# Fase 2 - 5 Mejoras a aplicar.

## 1️⃣ Credenciales fuera del código fuente

Las contraseñas, usuarios y nombres de BD nunca deben estar en archivos `.php`.

Si suben su código a GitHub (o hacen un backup), exponen la BD a cualquiera.

**⚠️ Antes (riesgo crítico):**

```php
// ❌ index.php
$con = mysqli_connect('localhost', 'admin', '1234567', 'huerta_db');
```

**✅ Después (buena práctica):**

```php
// ✅ config/db.php
$pass = getenv('DB_PASS'); // ← viene de .env
$con = mysqli_connect($host, $user, $pass, $name);
```

+ Archivo `.env` (en raíz, NO en GitHub):

```env
DB_PASS=1234567
```

+ Archivo `.gitignore` (SÍ en GitHub):

```gitignore
.env
```

**✅ Verificación:**

- No hay `'1234567'` ni `'admin'` en ningún `.php`.
- Existe `.env.example` (plantilla limpia) y `.gitignore` con `.env`.

---

## 2️⃣ Sentencias preparadas en TODAS las consultas con datos del usuario

Usar `mysqli_prepare()` + `bind_param()` en lugar de concatenar variables en SQL.

Evita inyección SQL - el ataque #1 en aplicaciones web (OWASP Top 10).

### ⚠️ Antes (vulnerable):

```php
// ❌ nuevo.php
$sql = "INSERT INTO cultivos (nombre) VALUES ('$_POST[nombre]')";
mysqli_query($con, $sql);
```

### ✅ Después (seguro):

```php
// ✅ nuevo.php
$stmt = mysqli_prepare($con, "INSERT INTO cultivos (nombre) VALUES (?)");
mysqli_stmt_bind_param($stmt, "s", $_POST['nombre']);
mysqli_stmt_execute($stmt);
```

### ✅ Verificación:

- Todas las consultas que usan `$_GET`, `$_POST` o variables de usuario usan `prepare()`.
- No hay comillas simples alrededor de variables en SQL (`'$var'` → prohibido).

---

## 3️⃣ Separación de responsabilidades: lógica ≠ BD ≠ vista

- `logic/*.php`: solo funciones puras (sin `echo`, sin `mysqli`, sin `$_POST`).
- Scripts (`index.php`, `nuevo.php`): solo coordinan (BD + lógica + HTML).

Facilita pruebas, mantenimiento y prepara para POO/MVC.

### ⚠️ Antes:

```php
// ❌ index.php
function guardarCultivo($n) {
    $con = conectar();
    mysqli_query($con, "INSERT ...");
    echo "✅ Guardado";
}
```

### ✅ Después (estructurado):

```php
// ✅ logic/cultivos.php
```

```
function esTipoValido(string $tipo): bool { ... }

// ✅ nuevo.php
require_once 'logic/cultivos.php';
if (esTipoValido($_POST['tipo'])) {
    // ... preparar y ejecutar INSERT
}
```

✅ **Verificación:**

- Carpeta `logic/` con archivos descriptivos (`cultivos.php`, no `funciones.php`).
- Ninguna función en `logic/` accede a `$_POST`, `mysqli` o imprime HTML.

---

## 4️⃣ Manejo seguro de errores: nunca mostrar detalles técnicos al usuario

Registrar errores con `error_log()`, pero mostrar mensajes genéricos al usuario.

Evita filtrar información sensible (estructura de BD, rutas, etc.).

⚠️ **Antes (peligroso):**

php

```
// ❌ nuevo.php
die("Error: " . mysqli_error($con)); // ← revela tabla, columnas, etc.
```

✅ **Después (seguro):**

php

```
// ✅ nuevo.php
if (!mysqli_stmt_execute($stmt)) {
    error_log("Fallo inserción: " . mysqli_stmt_error($stmt));
    header("Location: nuevo.php?error=No se pudo guardar.");
    exit;
}
```

✅ **Verificación:**

- Cero `die()`, `echo` o `print` con `mysqli_error()`, `connect_error()`, etc.
- Todos los errores críticos usan `error_log()`.

---

## 5️⃣ Comentarios estratégicos en el código modificado

Añadir comentarios solo en los bloques o líneas que hayan cambiado, explicando:

- *¿Qué se corrigió?*
- *¿Por qué es mejor?*

Demuestra comprensión (no solo copia) y facilita la corrección.

**✅ Ejemplo correcto:**

```php
// ✅ Nuevo: uso sentencia preparada para evitar inyección SQL
// Antes: "INSERT ... VALUES ('$_POST[nombre]')" → vulnerable
$stmt = mysqli_prepare($con, "INSERT INTO cultivos (nombre) VALUES (?)");
mysqli_stmt_bind_param($stmt, "s", $nombre);
```

**✅ Verificación:**

- Al menos 3 comentarios explicativos en cambios clave (ej: prepared statements, .env, separación lógica).
- Los comentarios usan lenguaje claro y técnico (no “arreglé esto”).

