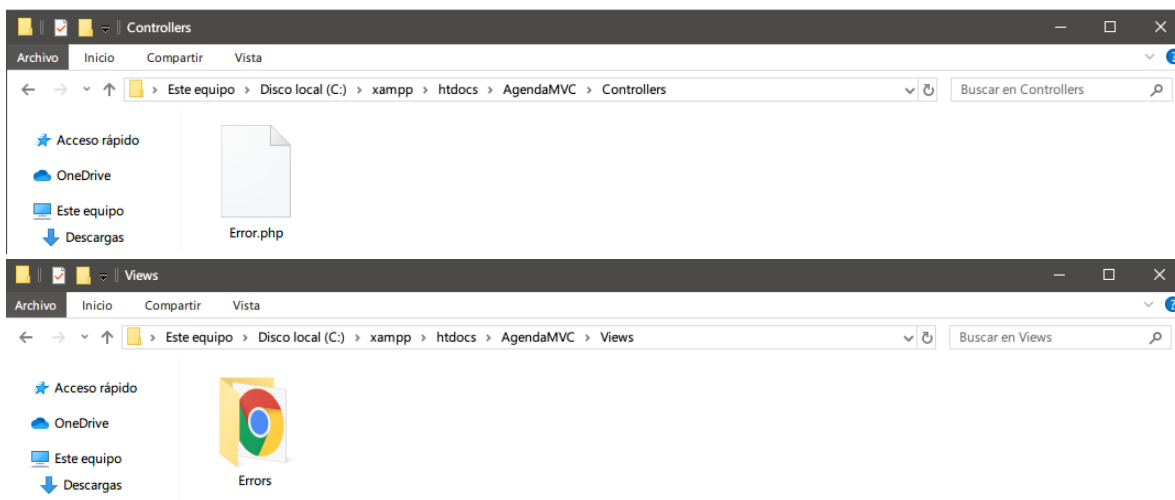
```
<?php
class Connection {
    function __construct() {
        // creamos una variable $db para instanciar a la clase PDOManager
        // y poder utilizarla desde un determinado modelo
        $this->db = new PDOManager("root", "", "db_contactos");
    }
}
```

Para continuar y poder hacer uso de nuestra lógica de la implementación del modelo arquitectónico MVC dentro de la carpeta Library, debemos utilizar el archivo “.htaccess” creado en la raíz de nuestro proyecto como se mostró antes. Este archivo con su nombre completo “Hypertext Access”, siendo muy popular en servidores web basados en apache, nos permite aplicar distintas políticas de acceso a directorios o archivos con la idea de mejorar la seguridad de nuestro proyecto.

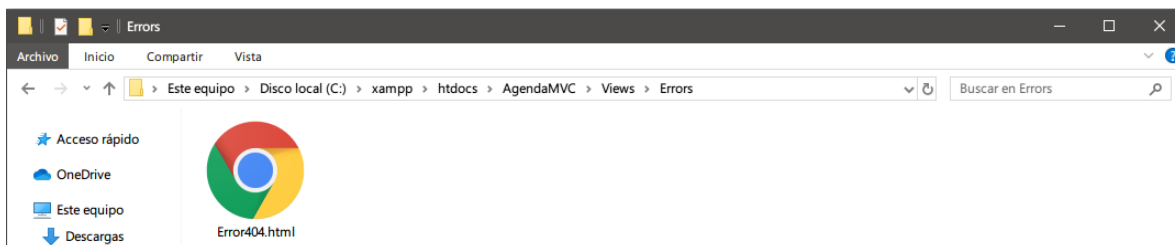
En este caso utilizaremos algunos comandos que nos permitirán realizar unas modificaciones al momento de obtener nuestra ruta de la aplicación. Esto con el fin de obtener una estructura sencilla en el enrutamiento de nuestra aplicación, lo que se conoce comúnmente como rutas amigables.

```
.htaccess
1 RewriteEngine on
2
3 RewriteCond %{REQUEST_FILENAME} !-d
4
5 RewriteCond %{REQUEST_FILENAME} !-f
6
7 RewriteRule ^(.*)$ index.php?url=$1 [QSA,L]
```

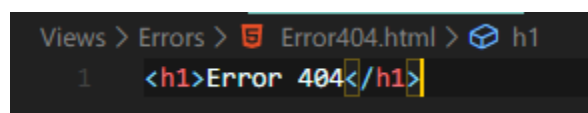
Antes de construir la lógica necesaria para nuestro archivo principal “index.php” creado en la raíz de nuestro proyecto, es necesario empezar a implementar la lógica de nuestra arquitectura que ya hemos creado. Construyendo un Controlador y una Vista que responde con un error 404 o error de recurso no encontrado, cuando se introduzca una ruta que no existe, para lo cual crearemos los siguientes archivos tanto en el directorio “Controllers” como en el directorio “Views”, los cuales también se encuentran creados en la raíz de nuestro proyecto.



Dentro de nuestro directorio “Views” es necesario crear el archivo HTML que finalmente se le responderá al cliente. Lo nombraremos “Error404.html”.



Que, para fines prácticos, simplemente construiremos una etiqueta h1 con el mensaje “Error 404”.





ACREDITACIÓN INSTITUCIONAL
Avanzamos... ¡Es nuestro objetivo!



