**PHP**

Se abre y se cierra:

<?php

¿>

Se muestra:

Echo”..”; o echo’…’;

Dependiendo que comillas se usen, el resto tienen que ser de las otras.

Tambien

Print “…“;

**Variables**

Se definen con un $ antes del nombre:

$nombre = “Cristian”;

Echo $nombre;

**Variables tipo String y concatenar**

Definimos 2 variables

$apellidos = “Bolaños”;

$mensaje = “String”;

Print “$apellidos: $mensaje”;

**Muestra:** Bolaños: String

**Sumar dos valores**

    $num1 = 10;

    $num2 = 20;

    $total = $num1 + $num2;

    echo $total;

**Funciones**

Se define la función y se llama, también al concatenar se usa el .

function sumar(){

        $var1=30;

        $var2=50;

        $totalsuma=$var1+$var2;

        echo "La suma es" . $totalsuma;

    }

    sumar();

**Variables locales SCOPE(Alcance)**

La función solo toma como referencia la variable local

    $contacto = "Cristiann";

    function mostrarContacto(){

        $contacto = "Alejandro";

        echo $contacto;

    }

    echo "<br />";

    mostrarContacto();

    echo "<br />";

    echo $contacto;

**Variables globales**

    global $mensaje;

    $mensaje = "holaa";

    function mostrarContacto2(){

        global $mensaje;

        echo $mensaje;

    }

    echo "<br />";

    mostrarContacto2();

    echo "<br />";

    echo $mensaje;

**Variables estáticas**

// Variables estaticas

    function contador(){

        static $num=1;

        echo $num;

        $num= $num+1;

    }

    echo "<br />";

    echo contador();

    echo "<br />";

    echo contador();

    echo "<br />";

**Constantes**

1. No se necesita usar $
2. Solo se definen con el método define
3. Se asignan una única vez
4. El alcance o SCOPE es global, se puede acceder desde cualquier lugar

**Definir una constante y su valor**

    define('CUOTA',2000);

    $valorcuota=CUOTA;

    echo "El valor de la cuota: $valorcuota";

    echo "<br />";

    echo "<br />";

    echo "El valor de la cuota: " . CUOTA . "<br>";

**Expresiones Operadores**

// El = es un operador de asignación, los operadores se usan para realizar operaciones sobre las variables

    $apellido = "bueno";

    $suma = 30 + 2;

// Diferentes tipos de operadores

// Aritmeticos

    $sueldo = 3000;

    $subsidioTransporte = 2000;

    $salud = 3000;

    $pension = 3000;

    $ingresos = $sueldo+$subsidioTransporte;

    $egresos = $salud+$pension;

    $total = $ingresos-$egresos;

    echo "<br />";

    // Area de un rectángulo

    $base = 10;

    $altura = 2;

    $area = $base\*$altura;

    echo $area;

    echo "<br />";

    // División

    $num1 = 4;

    $num2 = 2;

    $division = $num1 / $num2;

    echo $division;

    echo "<br />";

    // Porcentaje

    $sumanotas = 42;

    $cantidadMaterias = 5;

    $promedio = $sumanotas / $cantidadMaterias;

    echo $promedio;

    echo "<br />";

    // Módulo

    $a = 51%2;

    echo $a;

    // Exponenciación

    $c = 4;

    $d = 3;

    $resultadoExponenciacion = $c \*\* $d;

    echo $resultadoExponenciacion;

    echo "<br />";

**Operadores condicionales**

// Operador Ternario

    $puedeingresar = (true) ? "ENTRE" : "NO ENTRA";

    echo $puedeingresar;

    echo "<br />";

// Operadores lógicos and or %% ||

    $permiso = false;

    $autenticado = true;

    echo ($permiso and $autenticado) ? "Bienvenido administrador" : "Bienvenido invitado";

    echo "<br />";

    $total = 50;

    $pasarmateria = 50;

    $validarpasar = ($total > $pasarmateria) ? true : false;

    echo ($validarpasar) ? "PASA" : "NO PASA";

// Operador +=

    $contador = 1;

    // $contador = $contador+1;

    $contador += $contador;

    echo $contador;

// Operador -=

    $contador = 1;

    // $contador = $contador-1;

    $contador -= $contador;

    echo $contador;

// Operador \*=

    $contador = 1;

    // $contador = $contador\*1;

    $contador \*= $contador;

    echo $contador;

    echo "<br />";

// Operador /=

    $contador = 30;

    // $contador = $contador/2;

    $contador /= 2;

    echo $contador;

    echo "<br />";

// Operador %=

    $contador = 51;

    // $contador = $contador%2;

    $contador %= 2;

    echo $contador;

    echo "<br />";

// Operador .=

// Concatena todos los String

    $nombreCompleto = "Cristian";

    $nombreCompleto .= "Bolaños";

    $nombreCompleto .= "Ausecha";

    echo $nombreCompleto

**Operadores de incremento y decremento**

// Operadores de incremento

    $conta = 20;

    $conta +=1;

    echo $conta;

    echo "<br />";

// Post Incremento

    echo $conta++;

    echo "<br />";

    echo $conta;

    echo "<br />";

// Pre Incremento

// Imprime con el valor ya incrementado

    echo ++$conta;

    echo "<br />";

    echo $conta;

    echo "<br />";

// Operadores de decremento

    $conta = 20;

    $conta -=1;

    echo $conta;

    echo "<br />";

// Post Decremento

    echo $conta--;

    echo "<br />";

    echo $conta;

    echo "<br />";

// Pre Decremento

// Imprime con el valor ya incrementado

    echo --$conta;

    echo "<br />";

    echo $conta;

    echo "<br />";

**Tipos de datos**

**ENTEROS**

    echo "<br />";

    echo "---------- ENTEROS ----------";

    echo "<br />";

    $puntaje = 120;

    echo $puntaje;

    echo "<br />";

// Hexadecimal

    $numHexa = 0x6FFF6;

    echo $numHexa;

    echo "<br />";

**Convertir String a número**

// Convertir String a entero

    $puntaje = 20;

    $total = "120" + $puntaje;

    echo $total;

**Números negativos**

// Números negativos

    $puntajetotal = 8-20;

    echo $puntajetotal;

    echo "<br />";

**Obtener tamaño de un entero**

// Obtener tamaño de un entero (Determinado por el sistema)

    echo PHP\_INT\_SIZE;

    echo "<br />";

// Obtener el minimo numero de un entero (Determinado por el sistema)

    echo PHP\_INT\_MIN;

    echo "<br />";

// Obtener el máximo numero de un entero (Determinado por el sistema)

    echo PHP\_INT\_MAX;

    echo "<br />";

**DECIMALES**

// Definir decimales

    $pagototal = 102.45;

    echo $pagototal;

    echo "<br />";

// Negativos

    $pagopendiente = -56.23;

    echo $pagopendiente;

    echo "<br />";

**Redondear decimales**

// Redondear decimales

    $puntaje = 48.51;

    echo round($puntaje);

    echo "<br />";

**Comparar decimales con precisión**

// Comparar decimales con precisión 0.1

    $precio = 1.87;

    $estimado = 1.98;

    // Abs es valor absoluto

    echo (abs($precio-$estimado) < 0.1) ? "PAGA" : "NO PAGA";

**Convertir unidades de medida**

<?php

    /\*

        Se definen los datos que necesitamos:

            $valor: es el número de la medida que usaremos.

            $desde: es la unidad de medida desde la cual convertiremos.

            $hasta: es la unidad de medida a la cual necesitamos convertir

        ------------

        The data we need is defined:

            $valor: Is the number of the measurement we will use.

            $desde: Is the unit of measurement from which we will convert.

            $hasta: Is the unit of measurement to which we need to convert.

