Arquitectura:

Sistema de información: Es un conjunto de elementos “digitales” orientados al tratamiento y administración de datos e información, organizados y listos para su uso posterior, generados para cubrir una necesidad o un objetivo de una organización o individuo. Cumplen ciertas actividades:

|  |  |
| --- | --- |
| 1. Recopilación: captura o recolecta datos. | 1. Almacenamiento: guarda de forma estructurada la información recopilada. |
| 1. Procesamiento: convierte esa entrada de datos en una forma más significativa. | 1. Distribución: transfiere la información procesada a las personas o roles que la usaran. |

Sistema WEB: Es un sistema diseñado y desarrollado para que funcione a través de Internet, está basado en una arquitectura “Cliente - Servidor”. Utiliza tecnologías WEB para entregar información o servicios a otros sistemas o usuarios.

Arquitectura “CLIENTE-SERVIDOR”: Es la arquitectura dominante en la WEB. Los clientes y los servidores pueden trabajar con distintas tecnologías y diferentes sistemas operativos.

|  |  |
| --- | --- |
| ¿Que hacen un cliente?  Es quien inicia las solicitudes, espera y recibe las respuestas del servidor. Este puede conectarse a varios servidores a la vez e interactúa mediante una interfaz gráfica con el usuario.  Ejemplo: Navegador | ¿Que hace un servidor?  Espera a que llegue la solicitud de los clientes, tras la recepción de una solicitud, la procesan y luego envían la respuesta.  Acepta conexiones de un gran número de clientes.  Ejemplo: Apache |

Una arquitectura (servidor-cliente) WEB cuenta con:

|  |  |
| --- | --- |
| Navegador: Hace de cliente y realiza peticiones al servidor. | Servidor: Administra y responde las peticiones de clientes o de otros servidores. |
| API: Es la forma que se comunican a través de internet. | Protocolo HTTP: es el protocolo de comunicación entre cliente y servidor. |

Ventajas y desventajas Cliente-Servidor:

|  |  |
| --- | --- |
| Ventajas: | Desventajas: |
| Recursos centralizados (ej: bbdd) Fácil mantenimiento. Red escalable (posible quitar o agregar clientes sin afectar el funcionamiento) | Costo elevado. Al caerse el servidor se cae deja sin servicio para los clientes. Congestión del tráfico cuando hay múltiples peticiones simultáneas. |

Diseño de una arquitectura: La arquitectura le da “estructura” a la aplicación, permite analizar y diseñar sin programar los principales problemas que podemos tener en nuestra aplicación.

API (application programming interface): Define datos y cómo acceder a ellos. Permite la utilización desde el exterior del sistema. La implementamos con procedimientos, funciones y métodos Junto a API el REST es el estándar más lógico, eficiente y habitual en la creación de APIs para servicios de internet..

Paralelismo - Concurrencia: Existen múltiples usuarios conectados a nuestro sitio, por ello debe analizarse los posibles problemas de paralelismo (codigos que si se ejecutan en paralelo podrían dar resultados incorrectos). Muchos de estos problemas pueden resolverse con más y mejores equipos (pero no todos ellos. De todas formas esta solución pueden traer otros problemas)

Introducción a PHP: Las app web funcionan sobre un esquema “cliente-servidor”. El cliente realiza peticiones a un servidor, quien devolverá una respuesta.

Sus ventajas: Fácil de mantener y actualizar. Multiplataforma.

Programación Server-Side: Es el componente principal de las páginas dinámicas, donde un servidor recibe las solicitudes de los clientes, para luego procesarlas y devolver una respuesta.

-No requiere instalación en el servidor.

-No requiere instalación ni complementos adicionales en el cliente.

PHP: Lenguaje server-side de código abierto muy popular especialmente adecuado para el desarrollo web. Es un lenguaje interpretado, diseñado para producir sitios web dinámicos. Es soportados por múltiple plataformas. El código es procesado por el intérprete que genera la página web resultante.

|  |  |
| --- | --- |
| PHP-Variables | PHP-Constantes |
| Es un lenguajes no tipeado: el tipo se define por el contexto en el que es usado. No es necesario declararlas. El nombre siempre empieza con $. | Este valor no puede variar durante a ejecución del script. Se usa el método “define(nombre, valor)” y para leerlas usamos el nombre sin el $. |

Modelo de datos: (persistencia, archivos, base de datos, sql)

Persistencia: Es la acción de preservar la información de un objeto de forma permanente (ej: disco rigido) y de recuperar la misma para que pueda ser nuevamente utilizado.

