



UNIVERSIDAD PRIVADA DE TACNA

FACULTAD DE INGENIERÍA

Escuela Profesional de Ingeniería de Sistemas

PROYECTO DE UNIDAD 02:

SISTEMA DE INVENTARIO

Curso: Calidad y Pruebas de Software

Docente: PATRICK JOSÉ CUADROS QUIROGA

Integrantes:

Castañeda Centurión, Jorge Enrique (2021069822)

Cuadros Garcia, Mirian (2021071083)

Lopez Catunta, Brayar Christian (2020068946)

Briceño Diaz, Jorge Luis (2017059611)

Tacna – Perú

2024

Abstract

En un entorno donde la eficiencia y precisión en el control de inventario son fundamentales para el éxito de una organización, este proyecto se centra en desarrollar un sistema de gestión de inventario en PHP que permite agregar usuarios, categorías y productos, así como listarlos de manera organizada según las necesidades específicas de la empresa. El objetivo principal es mejorar la eficiencia operativa y la toma de decisiones informadas a través de la automatización y organización del proceso de gestión de inventario.

1. Antecedentes o introducción

En el contexto actual, se ha implementado con éxito un sistema de gestión de inventario en PHP que permite agregar usuarios, categorías y productos, así como listarlos de manera organizada según las necesidades específicas de la empresa. Este sistema ha mejorado significativamente la eficiencia operativa y la toma de decisiones informadas en relación con el inventario.

2. Título

“Sistema de Gestión de Inventario”

3. Autores

- **Testers:**

Brayar Christian LÓPEZ CATUNTA

Jorge Luis BRICEÑO DÍAZ

Jorge Enrique CASTAÑEDA CENTURION

MIRIAN CUADROS GARCIA

4. Planteamiento del problema

4.1. Problema

Durante la evaluación exhaustiva de nuestro sistema actual de gestión de inventario, hemos identificado varios problemas críticos que están afectando negativamente nuestra eficiencia operativa y nuestra capacidad para tomar decisiones informadas:

Uno de los principales desafíos que enfrentamos es la ineficiencia y pérdida de tiempo en el registro y seguimiento del inventario. Los

procesos manuales utilizados son laboriosos y consumen una cantidad significativa de tiempo, lo que resulta en retrasos en la actualización de datos y en una menor capacidad de respuesta a las demandas del mercado en tiempo real.

Además, la dependencia de métodos manuales para gestionar nuestro inventario ha resultado en una mayor probabilidad de errores humanos. Los errores como la duplicación de datos, omisiones y registros incorrectos están afectando la precisión de nuestros registros de inventario y, por ende, nuestra capacidad para tomar decisiones basadas en datos precisos.

La falta de visibilidad y control centralizado del inventario también es un problema importante. La ausencia de un sistema centralizado nos dificulta tener una visión completa y actualizada de nuestro inventario en tiempo real, lo que impide identificar rápidamente problemas como exceso o escasez de stock, afectando nuestra capacidad para satisfacer las necesidades de los clientes de manera eficiente.

Otro desafío crítico es la dificultad para análisis y generación de informes sobre el inventario. La falta de herramientas adecuadas para analizar nuestros datos de inventario dificulta la generación de informes detallados y análisis significativos. La información disponible no se presenta de manera estructurada ni se adapta a las necesidades específicas de análisis, lo que limita nuestra capacidad para identificar tendencias, patrones y oportunidades de mejora en la gestión de inventario.

Estos problemas están impactando directamente en nuestra capacidad para gestionar eficientemente nuestro inventario, lo que resulta en costos adicionales, retrasos en la entrega de productos y una experiencia del cliente subóptima. Es imperativo abordar estos problemas de manera integral y desarrollar un sistema de gestión de inventario robusto y eficiente para garantizar nuestra competitividad y éxito a largo plazo en el mercado.

4.2. Justificación

- La implementación de un sistema automatizado y centralizado de gestión de inventario se justifica por varios motivos fundamentales que abordan directamente los problemas identificados y benefician a la organización en múltiples aspectos:
- **Optimización de Procesos y Reducción de Errores**
La automatización de los procesos de registro, seguimiento y análisis de inventario permitirá optimizar nuestras operaciones internas. Al eliminar la dependencia de métodos manuales propensos a errores, reduciremos significativamente la posibilidad de duplicaciones, omisiones y registros incorrectos. Esto se traducirá en una mayor precisión en nuestros registros de inventario y una reducción de los costos asociados a errores operativos.
- **Mejora en la Toma de Decisiones y Gestión de Recursos**
Al contar con datos precisos, actualizados y fácilmente accesibles sobre nuestro inventario, podremos tomar decisiones más informadas y estratégicas. La visibilidad mejorada nos permitirá identificar rápidamente tendencias de demanda, ajustar niveles de stock de manera proactiva y optimizar la gestión de recursos para maximizar la eficiencia operativa y minimizar los costos de almacenamiento.
- **Ventaja Competitiva y Experiencia Mejorada para Clientes**
La implementación de un sistema de gestión de inventario eficiente no solo nos posicionará mejor en el mercado al mejorar nuestra capacidad de respuesta y adaptación, sino que también mejorará la experiencia para nuestros clientes. Podremos cumplir con sus expectativas de disponibilidad de productos, tiempos de entrega más rápidos y una atención más personalizada, lo que fortalecerá nuestra relación con ellos y aumentará la lealtad a nuestra marca.

- **Eficiencia en la Toma de Decisiones y Análisis Estratégico**

Con datos actualizados y análisis detallados disponibles de manera instantánea, podremos tomar decisiones estratégicas de manera más eficiente y oportuna. La generación de informes automatizados y personalizables nos permitirá identificar oportunidades de mejora, evaluar el desempeño de productos específicos y ajustar nuestras estrategias de inventario de acuerdo a las demandas del mercado y las necesidades de nuestros clientes.

Beneficios Tangibles:

- **Optimización de Procesos:** La automatización de tareas de registro, seguimiento y análisis de inventario reducirá los tiempos de trabajo y aumentará la eficiencia operativa.
- **Reducción de Errores:** La eliminación de métodos manuales propensos a errores minimizará las discrepancias en los registros y reducirá los costos asociados a correcciones y ajustes.
- **Mejora en la Gestión de Recursos:** La visibilidad mejorada del inventario permitirá una gestión más eficiente de los recursos, optimizando los niveles de stock y reduciendo los costos de almacenamiento.
- **Eficiencia en la Toma de Decisiones:** Datos actualizados y análisis detallados facilitarán la toma de decisiones informadas, permitiendo ajustes rápidos y precisos en las estrategias de inventario.

Beneficios Intangibles:

- **Ventaja Competitiva:** La capacidad de responder rápidamente a las demandas del mercado y ofrecer una experiencia mejorada para los clientes generará una ventaja competitiva significativa.
- **Mejora en la Experiencia del Cliente:** La disponibilidad de productos, tiempos de entrega más rápidos y una atención más

personalizada mejorarán la experiencia general de los clientes, fomentando la fidelidad a la marca.

- **Eficiencia en la Toma de Decisiones Estratégicas:** La disponibilidad de datos precisos y análisis estratégicos facilitará la identificación de oportunidades de mejora y el ajuste ágil de estrategias empresariales.
- **Aumento de la Confiabilidad de los Datos:** La automatización y centralización de los registros de inventario garantizarán la consistencia y fiabilidad de los datos, fortaleciendo la confianza en los sistemas de gestión interna.

