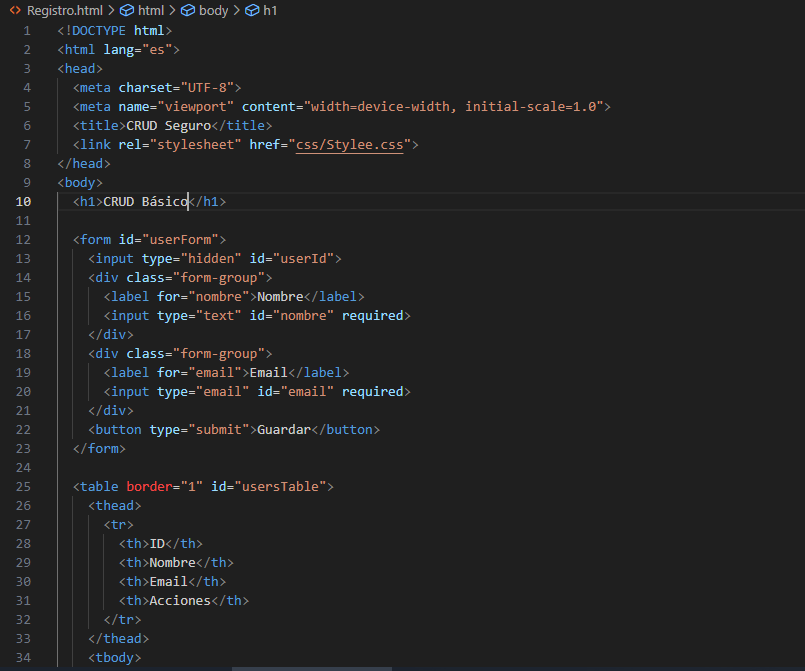
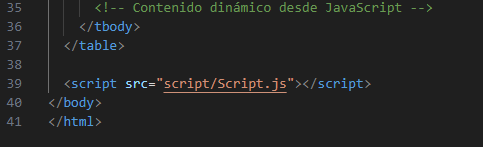
Manual Tecnico Explicado Línea por Línea - CRUD BASICO SEGURO

