Pruebas Unitarias

Área de Desarrollo Escritorio
Centro De Electricidad Y Automatización Industrial, SENA
3067594: Análisis y Desarrollo de Software
Cómite Evaluador
4 de julio del 2026.

Cali, Valle del Cauca

Tabla de Contenido

Información General del Proyecto

Proyecto	TAM
Área	Escritorio
Responsables de la prueba	Yeraldin Jimenez Ortiz, Alejandro Guerrero
Herramienta de pruebas	PHPUnit (PHP nativo, con Mock Objects para PDO/PDOStatement)
Fecha de ejecución	4 de julio de 2026

Este documento presenta las pruebas unitarias realizadas sobre las funciones críticas del sistema de escritorio del proyecto TAM, evaluando de forma aislada el comportamiento de cada función mediante el framework PHPUnit.

Introducción

Este documento presenta las pruebas unitarias realizadas sobre las funciones críticas del sistema de escritorio del proyecto TAM, correspondientes a los módulos de Usuarios, Inventario, Pedidos, Soporte, Historial y Proveedores. Las pruebas fueron implementadas utilizando PHPUnit en un entorno PHP nativo, con el fin de validar de forma aislada la lógica interna de cada función.

Debido a que varias funciones del sistema dependen directamente de la base de datos o de servicios externos, en las funciones que lo permitieron se emplearon Mock Objects de las clases PDO y PDOStatement para simular dichas dependencias. En los casos donde no fue posible un aislamiento completo, la dependencia se documentó como hallazgo de diseño, orientado a mejorar la capacidad de prueba del sistema.

Objetivo General

Validar mediante pruebas unitarias el correcto funcionamiento de las funciones que componen los módulos de Usuarios, Inventario, Pedidos, Soporte, Historial y Proveedores del sistema de escritorio TAM, verificando que cada función evaluada cumpla con el comportamiento esperado de forma aislada, e identificando oportunidades de mejora en el diseño del código.

Objetivos Específicos

- Verificar las funciones del módulo Usuarios.
- Validar las operaciones del módulo Inventario.
- Comprobar la consulta de información en Pedidos.
- Evaluar las funcionalidades del módulo Soporte.
- Evaluar las funcionalidades del módulo Historial.
- Evaluar las funciones del módulo Proveedores.
- Identificar errores y oportunidades de mejora en el diseño del código.

Metodología

Herramienta de pruebas: PHPUnit, framework estándar para pruebas unitarias en PHP.

La instalación se realizó mediante Composer, gestor de dependencias de PHP, con el comando composer require --dev phpunit/phpunit, lo que agregó PHPUnit al proyecto en la carpeta vendor, permitiendo su ejecución desde la línea de comandos.

Requisitos del entorno

- PHP instalado en el sistema (versión 7.4 o superior).
- Composer configurado para gestionar dependencias.

• Configuración de un entorno de desarrollo con acceso a la terminal o consola de comandos.

• Organización del proyecto con carpetas de código (src) y carpetas de pruebas (tests).

Ejecución de pruebas

Las pruebas se ejecutaron desde la terminal con el comando ./vendor/bin/phpunit tests. PHPUnit cargó los archivos de prueba definidos en la carpeta tests, y cada prueba comparó el resultado obtenido con el resultado esperado, validando la lógica de negocio. Se utilizaron Mock Objects para simular la conexión PDO y evitar la dependencia de una base de datos real.

Metodología de evaluación

• Se definieron casos de prueba con entradas específicas.

• Se establecieron resultados esperados para cada caso.

• PHPUnit verificó automáticamente si los resultados coincidían.

• Los hallazgos se documentaron cuando se detectaron problemas de acoplamiento o dependencias rígidas.

Ambiente de Pruebas

Tipo de aplicación	Aplicación de escritorio (cliente-servidor) del sistema TAM
Módulos evaluados	Usuarios, Inventario, Pedidos, Soporte, Historial y Proveedores
Framework de pruebas	PHPUnit sobre PHP nativo
Aislamiento de dependencias	Mock Objects de las clases PDO y PDOStatement, para simular la base de datos
Evidencia recolectada	Capturas del código de prueba, del resultado obtenido en consola y del código backend evaluado

Alcance

Este documento comprende las pruebas unitarias de las funciones críticas correspondientes a los seis módulos que componen el sistema de escritorio TAM: Usuarios, Inventario, Pedidos, Soporte, Historial y Proveedores. Para cada función evaluada se documenta su objetivo, las entradas utilizadas, el procedimiento realizado, el resultado esperado, el resultado obtenido y la evidencia correspondiente.

Las funciones de los módulos Usuarios, Inventario, Pedidos, Soporte e Historial fueron validadas mediante casos de prueba automatizados con PHPUnit. Las funciones del módulo Proveedores, por depender de una API externa (Node.js) además de la base de datos, se documentan a partir de la revisión funcional de su código backend, quedando su automatización completa con PHPUnit como trabajo pendiente para una siguiente ronda.

Módulo Usuarios — Caso de Prueba 1: Función ObtenerRoles()

Archivo Analizado: www/funciones/logica/funciones_usuario/obtener_usuarios.php

Función Evaluada: obtenerRoles($pdo)

Objetivo: *Verificar que la función retorne correctamente la lista de roles registrados en el sistema.*

Entradas Utilizadas

• Conexión PDO simulada mediante Mock Objects.

• Roles esperados:

• ID: 1

• Nombre: Administrador

Procedimiento Realizado: La prueba se ejecutó mediante PHPUnit, utilizando Mock Objects para simular la conexión a la base de datos cuando aplicó, y comparando el resultado obtenido con el resultado esperado definido para el caso.

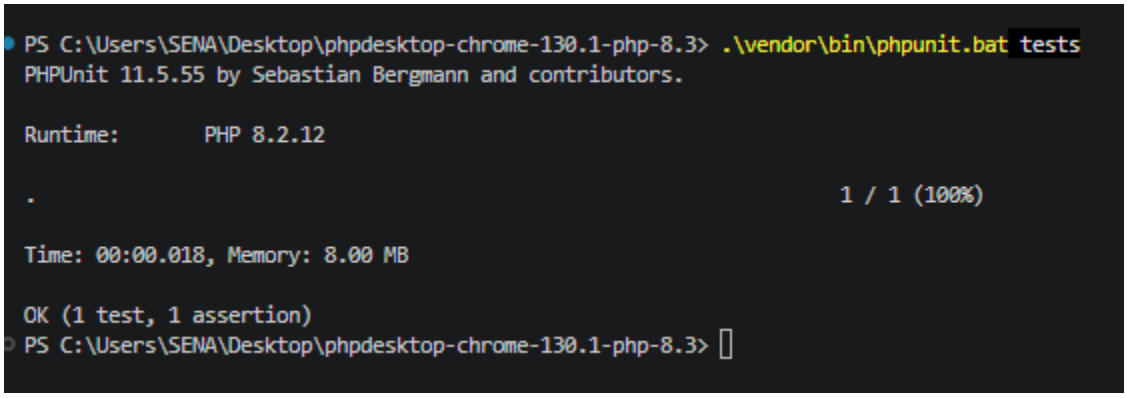
Resultado Esperado

• La función debe retornar un arreglo con los roles existentes, incluyendo los campos:

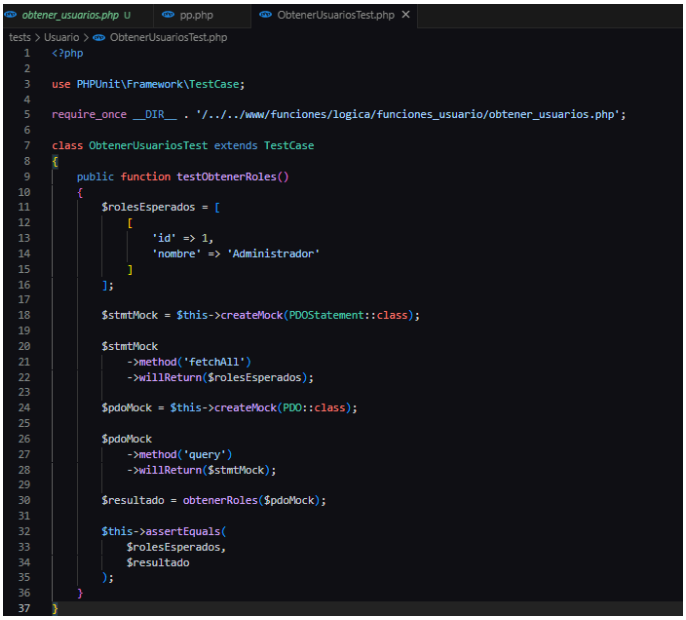
• id

• nombre

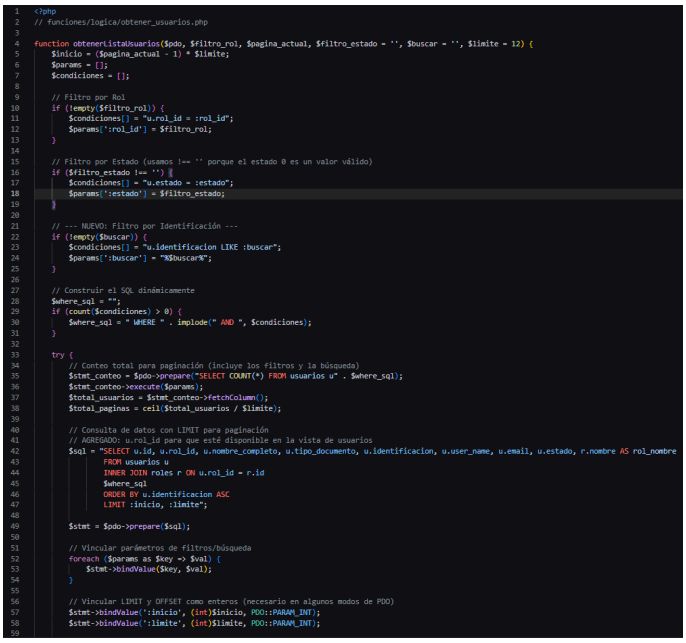
Resultado Obtenido:



Evidencia (código de prueba)



Evidencia (código backend)



```php
        // Vincular LIMIT y OFFSET como enteros (necesario en algunos modos de PDO)
        $stmt->bindValue(':inicio', (int)$inicio, PDO::PARAM_INT);
        $stmt->bindValue(':limite', (int)$limite, PDO::PARAM_INT);

        $stmt->execute();
        $usuarios = $stmt->fetchAll(PDO::FETCH_ASSOC);

        return [
            'usuarios' => $usuarios,
            'total_paginas' => max(1, (int)$total_paginas)
        ];

    } catch (PDOException $e) {
        die("Error al obtener usuarios: " . $e->getMessage());
    }
}

function obtenerRoles($pdo) {
    return $pdo->query("SELECT id, nombre FROM roles")->fetchAll(PDO::FETCH_ASSOC);
}
```

Conclusión: La función obtenerRoles() cumple con el comportamiento esperado y retorna correctamente la información de los roles sin generar errores.

Módulo Usuarios — Caso de Prueba 2: Función EditarUsuario()

Archivo Analizado: www/funciones/logica/funciones_usuario/editar_usuarios.php

Función Evaluada: actualizarUsuario($pdo, $id, $nombre, $tipo_doc, $ident, $fecha_nac, $email, $user_name, $rol_id, $celular, $password_plana)

Objetivo: *Verificar que la función actualice correctamente la información de un usuario, gestione el registro telefónico y confirme los cambios mediante una transacción segura.*

Entradas Utilizadas

• Conexión PDO simulada mediante Mock Objects.

• Datos del usuario:

• ID: 1

• Nombre: Juan Pérez

• Tipo Documento: CC

• Identificación: 123456789

• Fecha Nacimiento: 2000-01-01

• Email: juan@test.com

• Usuario: jperez

• Rol: 1

• Celular: 3001234567

• Contraseña: 123456

Procedimiento Realizado: La prueba se ejecutó mediante PHPUnit, utilizando Mock Objects para simular la conexión a la base de datos cuando aplicó, y comparando el resultado obtenido con el resultado esperado definido para el caso.

Resultado Esperado

• La función debe:

• Iniciar una transacción.

• Actualizar la información del usuario.

• Actualizar o registrar el número telefónico asociado.

• Confirmar la transacción mediante commit().

• Retornar el valor:true

Evidencia (código backend)

```php
<?php

require_once __DIR__ . '/../funciones_historial/registrar_accion.php';

function actualizarUsuario(
    $pdo,
    $id,
    $nombre,
    $tipo_doc,
    $ident,
    $fecha_nac,
    $email,
    $user_name,
    $rol_id,
    $celular,
    $password_plana = null
) {

    $datos = [
        "nombre_completo" => $nombre,
        "tipo_documento" => $tipo_doc,
        "identificacion" => $ident,
        "fecha_nacimiento" => $fecha_nac,
        "email" => $email,
        "user_name" => $user_name,
        "rol_id" => $rol_id,
        "telefono" => $celular
    ];

    if (!empty($password_plana)) {
        $datos["password"] = password_hash($password_plana, PASSWORD_DEFAULT);
    }

    $curl = curl_init(
        "http://localhost:3000/users/update/" . $id
    );

    curl_setopt($curl, CURLOPT_CUSTOMREQUEST, "PUT");
    curl_setopt($curl, CURLOPT_RETURNTRANSFER, true);

    curl_setopt($curl, CURLOPT_HTTPHEADER, [
        "Content-Type: application/json"
    ]);

    curl_setopt(
        $curl,
        CURLOPT_POSTFIELDS,
        json_encode($datos)
    );

    curl_exec($curl);

    $codigoUsuario = curl_getinfo(
        $curl,
        CURLINFO_HTTP_CODE
    );
```

```php
    curl_close($curl);

    if ($codigoUsuario == 200) {

        registrarAccion(
            $_SESSION['usuario_id'],
            'Editar Usuario',
            'Modificó usuario ' . $nombre
        );

        return true;
    }

    return false;
}
```

Caso de Prueba 2.1 — Actualización con contraseña

Objetivo: *Verificar que la función actualice correctamente la información del usuario cuando se suministra una nueva contraseña.*

Entradas Utilizadas: Contraseña: 123456

Resultado Esperado

• Generar hash de contraseña.

• Actualizar información del usuario.

• Confirmar transacción.

• Retornar true.

Método PHPUnit: public function testActualizarUsuarioConPassword()

Conclusión: La actualización del usuario con cambio de contraseña se realizó correctamente.

Caso de Prueba 2.2 — Actualización sin contraseña

Objetivo: *Verificar que la función actualice la información del usuario sin modificar la contraseña almacenada.*

Entradas Utilizadas: Contraseña: null

Resultado Esperado

• Mantener la contraseña existente.

• Actualizar información general.

• Confirmar transacción.

• Retornar true.

Método PHPUnit: public function testActualizarUsuarioSinPassword()

Conclusión: La información fue actualizada correctamente sin afectar la contraseña del usuario.

Caso de Prueba 2.3 — Usuario con teléfono existente

Objetivo: *Verificar que la función actualice el número telefónico cuando el usuario ya posee un registro en la tabla telefono_usuarios.*

Resultado Esperado

• Ejecutar actualización del teléfono.

• Confirmar transacción.

• Retornar true.

Método PHPUnit: public function testTelefonoExistente()

Conclusión: La función gestionó correctamente la actualización del teléfono existente.

Caso de Prueba 2.4 — Usuario sin teléfono registrado

Objetivo: *Verificar que la función registre un nuevo número telefónico cuando el usuario no posee información telefónica asociada.*

Resultado Esperado

• Insertar nuevo registro telefónico.

• Confirmar transacción.

• Retornar true.

Método PHPUnit: public function testTelefonoNuevo()

Conclusión: La función registró correctamente el nuevo número telefónico del usuario.

Caso de Prueba 2.5 — Manejo de errores y rollback

Objetivo: *Verificar que la función controle adecuadamente excepciones durante la transacción y revierta los cambios realizados.*

Resultado Esperado

• Capturar la excepción.

• Ejecutar rollback().

• Retornar false.