    \*/

    $valor = '';

    $desde = '';

    $hasta = '';

// Convertimos a la medida estandard, sería metros

// -----------

// We convert to the standard measurement, it would be meters.

/\* Esto para que todas las medidas la cual introduzca sean metros,

   por tanto los metros despues se convierten a lo que se necesite

--------------

   This is so that the measurements wich you enter are meters,

   therefore the meters are later converted to whatever is needed.

\*/

    function convertirAMetros($valor, $unidad\_desde){

        switch($unidad\_desde){

            case 'Milimetro':

                return $valor / 1000;

            break;

            case 'Centimetro':

                return $valor / 100;

            break;

            case 'Decimetro':

                return $valor / 10;

            break;

            case 'Metro':

                return $valor \* 1;

            break;

            case 'Decametro':

                return $valor \* 10;

            break;

            case 'Hectometro':

                return $valor \* 100;

            break;

            case 'Kilometro':

                return $valor \* 1000;

            break;

            default:

                return 'Unidad no disponible';

            break;

        }

    }

/\*

    Convertimos desde metros, es decir, despues de convertir la unidad que se introdujo a metros,

    la convertimos a lo que se solicite.

---------------

    We convert from meters, that is, after converting the unit that was entered to meters,

    we convert it to whatever is needed.

\*/

    function convertirDesdeMetros($valor, $unidad\_hasta){

        switch($unidad\_hasta){

            case 'Milimetro':

                return $valor \* 1000;

            break;

            case 'Centimetro':

                return $valor \* 100;

            break;

            case 'Decimetro':

                return $valor \* 10;

            break;

            case 'Metro':

                return $valor \* 1;

            break;

            case 'Decametro':

                return $valor / 10;

            break;

            case 'Hectometro':

                return $valor / 100;

            break;

            case 'Kilometro':

                return $valor / 1000;

            break;

        }

    }

/\*

    Traemos la solicitud POST del HTML, a través del botón "convertir" (el cual tiene un tipo name

    que lo identifica).

    Verificamos si existe por medio del método isset.

    Traemos los valores $valor, $desde, $hasta, los cules nos traen el valor, la unidad de medida actual

    y la unidad de medida a la cual hay que convertir respectivamente.

    Llamamos a la función convertirAMetros, la cual convierte el valor (contenido en $valor)

    desde la medida actual (contenido en $desde) a metros.

    Llamamos a la función convertirDesdeMetros, la cual convierte el valor ya establecido en metros

    (contenido en $calculoDesde) a la unidad de medida a la cual hay que convertir (contenida en $hasta).

    Guardamos el valor final en la variable $resultado, llamando al método number\_format para formatear este

    número y que muestre solo dos decimales.

    ------------------

    We bring the HTML POST request, through the "convertir" button (which has a name type that identifies it).

    We check if it exists using the isset method.

    We bring the values $valor, $desde, $hasta, which bring us the value, the current unit of measurement

    and the unit of measurement to which it must be converted respectively.

    we call the function convertirAMetros, which converts the value (contained in $valor) from the

    current measurement (content in $desde) to meters.

    We call the function convertirDesdeMetros, which converts the value already established in meters

    (contained in $calculoDesde) to the unit of measurement to which it must be converted

    (contained in $hasta).

    We save the final value in the variable $resultado, calling the number\_format method for format this

    number and display only two decimals places.

\*/

    if(isset($\_POST['convertir'])){

        // Obtener los valores

        $valor = $\_POST['valor'];

        $desde = $\_POST['desde'];

        $hasta = $\_POST['hasta'];

        $calculoDesde = convertirAMetros($valor, $desde);

        $calcularHasta = convertirDesdeMetros($calculoDesde, $hasta);

        // Mostrar dos decimales

        $resultado = number\_format($calcularHasta, 2);

    }

?>

**API’s**

Una API es el mecanismo más útil para conectar dos softwares entre sí para el intercambio de mensajes o datos en un formato estándar (Formato **JSON**). Así es como se convierte en un instrumento para buscar ingresos, abrirse al talento, innovar y automatizar procesos.

**Http Methods**

* **GET:** Receive information about an API resource.
* **POST:** Create an API resource.
* **PUT:** Update an API resource.
* **DELETE:** Delete an API resource.

**Códigos de estado**

**Respuestas satisfactorias:**

* **GET:** Se obtiene el recurso y se transmite el cuerpo del mensaje
* **HEAD:** Los encabezados de entidad están en el cuerpo del mensaje
* **PUT o POST:** El recurso que describe el resultado de la acción se transmite en el cuerpo del mensaje.
* **TRACE:** El cuerpo del mensaje contiene el mensaje de solicitud recibido por el servidor.

**200 OK:** La solicitud ha tenido éxito. El significado de éxito varía dependiendo del método HTTP.

**404 Not Found:** El servidor no pudo encontrar el contenido solicitado o que no existe ningún tipo de recurso en esa API. Este código de respuesta es uno de los más famosos dada su alta ocurrencia en la web.

**500 Internal Servidor Error:** El servidor ha encontrado una situación que no sabe cómo manejarla.

**Tipo de autorizaciones**

**0Auth:** Es un estándar abierto que permite el flujo de autorización para sitios web o aplicaciones informáticas (Es el menos seguro).

**OAuth 2.0:** Es un protocolo de autorización que otorga a un cliente API acceso limitado a los datos del usuario en un servicio web. Las API de GitHub, Google y Facebook lo utilizan notablemente. Este te brinda un Id y una key secreta, por lo cual se genera un token de autenticación para acceder a dicha API.

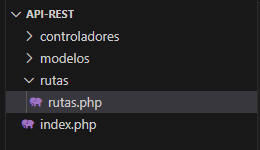
**Api Key:** Son las siglas de API: Application Programming Interfaces, que se traduciría como interfaz de programación de aplicaciones, y Key: Llave, por lo tanto, podríamos decir que es un identificador que sirve para la autenticación de un usuario para el uso de un servicio. Este tipo de autenticación se conecta mediante la URL, con parámetros GET, pasando la API key.

Tambien hay tipos de autenticación

* el tipo de autenticación por cabecera, es decir, mandar la API key, autenticación.
* Por URL, mandar la key para que brinde el acceso a la información.

**API REST**

**Estructura:** Se usa el patrón MVC.



La carpeta rutas sirve como las vistas, pero está api no va a mostrar nada, solo va a gestionar.

Rutas.php:

<?php

    echo "soy la ruta";

?>

Rutas.controlador.php: Conectamos las rutas al controlador (mediante POO), con una clase que incluya al archivo rutas.php.

<?php

    class ControladorRutas{

        public function inicio(){

            include "rutas/rutas.php";

        }

    }

?>

Index.php: Conectamos el controlador con el index mediante el método require\_once, en el cual quedan conectadas las rutas al index a través del controlador.

<?php

    require\_once "controladores/rutas.controlador.php";

    $rutas = new ControladorRutas();

    $rutas->inicio();

?>

**Htaccess:** Permite que todo recaiga sobre el index.php, por tanto, si se cae una ruta, o no se encuentra, este redirecciona al index.php.

Options All -Indexes

Options -MultiViews

RewriteEngine On

RewriteCond %{REQUEST\_FILENAME} !-f

RewriteRule ^ index.php [QSA,L]

**Rutas**

Aquí está el manejo de rutas por medio de URL, donde se manda según el usuario necesite, de igual manera, se maneja las peticiones GET, POST, PUT y DELETE del endpoint cursos, que está conectado cada uno a su propia función en su controlador como se verá más adelante.

<?php

    /\*

        Esto $\_SERVER['REQUEST\_URI'], nos sirve para obtener la ruta en la cual se encuentra el usuario

        La direccion URL donde se encuentra el usuario la guardamos dentro de $arrayRutas, donde

        se guarda como un array con indice que se separa por cada /. Es decir:

            [0] =>

            [1] => PROYECTOSWEB

            [2] => api-rest

            [3] => cursos

        This $\_SERVER['REQUEST\_URI'], helps us to obtain the route in which the user is located.

