UNIVERSIDAD PRIVADA DE TACNA

FACULTAD DE INGENIERÍA

Escuela Profesional de Ingeniería de Sistemas

Proyecto de implementación de un sistema de ventas para grandes almacenes

Curso: Calidad y Pruebas de Software

Docente: Ing. *Patrick Cuadros Quiroga*

Integrantes:

Edward Hernan Apaza Mamani	***(2018060915)***
Ronal Daniel Lupaca Mamani	***(2020067146)***
Carlos Andrés Escobar Rejas	***(2021070016)***
Aarón Pedro Paco Ramos	***(2018000654)***

Tacna – Perú

2024

Descripción

Descripción del Proyecto: " Implementación de un sistema de ventas para grandes almacenes "

Este proyecto se centra en la implementación de un sistema de ventas, diseñado específicamente para grandes almacenes o compañías como Plaza Vea. El sistema permite la recepción y procesamiento de ventas a través de múltiples cajas de atención, facilitando un control eficiente de productos, generación de recibos, gestión de usuarios para mayoristas y administración de personal. La duración del proyecto es de 90 días, comenzando el 1 de julio y finalizando el 28 de septiembre. El objetivo principal es mejorar la eficiencia en la gestión de ventas y la administración de productos en grandes almacenes.

Abstract

Project Description: " Implementation of a Sales System for Large Warehouses "

This project focuses on implementing a sales system specifically designed for large warehouses or companies like Plaza Vea. The system allows for the reception and processing of sales through multiple service counters, facilitating efficient product control, receipt generation, user management for wholesalers, and personnel administration. The project duration is 90 days, starting from July 1st to September 28th. The primary objective is to enhance efficiency in sales management and product administration in large warehouses.

Antecedentes o introducción

El presente proyecto se enfoca en la implementación de un sistema de ventas avanzado, especialmente diseñado para grandes almacenes y empresas de retail como Plaza Vea. La motivación principal detrás de este desarrollo es la necesidad de modernizar y optimizar los procesos de ventas y gestión de productos en estos entornos, donde la alta demanda y la variedad de productos requieren soluciones tecnológicas robustas y eficientes.

Actualmente, muchos grandes almacenes enfrentan desafíos significativos en la gestión de sus ventas y productos debido al uso de sistemas tradicionales que son propensos a errores y resultan ineficientes. La implementación de un sistema de ventas moderno permitirá mejorar estos procesos mediante la automatización y el uso de tecnologías avanzadas. Este sistema integrará múltiples funcionalidades clave, tales como la recepción y procesamiento de ventas en diversas cajas de atención, control de inventarios en tiempo real, generación automática de recibos, gestión de usuarios mayoristas, y administración del personal.

Al adoptar un enfoque integral y utilizando herramientas de análisis y pruebas de software como Snyk, SonarQube y SonarCloud, el proyecto garantiza que el sistema sea seguro, eficiente y fácil de usar. Esto no solo mejorará la experiencia de compra de los clientes, sino que también optimizará la operatividad interna de los almacenes, reduciendo errores humanos y aumentando la eficiencia operativa. En última instancia, este proyecto busca transformar la manera en que los grandes almacenes gestionan sus ventas, adaptándose a las demandas del mercado moderno y asegurando un servicio de alta calidad para sus clientes.

Título

Implementación de un sistema de ventas para grandes almacenes.

Autores

Edward Hernan Apaza Mamani 2018060915

Ronal Daniel Lupaca Mamani 2020067146

Carlos Andrés Escobar Rejas 2021070016

Aarón Pedro Paco Ramos 2018000654

Planteamiento del problema

Problema

La gestión de ventas en grandes almacenes presenta desafíos significativos debido a la gran cantidad de productos y transacciones diarias. El uso de sistemas tradicionales resulta en ineficiencias, errores humanos y tiempos de espera prolongados para los clientes.

Justificación

La implementación de un sistema de ventas avanzado mejorará la eficiencia operativa, reducirá errores y optimizará el tiempo de procesamiento de ventas. Esto se traducirá en una mejor experiencia para los clientes y en una administración más efectiva de los recursos del almacén.

Alcance

El proyecto abarca la implementación de un sistema de ventas en grandes almacenes, incluyendo la recepción de cajas, control de productos, generación de recibos, gestión de usuarios y administración de personal.

Objetivos

General

Implementar un sistema de ventas que permita gestionar eficientemente las ventas, control de productos, generación de recibos, y administración de usuarios y personal en grandes almacenes.

Específicos

- Facilitar la recepción y procesamiento de ventas a través de múltiples cajas de atención.
- Optimizar el control y la gestión de productos en el almacén.
- Generar recibos de venta de manera automatizada.
- Gestionar usuarios mayoristas y personal administrativo de manera eficiente.
- Mejorar la experiencia de compra para los clientes.