Base de datos: Es una herramienta para recopilar y organizar información, un contenedor de objetos para almacenar tablas que guardan datos interrelacionados. La usamos cuando la cantidad de datos es excesiva. Cuando haya redundancia (para evitar el duplicado de código)

SQL: Es un lenguaje de consulta estructurado. Vinculado a la gestión de bases de datos, permite crear y modificar esquemas, tablas e índices. Insertar, borrar y actualizar tablas en una relación.

Comunicación de los datos:

Cifrado: Los mensajes viajan en paquetes que pueden ser interceptados en la red y sin estar cifrados se pueden leer fácilmente, de esta forma se vuelve vulnerable todo el intercambio de información al no tener un cifrado.

Cifrado WEB:

|  |  |  |
| --- | --- | --- |
| Firma digital: Se genera con base a la clave privada de quien firma y por lo tanto es “única” | Autoridad certificada: Es confiable y se encarga de garantizar (al user) un certificado digital de quien el user dice ser. | Certificado Digital: Documento digital que garantiza vinculación entre una persona entidad con su clave pública. |
| Cifrado (no encriptado): proceso que transforma la información y se realiza con base a un único elemento (clave). | Clave pública y clave privada: Son claves asociadas a una persona o entidad generada en métodos criptográficos. La pública cifra información y la privada para descifrar. | HTTPS: Combinación del protocolo HTTP con el protocolo SSL/TLS para establecer comunicaciones cifradas en sitios web. |

Verificación de validez del certificado: Una vez que el navegador tiene el certificado, realiza algunas verificaciones:

* Integridad del certificado: Verifica que se encuentre íntegro, esto lo hace descifrando la firma digital incluida en el mediante la clave pública de la AC.
* Vigencia del certificado: Revisa el periodo de validez.
* Verifica emisor del certificado: Hace uso de una lista de certificados raíz almacenados en tu computadora que contienen las claves públicas de la ACs conocidas.

Route:

GET: muestra los parámetros en la url. POST: no se ven los parámetros.

Pretty URLs (Amigables o URLs semánticas): Son más fáciles de entender para los usuarios, mejoran el posicionamiento web. Proporcionan información sobre la estructura del sitio y son mas esteticas.

Routing: Este se encarga de determinar el PATH a donde redireccionaremos. Consiste en poder usar los datos de la URL como si fueran parámetros que se le pasan al script PHP.

Necesitamos un route que atienda TODOS los request, configurar Apache para que las URL las envíe siempre a un único archivo (route).

.htaccess: Es un archivo de configuración de Apache HTTP web server. Indicamos que URLs van a que archivo de php. Este archivo va en la carpeta base o la de URL base a rutear. Redirigimos la solicitud a un único archivo.

MVC:

Patrón de arquitectura de software utilizado en ampliamente en la industria. Delega responsabilidades, deja un codigo prolijo, da estabilidad y facilita al mantenimiento. Cada componente está hecho de clases.

MVC = Model - View - Controller

|  |  |  |
| --- | --- | --- |
| Modelo: | Vista: | Controlador: |
| Protege y persiste los datos del usuario. Asegura la integridad y consistencia de datos. Provee métodos para consultar datos (bbdd), insertar/modificar datos o borrar datos. | Presenta la información al usuario (front-end). Delega la responsabilidad del desarrollador de cómo se van a mostrar los datos.. (Se genera la interfaz de usuario). | Obtiene y procesa los pedidos del usuario. Procesa y valida los inputs de usuario. Controla el flujo de la aplicación. (Es el intermediario entre la vista y el modelo). |

Ventajas y desventajas de MVC:

|  |  |
| --- | --- |
| Ventajas de MVC: | Desventajas MVC: |
| Crea un sistema desacoplado, reduce la complejidad de cada parte del sistema. Facilita la escalabilidad y mantenimiento. | Agrega complejidad a la solución. La estructura predefinida puede no ser lo que estábamos buscando. |

Refactor: Es una técnica de la ingeniería de software para “reestructurar un código fuente, mejorando así su estructura interna sin cambiar su comportamiento externo”

-código más bonito, que hace lo mismo”

Templates:

Los templates engine son “herramientas que se utilizan para separar la lógica del programa y la presentación del contenido en dos partes independientes”. Separa la lógica de la vista, es decir, separa el código (php) de la presentación (html).

