1. INTRODUCCIÓN.

Son muchas las herramientas tecnológicas de PHP utilizadas durante el curso para general una aplicación web dinámica e interactiva. Este documento no contiene tecnologías nuevas para el desarrollo web del lado del servidor, sino metodologías a utiliza para diseñar nuestra aplicación web.

2. ARQUITECTURA WEB ORGANIZADA EN CAPAS Y MVC (REPASO UT1).

Antes de seguir, realizaremos un breve repaso de conceptos vistos en la UT1:

- **ARQUITECTURA WEB ORGANIZADAS EN CAPAS** *(cuando se asocia una función a una componente software)*

Una arquitectura en capas es aquella donde vemos diferentes partes del software cada una con un rol perfectamente diferenciado. Ésta es la forma conceptual (y no necesariamente física) de dividir componentes de un sistema de capas.

Las arquitecturas en capas **dividen la funcionalidad de la aplicación web** para optimizar el uso de recursos. Se consiguen soluciones mucho más **flexibles y escalables**.

La arquitectura web actual define 3 capas:

Capa de presentación
Capa de lógica de negocio
Capa de datos

- ✓ **Capa de presentación:** procesa las peticiones del cliente web y, tras procesarlas, las reenvía a la capa de nivel inferior (capa de lógica de negocio), y viceversa, procesa los datos procedentes de la capa inferior y los adapta al "lenguaje" entendible por la capa de cliente. Funciones asociadas a esta capa:
  - Navegabilidad del sistema
  - Validación de datos de entrada
  - Formato datos de salida
  - Internacionalización
  - Renderizado de presentación,…
- ✓ **Capa de lógica de negocio:** Lo que distingue una **aplicación Web** de un mero sitio Web reside en la posibilidad que ofrece al usuario de actuar sobre la **lógica de negocio en el servidor**. Por lógica de negocio se entiende un conjunto de procesos que implementan las reglas de funcionamiento de la aplicación Web. Por ejemplo, en el caso de una tienda Web, la lógica de negocio son los procesos que implementan el procedimiento de compra: selección de productos, carro de la compra, confirmación de compra, pago, seguimiento de la entrega,...Pero lo importante es que a los **usuarios** se les ofrece la posibilidad de **actuar sobre la lógica de negocio**. Lo cual conlleva que en las aplicaciones Web exista

una **lógica de negocio sensible a las interacciones del usuario;** es el punto donde se puede llevar a cabo la coordinación de transacciones (operaciones de múltiples usuarios). Las operaciones con un uso intensivo de datos deben ejecutarse en este nivel.

- ✓ **Capa de datos o persistencia:** Es la capa que interactúa con los sistemas de información como son las bases de datos (*inserción, borrado, búsquedas y actualizaciones*). Se especializa en dar un servicio de persistencia a los datos de la aplicación y permite manejar grandes volúmenes de ellos.

- **MODELOS ARQUITECTÓNICOS. MODELO-VISTA-CONTROLADOR (MVC)**

Este modelo se utiliza ampliamente en el entorno Web. La división de la aplicación en niveles supone la necesidad de establecer interfaces entre ellas. Al ser una aplicación Web, **la interfaz con el cliente** sigue siendo básicamente **HTTP. Para la interfaz entre el servidor y la base de datos las opciones son múltiples**, dependiendo del lenguaje de programación (Java, C, PHP, VisualBasic,...), el tipo de base de datos (relacional, XML,...), la base de datos concreta (MySQL, Oracle, ...), etc. En cualquier caso con cada configuración hay más de una opción y lo normal es plantearse la elección tras definir la BD o el lenguaje de programación a utilizar.