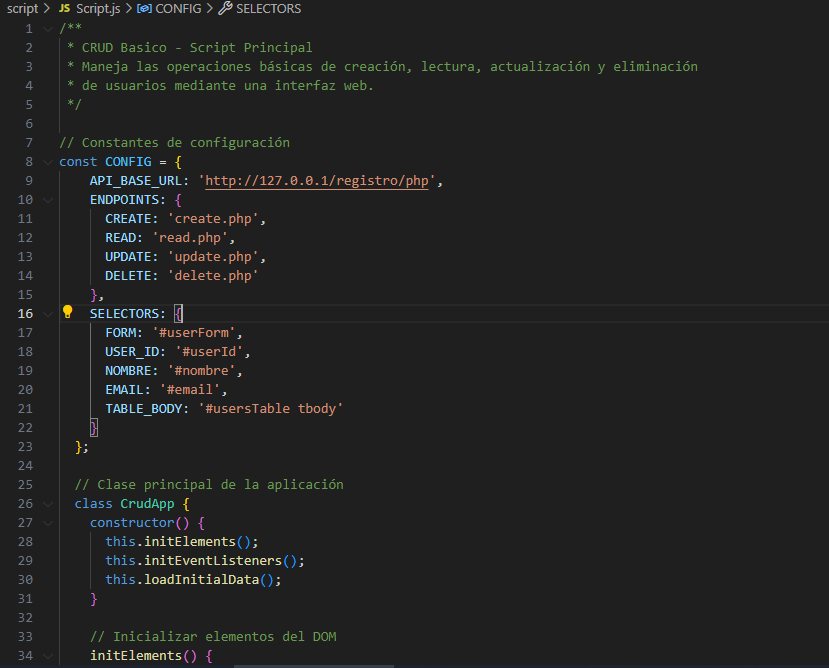
# Archivo: Registro.html

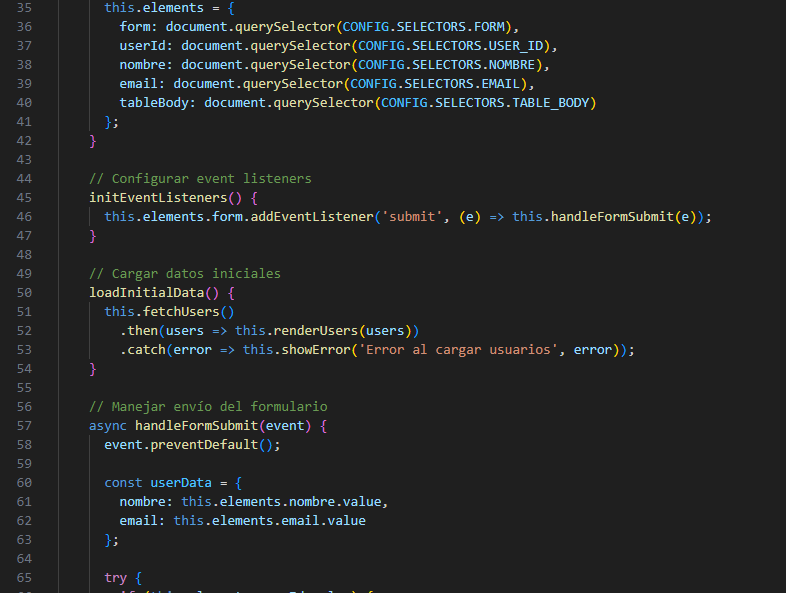


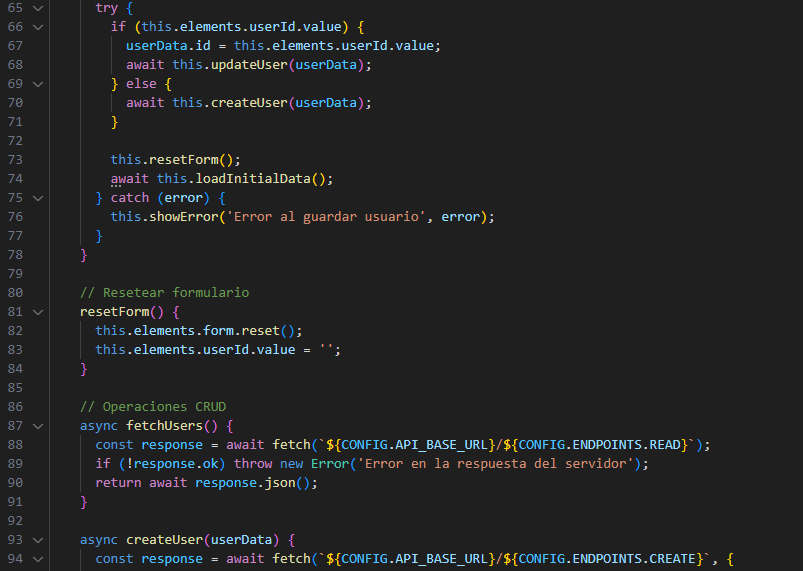


<!DOCTYPE html>  
- Define que el documento es HTML5.  
  
<html lang="es">  
- Establece que el idioma del contenido es español.  
  
<head> ... </head>  
- Contiene metadatos y enlaces a archivos externos.  
  
<meta charset="UTF-8">  
- Codificación para admitir caracteres especiales como tildes.  
  
<meta name="viewport"...>  
- Adaptación responsive para dispositivos móviles.  
  
<title>CRUD Seguro</title>  
- Título visible en la pestaña del navegador.  
  
<link rel="stylesheet"...>  
- Enlace al archivo de estilos CSS.  
  
<body> ... </body>  
- Cuerpo principal de la página.  
  
<h1>CRUD Básico</h1>  
- Encabezado principal de la interfaz.  
  
<form id="userForm"> ... </form>  
- Formulario para ingresar y editar datos de usuario.  
  
<table id="usersTable"> ... </table>  
- Tabla para mostrar la lista de usuarios registrados.  
  
<script src="script/Script.js"></script>  
- Vincula el archivo JavaScript con la lógica del CRUD.