Ahora es necesario codificar nuestro Controlador en el archivo “Error.php” que creamos. Debemos tener en cuenta que para esta situación no necesitamos el uso de un Modelo, ya que simplemente es un mensaje de error con el cual le queremos responder al cliente, por lo tanto, no es necesario tocar la base de datos para realizar este proceso.

```
<?php
class Errors extends Controllers {
    function error() {
        // Utilizamos la variable $view la cual es la instancia de
        // nuestra clase Views creada dentro de la clase Controllers
        // por ello tenemos acceso a esta, ya que estamos heredando
        // de la clase Controllers
        $this->view->render($this, "Error404");
    }
}
```

Debemos recordar que en la función de renderización de la vista el primer parámetro es utilizado para conocer el nombre de nuestro Controlador y poder buscar la carpeta perteneciente a ese controlador dentro de la carpeta Views creada en la raíz de nuestro proyecto, por tanto, es necesario llamar “Errors” tanto al Controlador como a la carpeta que se encuentra en Views, para que se puedan enlazar correctamente. También es necesario tener en cuenta que el segundo parámetro que recibe nuestra función render, es el nombre del archivo HTML que creamos para ese controlador específicamente.

Ahora bien, podremos continuar codificando la lógica de nuestro archivo principal “index.php”. El cual hará uso de toda la lógica creada anteriormente para tener control sobre el enrutamiento de nuestro proyecto. Es necesario aclarar, que las rutas se determinan por la siguiente estructura “<http://dominio/nameController/nameMethod>”, siendo estos dos últimos, quienes determinarán el flujo de ejecución de nuestra arquitectura.



SC-CER96940



“Formando líderes para la construcción de un nuevo país en paz”

Universidad de Pamplona
Pamplona - Norte de Santander - Colombia
Tels: (7) 5685303 - 5685304 - 5685305 - Fax: 5682750



ACREDITACIÓN INSTITUCIONAL

Avanzamos... ¡Es nuestro objetivo!



```
<?php
require "config.php";
$url = isset($_GET["url"]) ? $_GET["url"] : "Index/index";

$url = explode("/", $url); // separo todo lo que este en la cadena de texto $url por "/"
// Esta operación la realizamos para saber si la ruta de nuestro sitio web tiene la ruta
// que necesitamos, que es la siguiente: http://dominio/nameController/nameMethod.
// Por lo tanto nuestro vector $url tendrá el nameController en su primera posición
// y el nameMethod en su segunda posición.

$controller = "";
$method = "";
if(isset($url[0])) // verifico si existe el nombre del controlador en la ruta.
    $controller = $url[0];

if(isset($url[1]) && $url[1] != "") // verifico si existe el nombre del método en la ruta.
    $method = $url[1];

spl_autoload_register(function($class) {
    if(file_exists(LB.$class . ".php")) {
        require LB.$class . ".php";
    }
});
// Este proceso nos permite autocargar las clases que vamos a contruir dentro de
// nuestra carpeta Library las cuales van a tener la logica necesaria para
// implementar de forma nativa nuestra arquitectura MVC

require 'Controllers/Error.php';
$error = new Errors(); // creamos una instancia de la clase Errors para mostrar 404

$controllersPath = "Controllers/" . $controller . '.php';
// Construimos el enlace a la clase controlador que obtuvimos mediante la ruta

if(file_exists($controllersPath)) { // verificamos si el archivo controlador existe
    require $controllersPath;

    $controller = new $controller(); // creamos la instancia de nuestra clase controladora

    // verificamos si dentro de la ruta que obtuvimos se especificaba el metodo a ejecutar
    if(isset($method)) {
        // verificamos si el metodo obtenido en la ruta existe dentro de la clase controladora
        if(method_exists($controller, $method))
            $controller->{$method}(); // ejecutamos al metodo determinado por la ruta
        else
            $error->error(); // ejecutamos el método error que retornará la vista Error 404
    }
} else
    $error->error();
```



SC-CER96940



"Formando líderes para la construcción de un nuevo país en paz"

Universidad de Pamplona
Pamplona - Norte de Santander - Colombia
Tels: (7) 5685303 - 5685304 - 5685305 - Fax: 5682750

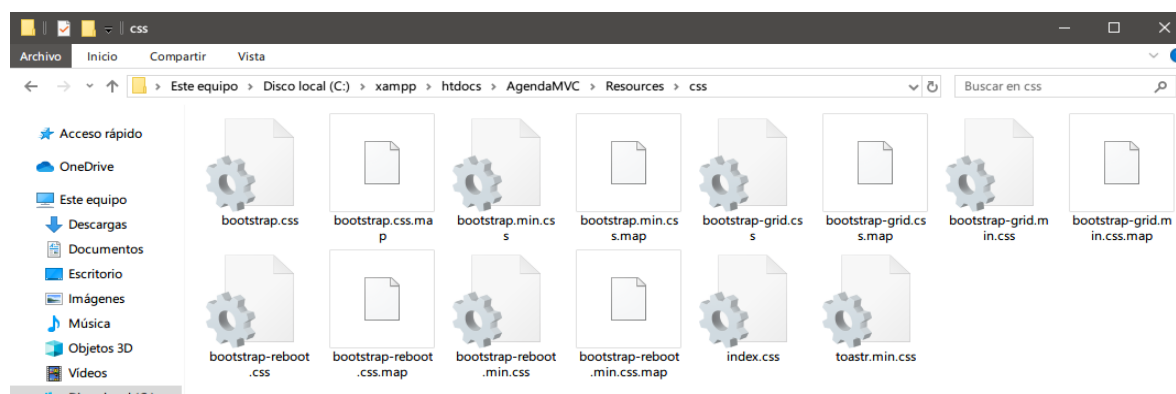
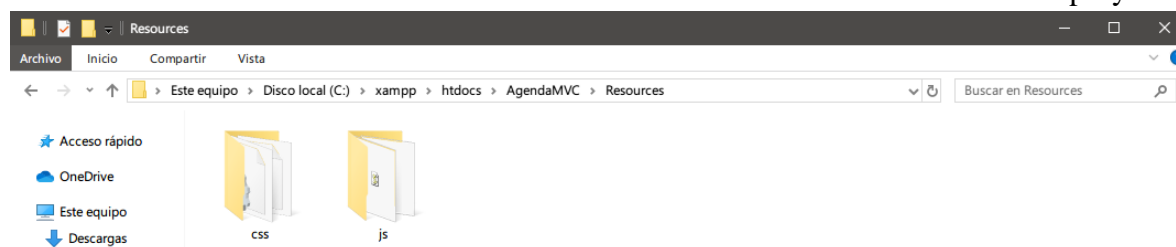


