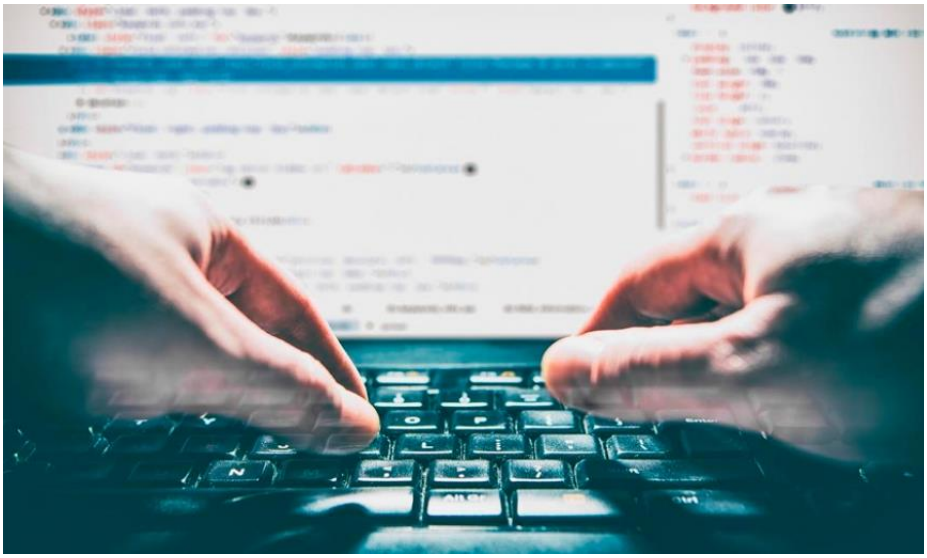
David Alejandro López Torres

25/03/2021



CENTRO DE ENSEÑANZA TÉCNICA INDUSTRIAL

Apuntes PDO vs MYSQLI

Materia: Programación Web 2

Carrera: Desarrollo de Software

Implementación

Instrucciones

Revisa la presentación MYSQLI VS PDO y realiza un resumen en Word, llamado apuntes_pdovsmysqli.docx.

Nota: La actividad que se menciona en la última diapositiva no es necesaria realizarla aún, pero es parte del producto integrador del segundo parcial.

Desarrollo

Hoy en día es muy importante por no decir imprescindible saber gestionar datos de una Base de Datos desde PHP. Claro que haremos muchas cosas que no tendrán que trabajar contra una base de datos pero hoy por hoy, la gestión de los datos es de suma importancia. Debemos almacenar, borrar, gestionar datos en una base de datos. Usualmente se trabaja con bases de datos MySQL , por lo que es indispensable estar familiarizado con el motor de bases de datos PHPMyAdmin.

La interacción con una base de datos desde el punto de vista de PHP consta los siguientes pasos:

1. Conexión
2. Consultas
    a. De recuperación de datos : select
    b. De modificación: insert, delete, update
3. Cerrar conexión

En general se habla de dos métodos para conectarse a una base de datos SQL desde PHP: son MySQLi y PDO. Uno de los cambios más importantes es que ambos admiten “declaraciones preparadas”, que eliminan la posibilidad de

ataques de inyección SQL al realizar cambios en la base de datos. Las funciones originales 'MySQL_' están en desuso y no deben utilizarse ya que no son seguras debido a que se ha eliminado su soporte.

MySQLi

MySQLi significa MySQL Improved (mejorado) y agrega nuevas características específicamente a la interfaz de MySQL. PDO significa PHP Data Object (Objeto de datos PHP). La diferencia principal entre PDO y MySQLi es que PDO es compatible con varios tipos de bases de datos diferentes (MySQL, MS SQL, Postgre DB) en el mismo script, sin embargo, solo tienes que escribir funciones relacionadas con los datos una vez. PDO está 'orientado a objetos', la conexión a la base de datos se realiza al crear un objeto variable.

Las nuevas implementaciones con respecto a la API de MySQL original son:

- Interfaz orientada a objetos y procedimental
- Soporte para sentencias preparadas
- Soporte para múltiples sentencias
- Soporte para transacciones
- Mejoradas las opciones de depuración
- Muchas más funcionalidades disponibles
- Soporte para procedimientos almacenados (PL/SQL)
- Mayor seguridad
- Mayor comodidad

En caso de programar en PHP con un enfoque a objetos, MySQLi cuenta con interfaz orientada a objetos, es decir, nos permite llamar a métodos de MySQLi con el operador de acceso "->" de esta forma sentiremos de modo que se vuelve una tarea intuitiva manipular a los objetos y métodos en PHP.