Lógica: La lógica de una aplicación es la parte del código que realiza todo lo referido a la obtención, almacenamiento procesamiento de los datos para entregarlos a una vista que sabe cómo visualizarlos. Es el “detrás de escena” para presentar los datos en pantalla.

|  |  |
| --- | --- |
| Ventajas: | Desventajas: |
| Facilita el desarrollo tanto de la lógica como de la presentación, mejora la flexibilidad y facilita la modificación y el mantenimiento. |  |

Smarty: Es un motor de plantillas para PHP, es rápido y eficiente. La plantilla es compilada solo una vez. Este usa una combinación de etiquetas HTML y etiquetas de plantilla. De esta forma mantiene separada la presentación y así el acoplamiento es menor. La lógica consiste en obtener la información de una bbdd, etc, la procesa y se asigna el contenido a mostrar (archivo php). Un template recibe la información y la muestra.

Caching: Acelera la carga. Al llamar a “display()” se guardará una copia de la salida a un archivo, el siguiente llamado utilizará la copia guardada.

Autenticación de usuarios:

Cookies: Es una pequeña información enviada por un sitio web y almacenada por el navegador del usuario, de manera que el sitio web puede consultar la actividad previa del usuario. Viven de lado del navegador, son fáciles de usar, editables y tienen tiempo de expiración.

Local Storage: Se utilizan para almacenar pares de clave-valor en el lado del cliente. Estos pares de clave-valor se pueden recuperar en páginas que provengan del mismo dominio. Los datos de almacenamiento local almacenan en el disco y se conservan después de reiniciar las aplicaciones. Cualquiera puede verlo y cualquiera puede modificarlo.

POO , PDO:

Usamos POO para evitar duplicar código (ya que es mala práctica), el código siempre cambia y crece (por nuevos requerimientos y corrección de fallas) pero no la lógica.

Una característica que lo define es “abstracción”: el cual es un proceso de identificar características y funcionalidades que hacen a un objeto particular: ya sea estado del objeto definido por “atributos” específicos, métodos propios y relaciones con otros objetos.

Clases e instancias:

|  |  |
| --- | --- |
| Clase: es la definición formal y abstracta de una entidad de un problema. | Objeto: Es una instancia de una clase, con atributos particulares. |

Tipo de acceso a variables:

Las variables de clase pueden ser:

* Privadas: solo accedidas por el objeto de manera interna.
* Públicas: accedidas por todos
* Protegidas: por algunos.

Al usar objetos tenemos 2 partes: La definición de la clase (dice cómo hace las cosas) y la declaración de objetos (crea un objeto, y al llamar métodos hace cosas).

Constructor: El el código que se ejecuta siempre que utilizamos la palabra “new” ($this->view = new revistaView();). Usamos el método “\_\_construct()” y dentro se coloca los estados de variables a inicializar.

Herencia: En POO, se basa en la herencia de clases o implementación de interfaces. Se redefinen “métodos” que tienen un comportamiento distinto, con el mismo nombre y la misma cantidad de parámetros. Para sobreescribir los métodos, es necesario que el número de argumentos DEBEN ser iguales.

Web Services:

Son componentes de una aplicación específicos para intercambiar información entre aplicaciones. Permiten la interoperabilidad (intercambio de información) entre plataformas de distintos fabricantes por medio de protocolos estándar y abiertos

Existen dos tipos de protocolos:

|  |  |
| --- | --- |
| SOAP | REST |
| Simple Object Access Protocol, muy usado en sistemas corporativos. | Representational State Transfer muy usado en la internet. |

API REST: Arquitectura de desarrollo web que se apoya totalmente en el estándar HTTP, proporciona una API que utiliza cada uno de los métodos del protocolo. Es el tipo de arquitectura más natural y estándar para crear APIs para servicios orientados a Internet, la mayoría usa JSON para comunicarse, puede acceder o modificar, mediante los métodos del protocolo HTTP (POST, GET, PUT, DELETE).

API: Application Programming Interface.

