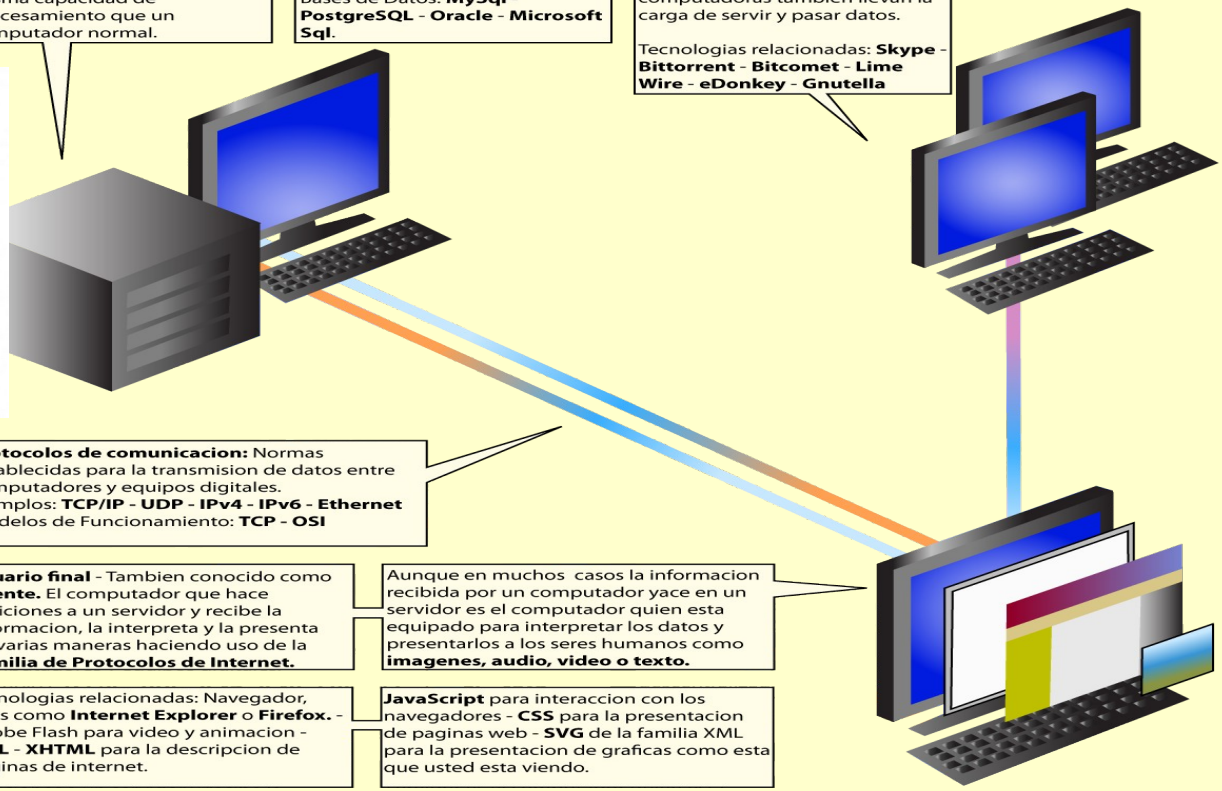
DWES (Desarrollo Web en Entorno Servidor)

Servidor: Computadora especialmente disenada para servir paginas de internet con la misma capacidad de procesamiento que un computador normal.

Tecnologias relacionadas: Scripting: **PHP** - **ASP** - **Java Server Pages**. Bases de Datos: **MySql** - **PostgreSQL** - **Oracle** - **Microsoft Sql**.

Red de Pares (P2P): Redes descentralizadas donde no hay un servidor central sino donde las computadoras tambien llevan la carga de servir y pasar datos.

Tecnologias relacionadas: **Skype** - **Bittorrent** - **Bitcomet** - **Lime Wire** - **eDonkey** - **Gnutella**



Protocolos de comunicacion: Normas establecidas para la transmision de datos entre computadores y equipos digitales. Ejemplos: **TCP/IP** - **UDP** - **IPv4** - **IPv6** - **Ethernet** Modelos de Funcionamiento: **TCP** - **OSI**

Usuario final - Tambien conocido como **agente.** El computador que hace peticiones a un servidor y recibe la informacion, la interpreta y la presenta de varias maneras haciendo uso de la **Familia de Protocolos de Internet.**

Aunque en muchos casos la informacion recibida por un computador yace en un servidor es el computador quien esta equipado para interpretar los datos y presentarlos a los seres humanos como **imagenes, audio, video o texto.**

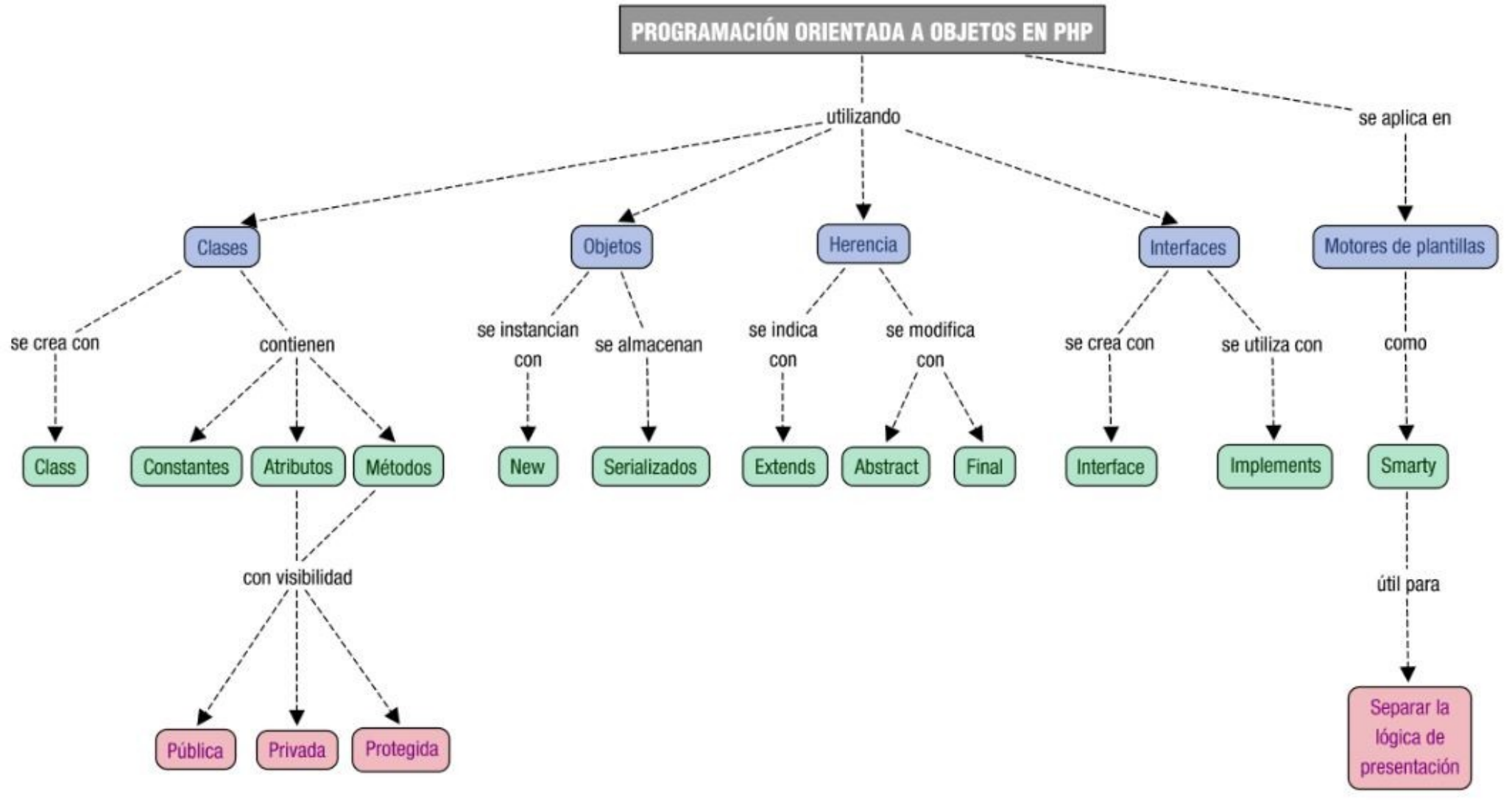
Tecnologias relacionadas: Navegador, tales como **Internet Explorer** o **Firefox.** - Adobe Flash para video y animacion - **XML** - **XHTML** para la descripcion de paginas de internet.

