Resumen de Resultados de Pruebas

SISTEMA GASOLINERA

Fecha de creación	09/10/2025
Analista de aseguramiento de calidad	Carlos Javier Sandoval Catalán

Análisis De Pruebas Unitarias

Las pruebas unitarias constituyen la primera línea de defensa en la validación de calidad del software. Su propósito es verificar de manera aislada el correcto funcionamiento de cada componente del sistema, garantizando que las funciones, métodos y clases cumplan con su lógica esperada antes de integrarse en un entorno más amplio.

Para este proyecto, se utilizó PHPUnit, el framework estándar de pruebas para aplicaciones desarrolladas en PHP. La elección de PHPUnit responde a su robustez, compatibilidad con arquitecturas MVC, y a su integración directa con flujos de desarrollo modernos basados en Composer y XAMPP.

Las pruebas se realizaron individualmente en cada módulo del sistema, para esto se crearon archivos Test de phpunit como los siguientes ejemplos:

Pruebas Unitarias Modulo Órdenes de Compra:

```php
public function testSaveCreatesPurchaseUpdatesStockAndTx(): void
{
    $beforeTx     = $this->tableCount('container_tx');
    $beforeItem   = $this->tableCount('purchase_items');
    $beforeStockA = (float)$this->pdo->query("SELECT qty_liters FROM container_stock WHERE container_id=1")->fetchColumn();

    $pid = $this->model->save(
        ['supplier_id'=>1, 'date'=>$this->today, 'payment_type'=>'CREDIT', 'notes'=>'Compra test'],
        [
            ['petrol_type_id'=>1, 'container_id'=>1, 'qty_liters'=>200.0, 'unit_cost'=>22.00],
            ['petrol_type_id'=>1, 'container_id'=>2, 'qty_liters'=> 50.5, 'unit_cost'=>22.00],
        ],
        7
    );

    $this->assertGreaterThan(0, $pid);

    // Correlativo PO-YYYYMM-0002
    $headerRow = $this->model->getHeader($pid);
    $this->assertStringStartsWith('PO-'.date('Ym').'-0002', $headerRow['receipt_no']);

    // Items y total correcto
    $its = $this->model->getItems($pid);
    $this->assertCount(2, $its);
    $expectedTotal = 200.0*22.00 + 50.5*22.00;
    $this->assertEqualsWithDelta($expectedTotal, (float)$headerRow['total_cost'], 0.0001);

    // TX +2 y stock sumó en Tank A
    $this->assertSame($beforeTx + 2, $this->tableCount('container_tx'));
    $afterStockA = (float)$this->pdo->query("SELECT qty_liters FROM container_stock WHERE container_id=1")->fetchColumn();
    $this->assertEqualsWithDelta($beforeStockA + 200.0, $afterStockA, 0.0001);

    // purchase_items +2
    $this->assertSame($beforeItem + 2, $this->tableCount('purchase_items'));
}
```

Esta prueba unitaria evalúa el método save() del módulo de Órdenes de Compra, verificando que al registrar una compra se creen correctamente los encabezados, los detalles, las transacciones y las actualizaciones de stock. Está estructurada bajo el enfoque Arrange–Act–Assert: primero obtiene los valores iniciales de las tablas, luego ejecuta la operación de guardado y finalmente compara los resultados esperados con los obtenidos.

```php
public function testCreateReceiptHappyPathUpdatesStockTxAndClosesOrder(): void
{
    // Crear OC con 1 ítem Regular
    $oid = $this->model->saveOrder(
        ['supplier_id'=>1, 'date'=>$this->today, 'notes'=>null],
        [
            ['petrol_type_id'=>1, 'qty_liters'=>60.0, 'unit_cost'=>0], // forzado por BD
        ],
        70
    );

    $ordenItemId   = (int)$this->pdo->query("SELECT id FROM compras_orden_items WHERE orden_id=$oid")->fetchColumn();
    $beforeStockA  = (float)$this->pdo->query("SELECT qty_liters FROM container_stock WHERE container_id=1")->fetchColumn();
    $beforeTxCount = $this->tableCount('container_tx');

    $reciboId = $this->model->createReceipt(
        $oid,
        ['fecha'=>$this->today, 'documento_proveedor'=>'FAC-001', 'observaciones'=>'ok'],
        [[
            'orden_item_id' => $ordenItemId,
            'contenedor_id' => 1,   // Tank A → Regular
            'petrol_type_id'=> 1,
            'rec_litros'    => 60.0,
            'costo_u'       => 22.0,
        ]],
        5
    );

    $this->assertGreaterThan(0, $reciboId);

    $afterStockA = (float)$this->pdo->query("SELECT qty_liters FROM container_stock WHERE container_id=1")->fetchColumn();
    $this->assertEqualsWithDelta($beforeStockA + 60.0, $afterStockA, 0.0001);
    $this->assertSame($beforeTxCount + 1, $this->tableCount('container_tx'));

    $estatus = $this->pdo->query("SELECT estatus FROM compras_orden WHERE id=$oid")->fetchColumn();
    $this->assertSame('RECIBIDO', $estatus);
}
```