Referentes teóricos

Diagrama de Clases

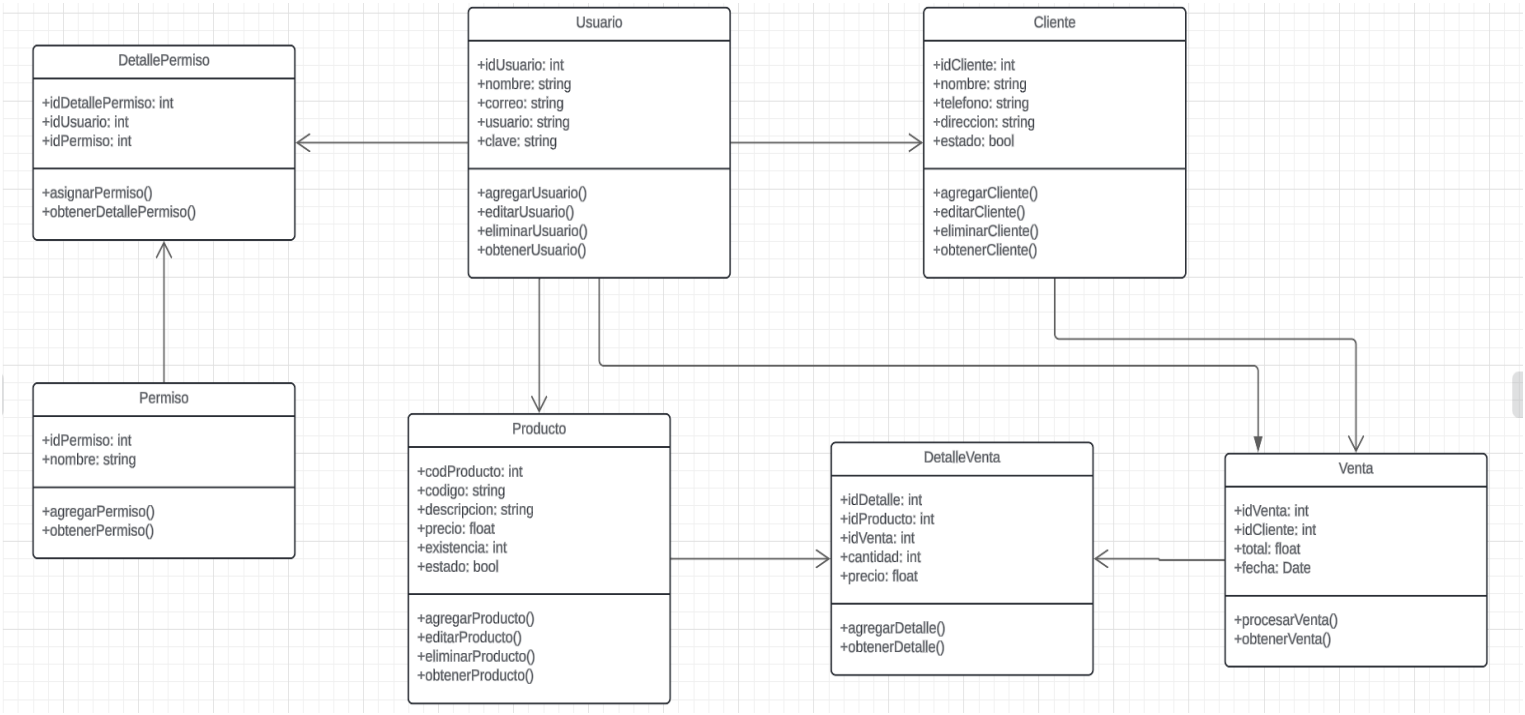


Diagrama de paquetes

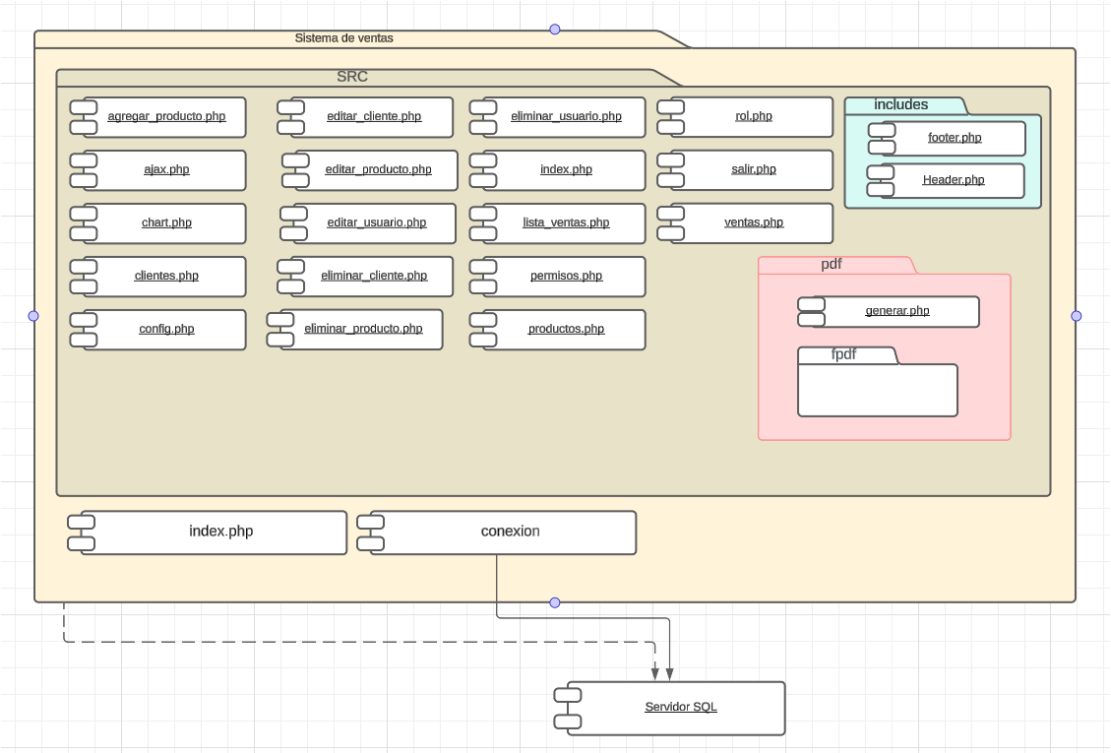


Diagrama de Casos de Uso

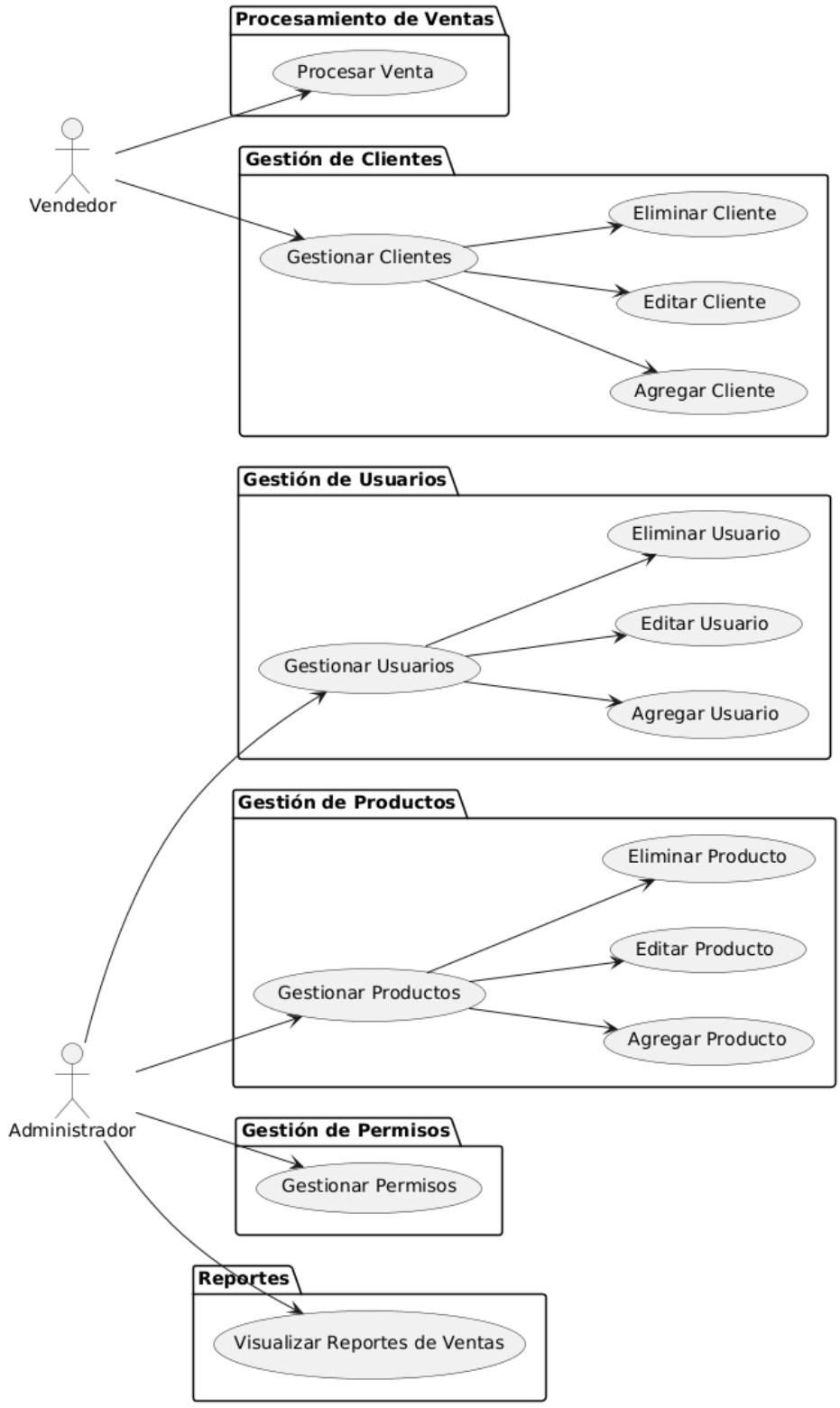


Diagrama de componentes

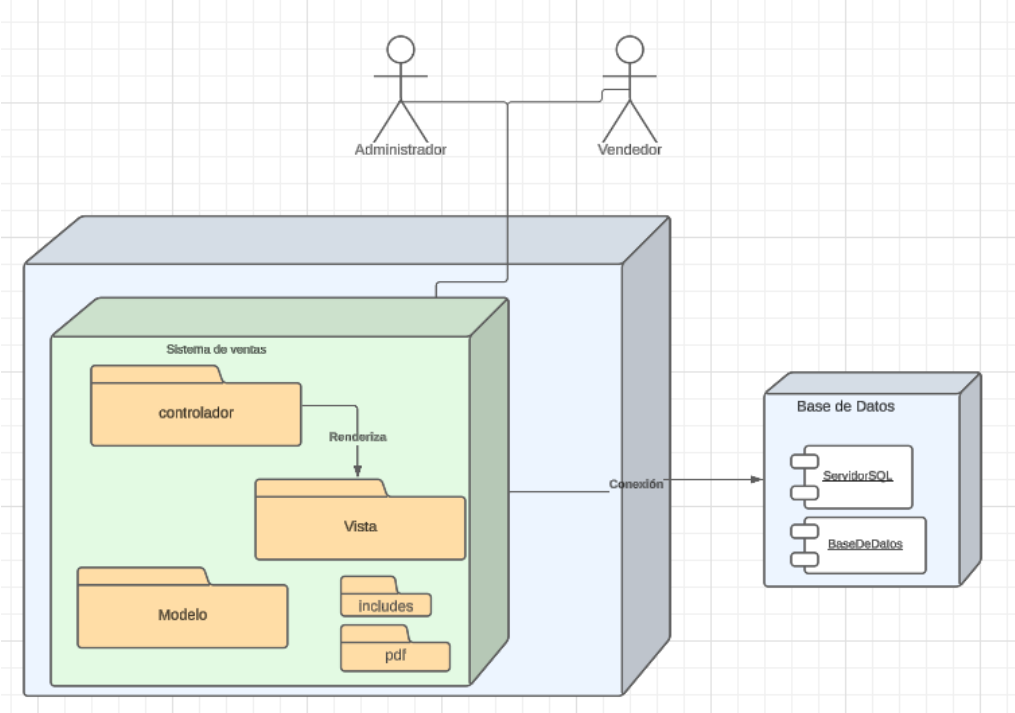


Diagrama de despliegue

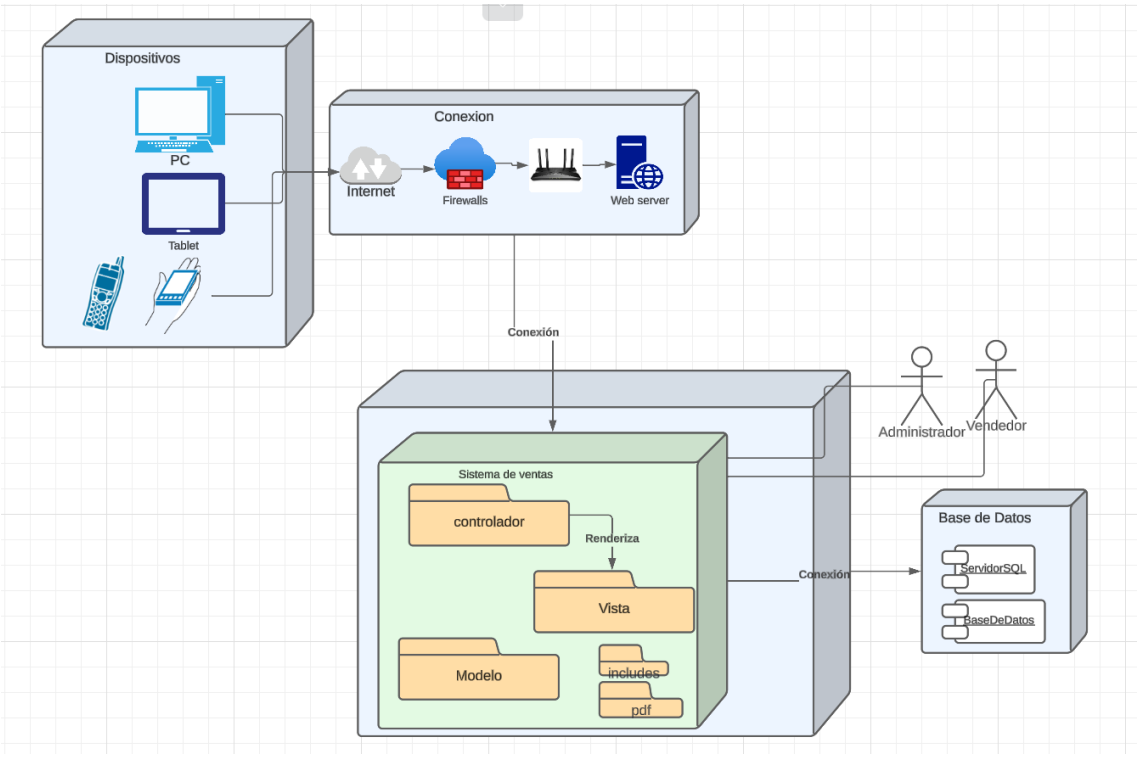
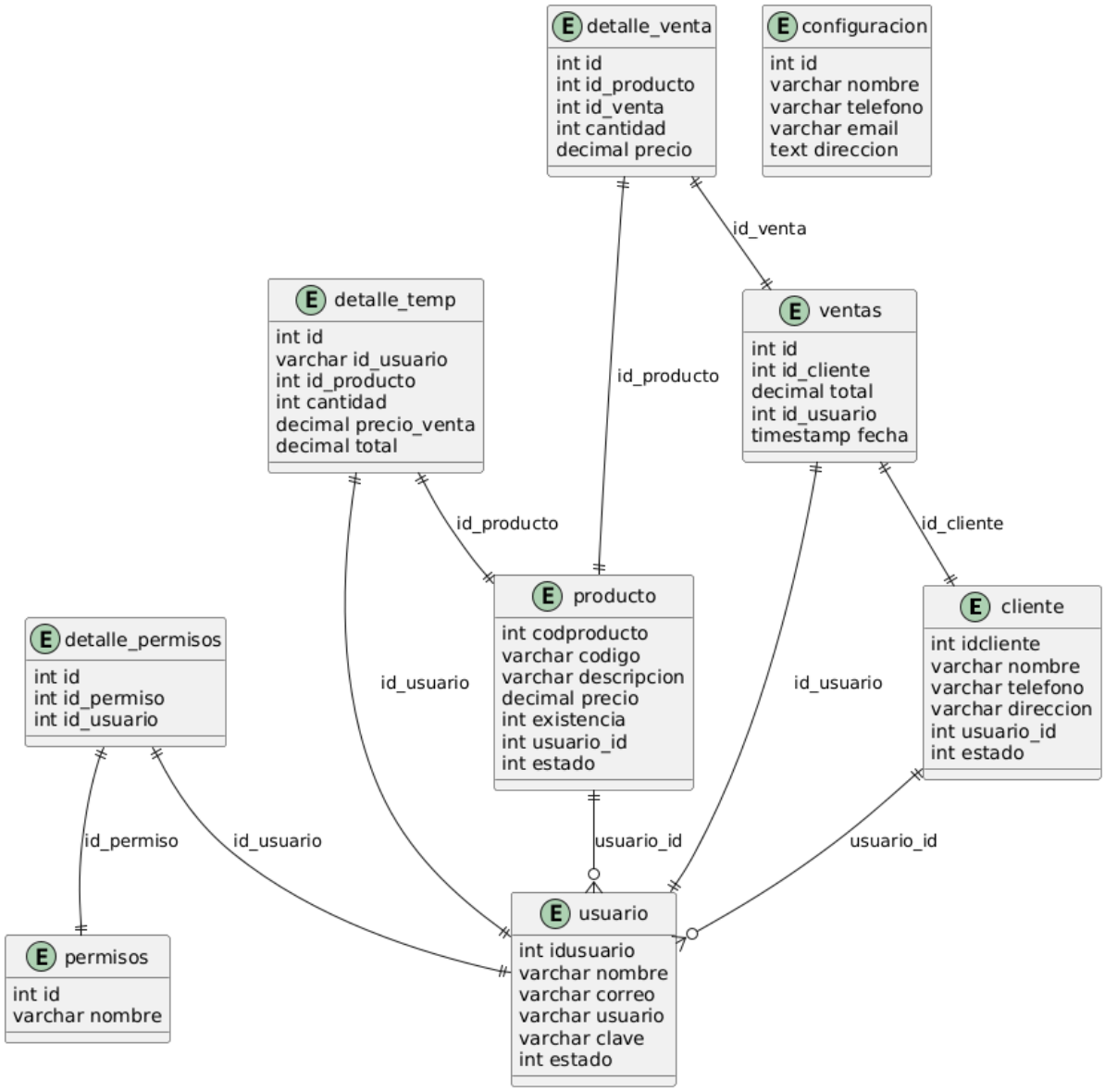


Diagrama de base de datos

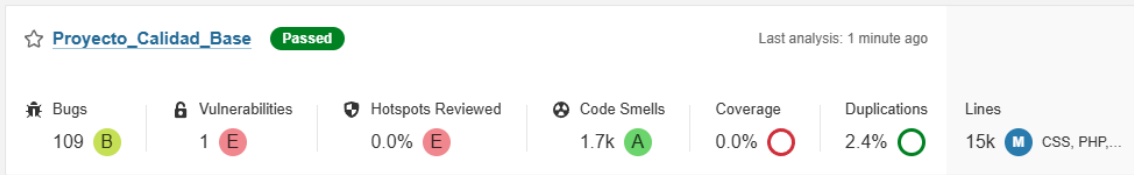


Desarrollo de la propuesta

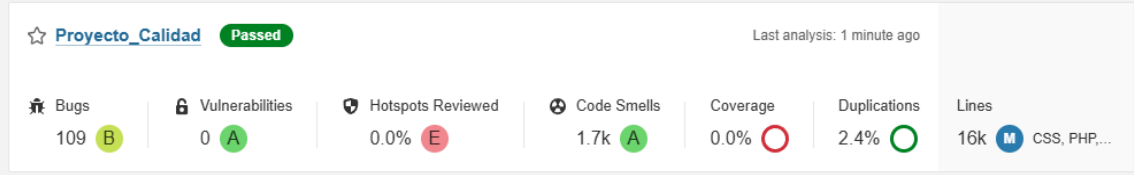
Para el desarrollo de la propuesta, se llevarán a cabo pruebas utilizando herramientas de análisis estático de código, tales como Snyk, SonarQube y SonarCloud. Estas herramientas nos permitirán identificar posibles vulnerabilidades, errores y problemas de calidad en el código, asegurando así un desarrollo más robusto y seguro. Además, facilitarán la implementación de buenas prácticas de programación y la mejora continua del software.

Pruebas realizadas en SonarQube

Antes:

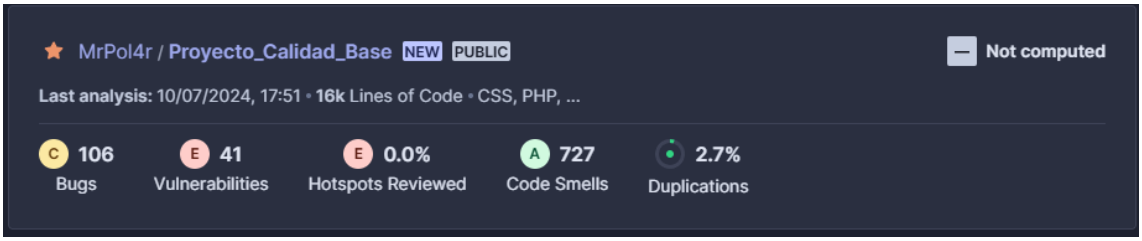


Después:

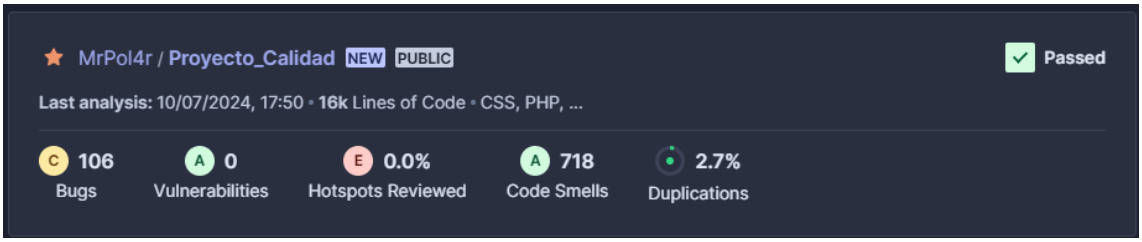


Pruebas realizadas en SonarCloud:

Antes:



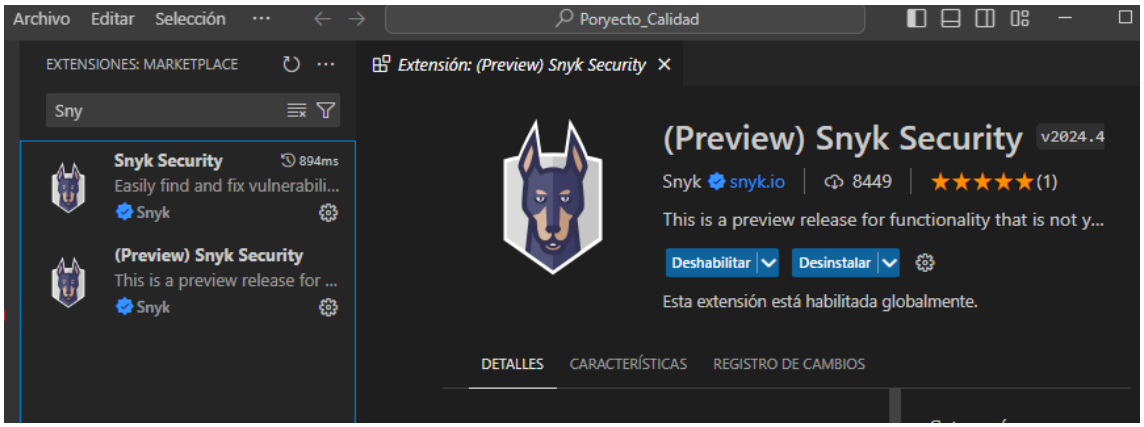
Después:

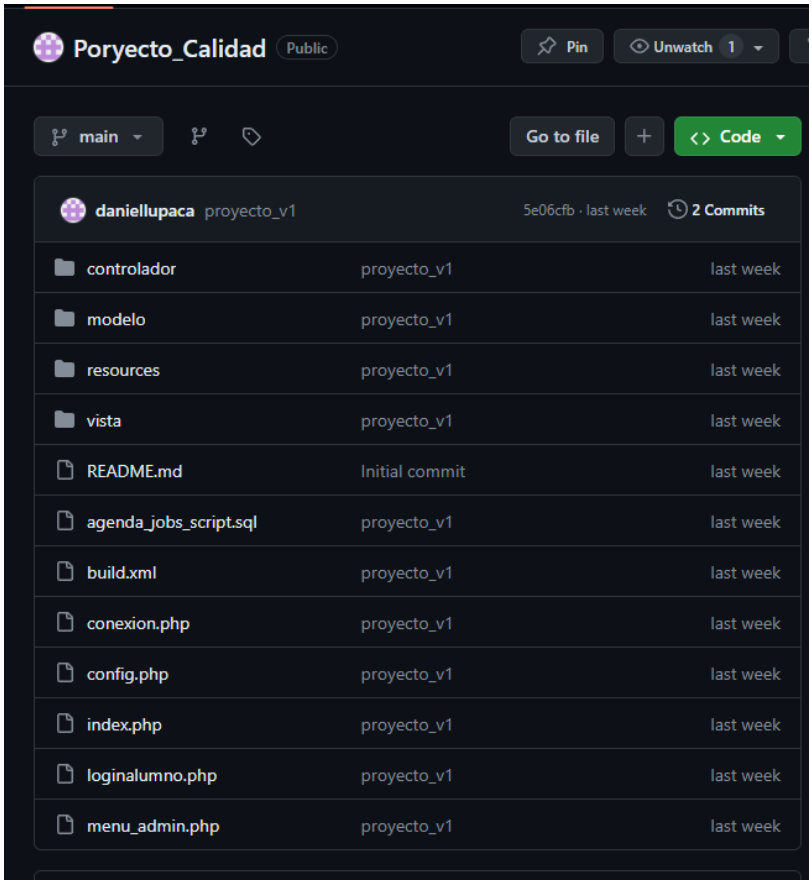


Pruebas realizadas en Snyk:

Antes:

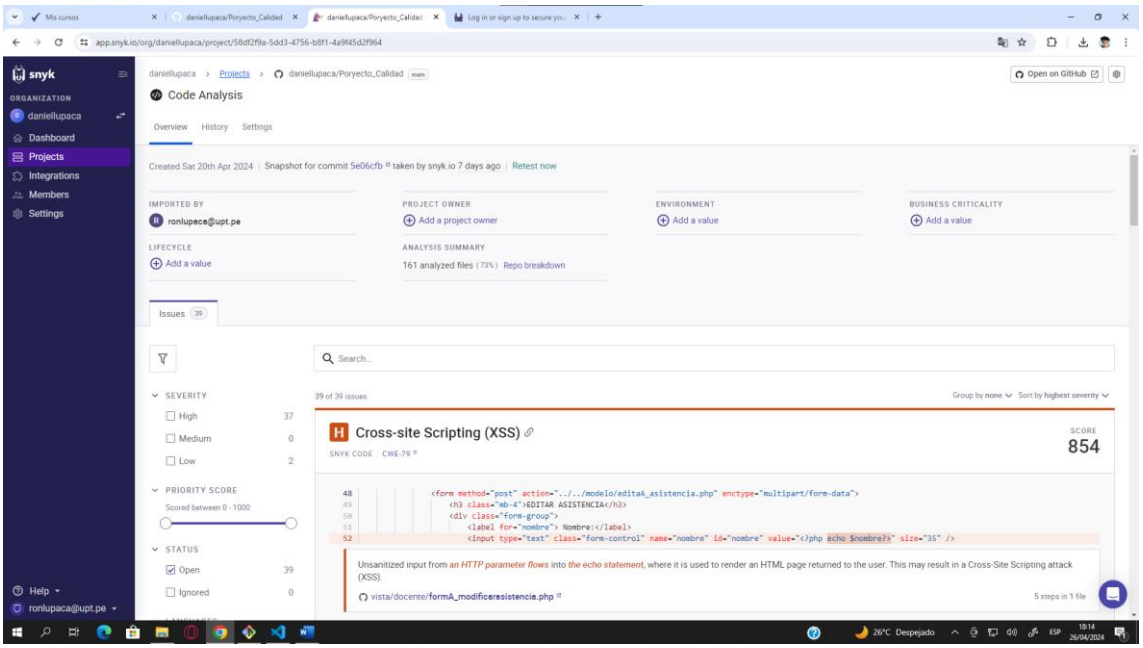
Después:

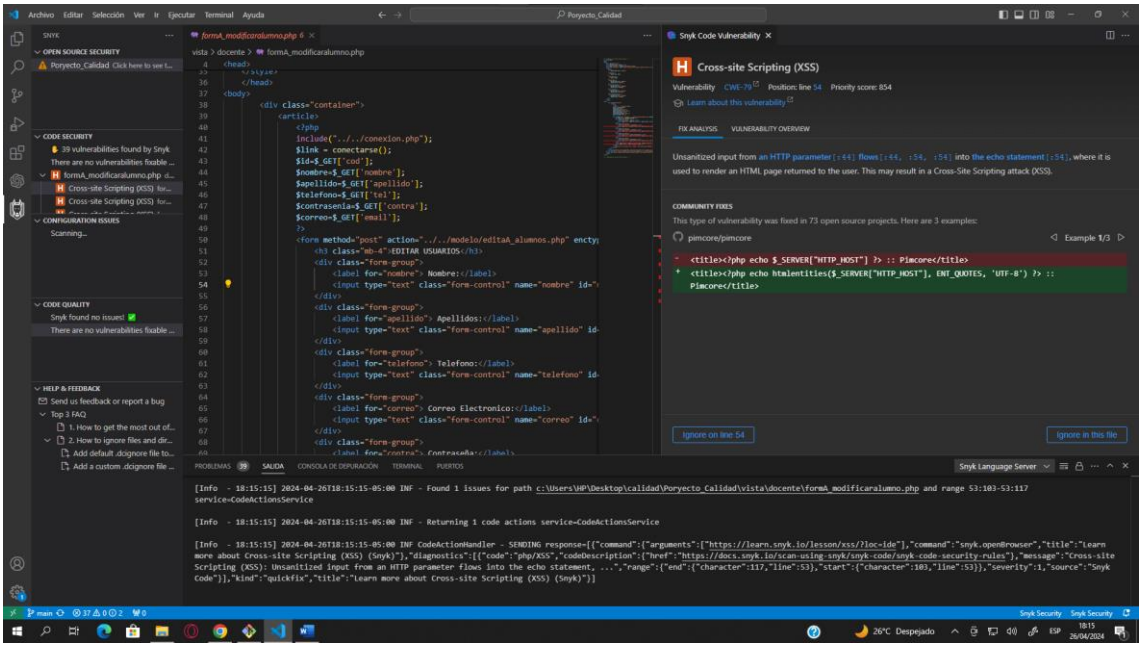

Archivo Editar Selección
Poryecto_Calidad
EXTENSIONES: MARKETPLACE
Sny
Snyk Security 894ms
Easily find and fix vulnerabili...
Snyk
(Preview) Snyk Security
This is a preview release for ...
Snyk
Extensión: (Preview) Snyk Security
(Preview) Snyk Security v2024.4
Snyk snyk.io 8449 (1)
This is a preview release for functionality that is not y...
Deshabilitar Desinstalar
Esta extensión está habilitada globalmente.
DETALLES CARACTERÍSTICAS REGISTRO DE CAMBIOS

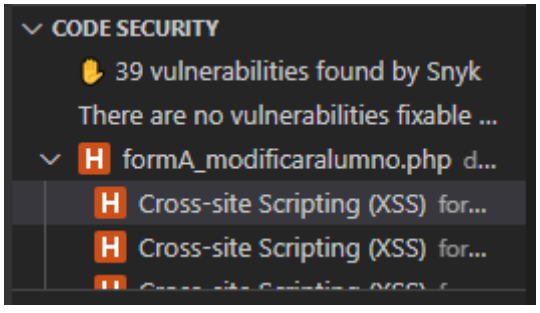

Poryecto_Calidad Public
Pin
Unwatch 1
main
Go to file
Code
daniellupaca proyecto_v1
5e06cfb · last week
2 Commits
controlador proyecto_v1 last week
modelo proyecto_v1 last week
resources proyecto_v1 last week
vista proyecto_v1 last week
README.md Initial commit last week
agenda_jobs_script.sql proyecto_v1 last week
build.xml proyecto_v1 last week
conexion.php proyecto_v1 last week
config.php proyecto_v1 last week
index.php proyecto_v1 last week
loginalumno.php proyecto_v1 last week
menu_admin.php proyecto_v1 last week

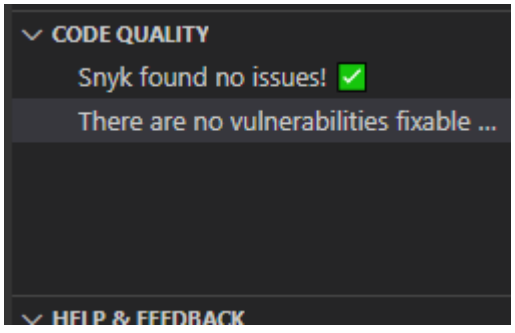
DEVELOPERS
JUNIORS SAC

Logosoft









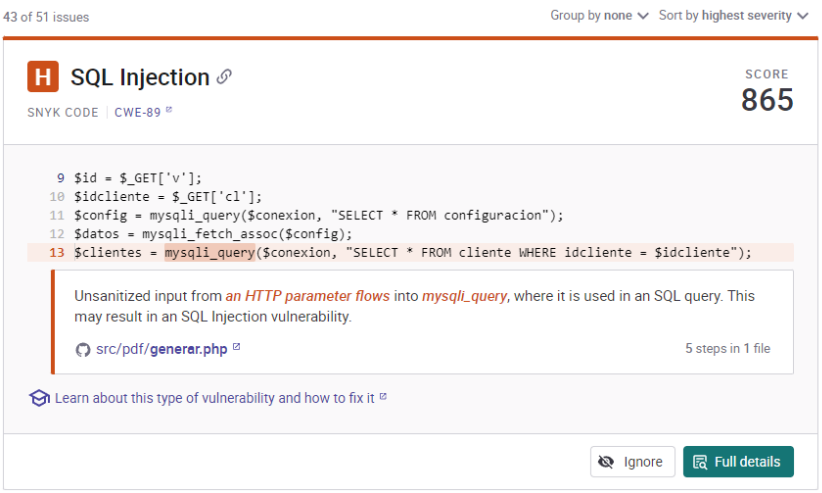
Durante la revisión del código del proyecto `sis_venta`, se identificaron varias vulnerabilidades de inyección SQL y otros problemas de seguridad. Utilizando la herramienta Snyk, se analizaron los archivos del proyecto y se implementaron las correcciones necesarias para mejorar la seguridad y robustez del sistema. A continuación, se presenta un resumen detallado de los errores encontrados y las soluciones aplicadas.

---

1. Inyección SQL

Archivo: `generar.php`

Problema: El código original permitía la inyección SQL debido al uso directo de parámetros de entrada (`$_GET['v']` y `$_GET['cl']`) en las consultas SQL sin ningún tipo de sanitización.



Código Original:

```
php
Copiar código
$id = $_GET['v'];
$idcliente = $_GET['cl'];
$config = mysqli_query($conexion, "SELECT * FROM configuracion");
$datos = mysqli_fetch_assoc($config);
$clientes = mysqli_query($conexion, "SELECT * FROM cliente WHERE
idcliente = $idcliente");
$datosC = mysqli_fetch_assoc($clientes);
$ventas = mysqli_query($conexion, "SELECT d.*, p.codproducto,
p.descripcion FROM detalle_venta d INNER JOIN producto p ON
d.id_producto = p.codproducto WHERE d.id_venta = $id");
```

Solución: Se utilizaron sentencias preparadas (`prepared statements`) para evitar la inyección SQL. Se asignaron los parámetros GET a variables seguras y se realizaron las consultas de manera segura.

Código Corregido:

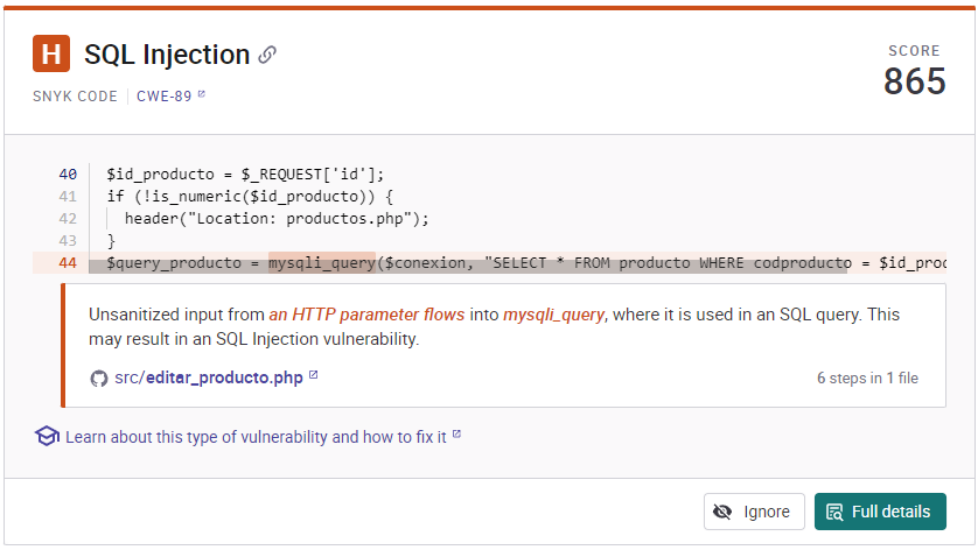
```
php
Copiar código
$idSeguro = $_GET['v'];
$idclienteSeguro = $_GET['cl'];

// Consulta configuración
$configStmt = $conexion->prepare("SELECT * FROM configuracion");
$configStmt->execute();
$config = $configStmt->get_result();
$datos = $config->fetch_assoc();

// Consulta cliente
$clienteStmt = $conexion->prepare("SELECT * FROM cliente WHERE
idcliente = ?");
$clienteStmt->bind_param("i", $idclienteSeguro);
$clienteStmt->execute();
$clientes = $clienteStmt->get_result();
$datosC = $clientes->fetch_assoc();

// Consulta ventas
$ventasStmt = $conexion->prepare("SELECT d.*, p.codproducto,
p.descripcion FROM detalle_venta d INNER JOIN producto p ON
d.id_producto = p.codproducto WHERE d.id_venta = ?");
$ventasStmt->bind_param("i", $idSeguro);
$ventasStmt->execute();
$ventas = $ventasStmt->get_result();
```

---

2. Manejo Seguro de Variables de Entrada

Archivo: **`editar_producto.php`**

Problema: El uso directo de los parámetros de entrada (`$_GET['id']`, `$_POST['codigo']`, `$_POST['producto']`, `$_POST['precio']`) en las consultas SQL sin validación permitía la inyección SQL.