ACREDITACIÓN INSTITUCIONAL
Avanzamos... ¡Es nuestro objetivo!



En este momento, podemos empezar a construir el código necesario para nuestro proyecto objetivo, que terminará siendo la agenda personal. Entonces para este caso práctico utilizaremos una biblioteca de front-end para desarrollo ágil de nuestras interfaces de usuario UI. Esta biblioteca se puede descargar desde el siguiente enlace, <https://getbootstrap.com/docs/4.4/getting-started/download>, el cual nos compartirá archivos de CSS como también de JavaScript, para su respectivo uso en la maquetación de nuestro sitio web.

Estos directorios tanto “css” como “js” los alojaremos dentro de nuestro directorio raíz Resources, en el cual almacenaremos todos estos tipos de recursos que nos ayudarán en la construcción de nuestro proyecto.



También utilizaremos una biblioteca llamada “Toastr” que nos permitirá pintar mensajes de alerta utilizando JavaScript. Esta biblioteca también contiene un archivo CSS y JavaScript por lo tanto los alojaremos de igual forma en las carpetas correspondientes.

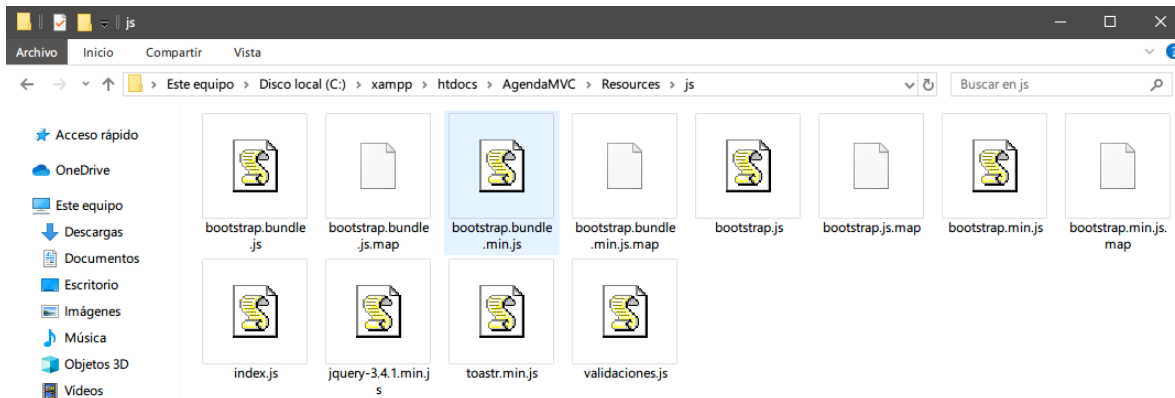


SC-CER96940



“Formando líderes para la construcción de un nuevo país en paz”

Universidad de Pamplona
Pamplona - Norte de Santander - Colombia
Tels: (7) 5685303 - 5685304 - 5685305 - Fax: 5682750



Podemos observar en las imágenes anteriores que creamos algunos archivos de uso personal, como “index.js”, “validaciones.js” e “index.css”. Los cuales nos permitirán agregarle características de diseño personales e interactividad.

Entonces seguiremos con la codificación del archivo “index.css” el cual utilizamos personalmente, para hacer uso de unas fuentes de Google Fonts llamada Raleway y utilizar algunos colores en la vista.

```
body {  
    font-family: 'Bellota Text', cursive;  
    background-color: lightgray;  
}  
.cajaContacto {  
    background-color: white;  
    border-radius: 5px;  
}
```

Ahora podemos continuar realizando la construcción o codificación del fichero “validaciones.js”, el cual contendrá dos funciones que me permiten validar que los campos tanto nombre como telefono se encuentren bien digitados y con la estructura correcta.