Esta prueba unitaria valida el flujo completo de recepción de una orden de compra dentro del sistema. Su propósito es comprobar que, al registrar un recibo, se actualice correctamente el stock del contenedor, se registre la transacción correspondiente y se cambie el estado de la orden a “RECIBIDO”. La prueba sigue la estructura Arrange–Act–Assert: primero prepara los datos de una orden simulada, luego ejecuta el método createReceipt() y finalmente verifica que el stock aumente, que exista una nueva transacción y que el estado de la orden se actualice.

Y luego de correr las pruebas, se obtienen los resultados en consola:

```
Purchase
  ✔Save creates purchase updates stock and tx
  ✔Save throws fuel mismatch on wrong container
  ✔List and get header items filters
  ✔List orders and save order uses d b costs
  ✔Create receipt happy path updates stock tx and closes order
  ✔Create receipt rejects wrong container fuel
  ✔Create receipt rejects not found and not editable
  ✔Create receipt rejects when order already closed
```

Pruebas Unitarias POS:

```php
public function testFindSatByNitAndEnsureCustomerFromSAT(): void
{
    $sat = $this->model->findSatByNit('1234567');
    $this->assertNotNull($sat);
    $this->assertSame('Nit Demo S.A.', $sat['fullname']);

    // Asegura cliente
    $cid = $this->model->ensureCustomerFromSAT($sat);
    $this->assertGreaterThan(0, $cid);

    // Segunda llamada debe devolver el mismo customer
    $cid2 = $this->model->ensureCustomerFromSAT($sat);
    $this->assertSame($cid, $cid2);

    $row = $this->pdo->query("SELECT fullname FROM customer_list WHERE customer_id=$cid")->fetchColumn();
    $this->assertSame('Nit Demo S.A.', $row);
}
```

Esta prueba unitaria comprueba la correcta integración del sistema con el servicio simulado de SAT y la gestión de clientes derivados de esa fuente. Evalúa que al buscar un NIT válido mediante findSatByNit(), el sistema reciba los datos del contribuyente y, al ejecutar ensureCustomerFromSAT(), se cree o confirme el registro del cliente correspondiente.

```php
public function testSaveTransactionHappyPathUsesDefaultContainerAndCreatesTx(): void
{
    $idReg = (int)$this->pdo->query("SELECT petrol_type_id FROM petrol_type_list WHERE name='Regular'")->fetchColumn();
    $cidCF = (int)$this->pdo->query("SELECT customer_id FROM customer_list WHERE customer_code='CF'")->fetchColumn();

    $beforeTx   = $this->tableCount('container_tx');
    $beforeTrx  = $this->tableCount('transaction_list');
    $beforeA    = (float)$this->pdo->query("SELECT qty_liters FROM container_stock WHERE container_id=1")->fetchColumn();

    $price  = number_format(24.25, 4, '.', '');    // Q/gal
    $gal    = number_format(10.500, 3, '.', '');   // galones (se guardan en 'liter')
    $amount = number_format(24.25 * 10.5, 2, '.', '');
    $total  = $amount;
    $cash   = $total;
    $change = number_format(0, 2, '.', '');

    [$trxId, $receipt] = $this->model->saveTransaction([
        'customer_id'     => $cidCF,
        'petrol_type_id'  => $idReg,
        'price'           => $price,
        'gallons'         => $gal,
        'amount'          => $amount,
        'total'           => $total,
        'tendered_amount' => $cash,
        'change'          => $change,
        'type'            => 1,
    ], $userId = 2);
```