Es una interfaz que permite la utilización desde el exterior de sistema. Definen datos y cómo acceder a ellos. Implementada como procedimientos, funciones, objetos y métodos, y servicios web. Ejemplos:

|  |  |
| --- | --- |
| GET/factura | Accede al listado de facturas. |
| POST/factura | Crea una factura nueva. |
| GET/factura/:id\_fact | Accede al detalle de una factura |
| PUT/factura/:id\_fact | Edita la factura, sustituyendo la totalidad de la información anterior por la nueva. |
| DELETE/factura/:id\_fact | Elimina la factura. |

Ventajas y desventajas de los servicios:

|  |  |
| --- | --- |
| Ventajas | Desventajas: |
| Cada servicio puede actualizarse independientemente, puede programarse en diferentes lenguajes, facilita la reutilización | Comunicación y coordinación, el costo de cambiar la especificación de servicios. Complejidad de cambiar las API y de planificar cambios a futuro. |

Final 2017:

La bicicleteria quiere un sitio web para publicar sus productos. Su base de datos deberá almacenar los datos de las bicicletas (marca, rodado, precio) que deberán estar clasificadas en categorías: Mountain Bike, Playera, Retro, etc.

1. Definir las tablas necesarias de la base de datos de la bicicletería. De acuerdo a esto, implemente las funciones del modelo (indicando nombre de clase) para obtener.
   1. Los datos de una bicicleta
   2. Cantidad de bicicletas de una categoría
   3. Lista de bicicletas por menos de $10000
   4. Lista de bicicletas de cierto rodado
   5. Lista de categorías

RTA:

Tablas DDBB:

|  |  |
| --- | --- |
| Tabla 1: Categorías. | Tabla 2: Bicicletas. |
| id\_categorias. A.I  categorias. | id\_categorias INDEX  marca  rodado  precio  id\_bicicleta |

bicicletaModel.php

|  |
| --- |
| class bicicletaModel(){ private $db; contruct(){ $this->db-> |
| 1. function getBicicleta($id){   $sentencia = $this->db->prepare(“SELECT \* FROM bicicletas WHERE id\_bicicleta = ?); $sentencia->execute(array($id)); return $sentencia->fetch(PDO::ASSOC); } |
| 1. function cantidadCat(){ $sentencia = $this->db->prepare(“SELECT bicicletas.\* categorías.\* from categorias WHERE id\_categorias = ? AND bicicletas.id\_categorias = categorias.id\_categorias”); $sentencia->execute(): return $sentencia->fetch(PDO::ASSOC); |
| 1. function Precio(){ $sentencia = $this->db->prepare(“SELECT \* FROM bicicletas WHERE precio<10000”) $sentencia->execute(); return $sentencia->fetchAll(PDO::ASSOC); |
| 1. function bicicletaRodado($id){ $sentencia = $this->db->prepare(“SELECT \* FROM bicicletas WHERE rodado = ?”); $sentencia->execute(array($id)); return $sentencia->fetchAll(PDO::ASSOC); |
| 1. function getCategorias(){ $sentencia = $this->db->prepare(“SELECT \* FROM categorias”); $sentencia->execute(); return $sentencia->fetchAll(PDO::ASSOC); |

1. Implementar la clase correspondiente al Controller para un servicio REST para obtener los datos de una bicicleta. Llame a la función del modelo que implementó anteriormente.

|  |
| --- |
| <?php  require\_once (“./model/bicicletaModel”);  require\_once (./api/JSONview);  class apiController{  private $model;  private view;  \_\_construct(){  $this->model = new bicicletaModel();  $this->view = new JSONview();  }  function getBicicleta($params = null){  $id = $params[‘:ID’];  $bicicleta = $this->model->getBicicleta($id);  if ($bicicleta){  $this->view->response($bicicleta, 200);  }else{  $this->view->response(“No hay ta bicicleta”, 404”)  }}} |

1. Implementar la vista y template Smarty para mostrar la lista de bicicletas. Mostrar un icono (“imgs/retro.png”) solo en bicicletas que son retro.

