HITO 2 DEL 2º TRIMESTRE DE PROGRAMACIÓN

Alejandro Cortés Díaz

CampusFP

Diseño de aplicaciones multiplataforma – PROGRAMACIÓN  14/02/2025

Índice

[Presentación de los archivos y sus funcionalidades 2](#_Toc190574939)

[Explicación de las funciones del código 32](#_Toc190574940)

[**Enlace a GitHub** 37](#_Toc190574941)

[**Bibliografía** 38](#_Toc190574942)

# Presentación de los archivos y sus funcionalidades

**conexion.php**:

* Este archivo se encarga de la conexión a la base de datos, rellenando los campos necesarios para conectar eficazmente, procediendo con un constructor, y posteriormente cerrando la conexión.
* <?php
* //Class conexion para la conexion a la base de datos con la información requerida para que se produzca
* class Conexion {
* private $servidor = 'localhost';
* private $usuario = 'root';
* private $password = 'curso';
* private $base\_datos = 'hito\_2\_2ºTrimestre\_Alejandro\_Cortes\_Diaz';
* public $conexion;
* // Constructor de la clase para crear una nueva conexión a la base de datos
* public function \_\_construct() {
* $this->conexion = new mysqli($this->servidor, $this->usuario, $this->password, $this->base\_datos);
* // Verificar si hay errores en la conexión
* if ($this->conexion->connect\_error) {
* die("Error de conexión: " . $this->conexion->connect\_error); // Mostrar mensaje de error y detener la ejecución
* }
* }
* // Método para cerrar la conexión a la base de datos
* public function cerrar() {
* $this->conexion->close();
* }
* }
* ?>

**TareasController.php**:

* Aquí se realiza el llamado, por medio de una clase TareasController, a todas las funciones que utilizaremos para el CRUD (AgregarTarea, ListarTareasPorEmail, ActualizarTarea, y EliminarTarea). Se realiza el llamado con los parámetros necesarios rellenos con los datos para proceder con las funciones.
* <?php
* //Referencio a la clase Tarea para usar sus metodos
* require\_once '../modelo/class\_tareas.php';
* //Clase controlador para las tareas
* class TareasController {
* private $modelo;
* // Constructor de la clase
* public function \_\_construct() {
* $this->modelo = new Tarea(); // Crea una nueva instancia del modelo de tareas
* }
* // Método para agregar una nueva tarea
* public function AgregarTarea($email, $titulo, $descripcion, $estado) {
* $this->modelo->AgregarTarea($email, $titulo, $descripcion, $estado);
* }
* // Método para listar todas las tareas
* public function ListarTareas() {
* return $this->modelo->ObtenerTareas();
* }
* // Método para eliminar una tarea por un id específico
* public function EliminarTarea($id\_tarea) {
* $this->modelo->EliminarTarea($id\_tarea);
* }
* // Método para actualizar el estado de una tarea pickeando el id de la tarea y una vez encontrada, modificar el estado con función php
* public function actualizarEstado($id\_tarea, $estado) {
* $this->modelo->ActualizarEstado($id\_tarea, $estado);
* }
* // Método para listar tareas por el correo electrónico único de cada usuario
* public function ListarTareasPorEmail($email) {
* return $this->modelo->ObtenerTareasPorEmail($email);
* }
* }
* ?>

**UsuariosController.php**:

* Similar a TareasController.php, este archivo contiene la clase UsuariosController con funciones para manejar usuarios (AgregarUsuario, ListarUsuarios, ObtenerUsuarioPorEmail, y EliminarUsuario).
* <?php
* //Referencio a la clase Usuario para usar sus metodos como en el archivo anterior
* require\_once '../modelo/class\_usuario.php';
* class UsuariosController {
* private $modelo;
* // Constructor de la clase
* public function \_\_construct() {
* $this->modelo = new Usuario(); // Crea una nueva instancia del modelo
* }
* // Método para agregar un nuevo usuario
* public function AgregarUsuario($email, $nombre, $password) {
* $this->modelo->AgregarUsuario($email, $nombre, $password);
* }
* // Método para listar todos los usuarios
* public function ListarUsuarios() {
* return $this->modelo->ObtenerUsuarios();
* }
* // Método para obtener un usuario por su correo electrónico único
* public function ObtenerUsuarioPorEmail($email) {
* return $this->modelo->ObtenerUsuarioPorEmail($email);
* }
* // Método para verificar las credenciales de un usuario al iniciar sesión buscando la referencia en la base de datos
* public function VerificarUsuario($email, $password) {
* return $this->modelo->VerificarUsuario($email, $password);
* }
* }
* ?>

