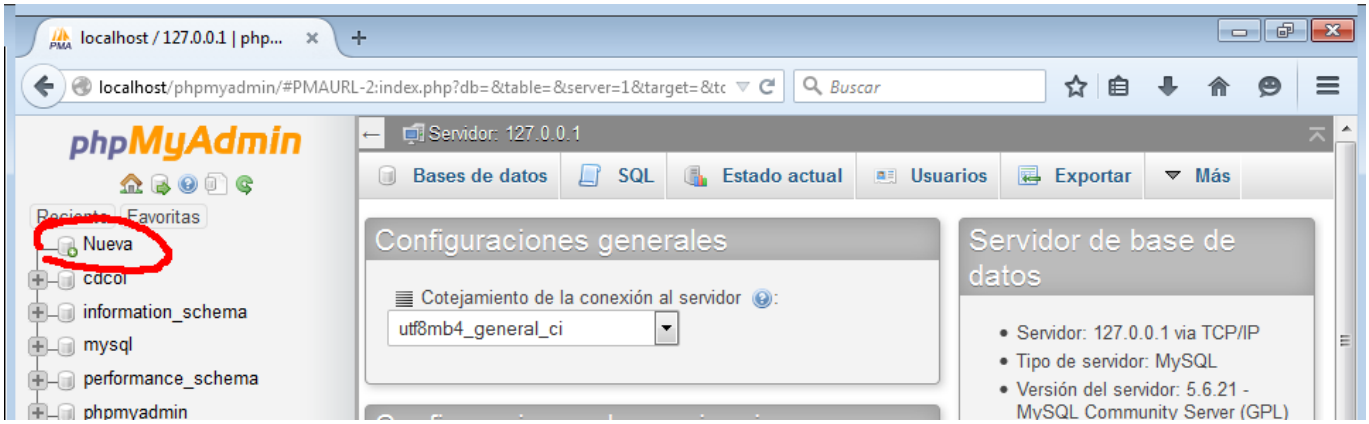
Creación de una aplicación CRUD con PHP utilizando el patrón MVC y objetos PDO

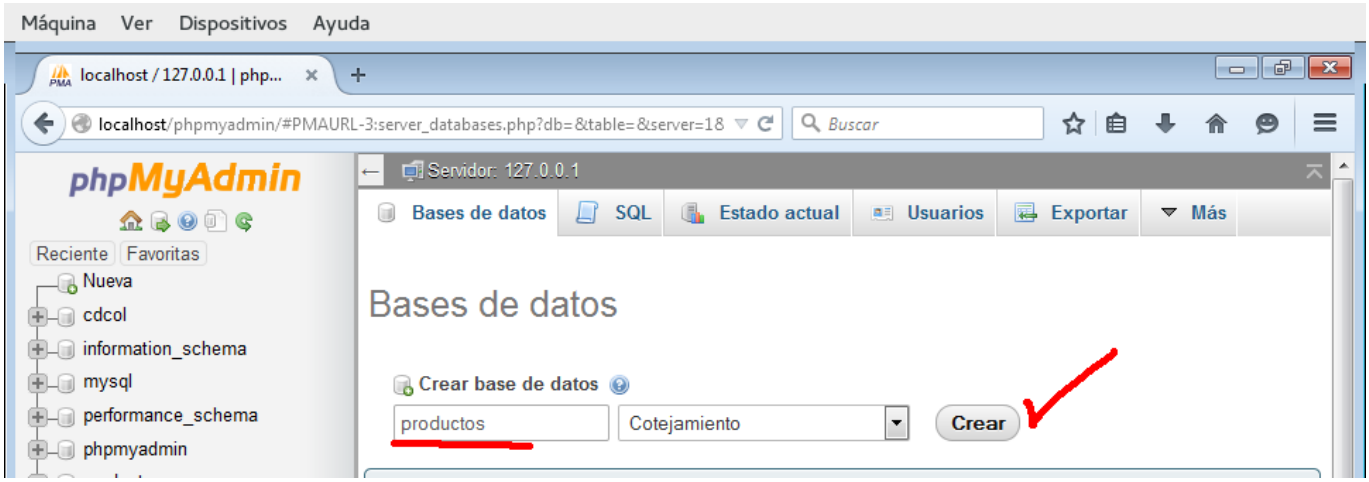
1. El objetivo de esta práctica es crear una aplicación web de tipo CRUD (Create-Read-Update-Delete) utilizando el patrón de diseño Modelo – Vista – Controlador. Para el acceso a la base de datos utilizaremos los componentes PDO (PHP Data Objects). Se requiere tener instalado el stack de aplicaciones XAMPP.

2. El siguiente diseño muestra la arquitectura de la aplicación:

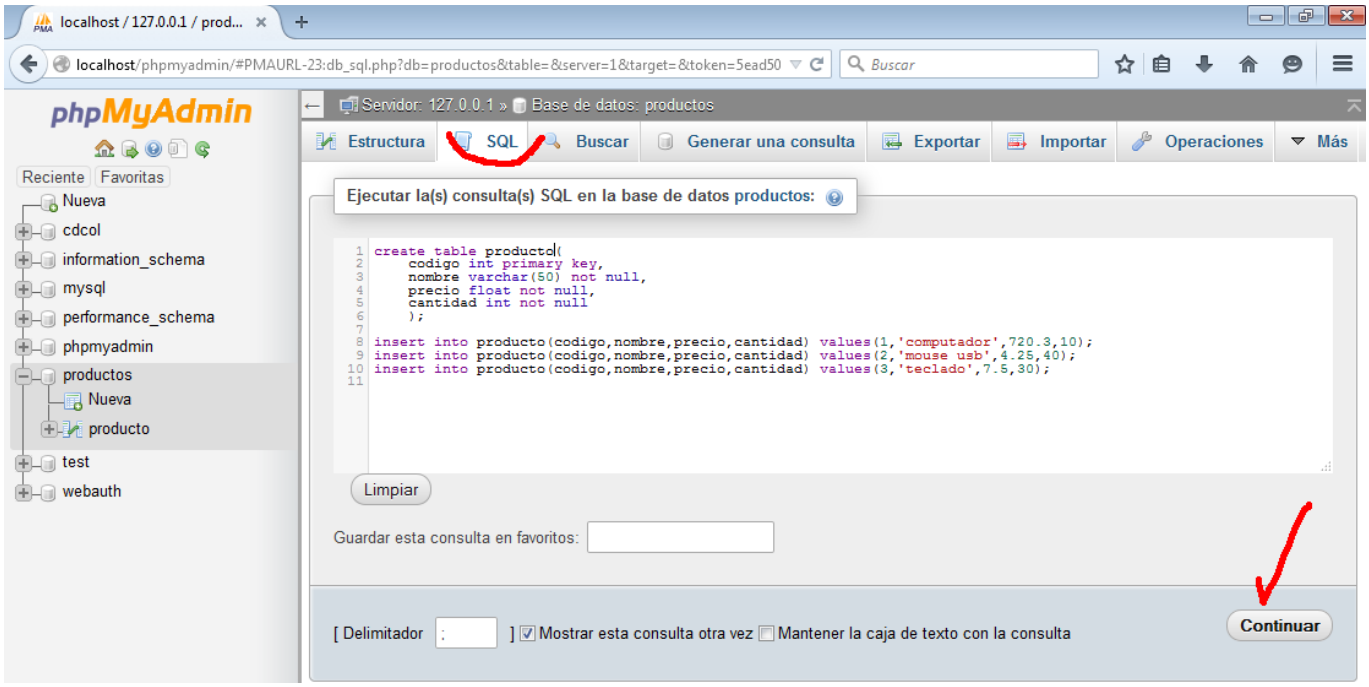
ARQUITECTURA DEL SISTEMA PHP-MVC-CRUD		
VIEW	index.php	Página inicial, muestra el listado de productos y las opciones para que el usuario pueda interactuar con el sistema.
	crear.php	Formulario de ingreso de nuevos productos.
	actualizar.php	Formulario de actualización de un producto existente.
CONTROLLER	controller.php	Componente controlador que maneja la navegación entre páginas y la ejecución de la lógica del sistema.
MODEL	ProductoModel.php	Clase para el manejo de lógica de la aplicación. En este caso implementa las operaciones CRUD utilizando componentes PDO (PHP Data Objects).
	Producto.php	Clase que representa a la tabla "producto". También se los llama "objetos de dominio" o "entidades".
	Database.php	Manejo de conexiones con PDO (PHP Data Objects).
BDD	MySQL	Tabla producto

