IPLACEX
TECNOLÓGICO NACIONAL

Programación de Componentes Web Open Source

UNIDAD Nº III

Introducción

En algunas ocasiones, deberás desarrollar aplicaciones web que utilizan varios lenguajes y tecnologías, donde, por su envergadura, se recomienda la utilización del patrón de diseño MVC, asignando las responsabilidades de ejecución de funciones a los componentes correctos dentro de la aplicación.

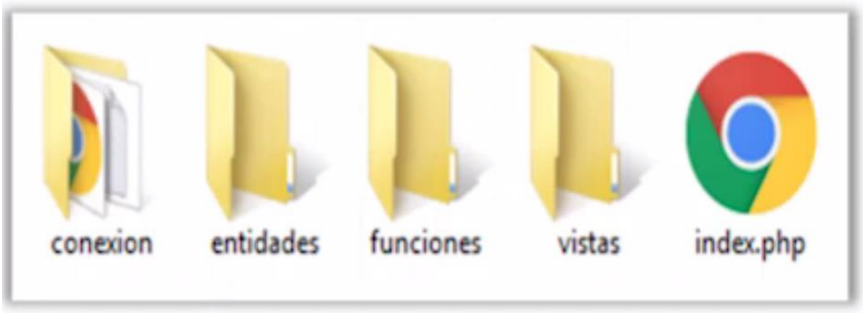
En esta experiencia, conocerás el concepto de patrón de diseño, y los beneficios que ofrece respecto del desarrollo de aplicaciones.

Conociendo la arquitectura MVC

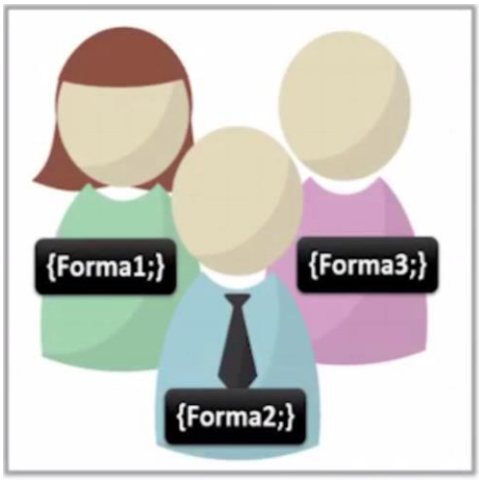
A continuación, aprenderemos a usar patrones de diseño, los cuales nos ayudarán a reordenar y mejorar la estructura y desempeño de nuestras aplicaciones.

Anteriormente, logramos desarrollar aplicaciones que nos permitieron dar solución a distintos requerimientos. Si analizamos la estructura de estos proyectos, podemos notar

que fuimos dividiendo en carpetas cada contexto y proceso, lo cual, hasta el momento, funcione sin problemas.



Pero esto debiese implementarse de manera más estandarizadas, para evitar que cada programador desarrolle según su criterio, aplicando metodologías personales. Esto resulta ser un gran problema a la hora de continuar con un proyecto en donde los gestores de éste ya no participarán o no se encuentran disponibles. Además, retomar un proyecto que no posee un patrón de desarrollo definido o desarrollado con prácticas lejos de los estándares, podría implicar una gran pérdida de tiempo, ya que comprender y dominar el código en su totalidad, podría tomar meses e incluso años.

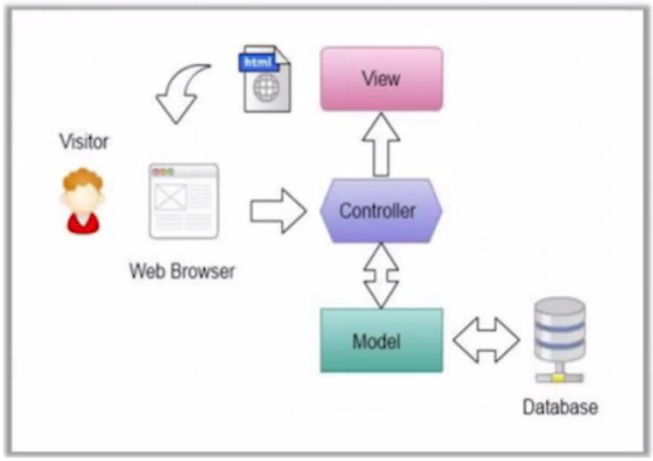


Es por este motivo que se hace necesario utilizar metodologías de desarrollo basado en patrones y arquitecturas de diseño.