**Class\_Tareas.php**:

* En este archivo, se contienen todas las funciones necesarias que se llaman en TareasController.php. Aquí, están definidas y creadas. Todo esto dentro de una clase Tarea, que comienza con una conexión nueva.
* <?php
* //Referencio al archivo de la conexión a la db puesto que será necesaria la conexión para usar las funciones con sintaxis SQL
* require\_once '../config/conexion.php';
* // Clase Tarea para gestionar la tabla de tareas
* class Tarea {
* private $conexion;
* public function \_\_construct() {
* $this->conexion = new Conexion();
* }
* // Método para agregar una nueva tarea
* public function AgregarTarea($email, $titulo, $descripcion, $estado) {
* // Sintaxis SQL para agregar una nueva tarea con el insert into
* $query = "INSERT INTO tarea (email, titulo, descripcion, estado) VALUES (?, ?, ?, ?)";
* // Preparar la query a traves de la conexión a la base de datos
* $stmt = $this->conexion->conexion->prepare($query);
* // Asignar los valores a los parámetros de la query
* $stmt->bind\_param("ssss", $email, $titulo, $descripcion, $estado);
* //Si la query se ejecuta correctamente, mostrar mensaje de éxito
* if ($stmt->execute()) {
* echo "Tarea agregada con éxito.";
* } else {
* //Sino, mostrar mensaje de error
* error\_log("Error al agregar tarea: " . $stmt->error);
* echo "Error al agregar tarea: " . $stmt->error;
* }
* //Cierro conexión
* $stmt->close();
* }
* // Método para obtener todas las tareas
* public function ObtenerTareas() {
* // Sintaxis SQL para seleccionar todas las tareas con el select from
* $query = "SELECT \* FROM tarea";
* // Ejecutar la query y lo guardo en la variable resultado
* $resultado = $this->conexion->conexion->query($query);
* // Creo un array vacío para guardar las tareas
* $tareas = [];
* // Si hay resultados, guardo cada fila con las tareas en el array a través del fetch
* if ($resultado) {
* while ($fila = $resultado->fetch\_assoc()) {
* $tareas[] = $fila;
* }
* // Sino, muestro mensaje de error
* } else {
* error\_log("Error al obtener tareas: " . $this->conexion->conexion->error);
* }
* //Devuelvo el contenido del array
* return $tareas;
* }
* // Método para obtener una tarea por correo electrónico
* public function ObtenerTareaPorEmail($email) {
* // Sintaxis SQL para seleccionar la tarea por correo electrónico
* $query = "SELECT \* FROM tarea WHERE email = ?";
* //Preparo la query
* $stmt = $this->conexion->conexion->prepare($query);
* //Asigno el valor al parámetro de la query
* $stmt->bind\_param("s", $email);
* //Ejecuto la query
* $stmt->execute();
* //Guardo el resultado en la variable resultado
* $resultado = $stmt->get\_result();
* //Guardo la tarea obtenida a través de la sentencia fetch, en la variable tarea
* $tarea = $resultado->fetch\_assoc();
* //Si no hay tarea, muestro mensaje de error
* if (!$tarea) {
* error\_log("Error al obtener tarea por email: " . $stmt->error);
* }
* //Cierro conexión
* $stmt->close();
* return $tarea;
* }
* // Método para eliminar una tarea por su id
* public function EliminarTarea($id\_tarea) {
* // Sintaxis SQL para eliminar la tarea tarea con el delete por el id de la tarea
* $query = "DELETE FROM tarea WHERE id\_tarea = ?";
* //Preparo la query
* $stmt = $this->conexion->conexion->prepare($query);
* //Asigno el valor al parámetro
* $stmt->bind\_param("i", $id\_tarea);
* //Si la query se ejecuta correctamente, mostrar mensaje de éxito
* if ($stmt->execute()) {
* echo "Tarea eliminada con éxito.";
* } else {
* //Sino, mostrar mensaje de error
* error\_log("Error al eliminar tarea: " . $stmt->error); // Registro de errores
* echo "Error al eliminar tarea: " . $stmt->error;
* }
* //Cierro conexión
* $stmt->close();
* }
* // Método para actualizar el estado de una tarea por el id de la misma
* public function ActualizarEstado($id\_tarea, $estado) {
* // Sintaxis SQL para modificar la tarea tarea con el update por el id de la tarea
* $query = "UPDATE tarea SET estado = ? WHERE id\_tarea = ?";
* //Preparo la query
* $stmt = $this->conexion->conexion->prepare($query);
* //Asigno los valores
* $stmt->bind\_param("si", $estado, $id\_tarea);
* //Si la query se ejecuta correctamente, mostrar mensaje de éxito
* if ($stmt->execute()) {
* echo "Estado actualizado con éxito.";
* } else {
* //Sino, mostrar mensaje de error
* error\_log("Error al actualizar estado: " . $stmt->error); // Registro de errores
* echo "Error al actualizar estado: " . $stmt->error;
* }
* //Cierro conexión
* $stmt->close();
* }
* // Método para obtener todas las tareas por correo electrónico único
* public function ObtenerTareasPorEmail($email) {
* // Sintaxis SQL para seleccionar todas las tareas por correo electrónico
* $query = "SELECT \* FROM tarea WHERE email = ?";
* //Preparo la query
* $stmt = $this->conexion->conexion->prepare($query);
* //Asigno el valor al parámetro
* $stmt->bind\_param("s", $email);
* //Ejecuto la query
* $stmt->execute();
* //Guardo el resultado en la variable result
* $result = $stmt->get\_result();
* //Guardo las tareas obtenidas a través de la sentencia fetch, en la variable tareas "MYSQLI\_ASSOC" obtiene todos los datos como un array de arrays asociativos, donde cada nombre de la columna actúa como clave
* $tareas = $result->fetch\_all(MYSQLI\_ASSOC);
* //Si no hay tareas, muestro mensaje de error
* if (!$tareas) {
* error\_log("Error al obtener tareas por email: " . $stmt->error); // Registro de errores
* }
* //Cierro conexión
* $stmt->close();
* return $tareas;
* }
* }
* ?>