Esta prueba unitaria valida el camino feliz de una venta en POS: ejecuta saveTransaction() con un cliente y un tipo de gasolina válidos, usando el contenedor default. Verifica que se genere un recibo con formato correcto (regex), que se creen los registros correspondientes (1 en transaction_list y 1 en container_tx), y que el stock del contenedor disminuya exactamente según los galones vendidos

Y luego de correr las pruebas, se obtienen los resultados en consola:

```
Sale
  ✔Get active fuels ordered and get sale price gal
  ✔Find sat by nit and ensure customer from s a t
  ✔Save transaction happy path uses default container and creates tx
  ✔Save transaction chooses most filled when no default
  ✔Save transaction throws when no container available
  ✔Save transaction throws invalid gallons and insufficient stock
  ✔Get receipt returns joined fields
```

Pruebas Unitarias Módulo Puestos:

```php
public function testSaveInsertCreatesNewPositionWithDefaults(): void
{
    $ok = $this->model->save([
        // sin position_id → INSERT
        'name' => 'Auxiliar',
        // 'description' => null (default)
        // 'suggested_role' => 0 (default)
        // 'status' => 1 (default)
    ]);
    $this->assertTrue($ok);

    $row = $this->pdo->query("SELECT * FROM positions WHERE name='Auxiliar'")->fetch(\PDO::FETCH_ASSOC);
    $this->assertNotFalse($row);
    $this->assertSame('Auxiliar', $row['name']);
    $this->assertNull($row['description']);
    $this->assertSame(0, (int)$row['suggested_role']);
    $this->assertSame(1, (int)$row['status']);
}
```

Esta prueba unitaria verifica la inserción de un nuevo puesto con valores predeterminados. Al llamar save() sin position_id (modo INSERT), se espera que se cree el registro con name = 'Auxiliar' y que los campos por defecto queden correctos: description = null, suggested_role = 0 y status = 1. Luego consulta la tabla positions para ese nombre y asegura que el registro exista y que todos los valores coincidan con los defaults definidos.

Y luego de correr las pruebas, se obtienen los resultados en consola:

```
Position
  ✔All returns ordered by status desc then name asc
  ✔Get returns row or null
  ✔Save insert creates new position with defaults
  ✔Save update modifies fields
  ✔Delete blocks when employees exist
  ✔Delete removes when no employees
```

Por último, luego de correr todas las pruebas se obtuvieron estos resultados:

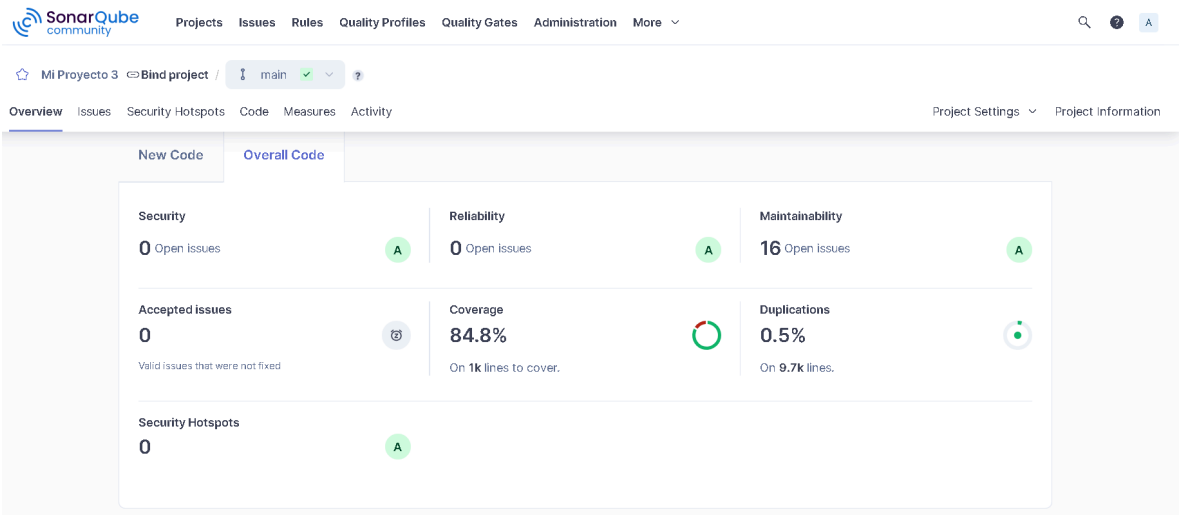
```
PS D:\xampp\htdocs\psms - copia> vendor\bin\phpunit.bat --testdox
PHPUnit 11.5.42 by Sebastian Bergmann and contributors.

Runtime:       PHP 8.2.12 with PCOV 1.0.11
Configuration: D:\xampp\htdocs\psms - copia\phpunit.xml

.................................................................. 65 / 80 ( 81%)
...............                                                    80 / 80 (100%)

Time: 00:07.015, Memory: 14.00 MB

Auth Controller
 ✔Flow login happy path
 ✔Flow invalid credentials
 ✔Flow missing fields
 ✔Flow method not allowed
 ✔Flow logout

Container
 ✔All returns joined and stock
 ✔List filtered by query and status
 ✔Find returns one with stock
 ✔Create inserts and initializes stock and resets default
 ✔Update resets default within same type
 ✔Default for type returns single default
 ✔Movements filter by date
 ✔Delete removes container

Customer
 ✔Get all returns sorted by fullname asc
 ✔Get by id returns row or null
 ✔Save insert creates customer
 ✔Save update modifies customer
 ✔Delete removes customer

Dashboard
 ✔Count petrol types returns active only
 ✔Count customers returns active only
 ✔Get balance subtracts payments from debts
 ✔Get today sales sums only today
 ✔Edge cases empty tables return zero

Employee
 ✔All returns joined with positions and order
 ✔Get returns single with join
 ✔Save insert generates monthly sequential code
 ✔Save update modifies fields
 ✔Search empty query returns limited ordered
 ✔Search by like across fields
 ✔Delete removes employee

Inventory
 ✔In adds stock and inserts tx
 ✔Out decreases stock and inserts tx
 ✔Out throws when insufficient stock
 ✔Transfer same type moves stock and records two tx
 ✔Transfer rejects different types
 ✔Transfer throws when insufficient from
```

```
Kardex
  ✔List filters and delta sign and ordering
  ✔Opening balance before date
  ✔Get tx by id returns joined record
  ✔Transfer pair orders out then in
  ✔Filter lists for u i only active

Petrol Type
  ✔All returns ordered by name asc
  ✔Find returns row or null
  ✔Save insert syncs legacy price to sale price
  ✔Save update syncs legacy price and changes fields
  ✔Delete removes row

Position
  ✔All returns ordered by status desc then name asc
  ✔Get returns row or null
  ✔Save insert creates new position with defaults
  ✔Save update modifies fields
  ✔Delete blocks when employees exist
  ✔Delete removes when no employees

Purchase
  ✔Save creates purchase updates stock and tx
  ✔Save throws fuel mismatch on wrong container
  ✔List and get header items filters
  ✔List orders and save order uses d b costs
  ✔Create receipt happy path updates stock tx and closes order
  ✔Create receipt rejects wrong container fuel
  ✔Create receipt rejects not found and not editable
  ✔Create receipt rejects when order already closed

Sale
  ✔Get active fuels ordered and get sale price gal
  ✔Find sat by nit and ensure customer from s a t
  ✔Save transaction happy path uses default container and creates tx
  ✔Save transaction chooses most filled when no default
  ✔Save transaction throws when no container available
  ✔Save transaction throws invalid gallons and insufficient stock
  ✔Get receipt returns joined fields

Sales Report
  ✔Get transactions by date range orders asc
  ✔Get transactions filters by user

Supplier
  ✔All returns ordered by name asc
  ✔List filtered by query and status
  ✔Get returns single supplier
  ✔Save insert and update
  ✔Delete removes supplier

User
  ✔Get by username and get by id
  ✔Update account without password change
  ✔Update account rejects duplicate username
  ✔Update account password change wrong old
  ✔Update account password change rejects m d 5 stored hash
  ✔Update account password change happy path

OK (80 tests, 359 assertions)

Generating code coverage report in Clover XML format ... done [00:00.138]

Generating code coverage report in HTML format ... done [00:00.258]
```

Análisis De SonarQube

SonarQube es nuestra capa de higiene técnica continua: un análisis estático que recorre el código PHP sin ejecutarlo y señala bugs, vulnerabilidades, code smells, duplicados y complejidad, además de consolidar cobertura de pruebas. Lo usamos porque convierte la calidad en datos objetivos: define un Quality Gate con umbrales claros y lo aplica de forma automática en cada cambio.