        The URL where the user is located is stored in $arrayRutas, where

        it is save as an array with an index separated by a /. That is:

            [0] =>

            [1] => PROYECTOSWEB

            [2] => api-rest

            [3] => cursos

    \*/

    $arrayRutas = explode("/",$\_SERVER['REQUEST\_URI']);

   // echo "<pre>"; print\_r($arrayRutas); echo "<pre>";

    /\*

        Dentro del if, vamos a verificar que el usuario si este especificando que necesita, si el usuario

        solo coloca PROYECTOSWEB/api-rest/, salta el array detalle no encontrado, pero si especifica, como

        por ejemplo, PROYECTOSWEB/api-rest/cursos, no se mostrará este array.

        El array\_filter se usa para eliminar todos los indices vacios, en este caso el indice 0 como vimos

        anteriormente.

        Dentro del else if verifica primero si esta en el indice 3, seguido a esto se verifica (mediante otro if)

        que busca el usuario, en este caso, cursos o registro.

        Inside of if, we will verify that the user is specifying what he needs, if the user only puts

        PROYECTOSWEB/api-rest/, the "detail no found" array is skipped, but if he specifies, of example,

        PROYECTOSWEB/api-rest/cursos, this array will not be displayed.

        The array\_filter is used to remove all empty indexes, in this case index 0 as we saw previously.

        Inside of else if first check if it is at the index 3, then it verifies(using another if)

        what the user looking for, in this case, courses or register.

    \*/

    // Aqui no se hacen peticiones

    // No request are made here

    if(count(array\_filter($arrayRutas))==2){

        $json=array(

            "detalle"=>"no encontrado"

        );

        echo json\_encode($json,true);

        return;

    // Aqui se hacen peticiones pero dentro se verifica que peticion es

    // Request are made here but inside it is verify what request it is

    }else if(count(array\_filter($arrayRutas))==3){

        // Aqui se hacen peticiones desde cursos

        // Request are made here from courses

        if(array\_filter($arrayRutas)[3]=="cursos"){

            if(isset($\_SERVER['REQUEST\_METHOD']) && $\_SERVER['REQUEST\_METHOD'] == "POST"){

                $cursos = new ControladorCursos;

                $cursos->create();

            }else if(isset($\_SERVER['REQUEST\_METHOD']) && $\_SERVER['REQUEST\_METHOD'] == "GET"){

                $cursos = new ControladorCursos;

                $cursos->index();

            }

        // Aqui se hacen peticiones desde registro

        // Request are madre here from register

        }else if(array\_filter($arrayRutas)[3]=="registro"){

            if(isset($\_SERVER['REQUEST\_METHOD']) && $\_SERVER['REQUEST\_METHOD'] == "GET"){

                $clientes = new ControladorClientes;

                $clientes->create();

            }

        }

    }else if(array\_filter($arrayRutas)[3]=="cursos" && is\_numeric(array\_filter($arrayRutas)[4])){

        // Petición GET

        if(isset($\_SERVER['REQUEST\_METHOD']) && $\_SERVER['REQUEST\_METHOD'] == "GET"){

            $curso = new ControladorCursos;

            $curso->show(array\_filter($arrayRutas)[4]);

        }

        // Petición PUT

        if(isset($\_SERVER['REQUEST\_METHOD']) && $\_SERVER['REQUEST\_METHOD'] == "PUT"){

            $editarCurso = new ControladorCursos;

            $editarCurso->update(array\_filter($arrayRutas)[4]);

        }

        // Petición DELETE

        if(isset($\_SERVER['REQUEST\_METHOD']) && $\_SERVER['REQUEST\_METHOD'] == "DELETE"){

            $borrarCurso = new ControladorCursos;

            $borrarCurso->delete(array\_filter($arrayRutas)[4]);

        }

    }

?>

**CONTROLADORES**

Se manejan 3 controladores para los clientes, cursos y rutas.

**Rutas:**

<?php

    class ControladorRutas{

        public function inicio(){

            include "rutas/rutas.php";

        }

    }

?>

**Clientes:**

<?php

    class ControladorClientes{

        public function create(){

            $json=array(

                "detalle"=>"estas en la vista registro"

            );

            echo json\_encode($json,true);

            return;

        }

    }

?>

**Cursos:** Aquí se manejan diferentes funciones para los métodos que necesita la API, por tanto, están separados cada una de las peticiones GET, POST, PUT y CREATE. Hasta el momento solo se crearon, no tienen ninguna función más que mostrar un mensaje que verifica en que petición se encuentra.

<?php

    class ControladorCursos{

        public function index(){

            $json=array(

                "detalle"=>"estas en la vista cursos"

            );

            echo json\_encode($json,true);

            return;

        }

        public function create(){

            $json=array(

                "detalle"=>"estas en la vista create"

            );

            echo json\_encode($json,true);

            return;

        }

        public function show($id){

            $json=array(

                "detalle"=>"este es el curos con el id ".$id

            );

            echo json\_encode($json,true);

            return;

        }

        public function update($id){

            $json=array(

                "detalle"=>"curso actualizado con el id ".$id

            );

            echo json\_encode($json,true);

            return;

        }

        public function delete($id){

            $json=array(

                "detalle"=>"curso borrado con el id ".$id

            );

            echo json\_encode($json,true);

            return;

        }

    }

?>

**Modelo – Conexión.php**

Aquí se define una clase la cual contiene los datos necesarios para la conexión a la base de datos.

<?php

    /\*

        Conexión a la base de datos, se hace por medio de un metodo de PHP llamado PDO, donde pide el host

        (en este caso localhost), el nombre de la base de datos (en este caso api-rest) y el usuario y la

        contraseña (en este caso root y no tiene contraseña).

        Luego se le manda el método exec, definiendo el formato utf-8.

        Todo esto dentro de una clase llamada Conexion la cual tiene como funcion conectar que se hara uso más

        adelante, retornando todos los datos mencionados anteriormente.

        Connection to the database, is done through PHP method called PDO, where it asks for the host (in this

        case localhost), the name of the database (in this case api-rest) and the user and password (in this

        case root and it has no password).

        Then the exec method is sent, defining the utf-8 format.

        All this within a class called Conexion which has the function of connect that will be used

        later, returning all the data mentioned above.

    \*/

    class Conexion{

        static public function conectar(){

            $link = new PDO("mysql:host=localhost;dbname=api-rest;","root","");

            $link->exec("set names utf8");

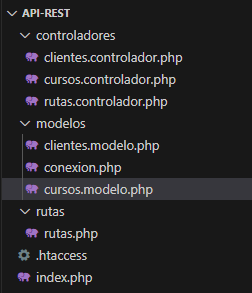
            return $link;

        }

    }

?>

Por el momento los archivos están asi:



**Cursos.modelo.php:**

Dentro de este creamos la conexión entre el controlador y el requisito SQL

<?php

    require\_once "conexion.php";

    class ModeloCursos{

        static public function index($tabla){

            /\*

                $stmt llama a la clase Conexion y con los : llama su funcion conectar junto al metodo PHP

                prepare donde se le pasa la linea de sql y llamar a la tabla que necesitamos

                $stmt calls the Conexion class and with the : calls its conectar function together with the PHP

                prepare method where the sql line is passed and calls the table we need.

            \*/

            $stmt = Conexion::conectar()->prepare("SELECT \* FROM $tabla");

            // Para ejecutar stmt

            // To run stmt

            $stmt->execute();

            /\*

                Retornamos stmt junto a la funcion fetchAll para devolver todos los datos de la base de datos.