3. Mediante el administrador de la base de datos MySQL (phpMyAdmin) creamos una nueva base de datos **productos** y una nueva tabla **producto**:





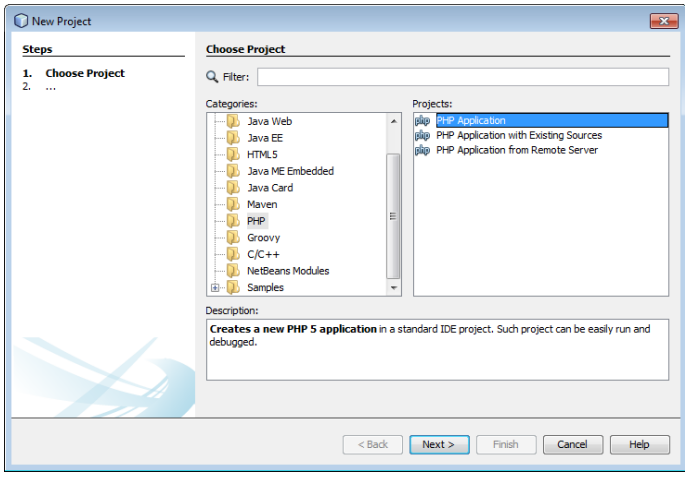
4. Luego creamos la tabla **producto**, se lo puede hacer mediante la consola SQL:

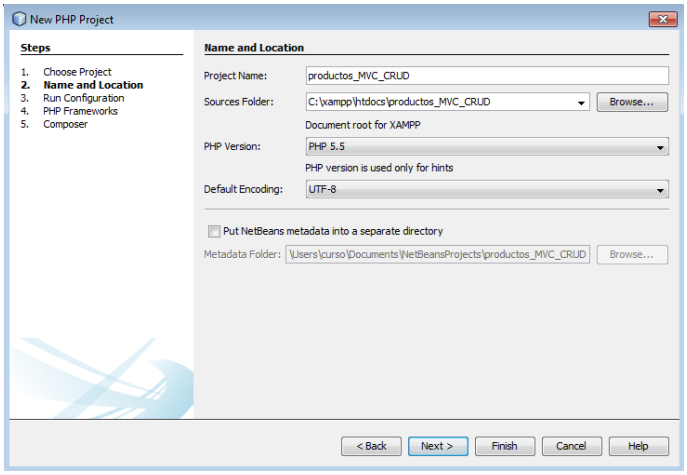


Copiamos las siguientes instrucciones SQL:

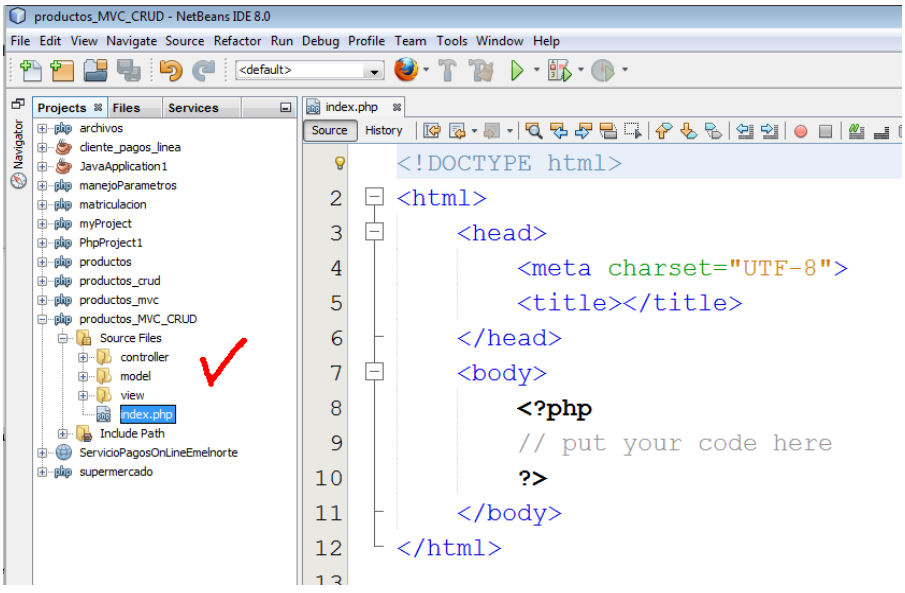
```
create table producto(
    codigo int primary key,
    nombre varchar(50) not null,
    precio float not null,
    cantidad int not null
    );

insert into producto(codigo,nombre,precio,cantidad) values(1,'computador',720.3,10);
insert into producto(codigo,nombre,precio,cantidad) values(2,'mouse usb',4.25,40);
insert into producto(codigo,nombre,precio,cantidad) values(3,'teclado',7.5,30);
```