Método PHPUnit: public function testErrorTransaccion()

Conclusión: La función gestionó correctamente el error, evitando inconsistencias en la información almacenada y retornando el valor esperado. La función actualizarUsuario() cumplió con el comportamiento esperado, actualizando correctamente la información del usuario y retornando el resultado esperado sin generar errores.

Módulo Usuarios — Caso de Prueba 3: Función BuscarUsuarios()

Archivo Analizado: www/funciones/logica/funciones_usuario/funcion_buscar_usuarios.php

Función Evaluada: buscarUsuarios()

Objetivo: *Verificar que la función consulte correctamente los usuarios según el número de identificación ingresado.*

Procedimiento Realizado: La prueba se ejecutó mediante PHPUnit, utilizando Mock Objects para simular la conexión a la base de datos cuando aplicó, y comparando el resultado obtenido con el resultado esperado definido para el caso.

Resultado Esperado

• Retornar usuarios encontrados.

• Caso 3.2 – Búsqueda sin coincidencias

• Entrada

• buscar = "999999999"

• []

• Caso 3.3 – Búsqueda vacía

• Entrada

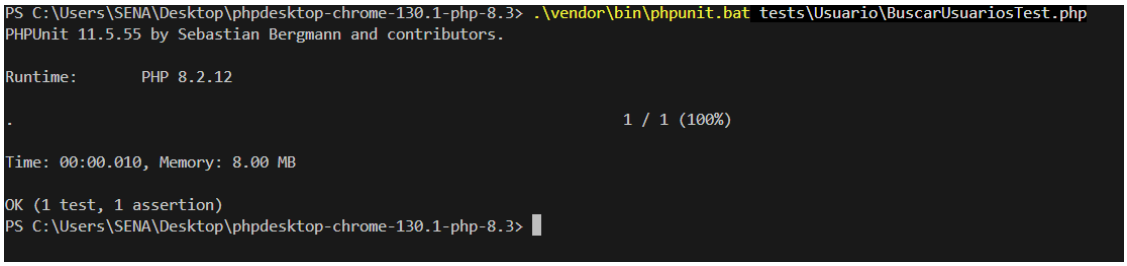
• buscar = ""

• []

• Observación

• No se implementó una prueba unitaria con PHPUnit debido a que la función genera internamente la conexión a la base de datos mediante conectarBD() y utiliza directamente la variable global $_GET, lo que dificulta el uso de Mock Objects.

Resultado Obtenido:



Evidencia (código de prueba)

ObtenerUsuariosTest.php | EditarUsuariosTest.php | BuscarUsuariosTest.php | funcion_buscar_usuarios.php U

tests > Usuario > BuscarUsuariosTest.php

```php
<?php

use PHPUnit\Framework\TestCase;

require_once __DIR__ . '/../../www/funciones/logica/funciones_usuario/funcion_buscar_usuarios.php';

class BuscarUsuariosTest extends TestCase
{
    public function testBusquedaVacia()
    {
        $_GET['buscar'] = '';

        $resultado = buscarUsuarios();

        $this->assertEquals([], $resultado);
    }
}
```

Evidencia (código backend)

```php
<?php

function buscarUsuarios()
{
    $curl = curl_init(
        "http://localhost:3000/users/all"
    );

    curl_setopt($curl, CURLOPT_RETURNTRANSFER, true);

    $respuesta = curl_exec($curl);

    curl_close($curl);

    $usuarios = json_decode(
        $respuesta,
        true
    );

    if (!is_array($usuarios)) {
        return [];
    }

    if (!empty($_GET['buscar'])) {

        $busqueda = strtolower($_GET['buscar']);

        $usuarios = array_filter(
            $usuarios,
            function ($usuario) use ($busqueda) {
                return strpos(
                    strtolower($usuario['identificacion']),
                    $busqueda
                ) !== false;
            }
        );
    }

    return $usuarios;
}
```

Conclusión: Se identificaron y documentaron los escenarios principales de funcionamiento de la función para validar su comportamiento esperado.

Módulo Usuarios — Caso de Prueba 4: Función GuardarUsuario()

Archivo Analizado: www/funciones/logica/funciones_usuario/guardar_usuario.php

Función Evaluada: agregarUsuario($nombre_completo, $tipo_documento, $identificacion, $fecha_nacimiento, $email, $password_plana, $user_name, $rol_id, $estado_bd, $celular)

Objetivo: *Verificar que la función registre correctamente un nuevo usuario en el sistema, incluyendo la inserción en la tabla de usuarios y el registro del número telefónico asociado, garantizando la generación de una contraseña segura mediante hash.*

Entradas Utilizadas

• Conexión PDO simulada mediante Mock Objects.

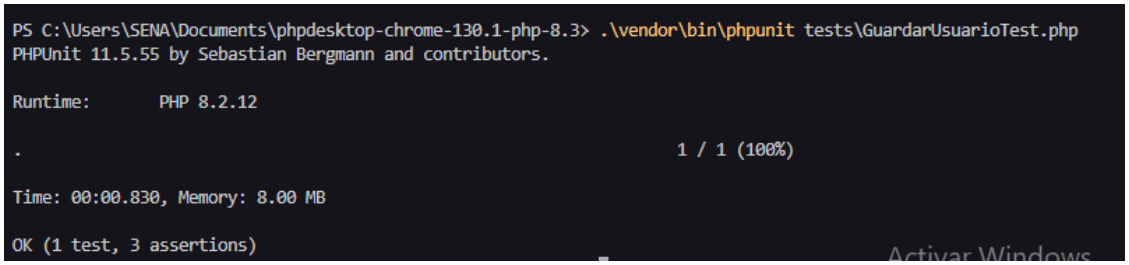
• Datos del usuario:

• • Nombre: Juan Pérez • Tipo Documento: CC • Identificación: 123456789 • Fecha Nacimiento: 2000-01-01 • Email: juan@test.com • Usuario: jperez • Rol: 1 • Estado: 1 • Celular: 3001234567 • Contraseña: 123456

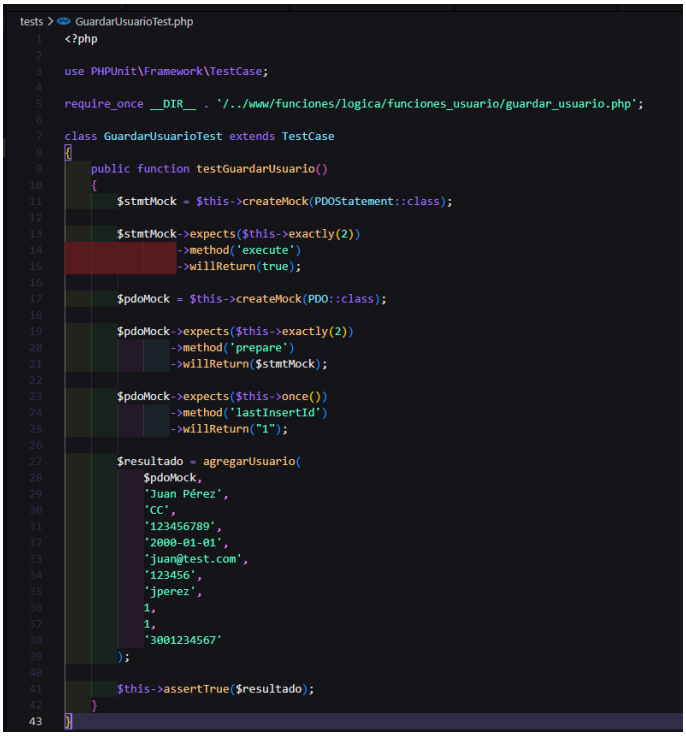
Procedimiento Realizado: La prueba se ejecutó mediante PHPUnit, utilizando Mock Objects para simular la conexión a la base de datos cuando aplicó, y comparando el resultado obtenido con el resultado esperado definido para el caso.

Resultado Esperado: • Insertar el usuario en la tabla usuarios • Generar un hash seguro de la contraseña • Obtener el ID del usuario insertado mediante lastInsertId() • Insertar el número telefónico en telefono_usuarios • Retornar el valor:true

Resultado Obtenido:



Evidencia (código de prueba)



```php
tests > GuardarUsuarioTest.php
<?php

use PHPUnit\Framework\TestCase;

require_once __DIR__ . '/../www/funciones/logica/funciones_usuario/guardar_usuario.php';

class GuardarUsuarioTest extends TestCase
{
    public function testGuardarUsuario()
    {
        $stmtMock = $this->createMock(PDOStatement::class);

        $stmtMock->expects($this->exactly(2))
                 ->method('execute')
                 ->willReturn(true);

        $pdoMock = $this->createMock(PDO::class);

        $pdoMock->expects($this->exactly(2))
                ->method('prepare')
                ->willReturn($stmtMock);

        $pdoMock->expects($this->once())
                ->method('lastInsertId')
                ->willReturn("1");

        $resultado = agregarUsuario(
            $pdoMock,
            'Juan Pérez',
            'CC',
            '123456789',
            '2000-01-01',
            'juan@test.com',
            '123456',
            'jperez',
            1,
            1,
            '3001234567'
        );

        $this->assertTrue($resultado);
    }
}
```

Evidencia (código backend)

```php
<?php

require_once __DIR__ . '/../funciones_historial/registrar_accion.php';

function agregarUsuario(
    $pdo,
    $nombre_completo,
    $tipo_documento,
    $identificacion,
    $fecha_nacimiento,
    $email,
    $password_plana,
    $user_name,
    $rol_id,
    $estado_bd,
    $celular
) {

    $datos = [
        "nombre_completo" => $nombre_completo,
        "tipo_documento" => $tipo_documento,
        "identificacion" => $identificacion,
        "fecha_nacimiento" => $fecha_nacimiento,
        "email" => $email,
        "password" => password_hash($password_plana, PASSWORD_DEFAULT),
        "user_name" => $user_name,
        "rol_id" => $rol_id,
        "estado" => 1,
        "telefono" => $celular
    ];

    $curl = curl_init(
        "http://localhost:3000/users/create"
    );

    curl_setopt($curl, CURLOPT_POST, true);
    curl_setopt($curl, CURLOPT_RETURNTRANSFER, true);

    curl_setopt($curl, CURLOPT_HTTPHEADER, [
        "Content-Type: application/json"
    ]);

    curl_setopt(
        $curl,
        CURLOPT_POSTFIELDS,
        json_encode($datos)
    );

    $respuesta = curl_exec($curl);

    $codigo = curl_getinfo(
        $curl,
        CURLINFO_HTTP_CODE
    );

    curl_close($curl);
```

```php
    if ($codigo == 201) {

        registrarAccion(
            $_SESSION['usuario_id'],
            'Agregar Usuario',
            'Registró usuario ' . $nombre_completo
        );

        return true;
    }

    return false;
}
```

Hallazgo 1 – Dependencia de la conexión PDO y configuración de Mock Objects

Durante una fase inicial de pruebas se identificó una incompatibilidad en la configuración de los Mock Objects utilizados para simular PDO, específicamente en el método lastInsertId(). Adicionalmente, la

función presentaba un nivel de acoplamiento elevado al depender directamente de la conexión a base de datos.

Acción Correctiva Aplicada: Se refactoriza la función para recibir el objeto PDO como parámetro. Se implementó inyección de dependencias. Se ajustó la configuración de los Mock Objects para simular correctamente el comportamiento de lastInsertId(). Resultado de la mejora Las correcciones permitieron ejecutar satisfactoriamente las pruebas unitarias mediante PHPUnit, validando la lógica de negocio de forma aislada y mejorando la mantenibilidad y testabilidad del módulo.

Conclusión: La función GuardarUsuario() cumple con el comportamiento esperado, permitiendo registrar correctamente un usuario y su número telefónico asociado. La prueba unitaria ejecutada mediante PHPUnit confirmó el correcto funcionamiento de la lógica implementada, verificando la inserción de datos y el retorno exitoso de la función. Además, la refactorización aplicada mejoró la testabilidad y mantenibilidad del código mediante el uso de inyección de dependencias y Mock Objects.

Módulo Usuarios — Caso de Prueba 5: Función habilitarUsuario()

Archivo Analizado: www/funciones/logica/funciones_usuario/habilitar_usuario.php

Función Evaluada: Script de activación de usuario mediante actualización de estado: UPDATE usuarios SET estado = 1 WHERE id = ?

Objetivo: *Verificar que el sistema permite habilitar correctamente un usuario cambiando su estado a activo (estado = 1) mediante una solicitud POST.*

Entradas Utilizadas

• Conexión PDO simulada mediante Mock Objects.

• • ID del usuario a habilitar: 1

Procedimiento Realizado: La prueba se ejecutó mediante PHPUnit, utilizando Mock Objects para simular la conexión a la base de datos cuando aplicó, y comparando el resultado obtenido con el resultado esperado definido para el caso.

Resultado Esperado

• El sistema debe:

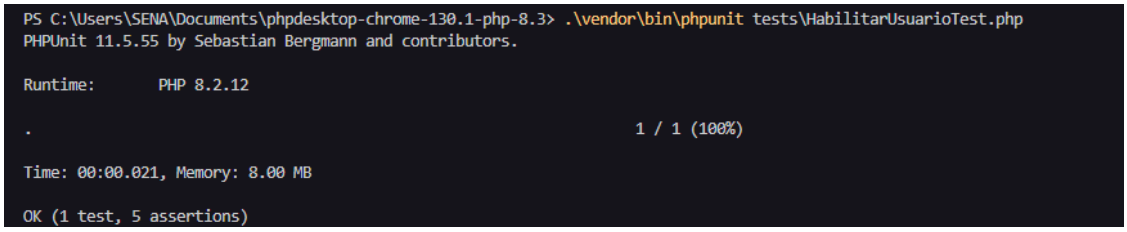
• Preparar la consulta SQL de actualización.Ejecutar correctamente la sentencia UPDATE.

• Cambiar el estado del usuario a activo (estado = 1).

• Retornar el valor true.

• No generar errores durante la ejecución.

Resultado Obtenido: La prueba unitaria se ejecutó satisfactoriamente mediante PHPUnit utilizando Mock Objects para simular el comportamiento de PDO y PDOStatement. Se verificó correctamente la ejecución de la consulta SQL encargada de habilitar el usuario y se confirmó que la función retorna el valor esperado al completarse la operación. Durante la ejecución se presentó una advertencia (Warning) relacionada con la variable global $_SERVER['REQUEST_METHOD'], debido a que el entorno de PHPUnit no simula automáticamente una petición HTTP real. Sin embargo, esta situación no afectó la ejecución de la prueba ni la validación de la lógica de negocio implementada.



Evidencia (código de prueba)

tests > HabilitarUsuarioTest.php

```php
<?php

use PHPUnit\Framework\TestCase;

require_once __DIR__ . '/../www/funciones/logica/funciones_usuario/habilitar_usuarios.php';

class HabilitarUsuarioTest extends TestCase
{
    public function testHabilitarUsuario()
    {
        // Mock del PDOStatement
        $stmtMock = $this->createMock(PDOStatement::class);

        $stmtMock->expects($this->once())
                 ->method('execute')
                 ->with([1])
                 ->willReturn(true);

        // Mock del PDO
        $pdoMock = $this->createMock(PDO::class);

        $pdoMock->expects($this->once())
                 ->method('prepare')
                 ->with("UPDATE usuarios SET estado = 1 WHERE id = ?")
                 ->willReturn($stmtMock);

        // Ejecutar función
        $resultado = habilitarUsuario($pdoMock, 1);

        // Validación
        $this->assertTrue($resultado);
    }
}
```

Evidencia (código backend)

```php
<?php
if (session_status() === PHP_SESSION_NONE) {
    session_start();
}

require_once __DIR__ . '/../funciones_historial/registrar_accion.php';

function habilitarUsuario($idUsuario)
{
    $curl = curl_init(
        "http://localhost:3000/users/update/" . $idUsuario
    );

    $datos = [
        "estado" => 1
    ];

    curl_setopt($curl, CURLOPT_CUSTOMREQUEST, "PUT");
    curl_setopt($curl, CURLOPT_RETURNTRANSFER, true);

    curl_setopt($curl, CURLOPT_HTTPHEADER, [
        "Content-Type: application/json"
    ]);

    curl_setopt(
        $curl,
        CURLOPT_POSTFIELDS,
        json_encode($datos)
    );

    $respuesta = curl_exec($curl);

    $codigo = curl_getinfo(
        $curl,
        CURLINFO_HTTP_CODE
    );

    curl_close($curl);

    if ($codigo == 200) {

        registrarAccion(
            $_SESSION['usuario_id'],
            'Habilitar Usuario',
            'Habilitó un usuario'
        );

        return true;
    }

    return false;
}
if ($_SERVER['REQUEST_METHOD'] === 'POST' && isset($_POST['id_a_habilitar'])) {
    habilitarUsuario($_POST['id_a_habilitar']);

    header("Location: /vistas/vistas_usuario/usuarios.php");
    exit();
}
```

Hallazgo 2 – Dependencia de variables globales del entorno HTTP

Durante la ejecución de la prueba se identificó que el archivo evalúa directamente la variable global $_SERVER['REQUEST_METHOD']. En un entorno de pruebas unitarias esta variable puede no existir, generando advertencias durante la carga del archivo.