|  |  |
| --- | --- |
| <?php  class bicicletaView(){  private $smarty  \_\_construct(){  $this->smarty = new Smarty();  }  function getBicicletas($bicicletas,$imgs){  $this->smarty->assign(‘bicicletas’,$bicicletas);  $this->smarty->assign(‘imgs’,$imgs);  $this->display(‘templates/bicicletas.tpl’); | <!DOCTYPE html>  <html>  <body>  {include file=”header.tpl”}  <table>  <tbody>  {foreach $bicicletas as $bicicleta}  <tr>  <td>{$bicicleta[’marca’]}</td>  <td>{$bicicleta[‘rodado’]</td>  <td>{$bicicleta[‘precio’]</td>  <td>{$bicicleta[‘id\_categoria’]</td>  {/foreach}  </tr>  </tbody>  </table> |

5. Marque verdadero o falso y justifique brevemente TODAS sus respuestas. Puede emplear un ejemplo para la justificación.

1. En MVC las clases de la vista necesitan conocer el modelo para obtener para obtener los datos a presentar.
   1. Falso, las vistas no necesitan conocer el modelo ya que el controlador se encarga de buscar la información en el modelo y luego pasarla al view junto a los parámetros necesarios (cada uno con su function correspondiente).
2. Usar PDO asegura conexiones seguras.
3. PDO permite interoperabilidad entre bases de datos.
4. Una sesión de usuario se crea y mantiene en el cliente.
   1. La sesión vive del lado del servidor. PHP SESSION()
5. Consultas con muchos datos se resuelven siempre en la base de datos.
6. El cliente y el servidor en una arquitectura distribuido se comunican mediante una API REST.
   1. Verdadero. La API funciona como mediadora entre ambos. Al ser el cliente y el servidor, sistemas independientes y totalmente autonomos, podemos programar cada parte incluso con por diferentes equipos de programación.
7. La búsqueda de una bicicleta en la bbdd se programa en el controller.
   1. Verdadero: El controller hace un request al model y view, en una function hace el pedido al model (bbdd) donde retorna algo, ese algo lo pasamos por parametro a la funcion que llama al view para mostrarlo por pantalla.
8. Para que el usuario no pueda borrar una bicicleta se debe ocultar el botón en smarty
   1. Habria que colocar un lock(); (no lo van a tomar, lo explicaron hoy 21/10)
9. Mostrar la marca de una bicicleta se hace en la vista.
   1. Verdadero: Se muestra en el template junto a los tags necesarios, previamente en el view debería colocarse el assign del pedido de la bbdd
10. Filtrar las bicicletas por categoría solo se puede hacer en SQL.
    1. Se hace en el model, donde en la función usamos código de SQL.
11. Explicar los inconvenientes que causa la siguiente función del modelo
    1. PHP solo se ejecuta con XAMPP
       1. No es necesario que sea XAMPP, cualquier servidor (que soporte php) puede usarse.
    2. Las contraseñas siempre hay que guardarlas encriptadas en la base de datos usando el algoritmo RSA.
       1. Falso, hay muchos algoritmos de encriptación. Lo que si, las contraseñas siempre deben ser guardadas usando uno de ellos, cada uno tiene sus debilidades y fortalezas.
    3. La única forma de recibir datos de una API REST es en formato JSON
       1. Falso, también puede ser en XML, por ejemplo. JSON es la mejor opción ya que podemos interpretar los datos con diferentes lenguajes y tecnologías, es como el formato universal.
    4. Una sesión de usuario se crea y mantiene en el cliente
       1. Falso. Depende del método de sesión que se utilice. En el caso de las Cookies y el Local Storage, la información se guarda del lado del cliente (En las cookies puedo guardar solamente el nombre de usuario del lado del cliente) y en el caso de la PHP SESSION la sesión vive del lado del servidor y dura mientras el usuario está conectado.
    5. En MVC las vistas necesitan conocer el Modelo para obtener los datos a presentar.
       1. En MVC el único que debe conocer el modelo es el controlador, el cual hace el llamado al model y el envio al view (junto al dato traído del model)
    6. La búsqueda de un empleado en la base de datos se programa en el controller.
       1. El pedido a base de datos se hace en el “model” el cual el llamado aparece en controller junto a la función ubicada en el modelo.
12. DESCRIBA de manera simple pero completa, todo el proceso involucrado en el ruteo de una página específica dentro de un sitio WEB. Explique la necesidad y ejemplifique con un caso. Comente si se requiere realizar cambios cuando se implementa una API REST..