5. Luego en el IDE Netbeans, creamos un nuevo proyecto PHP "**productos_MVC_CRUD**":





6. Para organizar los componentes de la aplicación, creamos las carpetas **model, view y controller**:

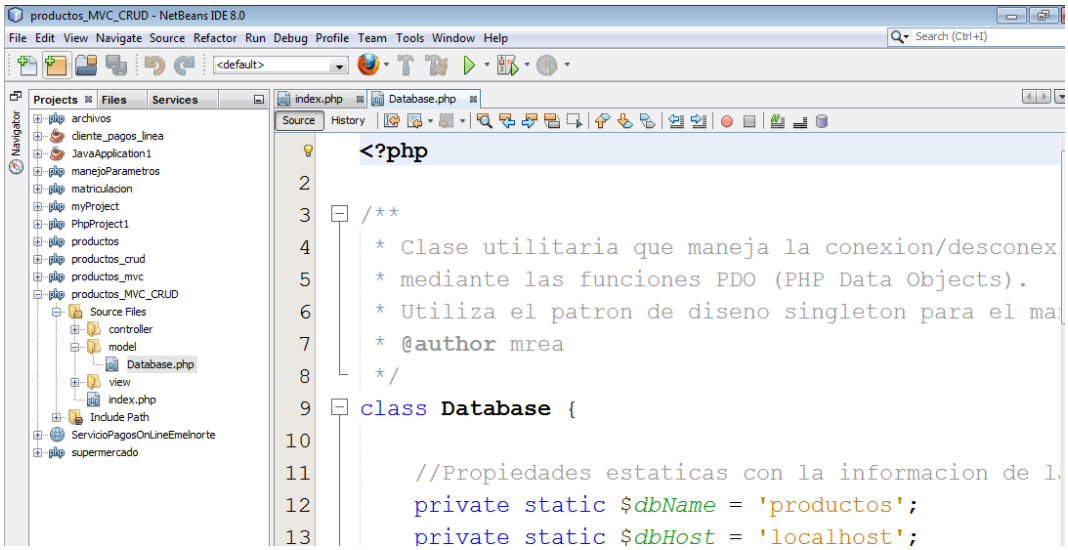


7. Iniciamos la implementación de la capa modelo. Creamos una nueva clase PHP llamada Database. Esta clase mantendrá los métodos necesarios para manejar la conexión a la base de datos. Para optimizar este recurso, utiliza el patrón de diseño singleton, que asegura que se mantenga una sola conexión en las transacciones. Crear una nueva clase PHP en el archivo Database.php y que deberá almacenarse en la carpeta **model**:

model/Database.php:

```php
<?php

/**
 * Clase utilitaria que maneja la conexion/desconexion a la base de datos
 * mediante las funciones PDO (PHP Data Objects).
 * Utiliza el patron de diseno singleton para el manejo de la conexion.
 * @author mrea
 */
class Database {

    //Propiedades estaticas con la informacion de la conexion (DSN):
    private static $dbName = 'productos';
    private static $dbHost = 'localhost';
    private static $dbUsername = 'root';
    private static $dbUserPassword = '';
    //Propiedad para control de la conexion:
    private static $conexion = null;

    /**
     * No se permite instanciar esta clase, se utilizan sus elementos
     * de tipo estatico.
     */
    public function __construct() {
        exit('No se permite instanciar esta clase.');
    }

    /**
     * Metodo estatico que crea una conexion a la base de datos.
     * @return type
     */
    public static function connect() {
        // Una sola conexion para toda la aplicacion (singleton):
        if (null == self::$conexion) {
            try {
                self::$conexion = new PDO("mysql:host=" . self::$dbHost . ";" . "dbname=" . self::
$dbName, self::$dbUsername, self::$dbUserPassword);
            } catch (PDOException $e) {
                die($e->getMessage());
            }
        }
        return self::$conexion;
    }

    /**
     * Metodo estatico para desconexion de la bdd.
     */
    public static function disconnect() {
        self::$conexion = null;
    }

}

?>
```

8. Nuestro proyecto debería verse así:



9. Dentro del mismo **model** creamos la clase **Producto** que representa a la tabla producto (objeto de dominio). Esta clase tendrá tantas propiedades como campos tenga la tabla:

model/Producto.php:

```php
<?php
/**
 * Entidad Producto. Representa a la tabla producto en la base de datos.
 *
 * @author curso
 */
class Producto {
    private $codigo;
    private $nombre;
    private $precio;
    private $cantidad;

    public function getCodigo() {
        return $this->codigo;
    }

    public function setCodigo($codigo) {
        $this->codigo = $codigo;
    }

    public function getNombre() {
        return $this->nombre;
    }

    public function getPrecio() {
        return $this->precio;
    }
    public function getCantidad() {
        return $this->cantidad;
    }
    public function setNombre($nombre) {
        $this->nombre = $nombre;
    }
    public function setPrecio($precio) {
        $this->precio = $precio;
    }
    public function setCantidad($cantidad) {
        $this->cantidad = $cantidad;
    }

}
```

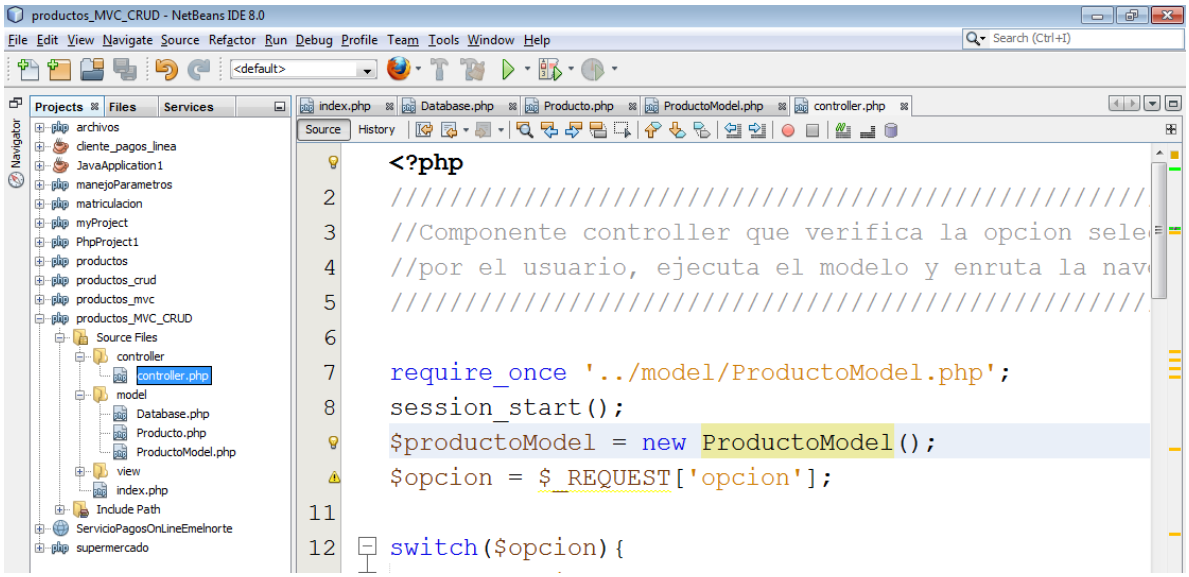
10. La lógica de la aplicación la implementamos en la clase **ProductoModel**. Esta clase se encarga de las funciones CRUD del sistema (Create,Read,Update,Delete) con la base de datos. Adicionalmente, se encarga de transformar los registros de la bdd a objetos de tipo "Producto" y viceversa. Esto permite que luego en la capa de la vista, se mantenga un modelo uniforme de información:

model/ProductoModel.php:

```php
<?php
    include 'Database.php';
    include 'Producto.php';

/**
 * Componente model para el manejo de productos.
 *
 * @author mrea
 */
class ProductoModel {
    /**
     * Obtiene todos los productos de la base de datos.
     * @return array
     */
    public function getProductos(){
        //obtenemos la informacion de la bdd:
        $pdo = Database::connect();
        $sql = "select * from producto order by nombre";
        $resultado = $pdo->query($sql);
        //transformamos los registros en objetos de tipo Producto:
        $listado = array();
        foreach ($resultado as $res){
            $producto = new Producto();
            $producto->setCodigo($res['codigo']);
            $producto->setNombre($res['nombre']);
            $producto->setPrecio($res['precio']);
            $producto->setCantidad($res['cantidad']);
            array_push($listado, $producto);
        }
        Database::disconnect();
        //retornamos el listado resultante:
        return $listado;
    }
    /**
     * Obtiene un producto especifico.
     * @param type $codigo El codigo del producto a buscar.
     * @return \Producto
     */
    public function getProducto($codigo){
        //Obtenemos la informacion del producto especifico:
        $pdo = Database::connect();
        //Utilizamos parametros para la consulta:
        $sql = "select * from producto where codigo=?";
        $consulta = $pdo->prepare($sql);
        //Ejecutamos y pasamos los parametros para la consulta:
        $consulta->execute(array($codigo));
        //Extraemos el registro especifico:
        $dato = $consulta->fetch(PDO::FETCH_ASSOC);
        //Transformamos el registro obtenido a objeto:
        $producto=new Producto();
        $producto->setCodigo($dato['codigo']);
        $producto->setNombre($dato['nombre']);
        $producto->setPrecio($dato['precio']);
        $producto->setCantidad($dato['cantidad']);
        Database::disconnect();
        return $producto;
    }
    /**
     * Crea un nuevo producto en la base de datos.
     * @param type $codigo
     * @param type $nombre
     * @param type $precio
     * @param type $cantidad
     */
```

```
    public function crearProducto($codigo,$nombre,$precio,$cantidad){
        //Preparamos la conexion a la bdd:
        $pdo=Database::connect();
        $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
        //Preparamos la sentencia con parametros:
        $sql="insert into producto (codigo,nombre,precio,cantidad) values(?,?,?,?)";
        $consulta=$pdo->prepare($sql);
        //Ejecutamos y pasamos los parametros:
        $consulta->execute(array($codigo,$nombre,$precio,$cantidad));
        Database::disconnect();
    }
    /**
     * Elimina un producto especifico de la bdd.
     * @param type $codigo
     */
    public function eliminarProducto($codigo){
        //Preparamos la conexion a la bdd:
        $pdo=Database::connect();
        $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
        $sql="delete from producto where codigo=?";
        $consulta=$pdo->prepare($sql);
        //Ejecutamos la sentencia incluyendo a los parametros:
        $consulta->execute(array($codigo));
        Database::disconnect();
    }
    /**
     * Actualiza un producto existente.
     * @param type $codigo
     * @param type $nombre
     * @param type $precio
     * @param type $cantidad
     */
    public function actualizarProducto($codigo,$nombre,$precio,$cantidad){
        //Preparamos la conexión a la bdd:
        $pdo=Database::connect();
        $sql="update producto set nombre=?,precio=?,cantidad=? where codigo=?";
        $consulta=$pdo->prepare($sql);
        //Ejecutamos la sentencia incluyendo a los parametros:
        $consulta->execute(array($nombre,$precio,$cantidad,$codigo));
        Database::disconnect();
    }
}
```

11. La siguiente es la capa de controladores. El controlador tiene la función de administrar la navegación entre páginas, al mismo tiempo que invoca a la capa de lógica (modelo). Mantiene la información disponible para la capa de la vista mediante el uso de sesiones (**session**). Implementamos el componente controller.php en la carpeta controller:



controller/controller.php:

```php
<?php
////////////////////////////////////////////////////////////////////////
//Componente controller que verifica la opcion seleccionada
//por el usuario, ejecuta el modelo y enruta la navegacion de paginas.
////////////////////////////////////////////////////////////////////////

require_once '../model/ProductoModel.php';
session_start();
$productoModel = new ProductoModel();
$opcion = $_REQUEST['opcion'];

switch($opcion){
    case "listar":
        //obtenemos la lista de productos:
        $listado = $productoModel->getProductos();
        //y los guardamos en sesion:
        $_SESSION['listado'] = serialize($listado);
        header('Location: ../index.php');
        break;
    case "crear":
        //navegamos a la pagina de creacion:
        header('Location: ../view/crear.php');
        break;
    case "guardar":
        //obtenemos los valores ingresados por el usuario en el formulario:
        $codigo=$_REQUEST['codigo'];
        $nombre=$_REQUEST['nombre'];
        $precio=$_REQUEST['precio'];
        $cantidad=$_REQUEST['cantidad'];
        //creamos un nuevo producto:
        $productoModel->crearProducto($codigo, $nombre, $precio, $cantidad);
        //actualizamos la lista de productos para grabar en sesion:
        $listado = $productoModel->getProductos();
        $_SESSION['listado'] = serialize($listado);
        header('Location: ../index.php');
        break;
    case "eliminar":
        //obtenemos el codigo del producto a eliminar:
        $codigo=$_REQUEST['codigo'];
        //eliminamos el producto:
        $productoModel->eliminarProducto($codigo);
        //actualizamos la lista de productos para grabar en sesion:
        $listado = $productoModel->getProductos();
        $_SESSION['listado'] = serialize($listado);
        header('Location: ../index.php');
        break;
    case "cargar":
        //para permitirle actualizar un producto al usuario primero
        //obtenemos los datos completos de ese producto:
        $codigo=$_REQUEST['codigo'];
        $producto=$productoModel->getProducto($codigo);
        //guardamos en sesion el producto para posteriormente visualizarlo
        //en un formulario para permitirle al usuario editar los valores:
        $_SESSION['producto']=$producto;
        header('Location: ../view/actualizar.php');
        break;
    case "actualizar":
        //obtenemos los datos modificados por el usuario:
        $codigo=$_REQUEST['codigo'];
        $nombre=$_REQUEST['nombre'];
        $precio=$_REQUEST['precio'];
        $cantidad=$_REQUEST['cantidad'];
        //actualizamos los datos del producto:
        $productoModel->actualizarProducto($codigo, $nombre, $precio, $cantidad);
        //actualizamos la lista de productos para grabar en sesion:
        $listado = $productoModel->getProductos();
        $_SESSION['listado'] = serialize($listado);
        header('Location: ../index.php');
        break;
    default:
        //si no existe la opcion recibida por el controlador, siempre
        //redirigimos la navegacion a la pagina index:
        header('Location: ../index.php');
}
```