**Class\_Usuario.php**:

* Similar a Class\_Tarea.php, este archivo contiene todas las funciones necesarias que se llaman en UsuariosController.php. Aquí, están definidas y creadas. Todo esto dentro de una clase Usuario, que comienza con una conexión nueva.
* <?php
* //Referencio al archivo conexion puesto que será necesario para el uso de funciones con la base de datos
* require\_once '../config/conexion.php';
* //Clase Usuario para gestionar la tabla de usuario
* class Usuario
* {
* //Establezco la conexión a la base de datos
* private $conexion;
* public function \_\_construct()
* {
* $this->conexion = new Conexion();
* }
* //Metodo para agregar un usuario atraves de los parametros nombre, password y email a la base de datos
* public function agregarUsuario($email, $nombre, $password)
* {
* //Encriptamos la contraseña a través de la funcion password\_hash
* $hashedPassword = password\_hash($password, PASSWORD\_DEFAULT);
* //Sintaxis SQL para agregar un usuario con el insert into
* $query = "INSERT INTO usuario (email, nombre, passwd) VALUES (?, ?, ?)";
* //Preparamos la consulta para agregar un usuario
* $stmt = $this->conexion->conexion->prepare($query);
* //Asignamos los valores a los parametros de la consulta
* $stmt->bind\_param("sss", $email, $nombre, $hashedPassword,);
* //Si la consulta se ejecuta correctamente, se muestra un mensaje de exito
* if ($stmt->execute()) {
* echo "Usuario agregado con éxito.";
* } else {
* //Si no, se muestra un mensaje de error
* echo "Error al agregar Usuario: " . $stmt->error;
* }
* $stmt->close();
* }
* //Metodo para obtener todos los usuarios de la base de datos atraves del email
* public function obtenerUsuarioPorEmail($email)
* {
* //Sintaxis SQL para seleccionar el usuario por email
* $query = "SELECT \* FROM usuario WHERE email = ?";
* //Preparamos la consulta
* $stmt = $this->conexion->conexion->prepare($query);
* //Si la consulta no se ejecuta correctamente, se muestra un mensaje de error
* if (!$stmt) {
* die("Error en la preparación de la consulta: " . $this->conexion->conexion->error);
* }
* //Asignamos el valor al parametro necesario para consulta
* $stmt->bind\_param("s", $email);
* //Si no se ejecuta correctamente, se muestra un mensaje de error
* if (!$stmt->execute()) {
* die("Error al ejecutar la consulta: " . $stmt->error);
* }
* //Guardamos el resultado de la consulta en la variable resultado
* $resultado = $stmt->get\_result();
* //Si el resultado de la consulta es mayor a 0, se retorna el resultado puesto que se ha encontrado al usuario
* if ($resultado->num\_rows > 0) {
* return $resultado->fetch\_assoc();
* } //Si no, se muestra un mensaje de error
* else {
* error\_log("Usuario no encontrado: " . $email);
* return null;
* }
* }
* public function ObtenerUsuarios()
* {
* // Preparamos la consulta para obtener todos los usuarios con el select from
* $query = "SELECT \* FROM usuario";
* // Ejecutar la query y lo guardo en la variable resultado
* $resultado = $this->conexion->conexion->query($query);
* // Creo un array vacío para guardar los usuarios
* $usuarios = [];
* // Si hay resultados, guardo cada fila con los usuarios en el array a través del fetch
* if ($resultado) {
* while ($fila = $resultado->fetch\_assoc()) {
* $usuarios[] = $fila;
* }
* } else {
* // Sino, muestro mensaje de error
* error\_log("Error al obtener usuarios: " . $this->conexion->conexion->error);
* }
* // Devuelvo el contenido del array
* return $usuarios;
* }
* //Metodo para verificar las credenciales de un usuario al iniciar sesión buscando las referencias en la base de datos
* public function VerificarUsuario($email, $passwd)
* {
* // Consulta para buscar al usuario por email en la base de datos
* $query = "SELECT \* FROM usuario WHERE email = ?";
* // Preparo la query
* $stmt = $this->conexion->conexion->prepare($query);
* // Asigno el valor al parámetro de la query
* $stmt->bind\_param("s", $email);
* // Si la consulta se ejecuta correctamente
* if ($stmt->execute()) {
* // Obtenemos el resultado
* $resultado = $stmt->get\_result();
* // Si hay resultados
* if ($resultado->num\_rows > 0) {
* // Guardo el usuario en la variable usuario
* $usuario = $resultado->fetch\_assoc();
* // Guardo el hash almacenado en la variable stored\_hash
* $stored\_hash = $usuario['passwd'];
* // Generamos el hash de la contraseña ingresada usando sha256
* $input\_hash = hash('sha256', $passwd);
* // Comparamos los hashes usando hash\_equals
* if (hash\_equals($stored\_hash, $input\_hash)) {
* // Cierro la conexión
* $stmt->close();
* $resultado->close(); // Cerrar el resultado
* // Guardo las credenciales verificadas en un archivo .log, esto me sirvió en su momento para verificar cómo se estaban guardando el email y constraseña
* $this->GuardarCredencialesEnLog($email, $passwd);
* // Si la contraseña es correcta, devolvemos los datos del usuario
* return $usuario;
* }
* } else {
* // Mensaje de error si las credenciales son incorrectas
* echo "Fallo de credenciales, incorrectas.";
* }
* } else {
* // Registrar el error en lugar de imprimirlo
* error\_log("Error al verificar usuario: " . $stmt->error);
* }
* // Cierro conexión
* $stmt->close();
* // Si no se encuentra el usuario o las credenciales son incorrectas
* return null;
* }
* //Metodo para guardar las credenciales en el archivo log
* private function GuardarCredencialesEnLog($email, $passwd)
* {
* // Nombre del archivo log
* $logFile = "credenciales\_verificadas.log";
* // Aquello que se va a guardar en el archivo log
* $logEntry = "Email: $email, Password: $passwd\n";
* // Guardar las credenciales en el archivo log
* if (file\_put\_contents($logFile, $logEntry, FILE\_APPEND | LOCK\_EX)) {
* echo "Credenciales guardadas en log correctamente.<br>";
* } //Sino, error
* else {
* error\_log("Error al guardar credenciales en el log.");
* }
* }
* }

**actualizar\_estado.php**:

* Este archivo se encarga de actualizar el estado de una tarea específica en la base de datos. Recibe los datos a través de una solicitud POST y utiliza la función ActualizarEstado de la clase Tarea.