JavaScript para interaccion con los navegadores - **CSS** para la presentacion de paginas web - **SVG** de la familia XML para la presentacion de graficas como esta que usted esta viendo.

Desarrollo web en entorno servidor

Tema 6: Programación Orientada a Objetos en PHP

- Mecanismos de separación de la lógica de negocio de la de presentación. Ventajas.
- Tecnologías asociadas basadas en la programación orientada a objetos y multicapa.
- Controles de servidor.
- Mantenimiento del estado de los controles.
- Mecanismos de generación dinámica del interface Web.
- Diseño de formularios de respuesta.


PROGRAMACIÓN ORIENTADA A OBJETOS EN PHP
utilizando
se aplica en
Clases
Objetos
Herencia
Interfaces
Motores de plantillas
se crea con
contienen
se instancian con
se almacenan
se indica con
se modifica con
se crea con
se utiliza con
como
Class
Constantes
Atributos
Métodos
New
Serializados
Extends
Abstract
Final
Interface
Implements
Smarty
con visibilidad
útil para
Pública
Privada
Protegida
Separar la lógica de presentación

Objetos en PHP5:

- Métodos estáticos métodos de clase
- Métodos constructores y destructores
- Métodos mágicos
- Clonación
- Serialización
- Herencia
- Interfaces
- Clases abstractas

PHP5 NO incluye

- Herencia múltiple
- Sobrecarga de métodos
- Sobrecarga de operadores

Objetos en PHP5:

Creación de clases: utilizaremos un fichero por clase (producto.php)

```
class Producto {
    private $codigo;
    public $nombre;
    public $PVP;

    public function muestra() {
        print "<p>".$this->codigo."<p>";
    }
}
```

Uso de clases:

```
require_once('producto.php');
$p = new Producto();
```

Objetos en PHP5:

Nivel de acceso: utilizado con atributos y métodos

public los atributos o métodos public pueden utilizarse directamente por los objetos de la clase en cualquier sitio donde se instancie el objeto.

private los atributos o métodos solo pueden ser accedidos y modificados por los métodos definidos en la clase, no directamente por los objetos de la clase.

protected atributos o métodos ("privados") visibles en las subclases (heredados)

getAtributo() método que nos permite acceder a un atributo privado

setAtributo() método que nos permite establecer un atributo privado

$this. referencia al objeto que hace la llamada

Métodos mágicos que, si se implementan, son utilizados por defecto

__set() __get() __construct() __destruct() __call() __clone()

Objetos en PHP5:

Nivel de acceso: utilizado con **atributos y métodos estáticos y constantes**

const **constantes de clase**

static **atributos y métodos estáticos** – **atributos y métodos de clase**
(public static o private static)

:: **operador de resolución e ámbito** (-> para objetos :: para clases)

self:: **clase actual** (this-> para objetos self:: para clases)

Utilizaremos self:: para acceder a las constantes de clase y a los atributos y métodos estáticos.

Utilizaremos Clase::metodoEstatico() para acceder a los métodos estáticos y públicos de una clase.

Utilizaremos objeto->metodo() para acceder a los métodos públicos de una clase.

Clases y objetos

Utilización de objetos en PHP5:

Operador instanceof: if ($p **instanceof** Producto)

Funciones útiles para el manejo de objetos y clases:

get_class devuelve el nombre de la clase del objeto

class_exits devuelve true si la clase está definida

get_declared_classes devuelve un array con los nombres de las clases definidas

class_alias crea un alias para una clase

get_class_methods devuelve un array con los nombres de los métodos accesibles

method_exits devuelve true si existe el método (sea accesible o no)

get_class_vars devuelve un array con los nombres de los atributos accesibles

get_object_vars devuelve un array con los nombres de los métodos accesibles

property_exists devuelve true si existe el atributo (sea accesible o no)

Desde PHP5 se puede indicar en las funciones y en los métodos de que clase deben ser los objetos que se pasen como parámetros (especificando el tipo, en este caso la clase, antes del parámetro).