                Dentro del fetchAll llamamos un parametro del PDO para que devuelva la informacion sin

                duplicar, solo las propiedades de la tabla (FETCH\_CLASS).

                We return stmt along with the fetchAll function to return all the data of the database.

                Inside the fetchAll we call a parameter of the PDO to return the information without duplicating

                it, only the properties of the table (FETCH\_CLASS).

            \*/

            return $stmt->fetchAll(PDO::FETCH\_CLASS);

            // Cerramos la conexión

            // We closed the conection

            $stmt->close();

            $stmt=null;

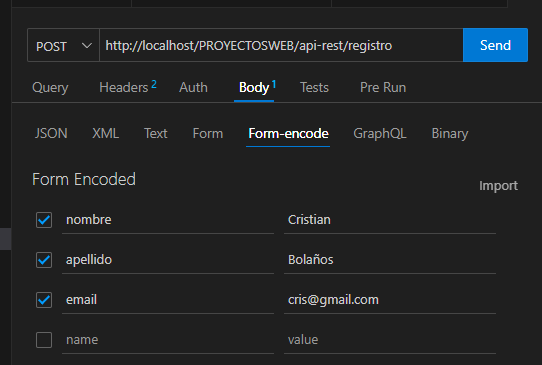
        }

    }

?>

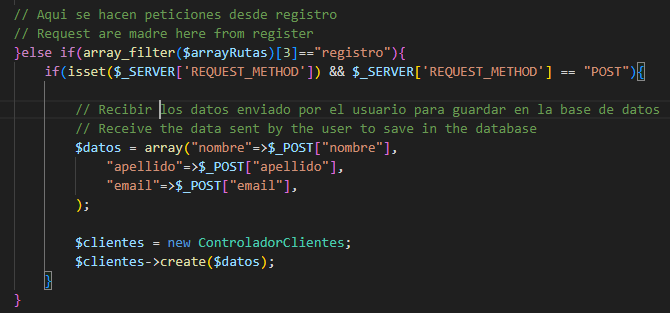
**Recibir datos para el registro**

Se envían desde el form-encode, donde se capturaran más adelante



**Rutas.php:**

Guardamos los datos en un array, que se reciben por medio de la petición POST y se los enviamos al método créate del ControladorClientes



**Validacion de datos de registro:**

Para la validación usaremos el método nativo de PHP llamado:

* preg\_match: este realiza una comparación con una expresión regular, permite que el correo se filtre (por ejemplo, con un solo @), un nombre solo con letras. Este permite el acceso a letras, números o el formato de un email.

**Clientes.controlador.php:**

Dentro de la función validamos por medio del método preg\_match si el nombre, apellido y email tienen un formato correcto.

<?php

    class ControladorClientes{

        public function create($datos){

            //echo "<pre>"; print\_r($datos); echo "<pre>";

            // Validación de datos

            // Data validation

            /\*

                Validar nombre

                Preguntamos si existe el campo nombre y usamos preg\_match con el patrón de permitir solo letras

                Name validation

                We asked if the name field exists and we use preg\_match with the pattern to allow only letters

            \*/

            if(isset($datos["nombre"]) && !preg\_match('/^[a-zA-ZáéíóúÁÉÍÓÚñÑ ]+$/', $datos["nombre"])){

                $json=array(

                    "status"=>404,

                    "detalle"=>"Error en el campo de nombre, solo ingrese letras"

                );

                echo json\_encode($json,true);

                return;

            }

            // Validar apellido

            // Last name validation

            if(isset($datos["apellido"]) && !preg\_match('/^[a-zA-ZáéíóúÁÉÍÓÚñÑ ]+$/', $datos["apellido"])){

                $json=array(

                    "status"=>404,

                    "detalle"=>"Error en el campo de apellido, solo ingrese letras"

                );

                echo json\_encode($json,true);

                return;

            }

            // Validar email

            // Email validation

            if(isset($datos["email"]) && !preg\_match('/^[^0-9][a-zA-Z0-9\_]+([.][a-zA-Z0-9\_]+)\*[@][a-zA-Z0-9\_]+([.][a-zA-Z0-9\_]+)\*[.][a-zA-Z]{2,4}$/', $datos["email"])){

                $json=array(

                    "status"=>404,

                    "detalle"=>"Error en el campo de email, ingrese un email correcto"

                );

                echo json\_encode($json,true);

                return;

            }

        }

    }

?>

**Correos repetidos**

Validamos si el correo que se introduzca ya está en la base de datos, por tanto, necesitamos la conexión con la tabla clientes, esta se genera por medio de clientes.modelo.php por la cual se establece la conexión y la sentencia SQL.

Después se realiza la validación en clientes.controlador.php por medio de un foreach que recorra toda la tabla y valide el correo.

**Clientes.modelo.php:**

<?php

    require\_once "conexion.php";

    class ModeloClientes{

        // Mostrar todos los registros

        static public function index($tabla){

            /\*

                $stmt llama a la clase Conexion y con los : llama su funcion conectar junto al metodo PHP

                prepare donde se le pasa la linea de sql y llamar a la tabla que necesitamos

                $stmt calls the Conexion class and with the : calls its conectar function together with the PHP

                prepare method where the sql line is passed and calls the table we need.

            \*/

            $stmt = Conexion::conectar()->prepare("SELECT \* FROM $tabla");

            // Para ejecutar stmt

            // To run stmt

            $stmt->execute();

            /\*

                Retornamos stmt junto a la funcion fetchAll para devolver todos los datos de la base de datos.

                We return stmt along with the fetchAll function to return all the data of the database.

            \*/

            return $stmt->fetchAll();

            // Cerramos la conexión

            // We closed the conection

            $stmt->close();

            $stmt=null;

        }

    }

?>

**Clientes.controlador.php:**

// Validar email repetido

            // Validate duplicate email

            $clientes = ModeloClientes::index("clientes");

            foreach($clientes as $key=>$value){

                if($value["email"] == $datos["email"]){

                    $json=array(

                        "status"=>404,

                        "detalle"=>"Error el email está repetido"

                    );

                    echo json\_encode($json,true);

                    return;

                }

            }

**Generar id y llave secreta del cliente:**

Generamos estos atributos por medio de la función crypt

// Generación credenciales del cliente

            // Generación credentials client

            /\*

                La función crypt pide un hash, que es una cadena de caracteres, con lo cual genera esa llave

                o id que necesitamos. En este caso reemplazamos el simbolo $ por una letra.

                The crypt function asks for a hash, which is a string of characters, with which it generates

                the key or id that we need. In this case we replace the $ simbol with a letter.

            \*/

            $id\_cliente = str\_replace("$","c",crypt($datos["nombre"].$datos["apellido"].$datos["email"] , '$2a$07$afartwetsdAD52356FEDGsfhsd$'));

            $llave\_secreta = str\_replace("$","a",crypt($datos["email"].$datos["apellido"].$datos["nombre"] , '$2a$07$afartwetsdAD52356FEDGsfhsd$'));

**Agregar clientes a la base de datos:**

En clientes.controlador.php definimos un array que contiene todos los datos del cliente.

Por medio de una función en clientes.modelo.php establecemos la sentencia SQL para insertar a la base de datos, esta llama a los datos mencionados anteriormente y los conecta por medio de la función bindParam.

Por ultimo en clientes.controlador.php, llamamos la función de clientes.modelo.php junto a los datos ya traídos para agregar a la base de datos.

**Clientes.controlador.php:**

/\*

                Traemos los datos que vienen desde rutas (nombre, apellido y email) para agregarlos junto

                al resto, con el id\_cliente y llave\_secreta generados anteriormente, además de la fecha de la

                creación y actualización.

                We bring the data that comes from routes (nombre, apellido and email) to add them together with

                the rest, with the id\_cliente and llave\_secreta generated previously, in addition to the date

                of creation and update.