<?php

require\_once '../controlador/TareasController.php'; // Incluye el archivo del controlador de tareas para la funcionalidad de actualizar estado

// Verifica si la solicitud es de tipo POST

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

    // Obtiene los valores de estado e id\_tarea desde el formulario

    $estado = $\_POST['estado'];

    $id\_tarea = $\_POST['id\_tarea'];

    // Crea una nueva instancia del controlador de tareas

    $controller = new TareasController();

    // Intenta actualizar el estado de la tarea

    try {

        $controller->actualizarEstado($id\_tarea, $estado);

        echo "Estado actualizado con éxito. Estado: $estado, ID Tarea: $id\_tarea"; // Mensaje de éxito

    } catch (Exception $e) {

        // Si ocurre un error, registra el error y muestra un mensaje de error

        error\_log("Error al actualizar el estado de la tarea: " . $e->getMessage()); // Registro de errores

        echo "Error al actualizar el estado de la tarea: " . $e->getMessage();

    }

}

?>

**alta\_tarea.php**:

* Se realiza el post de los datos recogidos desde el formulario, y a través de un nuevo objeto de la clase TareasController, se adjunta la función AgregarTarea para que se pueda efectuar el registro por medio de SQL preparado. Además del código HTML que nos muestra los campos para introducir los datos de la tarea (título, descripción, estado).
* <?php
* session\_start(); // Inicia la sesión
* session\_regenerate\_id(true); // Regenera el ID de la sesión para mayor seguridad
* error\_reporting(E\_ERROR); // Configura el nivel de reporte de errores
* require\_once '../controlador/TareasController.php'; // Incluye el archivo del controlador de tareas para la funcionalidad de agregar tarea
* // Verifica si el usuario está autenticado
* if ($\_SESSION['usuario'] == 'user') {
* // Usuario autenticado
* } else {
* // Redirecciona al índice si el usuario no está autenticado
* header("Location: ../index.php");
* exit();
* }
* // Verifica si la solicitud es de tipo POST
* if ($\_SERVER['REQUEST\_METHOD'] === 'POST') {
* // Obtiene los valores del formulario
* $titulo = $\_POST['titulo'];
* $descripcion = $\_POST['descripcion'];
* $estado = $\_POST['estado'];
* $email = $\_SESSION['email'];
* // Crea una nueva instancia del controlador de tareas
* $controller = new TareasController();
* // Intenta agregar la nueva tarea
* try {
* $controller->AgregarTarea($email, $titulo, $descripcion, $estado);
* // Redirecciona al índice después de agregar la tarea
* header("Location: ../index.php");
* exit();
* } catch (Exception $e) {
* // Si ocurre un error, registra el error y muestra un mensaje de error
* error\_log("Error al agregar la tarea: " . $e->getMessage()); // Registro de errores
* echo "Error al agregar la tarea: " . $e->getMessage();
* }
* }
* ?>
* <!DOCTYPE html>
* <html lang="es">
* <head>
* <meta charset="UTF-8">
* <title>Alta de Nueva Tarea</title>
* <!-- Integración de Bootstrap -->
* <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css">
* <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.7.1/jquery.min.js"></script>
* <script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.11.6/dist/umd/popper.min.js"></script>
* <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js"></script>
* <link rel="stylesheet" href="../vista/Estilo.css"><!-- Enlaza el archivo CSS -->
* <script src="JS.js"></script> <!-- Enlaza el archivo JavaScript -->
* </head>
* <body>
* <div class="container mt-5">
* <h1 class="text-center mb-4">Alta de Nueva Tarea</h1>
* <form action="alta\_tarea.php" method="post">
* <!-- Campo del titulo para agregar una nueva tarea -->
* <div class="form-group">
* <label for="titulo">Título</label>
* <input type="text" name="titulo" id="titulo" class="form-control" required>
* </div>
* <!-- Campo de descripción de la tarea -->
* <div class="form-group">
* <label for="descripcion">Descripción</label>
* <input type="text" name="descripcion" id="descripcion" class="form-control" required>
* </div>
* <!-- Campo de estado de la tarea -->
* <div class="form-group">
* <label for="estado">Estado</label>
* <select name="estado" id="estado" class="form-control" onchange="actualizarEstado(this.value)">
* <option value="Pendiente">Pendiente</option>
* <option value="En proceso">En proceso</option>
* <option value="Bloqueada">Bloqueada</option>
* <option value="Finalizada">Finalizada</option>
* </select>
* </div>
* <!-- Botón para guardar la tarea -->
* <div class="text-center mt-3">
* <button type="submit" class="btn btn1-purple">Guardar</button>
* </div>
* </form>
* </div>
* <!-- Integración de Bootstrap -->
* <script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>
* <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js"></script>
* </body>
* </html>

**alta\_usuario.php**:

* Se realiza el post de los datos recogidos desde el formulario, y a través de un nuevo objeto de la clase UsuariosController, se adjunta la función AgregarUsuario para que se pueda efectuar el registro por medio de SQL preparado. Además del código HTML que nos muestra los campos para introducir los datos del usuario (email, nombre, contraseña).