Código Original:

```
php
Copiar código
$codproducto = $_GET['id'];
$codigo = $_POST['codigo'];
$producto = $_POST['producto'];
$precio = $_POST['precio'];
$query_update = mysqli_query($conexion, "UPDATE producto SET codigo =
'$codigo', descripcion = '$producto', precio = $precio WHERE
codproducto = $codproducto");
```

Solución: Se implementaron `prepared statements` para realizar consultas SQL seguras.

Código Corregido:

```
php
Copiar código
$codproductoSeguro = $_GET['id'];
$codigoSeguro = $_POST['codigo'];
$productoSeguro = $_POST['producto'];
$precioSeguro = $_POST['precio'];

$query_update_seguro = $conexion->prepare("UPDATE producto SET codigo
= ?, descripcion = ?, precio = ? WHERE codproducto = ?");
$query_update_seguro->bind_param("ssdi", $codigoSeguro,
$productoSeguro, $precioSeguro, $codproductoSeguro);
$query_update_seguro->execute();
```

---

3. Validación y Saneamiento de Entradas de Usuario

Archivo: `agregar_producto.php`

Problema: El código original permitía la inserción de datos no validados directamente en la base de datos, lo que exponía el sistema a ataques de inyección SQL.

Código Original:

```
php
Copiar código
$codigo = $_POST['codigo'];
$producto = $_POST['producto'];
$precio = $_POST['precio'];
$query = mysqli_query($conexion, "SELECT * FROM producto WHERE codigo
= '$codigo'");
$result = mysqli_fetch_array($query);
$query_insert = mysqli_query($conexion, "INSERT INTO producto(codigo,
descripcion, precio) values ('$codigo', '$producto', '$precio')");
```

Solución: Se realizaron consultas seguras utilizando `prepared statements`.

Código Corregido:

```
php
Copiar código
$codigoSeguro = $_POST['codigo'];
$productoSeguro = $_POST['producto'];
$precioSeguro = $_POST['precio'];
$cantidadSeguro = $_POST['cantidad'];
$usuario_idSeguro = $_SESSION['idUser'];

$query_insert_seguro = $conexion->prepare("INSERT INTO
producto(codigo, descripcion, precio, existencia, usuario_id) VALUES
(?, ?, ?, ?, ?)");
$query_insert_seguro->bind_param("ssdii", $codigoSeguro,
$productoSeguro, $precioSeguro, $cantidadSeguro, $usuario_idSeguro);
$query_insert_seguro->execute();
```

---

4. Protección Contra Inyecciones SQL en Consultas de Selección

Archivo: **`clientes.php`**

Problema: El uso directo de parámetros GET y POST en las consultas SQL sin sanitización exponía el sistema a inyecciones SQL.

Código Original:

```
php
Copiar código
$id_cliente = $_GET['id'];
$cliente = mysqli_query($conexion, "SELECT * FROM cliente WHERE
idcliente = $id_cliente");
```

Solución: Se utilizaron `prepared statements` para realizar consultas seguras.

Código Corregido:

```
php
Copiar código
$id_clienteSeguro = $_GET['id'];
$clienteSeguro = $conexion->prepare("SELECT * FROM cliente WHERE
idcliente = ?");
$clienteSeguro->bind_param("i", $id_clienteSeguro);
$clienteSeguro->execute();
$cliente = $clienteSeguro->get_result();
```

Conclusión

Se identificaron y corrigieron múltiples vulnerabilidades de inyección SQL en el proyecto `sis_venta`. La utilización de `prepared statements` y la sanitización de entradas de usuario fueron las principales medidas implementadas para asegurar el sistema. Estas correcciones no solo mejoran la seguridad del sistema, sino que también aumentan su robustez y confiabilidad.

Cómo vistos al inicio se detectó 51 vulnerabilidades , 43 de grado Alto y dos de medio y 6 de nivel bajo

se trabajó en 4 tipos de vulnerabilidades :

1. **Uso de Credenciales Codificadas (Hardcoded Credentials)**:
    - **Descripción**: Se encontraron credenciales (como nombres de usuario y contraseñas) directamente en el código fuente.
    - **Impacto**: Esto puede permitir a atacantes acceder a sistemas críticos sin necesidad de realizar un ataque sofisticado.
    - **Solución**: Remover las credenciales codificadas y utilizar variables de entorno o sistemas de gestión de secretos.

2. **Cross-site Scripting (XSS)**:
    - **Descripción**: Se detectaron lugares en el código donde entradas de usuarios no son correctamente validadas y sanitizadas.
    - **Impacto**: Un atacante puede inyectar scripts maliciosos que se ejecuten en el navegador de otros usuarios.
    - **Solución**: Implementar validación y sanitización adecuada de todas las entradas de usuario.

3. **Inyección SQL (SQL Injection)**:
    - **Descripción**: Se encontraron posibles puntos de inyección de SQL debido a la construcción insegura de consultas SQL.
    - **Impacto**: Un atacante podría manipular las consultas SQL para acceder, modificar o eliminar datos en la base de datos.
    - **Solución**: Utilizar consultas preparadas y evitar la concatenación directa de entradas de usuario en las consultas SQL.

4. **Problemas de Seguridad Relacionados con la Configuración del Servidor**:
    - **Descripción**: Configuraciones incorrectas o débiles en el servidor, como la exposición del HTTP_HOST.
    - **Impacto**: Esto puede permitir a los atacantes explotar configuraciones inseguras para comprometer el servidor.
    - **Solución**: Revisar y asegurar la configuración del servidor, siguiendo las mejores prácticas de seguridad.

Proceso de Corrección

1. **Ambiente de Desarrollo**:
    - El código fue corregido utilizando Visual Studio Code.
    - Se realizaron múltiples commits en el repositorio público para asegurar que cada vulnerabilidad identificada fuese abordada y corregida.
2. **Re-escaneo del Proyecto**:
    - Después de implementar las correcciones, el proyecto fue vuelto a escanear utilizando Snyk.
    - El nuevo escaneo mostró la eliminación de 38 vulnerabilidades, reduciendo significativamente el riesgo de seguridad.

Estas correcciones mejoran significativamente la seguridad de tu aplicación, protegiendo tanto los datos de los usuarios como la integridad de tu sistema.

Tecnología de información

Snyk: Utilizado para el escaneo de vulnerabilidades en las dependencias y para realizar análisis estático del código.

SonarQube: Implementado para la revisión continua del código, enfocado en la detección de bugs, vulnerabilidades y deuda técnica.

SonarCloud: Servicio en la nube que proporciona análisis automatizado para detectar y corregir fallas en el código.

Git: Sistema de control de versiones empleado para el seguimiento de cambios y colaboración en el desarrollo del proyecto.

GitHub: Plataforma de alojamiento para el repositorio del proyecto, integrada con Snyk para el análisis de seguridad tras cada push.

MySQL: Sistema de gestión de base de datos para almacenar, modificar y extraer la información utilizada en la aplicación.

PHP: Lenguaje de programación del lado del servidor para la implementación de la lógica de negocio de la aplicación.

Metodología, técnicas usadas

Desarrollo Ágil: Implementación iterativa e incremental para facilitar la flexibilidad y la entrega continua de valor.

Integración Continua/Despliegue Continuo (CI/CD): Prácticas que permiten la integración de cambios de forma automática y sistemática.

Revisión de Código: Práctica colaborativa en la que los cambios de código son revisados por pares antes de ser integrados al proyecto.

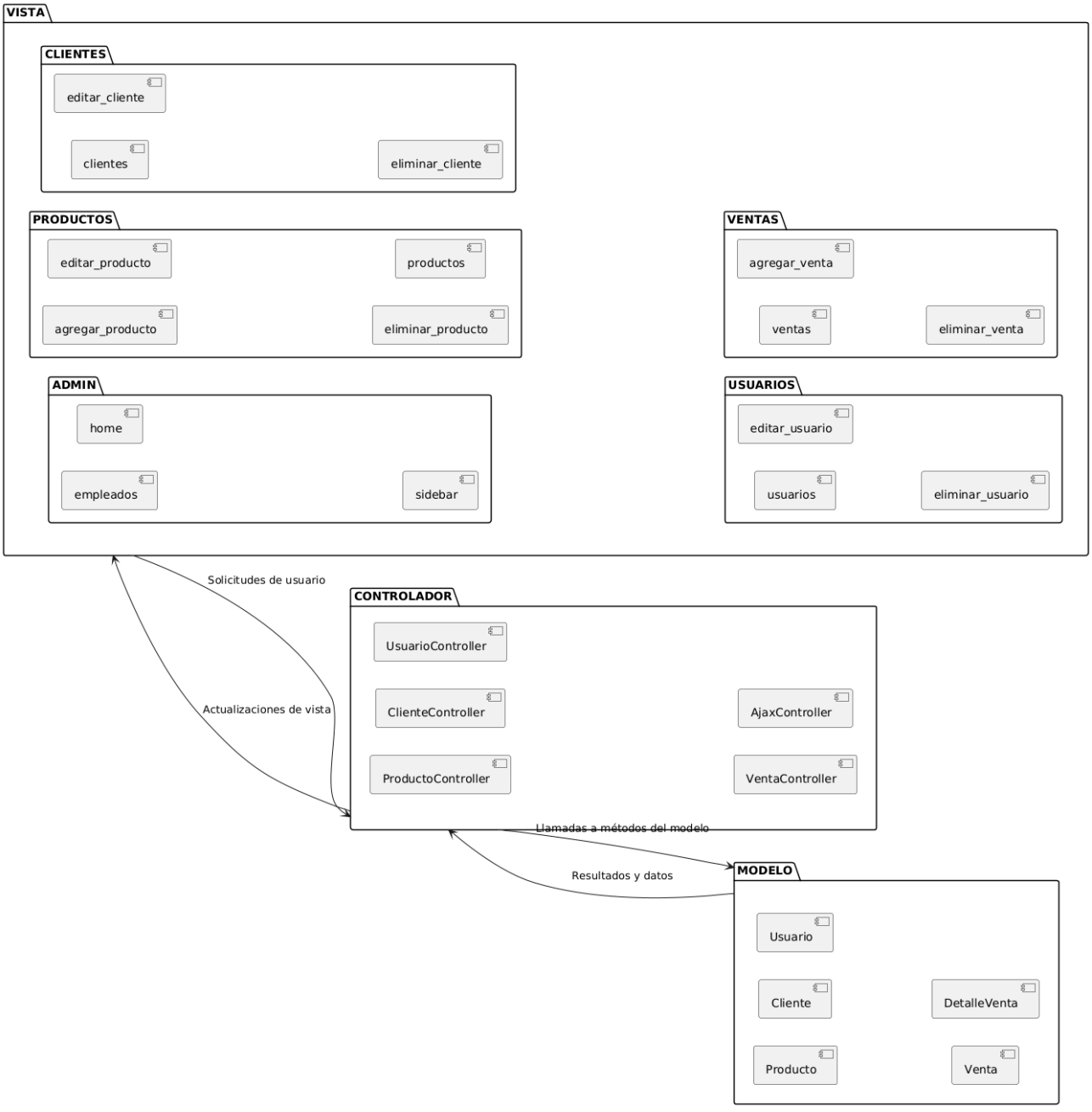
Análisis Estático de Código: Empleo de herramientas como Snyk y SonarQube para detectar problemas en el código sin ejecutarlo.

Seguridad de la Información: Aplicación de prácticas de codificación segura, tales como la sanitización de entradas y el uso de sentencias preparadas.

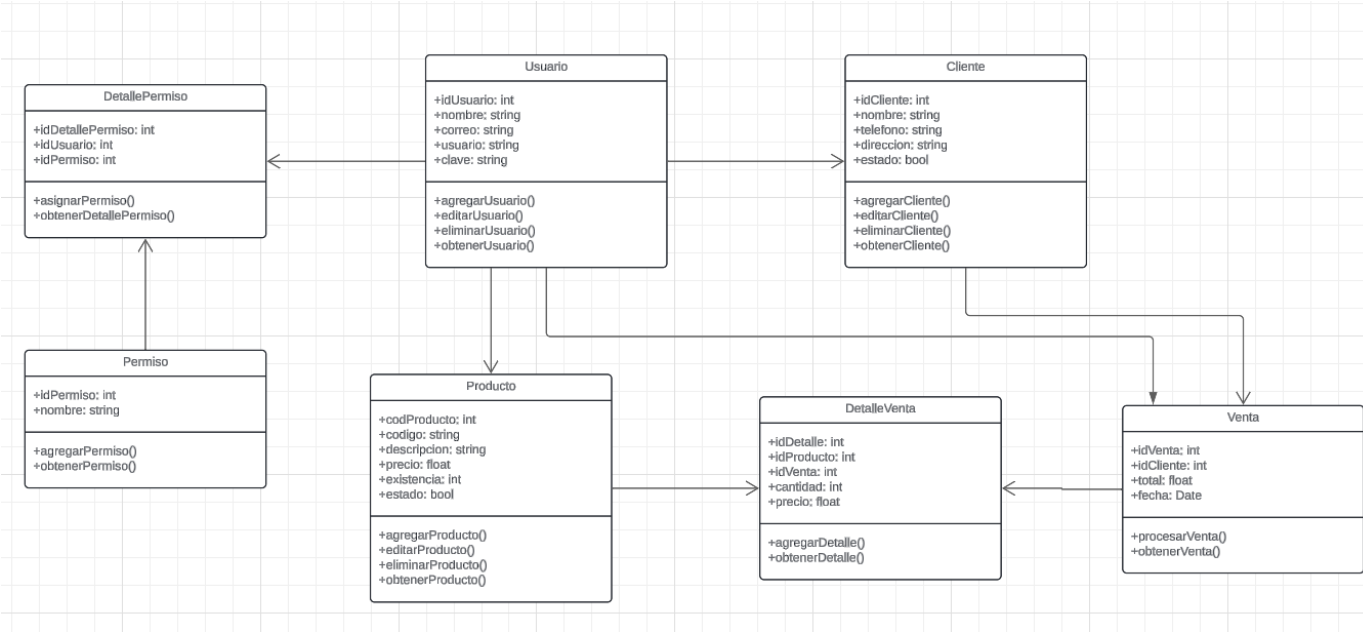
Cronograma (personas, tiempo, otros recursos) Basado en las observaciones que la herramienta SonarQube les informará sobre la aplicación, a fin de reducir la deuda técnica, vulnerabilidades, fallas, etc. a 0.