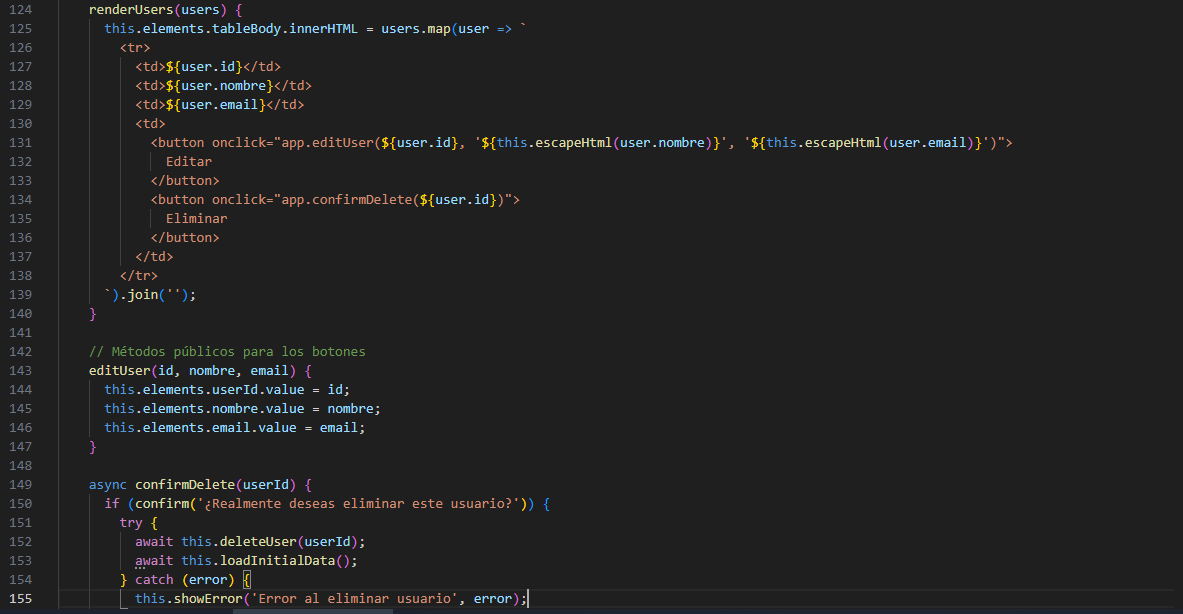
Archivo: Script.js











  
Explicación línea por línea del archivo Script.js

/\*\*

\* CRUD Basico - Script Principal

\* Maneja las operaciones básicas de creación, lectura, actualización y eliminación

\* de usuarios mediante una interfaz web.

\*/

Comentario de documentación que describe el propósito general del script.

// Constantes de configuración

const CONFIG = {

Declara un objeto constante CONFIG que almacena todas las configuraciones.

API\_BASE\_URL: 'http://127.0.0.1/registro/php',

URL base para las API endpoints.

ENDPOINTS: {

CREATE: 'create.php',

READ: 'read.php',

UPDATE: 'update.php',

DELETE: 'delete.php'

},

Nombres de los archivos PHP que manejan cada operación CRUD.

SELECTORS: {

FORM: '#userForm',

USER\_ID: '#userId',

NOMBRE: '#nombre',

EMAIL: '#email',

TABLE\_BODY: '#usersTable tbody'

}

};

Selectores CSS para los elementos del DOM.

// Clase principal de la aplicación

class CrudApp {

Define la clase principal CrudApp que encapsula toda la funcionalidad.

constructor() {

this.initElements();

this.initEventListeners();

this.loadInitialData();

}

Constructor que inicializa elementos, event listeners y carga datos.

// Inicializar elementos del DOM

initElements() {

this.elements = {

form: document.querySelector(CONFIG.SELECTORS.FORM),

userId: document.querySelector(CONFIG.SELECTORS.USER\_ID),

nombre: document.querySelector(CONFIG.SELECTORS.NOMBRE),

email: document.querySelector(CONFIG.SELECTORS.EMAIL),

tableBody: document.querySelector(CONFIG.SELECTORS.TABLE\_BODY)

};

}

Obtiene referencias a los elementos del DOM usando los selectores configurados.

// Configurar event listeners

initEventListeners() {

this.elements.form.addEventListener('submit', (e) => this.handleFormSubmit(e));

}

Configura el listener para el evento submit del formulario.

// Cargar datos iniciales

loadInitialData() {

this.fetchUsers()

.then(users => this.renderUsers(users))

.catch(error => this.showError('Error al cargar usuarios', error));

}

Carga los usuarios iniciales y los renderiza, manejando posibles errores.

