PROCESO DE GESTIÓN DE FORMACIÓN PROFESIONAL INTEGRAL

FORMATO GUÍA DE APRENDIZAJE

IDENTIFICACIÓN DE LA GUIA DE APRENDIZAJE

- **Denominación del Programa de Formación:** TGO Análisis y Desarrollo de Sistemas de Información
- **Código del Programa de Formación:** 228106 V102
- **Nombre del Proyecto:** Construcción de un sistema de información que cumpla con los requerimientos del cliente en procesos que se lleven a cabo en el sector productivo del departamento de Caldas
- **Fase del Proyecto:** IMPLEMENTACIÓN
- **Actividad de Proyecto:** Seleccionar la alternativa de solución que cumpla con los requerimientos establecidos por el cliente
- **Competencia:** Construir el sistema que cumpla con los requisitos de la solución informática.
- **Resultados de Aprendizaje Alcanzar:** Realizar la codificación de los módulos del sistema y el programa principal, a partir de la utilización del lenguaje de programación seleccionado, de acuerdo con las especificaciones del diseño
- **Duración de la Guía:** 40 horas

2. PRESENTACIÓN

*PHP e*s un lenguaje de programación del lado del servidor y de alto nivel, embebido en páginas HTML. Ha tenido una gran aceptación en la comunidad de desarrolladores, debido a la potencia y simplicidad que lo caracterizan, así como al soporte generalizado en la mayoría de los servidores de hosting, hasta los más simples y económicos.

Otra de las claves del éxito de PHP es que la mayoría de los CMS más populares (WordPress, Joomla!, Drupal) y los sistemas de comercio electrónico (Prestashop, Woocommerce, Magento), así como otros cientos de herramientas, están desarrollados en PHP. Por lo tanto, usar PHP es sinónimo de ser capaz de introducirnos en muchas herramientas gratuitas y de código abierto para realizar cualquier cosa en el ámbito de la web.

3. FORMULACIÓN DE LAS ACTIVIDADES DE APRENDIZAJE

- **Descripción de la(s) Actividad(es):**
    - **Actividades de aprendizaje:**
        - Nueva sintaxis de PHP 8

- Dominar la creación y uso de funciones en PHP para modularizar el código y reutilizarlo.
- Familiarizarse y utilizar variables superglobales como $_GET, $_POST y $_SESSION
- Crear formularios HTML y procesar los datos enviados por los usuarios utilizando PHP 8
- Conexión y manipulación de bases de datos MySQL incluyendo la ejecución de consultas SQL y la protección contra ataques de inyección SQL
- Comprender los conceptos de POO en PHP, como clases, objetos, propiedades y métodos
- Manejo de errores y excepciones de manera efectiva en PHP para depurar y mejorar la robustez del código
- Implementar sistemas de autenticación de usuarios y control de acceso en aplicaciones web PHP 8.

- **Actividad de Reflexión inicial**

Actividad de reflexión inicial

Antes de comenzar nuestro proyecto en PHP 8, los aprendices deberán conocer la evolución de PHP. Puede parecer que no, pero PHP ha cambiado muchísimo en este tiempo. Vamos a ver cómo era construir una clase con PHP 5.3 y cómo lo es con PHP 8.2:

De PHP 5.3 a PHP 8.2
https://www.youtube.com/watch?v=BHAYO6esXlw

- **Actividades de contextualización e identificación de conocimientos necesarios para el aprendizaje**

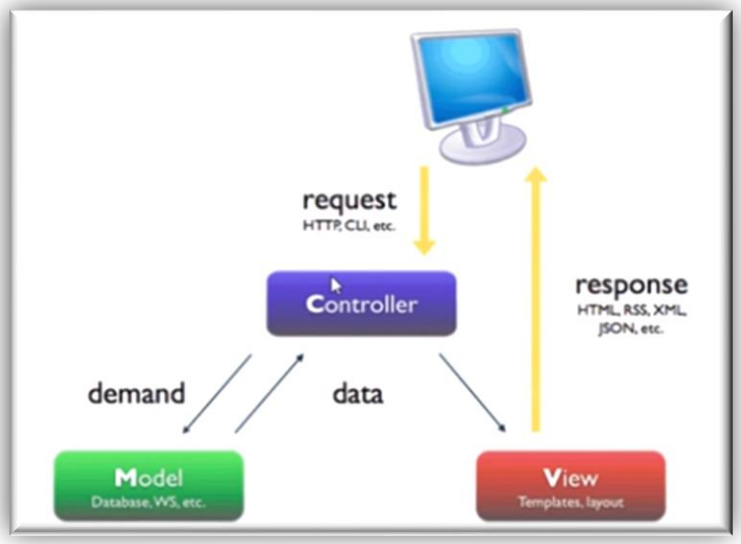
Definición de la arquitectura del proyecto formativo

Para la siguiente actividad los aprendices deberán tener muy presentes los conceptos ya vistos de Programación Orientada a Objetos y el patrón de arquitectura de software: **MVC** (Modelo Vista Controlador)

El MVC (Modelo-Vista-Controlador) es un patrón de arquitectura de software, no un patrón de diseño de software. La diferencia radica en su alcance y nivel de abstracción.