Luego de realizar el análisis al sistema, se obtuvieron los siguientes resultados generales:

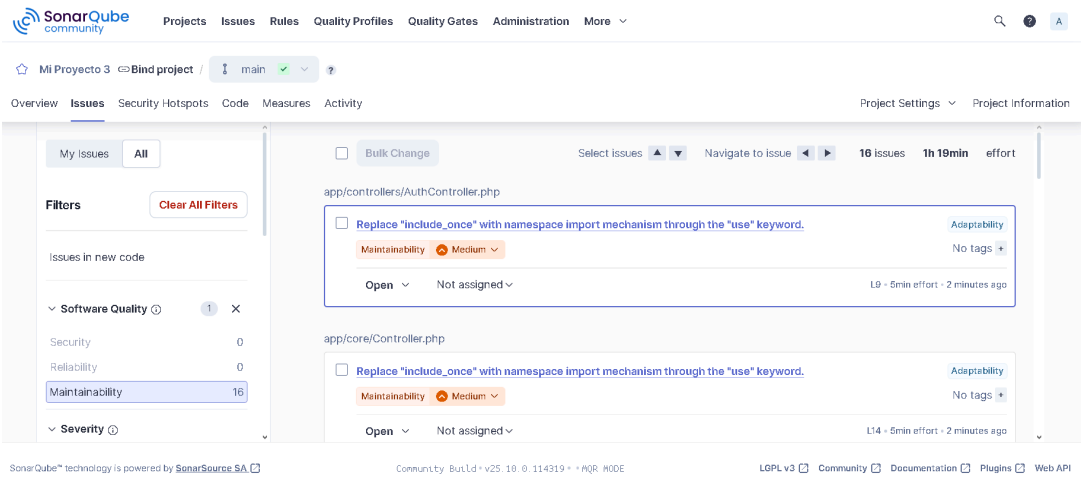


El análisis realizado sobre el proyecto arrojó un panorama muy positivo en términos de calidad, seguridad y mantenibilidad del código.

Los indicadores clave muestran un proyecto estable, seguro y bien probado, lo que refleja buenas prácticas de desarrollo y pruebas automatizadas.

Sin embargo, se pueden notar pequeñas incidencias en los siguientes espacios:

Mantenibilidad:



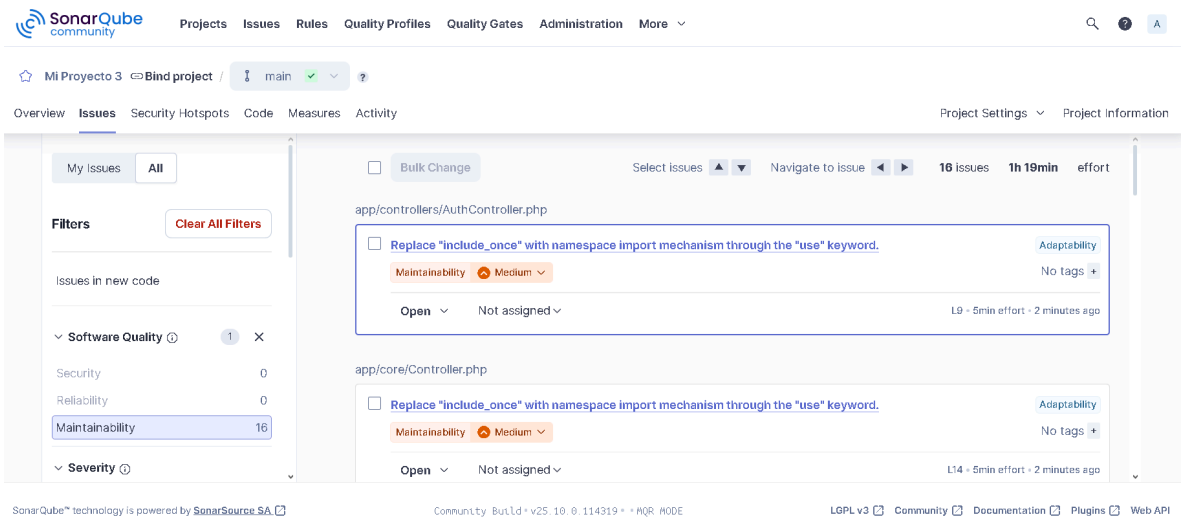
El análisis de SonarQube reportó 16 incidencias clasificadas como code smells. Estos hallazgos no representan errores funcionales, sino recomendaciones de refactorización

destinadas a mejorar la claridad y la estructura del código. En concreto, la herramienta sugiere reemplazar el uso de include_once por importaciones mediante el mecanismo de namespaces (use), siguiendo las convenciones modernas de PHP (PSR-4).