Clonación, alias, serialización, comparación de objetos (I):

Comparación de objetos – si son de la misma clase y tienen el mismo valor en sus atributos (pero son dos objetos distintos)

Comparación de objetos – si son el mismo objetos, dos referencias al mismo objeto

Alias – el mismo objeto con varios nombres – referencias al mismo objeto

Clonar – creamos otro objeto inicialmente igual al clonado (duplicar)

Serialización – almacenamiento de un objeto en un formato no orientado a objeto

Unserialize – recuperar un objeto de su almacenamiento no orientado a objeto

Serialización automática – cuando guardamos objetos en la sesión

Serialización explicita – para guardar objetos en una base de datos relacional

Clonación, alias, serialización, comparación de objetos (II):

= EL operador = crea un nuevo identificador del mismo objeto

clone() Crea un nuevo objeto con el mismo valor en los atributos que el copiado

== dos objetos serán iguales si son instancias de la misma clase y tienen el mismo valor en sus atributos

=== dos objetos serán iguales si son el mismo, si las variables se refieren al mismo objeto (hemos cargado las variables objeto con =)

Serialización

$_SESSION['nombreobjeto']=$miobjeto serializa automáticamente un objeto en el array $_SESSON sobre el elemento nombreobjeto; equivalente a serialize($miobjeto)

$miobjeto=$_SESSION['nombreobjeto'] revierte automáticamente la serialización de un objeto desde el array $_SESSON sobre $miobjeto; equivalente a unserialize()

serialize() unserialize() Utilizados para almacenar y recuperar objetos en/de la base de datos relacional

__sleep() __wakeup() Métodos mágicos asociados a serialize y unserialize

Clases y objetos

POO - Programación Orientada a Objetos:

Objeto: identidad – estado – comportamiento

Clase: molde que define objetos con los mismos atributos (propiedades) y métodos.

Interfaz: contrato de implementación de una clase, solo contiene declaraciones de métodos.

Herencia: se crea una clase a partir de otra, heredando comportamiento y características.

Abstracción: la clase oculta las peculiaridades de su implementación

Polimorfismo: un mismo método puede tener comportamientos distintos en función del objeto con el que se utilice.

Encapsulación: en POO se juntan en el mismo lugar los datos y el código que los manipula.

Herencia (I):

Mecanismo de la POO que nos permite definir nuevas clases en base a otra ya existente.

superclase o clase base – subclase

class *subclase* extends *superclase* {}

Los nuevos objetos de la subclase son también objetos de la superclase.

Utilizaremos la visibilidad **protected** en la superclase con aquellos atributos o métodos privados que queremos que se hereden en la subclase.
Los atributos o métodos privados no se heredan en la subclase.

Polimorfismo: Podemos redefinir los métodos protected creando otro método en la subclase con el mismo nombre.

Funciones asociadas a la herencia:

get_parent_class: Devuelve el nombre de la clase padre del objeto o clase que se le pasa.

is_subclass_of: Devuelve true si el objeto o clase del primer parámetro tiene como clase base la que se indica en el segundo parámetro.

Clases y objetos

Herencia (II):

Mecanismo de la POO que nos permite definir nuevas clases en base a otra ya existente.

superclase o clase base – subclase

final: Podemos evitar la herencia (que no se puedan definir subclases de una clase o que no se pueda redefinir un método en otra subclase) utilizando la palabra final delante de la clase o del método.

final class	en la clase que no queremos que se utilice para subclases
public final function	en el método que no queremos que se redefina en la subclase

abstract: Impide que una clase definida como abstracta se pueda utilizar en un programa directamente, salvo para otra cosa que no sea definir subclases. De igual modo un método abstracto obliga a redefinir dicho método en todas las subclases de la clase que lo contiene.

abstract class	define una superclase abstracta
abstract public function	define un método abstracto en una superclase

Obviamente una clase no puede ser abstracta y final a la vez

Clases y objetos

Herencia (III):

Mecanismo de la POO que nos permite definir nuevas clases en base a otra ya existente.

superclase o clase base – subclase

parent: En PHP5, si la clase heredada no tiene constructor propio, se llamará automáticamente al constructor de la clase base (si existe).
Si la clase heredada tiene constructor propio (en este constructor, si lo consideras necesario, deberás llamar al constructor de la clase base con la palabra parent.

```
...
public function __construct (){
    parent::__construct();
    ...
}
...
```

self:: hace referencia a la clase actual

parent:: hace referencia a la superclase de la clase actual

Interfaces:

Un interfaz es como una clase vacía que solo contiene declaraciones de métodos.
Se definen utilizando la palabra interface (sin la palabra class).

interface *nombreinterface* {}

Se utilizan en las clases que están obligadas a implementar estos métodos (entre otros).

La clase que implementa una interfaz o varias la añade en su definición con la palabra implemets seguida de las interfaces que implementa.

class *Nombredeclase* implements *nombreinterface* {}

Todos los métodos que se declaren en un interfaz deben ser públicos. Además de métodos, los interfaces podrán contener constantes pero no atributos.

Un interfaz es como un contrato que la clase debe cumplir.

En PHP5 se pueden crear interfaces heredando de otros (extends)

Funciones para interfaces:

get_declared_interfaces devuelve un array con el nombre de los interfaces declarados

interface_exists true si existe el interfaz

Interfaces vs Clases abstractas:

En las clases abstractas los métodos pueden contener código; donde programaremos el código común a todas las subclases.

Las interfaces no pueden contener código en los métodos.

Las clases abstractas pueden contener atributos.

Las interfaces no pueden tener atributos.

No se puede crear una clase que herede de dos clases abstractas.

Una clase puede implementar varias interfaces.

Utiliza las clases abstractas cuando tengas atributos y métodos comunes a varias subclases.

Utiliza interfaces para definir los métodos que deben contener las clases que implementen esa interfaz.

Clases y objetos

Ejemplo de POO en PHP:

(Documentado en las páginas 15-18 del tema)

Clases-ficheros:

modeloclases.jpg arbolnavegacion.jpg estructuraalmacenamiento.jpg
creaciondb.sql cargainicial.sql

DB.php: Clase encargada de interactuar con la base de datos, no almacena información, no se instancia ningún objeto de esta clase.

Producto.php: Las instancias de esta clase (objetos) representan los productos que se venden en la tienda.

CestaCompra.php: Instanciamos un objeto de esta clase que contiene los productos adquiridos por el cliente.

login.php
productos.php

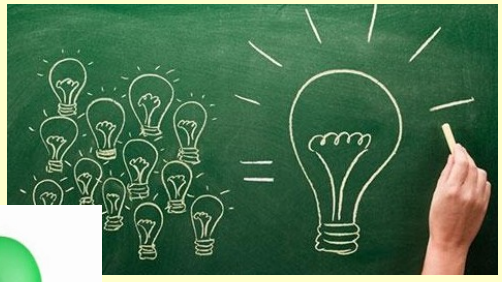
Discusión

Utilización de una clase (DB) que no almacena ninguna información y solamente tiene métodos estáticos… por tanto no se instancia ningún objeto de esta clase nunca.