4.3. Alcance

El alcance del proyecto se centra en satisfacer las necesidades de los testers, quienes desempeñan un papel crucial en la verificación de la seguridad, fiabilidad y funcionalidad del sistema de gestión de inventario en PHP. Para ello, se incluirán las siguientes áreas de enfoque:

- **Pruebas de Seguridad:**
Los testers llevarán a cabo pruebas exhaustivas para identificar posibles vulnerabilidades y puntos débiles en el sistema. Esto incluye pruebas de penetración, análisis de vulnerabilidades y evaluación de la resistencia a ataques.
- **Pruebas de Funcionalidad:**
Se realizarán pruebas funcionales para verificar que todas las funciones del sistema de gestión de inventario operen correctamente según los requisitos establecidos. Esto incluye la verificación de la creación y gestión de usuarios, categorías y productos, así como la generación de informes y análisis de datos.
- **Pruebas de Rendimiento:**
Se llevarán a cabo pruebas de rendimiento para evaluar la capacidad del sistema para manejar cargas de trabajo variables y mantener tiempos de respuesta aceptables. Esto implica pruebas de carga, estrés y escalabilidad del sistema.
- **Pruebas de Usabilidad:**
Los testers evaluarán la usabilidad y la experiencia del usuario del sistema, asegurando que la interfaz sea intuitiva, fácil de usar y cumpla con los estándares de accesibilidad.
- **Pruebas de Integración:**

Se verificará la integración del sistema de gestión de inventario con otros sistemas o componentes, asegurando que funcione de manera eficiente y sin conflictos.

- **Pruebas de Respaldo y Recuperación:**

Se realizarán pruebas para garantizar que los mecanismos de respaldo y recuperación de datos funcionen correctamente en caso de fallos o pérdida de información.

- **Pruebas de Cumplimiento Normativo:**

Se verificará que el sistema cumpla con las normativas y regulaciones pertinentes en cuanto a seguridad de datos, privacidad y protección de la información.

5. Objetivos

5.1. General

Desarrollar e implementar un sistema de gestión de inventario en PHP que permita gestionar eficientemente los registros de usuarios, categorías, productos y movimientos de inventario, con el fin de mejorar la eficiencia operativa y la toma de decisiones informadas dentro de la organización.

5.2. Específicos

- **Diseñar una Interfaz de Usuario Intuitiva:**

Crear una interfaz de usuario fácil de usar para la gestión de usuarios, categorías y productos.

Implementar funcionalidades para agregar, editar y eliminar usuarios con diferentes roles de acceso.

- **Gestión Eficiente de Inventarios:**

Desarrollar módulos para la creación, modificación y eliminación de categorías de productos.

Permitir el registro automatizado de movimientos de inventario, como compras, ventas, devoluciones y ajustes de stock.

- **Organización y Visualización del Inventario:**

Diseñar herramientas de búsqueda, filtros y organización para facilitar la visualización del inventario por categorías, proveedores, precios, existencias disponibles, etc.

Implementar funciones para generar informes detallados sobre el estado actual del inventario y análisis de datos relacionados.

- **Seguridad y Acceso Controlado:**

Integrar mecanismos de seguridad robustos para proteger la integridad de los datos del inventario y controlar el acceso de los usuarios.

Establecer roles de usuario con diferentes niveles de acceso y permisos.

- **Desarrollo de Funcionalidades Específicas:**
Implementar funcionalidades adicionales según las necesidades específicas de la empresa, como gestión de proveedores, seguimiento de pedidos, alertas de stock bajo, entre otros.
Incluir herramientas de análisis predictivo para pronósticos de demanda y planificación de inventario.
- **Integración con Herramientas de Análisis y Seguridad:**
Integrar el sistema de gestión de inventario con herramientas de análisis como SonarQube y Snyk para verificar la seguridad del código y detectar posibles vulnerabilidades.
Establecer mecanismos de monitoreo y alerta temprana para identificar y resolver problemas de manera proactiva.
- **Pruebas Continuas de Seguridad y Calidad:**
Realizar pruebas periódicas en SonarQube y Snyk para evaluar la seguridad del código, identificar vulnerabilidades y asegurar la calidad del sistema.
Implementar acciones correctivas y preventivas basadas en los resultados de las pruebas en SonarQube y Snyk para mejorar la seguridad y calidad del sistema de manera constante.
- **Capacitación y Soporte:**
Proporcionar capacitación y soporte técnico a los usuarios y administradores del sistema para asegurar su correcto uso y mantenimiento.
Establecer un plan de actualización y mejora continua del sistema basado en retroalimentación de usuarios y análisis de rendimiento.