---

# Guía paso a paso para la implementación:

| Paso | Acción | Archivos afectados |
|---|---|---|
| 1️⃣ | Crear .env y .gitignore | Raíz del proyecto |
| 2️⃣ | Mover conexión a config/db.php | `config/db.php` |
| 3️⃣ | Crear carpeta logic/ y mover funciones | `logic/cultivos.php, etc.` |
| 4️⃣ | Reemplazar TODAS las consultas por sentencias preparadas | `index.php, nuevo.php, editar.php` |
| 5️⃣ | Reemplazar die(mysqli_error()) por error_log() + redirección | Todos los scripts con BD |
| 6️⃣ | Añadir comentarios en los cambios clave | Todos los archivos modificados |

---

## 🌳 Estructura de proyecto ideal: `mi-huerta/`

```
mi-huerta/
├── .env                    ← 🔐 (solo local)
├── .env.example            ← 📄 (SÍ en GitHub)
├── .gitignore              ← ✅ con ".env"
├── config/
│   └── db.php              ← conexión segura
├── logic/
│   ├── cultivos.php        ← funciones puras
│   └── riego.php
├── index.php               ← listado
├── nuevo.php               ← alta segura
└── README.md
```

---

### Detalles clave por archivo

| Archivo | Contenido esencial | Buena práctica reforzada |
|---|---|---|
| `.env` | `env DB_HOST=localhost DB_USER=admin DB_PASS=1234567 DB_NAME=huerta_db` | ✅ Credenciales fuera del código |
| `.env.example` | Mismo contenido que .env, pero con valores genéricos (tu_contraseña) | ✅ Compartible en repositorio |
| `.gitignore` | `.env` | ✅ Evita subir secretos a GitHub |
| `config/db.php` | Usa getenv(), mysqli_connect(), error_log(), utf8mb4 | ✅ Único punto de conexión |
| `logic/cultivos.php` | Solo funciones: entrada → salida. Sin $_POST, sin echo, sin mysqli | ✅ Separación de responsabilidades |
| `index.php` | `require_once de db.php + cultivos.php; prepared statement para SELECT; htmlspecialchars() en` | ✅ Seguridad en capas |

| | salida | |
|---|---|---|
| nuevo.php | Validación → prepared INSERT → redirección (PRG) → error_log() en fallo | ✅ Patrones profesionales |

## Sanitización: Entradas vs. Salidas

| Contexto | Qué hacer | Función segura | Ejemplo en nuevo.php / index.php |
|---|---|---|---|
| 🟢 Entrada: validación (¿Es válido?) | Verificar tipo/rango | filter_input() | php $dias = filter_input(INPUT_POST, 'dias', FILTER_VALIDATE_INT); if ($dias === false |
| 🟡 Entrada: BD segura (¿Es seguro para SQL?) | ✅ Solo sentencias preparadas ❌ Nunca mysqli_real_escape_string() | mysqli_prepare() + bind_param() | php $stmt = mysqli_prepare($con, "INSERT INTO cultivos (nombre) VALUES (?)"); mysqli_stmt_bind_param($stmt, "s", $nombre); |
| 🔴 Salida: HTML (¿Es seguro para el navegador?) | Escapar caracteres | htmlspecialchars($x, ENT_QUOTES, 'UTF-8') | php echo "<td>" . htmlspecialchars($fila['nombre']) . "</td>"; |
| 🔵 Salida: URLs (¿Es seguro en enlaces?) | Codificar espacios/símbolos | urlencode() | php header("Location: nuevo.php?error=" . urlencode("Datos inválidos")); |

⚠️ NUNCA usar (obsoletos/inseguros):

- `FILTER_SANITIZE_STRING` (deprecated)
- `addslashes()`, `mysql_real_escape_string()`
- `FILTER_UNSAFE_RAW` como *única* medida

---

## ✅ Checklist:

| ✅ | Item |
|---|---|
| 1 | `.gitignore contiene .env` |
| 2 | No hay '1234567' en ningún .php |
| 3 | Todas las consultas con datos del usuario usan prepare() + bind_param() |
| 4 | Todos los echo de datos dinámicos usan htmlspecialchars() |

| | |
|---|---|
| 5 | `logic/cultivos.php no tiene $_POST, mysqli, ni echo` |
| 6 | Los errores usan error_log(), no die(mysqli_error()) |
| 7 | Hay al menos 3 comentarios explicando cambios clave (ej: “Preparada para evitar inyección SQL”) |
| 8 | `UTF-8 en HTML (<meta charset="UTF-8">) y utf8mb4 en BD (mysqli_set_charset($con, "utf8mb4"))` |

## Trucos de terminal (verifica rápido)

```
# ¿Contraseñas expuestas?
grep -r "CocoRock\|1234567" . --include="*.php"

# ¿Consultas inseguras?
grep -r "mysqli_query.*\$_\|VALUES.*\$_" . --include="*.php"

# ¿XSS posible?
grep -r "echo.*\$_" . --include="*.php"
```