Un patrón de diseño de software se enfoca en solucionar problemas específicos a nivel de diseño en la construcción de componentes de software. Los patrones de diseño, como el patrón de diseño *Singleton* o el patrón de diseño *Observer*, proporcionan soluciones a problemas recurrentes en el diseño de software.

Por otro lado, un patrón de arquitectura de software aborda la estructura general y la organización de un sistema a un nivel más alto. Define cómo se deben organizar los componentes de un sistema, cómo se comunican entre sí y cómo se gestionan los flujos de datos. El patrón MVC es un ejemplo de un patrón de arquitectura de software, ya que establece una estructura y una separación de responsabilidades para los componentes de una aplicación.



Actividad 1

Los aprendices deberán seguir el siguiente paso a paso para construir un proyecto PHP utilizando Programación Orientada a Objetos y un Patrón de Arquitectura de Software MVC. Con la ayuda del instructor deberá quedar muy clara toda la implementación ya que se utilizará como base para desarrollar los proyectos formativos con esta estructura. Para ello, los aprendices crearán un nuevo proyecto llamado: "**Senamás_EPS**"

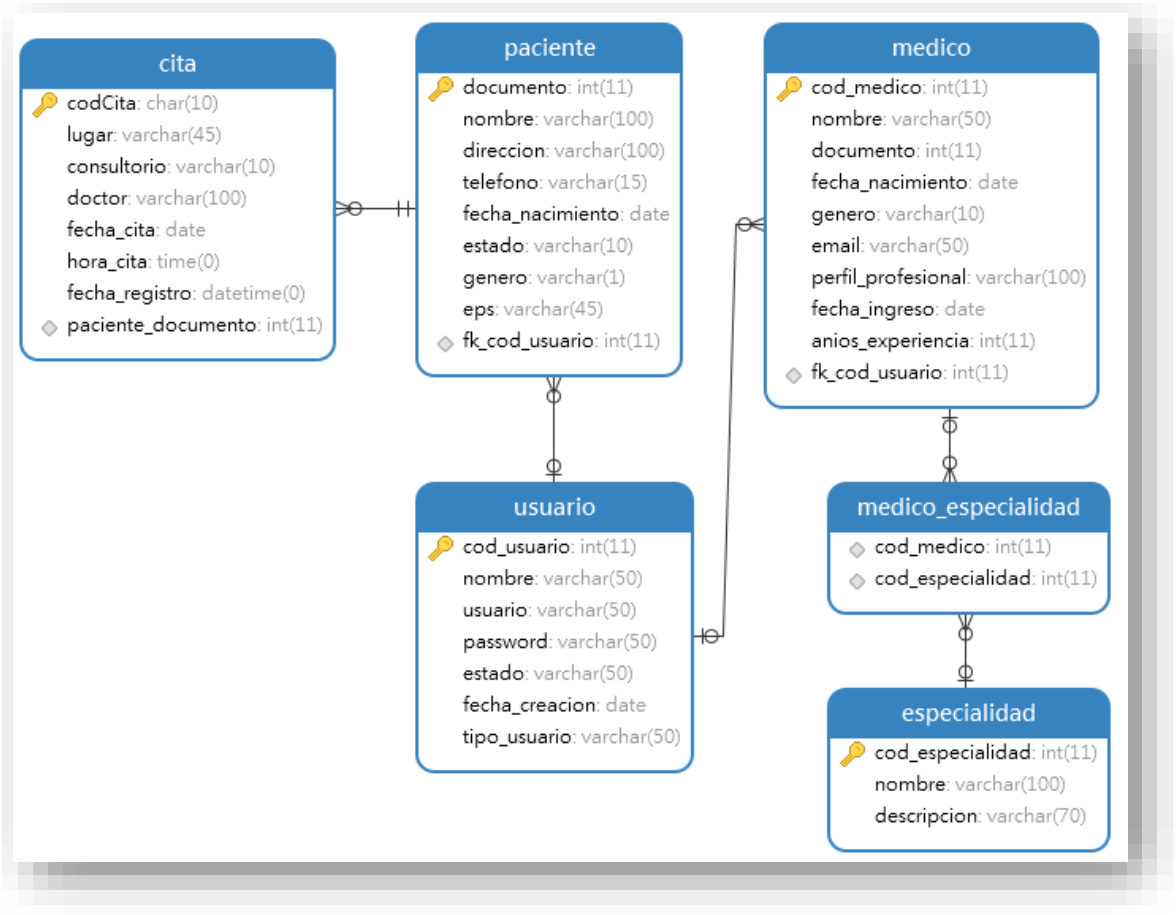
Estructura del proyecto con patrón de arquitectura MVC