ACREDITACIÓN INSTITUCIONAL

Avanzamos... ¡Es nuestro objetivo!



```
function validarNombre(input, minLength, maxLength) {
  let nombre = input.value, valid = true
  if(nombre == "") {
    input.focus()
    toastr.warning("Debe digitar un nombre de contacto", "Proceso detenido", {timeOut: 2000})
    valid = false
  } else if(nombre.length < minLength) {
    input.focus()
    toastr.warning(`La cantidad de caracteres mínimos permitidos para el nombre son ${minLength}`, "Proceso detenido", {timeOut: 2000})
    valid = false
  } else if(nombre.length > maxLength) {
    input.focus()
    toastr.warning(`La cantidad de caracteres máximos permitidos para el nombre son ${maxLength}`, "Proceso detenido", {timeOut: 2000})
    valid = false
  }
  return valid
}

function validarTelefonos(input, input2, minLength, maxLength) {
  let telFijo = input.value, telCel = input2.value, valid = true
  if(telFijo == "" && telCel == "") {
    input.focus()
    toastr.warning("Debe digitar por lo menos un teléfono de contacto", "Proceso detenido", {timeOut: 2000})
    valid = false
  }
  if(telFijo != "") {
    if(isNaN(telFijo)) {
      input.focus()
      toastr.warning("El teléfono fijo solo puede contener números", "Proceso detenido", {timeOut: 2000})
      valid = false
    } else if(telFijo.length < minLength) {
      input.focus()
      toastr.warning(`La cantidad de caracteres mínimos permitidos para el teléfono fijo son ${minLength}`, "Proceso detenido", {timeOut: 2000})
      valid = false
    } else if(telFijo.length > maxLength) {
      input.focus()
      toastr.warning(`La cantidad de caracteres máximos permitidos para el teléfono fijo son ${maxLength}`, "Proceso detenido", {timeOut: 2000})
      valid = false
    }
  }
  if(telCel != "") {
    if(isNaN(telCel)) {
      input2.focus()
      toastr.warning("El teléfono celular solo puede contener números", "Proceso detenido", {timeOut: 2000})
      valid = false
    } else if(telCel.length < minLength) {
      input2.focus()
      toastr.warning(`La cantidad de caracteres mínimos permitidos para el teléfono celular son ${minLength}`, "Proceso detenido", {timeOut: 2000})
      valid = false
    } else if(telCel.length > maxLength) {
      input2.focus()
      toastr.warning(`La cantidad de caracteres máximos permitidos para el teléfono celular son ${maxLength}`, "Proceso detenido", {timeOut: 2000})
      valid = false
    }
  }
  return valid
}
```

Por otra parte crearemos un nuevo directorio que nombraremos Default dentro del directorio raíz Views, el cual contendrá dos archivos HTML que se encargarán de crear la estructura



SC-CER96940



"Formando líderes para la construcción de un nuevo país en paz"

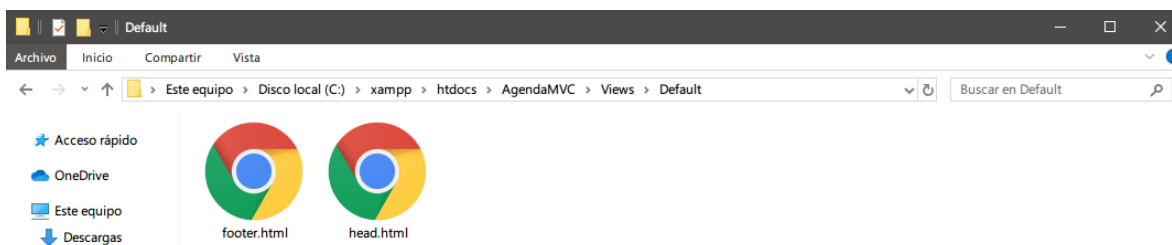
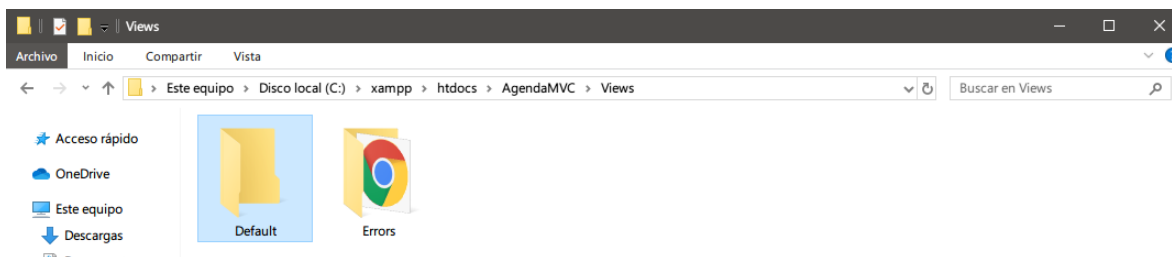
Universidad de Pamplona
Pamplona - Norte de Santander - Colombia
Tels: (7) 5685303 - 5685304 - 5685305 - Fax: 5682750



ACREDITACIÓN INSTITUCIONAL
Avanzamos... ¡Es nuestro objetivo!



principal de nuestras vistas e incluir importaciones a archivos tanto CSS como JS de las bibliotecas que utilizaremos y los archivos personales creados.