6. Referentes teóricos

Diagramas de Casos de Uso

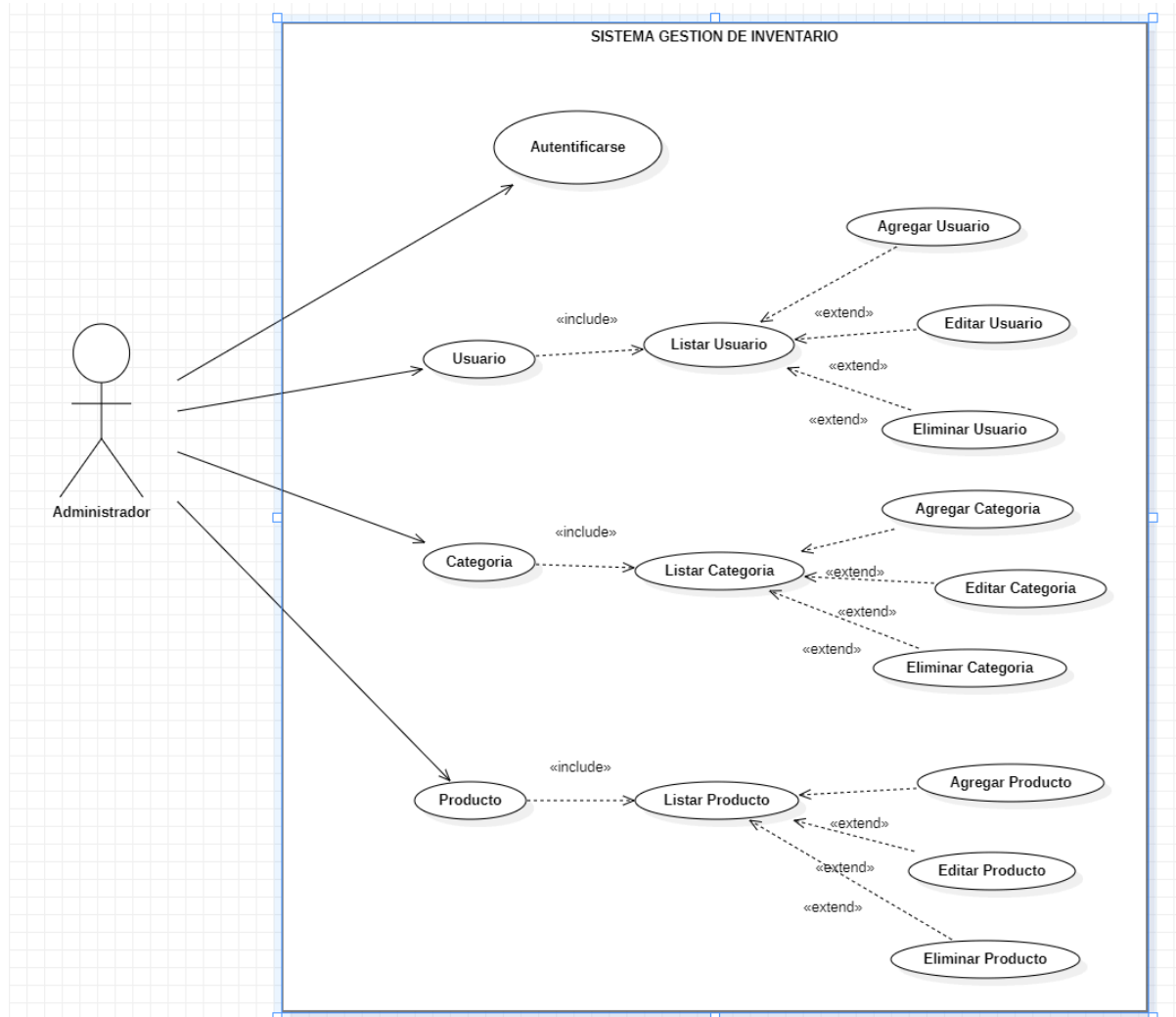


Diagrama de Clases

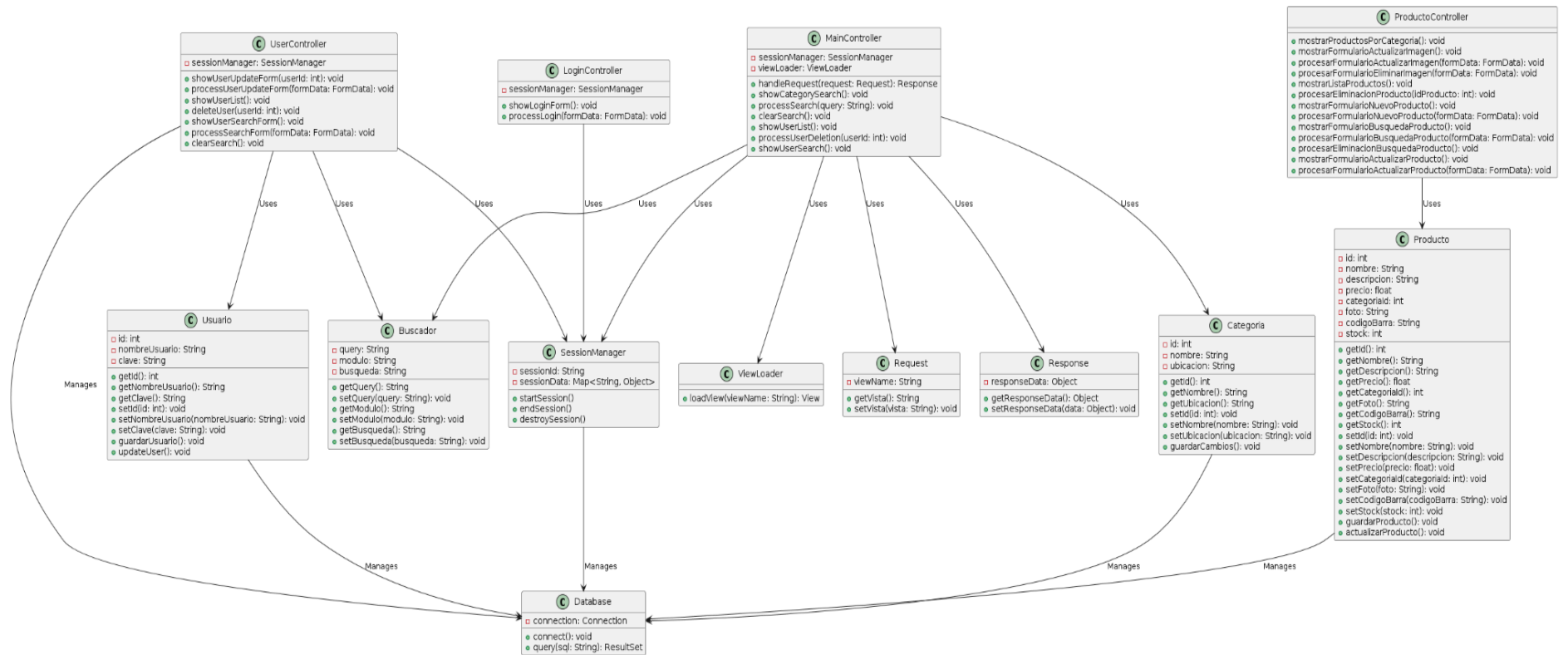
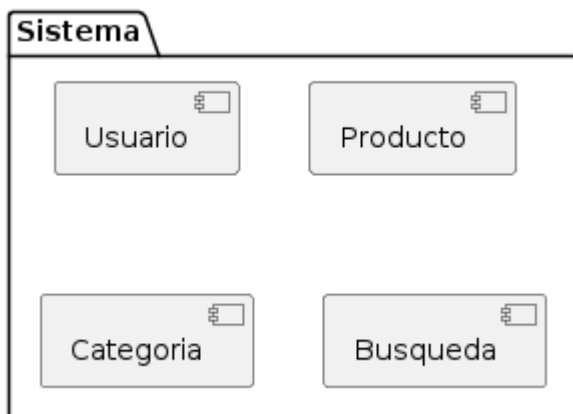
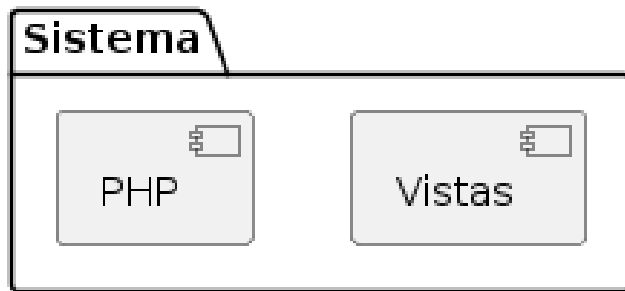


Diagrama de Componentes y Arquitectura



7. Desarrollo de la propuesta (Aqui va el análisis de su aplicación con SonarQube y Snyk, para que les muestre todos los aspectos a mejorar de su aplicación)

7.1. Tecnología de información

- **SonarSource**

Paso 01 : Se esta creando un repositorio público en tu cuenta personal llamado "ProInventario", el cual contiene todos los archivos de tu proyecto y su historial de revisiones. Puedes inicializarlo con un archivo README y un archivo .gitignore, y elegir una licencia para tu código.

Create a new repository
 A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Required fields are marked with an asterisk (*).

Repository template
 Start your repository with a template repository's contents.

Owner * **Repository name ***
 BCZLopezCatunta / ProInventario
 ProInventario is available.

Great repository names are short and memorable. Need inspiration? How about [supreme-palm-tree](#)?

Description (optional)

☒ **Public**
 Anyone on the internet can see this repository. You choose who can commit.

☐ **Private**
 You choose who can see and commit to this repository.

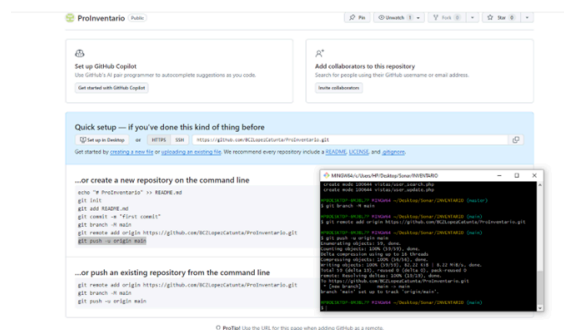
Initialize this repository with:
☐ Add a README file
 This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore
 Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

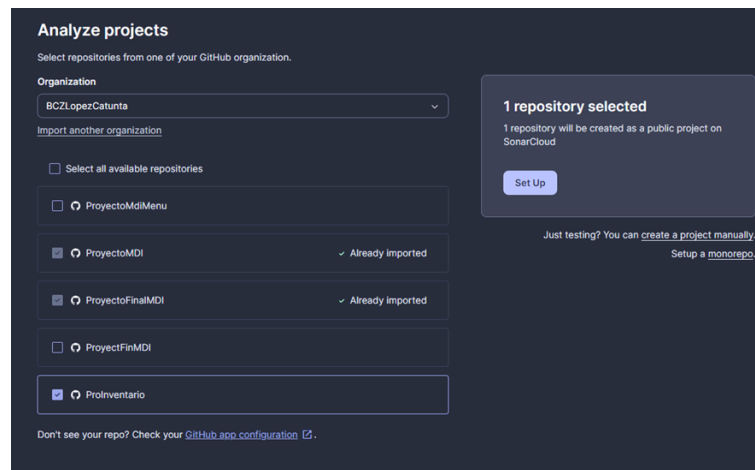
Choose a license
 A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

① You are creating a public repository in your personal account.