            \*/

            $datos=array(

                "nombre"=>$datos["nombre"],

                "apellido"=>$datos["apellido"],

                "email"=>$datos["email"],

                "id\_cliente"=>$id\_cliente,

                "llave\_secreta"=>$llave\_secreta,

                "create\_at"=>date('Y-m-d h:i:s'),

                "update\_at"=>date('Y-m-d h:i:s')

            );

            /\*

                Traemos la función create del ModeloClientes pasandole el nombre de la tabla (contenido en

                $clientes) y los datos para agregar.

                We bring the create funcion of the ModeloClientes passing it the name of the table

                (contained in $clientes) and the data to add.

            \*/

            $create = ModeloClientes::create($clientes,$datos);

            if($create=="ok"){

                $json=array(

                    "status"=>404,

                    "detalle"=>"se genero sus credenciales",

                    "credenciales"=>$id\_cliente,

                    "llave\_secreta"=>$llave\_secreta

                );

                echo json\_encode($json,true);

                return;

            }

**Clientes.modelo.php:**

// Agregar a la base de datos

        // Add to database

        static public function create($tabla,$datos){

            $stmt = Conexion::conectar()->prepare("

                INSERT INTO clientes(nombre, apellido,email, id\_cliente, llave\_secreta,

                created\_at, updated\_at) VALUES (:nombre, :apellido,:email, :id\_cliente, :llave\_secreta,

                :created\_at, :updated\_at)");

            /\*

                bindParam enlaza el :nombre con el array $datos y el PARAM\_STR lo define como parametro de tipo

                String

            \*/

            $stmt -> bindParam(":nombre", $datos["nombre"], PDO::PARAM\_STR);

            $stmt -> bindParam(":apellido", $datos["apellido"], PDO::PARAM\_STR);

            $stmt -> bindParam(":email", $datos["email"], PDO::PARAM\_STR);

            $stmt -> bindParam(":id\_cliente", $datos["id\_cliente"], PDO::PARAM\_STR);

            $stmt -> bindParam(":llave\_secreta", $datos["llave\_secreta"], PDO::PARAM\_STR);

            $stmt -> bindParam(":created\_at", $datos["created\_at"], PDO::PARAM\_STR);

            $stmt -> bindParam(":updated\_at", $datos["updated\_at"], PDO::PARAM\_STR);

            if($stmt->execute()){

                return "ok";

            }else{

                print\_r(Conexion::conectar()->errorInfo());

            }

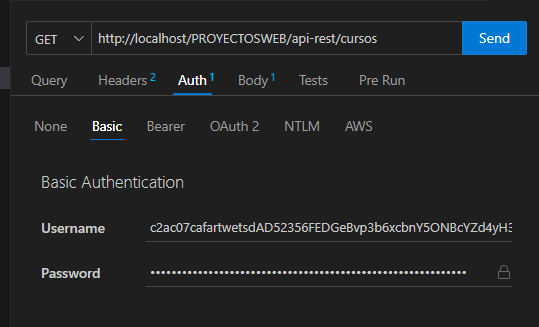
            $stmt->close();

            $stmt=null;

        }

**Acceso a la API por medio de user y password:**

Se usa la auth básica de username y password



Esto se hara por medio de verificaciones, en el método del GET (index), se usará variables globales de PHP, donde se verificará con la tabla para ver si coinciden con las existentes en la base de datos, de ser así mostrara los cursos que existen.

**Clientes.controlador.php:**

public function index(){

            // Validar credenciales del cliente

            // Validate client credentials

            $clientes = ModeloClientes::index("clientes");

            /\*

                Estas son 2 variables globales de PHP, para el user y el password

                These are 2 global PHP variables, for the user and the password

            \*/

            if(isset($\_SERVER['PHP\_AUTH\_USER']) && isset($\_SERVER['PHP\_AUTH\_PW'])){

                /\*

                    Recorremos la tabla definida anteriormente por medio del foreach

                    We traverse the table defined above using the foreach

                \*/

                foreach($clientes as $key=>$value){

                    /\*

                        Verificamos el usuario y la contraseña por las 2 variables globales con la tabla además

                        de usar el método base64\_encode para codificar más los datos

                        We verify the user and the password by the 2 global variables with the table in addition

                        to using the base64\_encode method to further encode the data

                    \*/

                    if(base64\_encode($\_SERVER['PHP\_AUTH\_USER'].":".$\_SERVER['PHP\_AUTH\_PW']) == base64\_encode($value["id\_cliente"].":".$value["llave\_secreta"])){

                            $cursos = ModeloCursos::index("cursos");

                            $json=array(

                                "detalle"=>$cursos,

                            );

                            echo json\_encode($json,true);

                            return;

                    }

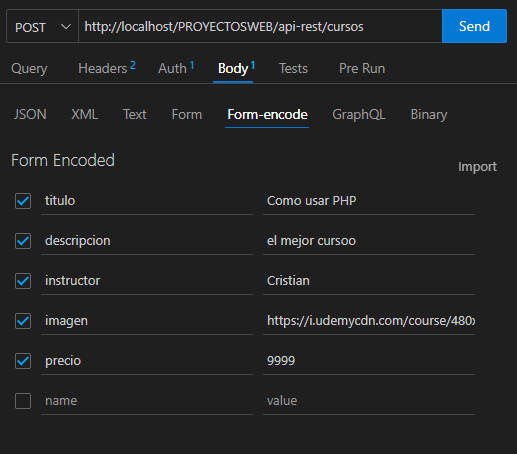
                }

            }

        }

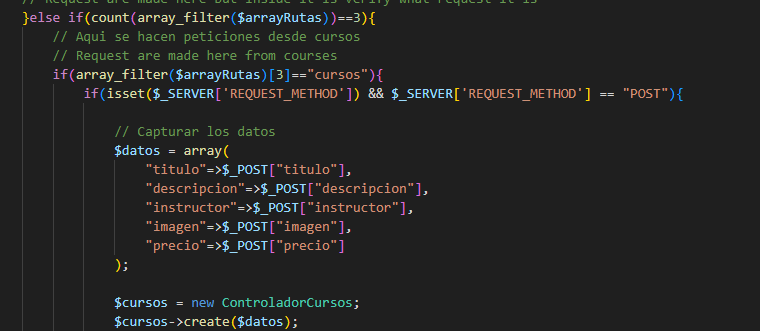
**Agregar un curso (POST):**

El usuario envía los datos por medio del método POST



**Rutas.php:**

En rutas.php capturamos los datos por medio del método POST y los guardamos en un array, estos datos los enviamos a la función créate del ControladorCursos



**Cursos.controlador.php:**

Dentro de la función créate se agrega la tabla clientes por medio del ModeloClientes; Se necesita la auth por medio de user y password como ya lo habíamos visto y se validan los datos ingresador por el usuario, además de verificar si el titulo o la descripción del curso ya están en la base de datos.

Por medio de un array donde se guardan todos los datos se le envía a ModeloCursos y si se obtiene una respuesta correcta se le informa al usuario.

public function create($datos){

            // Validar credenciales del cliente

            // Validate client credentials

            $clientes = ModeloClientes::index("clientes");

            if(isset($\_SERVER['PHP\_AUTH\_USER']) && isset($\_SERVER['PHP\_AUTH\_PW'])){

                foreach($clientes as $key=>$valueCliente){

                    if(base64\_encode($\_SERVER['PHP\_AUTH\_USER'].":".$\_SERVER['PHP\_AUTH\_PW']) == base64\_encode($valueCliente["id\_cliente"].":".$valueCliente["llave\_secreta"])){

                        // Validar datos

                        // Data validate

                        foreach($datos as $key=>$valueDatos){

                            /\*

                                Por medio del if valida si existen los datos y que se permitan simbolos y

                                letras por medio del preg\_match.