// Manejar envío del formulario

async handleFormSubmit(event) {

event.preventDefault();

Previene el comportamiento por defecto del formulario.

const userData = {

nombre: this.elements.nombre.value,

email: this.elements.email.value

};

Obtiene los valores del formulario.

try {

if (this.elements.userId.value) {

userData.id = this.elements.userId.value;

await this.updateUser(userData);

} else {

await this.createUser(userData);

}

Decide si crear o actualizar basado en si hay un ID de usuario.

this.resetForm();

await this.loadInitialData();

} catch (error) {

this.showError('Error al guardar usuario', error);

}

}

Resetea el formulario y recarga datos, manejando errores.

// Resetear formulario

resetForm() {

this.elements.form.reset();

this.elements.userId.value = '';

}

Limpia el formulario.

// Operaciones CRUD

async fetchUsers() {

const response = await fetch(`${CONFIG.API\_BASE\_URL}/${CONFIG.ENDPOINTS.READ}`);

if (!response.ok) throw new Error('Error en la respuesta del servidor');

return await response.json();

}

Obtiene todos los usuarios desde el servidor.

async createUser(userData) {

const response = await fetch(`${CONFIG.API\_BASE\_URL}/${CONFIG.ENDPOINTS.CREATE}`, {

method: 'POST',

headers: { 'Content-Type': 'application/json' },

body: JSON.stringify(userData)

});

if (!response.ok) throw new Error('Error al crear usuario');

return await response.json();

}

Crea un nuevo usuario.

async updateUser(userData) {

const response = await fetch(`${CONFIG.API\_BASE\_URL}/${CONFIG.ENDPOINTS.UPDATE}`, {

method: 'PUT',

headers: { 'Content-Type': 'application/json' },

body: JSON.stringify(userData)

});

if (!response.ok) throw new Error('Error al actualizar usuario');

return await response.json();

}

Actualiza un usuario existente.

async deleteUser(userId) {

const response = await fetch(`${CONFIG.API\_BASE\_URL}/${CONFIG.ENDPOINTS.DELETE}`, {

method: 'DELETE',

headers: { 'Content-Type': 'application/json' },

body: JSON.stringify({ id: userId })

});

if (!response.ok) throw new Error('Error al eliminar usuario');

return await response.json();

}

Elimina un usuario.

// Renderizar usuarios en la tabla

renderUsers(users) {

this.elements.tableBody.innerHTML = users.map(user => `

<tr>

<td>${user.id}</td>

<td>${user.nombre}</td>

<td>${user.email}</td>

<td>

<button onclick="app.editUser(${user.id}, '${this.escapeHtml(user.nombre)}', '${this.escapeHtml(user.email)}')">

Editar

</button>

<button onclick="app.confirmDelete(${user.id})">

Eliminar

</button>

</td>

</tr>

`).join('');

}

Genera el HTML para mostrar los usuarios en la tabla con botones de acciones.

// Métodos públicos para los botones

editUser(id, nombre, email) {

this.elements.userId.value = id;

this.elements.nombre.value = nombre;

this.elements.email.value = email;

}

Llena el formulario con datos para editar.

async confirmDelete(userId) {

if (confirm('¿Realmente deseas eliminar este usuario?')) {

try {

await this.deleteUser(userId);

await this.loadInitialData();

} catch (error) {

this.showError('Error al eliminar usuario', error);

}

}

}

Muestra confirmación antes de eliminar.

// Utilidades

escapeHtml(text) {

return text.toString()

.replace(/&/g, '&amp;')

.replace(/</g, '&lt;')

.replace(/>/g, '&gt;')

.replace(/"/g, '&quot;')

.replace(/'/g, '&#039;');

}

Previene XSS escapando caracteres HTML.

showError(message, error) {

console.error(`${message}:`, error);

alert(`${message}. Ver consola para detalles.`);

}

Muestra errores en consola y alerta.