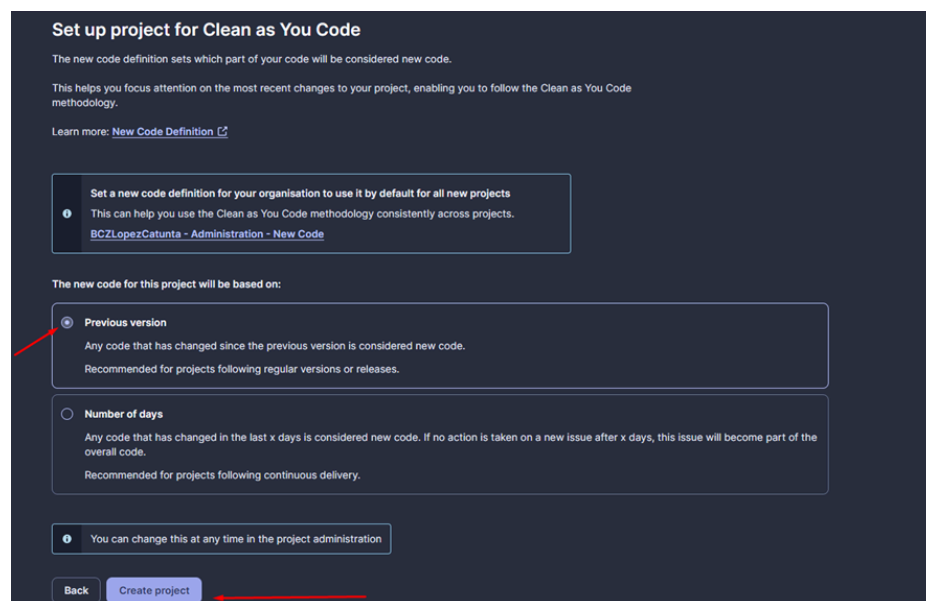
Paso 02: Se ha creado un nuevo repositorio en GitHub con la dirección URL "https://github.com/BCZLopezCatunta/ProInventario.git" y ha cambiado la rama local existente a "main". Luego, ha agregado este repositorio a su sistema local en la carpeta "INVENTARIO" en el escritorio y ha establecido una conexión remota con el repositorio de GitHub.



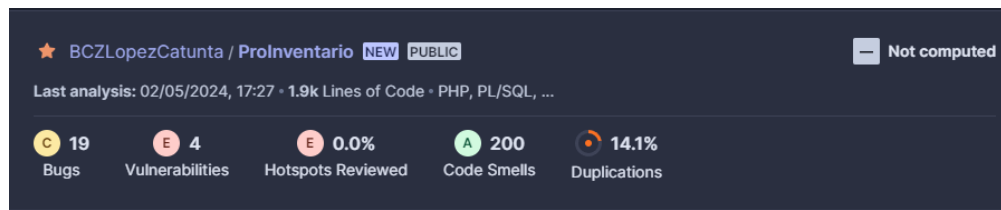
Paso 03: Seleccionamos el repositorio "ProInventario" de la organización BCZLopezCatunta en GitHub para configurarlo en SonarCloud. Ahora, después de seleccionar el repositorio deseado, das clic en la opción "Setup" para continuar con la configuración y prepararlo para su análisis en SonarCloud. Este proceso te permitirá evaluar la calidad del código de tu proyecto "ProInventario" utilizando las herramientas y funcionalidades de SonarCloud.



Paso 04: Seleccionamos "Previous Version" en el contexto dado, estás especificando que la nueva codificación para el proyecto se basará en la versión anterior del código. Esto significa que cualquier parte del código que haya sido modificada desde la última versión se considerará como "nuevo código". Esta opción es recomendada para proyectos que siguen versiones o lanzamientos regulares. Una vez que has seleccionado esta opción, al hacer clic en "Create project"



Paso 05: El Análisis de código realizado por SonarSource en el proyecto "Proinventario" de BCZLopezCatunta muestra que el proyecto tiene la calidad del código se evalúa en un 14,1% Duplications, lo que sugiere que hay espacio para mejorar. En cuanto a la seguridad, se han detectado 19 bugs y 4 vulnerabilidades.



Paso 06: Se pasa a resolver las vulnerabilidades

- En el archivo index.php del porque el problema:
Impacto potencial de las inyecciones de inclusión

Si una aplicación es vulnerable a inyecciones de inclusión, puede ser susceptible a ataques dirigidos al servidor subyacente. Un usuario malintencionado puede crear solicitudes que filtren datos valiosos o logren la ejecución remota de código en el sitio web del servidor. El atacante puede llevar a cabo estos ataques sin depender de requisitos previos.

El impacto de la vulnerabilidad depende de las medidas de control de acceso adoptadas en el sistema operativo de destino. En el peor de los casos, el proceso se ejecuta con privilegios de root, lo que significa que cualquier comando o programa del sistema operativo puede verse afectado.

Escenarios del mundo real que ilustran el impacto

a. Denegación de servicio y fuga de datos:

El ataque tiene como objetivo interrumpir las actividades de la organización y beneficiarse de la filtración de datos.

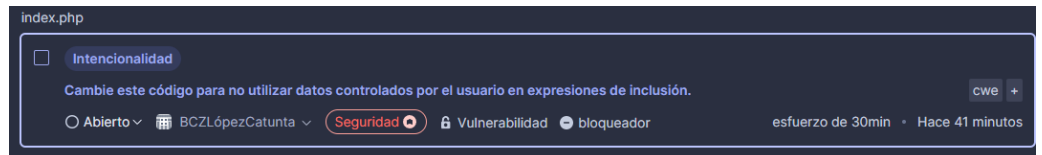
El atacante puede descargar datos del servidor interno para venderlos, modificar datos, enviar malware, detener servicios o agotar recursos.

Esta amenaza es especialmente peligrosa si la organización atacada no tiene un plan de recuperación ante desastres.

b. Escalada y pivote de privilegios de root

En este caso, el atacante puede elevar sus privilegios a nivel administrativo y atacar otros servidores.

El impacto dependerá de la concentración de la empresa objetivo en su Defensa en Profundidad.



- En el archivo principal.php del porque el problema:

Impacto potencial de la falta de contraseña en una base de datos

Cuando una base de datos no requiere una contraseña para la autenticación, se abre la posibilidad de acceso y manipulación de los datos almacenados por parte de cualquier persona. Explotar esta vulnerabilidad puede tener varios impactos potenciales:

a. Acceso no autorizado a datos confidenciales:

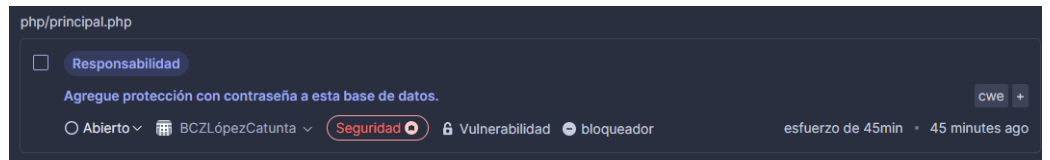
La ausencia de una contraseña para la autenticación permite que personas no autorizadas obtengan acceso a datos confidenciales, como información de identificación personal, registros financieros y propiedad intelectual. Esto puede provocar robo de identidad, pérdidas financieras y daños a la reputación.

b. Compromiso de la integridad del sistema:

Personas no autorizadas pueden obtener acceso ilimitado a la base de datos, lo que compromete la integridad del sistema. Los atacantes pueden inyectar código malicioso, alterar configuraciones o manipular datos, lo que puede provocar fallos en el funcionamiento del sistema y exposición a mayores riesgos de seguridad.

c. Modificaciones o eliminaciones no deseadas:

La falta de una contraseña para el acceso a la base de datos permite que cualquier persona realice modificaciones o eliminaciones de los datos almacenados. Esto plantea un riesgo significativo, ya que los cambios no autorizados pueden provocar daños en los datos, pérdida de información crítica o la introducción de contenido malicioso.



- En el archivo categoria_producto.php del porque el problema:

Impacto Potencial de las Inyecciones de Bases de Datos

Las inyecciones de bases de datos, como las inyecciones SQL, representan una seria amenaza para la seguridad de las aplicaciones web. Cuando una aplicación es vulnerable a este tipo de ataques, los atacantes pueden manipular consultas de base de datos para ejecutar comandos maliciosos y comprometer la integridad de los datos. Esta vulnerabilidad expone a la aplicación a una serie de riesgos, que pueden tener impactos significativos en la seguridad y el funcionamiento de las bases de datos afectadas.