Actividades	Mes 1	Mes 2	Mes 3	Mes 4	Mes 5
Vulnerabilidades Altas (Ronal Lupaca)	X	X			
Vulnerabilidades Medias (Ronal Lupaca)	X	X			
Vulnerabilidades Bajas (Ronal Lupaca y Carlos Escobar)	X	X	X		
Redundancia de Código (Paco Ramos, Apaza Mamani)			X		
Corrección de Bugs (Carlos Escobar, Ronal Lupaca)				X	
Pruebas Finales (Ronal Lupaca , Carlos Escobar, Paco Ramos, Apaza Mamani)					X

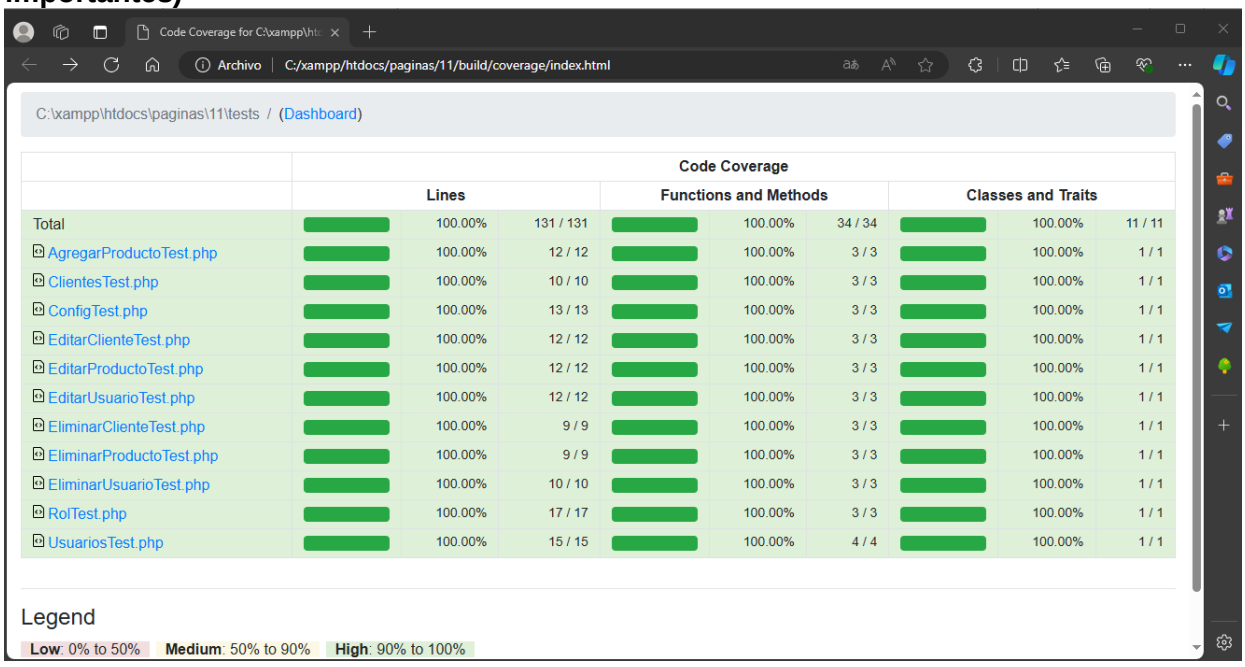
9. Desarrollo de Solución de Mejora
9.1 Diagrama de Arquitectura de la aplicación



9.2 Diagrama de Clases de la aplicación



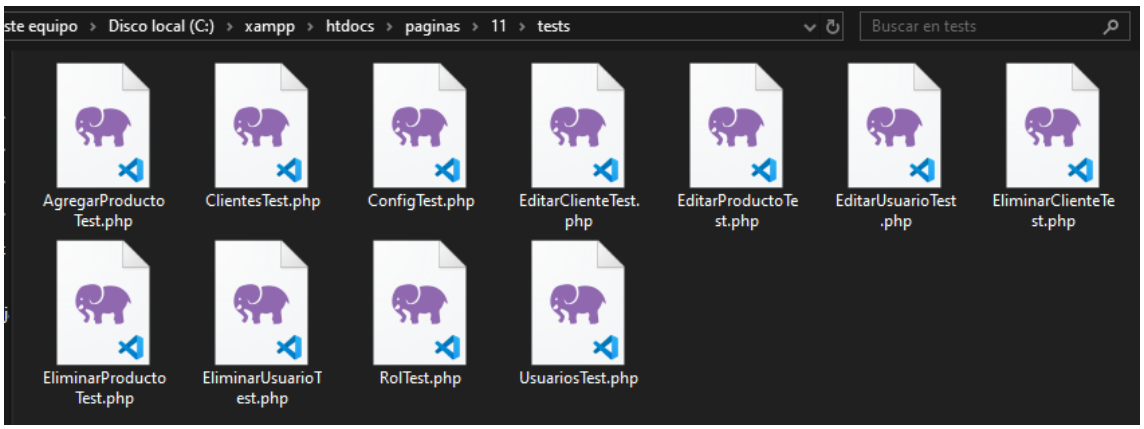
9.3 Métodos de pruebas implementados para coberturar la aplicación
a) Pruebas Unitarias (cobertura de al menos 70% de codigo - los métodos más importantes)



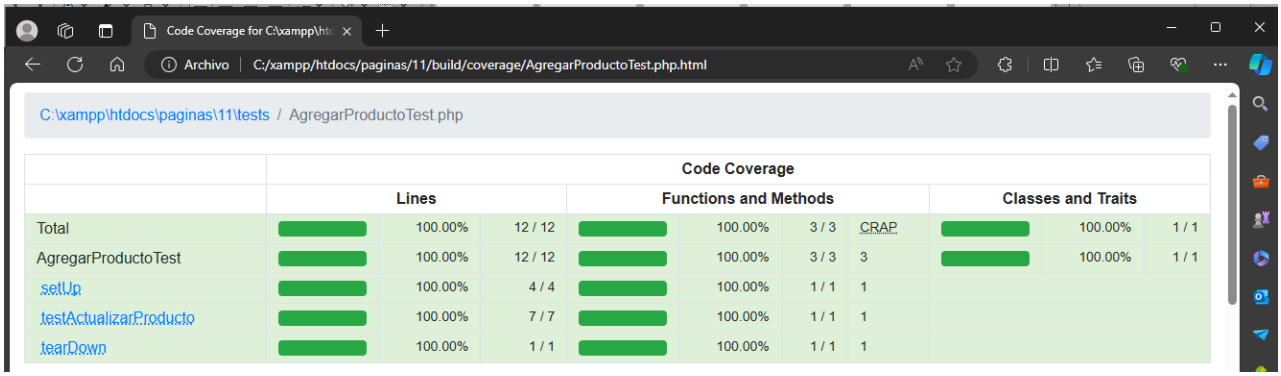
Se muestra nuestro reporte de cobertura de nuestra aplicación. En el reporte, cada archivo, como `AgregarProductoTest.php`, `ClientesTest.php`, `ConfigTest.php`, y otros, ha logrado una cobertura del 100% en tres categorías: líneas de código, funciones y métodos, y clases. Esto indica que las pruebas unitarias han ejecutado cada línea, función y clase en estos

archivos, asegurando que todos los aspectos del código funcionan correctamente bajo las condiciones probadas.

TEST CREADOS:



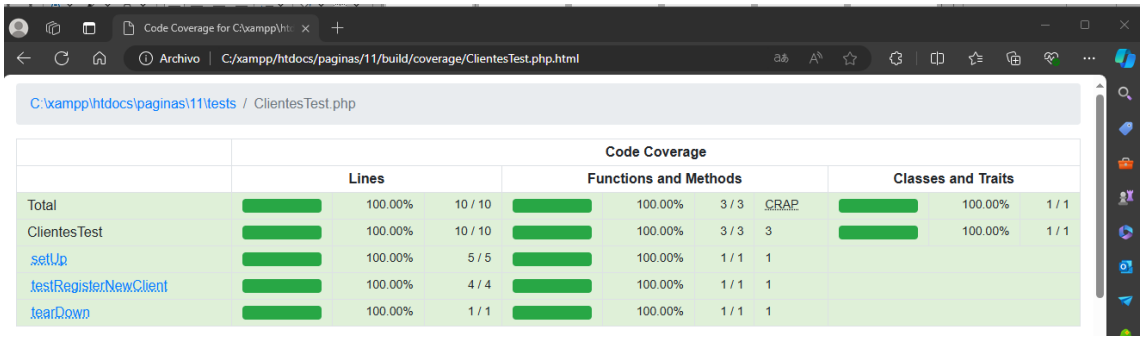
AgregarProductoTest.php



El objetivo principal del test `testActualizarProducto` en la clase `AgregarProductoTest` es verificar que la función `actualizarProducto` de la clase `ProductoModelTest` actualice correctamente un registro en la base de datos. Específicamente, este test busca asegurar que:

1. **Correcta Actualización de Datos**: Los campos `codigo`, `descripcion`, y `precio` del producto especificado deben ser actualizados correctamente en la base de datos. El test verifica que los valores en la base de datos coincidan con los nuevos valores proporcionados a la función `actualizarProducto`.
2. **Verificación de Resultados**: Además de actualizar la base de datos, el método puede retornar algún mensaje o valor que indique el éxito de la operación. El test verifica que la respuesta incluya una cadena específica (`'Producto Modificado'`), lo cual podría ser un mensaje de confirmación de que la actualización fue exitosa.

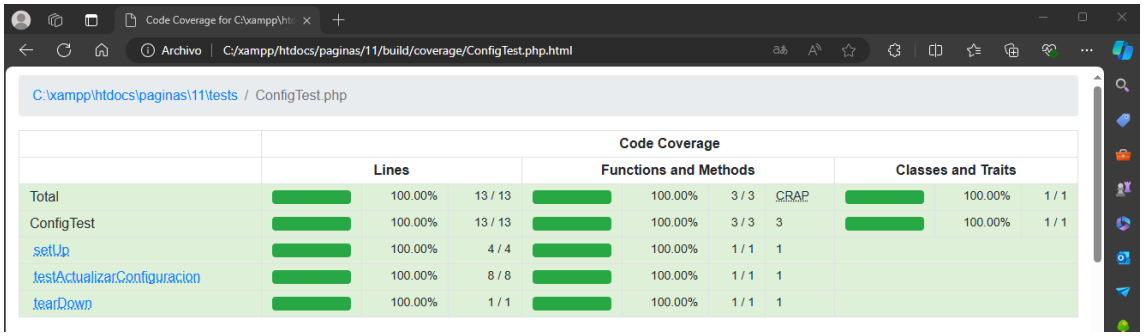
ClientesTest.php



El test `testRegisterNewClient` en la clase `ClientesTest` está diseñado para validar la funcionalidad de registrar un nuevo cliente en la base de datos mediante la clase `ClienteModelTest`. Su objetivo principal es asegurarse de que el método `registrarCliente` funciona correctamente al añadir un nuevo registro en la tabla `cliente` y verifica dos aspectos clave:

1. **Inserción Correcta en la Base de Datos**: Tras llamar al método `registrarCliente`, el test verifica que realmente se ha insertado un nuevo registro en la tabla `cliente`. Esto se realiza mediante una consulta SQL que busca el cliente por nombre, asegurándose de que el número de filas retornadas sea exactamente uno, lo que indica que el cliente fue agregado correctamente.
2. **Respuesta Esperada**: Verifica que el método `registrarCliente` retorne la cadena 'Cliente registrado', que se espera sea el mensaje de éxito tras registrar un nuevo cliente. Esto puede ser parte de la lógica de la aplicación para proporcionar feedback directo al usuario o a otras partes del sistema sobre el resultado de la operación.

ConfigTest.php



El test `testActualizarConfiguracion` en la clase `ConfigTest` tiene como objetivo verificar la funcionalidad del método `actualizarConfiguracion` dentro de la clase `ConfiguracionModelTest`, que se encarga de modificar los datos de configuración de una empresa en la base de datos. Este test se centra en asegurar que los cambios se

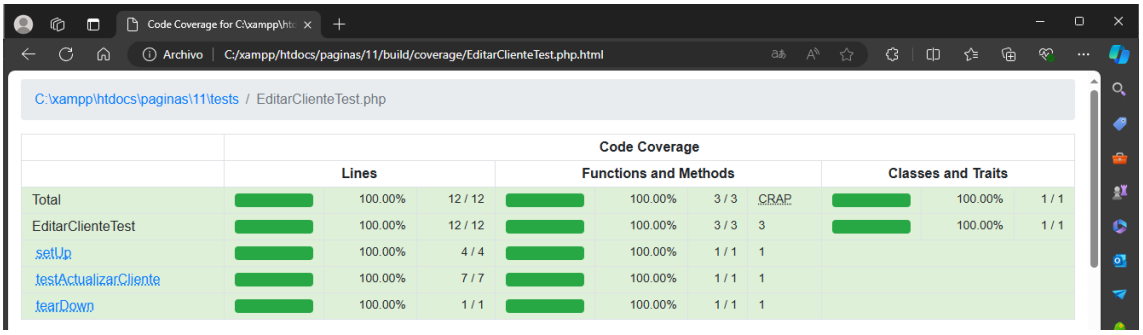
apliquen **correctamente y que el método devuelva un mensaje de confirmación adecuado.**

1. **Correcta Actualización en la Base de Datos**: Realiza consultas para asegurarse de que los datos en la base de datos correspondan a los nuevos valores suministrados, validando que cada campo (nombre, teléfono, email, dirección) se haya actualizado correctamente.
2. **Mensaje de Confirmación:** Comprueba que el resultado devuelto por el método contenga un mensaje específico (en este caso, algo como 'Datos modificados'), indicando el éxito de la operación.