<?php

// Inicio la sesión

session\_start();

// Regenero el ID de la sesión para mayor seguridad

session\_regenerate\_id(true);

// Configuro el nivel de reporte de errores

error\_reporting(E\_ERROR);

// Incluyo el archivo del controlador de usuarios para la funcionalidad de agregar usuario

require\_once '../controlador/UsuariosController.php';

// Verifica si la solicitud es de tipo POST

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

    // Obtiene los valores del formulario

    $email = $\_POST['email'];

    $password = $\_POST['passwd'];

    $nombre = $\_POST['nombre'];

    // Crea una nueva instancia del controlador de usuarios

    $controller = new UsuariosController();

    // Intenta agregar el nuevo usuario

    try {

        $controller->AgregarUsuario($email, $nombre, $password);

        // Redirecciona al índice después de agregar el usuario

        header("Location: ../index.php");

        exit();

    } catch (Exception $e) {

        // Si ocurre un error, muestra un mensaje de error

        error\_log("Error al agregar el usuario: " . $e->getMessage());

        echo "Error al agregar el usuario: " . $e->getMessage();

    }

}

?>

<!DOCTYPE html>

<html lang="es">

<head>

    <meta charset="UTF-8">

    <title>Alta de Nuevo Usuario</title>

    <!-- Integración de Bootstrap -->

    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css">

    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.7.1/jquery.min.js"></script>

    <script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.11.6/dist/umd/popper.min.js"></script>

    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js"></script>

    <link rel="stylesheet" href="../vista/Estilo.css"> <!-- Enlaza el archivo CSS -->

    <script src="JS.js"></script> <!-- Enlaza el archivo JavaScript -->

</head>

<body>

    <div class="container mt-5">

        <!-- Formulario para agregar un nuevo usuario -->

        <h1 class="text-center mb-4">Alta de Nuevo Usuario</h1>

        <form action="alta\_usuario.php" method="post">

            <!-- Campo del email para agregar un nuevo usuario -->

            <div class="form-group">

                <label for="email">Email</label>

                <input type="email" name="email" id="email" class="form-control" required>

            </div>

            <!-- Campo de la contraseña para agregar un nuevo usuario -->

            <div class="form-group">

                <label for="Password">Password</label>

                <input type="password" name="passwd" id="passwd" class="form-control" required>

            </div>

            <!-- Campo del nombre para agregar un nuevo usuario -->

            <div class="form-group">

                <label for="nombre">Nombre</label>

                <input type="text" name="nombre" id="nombre" class="form-control" required>

            </div>

            <!-- Checkbox para aceptar los términos de uso, con una referencia a la funcion del js -->

            <div class="form-group form-check">

                <input type="checkbox" name="terminos" id="checkbox" onchange="enabler()" class="form-check-input" required>

                <label for="aceptar\_terminos" class="form-check-label">Acepto los términos de uso</label>

            </div>

            <!-- Botón para guardar el nuevo usuario -->

            <div class="d-flex justify-content-center align-items-center">

                <button id="button" type="submit" class="btn btn1-purple" disabled>Guardar</button>

            </div>

        </form>

    </div>

    <!-- Integración de Bootstrap -->

    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js"></script>

</body>

</html>

**eliminar\_tarea.php**:

* Utiliza GET para obtener los datos del id\_tarea, una vez teniendo esto, se crea un nuevo objeto de TareasController utilizando la función EliminarTarea para, por medio de este ID, eliminar la tarea.
* <?php
* // Inicia la sesión
* session\_start();
* // Regenera el ID de la sesión para mayor seguridad
* session\_regenerate\_id(true);
* // Configura el nivel de reporte de errores
* error\_reporting(E\_ERROR);
* // Incluye el archivo del controlador de tareas para la funcionalidad de eliminar tarea
* require\_once '../controlador/TareasController.php';
* // Verifica si la solicitud es de tipo GET
* if ($\_SERVER['REQUEST\_METHOD'] === 'GET') {
* // Obtiene el ID de la tarea
* $id\_tarea = $\_GET['id'];
* // Crea una nueva instancia del controlador de tareas
* $controller = new TareasController();
* // Intenta eliminar la tarea
* try {
* $controller->EliminarTarea($id\_tarea);
* // Redirecciona al índice después de eliminar la tarea
* header("Location: ../index.php");
* exit();
* } catch (Exception $e) {
* // Si ocurre un error, registra el error y muestra un mensaje de error
* error\_log("Error al eliminar la tarea: " . $e->getMessage());
* echo "Error al eliminar la tarea: " . $e->getMessage();
* }
* }
* ?>

**Estilo.css**:

* Configura los estilos para que pueda mostrar la estética deseada cada elemento referenciado.
* /\* Botón con fondo púrpura \*/
* .btn1-purple {
* background-color: #6f42c1;
* border-color: #6f42c1;
* color: white;
* padding: 1% 2%;
* margin-top: 5%;
* font-size: 16px;
* align-items: center;
* }
* /\* Botón con fondo rojo \*/
* .btn2 {
* background-color: #dc3545;
* border-color: #dc3545;
* color: white;
* padding: 12px 78px;
* font-size: 16px;
* display: block;
* align-items: center;
* margin-top: 4%;
* }
* /\* Estilo para los elementos de navegación \*/
* .nav-item {
* padding: 6px 6px;
* }
* /\* Estilo para la marca de la barra de navegación \*/
* .navbar-brand {
* padding: 9px;
* }
* /\* Botón con fondo verde \*/
* .btn3 {
* color: white;
* background-color: green;
* margin-bottom: 5%;
* }
* /\* Estilo para el formulario de inicio de sesión \*/
* .login-form {
* font-family: "Artifika", serif;
* font-weight: 400;
* font-style: normal;
* margin-top: 7%;
* color: #ffe4df;
* text-align: center;
* }
* /\* Estilo para el título del formulario de inicio de sesión \*/
* .login.title {
* padding: 1.5%;
* }
* /\* Estilo para el campo de usuario del formulario de inicio de sesión \*/
* .login-user {
* padding: 1.5%;
* }
* /\* Estilo para el campo de contraseña del formulario de inicio de sesión \*/
* .login-passwd {
* padding: 1.5%;
* }
* /\* Estilo para el botón del formulario de inicio de sesión \*/
* .login-btn {
* font-family: "Artifika", serif;
* font-weight: 400;
* font-style: normal;
* background-color: bisque;
* padding: 10px 20px;
* margin-bottom: 35px;
* }
* /\* Estilo alternativo para el botón del formulario de inicio de sesión \*/
* .login-btn1 {
* font-family: "Artifika", serif;
* font-weight: 400;
* font-style: normal;
* background-color: bisque;
* padding: 10px 20px;
* color: white;
* }
* /\* Estilo para el cuerpo del formulario de inicio de sesión \*/
* .login-body {
* background-color: #351111;
* }
* /\* Botón con fondo púrpura \*/
* .btn-4 {
* background-color: #6f42c1;
* border-color: #6f42c1;
* color: white;
* font-size: 16px;
* align-items: center;
* margin-top: 4%;
* }
* /\* Botón primario con fondo púrpura \*/
* .btn-primary-1 {
* background-color: #6f42c1;
* border-color: #6f42c1;
* color: white;
* font-size: 16px;
* align-items: center;
* margin-top: 4%;
* }
* /\* Botón con fondo púrpura oscuro \*/
* .btn3 {
* color: white;
* background-color: rgb(136, 39, 182);
* margin-top: 4%;
* }
* /\* Estilo para el mensaje de bienvenida \*/
* .h5-Bienvenida {
* color: white;
* margin-left: 12%;
* align-items: center;
* }