Suplantación de Identidad y Manipulación de Datos

La explotación de una consulta maliciosa puede conducir a una escalada de privilegios o a la fuga de datos sensibles almacenados en las bases de datos. Esta amenaza representa uno de los impactos más extendidos, ya que los atacantes pueden obtener acceso no autorizado a información confidencial, lo que a su vez puede resultar en la suplantación de identidad y manipulación de datos.

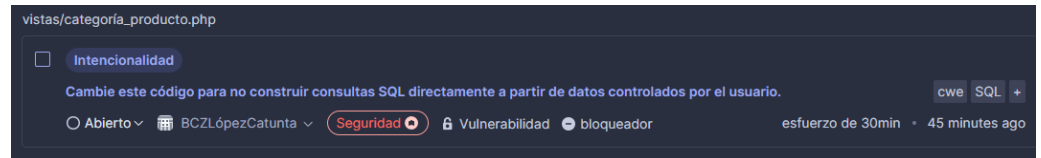
Eliminación de Datos y Denegación de Servicio

Los atacantes pueden utilizar consultas maliciosas para eliminar datos de las bases de datos afectadas, lo que puede tener consecuencias devastadoras si la organización no cuenta con un plan de recuperación ante desastres. Esta amenaza es particularmente insidiosa, ya que puede resultar en la pérdida irreversible de información crítica y en la interrupción del servicio.

Encadenamiento de Inyecciones de Bases de Datos con Otras Vulnerabilidades

Los atacantes que explotan las inyecciones SQL suelen buscar maximizar sus ganancias mediante el aprovechamiento de otras vulnerabilidades. Esto puede incluir la filtración de secretos almacenados sin cifrar en las bases de datos, comprometiendo otros componentes del sistema, o incluso la ejecución remota

de código si el sistema operativo del servidor o los permisos de la base de datos están mal configurados.



- En el archivo product_img.php del porque el problema:

Las inyecciones de bases de datos, como las inyecciones SQL, representan una seria amenaza para la seguridad de las aplicaciones web. Cuando una aplicación es vulnerable a este tipo de ataques, los atacantes pueden manipular consultas de base de datos para ejecutar comandos maliciosos y comprometer la integridad de los datos. Esta vulnerabilidad expone a la aplicación a una serie de riesgos, que pueden tener impactos significativos en la seguridad y el funcionamiento de las bases de datos afectadas.

Suplantación de Identidad y Manipulación de Datos

La explotación de una consulta maliciosa puede conducir a una escalada de privilegios o a la fuga de datos sensibles almacenados en las bases de datos. Esta amenaza representa uno de los impactos más extendidos, ya que los atacantes pueden obtener acceso no autorizado a información confidencial, lo que a su vez puede resultar en la suplantación de identidad y manipulación de datos.

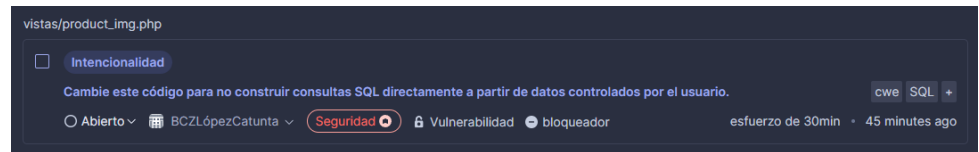
Eliminación de Datos y Denegación de Servicio

Los atacantes pueden utilizar consultas maliciosas para eliminar datos de las bases de datos afectadas, lo que puede tener consecuencias devastadoras si la organización no cuenta con un plan de recuperación ante desastres. Esta amenaza es particularmente insidiosa, ya que puede resultar en la pérdida irreversible de información crítica y en la interrupción del servicio.

Encadenamiento de Inyecciones de Bases de Datos con Otras Vulnerabilidades

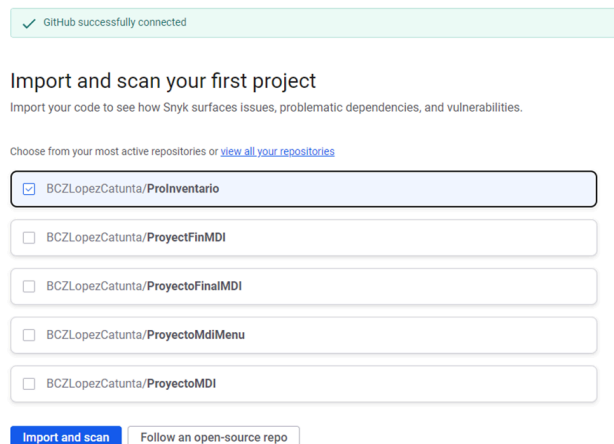
Los atacantes que explotan las inyecciones SQL suelen buscar maximizar sus ganancias mediante el aprovechamiento de otras vulnerabilidades. Esto puede incluir la filtración de secretos

almacenados sin cifrar en las bases de datos, comprometiendo otros componentes del sistema, o incluso la ejecución remota de código si el sistema operativo del servidor o los permisos de la base de datos están mal configurados.

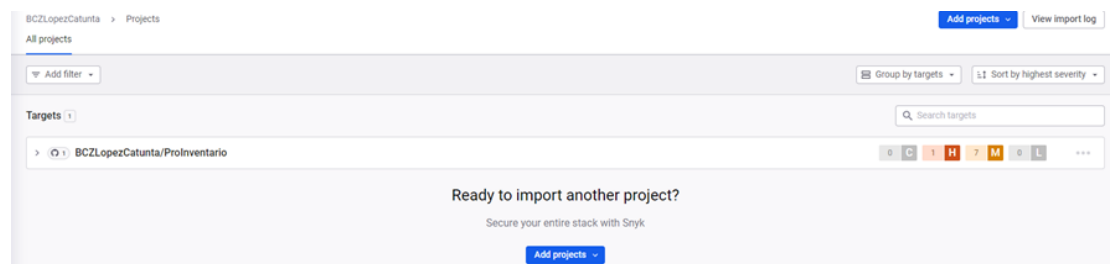


- Snyk

Repositorio "ProInventario" de "BCZLopezCatunta", el texto proporcionado describe los pasos para conectar el repositorio "ProInventario" de GitHub a Snyk y escanear el proyecto para detectar problemas de dependencias y vulnerabilidades.



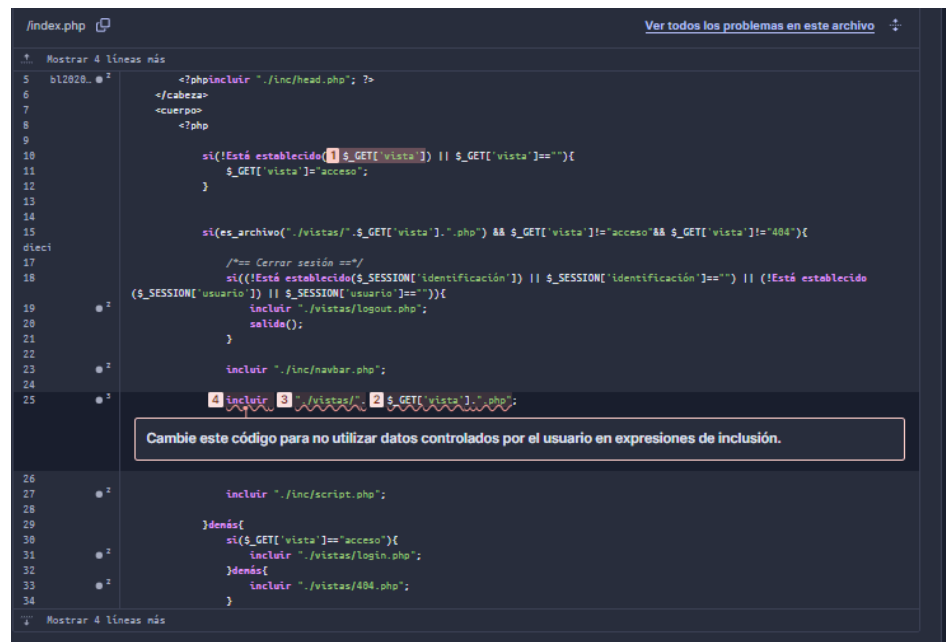
En este punto el sistema de gestión de proyectos personalizado para el usuario "BCZLopezCatunta", con una sección principal para "Todos los proyectos" que pueden ser filtrados y agrupados por objetivos, como "BCZLopezCatunta/Proinventario", y una selección de botones y enlaces para realizar acciones como "Añadir filtro", "Añadir proyectos", "Ver imp", "Búsqueda de objetivos Q" y "Ordenar por objetivos de mayor a menor". También se muestran algunas métricas o contadores con valores "C 1", "H 7" y "M 0L".