En las siguientes imágenes se mostrará la codificación del maquetado del archivo “head.html” y “footer.html” respectivamente en los cuales se define la estructura del sitio web e importaciones de los recursos a utilizar.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Agenda Personal</title>
  <link href="https://fonts.googleapis.com/css?family=Bellota+Text&display=swap" rel="stylesheet">
  <link rel="stylesheet" href="http://localhost/AgendaMVC/Resources/css/bootstrap.min.css">
  <link rel="stylesheet" href="http://localhost/AgendaMVC/Resources/css/toastr.min.css">
  <link rel="stylesheet" href="http://localhost/AgendaMVC/Resources/css/index.css">
</head>
<body>
```



SC-CER96940



“Formando líderes para la construcción de un nuevo país en paz”

Universidad de Pamplona
Pamplona - Norte de Santander - Colombia
Tels: (7) 5685303 - 5685304 - 5685305 - Fax: 5682750

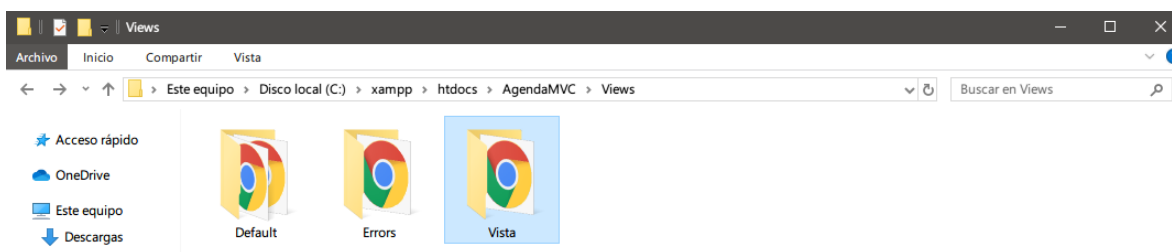


ACREDITACIÓN INSTITUCIONAL
Avanzamos... ¡Es nuestro objetivo!

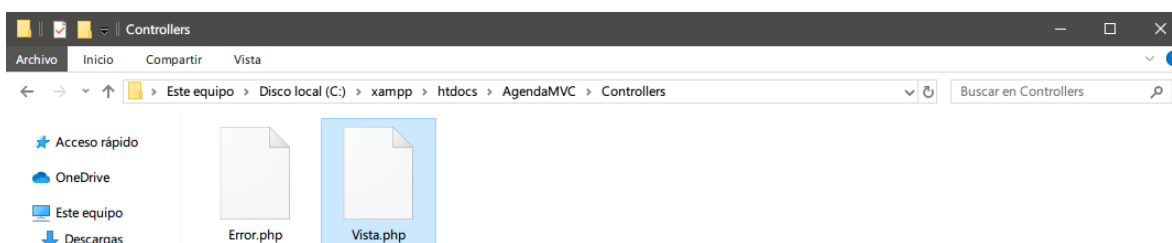
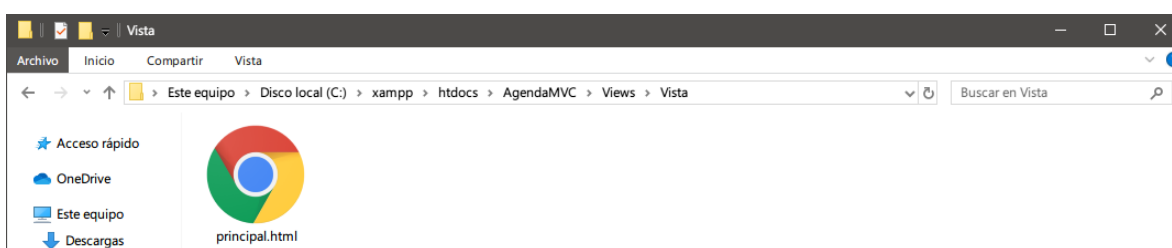


```
<script src="http://localhost/AgendaMVC/Resources/js/jquery-3.4.1.min.js"></script>
<script src="http://localhost/AgendaMVC/Resources/js/bootstrap.min.js"></script>
<script src="http://localhost/AgendaMVC/Resources/js/toastr.min.js"></script>
<script src="http://localhost/AgendaMVC/Resources/js/validaciones.js"></script>
<script src="http://localhost/AgendaMVC/Resources/js/index.js"></script>
</body>
</html>
```

Luego de todo este procedimiento, ahora si podemos empezar a realizar la maquetación de nuestra vista principal. Para ello crearemos un directorio llamado Vista dentro del directorio Views que se encuentra en la raíz, tomando en cuenta que tambien debemos crear nuestro Controlador con el mismo nombre dentro del directorio Controllers que tambien se encuentra en la raíz, para lograr entablar su comunicación.



Es necesario crear dentro de esta carpeta el archivo con el cual se le responderá al cliente. Lo llamaremos “principal.html”, por lo que es la vista principal.



SC-CER96940



“Formando líderes para la construcción de un nuevo país en paz”

Universidad de Pamplona
Pamplona - Norte de Santander - Colombia
Tels: (7) 5685303 - 5685304 - 5685305 - Fax: 5682750



Ahora teniendo en cuenta que ya contamos con estos archivos procedemos a codificarlos. Empezando primeramente, por estructurar la vista principal con código HTML en el archivo “principal.html”, de la siguiente manera.