Propósito del Test:

- **Integridad de Datos:** Asegurar que los datos críticos de la configuración de la empresa se puedan actualizar correctamente y reflejen los cambios esperados.
- **Confianza en el Método:** Probar que el método `actualizarConfiguracion` es fiable y realiza sus funciones como se espera, incluyendo la devolución de un mensaje apropiado que puede ser utilizado para feedback al usuario.
- **Automatización y Regresión:** Facilitar la ejecución automática de pruebas para detectar rápidamente cualquier regresión o error introducido en futuras modificaciones del código.

EditarClienteTest.php



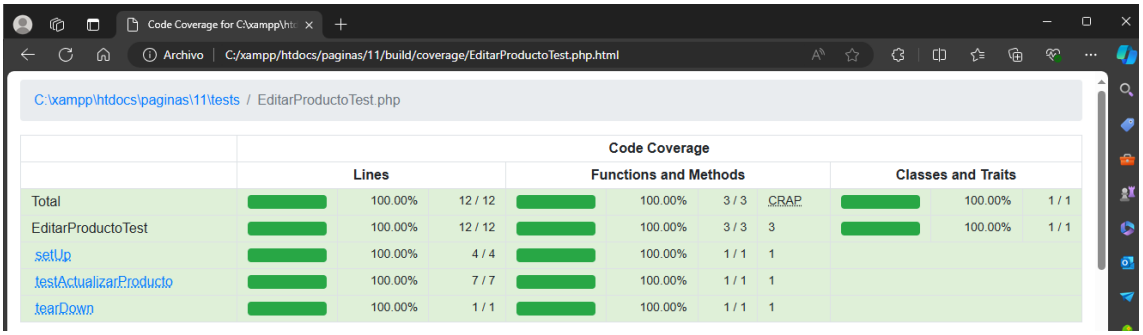
El test `testActualizarCliente` dentro de la clase `EditarClienteTest` está diseñado para comprobar la funcionalidad de actualizar los detalles de un cliente existente en la base de datos a través del método `actualizarCliente` en la clase `ClienteModelTest`. Este test se enfoca en asegurar que los datos del cliente se actualicen correctamente y que el método retorne un mensaje indicativo del éxito de la operación.

1. Comprobación de Datos en Base de Datos: Realiza una consulta para verificar que los datos del cliente en la base de datos reflejen los nuevos valores proporcionados.
2. Evaluación del Resultado del Método: Verifica que el string retornado por el método contenga un mensaje específico (como 'Cliente Actualizado correctamente'), lo que indica que la actualización fue exitosa y que el método está funcionando como se espera.

Objetivos del Test:

- **Integridad de Datos:** Confirma que los datos del cliente se pueden actualizar correctamente en la base de datos, lo que es crucial para mantener la precisión y relevancia de la información del cliente.
- **Confirmación del Proceso:** Asegura que el método proporciona feedback adecuado sobre el éxito de la operación, lo cual es esencial para la interacción del usuario final o para la lógica de control en aplicaciones más grandes.
- **Automatización de Regresiones:** Facilita la identificación rápida de regresiones o errores introducidos en el código relacionado con la actualización de clientes, permitiendo correcciones antes de que el código se despliegue en un entorno de producción.

EditarProductoTest.php



El test `testActualizarProducto` en la clase `EditarProductoTest` está diseñado para asegurar que la funcionalidad de actualizar los detalles de un producto específico en la base de datos funcione correctamente. Este se realiza a través de la clase `ProductoModelTest`, enfocándose en garantizar que los cambios se aplican correctamente y que el método devuelva una confirmación del éxito de la operación.

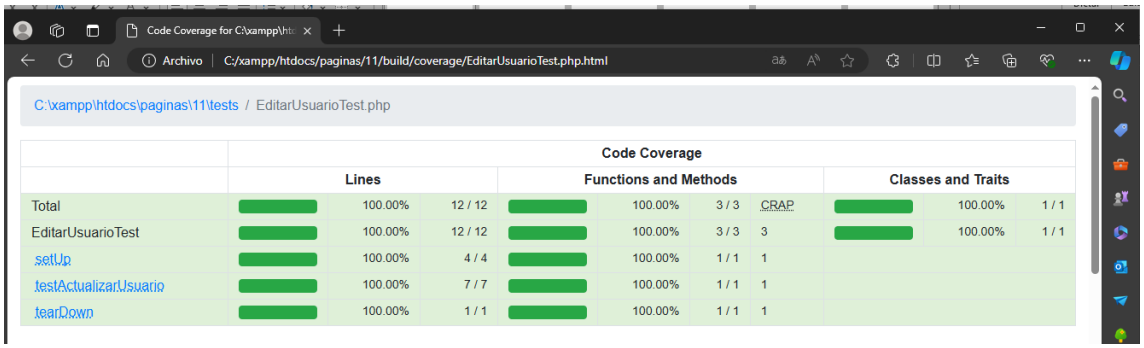
1. Comprobación de Datos en Base de Datos: Realiza una consulta para verificar que los datos del producto en la base de datos reflejen los nuevos valores proporcionados. Esto incluye comparar el código, descripción y precio.
2. Evaluación del Resultado del Método: Verifica que el string retornado por el método contenga un mensaje específico (como 'Producto Modificado'), lo que indica que la actualización fue exitosa y que el método está funcionando correctamente.

Objetivos del Test:

- **Integridad de Datos:** Confirma que los datos del producto se pueden actualizar correctamente en la base de datos, lo que es crucial para mantener la precisión y relevancia de la información del producto.

- **Confirmación del Proceso:** Asegura que el método proporciona feedback adecuado sobre el éxito de la operación, lo cual es esencial para la interacción del usuario final o para la lógica de control en aplicaciones más grandes.
- **Automatización de Regresiones:** Facilita la identificación rápida de regresiones o errores introducidos en el código relacionado con la actualización de productos, permitiendo correcciones antes de que el código se despliegue en un entorno de producción.

EditarUsuarioTest.php



El test `testActualizarUsuario` en la clase `EditarUsuarioTest` está diseñado para verificar la funcionalidad de actualizar la información de un usuario existente en la base de datos a través del método `actualizarUsuario` de la clase `UsuarioModelTest`. Este test se enfoca en asegurar que los cambios se realicen correctamente y que el método devuelva una confirmación apropiada del éxito de la operación.

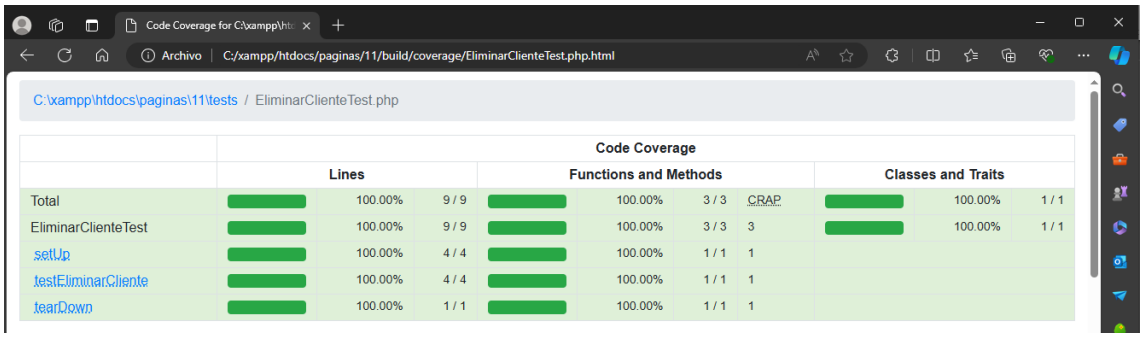
1. Comprobación de Datos en la Base de Datos: Realiza una consulta para verificar que los datos del usuario en la base de datos reflejen los nuevos valores proporcionados, incluyendo nombre, correo y nombre de usuario.
2. Evaluación del Resultado del Método: Verifica que el string retornado por el método contenga un mensaje específico (como 'Usuario Actualizado'), lo que indica que la actualización fue exitosa y que el método está funcionando correctamente.

Objetivos del Test:

- **Integridad de Datos:** Confirma que los datos del usuario se pueden actualizar correctamente en la base de datos, lo que es crucial para mantener la precisión y relevancia de la información del usuario.
- **Confirmación del Proceso:** Asegura que el método proporciona feedback adecuado sobre el éxito de la operación, lo cual es esencial para la interacción del usuario final o para la lógica de control en aplicaciones más grandes.
- **Automatización de Regresiones:** Facilita la identificación rápida de regresiones o errores introducidos en el código relacionado con la actualización de usuarios,

permitiendo correcciones antes de que el código se despliegue en un entorno de producción.

EliminarClienteTest.php



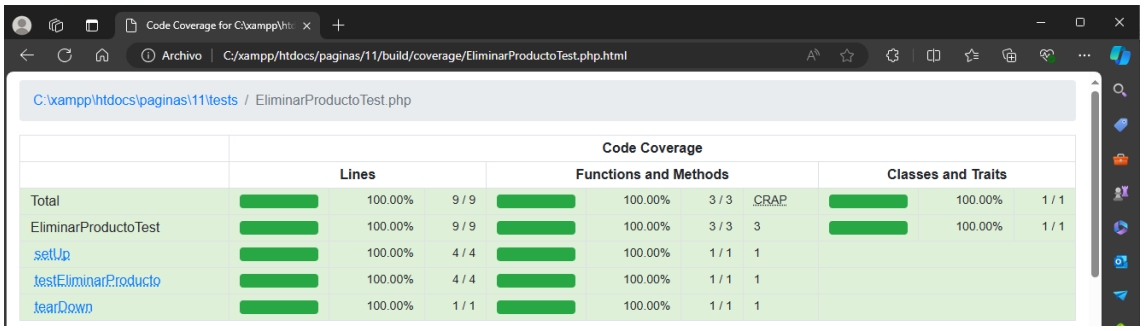
El test `testEliminarCliente` en la clase `EliminarClienteTest` tiene como objetivo verificar la correcta funcionalidad del método `eliminarCliente` de la clase `ClienteModelTest`, el cual está diseñado para cambiar el estado de un cliente en la base de datos, marcándolo como eliminado, en lugar de borrar completamente el registro. Esto es una práctica común en la gestión de datos para mantener la integridad del historial de datos.

1. Eliminación del Cliente: Invoca el método eliminarCliente para cambiar el estado del cliente especificado (en este caso, el cliente con idcliente = 1).
2. Verificación de la Eliminación:
    - Comprobación de Estado en la Base de Datos: Realiza una consulta para verificar que el estado del cliente en la base de datos refleje que ha sido marcado como eliminado (estado = 0), lo que indica que el cliente ya no está activo.

Objetivos del Test:

- **Correcta Funcionalidad del Método:** Confirmar que el método `eliminarCliente` realiza correctamente la transición del estado del cliente a 'inactivo' o 'eliminado', lo cual es vital para aplicaciones que requieren un control de estados para sus registros.
- **Integridad de Datos:** Asegurar que el método maneja adecuadamente los cambios de estado sin eliminar completamente los datos, permitiendo una recuperación fácil del estado anterior si es necesario.
- **Automatización de Pruebas:** Facilitar la detección temprana de errores en la implementación de la funcionalidad de eliminación de clientes, ayudando a mantener la calidad y la fiabilidad del software.

EliminarProductoTest.php



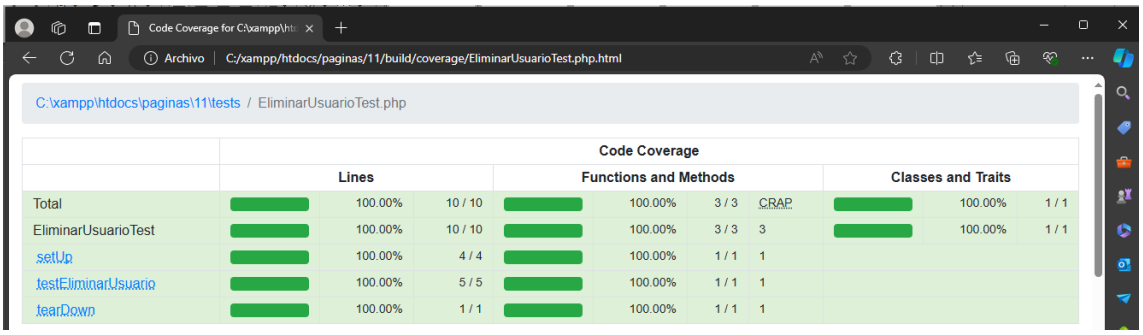
El test `testEliminarProducto` dentro de la clase `EliminarProductoTest` está diseñado para asegurar que el método `eliminarProducto` de la clase `ProductoModelTest` funcione correctamente al cambiar el estado de un producto en la base de datos, marcándolo como eliminado sin eliminar físicamente el registro. Este enfoque ayuda a mantener la integridad de los datos y permite la restauración del estado anterior del producto si es necesario.

1. Eliminación del Producto: Utiliza el método eliminarProducto para cambiar el estado del producto especificado (en este caso, el producto con codproducto = 1).
2. Verificación de la Eliminación:
    - Comprobación de Estado en la Base de Datos: Realiza una consulta para verificar que el estado del producto en la base de datos refleje que ha sido marcado como eliminado (estado = 0), indicando que el producto ya no está activo.

Objetivos del Test:

- **Correcta Funcionalidad del Método:** Confirmar que el método `eliminarProducto` realiza correctamente el cambio de estado del producto a 'inactivo' o 'eliminado', lo cual es esencial para aplicaciones que necesitan mantener un historial de cambios o revertir estados previos.
- **Integridad de Datos:** Asegurar que el método gestiona adecuadamente los cambios de estado sin eliminar los datos físicamente, permitiendo una fácil restauración del estado previo si es necesario.
- **Automatización de Pruebas:** Facilitar la detección temprana de errores en la implementación de la funcionalidad de eliminación de productos, ayudando a mantener la calidad y la fiabilidad del software.