7.2. Metodología, técnicas usadas

- SonarSource

a)El problema index.php:



The screenshot shows a code editor with a dark theme. The file is named `index.php`. The code is PHP and includes several conditional checks and file inclusions. A SonarSource warning is displayed over line 25, which contains an `include` statement. The warning message is: "Cambia este código para no utilizar datos controlados por el usuario en expresiones de inclusión." (Change this code to not use user-controlled data in inclusion expressions). The code snippet for line 25 is: `include $_GET['vista'] . ".php";`. The warning is highlighted with a yellow background and a red border. The code editor also shows line numbers on the left and a status bar at the bottom.

```
1. Mostrar 4 líneas más
5 b12020... <?phpinclude "..\inc/head.php"; ?>
6 </cabeza>
7 <cuero>
8 <?php
9
10 si(!Está establecido($_GET['vista']) || $_GET['vista']==""){
11     $_GET['vista']="acceso";
12 }
13
14
15 si(es_archivo("../vistas/".$_GET['vista'].".php") && $_GET['vista']!="acceso" && $_GET['vista']!="404"){
16     /*= Error sesión =*/
17     si(!Está establecido($_SESSION['identificación']) || $_SESSION['identificación']=="") || (!Está establecido
18     ($_SESSION['usuario']) || $_SESSION['usuario']==""){
19         include "..\vistas/logout.php";
20         salir();
21     }
22
23     include "..\inc/navbar.php";
24
25     include $_GET['vista'] . ".php";
26
27     include "..\inc/script.php";
28
29 }denás{
30     si($_GET['vista']=="acceso"){
31         include "..\vistas/login.php";
32     }denás{
33         include "..\vistas/404.php";
34     }
35 }
36
37 4. Mostrar 4 líneas más
```

Para evitar la vulnerabilidad de inyección en las expresiones de inclusión en PHP, es fundamental validar la entrada del usuario antes de utilizarla en una expresión de inclusión. Esto se logra creando una lista de archivos autorizados y seguros que la aplicación puede cargar con expresiones de inclusión. Si la entrada del usuario no coincide con esta lista, debe ser rechazada para garantizar la seguridad.

En el código proporcionado, se define una lista de archivos permitidos y seguros que la aplicación puede incluir. Luego, se verifica si la vista solicitada está en la lista permitida y si existe como archivo. Si cumple con estas condiciones, se incluye el archivo correspondiente; de lo contrario, se carga una página de error.

Es importante destacar que esta validación debe realizarse en el lado del servidor para garantizar su efectividad. Al implementar esta técnica, se reduce significativamente el riesgo de ataques de inyección en las expresiones de inclusión, lo que contribuye a fortalecer la seguridad de la aplicación.

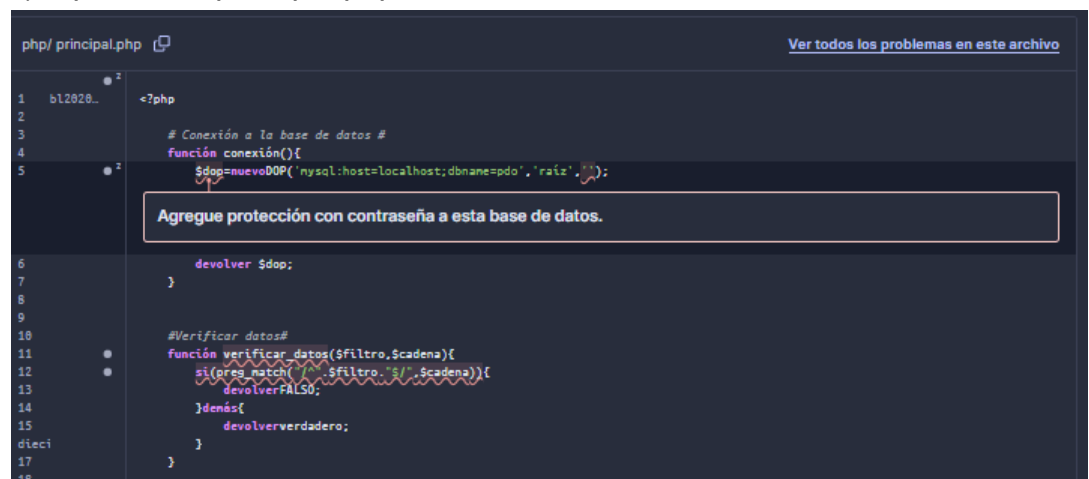
- **Solución del Problema:**

```
index.php
1  <?php
2  define('VISTAS_PATH', "../vistas/");
3
4  $INCLUDE_ALLOW_LIST = [
5      'casa.php',
6      'tablero.php',
7      'perfil.php',
8      'configuración.php'
9  ];
10
11 // Función para verificar la sesión activa
12 function verificarSesionActiva() {
13     if (empty($_SESSION['identificación']) || empty($_SESSION['us
14         require_once VISTAS_PATH . 'cerrar sesión.php';
15         exit();
16     }
17 }
18
19 // Obtener la vista solicitada
20 $vista = $_GET["vista"] ?? '';
21
22 // Verificar si la vista solicitada está en la lista permitida y
23 if ($vista && in_array($vista, $INCLUDE_ALLOW_LIST) && is_file(VI
24     require_once VISTAS_PATH . $vista . ".php";
25 } else {
26     // Si la vista no está permitida o no existe, cargar la página
27     require_once VISTAS_PATH . '404.php';
28 }
29
30 // Iniciar sesión si la vista solicitada es 'acceso'; de lo contr
31 if ($vista === 'acceso') {
32     require_once VISTAS_PATH . 'iniciar sesión.php';
33 } else {
34     verificarSesionActiva();
35 }
36
37 // Incluir la barra de navegación
38 require_once "../inc/navbar.php";
39 ?>
```

- **Se utilizan las siguientes metodologías y técnicas:**

- Lista de Permitidos: Se define una lista de archivos permitidos y seguros que la aplicación puede incluir. Esto se logra mediante la variable `$INCLUDE_ALLOW_LIST`, que contiene los nombres de los archivos autorizados, como 'casa.php', 'tablero.php', 'perfil.php' y 'configuración.php'.
- Validación de Sesión Activa: Se implementa una función llamada `verificarSesionActiva()` que verifica si la sesión del usuario está activa. Si la identificación del usuario o el usuario mismo están vacíos, se requiere el archivo 'cerrar sesión.php' desde la ruta de vistas y se finaliza la ejecución del script.
- Validación de Vista Solicitada: Se verifica si la vista solicitada por el usuario está en la lista permitida y si existe como archivo. Esto se realiza mediante la variable `$vista`, que obtiene el valor del parámetro "vista" en la URL. Si la vista está en la lista permitida y es un archivo válido, se incluye el archivo correspondiente; de lo contrario, se carga la página '404.php'.
- Inclusión Condicional de Archivos: Dependiendo de la vista solicitada, se realizan inclusiones condicionales de archivos. Si la vista solicitada es 'acceso', se requiere el archivo 'iniciar sesión.php'; de lo contrario, se verifica la sesión activa llamando a la función `verificarSesionActiva()`.
- Inclusión de Barra de Navegación: Finalmente, se incluye la barra de navegación desde el archivo `../inc/navbar.php`.