La estructura de este modelo o patrón arquitectónico es la siguiente:

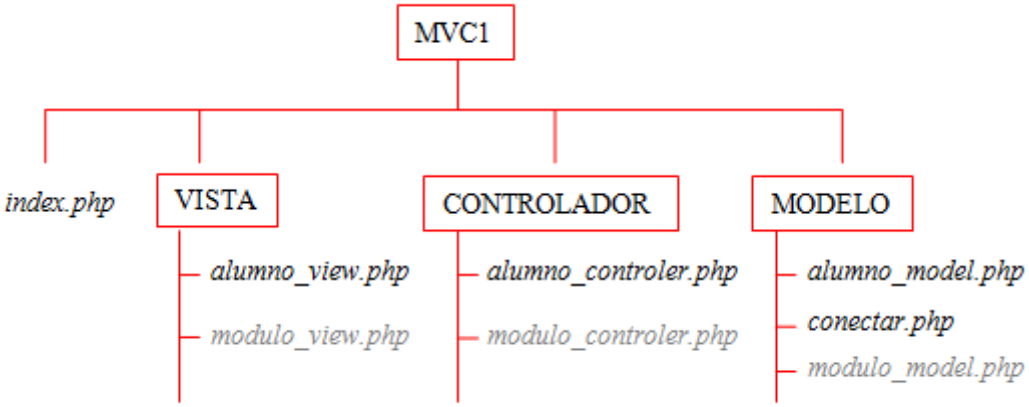
- **Vista:** presentan la información a los actores de la aplicación.
- **Controlador:** gestiona los eventos de los usuarios y permite la comunicación entre la Vista y el Modelo.
- **Modelo:** la lógica de negocio y los datos asociados a la aplicación.

Al mantener buena parte del código en la capa de lógica de negocio, **la aplicación se aísla de la interfaz de usuario y de la base de datos**. Por lo tanto, se pueden hacer cambios en el código en este nivel sin que esto afecte al resto. Además, los **datos** están **centralizados** y fácilmente accesibles sin necesidad de moverlos completamente hasta el cliente.

3. EJEMPLO PRÁCTICO.

La idea sería diseñar un CRUD (Create o insertar, Read o consultar, Update o modificar, Delete o borrar) para la BD ciclo, la cual cuenta con dos tablas (alumno y módulo). En este ejemplo sólo se implementa una de las operaciones del CRUD, en concreto la consulta (Read) de la tabla alumno (los archivos en *negro*)

La estructura de la aplicación (a partir de DOCUMENT_ROOT) sería la siguiente:



El fichero *index.php*, sería el controlador de controladores; una implementación sencilla sería la siguiente:

```php
<?php

/*este archivo será el controlador de los controladores
diseñaríamos un formulario previo para recoger la tabla de la BD con la cual operar (modulo o alumno por ejemplo);
la variable $_POST['tabla'] quedaría seleccionada la opción y a partir de ahí se plantearía un switch en este fichero
similar al siguiente:

switch($_POST['tabla'])
{
    'alumno':*/
        include_once("controlador/alumno_controler.php");
        /*break;
    'modulo':
        include_once("controlador/modulo_controler.php");
        break;
}*/

?>
```

En base a él, el siguiente código que ejecutaría la aplicación sería *alumno_controler.php*:

```php
<?php
require_once("modelo/alumno_model.php");
/* require_once("vista/operacionalumno.php"); sería un formulario
para elegir la operación a realizar sobre los alumnos (CRUD)
swtich ($_POST['alumoper'])
    case 1: para insertar
    case 2: para borrar
    case 3: para modificar
    case 4: para listas */
        $alumno=new Alumno_model();
        $res_consulta=$alumno->get_alumno();
        require_once("vista/alumno_view.php");

?>
```

Desde un controlador, a partir de los eventos recogidos (*en este ej., opción 4 elegida por el usuario a partir de un formulario*) se lleva a cabo el posible tratamiento de los datos obtenidos utilizando un modelo (*alumno_model1.php*). En este ejemplo, el tratamiento es mínimo.