// Inicializar la aplicación cuando el DOM esté listo

document.addEventListener('DOMContentLoaded', () => {

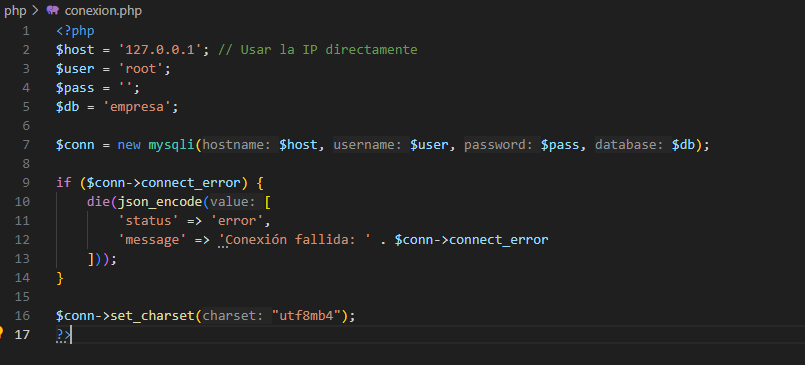
window.app = new CrudApp();

});

Inicia la aplicación cuando el DOM está cargado, asignándola a window.app

Explicación línea por línea de los archivos PHP

conexion.php



<?php

Inicio del script PHP.

$host = '127.0.0.1'; // Usar la IP directamente

Define la dirección del servidor de base de datos (localhost).

$user = 'root';

Nombre de usuario para la conexión a MySQL.

$pass = '';

Contraseña para la conexión (vacía en este caso).

$db = 'empresa';

Nombre de la base de datos a utilizar.

$conn = new mysqli($host, $user, $pass, $db);

Crea una nueva conexión MySQLi con los parámetros especificados.

if ($conn->connect\_error) {

Verifica si hubo error en la conexión.

die(json\_encode([

'status' => 'error',

'message' => 'Conexión fallida: ' . $conn->connect\_error

]));

}

Si hay error, termina el script y devuelve un JSON con el mensaje de error.

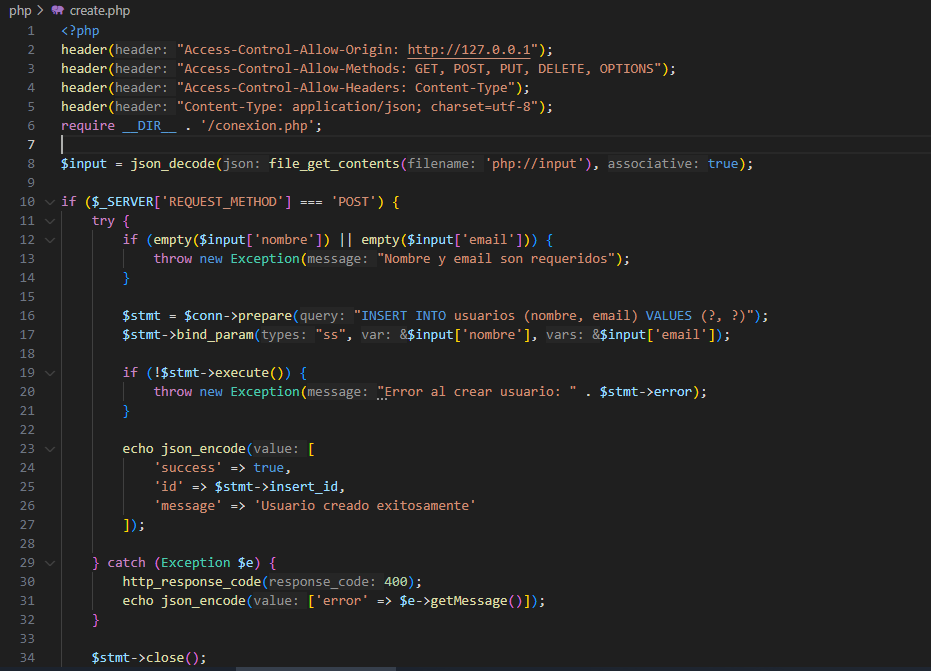
$conn->set\_charset("utf8mb4");

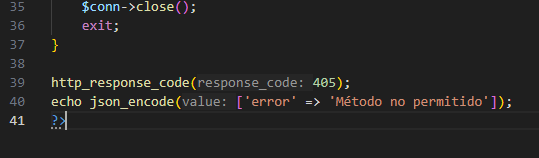
Establece el charset a utf8mb4 para soportar todos los caracteres Unicode.