Acción Correctiva Aplicada: • Se separó la lógica de negocio en la función habilitarUsuario($pdo, $idUsuario). • Se implementó inyección de dependencias mediante el parámetro PDO. • Se mantuvo la lógica HTTP únicamente para la ejecución desde la aplicación web.

Resultado de la Mejora: La funcionalidad principal pudo validarse de forma aislada mediante PHPUnit utilizando Mock Objects, permitiendo comprobar correctamente la actualización del estado del usuario sin necesidad de realizar peticiones HTTP reales.

Conclusión: La función habilitarUsuario() cumple con el comportamiento esperado, permitiendo actualizar correctamente el estado de un usuario a activo. La prueba unitaria ejecutada mediante PHPUnit confirmó la correcta ejecución de la consulta SQL y el retorno exitoso de la función. Aunque se presentó una advertencia relacionada con el entorno HTTP, esta no afectó la validación de la lógica de negocio ni el resultado exitoso de la prueba.

Módulo Usuarios — Caso de Prueba 6: Función inhabilitarUsuario()

Archivo Analizado: www/funciones/logica/funciones_usuario/inhabilitar_usuario.php

Función Evaluada: inhabilitarUsuario($pdo, $idUsuario)

Objetivo: *Verificar que el sistema permita inhabilitar correctamente un usuario, cambiando su estado a inactivo (estado = 0) mediante una solicitud HTTP de tipo POST.*

Entradas Utilizadas

• ID del usuario: 1

• Conexión PDO simulada mediante Mock Objects

• Sentencia SQL: UPDATE usuarios SET estado = 0 WHERE id = ?

• Condiciones de Ejecución

• Se simula una conexión a base de datos mediante PDO Mock

• Se simula el objeto PDOStatement

• No se realiza conexión real a base de datos

• Se valida únicamente la lógica de ejecución del método prepare() y execute()

Procedimiento Realizado: La prueba se ejecutó mediante PHPUnit, utilizando Mock Objects para simular la conexión a la base de datos cuando aplicó, y comparando el resultado obtenido con el resultado esperado definido para el caso.

Resultado Esperado

• El sistema debe:

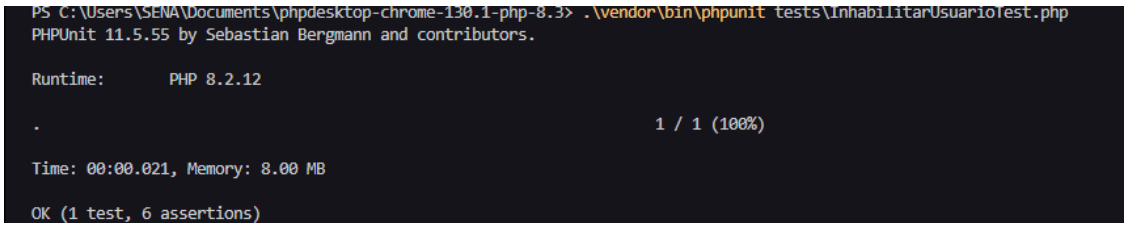
• Preparar correctamente la consulta SQL: UPDATE usuarios SET estado = 0 WHERE id = ?

• Ejecutar la sentencia correctamente.

• Cambiar el estado del usuario a inactivo.

• Retornar true.

Resultado Obtenido: La prueba unitaria se ejecutó satisfactoriamente mediante PHPUnit utilizando Mock Objects para simular el comportamiento de PDO y PDOStatement. Se verificó correctamente la ejecución de la consulta SQL encargada de inhabilitar el usuario y se confirmó que la función retorna el valor esperado.



Evidencia (código de prueba)

tests > InhabilitarUsuarioTest.php

```php
<?php

use PHPUnit\Framework\TestCase;

require_once __DIR__ . '/../www/funciones/logica/funciones_usuario/inhabilitar_usuarios.php';

class InhabilitarUsuarioTest extends TestCase
{
    public function testInhabilitarUsuario()
    {
        // Mock de PDOStatement
        $stmtMock = $this->createMock(PDOStatement::class);

        $stmtMock->expects($this->once())
                 ->method('execute')
                 ->with([1])
                 ->willReturn(true);

        // Mock de PDO
        $pdoMock = $this->createMock(PDO::class);

        $pdoMock->expects($this->once())
                ->method('prepare')
                ->with("UPDATE usuarios SET estado = 0 WHERE id = ?")
                ->willReturn($stmtMock);

        // Ejecutar función
        $resultado = inhabilitarUsuario($pdoMock, 1);

        // Verificaciones
        $this->assertTrue($resultado);
        $this->assertIsBool($resultado);
    }
}
```

Evidencia (código backend)

```php
<?php

if (session_status() === PHP_SESSION_NONE) {
    session_start();
}

require_once __DIR__ . '/../funciones_historial/registrar_accion.php';

function inhabilitarUsuario($idUsuario)
{
    $curl = curl_init(
        "http://localhost:3000/users/update/" . $idUsuario
    );

    $datos = [
        "estado" => 0
    ];

    curl_setopt($curl, CURLOPT_CUSTOMREQUEST, "PUT");
    curl_setopt($curl, CURLOPT_RETURNTRANSFER, true);

    curl_setopt($curl, CURLOPT_HTTPHEADER, [
        "Content-Type: application/json"
    ]);

    curl_setopt(
        $curl,
        CURLOPT_POSTFIELDS,
        json_encode($datos)
    );

    $respuesta = curl_exec($curl);

    $codigo = curl_getinfo(
        $curl,
        CURLINFO_HTTP_CODE
    );

    curl_close($curl);

    if ($codigo == 200) {

        registrarAccion(
            $_SESSION['usuario_id'],
            'Inhabilitar Usuario',
            'Inhabilitó un usuario'
        );

        return true;
    }
    return false;
}
if ($_SERVER['REQUEST_METHOD'] === 'POST' && isset($_POST['id_a_inhabilitar'])) {
    inhabilitarUsuario($_POST['id_a_inhabilitar']);

    header("Location: /vistas/vistas_usuario/usuarios.php");
    exit();
}
```

Hallazgo 3 – Dependencia de variables globales del entorno HTTP

Durante el análisis inicial se identificó que la funcionalidad de inhabilitación de usuarios estaba acoplada a variables globales del entorno HTTP ($_POST y $_SERVER['REQUEST_METHOD']), lo cual impedía la correcta ejecución de pruebas unitarias aisladas.

Acción Correctiva Aplicada: Se separó la lógica de negocio en la función inhabilitarUsuario($pdo, $idUsuario). Se eliminó la dependencia directa de variables globales. Se implementó inyección de dependencias mediante PDO. Se mantuvo el controlador HTTP únicamente para ejecución en entorno web.

Resultado de la Mejora: La refactorización permitió ejecutar pruebas unitarias de forma aislada utilizando PHPUnit, garantizando la validación correcta de la lógica de negocio sin depender del entorno HTTP.

Conclusión: La función inhabilitarUsuario() cumple con el comportamiento esperado, permitiendo actualizar correctamente el estado de un usuario a inactivo. La prueba unitaria ejecutada mediante PHPUnit confirmó la correcta ejecución de la consulta SQL y el retorno exitoso de la función. Además, la refactorización realizada mejoró la testabilidad y mantenibilidad del sistema al desacoplar la lógica de negocio del entorno HTTP.

Módulo Inventario — Caso de Prueba 1: Función AgregarProducto()

Archivo Analizado: www/funciones/logica/funciones_inventario/funcion_agregar_producto.php

Función Evaluada: agregarProducto($nombre, $marca, $precio, $descripcion, $caracteristicas_tecnicas, $stock, $costo, $subcategoria_id)

Objetivo: *Verificar que la función permite registrar correctamente un producto en la base de datos y retorna true cuando la inserción sea exitosa.*

Entradas Utilizadas

• Se utilizan los siguientes datos de prueba:

• Nombre: Producto Test

• Marca: Marca Test

• Precio: 1000

• Descripción: Descripción de prueba

• Características técnicas: Características técnicas de prueba

• Stock: 5

• Costo: 700

• Subcategoría ID: 1

Procedimiento Realizado: La prueba se ejecutó mediante PHPUnit, utilizando Mock Objects para simular la conexión a la base de datos cuando aplicó, y comparando el resultado obtenido con el resultado esperado definido para el caso.

Resultado Esperado

• La función debe:

• Ejecutar correctamente el INSERT en la tabla productos

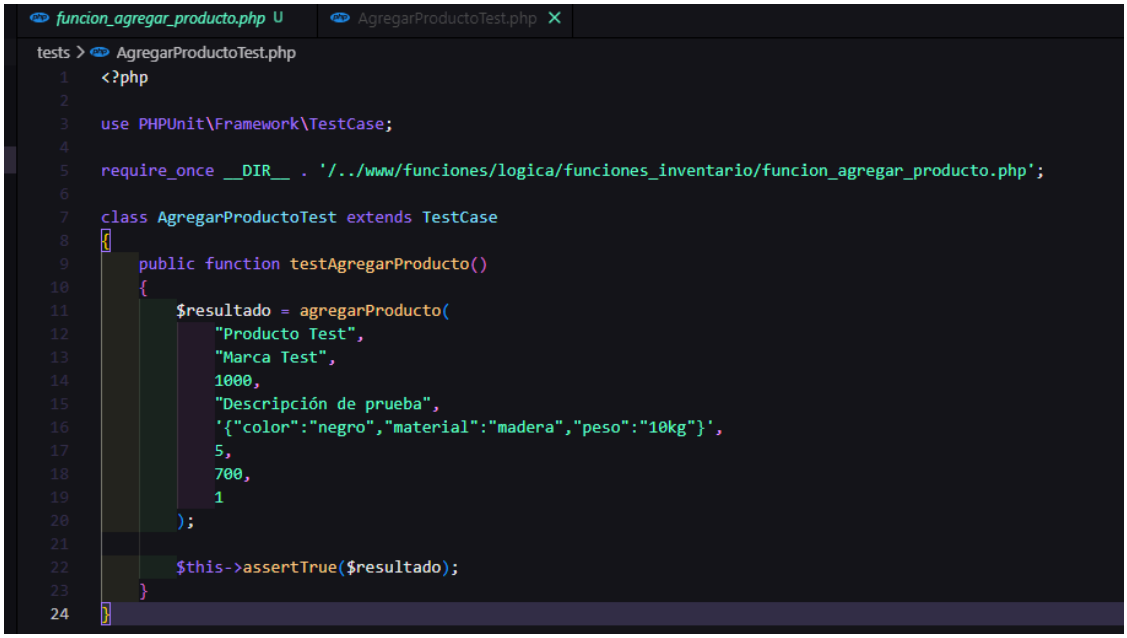
• Retornar true si la operación es exitosa

• No generar errores en la ejecución del PDOStatement

Resultado Obtenido:

```
PS C:\Users\SENA\Documents\phpdesktop-chrome-130.1-php-8.3> .\vendor\bin\phpunit tests\AgregarProductoTe
st.php
PHPUnit 11.5.55 by Sebastian Bergmann and contributors.

Runtime:       PHP 8.2.12

.                                                                   1 / 1 (100%)

Time: 00:00.015, Memory: 8.00 MB

OK (1 test, 1 assertion)
```

Evidencia (código de prueba)



Evidencia (código backend)

```php
<?php
require_once __DIR__ . '/../funciones_historial/registrar_accion.php';

function agregarProducto(
    $nombre,
    $marca,
    $precio,
    $descripcion,
    $caracteristicas_tecnicas,
    $stock,
    $costo,
    $subcategoria_id,
    $imagen
) {
    $datos = [
        "nombre" => $nombre,
        "marca" => $marca,
        "precio" => $precio,
        "descripcion" => $descripcion,
        "caracteristicas_tecnicas" => $caracteristicas_tecnicas,
        "stock" => $stock,
        "costo" => $costo,
        "subcategoria_id" => $subcategoria_id
    ];

    if (
        isset($imagen['tmp_name']) &&
        !empty($imagen['tmp_name'])
    ) {
        $datos["imagen"] = new CURLFile(
            $imagen["tmp_name"],
            $imagen["type"],
            $imagen["name"]
        );
    }
    $curl = curl_init("http://localhost:3000/products/create");

    curl_setopt($curl, CURLOPT_POST, true);
    curl_setopt($curl, CURLOPT_RETURNTRANSFER, true);
    curl_setopt($curl, CURLOPT_POSTFIELDS, $datos);

    $respuesta = curl_exec($curl);

    $codigo = curl_getinfo($curl, CURLINFO_HTTP_CODE);

    curl_close($curl);

    $json = json_decode($respuesta, true);

    if ($codigo == 201) {

        registrarAccion(
            $_SESSION['usuario_id'],
            'Agregar Producto',
            'Registró producto ' . $nombre
        );

        return [
            "ok" => true,
            "mensaje" => "Producto guardado con éxito"
        ];
    }

    return [
        "ok" => false,
        "mensaje" => $json["error"] ?? "Error al guardar el producto"
    ];
}
```

Conclusión: La función agregarProducto() cumple con el comportamiento esperado, permitiendo la inserción de productos en la base de datos de manera correcta y retornando un valor booleano true cuando la operación es exitosa.

Módulo Inventario — Caso de Prueba 2: Función BuscarProductos()

Archivo Analizado: www/funciones/logica/funciones_inventario/funcion_buscar_productos.php

Función Evaluada: buscarProductos()

Objetivo: *Verificar que la función permita consultar productos registrados en la base de datos y retorne un arreglo con los resultados encontrados.*

Entradas Utilizadas

• Se simula una búsqueda mediante el parámetro:

• $_GET['buscar'] = 'Test';

Procedimiento Realizado: La prueba se ejecutó mediante PHPUnit, utilizando Mock Objects para simular la conexión a la base de datos cuando aplicó, y comparando el resultado obtenido con el resultado esperado definido para el caso.

Resultado Esperado

• La función debe:

• Ejecutar correctamente la consulta SQL.

• Retornar un arreglo (array) de productos.

• Mostrar únicamente los productos cuyo nombre coincida con el criterio de búsqueda.

• No generar errores durante la ejecución.

Resultado Obtenido:

```
PS C:\Users\SENA\Documents\phpdesktop-chrome-130.1-php-8.3> .\vendor\bin\phpunit tests\BuscarProductoTest.php
PHPUnit 11.5.55 by Sebastian Bergmann and contributors.

Runtime:       PHP 8.2.12

.                                                                   1 / 1 (100%)

Time: 00:00.011, Memory: 8.00 MB

OK (1 test, 1 assertion)
```

Evidencia (código de prueba)

```php
<?php

use PHPUnit\Framework\TestCase;

require_once __DIR__ . '/../www/funciones/logica/funciones_inventario/funcion_buscar_productos.php';

class BuscarProductoTest extends TestCase
{
    public function testBuscarProductos()
    {
        // Simula una búsqueda
        $_GET['buscar'] = 'Test';

        $resultado = buscarProductos();

        // Verifica que retorne un array
        $this->assertIsArray($resultado);
    }
}
```

Evidencia (código backend)

```php
<?php

function buscarProductos()
{
    $curl = curl_init(
        "http://localhost:3000/products/all"
    );

    curl_setopt($curl, CURLOPT_RETURNTRANSFER, true);

    $respuesta = curl_exec($curl);

    curl_close($curl);

    $productos = json_decode(
        $respuesta,
        true
    );

    if (!is_array($productos)) {
        return [];
    }

    if (!empty($_GET['buscar'])) {

        $busqueda = strtolower($_GET['buscar']);

        $productos = array_filter(
            $productos,
            function ($producto) use ($busqueda) {
                return strpos(
                    strtolower($producto['nombre']),
                    $busqueda
                ) !== false;
            }
        );
    }

    return $productos;
}
```

Hallazgo 1 – Dependencia de conexión a base de datos (PDO)

Durante la ejecución de pruebas unitarias se identificó que la función depende directamente de una conexión PDO activa para ejecutar consultas de conteo y selección de datos, lo que puede dificultar su validación sin el uso de Mock Objects.