Clases y objetos

Trabajo colaborativo PHP - POO:

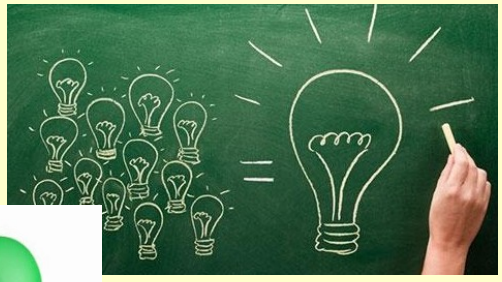
SOLUCIÓN Y PROBLEMA



OBJETIVO
COMUNICACIÓN
MÉTODO
DIRECCIÓN
COMPROMISO
PLANIFICACIÓN
DOCUMENTACIÓN
SEGUIMIENTO

RECOMPENSA

Trabajo colaborativo PHP - POO:



Coordinador	Santiago
Integración y pruebas	Juanjo
Clases-ficheros:	
• modeloclases.jpg	Fran
• arbolnavegacion.jpg	David G.
• estructuraalmacenamiento.jpg	Saul
• creaciondb.sql	Rome
• cargainicial.sql	Pablito
• DB.php:	David F.
• Producto.php:	Patri
• CestaCompra.php:	Jorge
• login.php	Pablo M.
• productos.php	Samuel

Trabajo colaborativo PHP - POO:

Coordinación – comunicación – dirección – liderazgo
Documentación
modeloclases.jpg
arbolnavegacion.jpg
estructuraalmacenamiento.jpg
Desarrollo
Integración y pruebas
Implantación y aceptación

Desarrollo
Base de datos
creaciondb.sql (Usuario, Producto)
cargainicial.sql
Modelo
DB.php
Usuario.php
Producto.php
CestaCompra.php
Controlador
Vista
login.php index.php
tienda.php
micesta.php
.css fonts

Programación en capas - Aplicaciones web multicapa

El patrón Modelo Vista Controlador MVC divide el código en 3 capas

Modelo encargado de manejar los datos propios de la aplicación
obtener y modificar la información
almacenar, recuperar y tratar la información almacenada en la base de datos

Vista encargado de la interacción con el usuario
interfaz con el usuario - HTML

Controlador recoge las solicitudes del usuario
solicita trabajo al modelo
indica a la vista lo que debe mostrar al usuario

Ejemplo de POO del patrón MVC en PHP:

(Documentado en las páginas 28-33 del tema)

Discusión I: MVC páginas 29-30

index.php	**Controlador**
vista.php	**Vista**
modelo.php	**Modelo**

Discusión II: Multicapa páginas 31-32

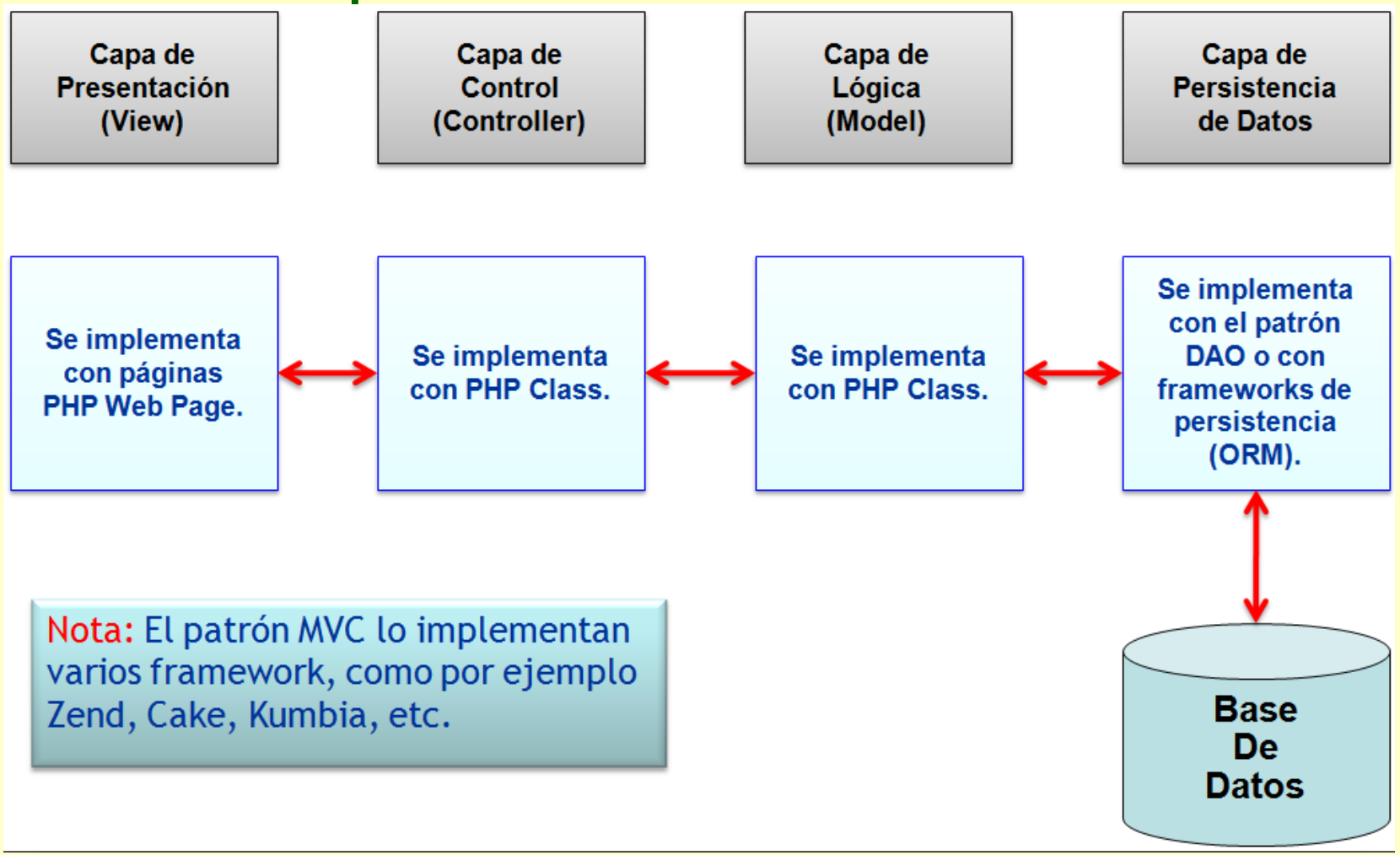
index.php	**(Controlador) frontal** – punto de entrada a la aplicación
controladorpag.php	**(Controlador) de página**
layout.php	**(Vista) Layout genérico**
logicalayout.php	**(Vista) Lógica de la vista genérico**
miplantilla.php	**(Vista) Plantilla**
modeloDB.php	**(Modelo) Capa de acceso a los datos** especifica para la base de datos
modeloabstracto.php	**(Modelo) Capa de abstracción de la base de datos**