```
<main class="container mt-3">
  <h1 class="text-center mb-3"><b>Agenda Personal</b></h1>
  <section class="row">
    <div class="col-sm-5 cajaContacto p-4">
      <form name="frmContacto" method="POST" action="../../Servidor/index.php">
        <div class="form-group">
          <label for="txtNombre">Nombre: </label>
          <input type="text" name="txtNombre" id="txtNombre" class="form-control"
            placeholder="Ingrese nombre">
        </div>
        <div class="form-group">
          <label for="txtNombre">Teléfono Fijo: </label>
          <input type="text" name="txtTelFijo" id="txtTelFijo" class="form-control"
            placeholder="Ingrese teléfono fijo">
        </div>
        <div class="form-group">
          <label for="txtNombre">Teléfono Celular: </label>
          <input type="text" name="txtTelCel" id="txtTelCel" class="form-control"
            placeholder="Ingrese teléfono celular">
        </div>
        <a class="btn btn-info btn-block">Agregar Contacto</a>
      </form>
    </div>
    <div class="col-sm-7">
      <div class="table-responsive">
        <table class="table table-bordered table-hover">
          <thead class="thead-dark text-center">
            <tr>
              <th>Nombre</th>
              <th>Teléfono Fijo</th>
              <th>Teléfono Celular</th>
            </tr>
          </thead>
          <tbody id="tablaContactos" style="background-color: white;">
            <tr>
              <td></td>
              <td></td>
              <td></td>
            </tr>
          </tbody>
        </table>
      </div>
    </div>
  </section>
</main>
```

Si analizamos esto por si solo no puede ser renderizado en la vista para responderle al cliente, por ello, es necesario codificar nuestro Controlador creado anteriormente que decidimos nombrar “Vista.php”, la cual se encargará de renderizarla.



SC-CER96940

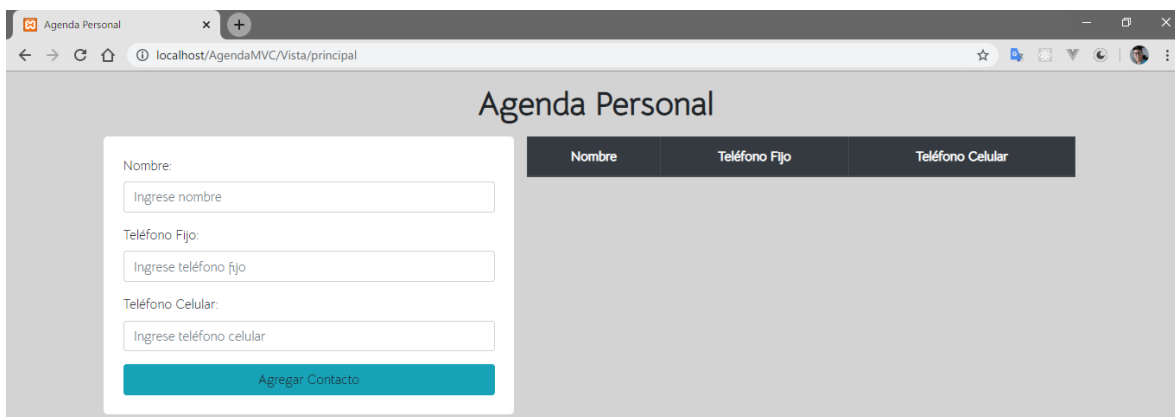


“Formando líderes para la construcción de un nuevo país en paz”

Universidad de Pamplona
Pamplona - Norte de Santander - Colombia
Tels: (7) 5685303 - 5685304 - 5685305 - Fax: 5682750

```
<?php
class Vista extends Controllers {
    function principal() {
        require VW . DF . "head.html";
        $this->view->render($this, "principal");
        require VW . DF . "footer.html";
    }
}
```

Entonces, si deseamos visualizar esta vista principal debemos realizar la petición a la siguiente dirección “<http://localhost/AgendaMVC/Vista/principal>”, según la estructura que mencionamos antes, la cual valida nuestro archivo principal “index.php” en la raíz.

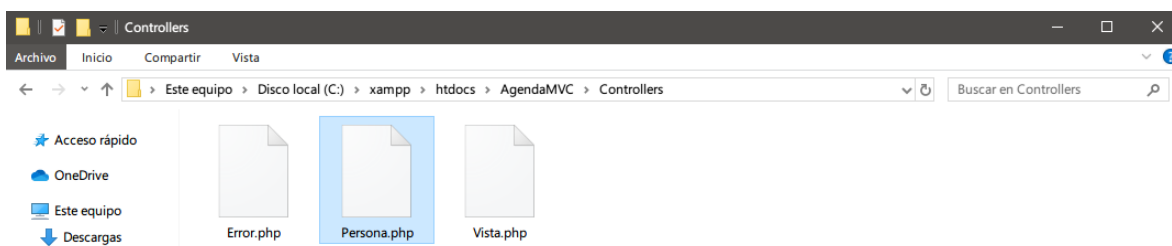


Y nos sigue controlando, que si llegamos a consultar una ruta que no exista pues simplemente nos retorna la vista de Error 404.

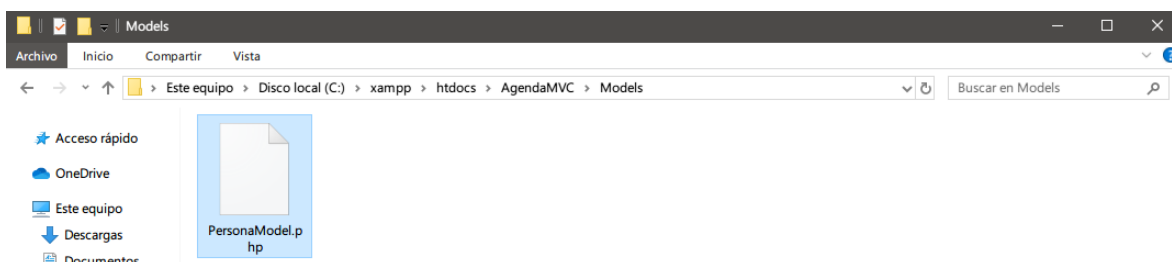


Ahora vamos a modelar el Controlador Persona, el cual utilizaremos para capturar las peticiones tanto de registro de contacto, como de obtener todos los contactos.

Crearemos el archivo “Persona.php” dentro la carpeta Controllers de la raíz del proyecto.



También debemos crear el Modelo al cual se conectará nuestro Controlador, para poder realizar de esa manera la manipulación de datos a nuestra base de datos “db_contactos” ya creada. Debemos recordar que el nombre de nuestro modelo debe tener la estructura que definimos en nuestro archivo “Controllers.php” dentro del directorio Library con la función loadClassModels, donde obligamos a crear el archivo con el mismo nombre del Controlador agregándole la cadena texto “Model” juntada, para que logren enlazarse ambos. Por lo tanto, crearemos el archivo “PersonaModel.php” dentro del directorio Models, que igualmente se encuentra en la raíz del proyecto.



Ahora es necesario codificar la funcionalidad que va a tener cada uno, por ello vamos a empezar con nuestro Controlador Persona que es el que se comunica directamente con el cliente para lograr obtener los datos que nos envíe el cliente y enviárselos a nuestro Modelo y de esta manera realizar lo que el cliente demande.