**JS.js**:

* Aquí se procesan las funciones JavaScript necesarias para la interacción en tiempo real con el servidor, como la actualización del estado de las tareas y la habilitación/deshabilitación de botones.
* // Función para actualizar el estado de una tarea en tiempo real
* function actualizarEstado(id\_tarea, estado) {
* //Agrego esta línea para saber si se está actovando la función correctamente
* console.log("Enviando solicitud para actualizar estado:", id\_tarea, estado);
* // Crea una nueva instancia de XMLHttpRequest para la solicitud al servidor
* const xhr = new XMLHttpRequest();
* //Marca la solicitud como POST y establece la URL a la que se enviarán los datos
* xhr.open("POST", "actualizar\_estado.php", true);
* // Establece el encabezado de la solicitud, es importante para que el servidor pueda interpretar el formato de los datos enviados en la solicitud
* xhr.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
* // Establece la función que se ejecutará cuando el estado de la solicitud cambie
* xhr.onreadystatechange = function () {
* // Verifica el estado de la solicitud, al ser 4 significa que la solicitud ha sido completada y la respuesta está lista de ser procesada
* if (xhr.readyState === 4) {
* // Verifica el estado de la respuesta, al ser 200 significa que la solicitud fue exitosa y se devolvieron los datos correctamente
* if (xhr.status === 200) {
* // Muestra un mensaje en la consola para indicar que el estado ha sido actualizado
* console.log("Estado actualizado");
* // Muestra la respuesta del servidor en la consola para verificar que los datos se han actualizado correctamente
* console.log("Respuesta del servidor:", xhr.responseText);
* } else {
* // Muestra un mensaje de error en la consola si la solicitud no se ha completado correctamente
* console.error("Error al actualizar estado: " + xhr.statusText);
* }
* }
* };
* // Envía la solicitud con los datos requeridos para actualizar el estado de la tarea
* xhr.send("id\_tarea=" + encodeURIComponent(id\_tarea) + "&estado=" + encodeURIComponent(estado));
* }
* // Función para habilitar o deshabilitar el botón basado en el estado del checkbox
* function enabler() {
* // Obtiene el botón por su ID
* const button = document.getElementById("button");
* // Obtiene el checkbox por su ID
* const check = document.getElementById("checkbox");
* // Habilita o deshabilita el botón basado en el estado del checkbox
* button.disabled = !check.checked;
* }

**lista\_tareas.php**:

* Crea un nuevo objeto de TareasController con la función ListarTareasPorEmail para mostrar las tareas del usuario. Incluye el código HTML para mostrar una tabla con las tareas registradas, permitiendo actualizar el estado de las tareas en tiempo real y eliminarlas.
* <?php
* // Inicio la sesión
* session\_start();
* // Regenera el ID de la sesión para mayor seguridad
* session\_regenerate\_id(true);
* // Configura el nivel de reporte de errores
* error\_reporting(E\_ERROR);
* // Incluye el archivo del controlador de tareas para la funcionalidad de listar tareas
* require\_once '../controlador/TareasController.php';
* // Crea una nueva instancia del controlador de tareas
* $controller = new TareasController();
* // Obtiene el email del usuario desde la sesión
* $email = $\_SESSION['email'];
* // Obtiene las tareas del usuario por email
* $tareas = $controller->ListarTareasPorEmail($email);
* ?>
* <!DOCTYPE html>
* <html lang="es">
* <head>
* <meta charset="UTF-8">
* <title>Listado de Tareas</title>
* <!-- Integración de Bootstrap -->
* <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css">
* <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.7.1/jquery.min.js"></script>
* <script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.11.6/dist/umd/popper.min.js"></script>
* <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js"></script>
* <link rel="stylesheet" href="../vista/Estilo.css"><!-- Enlaza el archivo CSS -->
* <script src="JS.js"></script> <!-- Enlaza el archivo JavaScript -->
* </head>
* <body>
* <nav class="navbar navbar-expand-lg navbar-dark bg-dark">
* <a class="navbar-brand" href="../index.php">Gestión de Tareas</a>
* <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarNav" aria-controls="navbarNav" aria-expanded="false" aria-label="Toggle navigation">
* <span class="navbar-toggler-icon"></span>
* </button>
* <div class="collapse navbar-collapse" id="navbarNav">
* <ul class="navbar-nav">
* <!-- Botón de cerrar sesión -->
* <li class="ml-auto">
* <a class="nav-link btn btn2 d-flex text-white" href="logout.php">Cerrar Sesión</a>
* </li>
* <!-- Bienvenida al usuario -->
* <li class="nav-item-Bienvenida">
* <h5 class="h5-Bienvenida">Bienvenido <?php echo $\_SESSION['nombre']; ?>!</h5>
* </li>
* </ul>
* </div>
* </nav>
* <div class="container mt-5">
* <!-- Tabla de tareas registradas -->
* <h1 class="text-center mb-4">Tareas Registradas</h1>
* <table class="table table-striped table-bordered">
* <thead class="thead-dark">
* <tr>
* <th>Email</th>
* <th>Título</th>
* <th>Descripción</th>
* <th>Estado</th>
* <th>Acciones</th>
* </tr>
* </thead>
* <tbody>
* <!-- Itera sobre las tareas y muestra los datos en la tabla -->
* <?php foreach ($tareas as $tarea): ?>
* <tr>
* <td><?= $tarea['email'] ?></td>
* <td><?= $tarea['titulo'] ?></td>
* <td><?= $tarea['descripcion'] ?></td>
* <td>
* <!-- En el caso del estado un desplegable, y relaciono con la función js para actualizar el campo en el momento -->
* <select class="form-control" onchange="actualizarEstado(<?= $tarea['id\_tarea'] ?>, this.value)">
* <!-- En el caso de que el estado sea igual a Pendiente, se selecciona esa opción, y viceversa -->
* <option value="Pendiente" <?= $tarea['estado'] == 'Pendiente' ? 'selected' : '' ?>>Pendiente</option>
* <option value="En proceso" <?= $tarea['estado'] == 'En proceso' ? 'selected' : '' ?>>En proceso</option>
* <option value="Bloqueada" <?= $tarea['estado'] == 'Bloqueada' ? 'selected' : '' ?>>Bloqueada</option>
* <option value="Finalizada" <?= $tarea['estado'] == 'Finalizada' ? 'selected' : '' ?>>Finalizada</option>
* </select>
* </td>
* <td>
* <!-- Botón para editar la tarea -->
* <button class="btn btn-primary-1 mb-3">
* <a href="eliminar\_tarea.php?id=<?= $tarea['id\_tarea'] ?>">Eliminar</a>
* </button>
* </td>
* </tr>
* <!-- Fin del foreach -->
* <?php endforeach; ?>
* </tbody>
* </table>
* <!-- Botón para agregar una nueva tarea -->
* <div class="text-center">
* <a href="alta\_tarea.php" class="btn btn3">Agregar nueva tarea</a>
* </div>
* </div>
* </body>
* </html>