                                Using if, validates whether the data exists and that simbols and

                                letter are allowed using preg\_match.

                            \*/

                            if(isset($valueDatos) && !preg\_match('/^[(\\)\\=\\&\\$\\;\\-\\\_\\\*\\"\\<\\>\\?\\¿\\!\\¡\\:\\,\\.\\0-9a-zA-ZñÑáéíóúÁÉÍÓÚ ]+$/', $valueDatos)){

                                $json = array(

                                    "status"=>404,

                                    "detalle"=>"Error en el campo ".$key

                                );

                                echo json\_encode($json, true);

                                return;

                            }

                        }

                        // Validar que el titulo o la descripcion no esten repetidos

                        // Validate that the title or the description is not repeated

                        $cursos = ModeloCursos::index("cursos");

                        foreach($cursos as $key=>$value){

                            if($value->titulo==$datos["titulo"]){

                                $json = array(

                                    "status"=>404,

                                    "detalle"=>"El titulo ya existe en la base de datos"

                                );

                                echo json\_encode($json, true);

                                return;

                            }

                            if($value->descripcion==$datos["descripcion"]){

                                $json = array(

                                    "status"=>404,

                                    "detalle"=>"La descripción ya existe en la base de datos"

                                );

                                echo json\_encode($json, true);

                                return;

                            }

                        }

                        // Llevar datos al modelo

                        // Bringing data into the model

                        $datos=array(

                            "titulo"=>$datos["titulo"],

                            "descripcion"=>$datos["descripcion"],

                            "instructor"=>$datos["instructor"],

                            "imagen"=>$datos["imagen"],

                            "precio"=>$datos["precio"],

                            "id\_creador"=>$valueCliente["id"],

                            "create\_at"=>date('Y-m-d h:i:s'),

                            "update\_at"=>date('Y-m-d h:i:s')

                        );

                        $create = ModeloCursos::create("cursos", $datos);

                        // Respuesta del ModeloCurso

                        // ModelCurso response

                        if($create=="ok"){

                            $json=array(

                                "status"=>200,

                                "detalle"=>"Registro exitoso, curso registrado"

                            );

                            echo json\_encode($json,true);

                            return;

                        }

                    }

                }

            }

        }

**Curso.modelo.php:**

Por medio de la función créate, se usa la conexión a la base de dato y se define la sentencia SQL para insertar los datos, que se conectan por medio del bindParam, donde estos datos vienen desde curso.controlador.php (como lo vimos anteriormente), si funciona de manera correcta se retorna ok para informar.

static public function create($tabla, $datos){

            $stmt = Conexion::conectar()->prepare("INSERT INTO $tabla(titulo, descripcion,

            instructor, imagen, precio, id\_creador, created\_at, updated\_at)

            VALUES (:titulo, :descripcion, :instructor, :imagen, :precio, :id\_creador, :created\_at, :updated\_at)");

            /\*

                bindParam enlaza el :nombre con el array $datos y el PARAM\_STR lo define como parametro de tipo

                String

            \*/

            $stmt->bindParam(":titulo", $datos["titulo"], PDO::PARAM\_STR);

            $stmt->bindParam(":descripcion", $datos["descripcion"], PDO::PARAM\_STR);

            $stmt->bindParam(":instructor", $datos["instructor"], PDO::PARAM\_STR);

            $stmt->bindParam(":imagen", $datos["imagen"], PDO::PARAM\_STR);

            $stmt->bindParam(":precio", $datos["precio"], PDO::PARAM\_STR);

            $stmt->bindParam(":id\_creador", $datos["id\_creador"], PDO::PARAM\_STR);

            $stmt->bindParam(":created\_at", $datos["created\_at"], PDO::PARAM\_STR);

            $stmt->bindParam(":updated\_at", $datos["updated\_at"], PDO::PARAM\_STR);

            if($stmt->execute()){

                return "ok";

            }else{

                print\_r(Conexion::conectar()->errorInfo());

            }

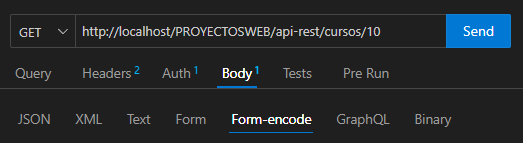
            $stmt->close();

            $stmt=null;

        }

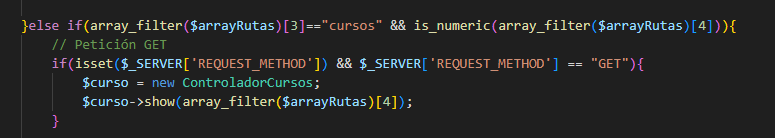
**Obtener cursos por medio del id:**

Pasamos el id del curso por medio de la URL



**Rutas.php:**

Dentro del if que ya teníamos para la petición GET usamos el método show, el cual necesita el id del curso que necesita el usuario.



**Curso.controlador.php:**

Dentro de la función show, traemos la tabla cliente para verificar si el usuario es correcto (como ya se hizo anteriormente), después se le manda el id al método show de curso.modelo.php, si se obtiene una respuesta correcta se manda al usuario.

// Mostrar cursos desde el id

        // Show course from the id

        public function show($id){

            // Validar credenciales del cliente

            // Validate client credentials

            $clientes = ModeloClientes::index("clientes");

            if(isset($\_SERVER['PHP\_AUTH\_USER']) && isset($\_SERVER['PHP\_AUTH\_PW'])){

                foreach($clientes as $key=>$valueCliente){

                    if(base64\_encode($\_SERVER['PHP\_AUTH\_USER'].":".$\_SERVER['PHP\_AUTH\_PW']) == base64\_encode($valueCliente["id\_cliente"].":".$valueCliente["llave\_secreta"])){

                        // Mostrar los cursos por medio del id

                        $curso = ModeloCursos::show("cursos",$id);

                        // Verificar que el curso exista

                        if(!empty($curso)){

                            $json=array(

                                "status"=>200,

                                "detalle"=>$curso

                            );

                            echo json\_encode($json,true);

                            return;

                        }else{

                            $json=array(

                                "status"=>200,

                                "total\_registros"=>0,

                                "detalle"=>"No hay ningun curso registrado"

                            );

                            echo json\_encode($json,true);

                            return;

                        }

                    }

                }

            }

        }

**Curso.modelo.php:**

Dentro de la función show, se realiza la conexión con la base de datos y se le manda la sentencia SQL del SELECT con el id especifico que se necesita, esto se le manda al curso.controlador.php.

static public function show($tabla, $id){

            /\*

                $stmt llama a la clase Conexion y con los : llama su funcion conectar junto al metodo PHP

                prepare donde se le pasa la linea de sql y llamar a la tabla que necesitamos

                $stmt calls the Conexion class and with the : calls its conectar function together with the PHP

                prepare method where the sql line is passed and calls the table we need.

            \*/

            $stmt = Conexion::conectar()->prepare("SELECT \* FROM $tabla WHERE id=:id");

            $stmt->bindParam(":id", $id, PDO::PARAM\_INT);

            // Para ejecutar stmt

            // To run stmt

            $stmt->execute();

            /\*

                Retornamos stmt junto a la funcion fetchAll para devolver todos los datos de la base de datos.

                Dentro del fetchAll llamamos un parametro del PDO para que devuelva la informacion sin

                duplicar, solo las propiedades de la tabla (FETCH\_CLASS).

                We return stmt along with the fetchAll function to return all the data of the database.

                Inside the fetchAll we call a parameter of the PDO to return the information without duplicating

                it, only the properties of the table (FETCH\_CLASS).