12. Finalmente la capa de la vista está compuesta de las páginas **index.php**, **view/crear.php** y **view/actualizar.php**. Estas se encargan de implementar la visualización de datos, botones y enlaces (links) de opciones y los formularios donde el usuario pueda ingresar o actualizar información.
    La manera en que estas páginas interactúan con el controlador es mediante controles de tipo **hidden** que indican la opción seleccionada por el usuario (en los formularios) o mediante enlaces generados dinámicamente (<**a href**>) que permiten que un usuario seleccione un producto determinado para eliminarlo o actualizarlo.
    A continuación el código de las tres páginas:

index.php:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>CRUD Productos</title>
    </head>
    <body>
        <table>
            <tr><td>
                    <form action="controller/controller.php">
                        <input type="hidden" value="listar" name="opcion">
                        <input type="submit" value="Consultar listado">
                    </form>
                </td><td>
                    <form action="controller/controller.php">
                        <input type="hidden" value="crear" name="opcion">
                        <input type="submit" value="Crear producto">
                    </form>
                </td></tr>
        </table>
        <table border="1">
            <tr>
                <th>CODIGO</th>
                <th>NOMBRE</th>
                <th>PRECIO</th>
                <th>CANTIDAD</th>
                <th>ELIMINAR</th>
                <th>ACTUALIZAR</th>
            </tr>
            <?php
            session_start();
            include './model/Producto.php';
            //verificamos si existe en sesion el listado de productos:
            if (isset($_SESSION['listado'])) {
                $listado = unserialize($_SESSION['listado']);
                foreach ($listado as $prod) {
                    echo "<tr>";
                    echo "<td>" . $prod->getCodigo() . "</td>";
                    echo "<td>" . $prod->getNombre() . "</td>";
                    echo "<td>" . $prod->getPrecio() . "</td>";
                    echo "<td>" . $prod->getCantidad() . "</td>";
                    //opciones para invocar al controlador indicando la opcion eliminar o cargar
                    //y la fila que selecciono el usuario (con el codigo del producto):
                    echo "<td><a href='controller/controller.php?opcion=eliminar&codigo=" . $prod-
>getCodigo() . "'>eliminar</a></td>";
                    echo "<td><a href='controller/controller.php?opcion=cargar&codigo=" . $prod-
>getCodigo() . "'>actualizar</a></td>";
                    echo "</tr>";
                }
            } else{
                echo "No se han cargado datos.";
            }
            ?>
        </table>
</body>
</html>
```

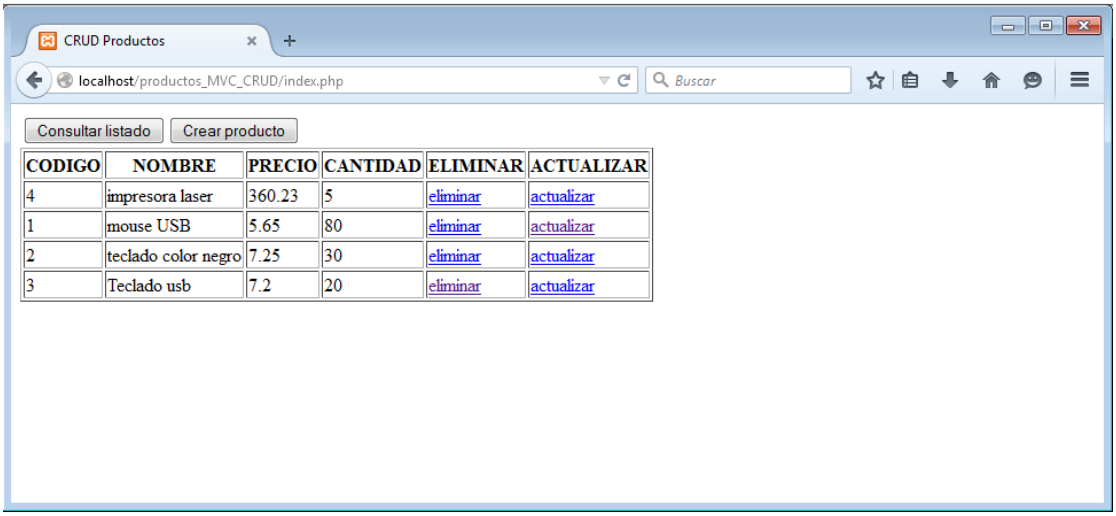
view/crear.php:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Crear producto</title>
    </head>
    <body>
        <form action="../controller/controller.php">
            <input type="hidden" value="guardar" name="opcion">
            Codigo:<input type="text" name="codigo"><br>
            Nombre:<input type="text" name="nombre"><br>
            Precio:<input type="text" name="precio"><br>
            Cantidad:<input type="text" name="cantidad"><br>
            <input type="submit" value="Crear">
        </form>
    </body>
</html>
```

view/actualizar.php:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Actualizar producto</title>
    </head>
    <body>
        <?php
        include '../model/Producto.php';
        //obtenemos los datos de sesion:
        session_start();
        $producto = $_SESSION['producto'];
        ?>
        <form action="../controller/controller.php">
            <input type="hidden" value="actualizar" name="opcion">
            <!-- Utilizamos pequeños scripts PHP para obtener los valores del producto: -->
            <input type="hidden" value="<?php echo $producto->getCodigo(); ?>" name="codigo">
            Codigo:<b><?php echo $producto->getCodigo(); ?></b><br>
            Nombre:<input type="text" name="nombre" value="<?php echo $producto->getNombre(); ?>"><br>
            Precio:<input type="text" name="precio" value="<?php echo $producto->getPrecio(); ?>"><br>
            Cantidad:<input type="text" name="cantidad" value="<?php echo $producto->getCantidad(); ?>"><br>
            <input type="submit" value="Actualizar">
        </form>
    </body>
</html>
```

13. Hasta aquí la aplicación básica final:



NOTA: A partir de este punto, adicionaremos varias funcionalidades a la aplicación básica con el afán de relacionarnos mejor con la arquitectura y mostrar su flexibilidad.

14. **Requerimiento**: mostrar en la página inicial la suma total de los valores de productos.
    a. Primero adicionamos el método **getValorProductos()** a la clase ProductoModel:

model/ProductoModel.php:

```php
<?php
    include 'Database.php';
    include 'Producto.php';

/**
 * Componente model para el manejo de productos.
 *
 * @author mrea
 */
class ProductoModel {
    /**
     * Calcula la suma total de valores de productos.
     * @return type
     */
    public function getValorProductos(){
        $listado=$this->getProductos();
        $suma=0;
        foreach ($listado as $prod) {
            $suma+=$prod->getPrecio()*$prod->getCantidad();
        }
        return $suma;
    }
    /**
     * Obtiene todos los productos de la base de datos.
     * @return array
     */
    public function getProductos(){
        //obtenemos la informacion de la bdd:
        $pdo = Database::connect();
        $sql = "select * from producto order by nombre";
        $resultado = $pdo->query($sql);
        //transformamos los registros en objetos de tipo Producto:
        $listado = array();
        foreach ($resultado as $res){
            $producto = new Producto();
            $producto->setCodigo($res['codigo']);
            $producto->setNombre($res['nombre']);
            $producto->setPrecio($res['precio']);
            $producto->setCantidad($res['cantidad']);
            array_push($listado, $producto);
        }
        Database::disconnect();
        //retornamos el listado resultante:
        return $listado;
    }

...
```

b. Luego modificamos en el controlador la opción “listar”:

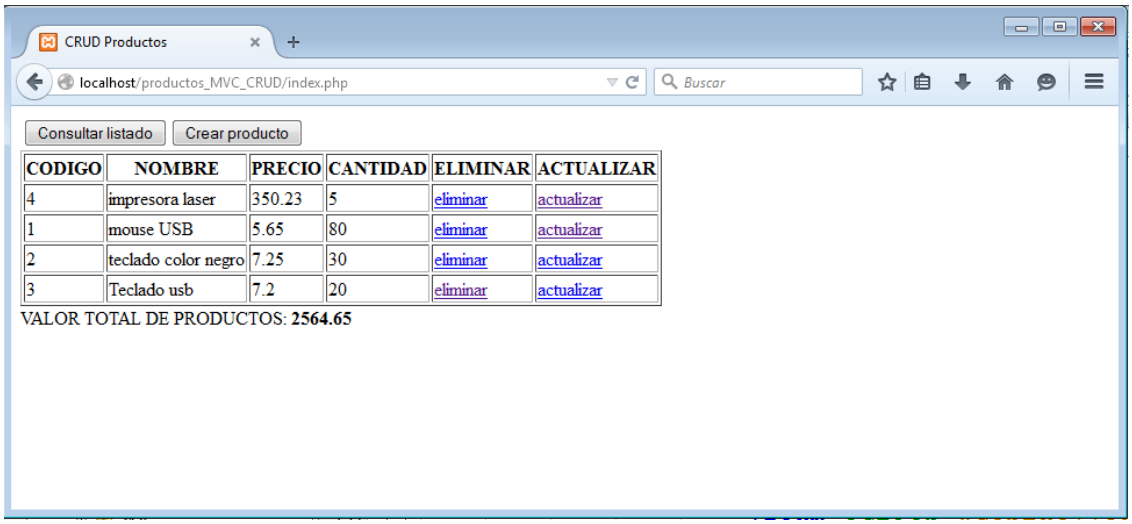
controller/controller.php:

```php
<?php
//////////////////////////////////////////////////////////////////////
//Componente controller que verifica la opcion seleccionada
//por el usuario, ejecuta el modelo y enruta la navegacion de paginas.
//////////////////////////////////////////////////////////////////////

require_once '../model/ProductoModel.php';
session_start();
$productoModel = new ProductoModel();
$opcion = $_REQUEST['opcion'];

switch($opcion){
    case "listar":
        //obtenemos la lista de productos:
        $listado = $productoModel->getProductos();
        //y los guardamos en sesion:
        $_SESSION['listado'] = serialize($listado);
        //obtenemos el valor total de productos y guardamos en sesion:
        $_SESSION['valorTotal']=$productoModel->getValorProductos();
        header('Location: ../index.php');
        break;
    case "crear":
        //navegamos a la pagina de creacion:
        header('Location: ../view/crear.php');
        break;
...
```

c. Finalmente mostramos el resultado en la página index y cada vez que el usuario pulse el botón “consultar listado” entonces se consultará la suma de valores:

index.php:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>CRUD Productos</title>
    </head>
    <body>
        <table>
            <tr><td>
                    <form action="controller/controller.php">
                        <input type="hidden" value="listar" name="opcion">
                        <input type="submit" value="Consultar listado">
                    </form>
                </td><td>
                    <form action="controller/controller.php">
                        <input type="hidden" value="crear" name="opcion">
                        <input type="submit" value="Crear producto">
                    </form>
                </td></tr>
        </table>
        <table border="1">
            <tr>
                <th>CODIGO</th>
                <th>NOMBRE</th>
                <th>PRECIO</th>
                <th>CANTIDAD</th>
                <th>ELIMINAR</th>
                <th>ACTUALIZAR</th>
            </tr>
            <?php
            session_start();
            include './model/Producto.php';
            //verificamos si existe en sesion el listado de productos:
            if (isset($_SESSION['listado'])) {
                $listado = unserialize($_SESSION['listado']);
                foreach ($listado as $prod) {
                    echo "<tr>";
                    echo "<td>" . $prod->getCodigo() . "</td>";
                    echo "<td>" . $prod->getNombre() . "</td>";
                    echo "<td>" . $prod->getPrecio() . "</td>";
                    echo "<td>" . $prod->getCantidad() . "</td>";
                    //opciones para invocar al controlador indicando la opcion eliminar o cargar
                    //y la fila que selecciono el usuario (con el codigo del producto):
                    echo "<td><a href='controller/controller.php?opcion=eliminar&codigo=" . $prod-
>getCodigo() . "'>eliminar</a></td>";
                    echo "<td><a href='controller/controller.php?opcion=cargar&codigo=" . $prod-
>getCodigo() . "'>actualizar</a></td>";
                    echo "</tr>";
                }
            } else{
                echo "No se han cargado datos.";
            }
            ?>
        </table>
        <?php
        if(isset($_SESSION['valorTotal'])){
            echo "VALOR TOTAL DE PRODUCTOS: <b>".$_SESSION['valorTotal']."</b>";
        }
        ?>
</body>
</html>
```