EliminarUsuarioTest.php



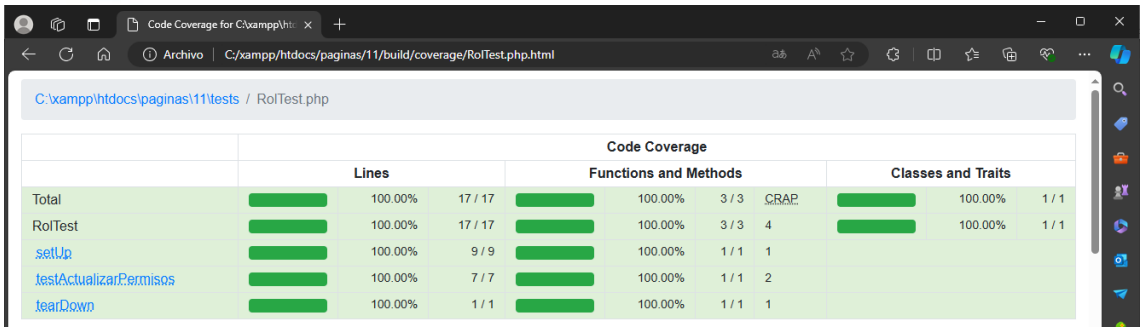
El test `testEliminarUsuario` en la clase `EliminarUsuarioTest` está diseñado para comprobar la correcta funcionalidad del método `eliminarUsuario` de la clase `UsuarioModelTest`. Este método está destinado a cambiar el estado de un usuario en la base de datos, marcándolo como eliminado (usualmente cambiando el estado a '0') en lugar de eliminar completamente el registro. Esto permite mantener la integridad del historial de datos y facilita la restauración de estados anteriores si es necesario.

1. Eliminación del Usuario: Utiliza el método eliminarUsuario para cambiar el estado del usuario especificado (en este caso, el usuario con idusuario = 1).
2. Verificación de la Eliminación:
    - Comprobación del Estado en la Base de Datos: Realiza una consulta para verificar que el estado del usuario en la base de datos refleje que ha sido marcado como eliminado (estado = 0).
    - Evaluación del Resultado del Método: Verifica que el string retornado por el método contenga un mensaje específico (como 'Usuario eliminado'), lo que indica que la eliminación fue exitosa y que el método está funcionando correctamente.

Objetivos del Test:

- **Correcta Funcionalidad del Método:** Confirmar que el método `eliminarUsuario` realiza correctamente el cambio de estado del usuario a 'inactivo' o 'eliminado', lo cual es vital para aplicaciones que requieren mantener un control de estados para sus registros.
- **Integridad de Datos:** Asegurar que el método maneja adecuadamente los cambios de estado sin eliminar los datos físicamente, permitiendo una fácil restauración del estado previo si es necesario.
- **Automatización de Pruebas:** Facilitar la detección temprana de errores en la implementación de la funcionalidad de eliminación de usuarios, ayudando a mantener la calidad y la fiabilidad del software.

RolTest.php



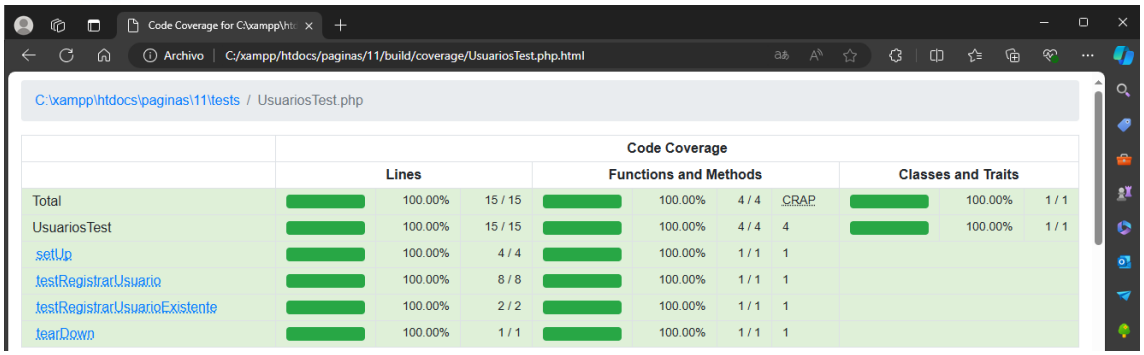
El test `testActualizarPermisos` en la clase `RolTest` está diseñado para verificar la funcionalidad de la clase `RolModelTest`, específicamente el método `actualizarPermisos`, que se encarga de gestionar los permisos asociados a un usuario en la base de datos. Este test tiene como objetivo asegurar que los permisos se actualizan correctamente y que el sistema refleja estos cambios en la tabla `detalle_permisos`.

1. Actualización de Permisos: Utiliza el método actualizarPermisos para asignar un conjunto completo de permisos (todos los disponibles) al usuario con idusuario = 1.
2. Verificación de la Actualización:
    - Comprobación de Permisos en la Base de Datos: Realiza una consulta para verificar que los permisos en detalle_permisos para el usuario indicado reflejen los cambios esperados, es decir, que contengan todos los permisos especificados.
    - Evaluación del Resultado del Método: Verifica que el resultado devuelto por el método indique que los permisos fueron actualizados correctamente, esperando un mensaje específico como 'Permisos actualizados'.

Objetivos del Test:

- **Funcionalidad Correcta del Método:** Confirmar que el método `actualizarPermisos` realiza correctamente la asignación de permisos, cambiando los permisos del usuario de acuerdo a lo especificado.
- **Integridad de Datos:** Asegurar que los cambios en la asignación de permisos se reflejen adecuadamente en la base de datos, lo que es crucial para el control de acceso dentro de la aplicación.
- **Automatización de Pruebas:** Facilitar la detección temprana de cualquier regresión o error en el manejo de permisos, contribuyendo a mantener la calidad y seguridad del sistema.

UsuariosTest.php



La clase `UsuariosTest` contiene pruebas unitarias diseñadas para validar la funcionalidad del manejo de usuarios a través de la clase `UsuarioModelTest`. Hay dos pruebas clave en esta clase: `testRegistrarUsuario` y `testRegistrarUsuarioExistente`. Cada una está destinada a asegurar que el registro de usuarios funcione de manera eficiente y correcta, tanto para nuevos usuarios como para casos donde el usuario podría estar duplicado.

1. **Objetivo:** Asegurar que el sistema maneje correctamente los intentos de registro de un usuario que usa un correo electrónico ya registrado.
2. **Proceso:**

    - **Registro de Usuario Duplicado:** Se intenta registrar un nuevo usuario usando un correo electrónico que ya existe en la base de datos.
    - **Verificación de Fallo de Registro:** Se comprueba que el sistema retorna un mensaje de error indicando que el correo ya existe.

Objetivos del Test:

- **Integridad de Datos:** Asegurar que los datos ingresados durante el registro de usuarios se almacenen correctamente y que no haya duplicados no deseados.
- **Manejo de Errores:** Verificar que el sistema identifique y maneje adecuadamente los casos donde pueda intentarse registrar un usuario con datos que violarían las reglas de unicidad, como el correo electrónico.
- **Automatización de Pruebas:** Facilitar la identificación rápida de problemas en el registro de usuarios, permitiendo que se detecten y corrijan antes de que el código se implemente en producción.

Pruebas de aceptación basadas en Desarrollo Guiado por el Comportamiento una por cada caso de uso o historia de usuario.

BDD (Desarrollo Guiado por el Comportamiento, por sus siglas en inglés) es una metodología de desarrollo de software que se centra en la colaboración entre desarrolladores, testers y partes interesadas (stakeholders) no técnicas como analistas de negocio o usuarios finales. El objetivo principal de BDD es asegurar que el software se desarrolle desde la perspectiva del

comportamiento del sistema, lo cual implica definir el comportamiento esperado antes de comenzar a implementar cualquier código.

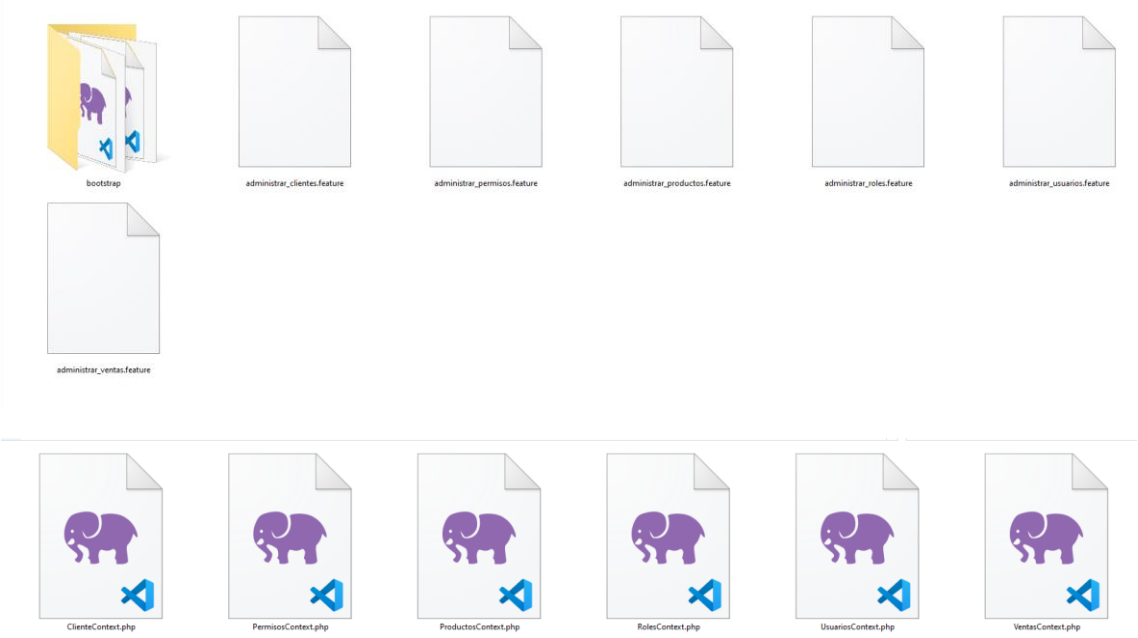
Given (Dado): Describe el estado inicial o contexto antes de que ocurra la acción que estamos probando.

When (Cuando): Describe la acción o evento que queremos probar.

Then (Entonces): Describe el resultado esperado después de que ocurra la acción.

Estos escenarios son escritos en colaboración con los stakeholders y son utilizados para guiar tanto el desarrollo como las pruebas. Los pasos de BDD aseguran que el equipo entienda claramente lo que se está construyendo y que todos tengan una visión común del comportamiento esperado del sistema.

TEST PARA LOS ESCENARIOS (FEATURE)



Escenario para cotización (feature)

```
Feature: Administrar Clientes
  Como Personal administrativo
  Quiero administrar clientes en el sistema
  Para poder crear, visualizar, actualizar y eliminar clientes

  Scenario: Crear Cliente
    Given que el Personal navega a la página de administración de clientes
    And selecciona "Crear nuevo cliente"
```

And no ingresa datos enviando el formulario
And envía el formulario
Then el sistema muestra un mensaje de error "Los campos del formulario no pueden estar vacíos"
And completa el formulario con la información del cliente
And envía el formulario
Then el sistema guarda el nuevo cliente en la base de datos
And muestra un mensaje de confirmación

Scenario: Visualizar Clientes
Given que el Personal navega a la página de administración de clientes
Then el sistema muestra la lista de clientes disponibles

Scenario: Actualizar Cliente
Given que el Personal navega a la página de administración de clientes
And selecciona un cliente para actualizar
And modifica la información del cliente
And envía el formulario
Then el sistema actualiza el cliente en la base de datos
And muestra un mensaje de confirmación

Scenario: Eliminar Cliente
Given que el Personal navega a la página de administración de clientes
And selecciona un cliente para eliminar
And confirma la eliminación
Then el sistema elimina el cliente de la base de datos
And muestra un mensaje de confirmación

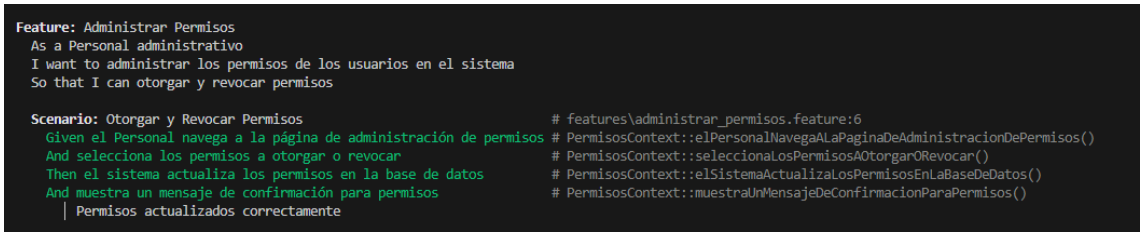
Resultado del Escenario

```
Feature: Administrar Clientes
  Como Personal administrativo
  Quiero administrar clientes en el sistema
  Para poder crear, visualizar, actualizar y eliminar clientes

  Scenario: Crear Cliente                                                                  # features\administrar_clientes.feature:6
    Given que el Personal navega a la página de administración de clientes                 # ClienteContext::queElPersonalNavegaALaPaginaDeAdministracionDeClientes()
    And selecciona "Crear nuevo cliente"                                                   # ClienteContext::seleccionaCrearNuevoCliente()
    And no ingresa datos enviando el formulario                                            # ClienteContext::noIngresaDatosEnviandoElFormulario()
    And envía el formulario                                                                # ClienteContext::enviaElFormulario()
    Then el sistema muestra un mensaje de error "Los campos del formulario no pueden estar vacíos" # ClienteContext::elSistemaMuestraUnMensajeDeError()
    And completa el formulario con la información del cliente                              # ClienteContext::completaElFormularioConLaInformacionDelCliente()
    And envía el formulario                                                                # ClienteContext::enviaElFormulario()
    Then el sistema guarda el nuevo cliente en la base de datos                            # ClienteContext::elSistemaGuardaElNuevoClienteEnLaBaseDeDatos()
    And muestra un mensaje de confirmación                                                 # ClienteContext::muestraUnMensajeDeConfirmacion()
      | Cliente creado correctamente

  Scenario: Visualizar Clientes                                  # features\administrar_clientes.feature:17
    Given que el Personal navega a la página de administración de clientes # ClienteContext::queElPersonalNavegaALaPaginaDeAdministracionDeClientes()
    Then el sistema muestra la lista de clientes disponibles     # ClienteContext::elSistemaMuestraLaListaDeClientesDisponibles()

  Scenario: Actualizar Cliente                                   # features\administrar_clientes.feature:21
    Given que el Personal navega a la página de administración de clientes # ClienteContext::queElPersonalNavegaALaPaginaDeAdministracionDeClientes()
    And selecciona un cliente para actualizar                    # ClienteContext::seleccionaUnClienteParaActualizar()
    And modifica la información del cliente                      # ClienteContext::modificaLaInformacionDelCliente()
    And envía el formulario                                      # ClienteContext::enviaElFormulario()
    Then el sistema actualiza el cliente en la base de datos     # ClienteContext::elSistemaActualizaElClienteEnLaBaseDeDatos()
    And muestra un mensaje de confirmación                       # ClienteContext::muestraUnMensajeDeConfirmacion()
      | Cliente creado correctamente

  Scenario: Eliminar Cliente                                     # features\administrar_clientes.feature:29
    Given que el Personal navega a la página de administración de clientes # ClienteContext::queElPersonalNavegaALaPaginaDeAdministracionDeClientes()
    And selecciona un cliente para eliminar                      # ClienteContext::seleccionaUnClienteParaEliminar()
    And confirma la eliminación                                  # ClienteContext::confirmaLaEliminacion()
    Then el sistema elimina el cliente de la base de datos       # ClienteContext::elSistemaEliminaElClienteDeLaBaseDeDatos()
    And muestra un mensaje de confirmación                       # ClienteContext::muestraUnMensajeDeConfirmacion()
      | Cliente creado correctamente
```