A continuación, conoceremos las bases para el desarrollo de proyectos utilizando el patrón de diseño Modelo Vista Controlador, más conocido como MVS.

¿Qué es MVC?

MVC es un patrón de diseño que permite a los desarrolladores construir y separar una aplicación en tres componentes principales: el modelo, la vista y el controlador. Esta metodología otorga mayor control, eficiencia y re utilización del código desarrollado. Dentro de sus principales características, podemos destacar las siguientes:



Está basado en buenas prácticas de desarrollo.

Establece niveles de seguridad sobre cada contexto

Permite a los desarrolladores o participantes de un proyecto utilizar una metodología en común.

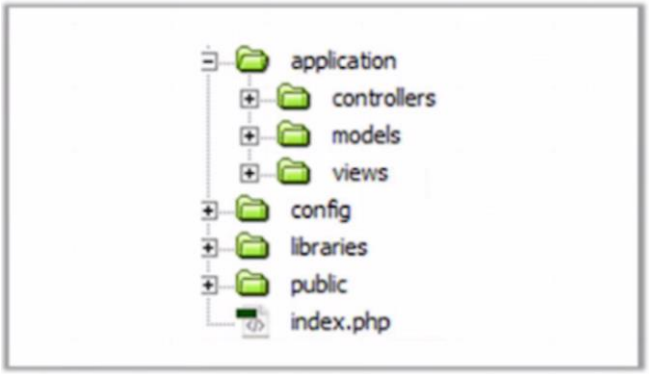
Otorga mayor calidad al producto desarrollado.

Es eficiente, ya que permite la re utilización de sus componente.

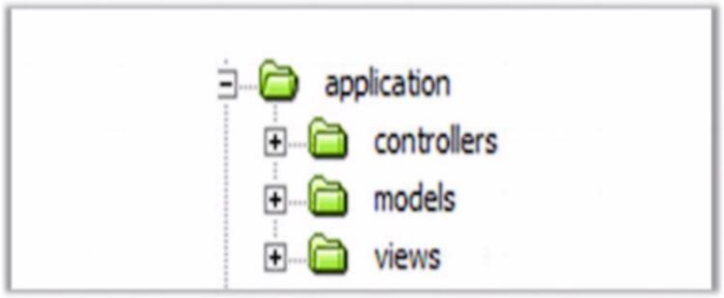
Su estructura está basada en múltiples capas que generalmente, están representadas por carpetas, en donde cada una de éstas representa un contexto dentro de un proyecto: Por ejemplo, todo lo relacionado con lo que el usuario visualizará de nuestra aplicación, está representado en el contexto de las vistas. Por otra parte, todo lo relacionado con la lógica de negocio, las acciones o comportamientos de nuestra aplicación, pertenecen al contexto del controlador. Por último, todo lo relacionado con la persistencia de datos, entidades, clases, etc., pertenece al contexto Modelo.

Creando un proyecto basado en MVC

Como hemos visto, cada contexto o capa en MVC tiene una función específica y está representada en una carpeta. En la siguiente imagen, podemos ver una estructura de carpetas que representan a este patrón de diseño. Esta forma de distribución de los elementos, está basada en la metodología utilizada por los frameworks de desarrollo de MVC. En nuestro caso, comenzaremos por comprender su funcionamiento sin pensar en los frameWorks de desarrollo.



Por ahora, iremos construyendo nuestras aplicaciones poco a poco, centrándonos en sus 3 carpetas principales.



Veamos un ejemplo: según la imagen que vemos a continuación, crearemos la estructura de carpetas para nuestro proyecto. En la carpeta models guardaremos nuestras clases referentes al modelo de datos, en controllers guardaremos los archivos para la lógica de negocios y la preparación de las vistas, y la carpeta view contendrá las páginas que el usuario visualizará.