?>

Fin del script PHP.

CREATE.PHP





<?php

header("Access-Control-Allow-Origin: http://127.0.0.1");

Permite solicitudes CORS desde el dominio especificado.

header("Access-Control-Allow-Methods: GET, POST, PUT, DELETE, OPTIONS");

Especifica los métodos HTTP permitidos.

header("Access-Control-Allow-Headers: Content-Type");

Permite el header Content-Type.

header("Content-Type: application/json; charset=utf-8");

Establece el tipo de contenido como JSON con charset UTF-8.

require \_\_DIR\_\_ . '/conexion.php';

Incluye el archivo de conexión a la base de datos.

$input = json\_decode(file\_get\_contents('php://input'), true);

Lee y decodifica el input JSON recibido.

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

Verifica que el método de solicitud sea POST.

try {

Inicia bloque try-catch para manejo de errores.

if (empty($input['nombre']) || empty($input['email'])) {

throw new Exception("Nombre y email son requeridos");

}

Valida que nombre y email no estén vacíos.

$stmt = $conn->prepare("INSERT INTO usuarios (nombre, email) VALUES (?, ?)");

Prepara la consulta SQL con parámetros.

$stmt->bind\_param("ss", $input['nombre'], $input['email']);

Vincula los parámetros a la consulta.

if (!$stmt->execute()) {

throw new Exception("Error al crear usuario: " . $stmt->error);

}

Ejecuta la consulta y maneja errores.

echo json\_encode([

'success' => true,

'id' => $stmt->insert\_id,

'message' => 'Usuario creado exitosamente'

]);

Devuelve respuesta JSON con el ID del nuevo usuario.

} catch (Exception $e) {

http\_response\_code(400);

echo json\_encode(['error' => $e->getMessage()]);

}

Maneja excepciones con código 400 y mensaje de error.

$stmt->close();

$conn->close();

exit;

Cierra conexiones y termina el script.

http\_response\_code(405);

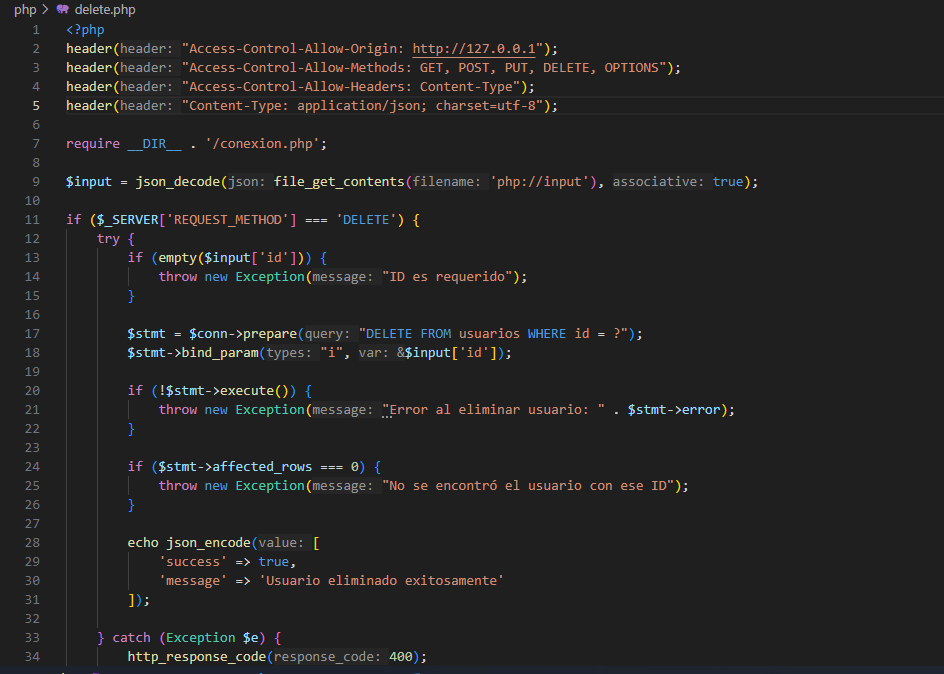
echo json\_encode(['error' => 'Método no permitido']);

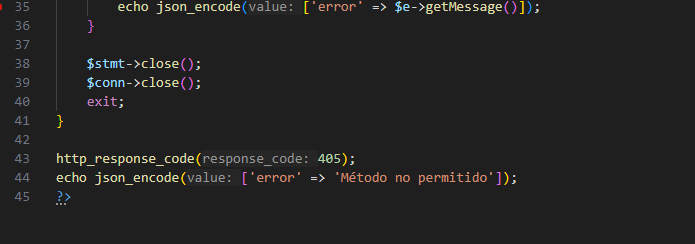
Respuesta para métodos no permitidos.