Discusión III: Multicapa - POO páginas 33

objetos de la vista – objetos del controlador – clases del modelo

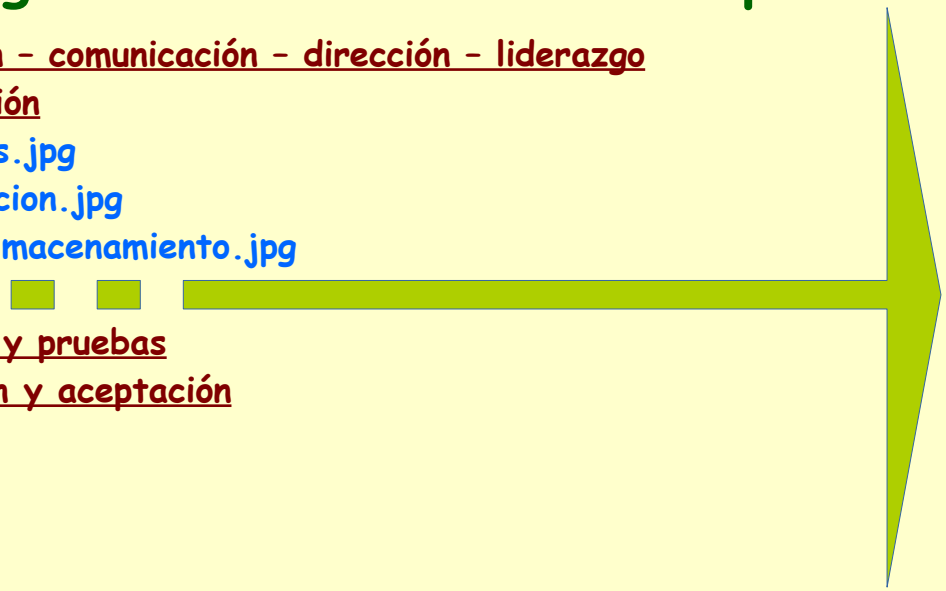
Clases y objetos

PHP POO Multicapa:



LoginLogoff PHP – POO - multicapa:

Coordinación – comunicación – dirección – liderazgo
Documentación
modeloclases.jpg
arbolnavegacion.jpg
estructuraalmacenamiento.jpg
Desarrollo
Integración y pruebas
Implantación y aceptación

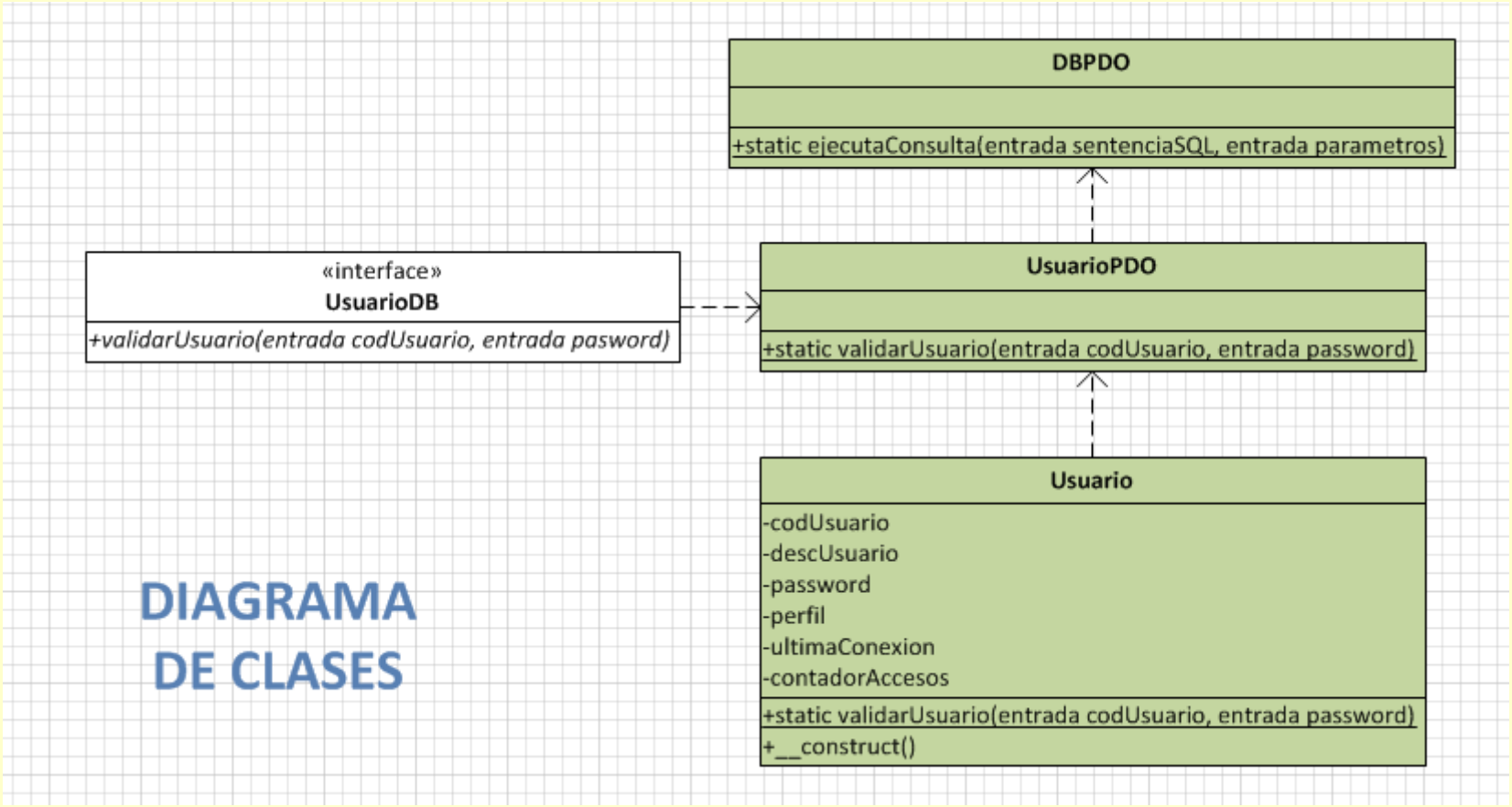


Desarrollo
Base de datos
creaciondb.sql (Usuario, Producto)
cargainicial.sql
Modelo
DBPDO.php constDB.php
UsuarioPDO.php ProductoPDO.php
Usuario.php Producto.php
Controlador
index.php
clogin.php
cinicio.php
Vista
layout.php
vlogin.php
vinicio.php
.css fonts/ js/

LoginLogoff PHP – POO - multicapa:

Documentación

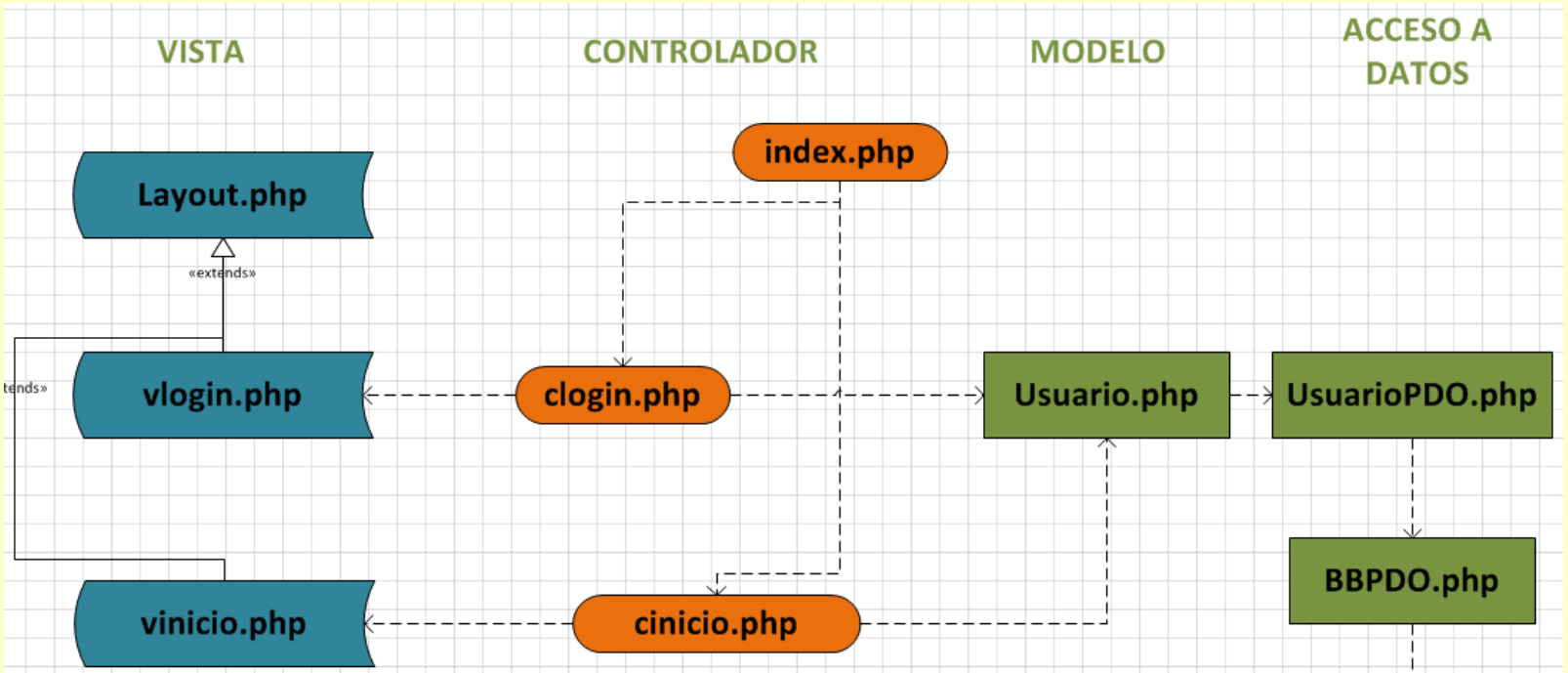
modeloclases.jpg

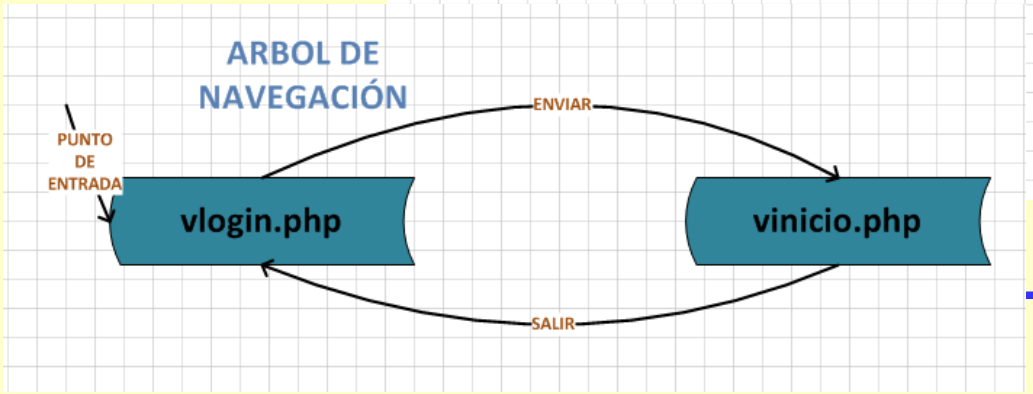


LoginLogoff PHP – POO - multicapa:

Documentación

arbolnavegacion.jpg



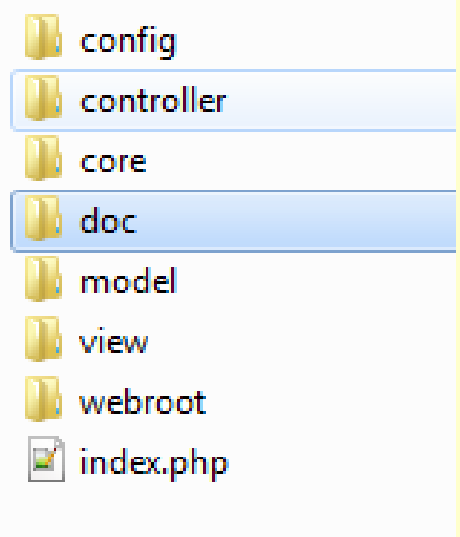


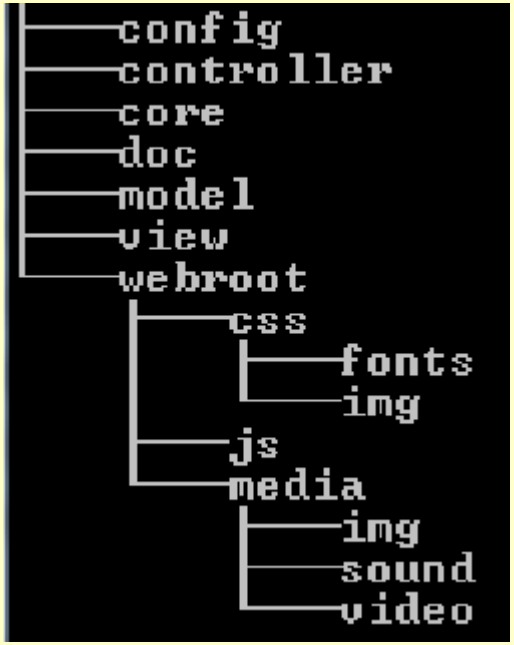
APLICACIÓN MULTICAPA

LoginLogoff PHP – POO - multicapa:

Documentación

estructuraalmacenamiento.jpg





Programación en capas – Aplicaciones web multicapa

Generación de código - Herramientas

Framework o marco de desarrollo implementación concreta del MVC

PHP Framework marcos de desarrollo para PHP

comparativa – características – porcentaje de mercado

CakePHP – Symfony - Laravel *Web Framework*

Motor de plantillas web aplicación que: utilizando un fichero con la información de la presentación: plantilla o template (vista del MVC) y otro con la lógica interna de la aplicación (modelo MVC), genera una página web

Smarty

Symfony

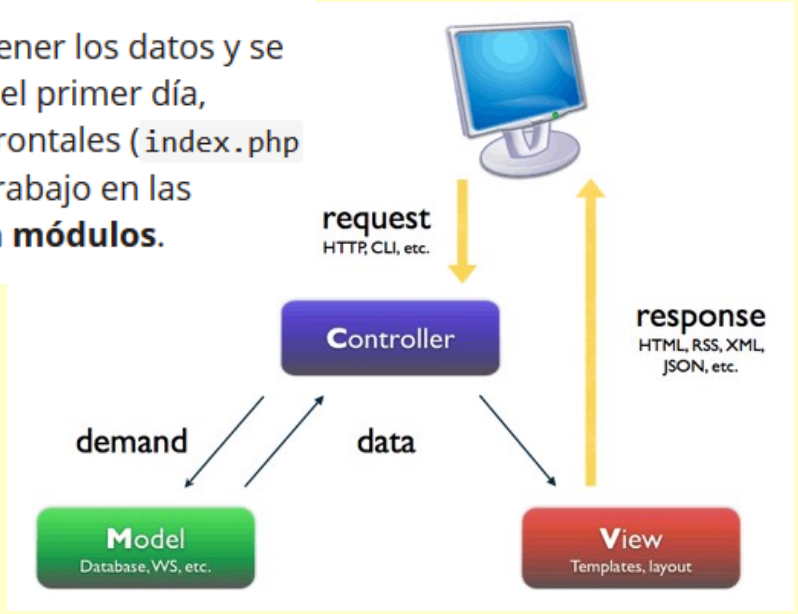
Framework o marco de desarrollo para aplicaciones basadas en PHP

CakePHP – Symfony - Laravel

Librosweb *Web Framework*

MVC en Symfony:

- La capa del **modelo** define la lógica de negocio (la base de datos pertenece a esta capa). Como ya sabes, Symfony guarda todas las clases y archivos relacionados con el modelo en el directorio `lib/model/`.
- La **vista** es lo que utilizan los usuarios para interactuar con la aplicación (los gestores de plantillas pertenecen a esta capa). En Symfony la capa de la vista está formada principalmente por plantillas en PHP. Estas plantillas se guardan en varios directorios llamados `templates/` repartidos por todo el proyecto, tal y como veremos hoy mismo.
- El **controlador** es un bloque de código que realiza llamadas al modelo para obtener los datos y se los pasa a la vista para que los muestre al usuario. Cuando instalamos Symfony el primer día, explicamos que todas las peticiones se canalizan a través de los controladores frontales (`index.php` y `frontend_dev.php`). Estos controladores frontales realmente delegan todo el trabajo en las **acciones**. Como vimos ayer, las agrupaciones lógicas de acciones se denominan **módulos**.



Teoría LibrosWeb

MVC en PHP - CRUD

Realizando un CRUD con el patrón MVC en PHP

¿Que es MVC?: Es un **patrón de arquitectura de software** que busca **desacoplar la lógica en 3 capas**:

- El **controlador se encarga de recibir las peticiones que manda la vista**.
- **El modelo** es el que **define las reglas de negocio** en nuestro caso las **consultas a MySQL**.
- La **vista es lo que va a ver el usuario** que en nuestro caso la **maquetación en HTML**.

Distribución de nuestras carpetas

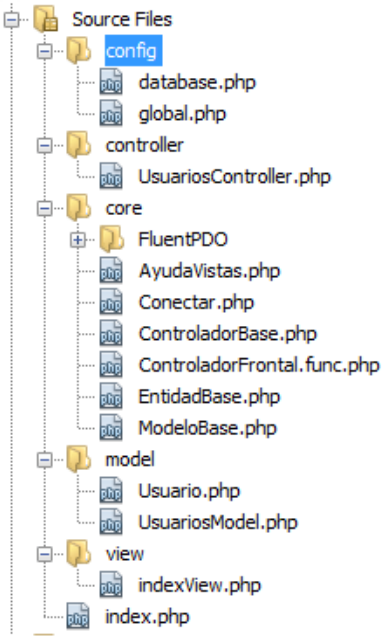
- **Controller**, contendra a todos nuestros controladores. Este esta compuesto por acciones, que en código son métodos/funciones.
- **Model**, contendra a nuestros modelos y entidades de negocio
- **View**, contendra a nuestras vistas (código HTML)
- **index.php**, sera nuestro **FrontController**

CRUD Anexsoft

MVC en PHP:

Estructura de directorios

Ejemplo PHP + POO + MVC

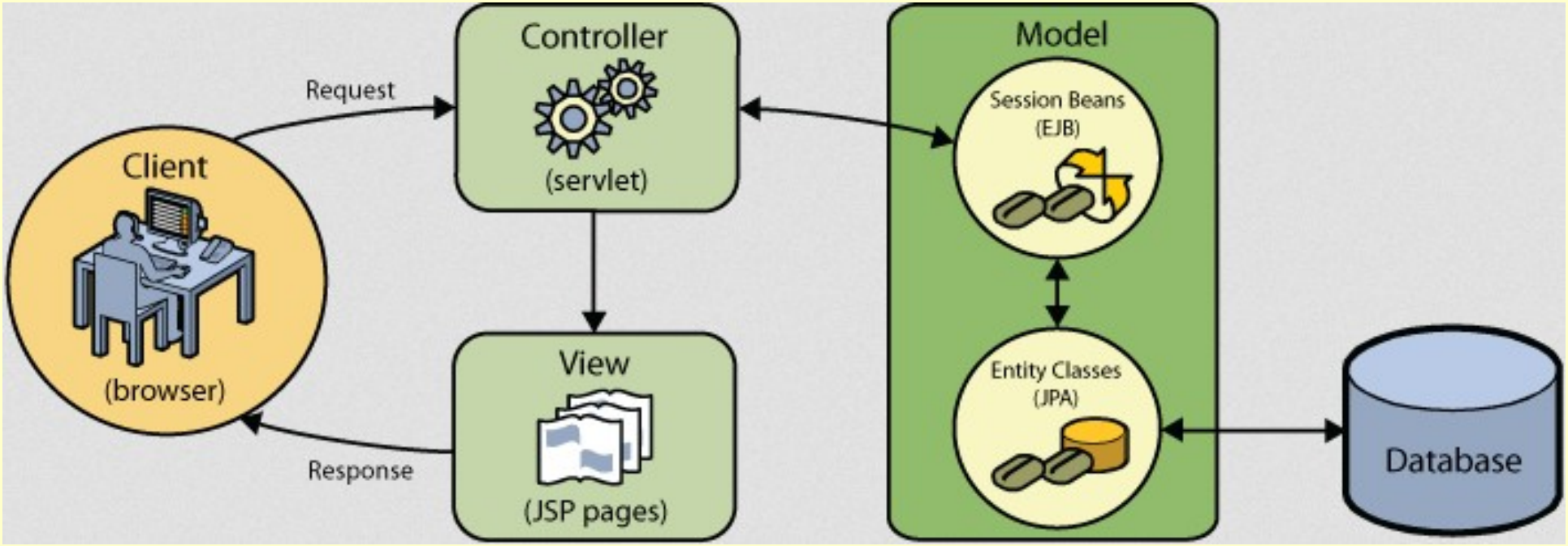


En nuestro "framework" tendremos varios directorios:

- **config**: aquí irán los ficheros de configuración de la base de datos, globales, etc.
- **controller**: como sabemos en la arquitectura MVC los controladores se encargarán de recibir y filtrar datos que le llegan de las vistas, llamar a los modelos y pasar los datos de estos a las vistas. Pues en este directorio colocaremos los controladores
- **core**: aquí colocaremos las clases base de las que heredarán por ejemplo controladores y modelos, y también podríamos colocar más librerías hechas por nosotros o por terceros, esto sería el núcleo del framework.
- **model**: aquí irán los modelos, para ser fieles al paradigma orientado objetos tenemos que tener una clase por cada tabla o entidad de la base de datos(excepto para las tablas pivote) y estas clases servirán para crear objetos de ese tipo de entidad(por ejemplo crear un objeto usuario para crear un usuario en la BD). También tendremos modelos de consulta a la BD que contendrán consultas más complejas que estén relacionadas con una o varias entidades.
- **view**: aquí iran las vistas, es decir, donde se imprimirán los datos y lo que verá el usuario.
- index.php será el controlador frontal por el que pasará absolutamente todo en la aplicación.

Ejemplo Víctor Robles

MVC en J2EE:

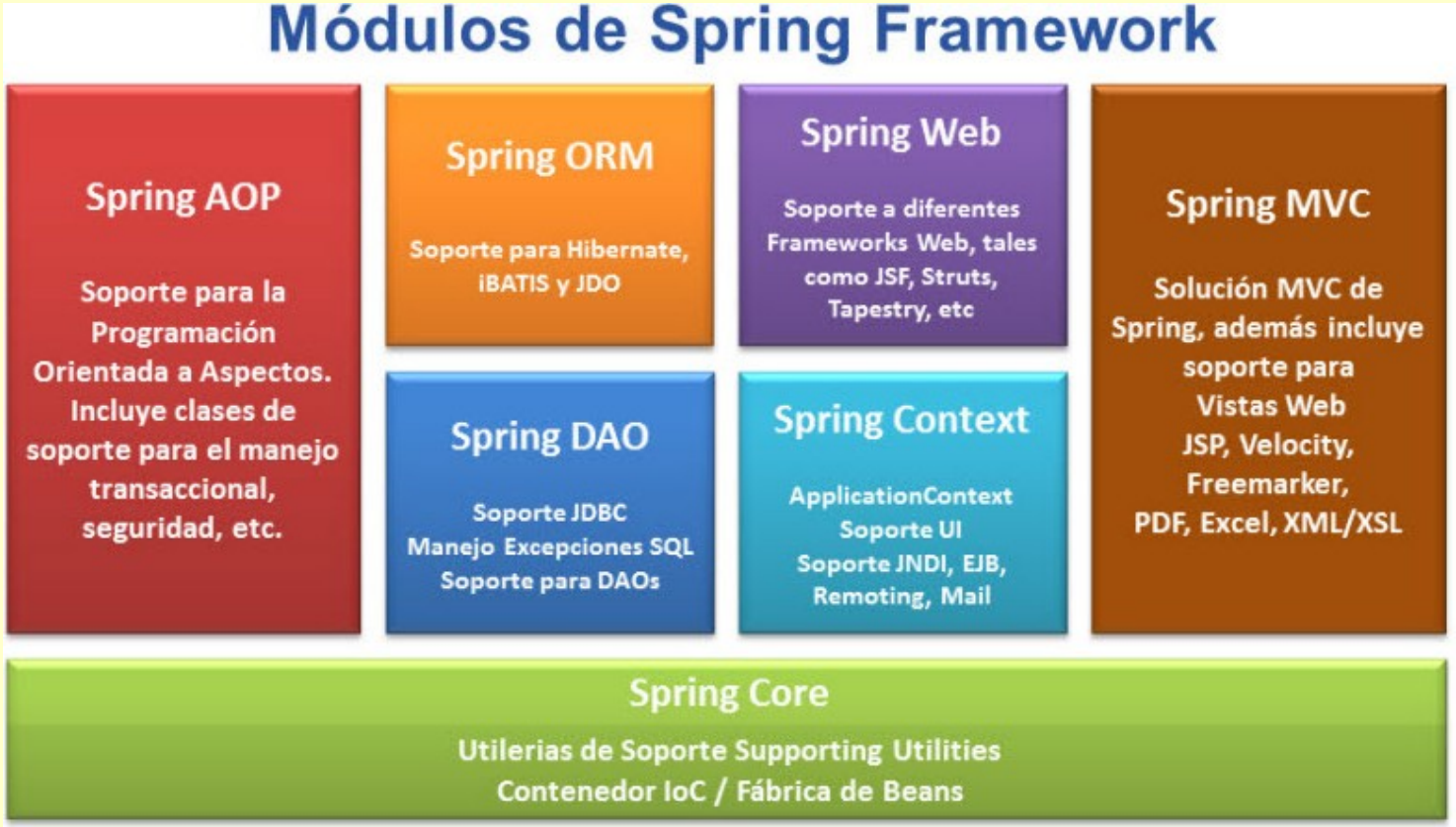


NetBeans

java T point

Spring Framework - Framework MVC en J2EE:

Spring
Java Application Framework



Spring Framework

java T point

¿Preguntas?