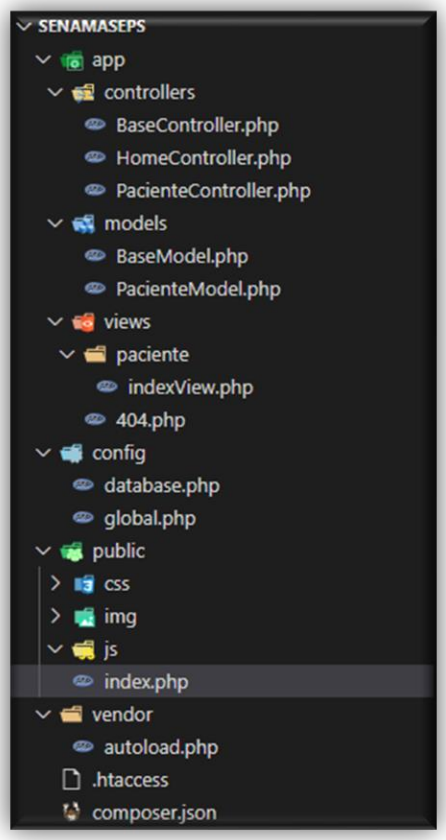
Nuestro proyecto tendrá la siguiente estructura de carpetas y archivos principales, los cuales serán creados desde el editor de texto o IDE seleccionado (Para esta guía se utilizará *Visual Studio Code*):

- **app:** Este directorio contendrá los controladores, modelos y vistas de la aplicación.
    - **controllers**: en este directorio se almacenarán los controladores, los cuales manejan las solicitudes y acciones del usuario. 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.
        - **BaseController.php:** clase padre de la cual pueden heredar otros controladores. Allí irán funcionalidades generales que son comunes entre controladores
        - **HomeController.php:** ejemplo inicial de un controlador específico
        - **PacienteController.php**: controlador para la gestión de pacientes.
    - **models**: aquí irán los modelos, que representan la lógica de negocio y se encargan de interactuar con la base de datos u otros servicios. Siguiendo el paradigma orientado objetos debemos tener una clase por cada tabla o entidad de la base de datos 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.
        - **BaseModel**: es una clase padre de la cual otros modelos pueden heredar.
        - **Paciente.class.php:** es un ejemplo de un modelo específico la tabla: paciente
    - **views**: aquí se ubicarán las vistas, que se encargan de la presentación y la interfaz de usuario.
- **public:** este directorio es accesible públicamente y contendrá los archivos estáticos de la aplicación, como CSS y JavaScript
    - **css**: carpeta para adicionar todos los estilos de la aplicación
    - **js**: carpeta para adicionar todos los archivos JavaScripts de la aplicación
    - **img:** carpeta que contendrá todas las imágenes de los diferentes layouts o plantillas de la aplicación
    - **index.php:** este archivo será el punto de entrada de la aplicación y se encargará de direccionar las solicitudes a los controladores requeridos
- **vendor:** Este directorio contendrá las dependencias externas instaladas a través de Composer
- **.htaccess:** archivo de configuración utilizado en servidores Apache para reescribir las URLs y redirigir todas las solicitudes a `index.php`.
- **composer.json:** archivo de configuración de Composer para gestionar las dependencias del proyecto
- **config**: aquí irán los archivos de configuración globales, etc.
    - **database.php:** archivo que contiene un array con todos los parámetros para conexión a una base de datos

Estructura de la base de Datos

A continuación se muestra la estructura de tablas que tendrá nuestra base de datos para la aplicación. Inicialmente empezaremos solamente con lo que tiene que ver con la tabla `paciente`, e iremos progresivamente incluyendo las demás tablas para el funcionamiento completo dentro de nuestra aplicación.





Creación de ficheros de configuración

En el directorio `config`, crearemos los diferentes ficheros de configuración de la aplicación web. Inicialmente crearemos un archivo llamado `config/global.php`**.** Este archivo contendrá la definición de las constantes globales que aplicarán a toda la aplicación:

```php
<?php
define('MAIN_APP_ROUTE', __DIR__.'/../app/');
```

La función `__DIR__` en PHP devuelve la ruta absoluta del directorio actual del archivo en el que se encuentra. Proporciona la ubicación completa del directorio que contiene el archivo en ejecución. Es útil para construir rutas de archivos y directorios de forma dinámica y relativa. Se puede

Luego crearemos el archivo **`config/database.php`**, el cual devolverá un array con los datos que permitirán realizar la conexión a la Base de Datos. Más adelante veremos otras formas de guardar estos parámetros sensibles de acceso:

```php
<?php
return [
    'driver' => 'mysql',
    'host' => 'localhost',
    'database' => 'senamaseps',
    'username' => 'root',
    'password' => '',
    'charset' => 'utf8mb4',
    'collation' => 'utf8mb4_unicode_ci'
];
```

Creación de las clases principales de la aplicación

Primero crearemos la clase **`BaseModel (app/models/BaseModel.php)`** que nos servirá para conectarnos a la base de datos utilizando el driver **`PDO`**. Luego también se podría implementar en otra versión conexión a bases de datos con un constructor de consultas como por ejemplo **`Eloquent`** (el que utiliza Laravel o algún otro ORM).