Si trabajamos con programación procedimental(obsoleta) en PHP no tendremos problema para usar MySQLi ya que cuenta con una interfaz procedimental, se usa de una forma muy similar al API tradicional de MySQL.

La sintaxis para las principales tareas realizadas con MySQLi tienen la siguiente sintaxis:

Conexión a la base de datos:

```
$conexion=new mysqli('servidor','usuario','contraseña','base de datos');
```

Consulta en la base de datos:

```
$conexion->query('CONSULTA SQL');
```

Recorrer los resultados de una consulta:

```
$consulta->fetch_assoc();
```

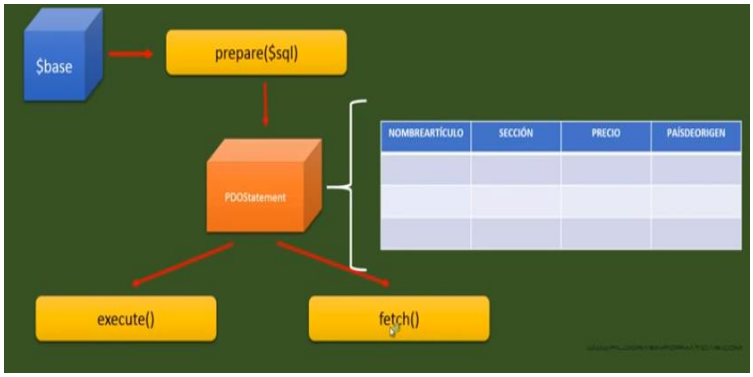
Número de filas de un conjunto de resultados de una sentencia:

```
$consulta->num_rows;
```

PDO

Se trata de una librería que tiene disponible para utilizar las funciones para programar las conexiones y las consultas. Pero con el paradigma de orientación a objetos. Está en medio del código PHP y la base de datos, llámese MySQL, Oracle, Sybase o de cualquier proveedor

El flujo normal de trabajo con PDO consiste en una serie de etapas parecidas a MySQLi: Se crea la conexión, instanciando al objeto $base. El método prepare() forma parte de la clase de la conexión y es el que admite un argumento del tipo SQL, es decir la instrucción. Nos devuelve un PDOStatement, que es como una tabla virtual. Utilizamos los métodos: execute(), para ejecutar las consultas, y fetch() después si es necesario asociarlo a un arreglo para utilizar los valores



MySQLi: Ejemplo de conexión

```
<?php
$server     = 'localhost'; //servidor
$username   = 'xxxxxx'; //usuario de la base de datos
$password   = 'yyyyyy'; //password del usuario de la base de datos
$database   = 'mibasedatos'; //nombre de la base de datos

$conexion = new mysqli();
@$conexion->connect($server, $username, $password, $database);
/*
Las 2 lineas de arriba se pueden resumir en:
$conexion = @new mysqli($server, $username, $password, $database);
El @ adelante de las funciones significa que no generará error o warnings
*/
if ($conexion->connect_error) //verificamos si hubo un error al conectar, recuerden
que pusimos el @ para evitarlo
{
    die('Error de conexión: ' . $conexion->connect_error); //si hay un error termina
la aplicación y mostramos el error
}
```

PDO: Ejemplo de conexión

Una conexión de base de datos PDO requiere que crees un nuevo 'objeto PDO' con un Nombre de origen de datos (DSN – Data Source Name), nombre de usuario y contraseña. El DSN define el tipo de base de datos, el nombre de la base de datos y cualquier otra información, si es necesario. El DSN puede ser una variable simple que luego se usa como parámetro al crear el objeto PDO real, como se muestra en el siguiente código.

PDO es compatible con varios tipos de bases de datos y el DSN es donde defines las conexiones alternativas, reemplazando la línea 'mysql:' con la sintaxis de la otra base de datos. En un script real, puedes permitirle al usuario elegir qué conexión utilizar y escribir código que elija la variable DSN requerida.