            \*/

            return $stmt->fetchAll(PDO::FETCH\_CLASS);

            // Cerramos la conexión

            // We closed the conection

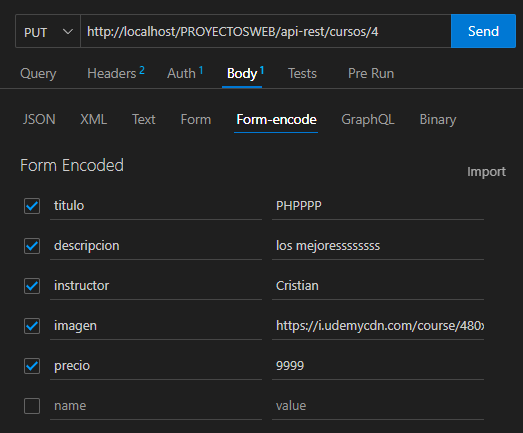
            $stmt->close();

            $stmt=null;

        }

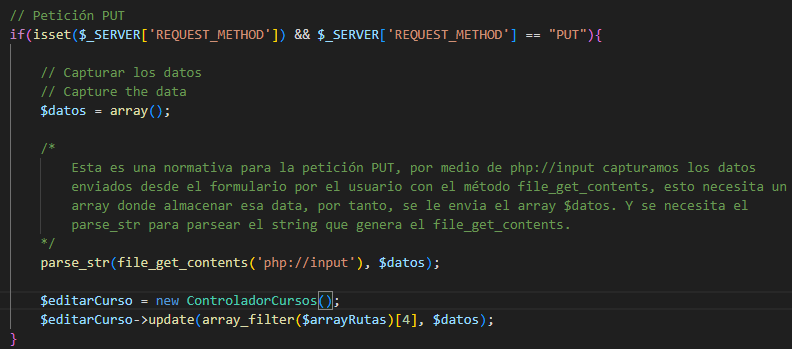
**Actualizar curso (UPDATE):**

Vamos a capturar los datos introducidos por el usuario, además de definir que curso por medio del URL



**Rutas.php:**

Por medio de file\_get\_contents y parse\_str capturamos los datos enviados por el usuario, estos son enviados al ControladorCusos y su función update



**Curso.controlador.php:**

En la función update recibimos los datos para actualizar del usuario y el id. Primero verificamos el auth de user y password, seguido validamos los datos que correspondan con símbolos correctos y validamos que el creador del curso sea quien este actualizando el curso, es decir, nadie más puede hacer esta acción. Seguido a esto capturamos todos los datos para enviárselos a la función update de clientes.modelo.php.

Si obtenemos una respuesta correcta, se informa al usuario de lo contrario se mostrará el error.

public function update($id, $datos){

            // Validar credenciales del cliente

            // Validate client credentials

            $clientes = ModeloClientes::index("clientes");

            if(isset($\_SERVER['PHP\_AUTH\_USER']) && isset($\_SERVER['PHP\_AUTH\_PW'])){

                foreach($clientes as $key=>$valueCliente){

                    //if(base64\_encode($\_SERVER['PHP\_AUTH\_USER'].":".$\_SERVER['PHP\_AUTH\_PW']) == base64\_encode($valueCliente["id\_cliente"].":".$valueCliente["llave\_secreta"])){

                    if( "Basic ".base64\_encode($\_SERVER['PHP\_AUTH\_USER'].":".$\_SERVER['PHP\_AUTH\_PW']) == "Basic ".base64\_encode($valueCliente["id\_cliente"].":".$valueCliente["llave\_secreta"]) ){

                        // Validar datos

                        // Data validation

                        foreach ($datos as $key => $valueDatos) {

                            if(isset($valueDatos) && !preg\_match('/^[(\\)\\=\\&\\$\\;\\-\\\_\\\*\\"\\<\\>\\?\\¿\\!\\¡\\:\\,\\.\\0-9a-zA-ZñÑáéíóúÁÉÍÓÚ ]+$/', $valueDatos)){

                                $json = array(

                                    "status"=>404,

                                    "detalle"=>"Error en el campo ".$key

                                );

                                echo json\_encode($json, true);

                                return;

                            }

                        }

                        // Validar id del creador

                        // Validate creator id

                        $curso = ModeloCursos::show("cursos",$id);

                        foreach($curso as $key=>$valueCurso){

                            if($valueCurso->id\_creador == $valueCliente["id"]){

                                // Llevar datos al modelo

                                // Bringing data into the model

                                $datos = array(

                                    "id"=>$id,

                                    "titulo"=>$datos["titulo"],

                                    "descripcion"=>$datos["descripcion"],

                                    "instructor"=>$datos["instructor"],

                                    "imagen"=>$datos["imagen"],

                                    "precio"=>$datos["precio"],

                                    "updated\_at"=>date('Y-m-d h:i:s')

                                );

                                $update = ModeloCursos::update("cursos", $datos);

                                if($update=="ok"){

                                    $json=array(

                                        "status"=>200,

                                        "detalle"=>"Actualización exitosa, curso actualizado"

                                    );

                                    echo json\_encode($json,true);

                                    return;

                                }else{

                                    $json=array(

                                        "status"=>404,

                                        "detalle"=>"No esta autorizado para actualizar este curso"

                                    );

                                    echo json\_encode($json,true);

                                    return;

                                }

                            }

                        }

                    }

                }

            }

        }

**Cursos.modelo.php:**

En la función update, realizamos la conexión con la base de datos y establecemos la sentencia UPDATE con el id que necesitamos, damos los datos que nos manda curso.controlador.php por medio del método bindParam y seguido a esto informamos si fue correcta la ejecución.

static public function update($tabla, $datos){

            $stmt=Conexion::conectar()->prepare("UPDATE $tabla SET titulo=:titulo,descripcion=:descripcion,

            instructor=:instructor,imagen=:imagen,precio=:precio,updated\_at=:updated\_at WHERE id=:id");

            $stmt->bindParam(":id", $datos["id"], PDO::PARAM\_STR);

            $stmt->bindParam(":titulo", $datos["titulo"], PDO::PARAM\_STR);

            $stmt->bindParam(":descripcion", $datos["descripcion"], PDO::PARAM\_STR);

            $stmt->bindParam(":instructor", $datos["instructor"], PDO::PARAM\_STR);

            $stmt->bindParam(":imagen", $datos["imagen"], PDO::PARAM\_STR);

            $stmt->bindParam(":precio", $datos["precio"], PDO::PARAM\_STR);

            $stmt->bindParam(":updated\_at", $datos["updated\_at"], PDO::PARAM\_STR);

            if($stmt->execute()){

                return "ok";

            }else{

                print\_r(Conexion::conectar()->errorInfo());

            }

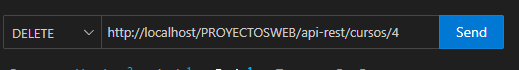
            $stmt->close();

            $stmt=null;

        }

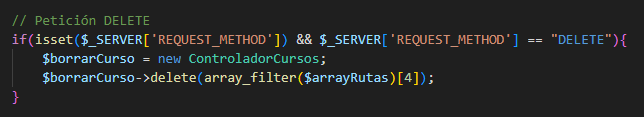
**Borrar curso (DELETE):**

Se elimina mediante la URL, pasándole el id del curso.



**Rutas.php:**

Ubicamos la petición DELETE y usamos la función delete del controlador cursos, pasándole el id que se encuentra en la URL.



**Cursos.controlador.php:**

En la función delete, recibimos el id y validamos que el cliente este en la base de datos, seguido a esto, validamos que este cliente si sea el que creo el curso para que pueda realizar esta acción. Después, le mandamos el id y el nombre de la tabla a la función delete de ModeloCursos.