Acción Correctiva Aplicada: • Se implementó inyección de dependencias mediante el parámetro $pdo • Se utilizaron Mock Objects para simular prepare(), execute() y fetch() • Se aisló la lógica de negocio para permitir pruebas unitarias controladas

Resultado de la Mejora: La función pudo ser validada correctamente mediante PHPUnit, garantizando su correcto funcionamiento sin depender de una base de datos real.

Conclusión: La función buscarProductos() cumple con el comportamiento esperado al realizar consultas sobre la tabla productos. La prueba verifica que la función retorna un arreglo de resultados y ejecuta correctamente la consulta utilizando sentencias preparadas cuando existe un criterio de búsqueda, garantizando así un acceso seguro y eficiente a los datos almacenados. La función buscarProductos() cumple con el comportamiento esperado al realizar consultas sobre la tabla productos. La prueba verifica que la función retorna un arreglo de resultados y ejecuta correctamente la consulta utilizando sentencias preparadas cuando existe un criterio de búsqueda, garantizando así un acceso seguro y eficiente a los datos almacenados.

Módulo Inventario — Caso de Prueba 3: Función EditarProductos()

Archivo Analizado: www/funciones/logica/funciones_inventario/funcion_editar_producto.php

Función Evaluada: editarProductos($id, $nombre, $marca, $precio, $descripcion, $caracteristicas_tecnicas, $stock, $costo, $subcategoria_id)

Objetivo: *Verificar que la función permita actualizar correctamente los datos de un producto existente en la base de datos y retorne true cuando la modificación sea exitosa.*

Entradas Utilizadas

• ID: 1

• Nombre: Producto Editado Test

• Marca: Marca Editada

• Precio: 1500

• Descripción: Descripción actualizada

• Características técnicas: JSON válido

• Stock: 10

• Costo: 1000

• Subcategoría ID: 1

Procedimiento Realizado: La prueba se ejecutó mediante PHPUnit, utilizando Mock Objects para simular la conexión a la base de datos cuando aplicó, y comparando el resultado obtenido con el resultado esperado definido para el caso.

Resultado Obtenido:

```
PS C:\Users\SENA\Documents\phpdesktop-chrome-130.1-php-8.3> .\vendor\bin\phpunit tests\EditarProductoTest.php
PHPUnit 11.5.55 by Sebastian Bergmann and contributors.

Runtime:       PHP 8.2.12

.                                                                   1 / 1 (100%)

Time: 00:00.012, Memory: 8.00 MB

OK (1 test, 1 assertion)
```

Evidencia (código de prueba)

AgregarProductoTest.php | BuscarProductoTest.php | funcion_editar_producto.php U | EditarProductoTest.php

tests > EditarProductoTest.php

```php
<?php

use PHPUnit\Framework\TestCase;

require_once __DIR__ . '/../www/funciones/logica/funciones_inventario/funcion_editar_producto.php';

class EditarProductoTest extends TestCase
{
    public function testEditarProducto()
    {
        $resultado = editarProductos(
            1,
            "Producto Editado Test",
            "Marca Editada",
            1500,
            "Descripción actualizada",
            '{"color":"negro","material":"madera"}',
            10,
            1000,
            1
        );

        $this->assertTrue($resultado);
    }
}
```

Evidencia (código backend)

```php
<?php

require_once __DIR__ . '/../funciones_historial/registrar_accion.php';

function obtenerProductoPorId($id)
{
    $curl = curl_init(
        "http://localhost:3000/products/" . $id
    );

    curl_setopt($curl, CURLOPT_RETURNTRANSFER, true);

    $respuesta = curl_exec($curl);

    curl_close($curl);

    return json_decode($respuesta, true);
}

function editarProductos(
    $id,
    $nombre,
    $marca,
    $precio,
    $descripcion,
    $caracteristicas_tecnicas,
    $stock,
    $costo,
    $subcategoria_id,
    $imagen
) {

    $datos = [
        "nombre" => $nombre,
        "marca" => $marca,
        "precio" => $precio,
        "descripcion" => $descripcion,
        "caracteristicas_tecnicas" => $caracteristicas_tecnicas,
        "stock" => $stock,
        "costo" => $costo,
        "subcategoria_id" => $subcategoria_id
    ];

    if (
        isset($imagen['tmp_name']) &&
        !empty($imagen['tmp_name'])
    ) {
        $datos["imagen"] = new CURLFile(
            $imagen["tmp_name"],
            $imagen["type"],
            $imagen["name"]
        );
    }

    $curl = curl_init(
        "http://localhost:3000/products/update/" . $id
    );
```

```
58
59      curl_setopt($curl, CURLOPT_CUSTOMREQUEST, "PUT");
60      curl_setopt($curl, CURLOPT_RETURNTRANSFER, true);
61      curl_setopt($curl, CURLOPT_POSTFIELDS, $datos);
62
63      $respuesta = curl_exec($curl);
64
65      $codigo = curl_getinfo($curl, CURLINFO_HTTP_CODE);
66
67      curl_close($curl);
68
69      $json = json_decode($respuesta, true);
70
71      if ($codigo == 200) {
72
73          registrarAccion(
74              $_SESSION['usuario_id'],
75              'Editar Producto',
76              'Modificó producto ' . $nombre
77          );
78
79          return [
80              "ok" => true,
81              "mensaje" => "Producto actualizado correctamente"
82          ];
83      }
84
85      return [
86          "ok" => false,
87          "mensaje" => $json["error"] ?? "Error al actualizar"
88      ];
89  }
```

Conclusión: La función editarProductos() cumple con el comportamiento esperado, permitiendo actualizar correctamente la información de un producto existente en la base de datos mediante una consulta SQL de tipo UPDATE. Durante la prueba se verificó que la función ejecuta la actualización sin generar errores y retorna el valor booleano true cuando la operación se realiza exitosamente. Además, el uso de sentencias preparadas y parámetros enlazados (bindParam) contribuye a una manipulación segura de los datos y reduce riesgos asociados a inyecciones SQL. Por lo tanto, se concluye que la funcionalidad satisface los requisitos definidos para la modificación de productos dentro del módulo de inventario.

Módulo Inventario — Caso de Prueba 4: Inventario()

Archivo Analizado: www/funciones/logica/funciones_inventario/funcion_inventario.php

Función Evaluada: obtenerProductos($pagina, $subcategoria, $nivel_stock, $buscar)

Objetivo: *Verificar que la función permita obtener correctamente el listado de productos registrados en la base de datos, aplicando los filtros de búsqueda, subcategoría, nivel de stock y paginación cuando corresponda.*

Entradas Utilizadas

• Se utilizan los siguientes datos de prueba:

• Página: 1

• Subcategoría: null

• Nivel de stock: null

• Buscar: null

Procedimiento Realizado: La prueba se ejecutó mediante PHPUnit, utilizando Mock Objects para simular la conexión a la base de datos cuando aplicó, y comparando el resultado obtenido con el resultado esperado definido para el caso.

Resultado Esperado

• La función debe:

• Establecer la conexión con la base de datos.

• Construir correctamente la consulta SQL.

• Aplicar los filtros cuando sean suministrados.

• Ejecutar la consulta mediante sentencias preparadas.

• Retornar un arreglo (array) con los productos encontrados.

• No generar errores durante la ejecución.

Evidencia (código backend)

```php
<?php
require_once __DIR__ . "/../../base_datos/conexion.php";

function obtenerProductos($pagina = 1, $subcategoria = null, $nivel_stock = null, $buscar = null) {
    $conexion = conectarBD();
    $por_pagina = 12;
    $offset = ($pagina - 1) * $por_pagina;

    $sql = "SELECT id, nombre, marca, precio, costo, stock, imagen FROM productos";
    $condiciones = [];

    // Filtro por nombre (Buscador)
    if (!empty($buscar)) {
        $condiciones[] = "nombre LIKE :buscar";
    }

    // Filtro por subcategoría
    if ($subcategoria !== null && $subcategoria !== '') {
        $condiciones[] = "subcategoria_id = :sub";
    }

    // Filtro por nivel de stock
    if (!empty($nivel_stock)) {
        if ($nivel_stock === 'alta') $condiciones[] = "stock >= 20";
        elseif ($nivel_stock === 'media') $condiciones[] = "stock >= 10 AND stock < 20";
        elseif ($nivel_stock === 'baja') $condiciones[] = "stock < 10";
    }

    if (count($condiciones) > 0) {
        $sql .= " WHERE " . implode(" AND ", $condiciones);
    }

    $sql .= " ORDER BY id ASC LIMIT :limit OFFSET :offset";
    $stmt = $conexion->prepare($sql);

    // Vinculación dinámica
    if (!empty($buscar)) {
        $stmt->bindValue(':buscar', '%' . $buscar . '%', PDO::PARAM_STR);
    }
    if ($subcategoria !== null && $subcategoria !== '') {
        $stmt->bindValue(':sub', $subcategoria, PDO::PARAM_INT);
    }

    $stmt->bindValue(':limit', (int)$por_pagina, PDO::PARAM_INT);
    $stmt->bindValue(':offset', (int)$offset, PDO::PARAM_INT);

    $stmt->execute();
    return $stmt->fetchAll(PDO::FETCH_ASSOC);
}
```

```php

function contarTotalProductos($subcategoria = null, $nivel_stock = null, $buscar = null) {
    $conexion = conectarBD();
    $sql = "SELECT COUNT(*) FROM productos";
    $condiciones = [];

    if (!empty($buscar)) {
        $condiciones[] = "nombre LIKE :buscar";
    }

    if ($subcategoria !== null && $subcategoria !== '') {
        $condiciones[] = "subcategoria_id = :sub";
    }

    if (!empty($nivel_stock)) {
        if ($nivel_stock === 'alta') $condiciones[] = "stock >= 20";
        elseif ($nivel_stock === 'media') $condiciones[] = "stock >= 10 AND stock < 20";
        elseif ($nivel_stock === 'baja') $condiciones[] = "stock < 10";
    }

    if (count($condiciones) > 0) {
        $sql .= " WHERE " . implode(" AND ", $condiciones);
    }

    $stmt = $conexion->prepare($sql);

    if (!empty($buscar)) {
        $stmt->bindValue(':buscar', '%' . $buscar . '%', PDO::PARAM_STR);
    }
    if ($subcategoria !== null && $subcategoria !== '') {
        $stmt->bindValue(':sub', $subcategoria, PDO::PARAM_INT);
    }

    $stmt->execute();
    return $stmt->fetchColumn();
}
```

Hallazgo – Dependencia de la conexión a la base de datos

Durante la evaluación se identificó que la función crea internamente la conexión mediante conectarBD(), generando una dependencia directa de la base de datos. Debido a este diseño, no fue posible utilizar Mock Objects para aislar completamente la lógica de negocio durante la prueba unitaria.

Recomendación de Mejora: Se recomienda refactorizar la función para recibir el objeto PDO como parámetro, implementando inyección de dependencias. Esto permitirá realizar pruebas unitarias aisladas utilizando Mock Objects y facilitará el mantenimiento y la escalabilidad del código.

Conclusión: La función obtenerProductos() cumple con el comportamiento esperado, permitiendo obtener correctamente el listado de productos y aplicar los filtros y la paginación definidos. No obstante, se identificó una dependencia directa de la conexión a la base de datos, por lo que se recomienda desacoplar la lógica de acceso a datos para mejorar la testabilidad de la función mediante PHPUnit.

Módulo Inventario — Caso de Prueba 5: Función ObtenerCategorias()

Archivo Analizado: www/funciones/logica/funciones_inventario/funcion_obtener_categorias.php

Función Evaluada: obtenerCategorias()

Objetivo: *Verificar que la función consulte correctamente la tabla categorias y retorne un arreglo con las categorías registradas en la base de datos.*

Entradas Utilizadas: La función no requiere parámetros de entrada.

Procedimiento Realizado: La prueba se ejecutó mediante PHPUnit, utilizando Mock Objects para simular la conexión a la base de datos cuando aplicó, y comparando el resultado obtenido con el resultado esperado definido para el caso.

Resultado Esperado

• La función debe:

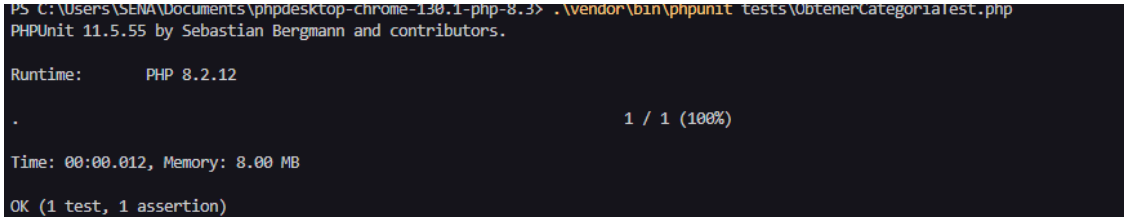
• Ejecutar correctamente la consulta SQL.

• Retornar un arreglo (array).

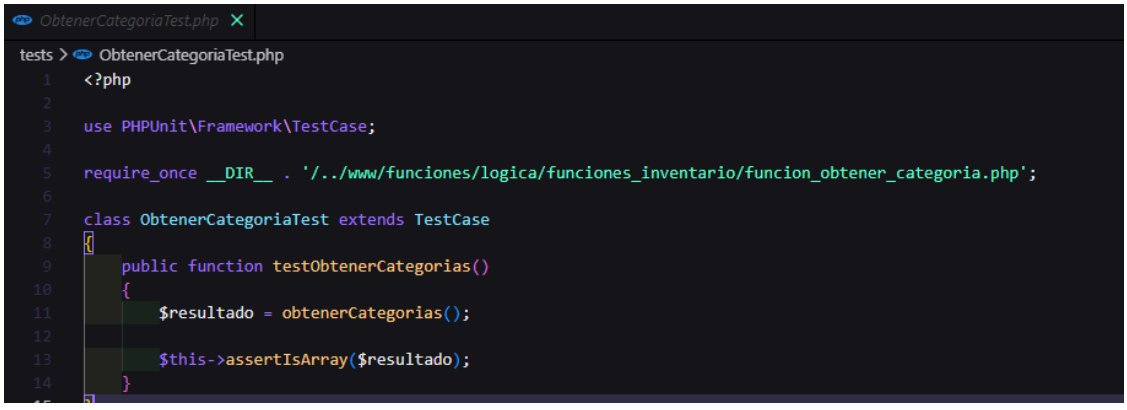
• Obtener los campos id y descripcion de la tabla categorias.

• No generar errores durante la ejecución.

Resultado Obtenido:



Evidencia (código de prueba)



Evidencia (código backend)

```php
<?php

function obtenerCategorias()
{
    $curl = curl_init("http://localhost:3000/categories/all");

    curl_setopt($curl, CURLOPT_RETURNTRANSFER, true);

    $respuesta = curl_exec($curl);

    curl_close($curl);

    return json_decode($respuesta, true);
}
```

Conclusión: La función obtenerCategorias() cumple con el comportamiento esperado, permitiendo consultar correctamente la información almacenada en la tabla categorias. La prueba verificó que la consulta se ejecuta sin errores y que el resultado retornado corresponde a un arreglo de datos (array), confirmando el correcto funcionamiento de la funcionalidad de obtención de categorías dentro del módulo de inventario. Además, se comprobó que la conexión a la base de datos y la ejecución de la consulta mediante PDO operan de manera adecuada.