b)El problema principal.php:



```
1  b12020... <?php
2
3      # Conexión a la base de datos #
4      función conexión(){
5          $dop=nuevoDOP('mysql:host=localhost;dbname=pdo','raiz','');
6
7          devolver $dop;
8      }
9
10     #Verificar datos#
11     función verificar_datos($filtro,$cadena){
12         si(preg_match('/^'.$filtro.$'/',$cadena)){
13             devolverFALSO;
14         }deñs{
15             devolververdadero;
16         }
17     }
18
```

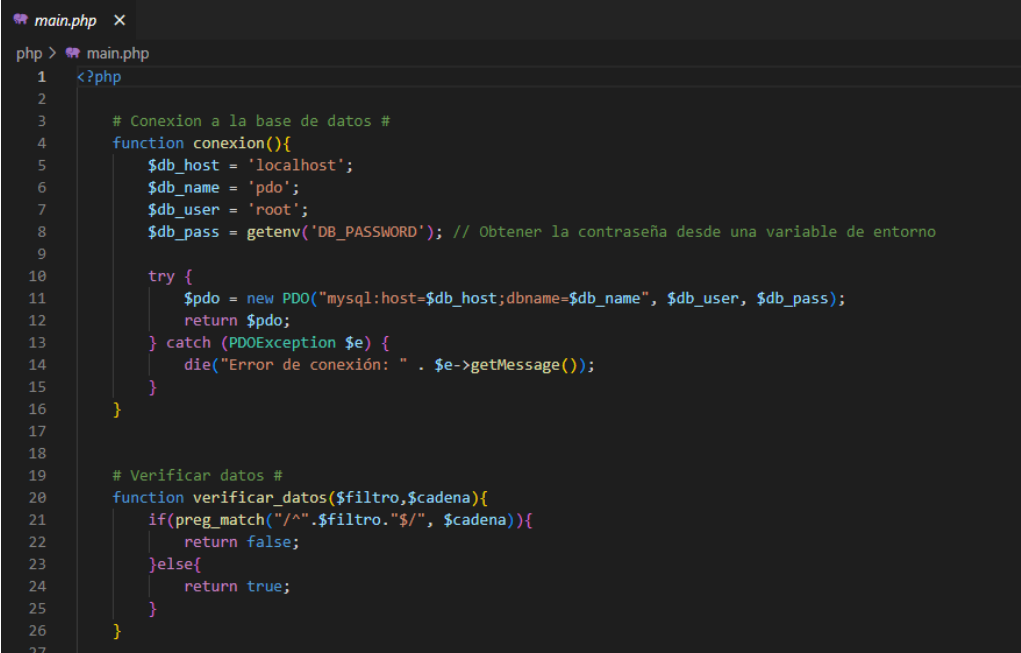
- Solución del Problema:

El código presenta una vulnerabilidad al utilizar una contraseña vacía para conectarse a una base de datos MySQL. Para solucionar esta vulnerabilidad, es recomendable utilizar una contraseña segura recuperada de una variable de entorno, como `MYSQL_SECURE_PASSWORD`, que se establece durante la implementación y debe ser única y robusta para cada base de datos.

Es importante evitar el uso de contraseñas codificadas en el código, ya que esto puede exponer la seguridad de la aplicación a riesgos significativos. Las contraseñas codificadas pueden ser descubiertas fácilmente por desarrolladores o atacantes, lo que podría resultar en acceso no autorizado a la base de datos y posibles violaciones de datos. Además, el uso de contraseñas codificadas dificulta la flexibilidad para cambiar las contraseñas sin modificar el código, lo que puede generar problemas de control de versiones y representar un riesgo para la seguridad.

Para mitigar estos riesgos, se recomienda utilizar métodos seguros para almacenar y recuperar contraseñas, como el uso de variables de entorno, archivos de configuración o sistemas seguros de administración de claves. Estas prácticas permiten una mayor seguridad, flexibilidad y separación de información

confidencial del código base, lo que contribuye a la protección de los datos y a la prevención de posibles vulnerabilidades.



```
main.php X
php > main.php
1  <?php
2
3  # Conexion a la base de datos #
4  function conexion(){
5      $db_host = 'localhost';
6      $db_name = 'pdo';
7      $db_user = 'root';
8      $db_pass = getenv('DB_PASSWORD'); // Obtener la contraseña desde una variable de entorno
9
10     try {
11         $pdo = new PDO("mysql:host=$db_host;dbname=$db_name", $db_user, $db_pass);
12         return $pdo;
13     } catch (PDOException $e) {
14         die("Error de conexión: " . $e->getMessage());
15     }
16 }
17
18
19 # Verificar datos #
20 function verificar_datos($filtro,$cadena){
21     if(preg_match("/^".$filtro."$/", $cadena)){
22         return false;
23     }else{
24         return true;
25     }
26 }
27
```

- **Se utilizan las siguientes metodologías y técnicas:**

- **Uso de Variables de Entorno:** Se utiliza la función `getenv()` para recuperar la contraseña de la base de datos desde una variable de entorno llamada 'DB_PASSWORD'. Esto asegura que la contraseña no esté codificada en el código y se obtenga de una fuente externa, lo que mejora la seguridad.
- **Conexión Segura a la Base de Datos:** Al utilizar la contraseña recuperada de la variable de entorno para establecer la conexión a la base de datos, se garantiza que se esté utilizando una contraseña segura al conectarse a la base de datos MySQL. Esto es fundamental para proteger la integridad de los datos y prevenir posibles violaciones de seguridad.
- **Manejo de Excepciones:** Se utiliza un bloque `try-catch` para manejar posibles errores de conexión a la base de datos. Esto asegura que cualquier error sea capturado y manejado adecuadamente, lo que contribuye a la robustez y estabilidad del sistema.

c) El problema categoria_producto.php:

```
vistas/ categoria_producto.php Ver todos los problemas en este archivo
+ Mostrar 23 líneas más
24 bl2020... $categorias=nulo;
25 ?>
26 </div>
27 <div class="column">
28 <?php
29     3 $categoria_id= (Esté establecido(1 $_GET['categoria ID'])) ? 2 $_GET['categoria ID']:0;
30
31     /*== Verificando categoria ==*/
32     $check_categoria=conexion();
33     $check_categoria->consultar( 3 'SELECCIONAR * DE categoria DONDE categoria_id= 4
34     $categoria_id');
35
36     Cambie este código para no construir consultas SQL directamente a partir de datos controlados por el usuario.
37
38     si($check_categoria->ContarFilas())>0){
39
40         $check_categoria=$check_categoria->buscar();
41
42         echo '
43         <h2 class="titulo tiene-texto-centrado">'. $check_categoria['categoria_nombre']. '</h2>
44         <p class="ha-text-centered pb-6">'. $check_categoria['categoria_ubicacion']. '</p>
45         ';
```

- **Solución del Problema:**

El código proporcionado presenta una vulnerabilidad de inyección de SQL, ya que los datos controlados por el usuario se insertan directamente en una cadena de consulta, lo que puede permitir a un atacante realizar consultas maliciosas. Para solucionar esta vulnerabilidad, se recomienda utilizar declaraciones preparadas en lugar de construir consultas SQL directamente a partir de datos controlados por el usuario.

La solución compatible implica el uso de declaraciones preparadas, donde los valores de los parámetros se pasan por separado, lo que evita la posibilidad de inyección de SQL. Al utilizar declaraciones preparadas, la lógica de la consulta se compila antes de que se pasen los valores de los parámetros, lo que evita que los datos controlados por el usuario afecten la consulta subyacente.

```
>>
</div>
<div class="column">
<?php
    $categoria_id = (isset($_GET['category_id'])) ? $_GET['category_id'] : 0;

    /*== Verificando categoria ==*/
    $pdo = conexion();
    $statement = $pdo->prepare("SELECT * FROM categoria WHERE categoria_id = :categoria_id");
    $statement->bindParam(':categoria_id', $categoria_id, PDO::PARAM_INT);
    $statement->execute();
    $result = $statement->fetch(PDO::FETCH_ASSOC);
```

- **Se utilizan las siguientes metodologías y técnicas:**

Se utiliza la técnica de declaración preparada para protegerse contra la inyección de SQL. En lugar de construir la consulta SQL directamente a partir de datos controlados por el usuario, se utilizan marcadores de posición en la consulta y luego se enlazan los valores de los parámetros a estos marcadores de posición. Esto evita que los datos controlados por el usuario afecten la lógica de la consulta y previene posibles ataques de inyección de SQL.