Sin embargo, en este proyecto dicha recomendación no aplica de forma práctica debido a la estructura personalizada del enrutador (Router.php), el cual resuelve dinámicamente los controladores y modelos a partir de la URL sin utilizar autoloading basado en Composer. Esta arquitectura, aunque distinta al enfoque tradicional, es completamente funcional y controlada, garantizando la correcta carga de dependencias mediante rutas absolutas internas.

Por tanto, los hallazgos detectados pueden considerarse falsos positivos o técnicamente ignorables, ya que el uso de include_once responde a una decisión de diseño necesaria para mantener la independencia del sistema y asegurar su portabilidad en entornos donde Composer o autoload no están disponibles.

Cobertura de Código:



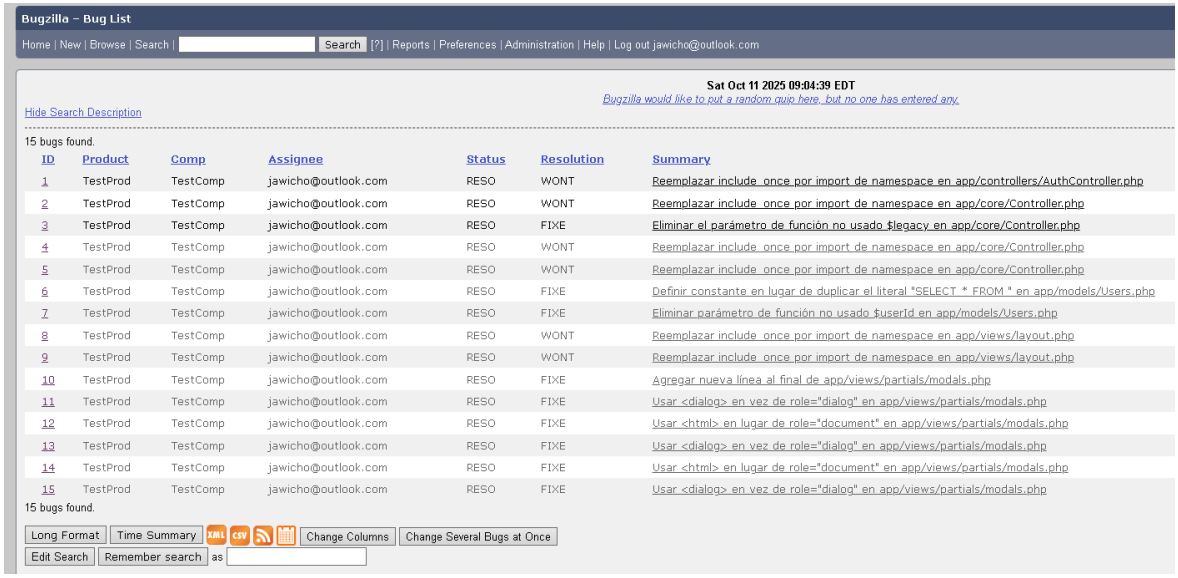
El reporte de SonarQube muestra una cobertura global de 84.8%, lo que significa que la gran mayoría del código del sistema fue ejecutada durante las pruebas unitarias con PHPUnit. Este valor es superior al estándar mínimo recomendado (80%), reflejando una excelente disciplina de pruebas y validación funcional.

Al analizar los archivos, se observa que los módulos principales del sistema alcanzan coberturas de hasta 83.9 %, confirmando que las funciones críticas (autenticación, gestión de usuarios y lógica de negocio) están exhaustivamente verificadas.

Sin embargo, algunos archivos de configuración o clases auxiliares, como app/config/config.php y app/models/Users.php, presentan 0 % de cobertura. Esto es

esperable y aceptable, ya que contienen parámetros estáticos o definiciones sin lógica ejecutable por lo que no requieren pruebas directas.

El registro y control de estos bugs que nos indicó SonarQube se llevó a cabo con la ayuda de Bugzilla, aquí se fueron registrando y actualizando según fueron resueltos o aceptados.



LINK AL VIDEO:

https://drive.google.com/file/d/1fnLjeMRbIzKL3xfl3SrSTB22iujwf3WT/view?usp=drive_link