De esta clase heredarán los modelos que representen entidades. Tendremos tantos métodos como queramos para ayudarnos con las peticiones a la BD a través de los objetos que iremos creando. Manejarlo de este modo tiene la ventaja de que estos métodos pueden ser reutilizados en otras clases ya que le indicamos la tabla con la que queremos trabajar.

```php
<?php

namespace App\Models;

use PDO;
use PDOException;

abstract class BaseModel
{
    protected $dbConnection; # Atributo que contiene la conexión a la BD con PDO
    protected $table; # Representa la tabla actual

    public function __construct()
    {
        $dbConfig = require_once MAIN_APP_ROUTE . "/../config/database.php";
        try { # Establecer la conexión a la base de datos
            $dsn = "{$dbConfig['driver']}:host={$dbConfig['host']};dbname={$dbConfig['database']}";
            $username = $dbConfig['username'];
            $password = $dbConfig['password'];
            # Se crea un objeto de la clase PDO para la conexión a la BD
            $this->dbConnection = new PDO($dsn, $username, $password);
            $this->dbConnection->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_SILENT);
        } catch (PDOException $e) {
            die('Error en la conexión: ' . $e->getMessage());
        }
    }
}
```

En este mismo archivo crearemos la función común `getAll()` como método común para traer todos los registros de una tabla de la base de datos:

```php
    public function getAll():array
    {
        try {
            $query = "SELECT * FROM $this->table";
            $statement = $this->dbConnection->query($query);

            // Obtener los resultados como un array asociativo
            $resultSet = $statement->fetchAll(PDO::FETCH_OBJ);

            // Devolver los elementos de la tabla
            return $resultSet;
        } catch (PDOException $ex) {
            die("Error en consulta> " . $ex->getMessage());
        }
    }
    /*
    * Aqui podemos incluir los demás métodos que nos ayuden
    * a hacer operaciones con la base de datos de forma común
    * a todas las tablas
    */
} # End Class
```

La siguiente clase que crearemos en el proyecto, es el controlador base, la clase `BaseController (app/controllers/BaseController.php)` de la cual heredarán los controladores. Esta clase permite

tener los médotos que son comunes a todos los controladores. También se pueden tener otras funciones que nos resulten útiles como por ejemplo renderización, validaciones, redirecciones, o formateo de números:

```php
<?php

namespace App\Controllers;

class BaseController
{
    protected function render($view, $data = []): void
    {
        // Lógica para renderizar la vista
        //extract($data);
        foreach ($data as $key => $value) {
            # Se extraen y crean las variables traidas en $data
            $$key = $value;
        }
        include_once MAIN_APP_ROUTE.'views/' . $view . 'View.php';
    }

    protected function formatCurrency($amount)
    {
        return '$' . number_format($amount, 2);
    }

    protected function redirectTo($url)
    {
        header('Location: ' . $url);
        exit();
    }
}
```

Archivo public/index.php

Es nuestro archivo de entrada única a la aplicación. Incluirá inicialmente el archivo `vendor/autoload.php` que se encargará de llamar e incluir automáticamente, las clases que se vayan requiriendo, adicionalmente agregará parámetros de configuración global y hará el respectivo manejo de las rutas.

```php
<?php
require_once '../vendor/autoload.php'; // Cargar clases de forma automática
require_once '../config/global.php'; // Configuración general

$url = $_SERVER['REQUEST_URI']; // Saber que llega en la URL

# Rutas válidas en nuestra aplicación
$routes = [
    '/' => [
        'controller' => 'App\Controllers\HomeController',
        'action' => 'index'
    ],
    '/home' => [
        'controller' => 'App\Controllers\HomeController',
        'action' => 'index'
    ],
    '/hello' => [
        'controller' => 'App\Controllers\HomeController',
        'action' => 'greetings'
    ],
    '/paciente/index' => [
        'controller' => 'App\Controllers\PacienteController',
        'action' => 'index'
    ]
];
```

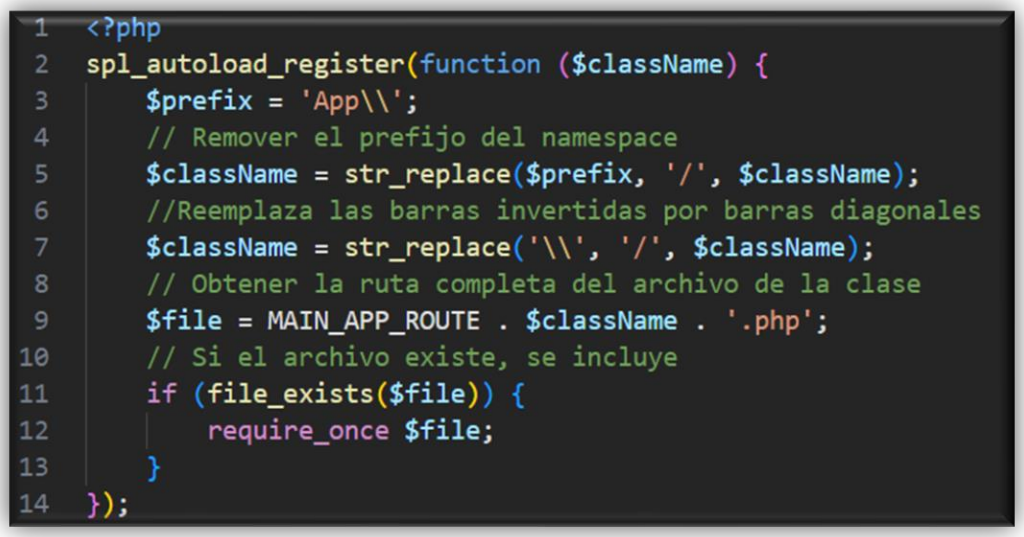
Se encargará de llamar el controlador y ejecutar la acción requerida, siguiendo el patrón MVC, y en caso de no encontrar una ruta y método válido, redirigirá a una página `404`, para indicar que la petición es incorrecta

```php
# Verificar si la ruta es válida
if (array_key_exists($url, $routes)) {
    $route = $routes[$url];
    $controllerName = $route['controller'];
    $action = $route['action'];

    # Verificar si la clase y el método (acción) del controlador existen
    if (class_exists($controllerName) && method_exists($controllerName, $action)) {
        // Instanciar el controlador y llamar a la acción
        $controller = new $controllerName();
        $controller->$action();
        exit;
    }
}

http_response_code(404);
require_once MAIN_APP_ROUTE.'views/404.php'; // Archivo de vista personalizada para el error 404
exit;
```