Supongamos que tenemos una tabla llamada Personas, que se compone de 3 columnas: ID, nombre, edad. Y debemos representarla como una entidad en nuestro proyecto.
Para esto, comenzaremos por crear su representación en la capa modelo a través de una clase llamada persona.

```
CREATE TABLE personas (id int PRIMARY KEY NOT NULL,
                       nombre varchar(50),
                       edad int);
```

Como vemos en la imagen, estamos creando la clase persona, y sus atributos, en función de las columnas de la tabla. Ten en cuenta que los nombres de las clases no deben declararse en plural, aunque el nombre de la tabla venga de esa forma.

```
class Persona {
    private $id, $nombre, $edad;

    public function __construct($id=null, $nombre=null, $edad=null)
    {
        $this->id = $id;
        $this->nombre = $nombre;
        $this->edad = $edad;
    }

    public function getId(){ return $this->id; }
    public function getNombre(){ return $this->nombre; }
    public function getEdad(){ return $this->edad; }

    public function setId($id){ $this->id = $id; }
    public function setNombre($nombre){ $this->nombre = $nombre; }
    public function setEdad($edad){ $this->edad = $edad; }

    public function crear(){}
    public function editar(){}
    public function eliminar(){}
    public function buscarPorId(){}
    public function buscarTodas(){}
}
```

Siguiendo un poco más abajo, se encuentra el constructor y accesadores y mutadores. Finalmente, bajo estos métodos, nos encontramos con los métodos que representarán la persistencia y el acceso a datos para la entidad.
El término por el que se conocen los métodos principales sobre una tabla de una base de datos, es conocido como CRUD, el cual quiere decir por sus siglas Create, Read, Update, y Delete, o insertar, seleccionar, editar y eliminar respectivamente.
En esta zona es posible agregar todos los métodos que necesites a parte de los principales. También es posible tomar estos métodos y dejarlos en un nuevo archivo

dentro de la capa modelo. En nuestro caso, utilizaremos estos métodos dentro de la misma clase.

Continuando en la capa modelo, crearemos nuestra clase para conectarnos a la base de datos. El nombre que utilizaremos para esta será db.

```php
class DB
{
    private $conexion;
    function __construct()
    {
        try{
            $this->conexion = new PDO('mysql:host=localhost;dbname=bdprueba', 'root', '');
        }
        catch(PDOException $e){
            $this->conexion = null;
        }
    }
    function getConexion(){
        return $this->conexion;
    }
}
```

Comenzaremos por crear un método que busque a todos los registros o personas de la tabla. Para esto, utilizaremos el método llamado buscarTodas. Recuerda siempre utilizar nombres coherentes y que tengan relación con la finalidad del método.

```php
<?php
    require_once "DB.php";
```

```php
public function crear(){}
public function editar(){}
public function eliminar(){}
public function buscarPorId(){}

public function buscarTodas(){
    $db = new DB();
    $query = "SELECT id,nombre,edad FROM personas";
    $sentencia = $db->getConexion()->prepare($query);
    $sentencia->execute();
    $rs= $sentencia->fetchAll();
    foreach($rs as $fila){
        $personas[] = new Persona($fila["id"], $fila["nombre"], $fila["edad"]);
    }
    return $personas;
}
```

Ahora, crearemos al controlador que representará la lógica y los procesos de negocios de la entidad Persona. Y a su vez, prepararemos la presentación de los datos que el usuario visualizará. Recuerda que los controladores deben crearse dentro de la carpeta Controller.

Como vemos en la imagen estamos creando una clase con el nombre Persona Controller, la cual representará al controlador de la clase Persona, con toda la lógica de negocio correspondiente. Ten en cuenta que para crear el nombre de un controlador debes agregar al final la palabra controller. Por ejemplo, si queremos crear un controlador para la entidad libro, su nombre sería libroController.

Más abajo, encontramos el atributo persona, el cual utilizaremos para crear la instancia de la clase persona cada vez que invoquemos a este controlador. Para esto, crearemos la instancia dentro del constructor.
Finalmente, creamos un método llamado buscarTodas, el cual invocará al método buscarTodas de la clase persona, para entregar los datos a una vista, la cual será visualizada por el usuario.