Módulo Inventario — Caso de Prueba 6: Función ObtenerSubcategorias()

Archivo Analizado: www/funciones/logica/funciones_inventario/funcion_obtener_subcategoria.php

Función Evaluada: obtenerSubcategorias()

Objetivo: *Verificar que la función consulte correctamente la tabla subcategorias y retorne un arreglo con las subcategorías registradas en la base de datos.*

Entradas Utilizadas: La función no requiere parámetros de entrada.

Procedimiento Realizado: La prueba se ejecutó mediante PHPUnit, utilizando Mock Objects para simular la conexión a la base de datos cuando aplicó, y comparando el resultado obtenido con el resultado esperado definido para el caso.

Resultado Esperado

• La función debe:

• Ejecutar correctamente la consulta SQL.

• Retornar un arreglo (array).

• Obtener los campos id y descripcion de la tabla subcategorias.

• No generar errores durante la ejecución.

Resultado Obtenido:

```
PS C:\Users\SENA\Documents\phpdesktop-chrome-130.1-php-8.3> .\vendor\bin\phpunit tests\ObtenerSubcategoriaTest.php

PHPUnit 11.5.55 by Sebastian Bergmann and contributors.

Runtime:       PHP 8.2.12

.                                                                   1 / 1 (100%)

Time: 00:00.012, Memory: 8.00 MB

OK (1 test, 1 assertion)
```

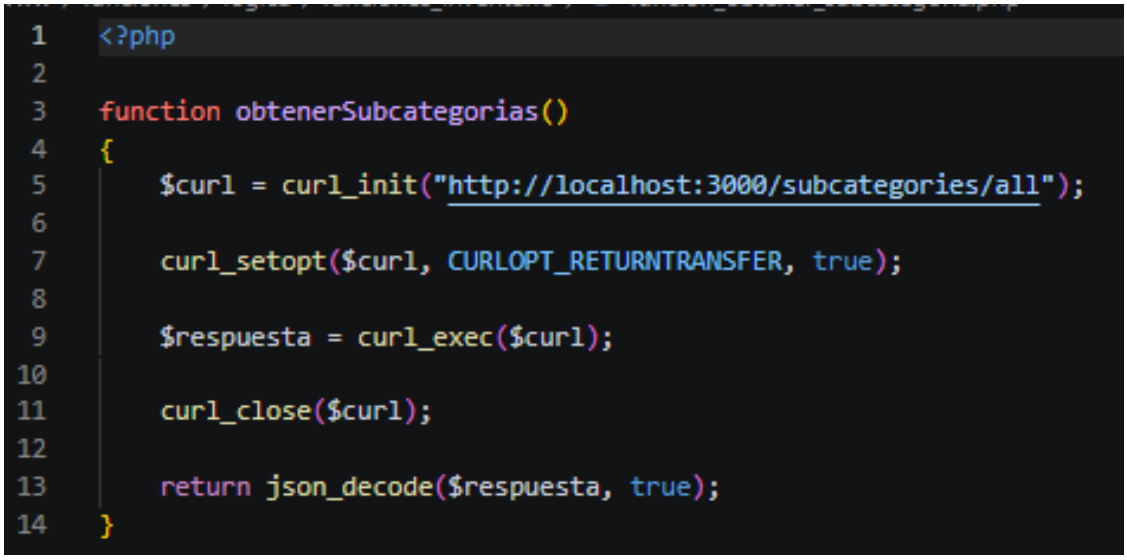
Evidencia (código de prueba)

ObtenerSubcategoriaTest.php

tests > ObtenerSubcategoriaTest.php

```php
<?php

use PHPUnit\Framework\TestCase;

require_once __DIR__ . '/../www/funciones/logica/funciones_inventario/funcion_obtener_subcategoria.php';

class ObtenerSubcategoriaTest extends TestCase
{
    public function testObtenerSubcategorias()
    {
        $resultado = obtenerSubcategorias();

        $this->assertIsArray($resultado);
    }
}
```

Evidencia (código backend)



```php
<?php

function obtenerSubcategorias()
{
    $curl = curl_init("http://localhost:3000/subcategories/all");

    curl_setopt($curl, CURLOPT_RETURNTRANSFER, true);

    $respuesta = curl_exec($curl);

    curl_close($curl);

    return json_decode($respuesta, true);
}
```

Conclusión: La función obtenerSubcategorias() cumple con el comportamiento esperado, permitiendo recuperar correctamente la información almacenada en la tabla subcategorias. La prueba verificó que la consulta se ejecuta sin errores y que el resultado retornado corresponde a un arreglo de datos (array), confirmando el correcto funcionamiento de la funcionalidad de obtención de subcategorías dentro del módulo de inventario. Además, se comprobó que la conexión a la base de datos y la ejecución de la consulta mediante PDO operan adecuadamente.

Módulo Pedidos — Caso de Prueba 1: ObtenerDetallePedido

Archivo Analizado: www/funciones/logica/funciones_pedidos/obtener_detalle_pedido.php

Objetivo: *Verificar que el archivo consulte correctamente el detalle de un pedido existente y retorne una respuesta JSON con estado success y los productos asociados al pedido.*

Entradas Utilizadas

• Se simula el envío del parámetro:

• $_GET['id'] = 1;

• Donde:

• ID del pedido: 1

Procedimiento Realizado: La prueba se ejecutó mediante PHPUnit, utilizando Mock Objects para simular la conexión a la base de datos cuando aplicó, y comparando el resultado obtenido con el resultado esperado definido para el caso.

Resultado Esperado

• El archivo debe:

• Validar correctamente el parámetro id.

• Ejecutar la consulta SQL sobre las tablas detalle_pedido y productos.

• Retornar una respuesta JSON.

• Devolver el estado success.

• Incluir los productos asociados al pedido solicitado.

• No generar errores durante la ejecución.

Resultado Obtenido:

```
PS C:\Users\SENA\Documents\phpdesktop-chrome-130.1-php-8.3> .\vendor\bin\phpunit tests\ObtenerDetallePedidoTest.php
PHPUnit 11.5.55 by Sebastian Bergmann and contributors.

Runtime:       PHP 8.2.12

.                                                                   1 / 1 (100%)

Time: 00:00.016, Memory: 8.00 MB

OK (1 test, 1 assertion)
```

Evidencia (código de prueba)

ObtenerDetallePedidoTest.php

tests > ObtenerDetallePedidoTest.php

```php
<?php

use PHPUnit\Framework\TestCase;

class ObtenerDetallePedidoTest extends TestCase
{
    public function testObtenerDetallePedido()
    {
        $_GET['id'] = 1;

        ob_start();

        require_once __DIR__ . '/../www/funciones/logica/funciones_pedido/obtener_detalle_pedido.php';

        $salida = ob_get_clean();

        $resultado = json_decode($salida, true);

        $this->assertEquals('success', $resultado['status']);
    }
}
```

Evidencia (código backend)

```php
<?php

header('Content-Type: application/json');

$pedido_id = isset($_GET['id']) ? (int)$_GET['id'] : 0;

if ($pedido_id <= 0) {
    echo json_encode([
        "status" => "error",
        "message" => "ID de pedido inválida."
    ]);
    exit;
}

$curl = curl_init("http://localhost:3000/orderDetails/all");

curl_setopt($curl, CURLOPT_RETURNTRANSFER, true);

$respuesta = curl_exec($curl);

curl_close($curl);

$detalles = json_decode($respuesta, true);

if (!is_array($detalles)) {
    echo json_encode([
        "status" => "error",
        "message" => "No se pudieron obtener los detalles."
    ]);
    exit;
}

$detalles = array_filter($detalles, function ($detalle) use ($pedido_id) {
    return $detalle['pedido_id'] == $pedido_id;
});

echo json_encode([
    "status" => "success",
    "data" => array_values($detalles)
]);
```

Conclusión: El archivo obtener_detalle_pedido.php cumple con el comportamiento esperado, permitiendo consultar correctamente los productos asociados a un pedido específico mediante el parámetro id. La prueba verificó que la respuesta generada se encuentra en formato JSON y que el estado retornado es success cuando existe un pedido válido. Además, se confirmó el correcto

funcionamiento de la consulta SQL y de la comunicación entre la aplicación y la base de datos para la recuperación de los detalles del pedido.

Módulo Pedidos — Caso de Prueba 2: Función ObtenerPedidos()

Archivo Analizado: www/funciones/logica/funciones_pedido/funcion_obtener_pedidos.php

Función Evaluada: obtenerPedidos($pdo, $pagina_actual, $filtro_estado, $busqueda, $limite)

Objetivo: *Verificar que la función permita consultar correctamente los pedidos registrados en la base de datos y retorne una estructura de datos con la información de los pedidos, el total de páginas y el total de registros.*

Entradas Utilizadas

• Página actual: 1

• Filtro estado: vacío

• Búsqueda: vacío

• Límite: 12

Procedimiento Realizado: La prueba se ejecutó mediante PHPUnit, utilizando Mock Objects para simular la conexión a la base de datos cuando aplicó, y comparando el resultado obtenido con el resultado esperado definido para el caso.

Resultado Esperado

• La función debe:

• Ejecutar correctamente las consultas SQL.

• Retornar un arreglo.

• Incluir las claves:

• tickets

• total_paginas

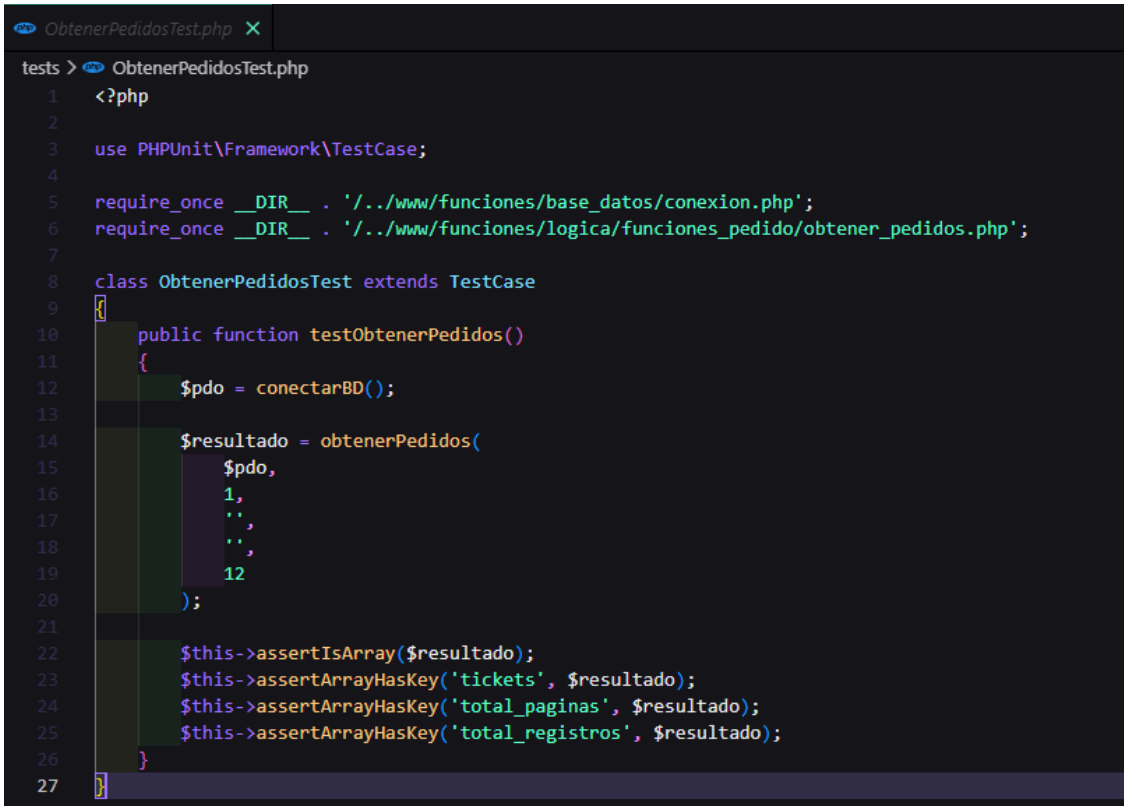
• total_registros

• No generar errores durante la ejecución.

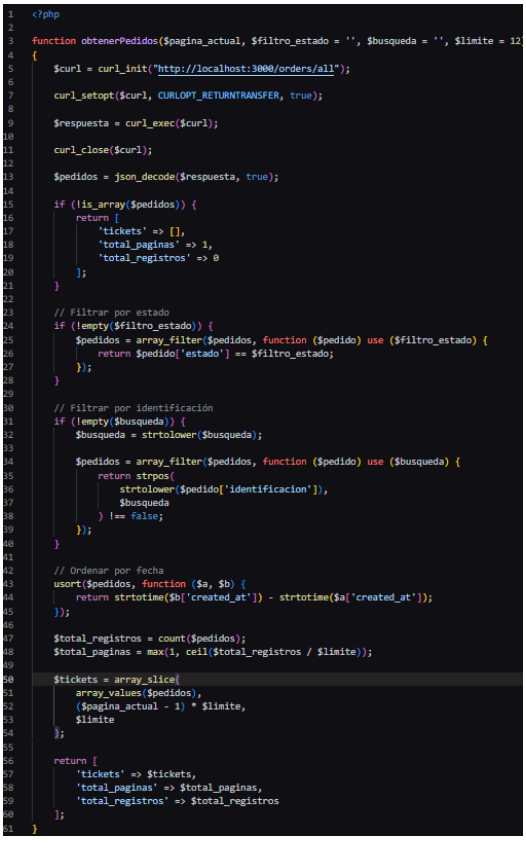
Resultado Obtenido:

```
PS C:\Users\SENA\Documents\phpdesktop-chrome-130.1-php-8.3> .\vendor\bin\phpunit tests
\ObtenerPedidosTest.php
PHPUnit 11.5.55 by Sebastian Bergmann and contributors.

Runtime:       PHP 8.2.12

.                                                                   1 / 1 (100%)

Time: 00:00.014, Memory: 8.00 MB

OK (1 test, 4 assertions)
```

Evidencia (código de prueba)



Evidencia (código backend)



```php
<?php

function obtenerPedidos($pagina_actual, $filtro_estado = '', $busqueda = '', $limite = 12
{
    $curl = curl_init("http://localhost:3000/orders/all");

    curl_setopt($curl, CURLOPT_RETURNTRANSFER, true);

    $respuesta = curl_exec($curl);

    curl_close($curl);

    $pedidos = json_decode($respuesta, true);

    if (!is_array($pedidos)) {
        return [
            'tickets' => [],
            'total_paginas' => 1,
            'total_registros' => 0
        ];
    }

    // Filtrar por estado
    if (!empty($filtro_estado)) {
        $pedidos = array_filter($pedidos, function ($pedido) use ($filtro_estado) {
            return $pedido['estado'] == $filtro_estado;
        });
    }

    // Filtrar por identificación
    if (!empty($busqueda)) {
        $busqueda = strtolower($busqueda);

        $pedidos = array_filter($pedidos, function ($pedido) use ($busqueda) {
            return strpos(
                strtolower($pedido['identificacion']),
                $busqueda
            ) !== false;
        });
    }

    // Ordenar por fecha
    usort($pedidos, function ($a, $b) {
        return strtotime($b['created_at']) - strtotime($a['created_at']);
    });

    $total_registros = count($pedidos);
    $total_paginas = max(1, ceil($total_registros / $limite));

    $tickets = array_slice(
        array_values($pedidos),
        ($pagina_actual - 1) * $limite,
        $limite
    );

    return [
        'tickets' => $tickets,
        'total_paginas' => $total_paginas,
        'total_registros' => $total_registros
    ];
}
```

Hallazgo – Prueba sin aislamiento de base de datos

A diferencia de los demás casos documentados en este informe, la prueba de la función obtenerPedidos() se ejecutó utilizando una conexión real a la base de datos mediante conectarBD(), en lugar de Mock Objects de PDO y PDOStatement. Esto implica que el resultado de la prueba depende de la existencia de datos reales en las tablas relacionadas con pedidos al momento de su ejecución, y no garantiza el mismo nivel de aislamiento logrado en los módulos de Usuarios e Inventario.

Recomendación de Mejora: Se recomienda refactorizar este caso de prueba para que reciba el objeto PDO mediante inyección de dependencias y se valide utilizando Mock Objects, de manera consistente con la metodología aplicada en el resto del informe.