Al crear el objeto PDO que representa la conexión de la base de datos, puedes envolverlo en el código 'try … catch …'. Esto significa que el script intentará conectarse usando el código provisto, y si hay algún problema, se ejecutará el código en la sección 'catch'. Puedes usar el bloque catch para mostrar mensajes de error o ejecutar un código alternativo si falla la prueba. En este ejemplo, se muestra un mensaje de error simple para decirte qué parte de la información era incorrecta.

```
<?php
if($_SERVER['REQUEST_METHOD'] == 'POST') {
    // guardas
    $servername = "localhost";
    $database = "practicaspw";
    $username = "root";
    $password = "";
    $sql = "mysql:host=$servername;dbname=$database;";
    $dsn_Options = [PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION];
```

```
  try {
    $cnx = new PDO($sql, $username, $password, $dsn_Options);
    echo "Conexión exitosa por PDO";
} catch (PDOException $error) {
    echo 'Error de conexión: ' . $error->getMessage();
}
finally{
    echo "<br>";

}
```

Errores de conexión PHP MySQL

Existen pequeñas diferencias entre los errores de conexión entre PHP y MySQLi por parte de PDO y MySQLi. Los errores más generales que pueden aparecer son:

Error de contraseña incorrecta: Por ejemplo, si cambiamos la contraseña en el código PHP un poco (pero no la cambiamos en la base de datos).

Error con MySQLi

Connection failed: Access denied for user 'suchastu_buyusr'@'localhost' (using password: YES)

Error con PDO

Connection failed: SQLSTATE[HY000] [1045] Access denied for user 'u266072517_user'@'mysql.hostinger.co.uk' (using password: YES)

No se puede conectar con el servidor MySQL

Error con MySQLi

Connection failed: Can't connect to MySQL server on 'server' (110)

Significa que el script no obtuvo una respuesta de un servidor, eso sucedió porque hemos indicado el **servidor** en lugar del **localhost** como el nombre de servidor, y este nombre no es reconocido.

Error con PDO

Connection failed: SQLSTATE[HY000] [2002] php_network_getaddresses: getaddrinfo failed: No such host is known.

[HY000] significa *error general*; **[2002]** significa que *no se puede conectar al servidor MySQL local*. El resto de los mensajes proporciona más detalles, indicando que no se encontró el “host”.

Operaciones con la base de datos

Añadir registros

Se utiliza la operación insert. La sintaxis que se utiliza es:

Insert into tabla(campos) values (valores);

Ejemplo:

Insert into alumnos(registro, nombre, email) VALUES (“123”, “Tom As”, tom.as@correo.com);

Ejemplo con PDO

```
//los valores que se van agregar a la bd
$first_Name = "Tom";
$last_Name = “As";
$email = "tom.as@correo.com";
//llamamos al metodo prepare que ejecutará la inserción, utilizando la forma ? o
:nombredevariable o el nombre de la variable
$stmt = $pdo->prepare("INSERT INTO Students (name, lastname, email) VALUES (?,  ?,
?)");

// ejecutamos con los datos y nos regresará TRUE  si se logró insertar o FALSE en caso
contrario
if $stmt->execute(array($fistname, $lastname, $email)) {
      Mensaje de exito o fallo segun aplique
Echo “Insertados exitosamente”;
}

// Ejecutamos con los cambios
if ($stmt->execute()) {
      echo “Nuevo registro creado";
} else {
      echo “No se realizó la insercion";
}
```

Listar registros

Se utiliza la operación select. Su sintaxis es:

Select campos from tabla where condicion

Ejemplo:

Select * from mitabla

Ejemplo con MySQLi

```
$sql="SELECT * from mitabla limit 10"; //traemos 10 registros
$result = $conexion->query($sql); //usamos la conexion para dar un resultado a
 la variable
 if ($result->num_rows > 0) //si la variable tiene al menos 1 fila entonces
 seguimos con el codigo
{
    while ($row = $result->fetch_array())
    //MYSQLI_ASSOC= El arreglo como siempre, es decir [campo1]->valor, [campo2]-
 >valor
    //MYSQLI_NUM= Como indice [0]-> Valor de campo 1, [0]->valor de campo2
    //MYSQLI_BOTH= por defecto, trae ambos, es decir que si la tabla es de 2
 campos, tendremos una array de 4 posiciones
    // [0] -> Valor campo1, [campo1] -> Valor de campo1, [1] -> Valor de campo2,
 [campo2]-> Valor de Campo1
    {
        echo "<pre>";
        print_r($row); //imprimimos el contenido de array
        echo "</pre>";
        echo "------------";
    }
}
else
{
    echo "No hubo resultados";
}
/* array numérico */

$row = $result->fetch_array(MYSQLI_NUM);

printf ("%s (%s)\n", $row[0], $row[1]);


/* array asociativo */

$row = $result->fetch_array(MYSQLI_ASSOC);

printf ("%s (%s)\n", $row[“name"], $row["email"]);


/* array numérico y asociativo */

$row = $result->fetch_array(MYSQLI_BOTH);

echo “Nombre “.$row[0] . “Apellidos “. $row["lastname"]);

/*array de objetos*/

$fila = $result->fetch_object();

echo $fila->name;
```

Cerrar la conexión

Aunque la teoría dice que cuando no se usa un objeto o se termina una rutina se liberan los recursos, es muy recomendable evitar que nuestra aplicación web se sature de conexiones sin cerrar, sobre todo cuando se trabaja con conexiones persistentes, y cada una de las formas de conectarse a la base de datos en php suele tener su propia función de desconexión. Normalmente con un close() o disconnect().

PDO:

```
$ stmt->closeCursor(); // opcional en MySQL, dependiendo del controlador de base de datos puede ser obligatorio
$ stmt = null; // obligado para cerrar la conexión
$pdo = null;
```

Mysqli:

```
$result->free();
$conexion->close();
```

Nota:

Las conexiones persistentes no son cerradas al final del script, sino que son almacenadas en caché y reutilizadas cuando otro script solicite una conexión que use las mismas credenciales. La caché de conexiones persistentes permite evitar la carga adicional de establecer una nueva conexión cada vez que un script necesite comunicarse con la base de datos, dando como resultado una aplicación web más rápida.

Historia de PDO

Empezó a desarrollarse en 2003 tras unas reuniones en LinuxTag. Fue considerada experimental hasta PHP 5.0 (en el cual está disponible como una extensión PECL); a partir de PHP 5.1 se considera estable y la interfaz viene incluida por defecto. Está implementada con tecnología orientada a objetos. La conexión a una base de datos se realiza creando una instancia de la clase base PDO. Algunos métodos son: prepare, execute, exec, beginTransaction, bindParam, commit.

Los siguientes controladores actualmente implementan la interfaz PDO:

- PDO_DBLIB: FreeTDS / Microsoft SQL Server / Sybase
- PDO_FIREBIRD: Firebird / Interbase 6
- PDO_IBM: IBM DB2
- PDO_INFORMIX: IBM Informix Dynamic Server
- PDO_SQLSRV: Microsoft SQL Server
- PDO_MYSQL: MySQL 3.x/4.x/5.x

- PDO_OCI: Oracle Call Interface
- PDO_ODBC: ODBC v3 (IBM DB2, unixODBC y win32 ODBC)
- PDO_PGSQL: PostgreSQL
- PDO_SQLITE: SQLite 3 y SQLite 2

De las mencionadas, PDO_DBLIB, PDO_FIREBIRD y PDO_OCI son consideradas experimentales y su comportamiento puede variar en nuevas versiones. La última versión de PDO_SQLSRV se encuentra en Microsoft Drivers for PHP for SQL Server 2.0.1.

Ejemplo completo MySQLi

1. Base de datos: php_asistenteregistro.sql

```
CREATE TABLE IF NOT EXISTS `usuarios` (
`id` int(11) NOT NULL,
  `usuario` varchar(100) DEFAULT NULL,
  `email` varchar(150) DEFAULT NULL,
  `nombres` varchar(250) DEFAULT NULL,
  `password` varchar(80) DEFAULT NULL,
  `genero` varchar(80) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

--
-- Índices para tablas volcadas
--

--
-- Indices de la tabla `usuarios`
--
ALTER TABLE `usuarios`
 ADD PRIMARY KEY (`id`);

--
-- AUTO_INCREMENT de las tablas volcadas
--

--
-- AUTO_INCREMENT de la tabla `usuarios`
--
ALTER TABLE `usuarios`
MODIFY `id` int(11) NOT NULL AUTO_INCREMENT;
```