```
<?php
class Persona extends Controllers {
    function registrar() {
        $nombre = $_POST["pers_nombre"];
        $fijo = $_POST["tele_fijo"];
        $celular = $_POST["tele_celular"];
        $res = $this->model->insertarContacto($nombre, $fijo, $celular);
        echo $res;
    }

    function obtenerTodo() {
        $res = $this->model->seleccionarTodosContactos();
        if(is_array($res))
            echo json_encode($res);
        else
            echo $res;
    }
}
```

De esta manera logramos cubrir las peticiones que necesita realizar el cliente para lograr agregar y ver sus contactos. Ahora es necesario codificar nuestro Modelo PersonaModel el cual se encargará de insertar ese contacto y obtener todos los contactos para que el Controlador sepa que es lo que el Modelo hizo mediante su enlace.

Utilizaremos dos metodos de la clase PDOManager ya creada. El método insert el cual nos permitirá realizar el proceso de inserción en nuestro sistema gestor de base de datos, para este caso MySQL y el metodo select, que nos permitirá obtener todas las filas que se encuentren registradas en la tabla persona, para que por medio del Controlador, la vista logre renderizar dicho datos.

El método insert no devuelve un valor true cuando la sentencia insert se ejecutó con éxito. De otra forma, nos devolverá una cadena de texto indicando cual fue el error que esté generó en su ejecución. Por este motivo, validamos que si llega a realizarse correctamente el proceso, vamos a devolverle al Controlador un 1, el cual indicará que todo ha salido bien. Para que la vista logre saberlo e informarle al cliente.

```
<?php
class PersonaModel extends Connection {
    function insertarContacto($nombre, $fijo, $celular) {
        $table = "persona";
        $values = "";
        $params = array();
        if($fijo != "" && $celular != "") {
            $values = "(pers_nombre, pers_fijo, pers_celular)
                VALUES (:nombre, :fijo, :celular)";
            $params = array(
                "nombre" => $nombre,
                "fijo" => $fijo,
                "celular" => $celular
            );
        } else if($fijo != "") {
            $values = "(pers_nombre, pers_fijo)
                VALUES (:nombre, :fijo)";
            $params = array(
                "nombre" => $nombre,
                "fijo" => $fijo
            );
        } else {
            $values = "(pers_nombre, pers_celular)
                VALUES (:nombre, :celular)";
            $params = array(
                "nombre" => $nombre,
                "celular" => $celular
            );
        }
        $data = $this->db->insert($table, $values, $params);
        if($data === true)
            return 1;
        else
            return $data;
    }

    function seleccionarTodosContactos() {
        $columns = "*";
        $tables = "persona";
        $where = "";
        $params = "";
        return $this->db->select($columns, $tables, $where, $params);
    }
}
```

Ahora debemos codificar nuestro archivo “index.js” el cual se encuentra alojado dentro del directorio js y este a su vez dentro de Resources el cual tomará un papel importante en las



ACREDITACIÓN INSTITUCIONAL

Avanzamos... ¡Es nuestro objetivo!



vistas, para poder realizar las respectivas peticiones. Es importante aclarar que para este proceso utilizaremos la biblioteca JQuery ya agregada en el proyecto.

```
toastr.options.preventDuplicates = true

function registrarContacto() {
  let inpNombre = document.getElementById("txtNombre")
  let inpTel1 = document.getElementById("txtTelFijo")
  let inpTel2 = document.getElementById("txtTelCel")
  if(validarNombre(inpNombre, 1, 40) && validarTelefonos(inpTel1, inpTel2, 6, 15)) {
    let data = new FormData()
    data.append("pers_nombre", inpNombre.value)
    data.append("tele_fijo", inpTel1.value)
    data.append("tele_celular", inpTel2.value)
    $.ajax({
      url: "http://localhost/AgendaMVC/Persona/registrar",
      data,
      cache: false,
      contentType: false,
      processData: false,
      type: 'POST',
      success: res => {
        //console.log(res)
        if(res == 1) {
          toastr.success('Registrado Correctamente', 'Proceso Exitoso')
          inpNombre.value = ""
          inpTel1.value = ""
          inpTel2.value = ""
          llenarTablaContactos()
        }
        else toastr.error(res, "Algo ha salido mal")
      }
    })
  }
}