Para la vista, crearemos una página llamada listado_personas.php dentro de la carpeta views, Es en esta página en donde desplegaremos los datos para el usuario. Como vemos, dentro de la página listado_persona.php sólo nos enfocamos en mostrar los datos que el controlador provee. En este caso, estamos recibiendo un array de personas, y desplegamos su contenido en formato tabla.

```php
<h1>Listado de Personas</h1>

<table>
    <tr>
        <td>Id</td>
        <td>Nombre</td>
        <td>Edad</td>
    </tr>
    <?php foreach ($personas as $persona){ ?>
    <tr>
        <td><?php echo $persona->getId(); ?></td>
        <td><?php echo $persona->getNombre(); ?></td>
        <td><?php echo $persona->getEdad(); ?></td>
    </tr>
    <?php
        }
    ?>
</table>
```

Array con listado de Personas enviado por el Controlador

Volviendo al controlador de personaController debemos invocar a la vista listado_personas.php para que ésta pueda utilizar el array de personas que está desplegando por pantalla. Para esto, sólo nos basta con referenciarla.

```php
require_once "/../model/Persona.php";

class PersonaController {

    public $persona;

    public function __construct()
    {
        $this->persona = new Persona();
    }

    public function buscarTodas()
    {
        $personas = $this->persona->buscarTodas();
        include 'view/listado_personas.php';
    }
}
```

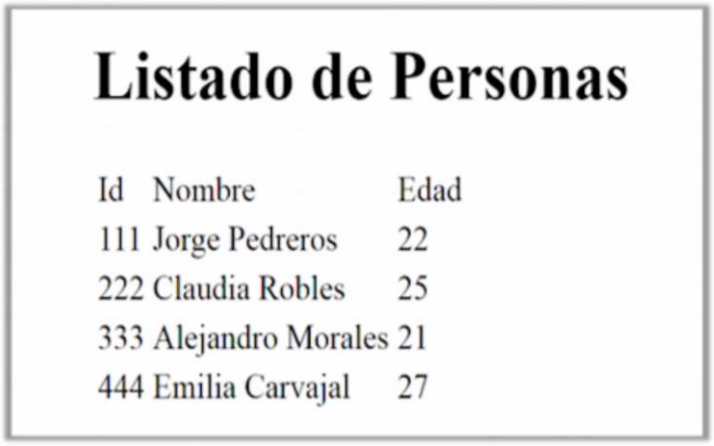
Invocamos a la vista

Finalmente, creamos la página index.php para invocar al listado de personas que provee el controlador personaController. Ésta página la crearemos fuera de las carpetas model, views y controller, como se ve en la imagen.

En la página index, estamos invocando al método buscarTodas , proveniente del controlador personaController, el cual no devolverá un array con el listado de todas las personas de la tabla Personas.

```php
<?php

    require_once "controllers/PersonaController.php";

    $personaController = new PersonaController();

    $personaController->buscarTodas();

?>
```

Finalmente, si ejecutamos nuestra aplicación, veremos el listado de personas en formato tabla. Recuerda insertar datos de prueba sobre la tabla personas, de lo contrario, no visualizarás ningún registro.



Listado de Personas

Id	Nombre	Edad
111	Jorge Pedreros	22
222	Claudia Robles	25
333	Alejandro Morales	21
444	Emilia Carvajal	27

Ahora, agregaremos unos pequeños estilos para las páginas, creando los siguientes directorios dentro de la carpeta Views. La carpeta CSS la utilizaremos para crear nuestros estilos. La carpeta img almacenará las imágenes del sitio. La carpeta js almacenará nuestras librerías javascript que tengamos que utilizar. Y por último, la carpeta sections almacenará las secciones de nuestra página principal.

Comenzaremos creando un archivo css dentro de la carpeta css con los siguientes estilos.

```
body{font-family: "Verdana";
     font-size: 20pt;}

table tr td{
    border:1px solid #000;
}
```

Como puedes ver, estamos agregando unos estilos para las letras o fuentes de las páginas y para las tablas, agregamos un estilo de bordes para sus celdas.