Ejercicios Tipo Parcial:

1. Indique si las siguientes premisas son verdaderas o falsas justificando apropiadamente.
   1. Las clases del Modelo necesitan conocer el Controlador en MVC.
   2. Falso. Es el controlador quien conoce y llama las funciones del Modelo, las cuales le retornan un valor determinado.
   3. Los servidores de servicios REST mantienen estados.
   4. Falso. Los servicios REST son stateless. Cada request del cliente debe contener toda la información necesaria para entender el pedido.
   5. Usar PDO asegura conexiones seguras.
   6. no tengo ni a mas recontra re putisima idea
   7. Smarty permite interoperabilidad entre bases de datos.
   8. Falso, Smarty es una plantilla para presentar el contenido al cliente, y mantener una separación entre código PHP y HTML.
   9. Los datos de un usuario se almacenan encriptados utilizando MD5.
   10. Falso. Los datos de un usuario siempre deben almacenarse encriptados pero hay distintos algoritmos que pueden ser utilizados.

|  |
| --- |
| Controller Tarea:  **<?php**  **class Tareacontroller{**  **private $model;**  **private $view;**  **function \_\_construct(){**  **$this->model = new tareasModel();**  **$this->view = new tareasView();**  **}**  function getTareas(){  $tareas = $this->model->getTareas();  $this->view->showTareas($tareas);  }  function getTarea($id){  $tarea = $this->model->getTarea();  $this->view->showTarea($tarea);  }  function insertTarea(){  $this->model->insertTarea($\_POST[‘titulo’],$\_POST[‘descripcion’],$\_POST[‘fecha’])  header(“location: “. tarea);  }  function deleteTarea($id){  $this->model->deleteTarea($id);  header (“location: “ . tarea);  }  function editTarea($id){  $this->model->editTarea($id, $\_POST[‘titulo’],$\_POST[‘descripcion’],$\_POST[‘fecha’]);  header (“location: “ . tarea); |

|  |
| --- |
| Model Tarea:  **<?php**  **class tareasModel{**  **private $db;**  **function \_\_construct(){**  **$this->db = new PDO(‘....’);**  **}**  function getTareas(){  $sentencia = $this->db->prepare(“SELECT \* FROM tarea”);  $sentencia->execute();  $tareas = $sentencia->fetchAll(PDO::FETCH\_ASSOC);  return $tareas;  }  function getTarea($id){  $sentenca = $this->db->prepare(“SELECT \* FROM tarea WHERE id=?”);  $sentencia->execute();  $tarea = $sentencia->fetch(PDO::FETCH\_ASSOC);  return $tarea;  }  function insertTarea($título,$descripción,$fecha){  $sentencia = $this->db->prepare(“INSERT INTO tarea(título, descripción, fecha) VALUES(?,?,?)”);  $sentencia->execute(array($título,$descripción,$fecha);  }  function deleteTarea($id){  $sentencia = $this->db->prepare(“DELETE FROM tarea WHERE id=?”);  $sentencia->execute(array($id));  }  function editTarea($id,$titulo,$descripcion,$fecha){  $sentencia = $this->db->prepare(“UPDATE tarea SET título=?, descripcion=?, fecha=? WHERE id=?);  $sentencia->execute(array($título,$descripcion,$fecha,$id));  } |

|  |
| --- |
| View tareas:  <?php  require\_once (‘libs/smarty...’);  class TareasView{  private $smarty;  function \_\_construct(){  $this-smarty = new Smarty();  }  \*\*\*\* FUNCIÓN DESEADA($parámetros) {  $this->smarty->assign(\*\*\*\*‘parámetros a usar’, $parámetro traído\*\*\*\*);  $this->smarty->display(\*\*\*\*‘template a usar’\*\*\*\*);  } |

|  |  |
| --- | --- |
| Template:  {include file=’header.tpl’)  <table>  <thead>  <tr>  <th>Título</th>  <th>Descripcion</th>  <th>Fecha</th>  </tr>  </thead>  <tbody>  <tr>  {foreach $parametro as $parametroS}  <td> <{$parametros[‘titulo’]</td>  <td> <{$parametros[‘descripcion’]</td>  <td> <{$parametros[‘fecha’]</td>  {/foreach}  </tr>  </tbody> | </table>  {include file=”footer.tpl”} |