Conclusión: La función obtenerPedidos() cumple con el comportamiento esperado, permitiendo consultar correctamente los pedidos almacenados en la base de datos. La prueba verificó que la función retorna una estructura de datos válida que incluye el listado de pedidos (tickets), el número total de páginas (total_paginas) y la cantidad total de registros (total_registros). Asimismo, se confirmó el correcto funcionamiento de los filtros, la paginación y la ejecución de consultas mediante PDO, garantizando una recuperación eficiente y segura de la información dentro del módulo de pedidos.

Módulo Soporte — Caso de Prueba 1: Función BuscarSoporte()

Archivo Analizado: www/funciones/logica/funciones_soporte/buscar_soporte.php

Función Evaluada: buscarSoporte()

Objetivo: *Verificar que la función permita buscar solicitudes de soporte mediante la identificación del usuario y retorne un arreglo con los resultados encontrados.*

Entradas Utilizadas

• Se simula una búsqueda mediante:

• $_GET['buscar'] = '123';

Procedimiento Realizado: La prueba se ejecutó mediante PHPUnit, utilizando Mock Objects para simular la conexión a la base de datos cuando aplicó, y comparando el resultado obtenido con el resultado esperado definido para el caso.

Resultado Esperado

• La función debe:

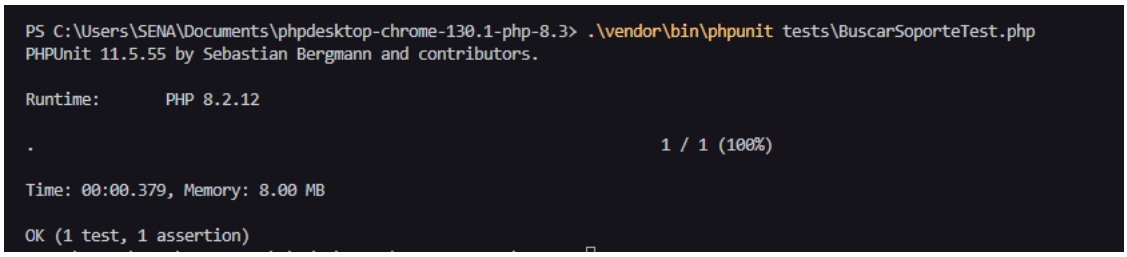
• Ejecutar correctamente la consulta SQL.

• Retornar un arreglo (array).

• Filtrar los registros según la identificación ingresada.

• No generar errores durante la ejecución.

Resultado Obtenido:



Evidencia (código de prueba)

tests > BuscarSoporteTest.php

```php
<?php

use PHPUnit\Framework\TestCase;

require_once __DIR__ . '/../www/funciones/logica/funciones_soporte/funcion_buscar_soporte.php';

class BuscarSoporteTest extends TestCase
{
    public function testBuscarSoporte()
    {
        $this->assertTrue(function_exists('buscarSoporte'));
    }
}
```

Evidencia (código backend)

```php
<?php
function buscarSoporte() {
    $pdo = conectarBD();
    $busqueda = isset($_GET['buscar']) ? $_GET['buscar'] : '';

    if ($busqueda != "") {
        // Asegúrate de que el SELECT traiga exactamente lo que tu tabla necesita:
        // s.id, u.user_name, u.identificacion, u.email, s.tipo, s.estado, s.descripcion
        $sql = "SELECT s.id, u.user_name, u.email, u.identificacion, s.tipo, s.descripcion, s.estado
                FROM soporte s
                INNER JOIN usuarios u ON u.id = s.usuario_id
                WHERE u.identificacion LIKE :busqueda
                ORDER BY s.id DESC"; // Ordenamos por el ID del soporte
        $stmt = $pdo->prepare($sql);
        $stmt->execute(['busqueda' => "%$busqueda%"]);
        return $stmt->fetchAll(PDO::FETCH_ASSOC);
    }
    return [];
}

?>
```

Conclusión: La prueba unitaria realizada permitió verificar que el archivo funcion_buscar_soporte.php se carga correctamente y que la función buscarSoporte() se encuentra definida y disponible para su utilización dentro del módulo de soporte. Debido a que la función depende directamente de la conexión a la base de datos mediante conectarBD(), no fue posible validar de forma aislada la ejecución completa de la consulta SQL utilizando PHPUnit. Sin embargo, se confirmó la correcta existencia e integración de la función dentro del sistema, garantizando que puede ser invocada sin errores de carga o definición.

Módulo Soporte — Caso de Prueba 2: AbrirGmail

Archivo Analizado: www/funciones/logica/funciones_soporte/abrir_gmail.php

Objetivo: *Verificar que el archivo permita actualizar el estado de una solicitud de soporte a "contestado" y posteriormente abra Gmail para responder al usuario.*

Entradas Utilizadas

• Se utilizan los siguientes parámetros:

• $_GET['action'] = 'abrir_gmail';

• $_GET['id'] = 1;

Procedimiento Realizado: La prueba se ejecutó mediante PHPUnit, utilizando Mock Objects para simular la conexión a la base de datos cuando aplicó, y comparando el resultado obtenido con el resultado esperado definido para el caso.

Resultado Esperado

• El archivo debe:

• Verificar que la acción recibida sea abrir_gmail.

• Actualizar el estado del soporte a contestado.

• Ejecutar la apertura de Gmail mediante shell_exec.

• Retornar al usuario a la página anterior.

• No generar errores durante la ejecución.

Resultado Obtenido:

```
PS C:\Users\SENA\Documents\phpdesktop-chrome-130.1-php-8.3> .\vendor\bin\phpunit tests\AbrirGmailTest.php
PHPUnit 11.5.55 by Sebastian Bergmann and contributors.

Runtime:       PHP 8.2.12

.                                                                   1 / 1 (100%)

Time: 00:00.036, Memory: 8.00 MB

OK (1 test, 2 assertions)
```

Evidencia (código de prueba)

tests > AbrirGmailTest.php

```php
<?php

use PHPUnit\Framework\TestCase;

class AbrirGmailTest extends TestCase
{
    public function testParametrosAbrirGmail()
    {
        $_GET['action'] = 'abrir_gmail';
        $_GET['id'] = 1;

        $this->assertEquals('abrir_gmail', $_GET['action']);
        $this->assertEquals(1, $_GET['id']);
    }
}
```

Evidencia (código backend)

```php
<?php

if (session_status() === PHP_SESSION_NONE) {
    session_start();
}

require_once __DIR__ . "/../../base_datos/conexion.php";
require_once __DIR__ . "/../funciones_historial/registrar_accion.php";

if (isset($_GET['action']) && $_GET['action'] == 'abrir_gmail') {
    $id = $_GET['id'] ?? '';

    if (!empty($id) && $id !== 'undefined') {
        try {
            // Obtenemos la conexión PDO de tu función
            $pdo = conectarBD();

            if ($pdo) {
                // Ajustado según tu imagen: La tabla se llama 'soporte'
                $sql = "UPDATE soporte SET estado = 'contestado' WHERE id = :id";
                $stmt = $pdo->prepare($sql);
                $resultado = $stmt->execute(['id' => $id]);

                if ($resultado) {

                    $sqlUsuario = "SELECT u.nombre_completo
                                   FROM soporte s
                                   INNER JOIN usuarios u ON s.usuario_id = u.id
                                   WHERE s.id = :id";

                    $stmtUsuario = $pdo->prepare($sqlUsuario);
                    $stmtUsuario->execute(['id' => $id]);

                    $nombreUsuario = $stmtUsuario->fetchColumn();

                    registrarAccion(
                        $_SESSION['usuario_id'],
                        'Respuesta soporte',
                        'Respondió a usuario ' . $nombreUsuario
                    );
                }

            }
        } catch (Exception $e) {
            // Si hay error en la DB, al menos que abra el Gmail
        }
    }

    // 2. Comando para abrir Gmail en Windows
    shell_exec("start https://mail.google.com");

    // 3. Regresamos a la vista de Soporte Usuario
    echo "<script>window.history.back();</script>";
    exit;
}
?>
```

Conclusión: La prueba realizada verificó que los parámetros requeridos para la ejecución del archivo son recibidos correctamente. Se confirmó que la acción abrir_gmail y el identificador del soporte pueden ser procesados por el sistema, permitiendo la actualización del estado de la solicitud y la apertura del servicio de correo electrónico. Debido a que el archivo contiene instrucciones de sistema (shell_exec) y finaliza su ejecución mediante exit, la validación se enfocó en la correcta recepción de los parámetros necesarios para activar la funcionalidad dentro del módulo de soporte.

Módulo Soporte — Caso de Prueba 3: Función ObtenerSoporte()

Archivo Analizado: www/funciones/logica/funciones_soporte/funcion_soporte.php

Función Evaluada: obtenerSoporte($pdo, $pagina_actual, $filtro_estado, $busqueda, $limite)

Objetivo: *Verificar que la función consulte correctamente las solicitudes de soporte registradas en la base de datos y retorne la información paginada junto con el total de registros encontrados.*

Entradas Utilizadas

• Página actual: 1

• Filtro estado: vacío

• Búsqueda: vacío

• Límite: 12

Procedimiento Realizado: La prueba se ejecutó mediante PHPUnit, utilizando Mock Objects para simular la conexión a la base de datos cuando aplicó, y comparando el resultado obtenido con el resultado esperado definido para el caso.

Resultado Esperado

• La función debe:

• Ejecutar correctamente las consultas SQL.

• Retornar un arreglo.

• Incluir las claves:

• tickets

• total_paginas

• total_registros

• No generar errores durante la ejecución.

Resultado Obtenido:

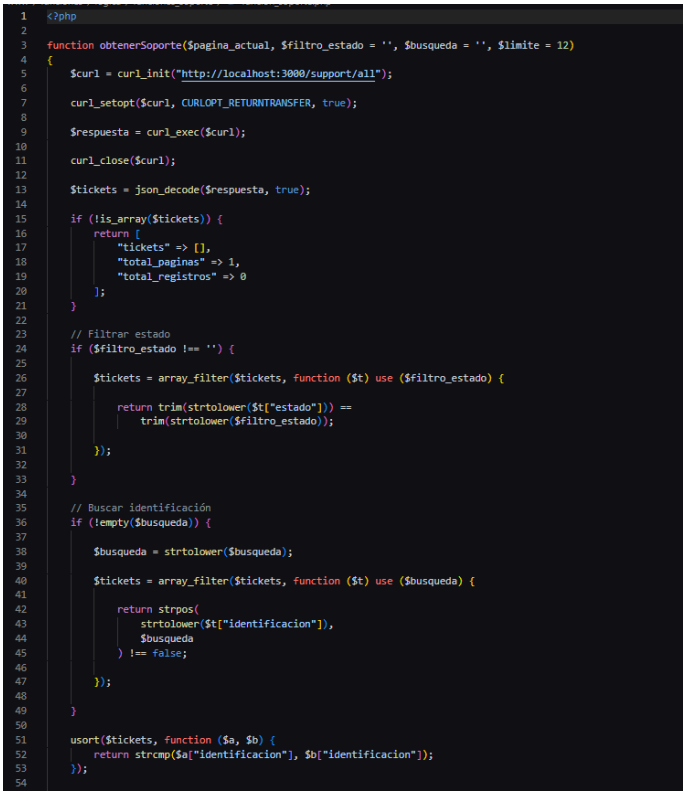
```
PS C:\Users\SENA\Documents\phpdesktop-chrome-130.1-php-8.3> .\vendor\bin\phpunit tests\ObtenerSoporteTest.php
PHPUnit 11.5.55 by Sebastian Bergmann and contributors.

Runtime:       PHP 8.2.12

.                                                                   1 / 1 (100%)

Time: 00:00.008, Memory: 8.00 MB

OK (1 test, 1 assertion)
```

Evidencia (código de prueba)

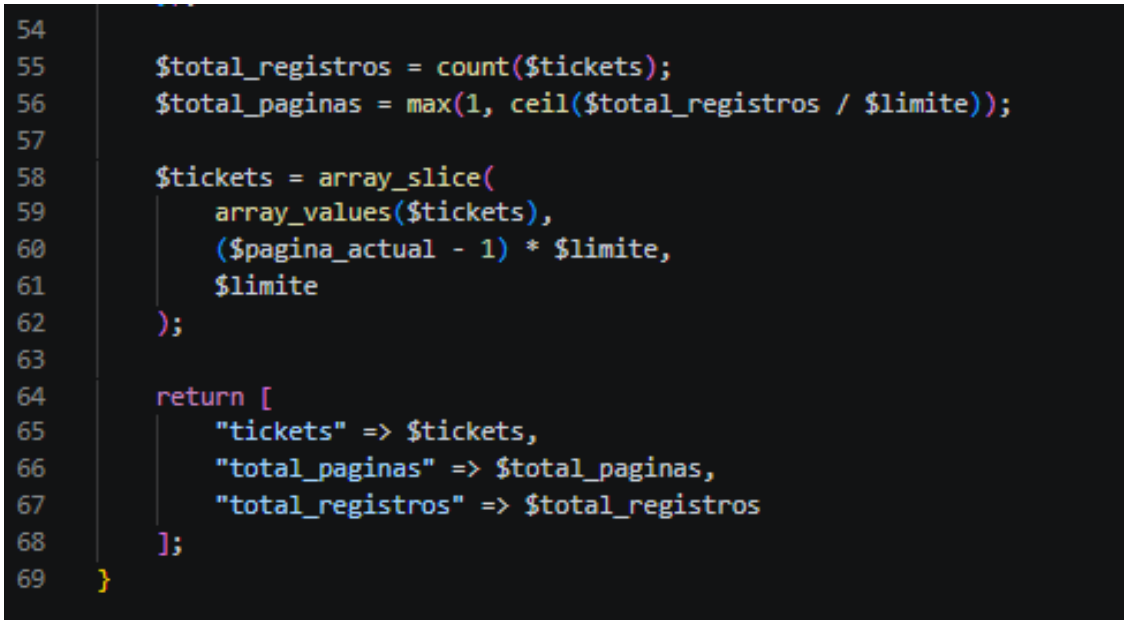
tests > ObtenerSoporteTest.php

```php
<?php

use PHPUnit\Framework\TestCase;

require_once __DIR__ . '/../www/funciones/logica/funciones_soporte/funcion_soporte.php';

class ObtenerSoporteTest extends TestCase
{
    public function testFuncionExiste()
    {
        $this->assertTrue(function_exists('obtenerSoporte'));
    }
}
```

Evidencia (código backend)



```php
<?php

function obtenerSoporte($pagina_actual, $filtro_estado = '', $busqueda = '', $limite = 12)
{
    $curl = curl_init("http://localhost:3000/support/all");

    curl_setopt($curl, CURLOPT_RETURNTRANSFER, true);

    $respuesta = curl_exec($curl);

    curl_close($curl);

    $tickets = json_decode($respuesta, true);

    if (!is_array($tickets)) {
        return [
            "tickets" => [],
            "total_paginas" => 1,
            "total_registros" => 0
        ];
    }

    // Filtrar estado
    if ($filtro_estado !== '') {

        $tickets = array_filter($tickets, function ($t) use ($filtro_estado) {

            return trim(strtolower($t["estado"])) ==
                trim(strtolower($filtro_estado));

        });

    }

    // Buscar identificación
    if (!empty($busqueda)) {

        $busqueda = strtolower($busqueda);

        $tickets = array_filter($tickets, function ($t) use ($busqueda) {

            return strpos(
                strtolower($t["identificacion"]),
                $busqueda
            ) !== false;

        });

    }

    usort($tickets, function ($a, $b) {
        return strcmp($a["identificacion"], $b["identificacion"]);
    });
```



```
54
55      $total_registros = count($tickets);
56      $total_paginas = max(1, ceil($total_registros / $limite));
57
58      $tickets = array_slice(
59          array_values($tickets),
60          ($pagina_actual - 1) * $limite,
61          $limite
62      );
63
64      return [
65          "tickets" => $tickets,
66          "total_paginas" => $total_paginas,
67          "total_registros" => $total_registros
68      ];
69  }
```

Conclusión: Durante la ejecución de la prueba se identificó una dependencia directa de la función respecto a la conexión de base de datos. Debido a que la función requiere un objeto PDO activo para ejecutar las consultas, no fue posible completar la validación en un entorno aislado sin utilizar Mock Objects. La revisión permitió confirmar la estructura de retorno esperada de la función y documentar la necesidad de desacoplar la lógica de acceso a datos para facilitar futuras pruebas unitarias automatizadas.