Dentro de la carpeta sections creamos una página php llamada cabecera. Esta contendrá una porción o parte del código fuente de una página html. Aquí sólo dejaremos todo lo referente a la zona superior de una página, desde el tag doctype html, hasta el tag de inicio body. Aprovechamos también de referenciar el archivo css con nuestros estilos.

```
<!DOCTYPE html>
<html>
<head>
    <title></title>
    <link rel="stylesheet" href="views/css/estilos.css">
</head>
<body>
```

Ahora crearemos la otra mitad de página que falta. Esto lo haremos dentro de una página php llamada pie. Como puedes ver sólo debemos crear la parte faltante de nuestra página con los tags de cierre body y html.

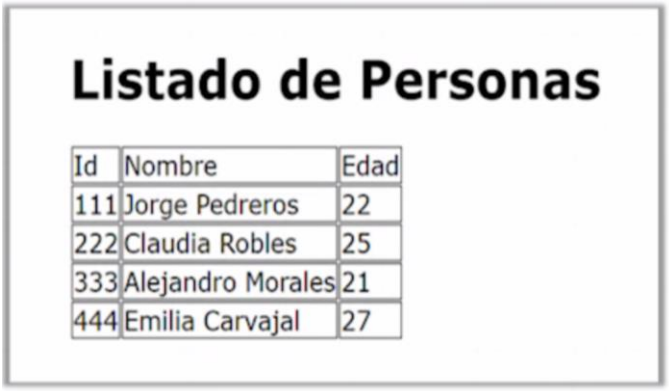
```
</body>
</html>
```

Finalmente, iremos al controlador de la clase persona e invocaremos a las 2 páginas que acabamos de crear entre el listado de persona. Como vemos en la imagen, estamos haciendo 3 invocaciones, que finalmente, crearán un ensamblado de estas. Primero llamamos a la página cabecera, luego al listado de personas, y finalmente, a la página pie, que representa en la zona inferior. Recuerda respetar el orden de las invocaciones, tal y como se muestra en este ejemplo.

```
public function buscarTodas()
{
    $personas = $this->persona->buscarTodas();
    include "views/sections/cabecera.php";
    include 'views/listado_personas.php';
    include "views/sections/pie.php";
}
```

CABEZA
CUERPO
PIE

Si ejecutamos nuestra aplicación veremos el siguiente resultado:

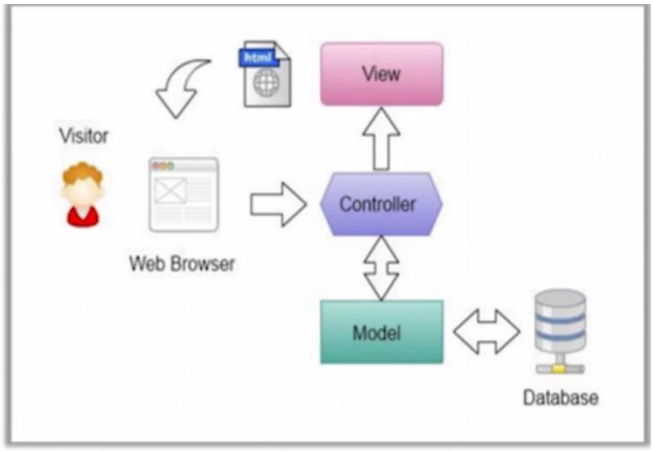


Listado de Personas

Id	Nombre	Edad
111	Jorge Pedreros	22
222	Claudia Robles	25
333	Alejandro Morales	21
444	Emilia Carvajal	27

Como puedes ver, con todos los cambios que realizamos, no se vio afectado su funcionamiento, ya que sólo nos dedicamos a trabajar a nivel presentación, específicamente, sobre las vistas.

En resumen, hemos visto el funcionamiento de cada capa del patrón y cómo, entre estas, se delegan responsabilidades, respetando los distintos niveles de acceso definidos.



Conclusión

En esta unidad conociste el patrón de diseño MVS en un contexto general y cómo se puede aplicar en una aplicación php. Además, pudiste comprender cómo utilizar el paradigma orientado a objetos.

IPLACEX
TECNOLÓGICO NACIONAL