?>

Fin del script.

DELETE.PHP





<?php

// Headers similares a create.php

require \_\_DIR\_\_ . '/conexion.php';

$input = json\_decode(file\_get\_contents('php://input'), true);

Configuración inicial similar a create.php.

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

Verifica método DELETE.

try {

if (empty($input['id'])) {

throw new Exception("ID es requerido");

}

Valida que el ID no esté vacío.

$stmt = $conn->prepare("DELETE FROM usuarios WHERE id = ?");

$stmt->bind\_param("i", $input['id']);

Prepara y vincula consulta de eliminación.

if (!$stmt->execute()) {

throw new Exception("Error al eliminar usuario: " . $stmt->error);

}

if ($stmt->affected\_rows === 0) {

throw new Exception("No se encontró el usuario con ese ID");

}

Verifica ejecución y si se afectaron filas.

echo json\_encode([

'success' => true,

'message' => 'Usuario eliminado exitosamente'

]);

Respuesta de éxito.

} catch (Exception $e) {

http\_response\_code(400);

echo json\_encode(['error' => $e->getMessage()]);

}

Manejo de errores.

$stmt->close();

$conn->close();

exit;

}

Limpieza y salida.

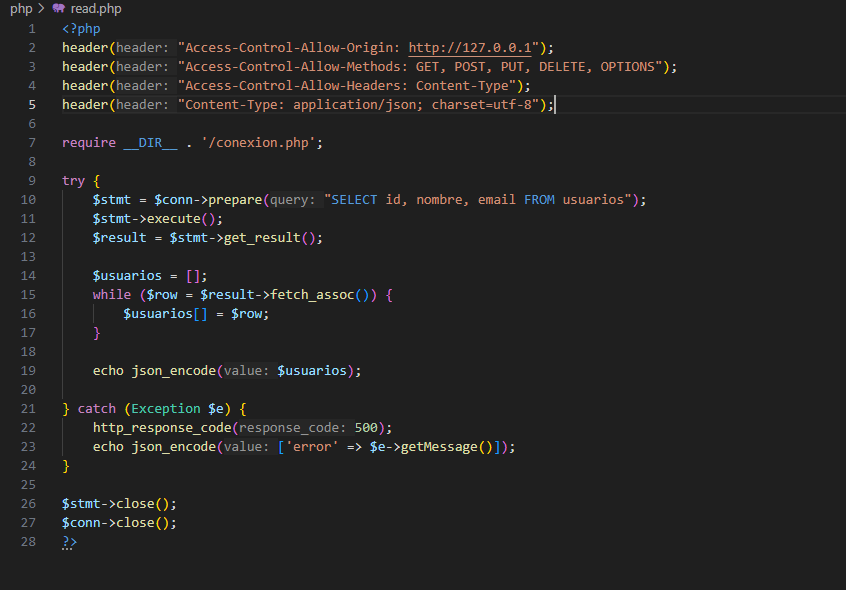
http\_response\_code(405);

echo json\_encode(['error' => 'Método no permitido']);

?>

Método no permitido.

READ.PHP



<?php

// Headers similares

require \_\_DIR\_\_ . '/conexion.php';

Configuración inicial.

try {

$stmt = $conn->prepare("SELECT id, nombre, email FROM usuarios");

$stmt->execute();

$result = $stmt->get\_result();

Prepara y ejecuta consulta SELECT.

$usuarios = [];

while ($row = $result->fetch\_assoc()) {

$usuarios[] = $row;

}

Recupera todos los usuarios en un array.

echo json\_encode($usuarios);

Devuelve los usuarios como JSON.

} catch (Exception $e) {

http\_response\_code(500);

echo json\_encode(['error' => $e->getMessage()]);

}

Manejo de errores con código 500.