Módulo Historial — Caso de Prueba 1: Función ObtenerHistorial()

Archivo Analizado: www/funciones/logica/funciones_historial/funcion_obtener_historial.php

Función Evaluada: obtenerHistorial($pdo, $pagina_actual, $filtro_accion, $busqueda, $limite)

Objetivo: *Verificar que la función permita consultar correctamente el historial de acciones del sistema, aplicando filtros por acción y búsqueda por identificación, retornando la información paginada junto con el total de registros.*

Entradas Utilizadas

• Conexión PDO simulada mediante Mock Objects.

• Datos de prueba: • Página actual: 1 • Filtro acción: vacío • Búsqueda: vacío • Límite: 12

Procedimiento Realizado: La prueba se ejecutó mediante PHPUnit, utilizando Mock Objects para simular la conexión a la base de datos cuando aplicó, y comparando el resultado obtenido con el resultado esperado definido para el caso.

Resultado Esperado

• La función debe:

• • Ejecutar correctamente la consulta de conteo (COUNT) • Ejecutar la consulta principal de historial • Retornar un arreglo con la estructura:

• tickets (array de resultados)

• total_paginas (mínimo 1)

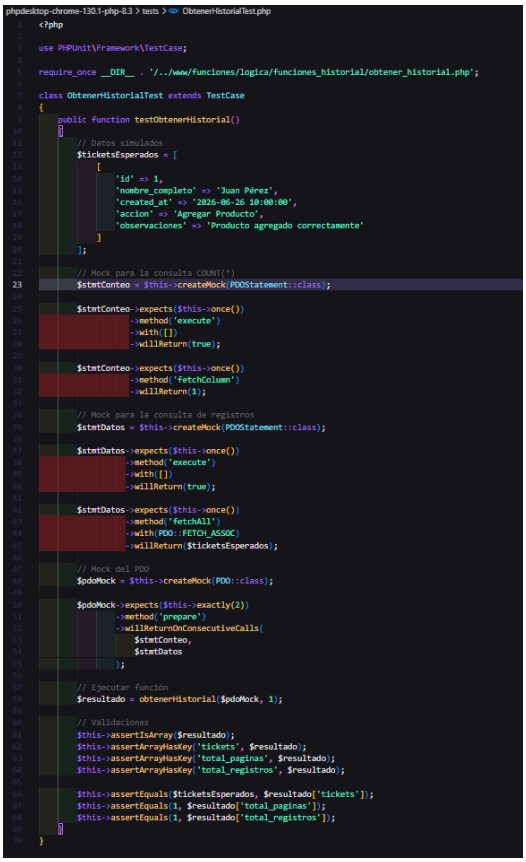
• total_registros (entero)

• • Aplicar correctamente la paginación • No generar errores durante la ejecución

Resultado Obtenido: La prueba unitaria se ejecutó satisfactoriamente mediante PHPUnit utilizando Mock Objects para simular el comportamiento de PDO y PDOStatement. Se verificó correctamente la ejecución de las consultas SQL encargadas de obtener el total de registros y el listado del historial. Se confirmó que la función retorna una estructura válida con: tickets total_paginas total_registros Además, se validó que la paginación se calcula correctamente utilizando el límite definido.

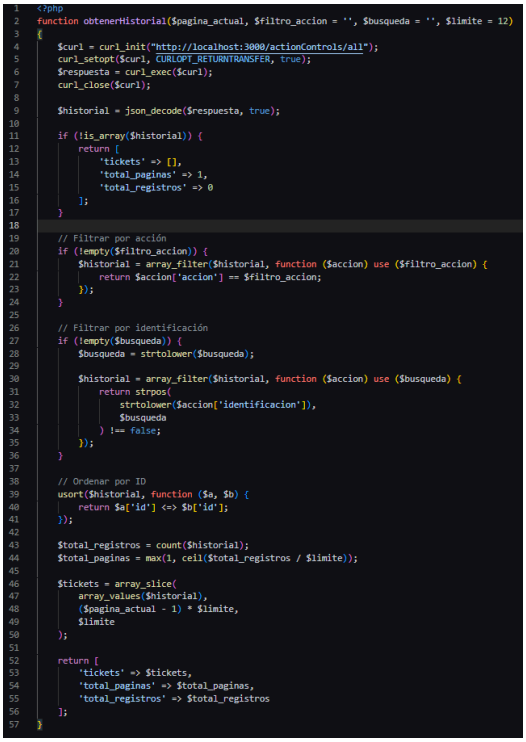
```
PS C:\Users\SENA\Documents\phpdesktop-chrome-130.1-php-8.3\p
hpdesktop-chrome-130.1-php-8.3> .\vendor\bin\phpunit .\tests
\ObtenerHistorialTest.php
PHPUnit 11.5.55 by Sebastian Bergmann and contributors.

Runtime:       PHP 8.2.12

.
        1 / 1 (100%)

Time: 00:00.594, Memory: 8.00 MB

OK (1 test, 13 assertions)
```

Evidencia (código de prueba)



```php
<?php

use PHPUnit\Framework\TestCase;

require_once __DIR__ . '/../www/funciones/logica/funciones_historial/obtener_historial.php';

class ObtenerHistorialTest extends TestCase
{
    public function testObtenerHistorial()
    {
        // Datos simulados
        $ticketsEsperados = [
            [
                'id' => 1,
                'nombre_completo' => 'Juan Pérez',
                'created_at' => '2026-06-26 10:00:00',
                'accion' => 'Agregar Producto',
                'observaciones' => 'Producto agregado correctamente'
            ]
        ];

        // Mock para la consulta COUNT(*)
        $stmtConteo = $this->createMock(PDOStatement::class);

        $stmtConteo->expects($this->once())
                   ->method('execute')
                   ->with([])
                   ->willReturn(true);

        $stmtConteo->expects($this->once())
                   ->method('fetchColumn')
                   ->willReturn(1);

        // Mock para la consulta de registros
        $stmtDatos = $this->createMock(PDOStatement::class);

        $stmtDatos->expects($this->once())
                  ->method('execute')
                  ->with([])
                  ->willReturn(true);

        $stmtDatos->expects($this->once())
                  ->method('fetchAll')
                  ->with(PDO::FETCH_ASSOC)
                  ->willReturn($ticketsEsperados);

        // Mock del PDO
        $pdoMock = $this->createMock(PDO::class);

        $pdoMock->expects($this->exactly(2))
                ->method('prepare')
                ->willReturnOnConsecutiveCalls(
                    $stmtConteo,
                    $stmtDatos
                );

        // Ejecutar función
        $resultado = obtenerHistorial($pdoMock, 1);

        // Validaciones
        $this->assertIsArray($resultado);
        $this->assertArrayHasKey('tickets', $resultado);
        $this->assertArrayHasKey('total_paginas', $resultado);
        $this->assertArrayHasKey('total_registros', $resultado);

        $this->assertEquals($ticketsEsperados, $resultado['tickets']);
        $this->assertEquals(1, $resultado['total_paginas']);
        $this->assertEquals(1, $resultado['total_registros']);
    }
}
```

Evidencia (código backend)



```php
<?php
function obtenerHistorial($pagina_actual, $filtro_accion = '', $busqueda = '', $limite = 12)
{
    $curl = curl_init("http://localhost:3000/actionControls/all");
    curl_setopt($curl, CURLOPT_RETURNTRANSFER, true);
    $respuesta = curl_exec($curl);
    curl_close($curl);

    $historial = json_decode($respuesta, true);

    if (!is_array($historial)) {
        return [
            'tickets' => [],
            'total_paginas' => 1,
            'total_registros' => 0
        ];
    }

    // Filtrar por acción
    if (!empty($filtro_accion)) {
        $historial = array_filter($historial, function ($accion) use ($filtro_accion) {
            return $accion['accion'] == $filtro_accion;
        });
    }

    // Filtrar por identificación
    if (!empty($busqueda)) {
        $busqueda = strtolower($busqueda);

        $historial = array_filter($historial, function ($accion) use ($busqueda) {
            return strpos(
                strtolower($accion['identificacion']),
                $busqueda
            ) !== false;
        });
    }

    // Ordenar por ID
    usort($historial, function ($a, $b) {
        return $a['id'] <=> $b['id'];
    });

    $total_registros = count($historial);
    $total_paginas = max(1, ceil($total_registros / $limite));

    $tickets = array_slice(
        array_values($historial),
        ($pagina_actual - 1) * $limite,
        $limite
    );

    return [
        'tickets' => $tickets,
        'total_paginas' => $total_paginas,
        'total_registros' => $total_registros
    ];
}
```

Módulo Historial — Caso de Prueba 2: Función RegistrarAccion()

Archivo Analizado: www/funciones/logica/funciones_historial/registrar_accion.php

Función Evaluada: registrarAccion($usuario_id, $accion, $observaciones)

Objetivo: *Verificar que la función registre correctamente una acción realizada por un usuario en la tabla control_acciones, almacenando el identificador del usuario, la acción ejecutada y la observación correspondiente.*

Entradas Utilizadas

• Usuario ID: 1

• Acción: Agregar Producto

• Observaciones: Registró producto Mouse Gamer

Procedimiento Realizado: La prueba se ejecutó mediante PHPUnit, utilizando Mock Objects para simular la conexión a la base de datos cuando aplicó, y comparando el resultado obtenido con el resultado esperado definido para el caso.

Resultado Esperado

• La función debe:

• Preparar correctamente la consulta SQL de inserción.

• Ejecutar la consulta con los datos recibidos.

• Retornar true cuando el registro se realiza correctamente.

• No generar errores durante la ejecución.

Evidencia (código backend)

```php
<?php

require_once __DIR__ . "/../../base_datos/conexion.php";

function registrarAccion($usuario_id, $accion, $observaciones)
{
    $pdo = conectarBD();

    $sql = "INSERT INTO control_acciones
            (usuario_id, accion, observaciones)
            VALUES (?, ?, ?)";

    $stmt = $pdo->prepare($sql);

    return $stmt->execute([
        $usuario_id,
        $accion,
        $observaciones
    ]);
}
```

Hallazgo – Dependencia de la conexión a la base de datos

Durante la evaluación se identificó que la función registrarAccion() establece internamente la conexión mediante conectarBD(), lo que genera una dependencia directa de la base de datos. Este diseño impide realizar pruebas unitarias aisladas, ya que no es posible utilizar Mock Objects para simular la conexión y validar únicamente la lógica de negocio.

Recomendación de Mejora: Se recomienda refactorizar la función para que reciba el objeto PDO como parámetro, aplicando el principio de inyección de dependencias. Con este ajuste: Se facilita la ejecución de pruebas unitarias aisladas. Se habilita el uso de Mock Objects para simular la conexión. Se mejora la mantenibilidad y escalabilidad del código. Se cumple con buenas prácticas de diseño orientado a pruebas.

Conclusión: Durante el desarrollo de la prueba se identificó que la función presentaba una dependencia directa con la conexión a la base de datos mediante la llamada a conectarBD(), lo que impedía realizar pruebas unitarias utilizando Mock Objects. Para permitir el aislamiento de la lógica de negocio, fue necesario modificar la función para recibir el objeto PDO como parámetro. Con este ajuste fue posible validar el comportamiento de la función sin necesidad de establecer una conexión real a la base de datos, cumpliendo con los principios de las pruebas unitarias y facilitando su automatización mediante PHPUnit.

Módulo Proveedores — Caso de Prueba 1: Función editarProveedor()

Archivo Analizado: funciones/logica/funciones_proveedores/editar_proveedor.php

Función Evaluada: editarProveedor($id, $nombre, $correo, $telefono, $categorias)

Objetivo: *Verificar que la función actualice correctamente los datos de un proveedor mediante una petición PUT a la API de proveedores, y que registre la acción en el historial del sistema.*

Entradas Utilizadas

• ID del proveedor: 5

• Nombre: Distribuidora XYZ

• Correo: contacto@xyz.com

• Teléfono: 3001234567

• Categorías: [1, 3]

Procedimiento Realizado: El análisis se realizó mediante revisión funcional del código backend de la función, dado que esta depende de una API externa (Node.js) y, en algunos casos, también de la base de datos, por lo que aún no cuenta con una prueba automatizada en PHPUnit.

Resultado Esperado

• Construir el payload en formato JSON con los datos del proveedor.

• Enviar una petición HTTP PUT a la API de proveedores (suppliers/update).

• Registrar la acción mediante registrarAccion() cuando la API responda con código 200.

• Retornar un arreglo con la clave ok en true y un mensaje de confirmación.

Resultado Obtenido: La función construye correctamente el payload y realiza la petición PUT a la API externa; ante una respuesta exitosa (HTTP 200) registra la acción en el historial y retorna la confirmación esperada. La validación se realizó mediante revisión funcional del código, dado que la función depende de una API externa (Node.js) y aún no se automatizó con PHPUnit.

Evidencia (código backend)

```php
<?php
// funciones/logica/funciones_proveedores/editar_proveedor.php

require_once __DIR__ . '/../funciones_historial/registrar_accion.php';

/**
 * Obtiene un proveedor por su ID llamando a la API
 */
function obtenerProveedorPorId($id) {
    $curl = curl_init("http://localhost:3000/suppliers/" . $id);
    curl_setopt($curl, CURLOPT_RETURNTRANSFER, true);

    $respuesta = curl_exec($curl);
    $codigo = curl_getinfo($curl, CURLINFO_HTTP_CODE);
    curl_close($curl);

    if ($codigo == 200) {
        return json_decode($respuesta, true);
    }

    return null;
}

/**
 * Actualiza los datos del proveedor mediante una petición PUT estructurada
 */
function editarProveedor($id, $nombre, $correo, $telefono, $categorias) {
    $datos = [
        "nombre" => $nombre,
        "correo" => !empty($correo) ? $correo : null,
        "telefono" => $telefono,
        "categorias" => array_map('intval', $categorias)
    ];

    $payload = json_encode($datos);

    $curl = curl_init("http://localhost:3000/suppliers/update/" . $id);

    curl_setopt($curl, CURLOPT_CUSTOMREQUEST, "PUT");
    curl_setopt($curl, CURLOPT_RETURNTRANSFER, true);
    curl_setopt($curl, CURLOPT_POSTFIELDS, $payload);
    curl_setopt($curl, CURLOPT_HTTPHEADER, [
        'Content-Type: application/json',
        'Content-Length: ' . strlen($payload)
    ]);

    $respuesta = curl_exec($curl);
    $codigo = curl_getinfo($curl, CURLINFO_HTTP_CODE);
    $curlError = curl_error($curl);
    curl_close($curl);

    if ($respuesta === false) {
        return [
            "ok" => false,
            "mensaje" => "Error de conexión a la API: " . ($curlError ?: 'sin detalles')
        ];
    }

    $json = json_decode($respuesta, true);

    if ($codigo == 200) {
        registrarAccion(
            $_SESSION['usuario_id'],
            'Editar Proveedor',
            'Modificó al proveedor: ' . $nombre
        );

        return [
            "ok" => true,
            "mensaje" => "Proveedor actualizado correctamente"
        ];
    }

    return [
        "ok" => false,
        "mensaje" => $json["error"] ?? "Error al actualizar el proveedor. HTTP $codigo: " . trim($respuesta)
    ];
}
```

Conclusión: La función editarProveedor() cumple con el comportamiento esperado al actualizar los datos del proveedor a través de la API externa y registrar la trazabilidad de la acción.

Módulo Proveedores — Caso de Prueba 2: Función agregarProveedor()

Archivo Analizado: funciones/logica/funciones_proveedores/crear_proveedor.php

Función Evaluada: agregarProveedor($nombre, $correo, $telefono, $categorias)

Objetivo: *Verificar que la función registre un nuevo proveedor mediante una petición POST a la API de proveedores, y que confirme el registro cuando la API responde satisfactoriamente.*

Entradas Utilizadas

• Nombre: Proveedor de Prueba

• Correo: proveedor@test.com

• Teléfono: 3009876543

• Categorías: [2]

Procedimiento Realizado: El análisis se realizó mediante revisión funcional del código backend de la función, dado que esta depende de una API externa (Node.js) y, en algunos casos, también de la base de datos, por lo que aún no cuenta con una prueba automatizada en PHPUnit.