Archivo autoload.php

En este archivo, el autoloading se basa en el hecho de que las clases están organizadas en la estructura de directorios `app/controllers` y `app/models`. El prefijo App se utiliza como el namespace base para todas las clases de la aplicación.

Cuando se intenta instanciar una clase que aún no ha sido incluida, el autoloading se activará y buscará automáticamente (gracias al método de PHP `spl_autoload_register`) el archivo correspondiente en la ruta `app/{clase}.php`. Si encuentra el archivo, lo incluirá automáticamente, lo que permitirá utilizar la clase sin necesidad de requerirla manualmente.



```php
<?php
spl_autoload_register(function ($className) {
    $prefix = 'App\\';
    // Remover el prefijo del namespace
    $className = str_replace($prefix, '/', $className);
    //Reemplaza las barras invertidas por barras diagonales
    $className = str_replace('\\', '/', $className);
    // Obtener la ruta completa del archivo de la clase
    $file = MAIN_APP_ROUTE . $className . '.php';
    // Si el archivo existe, se incluye
    if (file_exists($file)) {
        require_once $file;
    }
});
```

Modelos y objetos

Si queremos seguir el paradigma de la `Programación Orientada a Objetos` teóricamente deberíamos tener una clase por cada tabla de la base de datos que haga referencia a un objeto de la vida real, en este caso el objeto que crearíamos sería “Paciente” y tendría un nombre, un telefono, un estado, etc, los cuales corresponden a los atributos del objeto. Esta clase hereda de `models/BaseModel` y tiene un método **save** para guardar el paciente en la base de datos, podríamos tener otro método **update** que sería similar, etc. En el IDE o editor de texo seleccionado se pueden generar los getters y setters (solamente si es necesario).

```php
<?php
namespace App\Models;

class PacienteModel extends BaseModel
{
    public function __construct(
        private ?int $documento=null,
        private ?string $nombre=null,
        private ?string $direccion=null,
        private ?string $telefono=null,
        private ?string $fecha_nacimiento=null,
        private ?string $estado=null,
        private ?string $genero=null,
        private ?string $eps=null,
        private ?string $email=null,
        private ?string $fkUsuario=null
    ) {
        $this->table = 'paciente'; # se define la tabla especifica
        parent::__construct(); # Se llama al constructor del padre
    }
    // Resto de métodos del Modelo
}
```

*Si se desea definir y asignar los atributos de la clase directamente dentro del constructor sin tener que declararlos previamente, se puede utilizar la sintaxis "property promotion" disponible a partir de **PHP 8**. Esto permite definir y asignar los valores de los atributos directamente en la firma del constructor.*

No es necesario declarar los atributos previamente dentro de la clase, ya que se definen automáticamente en el contexto del constructor.

Si tuviéramos versiones antiguas de PHP, tendríamos la necesidad de construir los **setters** y **getters** de cada uno de los atributos de la clase, para poder acceder de forma apropiada a cada uno de sus valores; para lo cual podríamos tener la siguiente sintaxis aplicada a PHP 8:

```php
/**
 * Get the value of documento
 */
public function getDocumento():int
{
    return $this->documento;
}
/**
 * Set the value of documento
 * @return  self
 */
public function setDocumento($documento):void
{
    $this->documento = $documento;
}
```

*Sin embargo, la sintaxis de **property promotion** en PHP 8 nos brinda una forma más concisa y legible de definir y utilizar atributos en las clases, eliminando la necesidad de escribir código adicional para **getters** y **setters** cuando solo se necesite instanciar el objeto con los atributos.*

*Los atributos promocionados **(promoted properties**) si bien permiten definir todos los atributos en el constructor, aún podemos implementar métodos getters y setters adicionales si se requieren realizar validaciones u otras operaciones específicas en los atributos*

la property promotion en PHP 8 ofrece una forma más concisa de definir atributos y acceder a ellos, pero es importante usarla con moderación y tener en cuenta los principios de encapsulamiento y diseño de nuestra aplicación.

*El símbolo de interrogación `(?)` antes del tipo de dato en la definición de los atributos indica que ese atributo puede ser nulo. Es parte de la sintaxis de **PHP 8** llamada "**`Nullable Types`**" que permite indicar que un atributo puede tener un valor nulo `(null)` además del tipo de dato especificado.*

Esto brinda flexibilidad al trabajar con los atributos, ya que se les puede asignar valores nulos cuando sea necesario, como cuando no se tiene información disponible o cuando se quiere indicar que un atributo no tiene un valor específico.

Los controladores

Los crearemos en la carpeta `controllers` y se encargarán de llamar a los modelos y vistas correspondientes según sea la solicitud. Para este ejemplo crearemos 2 controladores, `HomeController` y `PacienteController`, ambos heredarán de la clase `BaseControllers`.