2. Conexión con la base de datos: dbcontroller.php

```
<?php
class DBController {
```

```
        private $host = "localhost";
        private $user = "root";
        private $password = "root";
        private $database = "php_asistenteregistro";
        private $conn;

        function __construct() {
                $this->conn = $this->connectDB();
        }

        function connectDB() {
                $conn = mysqli_connect($this->host,$this->user,$this->password,$this-
>database);
                return $conn;
        }

        function runQuery($query) {
                $result = mysqli_query($this->conn,$query);
                while($row=mysqli_fetch_array($result)) {
                        $resultset[] = $row;
                }
                if(!empty($resultset))
                        return $resultset;
        }
}
?>
```

3. Validación jQuery

```
<script>
function validate() {
var output = true;
$(".registro-error").html('');
if($("#account-field").css('display') != 'none') {
        if(!($("#usuario").val())) {
                output = false;
                $("#usuario-error").html("Campo Requerido");
        }
        if(!($("#email").val())) {
                output = false;
                $("#email-error").html("Campo Requerido");
        }
        if(!$("#email").val().match(/^([\w-\.]+@([\w-]+\.)+[\w-]{2,4})?$/)) {
                $("#email-error").html("Correo invalido");
                output = false;
        }
}

if($("#password-field").css('display') != 'none') {
        if(!($("#user-password").val())) {
                output = false;
                $("#password-error").html("Campo Requerido");
        }
        if(!($("#confirm-password").val())) {
                output = false;
                $("#confirm-password-error").html("No coinciden");
        }
        if($("#user-password").val() != $("#confirm-password").val()) {
                output = false;
                $("#confirm-password-error").html("No coinciden");
        }
}
return output;
```

```
}
$(document).ready(function() {
        $("#next").click(function(){
                var output = validate();
                if(output) {
                        var current = $(".highlight");
                        var next = $(".highlight").next("li");
                        if(next.length>0) {
                                $("#"+current.attr("id")+"-field").hide();
                                $("#"+next.attr("id")+"-field").show();
                                $("#atras").show();
                                $("#finish").hide();
                                $(".highlight").removeClass("highlight");
                                next.addClass("highlight");
                                if($(".highlight").attr("id") ==
$("li").last().attr("id")) {
                                        $("#next").hide();
                                        $("#finish").show();
                                }
                        }
                }
        });
        $("#atras").click(function(){
                var current = $(".highlight");
                var prev = $(".highlight").prev("li");
                if(prev.length>0) {
                        $("#"+current.attr("id")+"-field").hide();
                        $("#"+prev.attr("id")+"-field").show();
                        $("#next").show();
                        $("#finish").hide();
                        $(".highlight").removeClass("highlight");
                        prev.addClass("highlight");
                        if($(".highlight").attr("id") == $("li").first().attr("id")) {
                                $("#atras").hide();
                        }
                }
        });
});
</script>
```

4. Registro en la base de datos

```
<?php
if(count($_POST)>0) {
        // Llamamos al fichero dbcontroller que sera el encargado de realizar la
conexión con el servidor
        // Este controlador usa PDO para conectar. Uno de las conexiones mas seguras
        require_once("dbcontroller.php");
        // Instanciamos la clase DBController
        $db_handle = new DBController();
        // Aquí realizaremos una consulta para insertar registros a la base de datos
        $ConsultaInsercion = "INSERT INTO usuarios (usuario, email, nombres, password,
genero) VALUES
        ('" . $_POST["usuario"] . "', '" . $_POST["email"] . "', '" . $_POST["nombres"]
. "', '" . $_POST["password"] . "', '" . $_POST["genero"] . "')";
        $Resultados = $db_handle->insertQuery($ConsultaInsercion);
        // Comprobamos si la insercion fue exitosa o tuvo algun problema
        if(!empty($Resultados)) {
        $mensajeresultado = "<div class='alert alert-primary' role='alert'>Usuario
registrado correctamente!</div>";
                unset($_POST);
        } else {
```

```
        $mensajeresultado = "<div class='alert alert-danger' role='alert'>Hubo un
problema al registrar. Intente nuevamente!</div>";

        }
}
?>
```