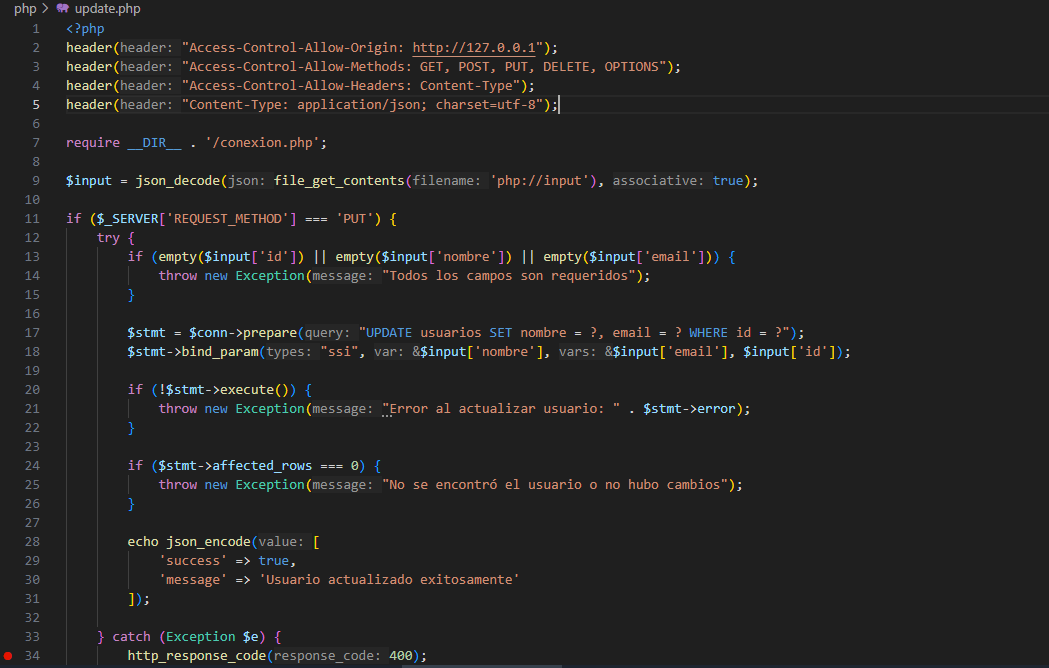
$stmt->close();

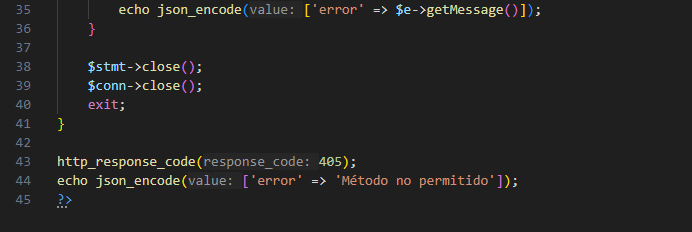
$conn->close();

?>

Limpieza.

UPDATE.PHP





<?php

// Headers similares

require \_\_DIR\_\_ . '/conexion.php';

$input = json\_decode(file\_get\_contents('php://input'), true);

Configuración inicial.

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

Verifica método PUT.

try {

if (empty($input['id']) || empty($input['nombre']) || empty($input['email'])) {

throw new Exception("Todos los campos son requeridos");

}

Valida todos los campos.

$stmt = $conn->prepare("UPDATE usuarios SET nombre = ?, email = ? WHERE id = ?");

$stmt->bind\_param("ssi", $input['nombre'], $input['email'], $input['id']);

Prepara y vincula consulta UPDATE.

if (!$stmt->execute()) {

throw new Exception("Error al actualizar usuario: " . $stmt->error);

}

if ($stmt->affected\_rows === 0) {

throw new Exception("No se encontró el usuario o no hubo cambios");

}

Verifica ejecución y cambios.

echo json\_encode([

'success' => true,

'message' => 'Usuario actualizado exitosamente'

]);

Respuesta de éxito.

} catch (Exception $e) {

http\_response\_code(400);

echo json\_encode(['error' => $e->getMessage()]);

}

Manejo de errores.

$stmt->close();

$conn->close();

exit;

}

Limpieza.

http\_response\_code(405);

echo json\_encode(['error' => 'Método no permitido']);

?>

Método no permitido.