function llenarTablaContactos() {
  $.ajax({
    url: "http://localhost/AgendaMVC/Persona/obtenerTodo",
    data: {},
    cache: false,
    contentType: false,
    processData: false,
    type: 'POST',
    success: res => {
      //console.log(res)
      try {
        let data = JSON.parse(res)
        let tr = ""
        data.results.forEach(element => {
          tr += `
            <tr>
              <td>${element.pers_nombre}</td>
              <td>${element.pers_fijo ? element.pers_fijo : "No tiene"}</td>
              <td>${element.pers_celular ? element.pers_celular : "No tiene"}</td>
            </tr>
          `
        })
        $("#tablaContactos > tr").remove()
        $("#tablaContactos").append(tr)
      } catch (error) {
        toastr.error(error, "Algo ha salido mal")
      }
    }
  })
}

$(().ready(() => {
  llenarTablaContactos()
}))
```



SC-CER96940



"Formando líderes para la construcción de un nuevo país en paz"

Universidad de Pamplona
Pamplona - Norte de Santander - Colombia
Tels: (7) 5685303 - 5685304 - 5685305 - Fax: 5682750



ACREDITACIÓN INSTITUCIONAL
Avanzamos... ¡Es nuestro objetivo!



Para finalizar, solo debemos implementar el método “registrarContacto”, dentro de nuestra vista principal, específicamente controlando el evento del click de nuestro botón de agregar contacto, de la siguiente manera.

```
<a onclick="registrarContacto()" class="btn btn-info btn-block">Agregar Contacto</a>
```

Ahora solo quedaría ejecutar nuestro proyecto.

Verificamos que nuestras validaciones estén funcionando correctamente.

Y si todo sale bien, podríamos intentar ejecutar nuestro primer contacto.



ACREDITACIÓN INSTITUCIONAL
Avanzamos... ¡Es nuestro objetivo!



Agenda Personal

Proceso Exitoso
Registrado Correctamente

Nombre: Ingrese nombre

Teléfono Fijo: Ingrese teléfono fijo

Teléfono Celular: Ingrese teléfono celular

Agregar Contacto

Nombre	Teléfono Fijo	Teléfono Celular
Pedro Rodríguez	5776542	2147483647

Agenda Personal

Proceso Exitoso
Registrado Correctamente

Nombre: Ingrese nombre

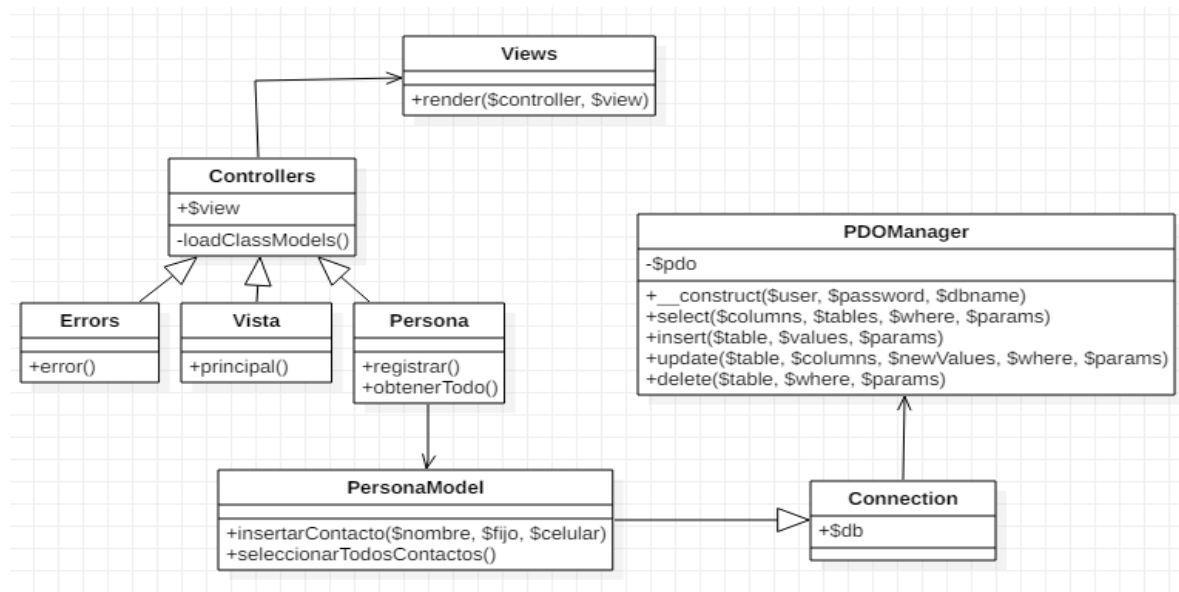
Teléfono Fijo: Ingrese teléfono fijo

Teléfono Celular: Ingrese teléfono celular

Agregar Contacto

Nombre	Teléfono Fijo	Teléfono Celular
Pedro Rodríguez	5776542	2147483647
Manuel Pedragoza	No tiene	315432133

Para concluir podemos resumir esta arquitectura visualizando el siguiente diagrama de clases.



SC-CER96940



"Formando líderes para la construcción de un nuevo país en paz"

Universidad de Pamplona
 Pamplona - Norte de Santander - Colombia
 Tels: (7) 5685303 - 5685304 - 5685305 - Fax: 5682750