5. Librerías externas

```
<!-- Bootstrap core CSS -->
<link href="dist/css/bootstrap.min.css" rel="stylesheet">
<!-- Custom styles for this template -->
<link href="assets/sticky-footer-navbar.css" rel="stylesheet">
<script src="http://code.jquery.com/jquery-1.10.2.js"></script>
<style>
#registro-pasos {
        margin:0;
        padding:0;
}
#registro-pasos li {
        list-style:none;
        float:left;
        padding:5px 10px;
        border-top:#EEE 1px solid;
        border-left:#EEE 1px solid;
        border-right:#EEE 1px solid;
}
#registro-pasos li.highlight {
        color: #fff;
        background-color: #17a2b8;
        border-color: #17a2b8;
        font-weight:bold;
}
#registro-form {
        clear:both;
        border:1px #EEE solid;
        padding:20px;
}
.claseinput {
        padding: 10px;
        border: #F0F0F0 1px solid;
        border-radius: 4px;
        background-color: #FFF;
        width: 50%;
}
.registro-error {
        color:#FF0000;
        padding-left:15px;
}
.mensajeresultado {
        font-weight: bold;
        width: 100%;
        padding: 10;
}
.btnOpcion {
        padding: 5px 10px;
        background-color: #09F;
        border: 0;
        color: #FFF;
        cursor: pointer;
        margin-top:15px;
}
</style>
```

6. El formulario del asistente

```
<div class="mensajeresultado">
  <?php if(isset($mensajeresultado)) echo $mensajeresultado; ?>
</div>
<ul id="registro-pasos">
  <li id="account" class="highlight">Cuenta</li>
  <li id="password">Credenciales</li>
  <li id="general">General</li>
</ul>
<form name="frmRegistration" id="registro-form" method="post">
  <div id="account-field">
    <label>Usuario</label>
    <span id="usuario-error" class="registro-error"></span>
    <div>
      <input type="text" name="usuario" id="usuario" class="claseinput" />
    </div>
    <label>Email</label>
    <span id="email-error" class="registro-error"></span>
    <div>
      <input type="text" name="email" id="email" class="claseinput" />
    </div>
  </div>
  <div id="password-field" style="display:none;">
    <label>Ingrese Contraseña</label>
    <span id="password-error" class="registro-error"></span>
    <div>
      <input type="password" name="password" id="user-password" class="claseinput" />
    </div>
    <label>Re-ingrese Contraseña</label>
    <span id="confirm-password-error" class="registro-error"></span>
    <div>
      <input type="password" name="confirm-password" id="confirm-password"
class="claseinput" />
    </div>
  </div>
  <div id="general-field" style="display:none;">
    <label>Nombre para mostrar</label>
    <div>
      <input type="text" name="nombres" id="nombres" class="claseinput"/>
    </div>
    <label>Genero</label>
    <div>
      <select name="genero" id="genero" class="claseinput" required>
        <option value=""><< Seleccione >></option>
        <option value="mujer">Mujer</option>
        <option value="hombre">Hombre</option>
      </select>
    </div>
  </div>
  <div style="margin-top:15px;">
    <input class="btn btn-primary" type="button" name="atras" id="atras" value="Atras"
style="display:none;">
    <input class="btn btn-primary" type="button" name="next" id="next"
value="Siguiente" >
    <input class="btn btn-primary" type="submit" name="finish" id="finish"
value="Finalizar" style="display:none;">
  </div>
</form>
```

Conclusiones

Gracias a la actividad desarrollada se reforzaron los conocimientos correspondientes a los conectores mysql con php que se habían abordado en la materia de bases de datos II, así como visualizar prácticamente como se realiza una implementación de código php. Los conectores de php con mysql son de suma importancia para poder realizar las operaciones básicas de un sistema de registro y consulta (CRUD), los cuáles son la base una gran cantidad de sitios web que existen hoy en día . En particular, en el desarrollo de esta actividad también se reforzaron los conocimientos de sentencias MySQL y su manera de trabajar con conexiones remotas para gestionar consultas mediante un conector. Ambos conceptos son muy importantes para el desarrollo web de hoy en día debido a la gran cantidad de interacciones que se espera por parte de los usuarios hacia la información contenida en las bases de datos, de forma que es vital conocer las configuraciones para cumplir con los requisitos de funcionalidad y seguridad mínimos para estas interacciones tan comunes.

Bibliografía

- Ninguna adicional a la brindad por la maestra en el portal.