Iniciaremos con la clase `HomeController` la cual contiene dos acciones definidas, la función *index* y la función *greetings*. Ambas acciones simplemente muestran información:

```php
<?php
namespace App\Controllers;

class HomeController extends BaseController
{
    public function index()
    {
        echo "<br>CONTROLER: HomeController";
        echo "<br>ACCIÓN:    Index";
    }
    public function greetings()
    {
        echo "<br>CONTROLER: HomeController";
        echo "<br>ACCIÓN:    Greetings";
    }
}
```

Luego crearemos la clase `PacienteController` con dos ejemplos de acciones definidas que permiten traer los pacientes que se encuentren en base de datos a través del *modelo*.

El primer método hace llamado a la vista de manera directa, mientras que el segundo llama al método `render`, que está implementado en la clase de padre.

```php
<?php
namespace App\Controllers;
use App\Models\PacienteModel;

class PacienteController extends BaseController
{
    public function indexTest()
    {
        // Crear una instancia del modelo Paciente
        $pacienteObj = new PacienteModel();
        // Obtener todos los pacientes desde el modelo
        $pacientes = $pacienteObj->getAll();
        // Pasar los datos a la vista
        $data = [
            'patients' => $pacientes
        ];
        # Renderizar la vista directamente desde este controlador
        require_once MAIN_APP_ROUTE. 'views/paciente/index.php';
    }
```

```php
    public function index()
    {
        // Crear una instancia del modelo Paciente
        $pacienteObj = new \App\models\PacienteModel();
        // Obtener todos los pacientes desde el modelo
        $pacientes = $pacienteObj->getAll();
        // Pasar los datos a la vista
        $data = [
            'patients' => $pacientes
        ];
        # Renderizar la vista a través del método de BaseController
        $this->render('paciente/index', $data);
    }
}

```

Las vistas

En este caso crearemos la vista `views/paciente/IndexView.php` para mostrar todos los pacientes que se consultaron a través del modelo por medio del controlador. En este caso solo estamos haciendo una prueba para mostrar los nombres:

```php
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <div class="container">
        <h1>Lista pacientes</h1>
        <p>
            <?php
            foreach($data['patients'] as $item):
                echo "<br>Nombre: $item->nombre";
            endforeach;
            ?>
        </p>
    </div>
</body>
</html>
```

A continuación, crearemos la vista para adicionar un nuevo paciente a través de la vista: `views/paciente/createView.php` para enviar todos los datos al controlador.

```php
<!DOCTYPE html>
<html lang="en">
<head>
    <!-- Required meta-tags -->
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <div class="container">
        <h1>Nuevo paciente</h1>
```

```
        <div class="cont-paciente">
            <form action="/pacientes/new" method="post">
                <div class="form-group">
                    <label for="txtDocumento">Documento</label>
                    <input type="number" name="txtDocumento" id="txtDocumento" class="form-control">
                </div>
                <div class="form-group">
                    <label for="txtNombre">Nombre</label>
                    <input type="text" name="txtNombre" id="txtNombre" class="form-control">
                </div>
                <div class="form-group">
                    <label for="txtDireccion">Dirección</label>
                    <input type="text" name="txtDireccion" id="txtDireccion" class="form-control">
                </div>
                <div class="form-group">
                    <label for="txtTelefono">Telefono</label>
                    <input type="text" name="txtTelefono" id="txtTelefono" class="form-control">
                </div>
                <div class="form-group">
                    <label for="txtEmail">Email</label>
                    <input type="text" name="txtEmail" id="txtEmail" class="form-control">
                </div>
                <div class="form-group">
                    <label for="txtFechaNac">Fecha Nacimiento</label>
                    <input type="date" name="txtFechaNac" id="txtFechaNac" class="form-control">
                </div>
                <div class="form-group">
                    <label for="txtEstado">Estado</label>
                    <input type="text" name="txtEstado" id="txtEstado" class="form-control">
                </div>
                <div class="form-group">
                    <label for="txtGenero">Género</label>
                    <input type="text" name="txtGenero" id="txtGenero" class="form-control">
                </div>
```

```
                <div class="form-group">
                    <label for="txtEps">EPS</label>
                    <input type="text" name="txtEps" id="txtEps" class="form-control">
                </div>
                <div class="form-group">
                    <label for="txtUsuario">Código Usuario</label>
                    <input type="text" name="txtUsuario" id="txtUsuario" class="form-control">
                </div>
                <div class="form-group">
                    <button type="submit">Guardar</button>
                </div>
            </form>
        </div>
    </div>
</body>
</html>
```