**login.php**:

* Maneja el inicio de sesión del usuario. Verifica las credenciales del usuario y, si son correctas, inicia la sesión y redirige a la lista de tareas. Si las credenciales son incorrectas, muestra un mensaje de error.
* <?php
* // Inicio la sesión
* session\_start();
* // Regenera el ID de la sesión para mayor seguridad
* session\_regenerate\_id(true);
* // Configura el nivel de reporte de errores
* error\_reporting(E\_ERROR);
* // Incluye el archivo del controlador de usuarios para la funcionalidad de obtenención usuario por email
* require\_once '../controlador/UsuariosController.php';
* if ($\_SERVER['REQUEST\_METHOD'] === 'POST') {
* // Obtiene el valor del campo de email
* $email = $\_POST['email'];
* // Obtiene el valor del campo de contraseña y elimina los espacios en blanco
* $password = trim($\_POST['passwd']);
* // Crea una nueva instancia del controlador de usuarios y obtiene el usuario por email
* $controller = new UsuariosController();
* $user = $controller->obtenerUsuarioPorEmail($email);
* // Si el usuario existe y la contraseña es correcta, iniciamos sesión
* if ($user && $email == $user['email'] && password\_verify($password, $user['passwd'])) {
* // Guardamos los datos del usuario en la sesión
* $\_SESSION['usuario'] = 'user';
* $\_SESSION['nombre'] = $user['nombre'];
* $\_SESSION['email'] = $user['email'];
* // Redirigimos a la página de tareas
* header("Location: lista\_tareas.php");
* exit();
* } else {
* // Si el usuario no existe o la contraseña es incorrecta, mostramos un mensaje de error
* error\_log("Error de inicio de sesión para usuario: " . $email);
* $error\_message = "Usuario o contraseña incorrectos.";
* }
* }
* ?>
* <!DOCTYPE html>
* <html lang="es">
* <head>
* <meta charset="UTF-8">
* <title>Iniciar Sesión</title>
* <!-- Integración de Bootstrap -->
* <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css">
* <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.7.1/jquery.min.js"></script>
* <script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.11.6/dist/umd/popper.min.js"></script>
* <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js"></script>
* <link rel="stylesheet" href="../vista/Estilo.css"> <!-- Enlaza el archivo CSS -->
* <script src="JS.js"></script> <!-- Enlaza el archivo JavaScript -->
* </head>
* <body class="login-body">
* <form action="login.php" method="POST" class="login-form">
* <h2 class="login-h2">Iniciar Sesión</h2>
* <!-- Campo del email para iniciar sesión -->
* <div class="login-user">
* <label for="email">Email</label>
* <input type="email" id="email" name="email" required>
* </div>
* <!-- Campo de la contraseña para iniciar sesión -->
* <div class="login-passwd">
* <label for="passwd">Contraseña</label>
* <input type="password" id="passwd" name="passwd" required>
* </div>
* <!-- Botón de inicio de sesión -->
* <div>
* <button class="login-btn" type="submit">Iniciar Sesión</button>
* </div>
* <!-- Enlace para registrarse -->
* <div>
* <a href="alta\_usuario.php" class="btn login-btn1">Registrarse</a>
* </div>
* <!-- Muestra un mensaje de error si existe -->
* <?php if (isset($error\_message)): ?>
* <div class="alert alert-danger mt-3">
* <?= $error\_message ?>
* </div>
* <?php endif; ?>
* </form>
* </body>
* </html>