Validamos si fue correcta la función y notificamos al usuario.

public function delete($id){

            // Validar credenciales del cliente

            // Validate client credentials

            $clientes = ModeloClientes::index("clientes");

            if(isset($\_SERVER['PHP\_AUTH\_USER']) && isset($\_SERVER['PHP\_AUTH\_PW'])){

                foreach($clientes as $key=>$valueCliente){

                    //if(base64\_encode($\_SERVER['PHP\_AUTH\_USER'].":".$\_SERVER['PHP\_AUTH\_PW']) == base64\_encode($valueCliente["id\_cliente"].":".$valueCliente["llave\_secreta"])){

                    if( "Basic ".base64\_encode($\_SERVER['PHP\_AUTH\_USER'].":".$\_SERVER['PHP\_AUTH\_PW']) == "Basic ".base64\_encode($valueCliente["id\_cliente"].":".$valueCliente["llave\_secreta"]) ){

                        // Validar id del creador

                        // Validate creator id

                        $curso = ModeloCursos::show("cursos",$id);

                        foreach($curso as $key=>$valueCurso){

                            if($valueCurso->id\_creador == $valueCliente["id"]){

                                // Llevar datos al modelo

                                // Bringing data into the model

                                $delete = ModeloCursos::delete("cursos",$id);

                                if($delete=="ok"){

                                    $json=array(

                                        "status"=>200,

                                        "detalle"=>"El curso se borro exitosamente"

                                    );

                                    echo json\_encode($json,true);

                                    return;

                                }

                            }

                        }

                    }

                }

            }

        }

**Curso.modelo.php:**

En la función delete, realizamos la conexión con la base de datos y pasamos la sentencia SQL para realizar el delete y le pasamos el id proveniente del CursoControlador por medio del bindParam, seguido a esto notificamos si se realizó de manera exitosa.

static public function delete($tabla, $id){

            $stmt = Conexion::conectar()->prepare("DELETE FROM $tabla WHERE id=:id");

            $stmt->bindParam(":id", $id, PDO::PARAM\_INT);

            if($stmt->execute()){

                return "ok";

            }else{

                print\_r(Conexion::conectar()->errorInfo());

            }

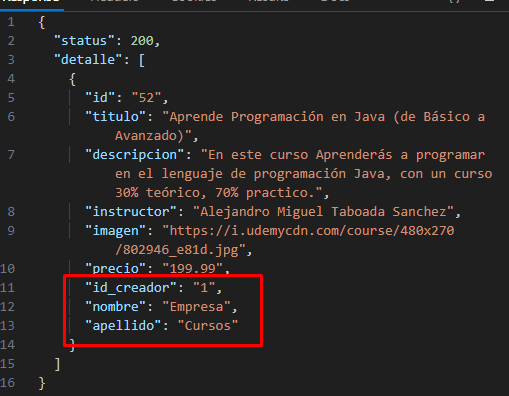
            $stmt->close();

            $stmt=null;

        }

**Selección múltiple en el GET:**

Para realizar una consulta múltiple, en este caso, hacia la tabla cursos y clientes, se realiza con un cambio en la consulta SQL.



**Cursos.controlador.php:**

Vamos a enviarle el nombre de la tabla clientes adicional a la de cursos que ya teníamos, esto lo realizamos con todas las funciones index que vengan desde ModeloCursos

$cursos = ModeloCursos::index("cursos","clientes");

Y de igual manera a todas las funciones show que vienen desde ModeloCursos para realizar estas consultas con un id especifico.

$curso = ModeloCursos::show("cursos","clientes",$id);

**Curso.modelo.php:**

Adicionamos otro atributo a la función index, en este caso $tabla1 y $tabla2

static public function index($tabla1,$tabla2)

Realizamos un cambio en la función index, en la sentencia SQL, colocamos los campos que necesitamos, especificando la tabla que pertenece cada campo y realizamos un INNER JOIN hacia la tabla 2.

$stmt = Conexion::conectar()->prepare("SELECT $tabla1.id, $tabla1.titulo, $tabla1.descripcion,

                $tabla1.instructor, $tabla1.imagen, $tabla1.precio, $tabla1.id\_creador, $tabla2.nombre,

                $tabla2.apellido FROM $tabla1 INNER JOIN $tabla2 ON $tabla1.id\_creador = $tabla2.id");

Para la función show realizamos la misma acción.

static public function show($tabla1, $tabla2, $id){

Y de igual manera cambiamos la sentencia SQL sin olvidar el WHERE para la búsqueda especifica.

 $stmt = Conexion::conectar()->prepare("SELECT $tabla1.id, $tabla1.titulo, $tabla1.descripcion,

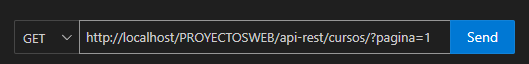
                $tabla1.instructor, $tabla1.imagen, $tabla1.precio, $tabla1.id\_creador, $tabla2.nombre,

                $tabla2.apellido FROM $tabla1 INNER JOIN $tabla2 ON $tabla1.id\_creador = $tabla2.id WHERE $tabla1.id=:id");

            $stmt->bindParam(":id", $id, PDO::PARAM\_INT);

**Paginación API:**

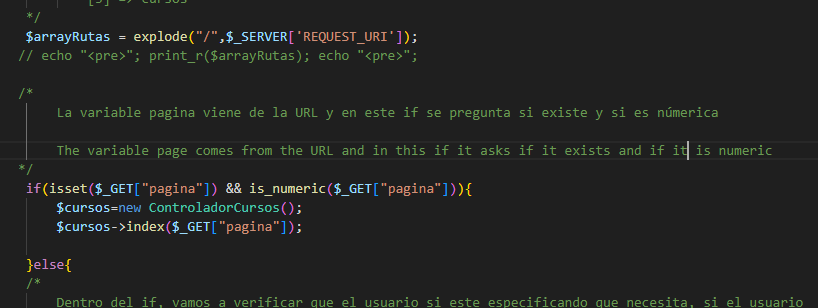
Para realizar la paginación, se hará mediante el URL y la palabra “pagina” junto al número de la página.



**Rutas.php:**

Se realizará un if para verificar que existe la variable página dentro del URL, con una modificación en la función index del ControladorCurso, se mandara el número de esta variable (página).

Todo el resto del código se colocara dentro del else.

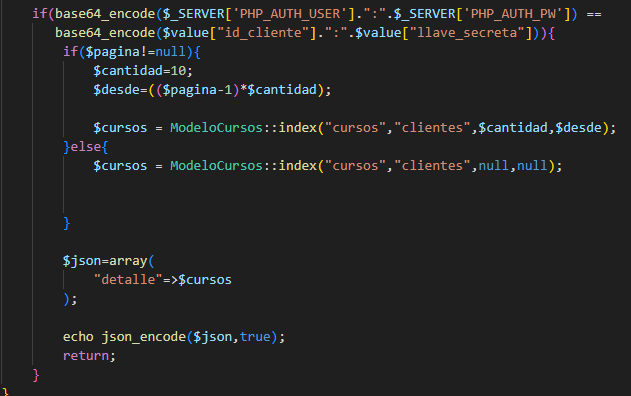


**Cursos.controlador.php:**

En la función index se recibirá el número de la página.

public function index($pagina){

Dentro de la verificación del usuario, se pregunta si se está pidiendo página, de ser así se le manda (con una modificación en la función index de ModeloCursos) la cantidad de la página, en este caso 10, y desde que página, donde con una formula manda desde que registro mostrar. De no necesitar página se manda null.



**Curso.modelo.php:**

Se modifica la función index para recibir la cantidad de la página y desde que registro mostrar.

static public function index($tabla1,$tabla2,$cantidad,$desde){

Mediante un if se verifica si se necesita una cantidad, de ser así mediante la sentencia SQL se define un LIMIT usando las variables $cantidad y $desde. De no necesitar un número de página, se usa la misma sentencia SQL sin el LIMIT.