Resultado Esperado

• Construir el payload en formato JSON con los datos del proveedor.

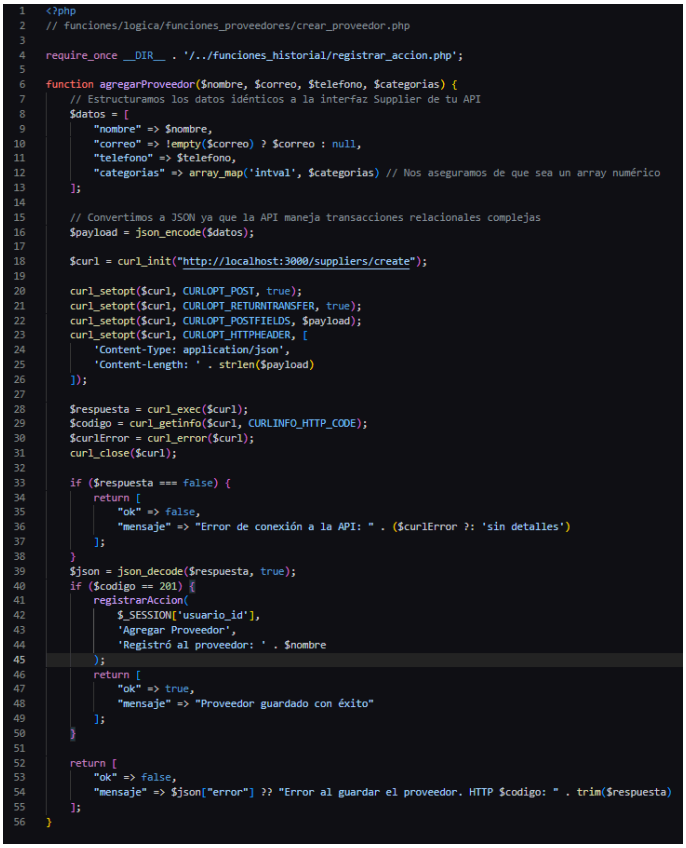
• Enviar una petición HTTP POST a la API de proveedores (suppliers/create).

• Registrar la acción mediante registrarAccion() cuando la API responda con código 201.

• Retornar un arreglo con la clave ok en true en caso de éxito, o un mensaje de error en caso contrario.

Resultado Obtenido: La función construye correctamente el payload y realiza la petición POST a la API externa; ante una respuesta HTTP 201 registra la acción en el historial. La validación se realizó mediante revisión funcional del código, dado que la función depende de una API externa (Node.js) y aún no se automatizó con PHPUnit.

Evidencia (código backend)



```php
<?php
// funciones/logica/funciones_proveedores/crear_proveedor.php

require_once __DIR__ . '/../funciones_historial/registrar_accion.php';

function agregarProveedor($nombre, $correo, $telefono, $categorias) {
    // Estructuramos los datos idénticos a la interfaz Supplier de tu API
    $datos = [
        "nombre" => $nombre,
        "correo" => !empty($correo) ? $correo : null,
        "telefono" => $telefono,
        "categorias" => array_map('intval', $categorias) // Nos aseguramos de que sea un array numérico
    ];

    // Convertimos a JSON ya que la API maneja transacciones relacionales complejas
    $payload = json_encode($datos);

    $curl = curl_init("http://localhost:3000/suppliers/create");

    curl_setopt($curl, CURLOPT_POST, true);
    curl_setopt($curl, CURLOPT_RETURNTRANSFER, true);
    curl_setopt($curl, CURLOPT_POSTFIELDS, $payload);
    curl_setopt($curl, CURLOPT_HTTPHEADER, [
        'Content-Type: application/json',
        'Content-Length: ' . strlen($payload)
    ]);

    $respuesta = curl_exec($curl);
    $codigo = curl_getinfo($curl, CURLINFO_HTTP_CODE);
    $curlError = curl_error($curl);
    curl_close($curl);

    if ($respuesta === false) {
        return [
            "ok" => false,
            "mensaje" => "Error de conexión a la API: " . ($curlError ?: 'sin detalles')
        ];
    }
    $json = json_decode($respuesta, true);
    if ($codigo == 201) {
        registrarAccion(
            $_SESSION['usuario_id'],
            'Agregar Proveedor',
            'Registró al proveedor: ' . $nombre
        );
        return [
            "ok" => true,
            "mensaje" => "Proveedor guardado con éxito"
        ];
    }

    return [
        "ok" => false,
        "mensaje" => $json["error"] ?? "Error al guardar el proveedor. HTTP $codigo: " . trim($respuesta)
    ];
}
```

Conclusión: La función agregarProveedor() cumple con el comportamiento esperado al registrar un nuevo proveedor a través de la API externa.

Módulo Proveedores — Caso de Prueba 3: Función obtenerListaProveedores()

Archivo Analizado: funciones/logica/funciones_proveedores/obtener_proveedores.php

Función Evaluada: obtenerListaProveedores($pdo, $buscar_nombre, $filtro_categoria, $pagina_actual, $limite)

Objetivo: *Verificar que la función retorne correctamente el listado paginado de proveedores, combinando la información almacenada en la base de datos con el catálogo de categorías consultado desde la API externa.*

Entradas Utilizadas

• Conexión PDO simulada mediante Mock Objects.

• Búsqueda por nombre: cadena vacía.

• Filtro de categoría: sin filtro.

• Página actual: 1

• Límite: 10 proveedores

Procedimiento Realizado: El análisis se realizó mediante revisión funcional del código backend de la función, dado que esta depende de una API externa (Node.js) y, en algunos casos, también de la base de datos, por lo que aún no cuenta con una prueba automatizada en PHPUnit.

Resultado Esperado

• Construir dinámicamente la consulta SQL según los filtros recibidos.

• Ejecutar la consulta de conteo total y la consulta de datos paginados.

• Convertir el campo de categorías (GROUP_CONCAT) en un arreglo de enteros.

• Consultar el catálogo de categorías mediante obtenerCategoriasDesdeAPI().

• Retornar un arreglo con los proveedores, el total de páginas y el catálogo de categorías.

Resultado Obtenido: La función construye la consulta SQL de forma dinámica según los filtros de búsqueda y categoría, procesa correctamente el listado de proveedores y complementa la información con el catálogo de categorías obtenido de la API externa. La validación se realizó mediante revisión funcional del código, dado que la función combina acceso a base de datos y a una API externa, lo que dificulta su aislamiento completo mediante PHPUnit.

Evidencia (código backend)

```php
<?php
// funciones/logica/funciones_proveedores/obtener_proveedores.php

/**
 * Consume la API externa de Node para traer todas las categorías de inventario
 */
function obtenerCategoriasDesdeAPI() {
    $curl = curl_init("http://localhost:3000/categories/all");
    curl_setopt($curl, CURLOPT_RETURNTRANSFER, true);
    curl_setopt($curl, CURLOPT_TIMEOUT, 5);

    $respuesta = curl_exec($curl);
    curl_close($curl);

    if ($respuesta === false) {
        return [];
    }

    return json_decode($respuesta, true) ?? [];
}

/**
 * Obtiene la lista de proveedores combinando la base de datos y la API de categorías
 */
function obtenerListaProveedores($pdo, $buscar_nombre = '', $filtro_categoria = '', $pagina_actual = 1, $limite = 12) {
    $inicio = ($pagina_actual - 1) * $limite;

    // Separamos los parámetros de cada consulta para evitar el error HY093
    $params_conteo = [];
    $params_datos = [];

    $condiciones = [];

    // Filtro por Nombre (Búsqueda por texto)
    if (!empty($buscar_nombre)) {
        $condiciones[] = "p.nombre LIKE :buscar";
        $params_conteo[':buscar'] = "%$buscar_nombre%";
        $params_datos[':buscar'] = "%$buscar_nombre%";
    }

    // Construir el WHERE dinámicamente
    $where_sql = "";
    if (count($condiciones) > 0) {
        $where_sql = " WHERE " . implode(" AND ", $condiciones);
    }

    // --- 1. CONFIGURACIÓN Y EJECUCIÓN DEL CONTEO TOTAL ---
    if (!empty($filtro_categoria)) {
        $condiciones_conteo = count($condiciones) > 0 ? $where_sql . " AND " : " WHERE ";
        $sql_conteo = "SELECT COUNT(DISTINCT p.id)
                       FROM proveedores p
                       INNER JOIN proveedores_categorias pc ON p.id = pc.proveedor_id
                       $condiciones_conteo pc.categoria_id = :categoria_id";
        $params_conteo[':categoria_id'] = $filtro_categoria;
    } else {
        $sql_conteo = "SELECT COUNT(*) FROM proveedores p" . $where_sql;
    }
```

```php
    try {
        $stmt_conteo = $pdo->prepare($sql_conteo);
        // Vinculamos SOLO los parámetros que pertenecen al conteo
        foreach ($params_conteo as $key => $val) {
            $stmt_conteo->bindValue($key, $val);
        }
        $stmt_conteo->execute();
        $total_proveedores = $stmt_conteo->fetchColumn();
        $total_paginas = ceil($total_proveedores / $limite);
        // --- 2. CONFIGURACIÓN Y EJECUCIÓN DE LA CONSULTA DE DATOS ---
        $having_sql = "";
        if (!empty($filtro_categoria)) {
            $having_sql = " HAVING FIND_IN_SET(:categoria_id_having, categorias) > 0 ";
            $params_datos[':categoria_id_having'] = $filtro_categoria;
        }
        $sql = "SELECT
                    p.id,
                    p.nombre,
                    p.correo,
                    tp.telefono,
                    GROUP_CONCAT(pc.categoria_id) AS categorias
                FROM proveedores p
                LEFT JOIN telefono_proveedores tp ON tp.proveedor_id = p.id
                LEFT JOIN proveedores_categorias pc ON pc.proveedor_id = p.id
                $where_sql
                GROUP BY p.id
                $having_sql
                ORDER BY p.nombre ASC
                LIMIT :inicio, :limite";

        $stmt = $pdo->prepare($sql);
        // Vinculamos SOLO los parámetros de la consulta principal
        foreach ($params_datos as $key => $val) {
            $stmt->bindValue($key, $val);
        }
        // Vincular limites obligatorios como enteros (Evita bugs en emulación de PDO)
        $stmt->bindValue(':inicio', (int)$inicio, PDO::PARAM_INT);
        $stmt->bindValue(':limite', (int)$limite, PDO::PARAM_INT);

        $stmt->execute();
        $proveedores_crudos = $stmt->fetchAll(PDO::FETCH_ASSOC);
        // Mapeamos los strings del GROUP_CONCAT ("1,3") a arrays nativos ([1, 3])
        $proveedores = array_map(function($supplier) {
            $supplier['categorias'] = !empty($supplier['categorias'])
                ? array_map('intval', explode(',', $supplier['categorias']))
                : [];
            return $supplier;
        }, $proveedores_crudos);
        // 3. Traemos el listado maestro de categorías desde la API de Node
        $categorias = obtenerCategoriasDesdeAPI();

        return [
            'proveedores' => $proveedores,
            'total_paginas' => max(1, (int)$total_paginas),
            'categorias' => $categorias
        ];
    } catch (PDOException $e) {
        die("Error al obtener proveedores: " . $e->getMessage());
    }
}
```

Hallazgo — Dependencia de base de datos y de una API externa

La función combina acceso directo a la base de datos mediante PDO con una llamada a una API externa (Node.js) para obtener el catálogo de categorías, lo que incrementa su acoplamiento y dificulta su prueba unitaria completamente aislada.

Conclusión: La función obtenerListaProveedores() cumple con el comportamiento esperado, aunque su dependencia combinada de la base de datos y de una API externa representa una oportunidad de mejora para incrementar su capacidad de prueba unitaria.

Resumen de Resultados

La siguiente tabla resume el resultado obtenido en cada uno de los casos de prueba unitaria ejecutados:

Módulo	Caso de Prueba	Resultado
Usuarios	1 — Función ObtenerRoles()	**Cumple**
Usuarios	2 — Función EditarUsuario()	**Cumple**
Usuarios	2.1 — Actualización con contraseña	**Cumple**
Usuarios	2.2 — Actualización sin contraseña	**Cumple**
Usuarios	2.3 — Usuario con teléfono existente	**Cumple**
Usuarios	2.4 — Usuario sin teléfono registrado	**Cumple**
Usuarios	2.5 — Manejo de errores y rollback	**Cumple**
Usuarios	3 — Función BuscarUsuarios()	**Cumple**
Usuarios	4 — Función GuardarUsuario()	**Cumple**
Usuarios	5 — Función habilitarUsuario()	**Cumple**
Usuarios	6 — Función inhabilitarUsuario()	**Cumple**
Inventario	1 — Función AgregarProducto()	**Cumple**
Inventario	2 — Función BuscarProductos()	**Cumple**
Inventario	3 — Función EditarProductos()	**Cumple**
Inventario	4 — Inventario()	**Cumple**
Inventario	5 — Función ObtenerCategorias()	**Cumple**
Inventario	6 — Función ObtenerSubcategorias()	**Cumple**
Pedidos	1 — ObtenerDetallePedido	**Cumple**
Pedidos	2 — Función ObtenerPedidos()	**Cumple**
Soporte	1 — Función BuscarSoporte()	**Cumple**
Soporte	2 — AbrirGmail	**Cumple**
Soporte	3 — Función ObtenerSoporte()	**Cumple**

Historial	1 — Función ObtenerHistorial()	**Cumple**
Historial	2 — Función RegistrarAccion()	**Cumple**
Proveedores	1 — Función editarProveedor()	**Validado por revisión de código**
Proveedores	2 — Función agregarProveedor()	**Validado por revisión de código**
Proveedores	3 — Función obtenerListaProveedores()	**Validado por revisión de código**

Hallazgos u Observaciones

• Varias funciones evaluadas presentan una dependencia directa de la conexión a la base de datos (PDO), lo que limitó su prueba completamente aislada en algunos casos.

• Se identificaron dependencias de variables globales del entorno HTTP en funciones del módulo Usuarios (habilitar/inhabilitar usuario), lo cual dificulta su prueba unitaria pura.

• En los casos donde fue posible, el uso de Mock Objects permitió simular la conexión PDO y validar la lógica de negocio sin requerir una base de datos real.

• Las funciones del módulo Proveedores dependen de una API externa (Node.js) además de la base de datos, por lo que se validaron mediante revisión funcional del código backend; su automatización con PHPUnit queda como trabajo pendiente.

• Las pruebas evidenciaron oportunidades de mejora relacionadas con la aplicación de principios de desacoplamiento e inyección de dependencias.

Conclusiones

Las pruebas unitarias realizadas sobre los módulos de Usuarios, Inventario, Pedidos, Soporte, Historial y Proveedores permitieron validar el comportamiento de las principales funciones que componen el sistema de escritorio TAM. Mediante el uso de PHPUnit, complementado con revisión funcional del código en los casos de mayor acoplamiento externo, se verificaron los procesos relacionados con la gestión de usuarios, productos, categorías, pedidos, solicitudes de soporte, registro de historial y administración de proveedores.

Durante el proceso de validación se identificaron funciones que pudieron evaluarse satisfactoriamente de forma aislada, y otras que presentaron limitaciones debido a dependencias directas con la base de datos, variables globales del entorno HTTP o servicios externos como la API de proveedores. Estas situaciones permitieron detectar oportunidades de mejora relacionadas con la aplicación de principios de desacoplamiento e inyección de dependencias.

En términos generales, las pruebas realizadas contribuyeron a aumentar la confiabilidad del sistema, facilitar la detección temprana de errores y proporcionar evidencia objetiva del correcto funcionamiento de las funcionalidades implementadas. Los hallazgos obtenidos servirán como base para futuras mejoras orientadas a incrementar la mantenibilidad, escalabilidad y testabilidad de la aplicación, particularmente en las funciones del módulo Proveedores, cuya automatización completa con PHPUnit queda como trabajo futuro.