Además, el código utiliza la función `isset()` para verificar si el parámetro `'category_id'` está presente en la URL antes de asignarlo a la variable `$categoria_id`. Esto ayuda a evitar errores de ejecución si el parámetro no está presente en la URL.

d) El problema `product_img.php`:



```
7 <?php
8     incluir "../inc/btn_back.php";
9
10    requerir una vez "../php/principal.php";
11
12    $identificación= (Está establecido(1 $_GET['product_id_up'])) ? 2 $_GET['product_id_up']:0;
13
14    /*== Verificando producto ==*/
15    $check_producto=conexión();
dieci    $check_producto->consulta(5 "SELECCIONAR * DEL producto DONDE producto_id= 4 $identificación");
16
17
18    si($check_producto->ContarFilas())>0){
19        $datos=$check_producto->buscar();
20    }>
21
22    <div class="form-rest mb-6 mt-6"></div>
23
24    <div clase="columnas">
25        <div class="columna es-dos-quintos">
```

- **Solución del Problema:**

El código proporcionado presenta una vulnerabilidad de inyección de SQL, ya que los datos controlados por el usuario se insertan directamente en una cadena de consulta, lo que puede permitir a un atacante realizar consultas maliciosas. Para solucionar esta vulnerabilidad, se recomienda utilizar declaraciones preparadas en lugar de construir consultas SQL directamente a partir de datos controlados por el usuario.

La solución compatible implica el uso de declaraciones preparadas, donde los valores de los parámetros se pasan por

separado, lo que evita la posibilidad de inyección de SQL. Al utilizar declaraciones preparadas, la lógica de la consulta se compila antes de que se pasen los valores de los parámetros, lo que evita que los datos controlados por el usuario afecten la consulta subyacente.

```
<?php
include_once "../inc/btn_back.php";
require_once "../php/main.php";

$id = (isset($_GET['product_id_up'])) ? $_GET['product_id_up'] : 0;
/*== Verificando producto ==*/
$pdo = conexion();
$stmt = $pdo->prepare("SELECT * FROM producto WHERE producto_id = :id");
$stmt->bindParam(':id', $id, PDO::PARAM_INT);
$stmt->execute();
$result = $stmt->fetch(PDO::FETCH_ASSOC);

if($check_producto->rowCount()>0){
    $datos=$check_producto->fetch();
}
?>
```

- **Se utilizan las siguientes metodologías y técnicas:**

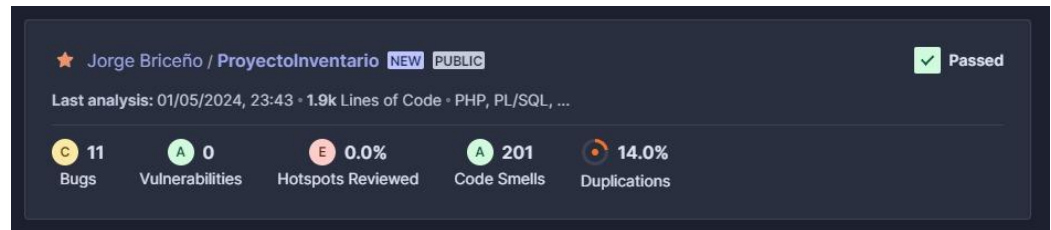
El código proporcionado utiliza la técnica de declaración preparada para protegerse contra la inyección de SQL. En lugar de construir la consulta SQL directamente a partir de datos controlados por el usuario, se utilizan marcadores de posición en la consulta y luego se enlazan los valores de los parámetros a estos marcadores de posición. Esto evita que los datos controlados por el usuario afecten la lógica de la consulta y previene posibles ataques de inyección de SQL.

Además, el código utiliza la función `isset()` para verificar si el parámetro `'product_id_up'` está presente en la URL antes de asignarlo a la variable `$id`. Esto ayuda a evitar errores de ejecución si el parámetro no está presente en la URL.

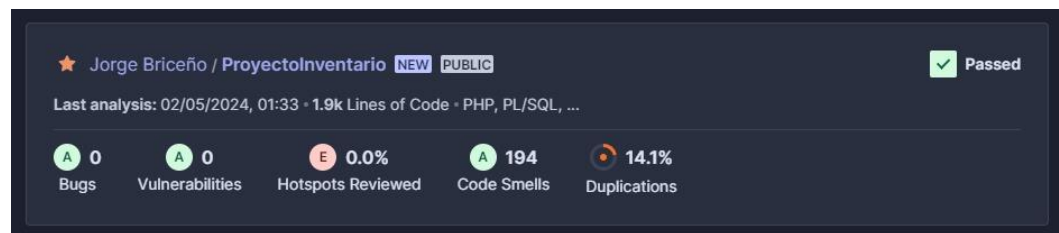
e) Solución del Problema:

- Existencia de bugs: 14.0% (C11)
- Vulnerabilidades: 0.0%
- Hotspots revisados: 201
- Código con olor a bug: existente
- Duplicaciones: presentes

El análisis reveló la presencia de bugs y código con olor a bug, así como duplicaciones en el código fuente del proyecto. Además, se identificaron 201 hotspots que requieren revisión. Por otro lado, no se encontraron vulnerabilidades en el código.

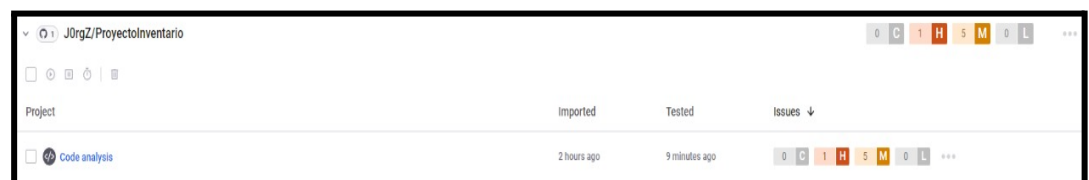


El análisis más reciente de Jorge Briceño / ProyectoInventario se llevó a cabo el 02/05/2024 a las 01:33 y muestra un total de 1.9k líneas de código en PHP, PL/SQL y otras lenguajes. Se encontró que el 14.1% del código analizado contenía errores (bugs) y vulnerabilidades, aunque ningún hotspot crítico fue identificado (0.0%). Se revisaron 194 puntos calientes y se encontraron problemas de código duplicado.



- **Snyk**

De acuerdo del interfaz de proyecto, específicamente un repositorio llamado "ProyectoInventario" por un usuario llamado "JorgZ". El proyecto ha sido importado y probado, y actualmente no hay problemas críticos (C), hay un problema de alta prioridad (H) y cinco problemas de baja prioridad (M o L).



- Semgrep

Dashboard

Projects

Code

Secrets

Supply Chain

Rules

Get Started 25%

Feedback

Settings

Docs

Help

oo

Dashboard

All projectsAll time

Supply Chain

0 REACHABLE0 UNREACHABLE0 UNDETERMINED

Most vulnerabilities

Project name	Reachable	Unreachable	Undetermined
No projects found Try adjusting your filters			

Code

29 HIGH SEVERITY65 OPEN FINDINGS0 % PR/MR FIX RATE

Most findings

Project name	Open findings	High severity	Fix rate
JOrgZ/ProyectoInventario	65	29	0%
JOrgZ/Asistencia	-	-	-
JOrgZ/Proyecto	-	-	-
JOrgZ/PRUEBA_BRANCH	-	-	-
-	-	-	-

Rules summary

Rule	Ignored	Fix rate	Total
tainted-sql-string	0%	0%	29
taint-unsafe-echo-tag	0%	0%	17
tainted-filename	0%	0%	13
unlink-use	0%	0%	5
unsafe-document-method	0%	0%	1