15. **Requerimiento**: Permitir al usuario ordenar el listado de productos por nombre en forma descendente.
    a. Modificamos el metodo ProductoModel::getListado() de manera que reciba como parametro un indicador de ascendente o descendente:

model/ProductoModel.php:

```
    ...

    /**
     * Obtiene todos los productos de la base de datos.
     * @return array
     */
    public function getProductos($orden){
        //obtenemos la informacion de la bdd:
        $pdo = Database::connect();
        //verificamos el ordenamiento asc o desc:
        if($orden==true)//asc
            $sql = "select * from producto order by nombre";
        else //desc
            $sql = "select * from producto order by nombre desc";
        $resultado = $pdo->query($sql);
        //transformamos los registros en objetos de tipo Producto:
        $listado = array();
        foreach ($resultado as $res){
            $producto = new Producto();
            $producto->setCodigo($res['codigo']);
            $producto->setNombre($res['nombre']);
            $producto->setPrecio($res['precio']);
            $producto->setCantidad($res['cantidad']);
            array_push($listado, $producto);
        }
        Database::disconnect();
        //retornamos el listado resultante:
        return $listado;
    }

    ...
```

b. Modificamos el controlador y adicionamos una nueva opción “listar_desc”:

controller/controller.php:

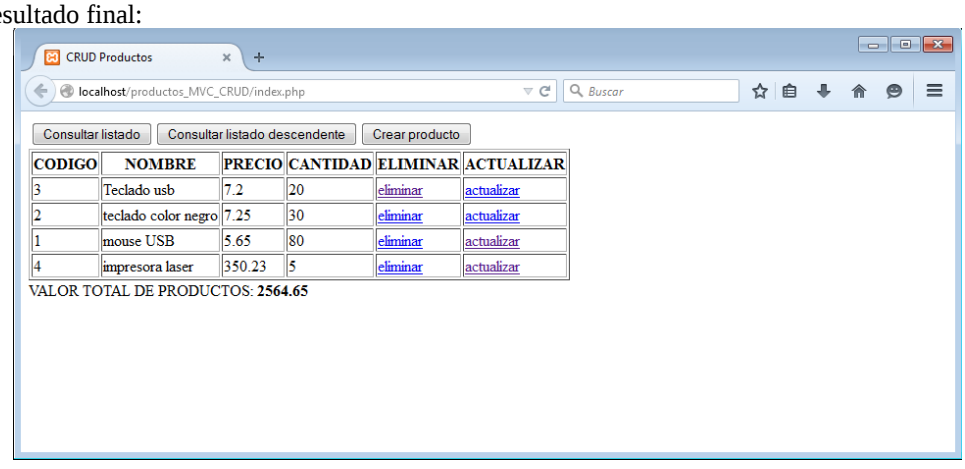
```
...
switch($opcion){
    case "listar":
        //obtenemos la lista de productos:
       $listado = $productoModel->getProductos(true);
       //y los guardamos en sesion:
       $_SESSION['listado'] = serialize($listado);
       //obtenemos el valor total de productos:
       $_SESSION['valorTotal']=$productoModel->getValorProductos();
       header('Location: ../index.php');
        break;
    case "listar_desc":
        //obtenemos la lista de productos:
        $listado = $productoModel->getProductos(false);
        //y los guardamos en sesion:
        $_SESSION['listado'] = serialize($listado);
        //obtenemos el valor total de productos:
        $_SESSION['valorTotal']=$productoModel->getValorProductos();
        header('Location: ../index.php');
        break;
...
```

c. Finalmente adicionamos un botón en la página index para invocar al controlador:

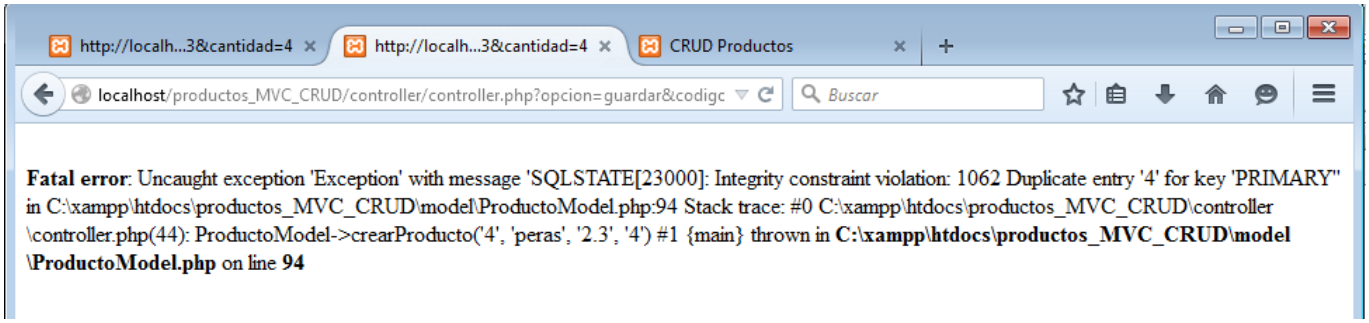
index.php:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
       <title>CRUD Productos</title>
    </head>
    <body>
        <table>
            <tr><td>
                    <form action="controller/controller.php">
                        <input type="hidden" value="listar" name="opcion">
                        <input type="submit" value="Consultar listado">
                    </form>
                </td>
                <td>
                    <form action="controller/controller.php">
                        <input type="hidden" value="listar_desc" name="opcion">
                        <input type="submit" value="Consultar listado descendente">
                    </form>
                </td><td>
                    <form action="controller/controller.php">
                        <input type="hidden" value="crear" name="opcion">
                        <input type="submit" value="Crear producto">
                    </form>
                </td></tr>
        </table>
          ...
```

d. El resultado final:



16. **Requerimiento**: controlar en la aplicación de manera que cuando se intente crear un producto con un código ya existente, presente un mensaje de error en la página principal.
    a. Inicialmente el lugar donde generará error al momento de intentar insertar un código duplicado es en el ProductoModel. En nuestro ejemplo si creamos un nuevo producto con código 4 (que ya existe) dará el siguiente error y la aplicación cae:



    b. Empezamos por adicionar try/catch en el método ProductoModel::crearProducto(). De esta manera logramos atrapar el error y enviar la excepción hacia la capa superior, en este caso al controlador:

model/ProductoModel.php:

```
...
    public function crearProducto($codigo,$nombre,$precio,$cantidad){
        //Preparamos la conexion a la bdd:
        $pdo=Database::connect();
        $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
        //Preparamos la sentencia con parametros:
        $sql="insert into producto (codigo,nombre,precio,cantidad) values(?,?,?,?)";
        $consulta=$pdo->prepare($sql);
        //Ejecutamos y pasamos los parametros:
        try{
            $consulta->execute(array($codigo,$nombre,$precio,$cantidad));
        }  catch (PDOException $e){
            Database::disconnect();
            throw new Exception($e->getMessage());
        }
        Database::disconnect();
    }
...
```

    c. Modificamos el controlador para manejar los mensajes generados por el sistema y para atrapar posibles errores al momento de guardar un nuevo producto:

controller/controller.php:

```
<?php
//////////////////////////////////////////////////////////////////////////
//Componente controller que verifica la opcion seleccionada
//por el usuario, ejecuta el modelo y enruta la navegacion de paginas.
//////////////////////////////////////////////////////////////////////////

require_once '../model/ProductoModel.php';
session_start();
$productoModel = new ProductoModel();
$opcion = $_REQUEST['opcion'];
//limpiamos cualquier mensaje previo:
unset($_SESSION['mensaje']);

...

    case "guardar":
        //obtenemos los valores ingresados por el usuario en el formulario:
        $codigo=$_REQUEST['codigo'];
        $nombre=$_REQUEST['nombre'];
        $precio=$_REQUEST['precio'];
        $cantidad=$_REQUEST['cantidad'];
```

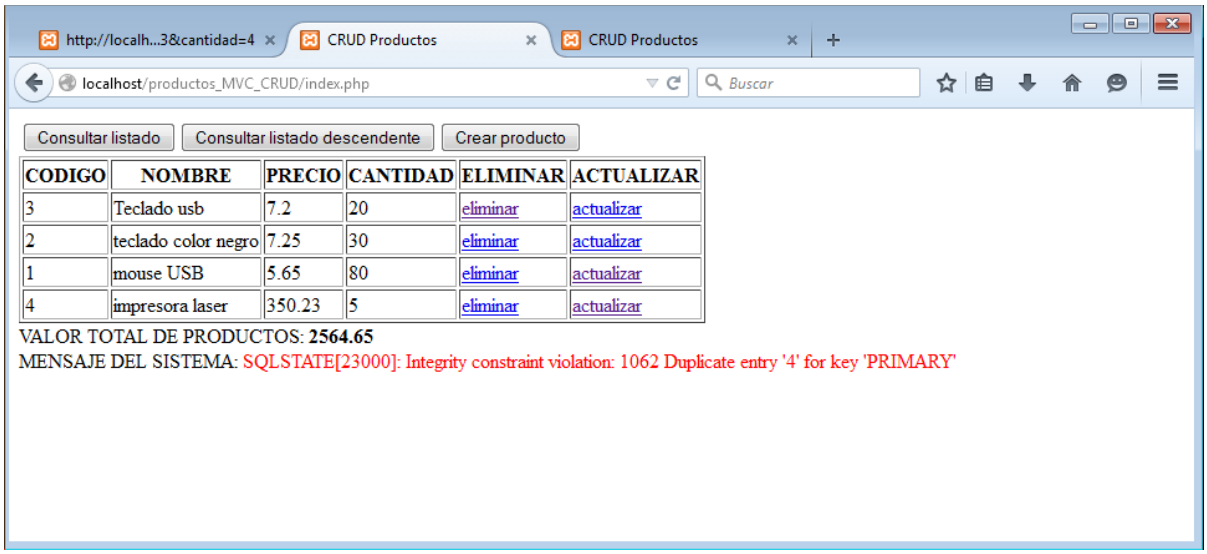
```
        //creamos un nuevo producto:
        try{
            $productoModel->crearProducto($codigo, $nombre, $precio, $cantidad);
        }catch(Exception $e){
            //colocamos el mensaje de la excepcion en sesion:
            $_SESSION['mensaje']=$e->getMessage();
        }
        //actualizamos la lista de productos para grabar en sesion:
        $listado = $productoModel->getProductos();
        $_SESSION['listado'] = serialize($listado);
        header('Location: ../index.php');
        break;
...
```

d. Finalmente adicionamos la presentación de mensajes en la parte final de la página index:

index.php:

```
...
        <?php
        if (isset($_SESSION['valorTotal'])) {
            echo "VALOR TOTAL DE PRODUCTOS: <b>" . $_SESSION['valorTotal'] . "</b>";
        }
        if (isset($_SESSION['mensaje'])) {
            echo "<br>MENSAJE DEL SISTEMA: <font color='red'>" . $_SESSION['mensaje'] . "</font><br>";
        }
        ?>
    </body>
</html>
```

e. El resultado final al intentar duplicar un código de producto:



17. EJERCICIO A:
    a. Crear una nueva base de datos llamada biblioteca, la cual debe contener la tabla libro. De acuerdo al modelo explicado en este tutorial, crear una aplicación CRUD que permita manejar la información de libros. El bosquejo de la tabla libro es:

CODIGO	TITULO	AÑO	AUTOR	PAGINAS
1005	Programando con PHP – básico.	2010	John Carter	310
0401	Desarrolle aplicaciones MVC en 15 días.	2013	Hannibal Lecter	250
0404	PHP for dummies.	2011	Stephen King	18

18. EJERCICIO B:
    a. Crear una nueva base de datos llamada academico, la cual debe contener la tabla notas. De acuerdo al modelo explicado en este tutorial, crear una aplicación CRUD que permita manejar la información de notas de estudiantes. El bosquejo de la tabla notas es:

CEDULA (clave primaria)	NOMBRES	NOTA1	NOTA2	PROMEDIO
1005487659	ANDRADE LUIS	7	8	7.5
0401234567	RUIZ ANA	9	9	9
0404578452	ZAPATA JORGE	7	7	7

    b. El promedio debe ser calculado automáticamente por el sistema.
    c. El sistema debe permitir ordenar por nombres de manera ascendente o descendente.
    d. Presentar en la página inicial (index) el promedio general de todos los alumnos.
    e. No se debe permitir ingresar dos veces al mismo estudiante. Presentar un mensaje de error si es el caso.
    f. El sistema no debe permitir el ingreso de notas con valores negativos. Presentar un mensaje de error si es el caso.
    g. Adicionar una opción (botón) en la página principal con la que el usuario pueda actualizar todas las notas a 0 (cero) previo a la confirmación del usuario (el sistema debe preguntar SI/NO).
    h. Tanto en la página de ingreso de nuevas notas, como en la de actualización, adicionar un botón que permita cancelar la operación y dirigirse a la página principal.

INDICACIONES:

- Cada ejercicio resolverlo en un proyecto individual y empaquetarlo en un archivo **crud_A_apellido_nombre.zip** y **crud_B_apellido_nombre.zip**.
- Enviarlos por correo hasta el día viernes 5/mayo/2015 al correo mrea@utn.edu.ec hasta las 17h00.
- Suerte :)