Es en el modelo donde se definen las clases que acceden a la BD. Es frecuente implementar un modelo por cada una de las tablas de la BD manejada e incluso implementar un modelo con una clase abstracta de la cual luego hereden el resto de las clases de los demás modelos con el fin de incluir en ellas, por ejemplo, un método para establecer la conexión a una BD, de tal forma que bastaría con modifica dicha clase si se cambia el gestor de la BD, el nombre de la BD,…

alumno_model.php implementaría la clase alumno_model, la cual incluiría como atributos los campos de la tabla alumno y todos los métodos necesarios.

```php
<?php
// definición de la clase alumno_model, con la cual se accederá a la BD ciclo

class Alumno_model{

private $id_al,$nom,$edad,$curso;
private $bd;
private $row_alumnos;

public function __construct(){
    require_once("Conectar.php");
    $this->bd=Conectar::conexion(); //llamar a un método estático
                                    //de la clase conectar para crear la conexion
    $this->row_alumnos=array();
}

public function get_alumno(){
    $consulta=$this->bd->query("select * from alumno");
    while ($fila=$consulta->fetch(PDO::FETCH_ASSOC)){
        $this->row_alumnos[]=$fila; //se almacena el array resultante
                                    //de la selección en el atributo
                                    //row_alumnos[] de la clase
    }
    return $this->row_alumnos;
}
```

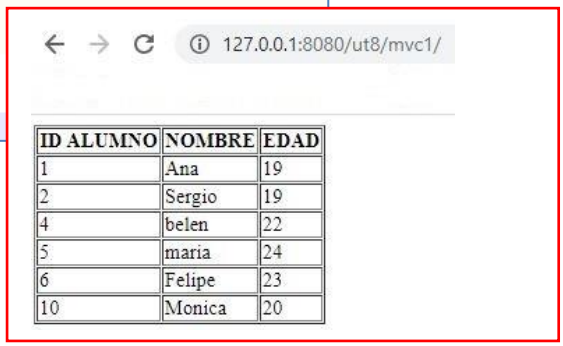
```php
public function delete_alumno($al){
    $borrar=$this->bd->query("delete from alumno where id_al='$al'");
    if (!$borrar)
        $mensaje="Borrado";
    else
        $mensaje="El borrado no se ha realizado";
    return $mensaje;
}

public function insert_alumno ($al,$nom,$ed,$cur){
    $insertar=$this->bd->query("insert into alumno values ('$al','$nom','$ed','$cur')");
    if (!$insertar)
        $mensaje="Insertado";
    else
        $mensaje="La inserción no se ha realizado";
    return $mensaje;
}

}

?>
```

Además, en este ejemplo también forma parte del modelo otra clase, la *clase Conectar* que es utilizada por la *clase alumno_model,* clase que podría ser *abstacta* como se comentó anteriormente. Dicha clase está definida en el fichero *conectar.php*

```php
<?php
class Conectar{

    public static function conexion(){

        try
        {
            $conexion=new PDO('mysql:host=localhost:3308;dbname=ciclo','root','');
            $conexion->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
            $conexion->exec("SET CHARACTER SET UTF8");
        }
        catch(Exception $e)
        {
            //en un sistema en producción, los errores se registrarían en
            //el archivo de errores especificado en la directiva error_log de php.ini
            die("Error".$e->getMessage());
            echo "Linea del error".$e->getLine();
        }
        return $conexion;
    }
}

?>
```

Por último, *alumno_view.php*, incluirá el código asociado a la presentación de los datos. Deberá ser lo más independiente del tratamiento realizado sobre ellos y así conseguir uno de los **objetivos de este patrón arquitectónico que es la facilidad y mejora del mantenimiento de las aplicaciones.**

```php
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
</head>
<body>

<?php
echo "<table border=1>";
echo "<tr><th>ID ALUMNO</th><th>NOMBRE</th><th>EDAD</th></tr>";
foreach ($res_consulta as $al)
{
    echo "<tr><td>".$al['id_al']."</td>";
    echo "<td>".$al['nombre']."</td>";
    echo "<td>".$al['edad']."</td>";
}
echo "</table>";
?>
</body>
</html>
```



ID ALUMNO	NOMBRE	EDAD
1	Ana	19
2	Sergio	19
4	belen	22
5	maria	24
6	Felipe	23
10	Monica	20

- Prueba el código incluido en este documento.
- Implementa las opciones que están incluidas en un comentario del *alumno_controler.php*. Puedes realizarlas por separado.
- Tras la implementación de las opciones anteriores, diseña el alumno_controler.php completo.
- Implementa *modulo_controler.php*