**logout.php**:

* Maneja el cierre de sesión del usuario. Elimina todas las variables de sesión y destruye la sesión, luego redirige al usuario a la página de inicio de sesión.
* <?php
* // Configuración de los reportes de errores para mostrar solo errores fatales
* error\_reporting(E\_ERROR);
* // Inicia la sesión
* session\_start();
* // Elimina todas las variables de sesión
* session\_unset();
* // Destruye la sesión
* session\_destroy();
* // Redirige al usuario a la página de inicio de sesión
* header("Location: login.php");
* // Finaliza el script
* exit();
* ?>

**DB.sql**:

* Muestra los datos necesarios para formar la db y los atributos de la tabla.
* CREATE DATABASE hito\_2\_2ºTrimestre\_Alejandro\_Cortes\_Diaz
* use hito\_2\_2ºTrimestre\_Alejandro\_Cortes\_Diaz
* create table usuario(
* email varchar (250) primary key,
* nombre varchar (250),
* passwd VARCHAR(250)
* );
* CREATE TABLE tarea (
* id\_tarea INT PRIMARY KEY AUTO\_INCREMENT,
* email VARCHAR(250),
* titulo VARCHAR(250),
* descripcion VARCHAR(500),
* estado SET ("Pendiente", "En proceso", "Bloqueada", "Finalizada"),
* FOREIGN KEY (email) REFERENCES usuario(email)
* );

**index.php**:

* Redirecciona automáticamente a lista\_tareas.php si el usuario está autenticado, o a login.php si no lo está.
* <?php
* // Inicia la sesión
* session\_start();
* // Regenera el ID de la sesión para mayor seguridad
* session\_regenerate\_id(true);
* // Configura el nivel de reporte de errores para mostrar solo errores graves
* error\_reporting(E\_ERROR);
* // Verifica si el usuario está autenticado
* if ($\_SESSION['usuario'] == 'user') {
* // Redirige a la lista de tareas si el usuario está autenticado
* header("Location: vista/lista\_tareas.php");
* } else {
* // Redirige a la página de inicio de sesión si el usuario no está autenticado
* header("Location: vista/login.php");
* }
* // Finaliza el script
* exit();
* ?>

# Explicación de las funciones del código

**AgregarTarea:**

* Se encarga de agregar una nueva tarea a la base de datos.

Se realiza de la siguiente forma:

* Partimos de la lista de las tareas para utilizar el botón “Agregar nueva tarea”:

Interfaz de usuario gráfica, Aplicación, Teams

El contenido generado por IA puede ser incorrecto.

* Presionamos el botón;

Interfaz de usuario gráfica, Aplicación, Teams

El contenido generado por IA puede ser incorrecto.

* Rellenamos campos y presionamos botón;

Captura de pantalla de un celular

El contenido generado por IA puede ser incorrecto.

* Pasa añadirse

**ObtenerTareasPorEmail:**

* Este método obtiene todas las tareas almacenadas en la base de datos con consulta SQL tomando el email como referencia para mostrar las tareas relacionadas con ese email.

Interfaz de usuario gráfica, Texto, Aplicación, Teams

El contenido generado por IA puede ser incorrecto.

**EliminarTarea:**

* Este método elimina una tarea específica basada en su ID.
* Eliminamos una tarea a través del botón “Eliminar”

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

* Antes

Interfaz de usuario gráfica, Aplicación, Teams

El contenido generado por IA puede ser incorrecto.

* Después

**ActualizarEstado**:

* Este método actualiza el estado de una tarea específica a través de su ID.

Captura de pantalla de un celular

El contenido generado por IA puede ser incorrecto.

* Al cambiar el estado, automáticamente se produce el cambio, aun refrescándose, no cambia.

**agregarUsuario**:

* Este método se encarga de agregar un nuevo usuario a la base de datos.
* Nos registramos:

Imagen que contiene Gráfico

El contenido generado por IA puede ser incorrecto.

* Rellenamos los campos y guardamos

Interfaz de usuario gráfica, Aplicación, Teams

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Aplicación, Teams

El contenido generado por IA puede ser incorrecto.

* Vemos como se ha agregado y ya puede iniciar sesión

**obtenerUsuarioPorEmail**:

* **Este** método obtiene un usuario específico basado en su correo electrónico. Se emplea por atrás en el login para comprobar si hay un usuario con la sesión iniciada por el email.

**enabler**:

* **Habilita o deshabilita el botón basado en el estado del checkbox**
* Boton sin aceptar condiciones

Interfaz de usuario gráfica, Aplicación, Teams

El contenido generado por IA puede ser incorrecto.

* Boton con condiciones aceptadas

# **Enlace a GitHub**

<https://github.com/Cortes-cmd/Programacion.git>

# **Bibliografía**

$\_SESSION. (s/f). Php.net. Recuperado el 16 de febrero de 2025, de <https://www.php.net/manual/en/reserved.variables.session.php>

ChatGPT. (s/f). Chatgpt.com. Recuperado el 16 de febrero de 2025, de <https://chatgpt.com>

PHP - Sessions. (s/f). Tutorialspoint.com. Recuperado el 16 de febrero de 2025, de <https://www.tutorialspoint.com/php/php_sessions.htm>

PHP sessions. (s/f). W3schools.com. Recuperado el 16 de febrero de 2025, de <https://www.w3schools.com/Php/php_sessions.asp>

Using PHP database sessions and needing to regenerate session id issue. (2023, mayo 14). SitePoint Forums | Web Development & Design Community. <https://www.sitepoint.com/community/t/using-php-database-sessions-and-needing-to-regenerate-session-id-issue/415716>

When and why I should use session\_regenerate\_id()? (s/f). Stack Overflow. Recuperado el 16 de febrero de 2025, de <https://stackoverflow.com/questions/22965067/when-and-why-i-should-use-session-regenerate-id>