Modificando el modelo y controlador para insertar un paciente

Después de generar la vista para crear el paciente, modificamos el controlador para incluir las siguientes acciones:

```php
public function create()
{
    $this->render('paciente/create');
}

public function new()
{
    if(isset($_POST['txtDocumento'])){
        $documento = $_POST['txtDocumento'] ?? '';
        $nombre = $_POST['txtNombre'] ?? '';
        $direccion = $_POST['txtDireccion'] ?? '';
        $telefono = $_POST['txtTelefono'] ?? '';
        $email = $_POST['txtEmail'] ?? '';
        $fechaNac = $_POST['txtFechaNac'] ?? '';
        $estado = $_POST['txtEstado'] ?? '';
        $genero = $_POST['txtGenero'] ?? '';
        $eps = $_POST['txtEps'] ?? '';
        $fkUsuario = $_POST['txtUsuario'] ?? '';
        // Se crea objeto paciente con los datos recibidos de la vista
        $pacienteBD = new PacienteModel($documento,$nombre,$direccion,$telefono,$fechaNac,
        $estado,$genero,$eps,$email,$fkUsuario);
        // LLamar al metodo save del modelo para guardar en BD
        $pacienteBD->save();
    }
    header("Location:/pacientes/index");
}
```

También modificamos el modelo para incluir un nuevo método que se encargará de guardar el paciente en la base de datos. Todos los datos del paciente ya fueron previamente recibidos en el constructor, por lo que simplemente será llamarlos y ejecutar la consulta requerida:

```php
    public function save()
    {
        try {
            // Preparar la consulta para insertar un paciente en la BD
            $sql = $this->dbConnection->prepare("INSERT INTO paciente
(documento,nombre,direccion,telefono,email,fecha_nacimiento,estado,genero,eps,fk_cod_usuario)
VALUES (?,?,?,?,?,?,?,?,?,?)");
            $sql->bindParam(1, $this->documento);
            $sql->bindParam(2, $this->nombre);
            $sql->bindParam(3, $this->direccion);
            $sql->bindParam(4, $this->telefono);
            $sql->bindParam(5, $this->email);
            $sql->bindParam(6, $this->fecha_nacimiento);
            $sql->bindParam(7, $this->estado);
            $sql->bindParam(8, $this->genero);
            $sql->bindParam(9, $this->eps);
            $sql->bindParam(10, $this->fkUsuario);
            # Se ejecuta la consulta
            $sql->execute();
        } catch (\PDOException $ex) {
            die('Error guardando paciente '.$ex->getMessage());
        }
    }
}
```

Modificamos el index para organizar las rutas

Las rutas las enviamos para un nuevo archivo llamado `config/routes.php` el cual va contener solamente el array con las rutas, controladores y acciones.

```php
<?php
#error_reporting(0);
require_once '../vendor/autoload.php'; // Cargar clases de forma automática
require_once '../config/global.php'; // Configuración general

$url = $_SERVER['REQUEST_URI']; // Saber que llega en la URL

# Rutas válidas en nuestra aplicación
$routes = require_once '../config/routes.php';

```

`config/routes.php`

```php
<?php
return [
    '/' => [
        'controller' => 'App\Controllers\HomeController',
        'action' => 'index'
    ],
    '/home' => [
        'controller' => 'App\Controllers\HomeController',
        'action' => 'index'
    ],
    '/hello' => [
        'controller' => 'App\Controllers\HomeController',
        'action' => 'greetings'
    ],
    '/pacientes/index' => [
        'controller' => 'App\Controllers\PacienteController',
        'action' => 'index'
    ],
    '/pacientes/create' => [
        'controller' => 'App\Controllers\PacienteController',
        'action' => 'create'
    ],
    '/pacientes/new' => [
        'controller' => 'App\Controllers\PacienteController',
        'action' => 'new'
    ]
];
```

Ejecución de la aplicación

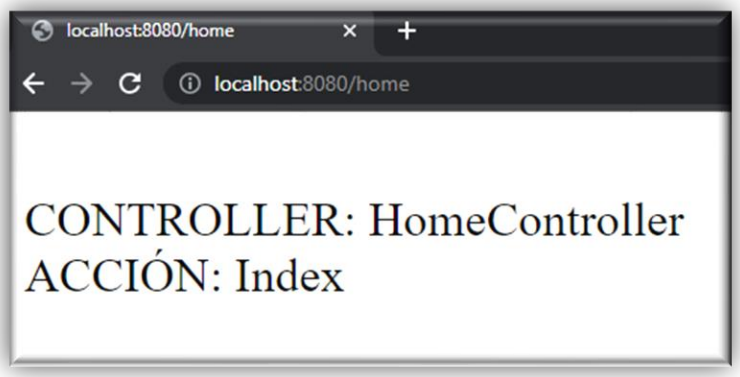
Para ejecutar nuestra aplicación, debemos ingresar a la terminal, y desde allí ir a la carpeta `/public`.

```
jupin@jusapi MINGW64 /c/xampp/htdocs/SenamasEPS
$ cd public/
```

Estando allí ingresaremos el siguiente comando para ejecutar el servidor y correr el intérprete de PHP:

Ejecución de la aplicación

Ahora vamos al navegador e ingresamos a la siguiente URL donde podremos ver que se ejecuta el controlador `HomeController` y la acción `Index`



Actividad 2

Con el fin de interiorizar y comprender mejor la estructura del proyecto a realizar, los aprendices deberán crear un ejemplo completo de la tabla **usuario** y de la tabla **cita** que permita crear un nuevo registro de ambas tablas y listar los registros de ambas tablas

Se deben crear las rutas, controladores, vistas y modelos necesarios para cumplir con este requerimiento

- **Actividades de transferencia del conocimiento**
  Los aprendices deberán demostrar los conocimientos conceptuales y procedimentales que adquirieron a partir de las actividades de apropiación mediante el siguiente instrumento de evaluación:

Los aprendices deberán realizar cada una de las actividades propuestas para adquirir dominio en los conocimientos relacionados con los diferentes temas. Se evaluarán los conocimientos adquiridos mediante instrumentos de evaluación de Desempeño y Producto relacionados en la presente guía

- **Ambiente Requerido**
  - Ambiente de SISTEMAS con conexión eléctrica e internet
- **Materiales**
  - Computadores (30)
  - Sillas (3)
  - Televisor (1)
  - Resma tamaño carta (1)
  - Marcadores (3)
  - Lápiz (1)
  - Lapicero (1)

4. ACTIVIDADES DE EVALUACIÓN

Evidencias de Aprendizaje	Criterios de Evaluación	Técnicas e Instrumentos de Evaluación
Evidencias de Conocimiento : **Evidencias de Desempeño** **Asistencia y participación activa en las diferentes actividades propuestasz** **Evidencias de Producto:**	Crea la base de datos en el motor de base de datos seleccionado, siguiendo especificaciones técnicas del informe, según normas y protocolos de la empresa.	**Observación:** EXC_D_01 **Valoración del Producto:** EXC_P_01

5. GLOSARIO DE TÉRMINOS

Sistema de información: es un conjunto de elementos orientados al tratamiento y administración de datos e información, organizados y listos para su uso posterior, generados para cubrir una necesidad o un objetivo

Sistema operativo – Es un conjunto de programas que sirven para manejar un ordenador.

Software - El conjunto de programas, procedimientos y documentación asociado a un sistema informático.

Javascript: es un lenguaje de programación del lado del cliente que se utiliza con frecuencia en diseño WEB para generar efectos más complejos que no se puedan alcanzar usando HTML.

HTML: Siglas de las palabras inglesas: Hypertext Markup Language. Es decir, lenguaje de marcado de hipertexto. Lenguaje informático para crear páginas web. Conjunto de etiquetas o instrucciones que permiten estructurar el contenido de una web e incluir los hipervínculos o enlaces enlaces a otras páginas. Este lenguaje lo inventó en 1991 el Doctor Berners-Lee del CERN en Suiza.

HTTPS: Siglas de las palabras inglesas: HyperText Transfer Protocol Secure o versión segura del protocolo HTTP. Es el protocolo empleado para la transferencia de ficheros HTML cifrados que puedan contener información confidencial.

HTTP: siglas de las palabras inglesas: Hypertext Transfer Protocol. A saber en español: Protocolo de Transmisión de Hipertexto. Protocolo estándar de transferencia de hipertexto. Es decir: el protocolo de comunicaciones en el que está basado la Word Wide Web.

Script: es un archivo de órdenes o archivo de procesamiento por lotes. Es un programa usualmente simple, que por lo regular se almacena en un archivo de texto plano.

MySQL: es un sistema de gestión de bases de datos de código abierto que, junto con PHP, permite darle a las páginas web cierto dinamismo, es decir, disponer de manera adecuada los datos solicitados por los navegadores. Es un sistema multiplataforma y su uso está tan extendido en las bases de datos que podría considerarse un estandar.

SEO (Search Engine Optimisation) Optimización en buscadores: técnica utilizada para asegurar que una página Web es compatible con los motores de búsqueda y así tener la posibilidad de aparecer en las posiciones más altas en los resultados de búsqueda.

Diseño web adaptable (responsive web design): se llama así al diseño web de aquellas páginas que se adaptan al tamaño de la pantalla o ventana en que se despliegan, por medio del uso de, idealmente, un solo documento HTML y un solo documento CSS. Esto permite hacer una sola página web para smartphones, phablets, tablets y PC.

Diagrama o Modelo Entidad Relación (DER): denominado por sus siglas en inglés, E-R "Entity relationship", o del español DER "Diagrama de Entidad Relación") es una herramienta para el modelado de datos que permite representar las entidades relevantes de un sistema de información así como sus interrelaciones y propiedades

Bases de Datos (BD): es un banco de información que contienen datos relativos a diversas temáticas y categorizados de distinta manera, pero que comparten entre sí algún tipo de vínculo o relación que busca ordenarlos y clasificarlos en conjunto

6. REFERENTES BILBIOGRÁFICOS

- Documentos técnicos relacionados en la plataforma

https://www.youtube.com/watch?v=6ERdu4k62wI https://www.freecodecamp.org/news/create-an-mvc-framework-from-scratch-with-php/	Use PHP to Create an MVC Framework - Full Course
https://codeburst.io/mvc-design-pattern-analogy-to-an-old-school-landline-3dcd2e994063	MVC Design pattern: analogy to an old school landline
https://www.freecodecamp.org/news/the-model-view-controller-pattern-mvc-architecture-and-frameworks-explained/	The Model View Controller Pattern – MVC Architecture and Frameworks Explained

7. CONTROL DEL DOCUMENTO

	Nombre	**Cargo**	**Dependencia**	**Fecha**
Autor (es)	**Julian Salazar Pineda**	**Instructor**	**CPIC**	**14-11-2022**

8. CONTROL DE CAMBIOS (diligenciar únicamente si realiza ajustes a la guía)

	Nombre	**Cargo**	**Dependencia**	**Fecha**	**Razón del Cambio**
Autor (es)					