Feature: Administrar Permisos
As a Personal administrativo
I want to administrar los permisos de los usuarios en el sistema
So that I can otorgar y revocar permisos

Scenario: Otorgar y Revocar Permisos
Given el Personal navega a la página de administración de permisos
And selecciona los permisos a otorgar o revocar
Then el sistema actualiza los permisos en la base de datos
And muestra un mensaje de confirmación para permisos

Resultado del Escenario



Feature: Administrar Productos
As a Personal administrativo
I want to administrar productos en el sistema
So that I can crear, visualizar, actualizar y eliminar productos

Scenario: Crear Producto
Given el Personal ha navegado a la página de administración de productos
When crea un nuevo producto con nombre "Nuevo Producto", precio "100" y descripción "Descripción del nuevo producto"
Then el sistema debería guardar el nuevo producto en la base de datos
And debería mostrar un mensaje de confirmación

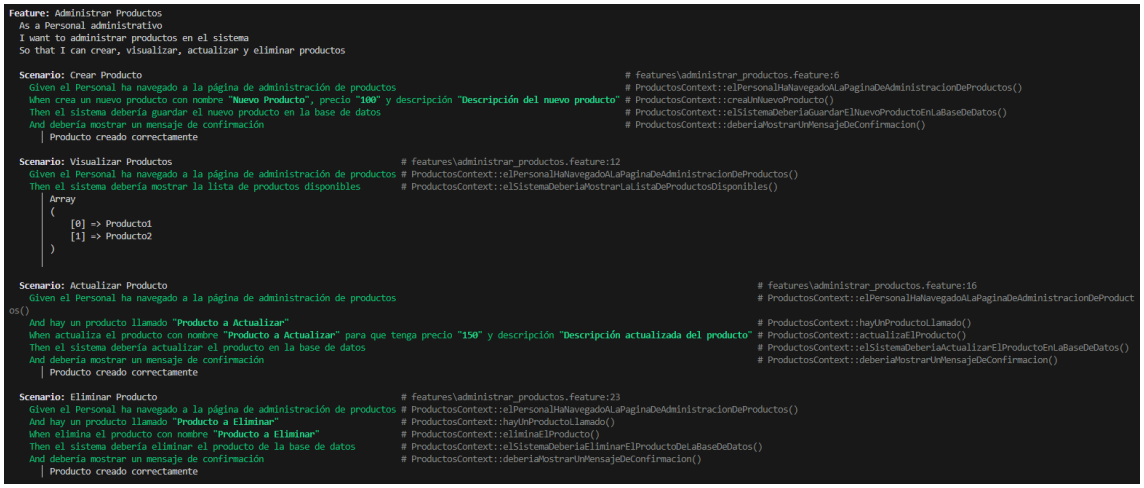
Scenario: Visualizar Productos
Given el Personal ha navegado a la página de administración de productos
Then el sistema debería mostrar la lista de productos disponibles

Scenario: Actualizar Producto
Given el Personal ha navegado a la página de administración de productos
And hay un producto llamado "Producto a Actualizar"
When actualiza el producto con nombre "Producto a Actualizar" para que tenga precio "150" y descripción "Descripción actualizada del producto"
Then el sistema debería actualizar el producto en la base de datos
And debería mostrar un mensaje de confirmación

Scenario: Eliminar Producto
Given el Personal ha navegado a la página de administración de productos

And hay un producto llamado "Producto a Eliminar"
When elimina el producto con nombre "Producto a Eliminar"
Then el sistema debería eliminar el producto de la base de datos
And debería mostrar un mensaje de confirmación

Resultado del Escenario

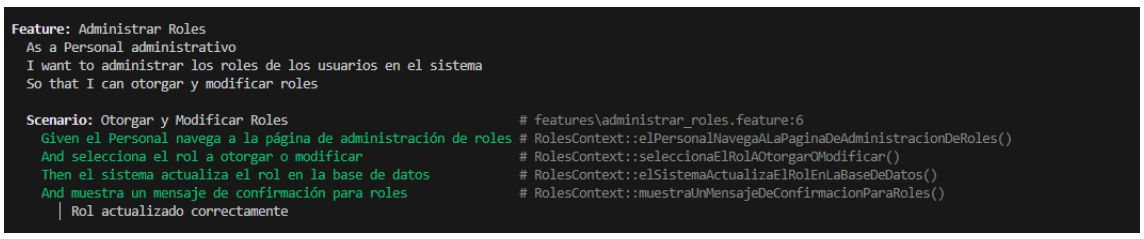


```
Feature: Administrar Productos
  As a Personal administrativo
  I want to administrar productos en el sistema
  So that I can crear, visualizar, actualizar y eliminar productos

  Scenario: Crear Producto                                                                    # features\administrar_productos.feature:6
    Given el Personal ha navegado a la página de administración de productos                  # ProductosContext::elPersonalHaNavegadoALaPaginaDeAdministracionDeProductos()
    When crea un nuevo producto con nombre "Nuevo Producto", precio "100" y descripción "Descripción del nuevo producto" # ProductosContext::creaUnNuevoProducto()
    Then el sistema debería guardar el nuevo producto en la base de datos                    # ProductosContext::elSistemaDeberiaGuardarElNuevoProductoEnLaBaseDeDatos()
    And debería mostrar un mensaje de confirmación                                           # ProductosContext::deberiaMostrarUnMensajeDeConfirmacion()
      | Producto creado correctamente

  Scenario: Visualizar Productos                                  # features\administrar_productos.feature:12
    Given el Personal ha navegado a la página de administración de productos # ProductosContext::elPersonalHaNavegadoALaPaginaDeAdministracionDeProductos()
    Then el sistema debería mostrar la lista de productos disponibles       # ProductosContext::elSistemaDeberiaMostrarLaListaDeProductosDisponibles()
      | Array
      | (
      |     [0] => Producto1
      |     [1] => Producto2
      | )

  Scenario: Actualizar Producto                                                                                  # features\administrar_productos.feature:16
    Given el Personal ha navegado a la página de administración de productos                                     # ProductosContext::elPersonalHaNavegadoALaPaginaDeAdministracionDeProduct
os()
    And hay un producto llamado "Producto a Actualizar"                                                          # ProductosContext::hayUnProductoLlamado()
    When actualiza el producto con nombre "Producto a Actualizar" para que tenga precio "150" y descripción "Descripción actualizada del producto" # ProductosContext::actualizaElProducto()
    Then el sistema debería actualizar el producto en la base de datos                                           # ProductosContext::elSistemaDeberiaActualizarElProductoEnLaBaseDeDatos()
    And debería mostrar un mensaje de confirmación                                                               # ProductosContext::deberiaMostrarUnMensajeDeConfirmacion()
      | Producto creado correctamente

  Scenario: Eliminar Producto                                     # features\administrar_productos.feature:23
    Given el Personal ha navegado a la página de administración de productos # ProductosContext::elPersonalHaNavegadoALaPaginaDeAdministracionDeProductos()
    And hay un producto llamado "Producto a Eliminar"             # ProductosContext::hayUnProductoLlamado()
    When elimina el producto con nombre "Producto a Eliminar"     # ProductosContext::eliminaElProducto()
    Then el sistema debería eliminar el producto de la base de datos # ProductosContext::elSistemaDeberiaEliminarElProductoDeLaBaseDeDatos()
    And debería mostrar un mensaje de confirmación                # ProductosContext::deberiaMostrarUnMensajeDeConfirmacion()
      | Producto creado correctamente
```

Feature: Administrar Roles
As a Personal administrativo
I want to administrar los roles de los usuarios en el sistema
So that I can otorgar y modificar roles

Scenario: Otorgar y Modificar Roles
Given el Personal navega a la página de administración de roles
And selecciona el rol a otorgar o modificar
Then el sistema actualiza el rol en la base de datos
And muestra un mensaje de confirmación para roles

Resultado del Escenario



```
Feature: Administrar Roles
  As a Personal administrativo
  I want to administrar los roles de los usuarios en el sistema
  So that I can otorgar y modificar roles

  Scenario: Otorgar y Modificar Roles                          # features\administrar_roles.feature:6
    Given el Personal navega a la página de administración de roles # RolesContext::elPersonalNavegaALaPaginaDeAdministracionDeRoles()
    And selecciona el rol a otorgar o modificar                # RolesContext::seleccionaElRolAOtorgarOModificar()
    Then el sistema actualiza el rol en la base de datos       # RolesContext::elSistemaActualizaElRolEnLaBaseDeDatos()
    And muestra un mensaje de confirmación para roles          # RolesContext::muestraUnMensajeDeConfirmacionParaRoles()
      | Rol actualizado correctamente
```

Feature: Administrar Usuarios
As a Personal administrativo
I want to administrar usuarios en el sistema
So that I can crear, visualizar, actualizar y eliminar usuarios

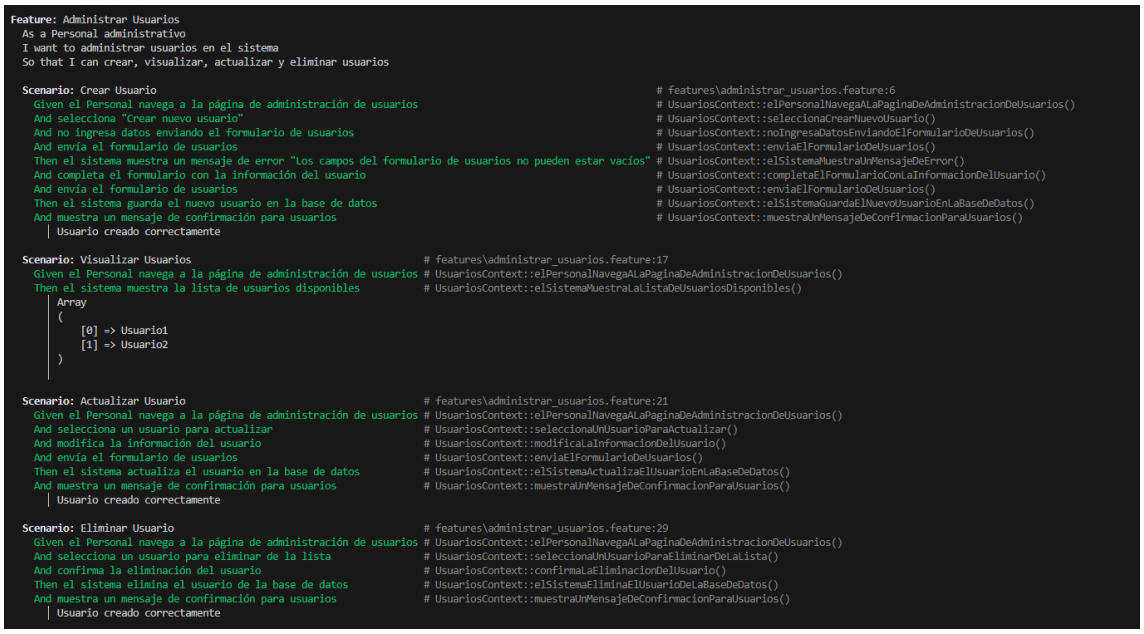
Scenario: Crear Usuario
Given el Personal navega a la página de administración de usuarios
And selecciona "Crear nuevo usuario"
And no ingresa datos enviando el formulario de usuarios
And envía el formulario de usuarios
Then el sistema muestra un mensaje de error "Los campos del formulario de usuarios no pueden estar vacíos"
And completa el formulario con la información del usuario
And envía el formulario de usuarios
Then el sistema guarda el nuevo usuario en la base de datos
And muestra un mensaje de confirmación para usuarios

Scenario: Visualizar Usuarios
Given el Personal navega a la página de administración de usuarios
Then el sistema muestra la lista de usuarios disponibles

Scenario: Actualizar Usuario
Given el Personal navega a la página de administración de usuarios
And selecciona un usuario para actualizar
And modifica la información del usuario
And envía el formulario de usuarios
Then el sistema actualiza el usuario en la base de datos
And muestra un mensaje de confirmación para usuarios

Scenario: Eliminar Usuario
Given el Personal navega a la página de administración de usuarios
And selecciona un usuario para eliminar de la lista
And confirma la eliminación del usuario
Then el sistema elimina el usuario de la base de datos
And muestra un mensaje de confirmación para usuarios

Resultado del Escenario



```
Feature: Administrar Usuarios
  As a Personal administrativo
  I want to administrar usuarios en el sistema
  So that I can crear, visualizar, actualizar y eliminar usuarios

  Scenario: Crear Usuario                                                                                  # features\administrar_usuarios.feature:6
    Given el Personal navega a la página de administración de usuarios                                     # UsuariosContext::elPersonalNavegaALaPaginaDeAdministracionDeUsuarios()
    And selecciona "Crear nuevo usuario"                                                                   # UsuariosContext::seleccionaCrearNuevoUsuario()
    And no ingresa datos enviando el formulario de usuarios                                                # UsuariosContext::noIngresaDatosEnviandoElFormularioDeUsuarios()
    And envía el formulario de usuarios                                                                    # UsuariosContext::enviaElFormularioDeUsuarios()
    Then el sistema muestra un mensaje de error "Los campos del formulario de usuarios no pueden estar vacios" # UsuariosContext::elSistemaMuestraUnMensajeDeError()
    And completa el formulario con la información del usuario                                              # UsuariosContext::completaElFormularioConLaInformacionDelUsuario()
    And envía el formulario de usuarios                                                                    # UsuariosContext::enviaElFormularioDeUsuarios()
    Then el sistema guarda el nuevo usuario en la base de datos                                            # UsuariosContext::elSistemaGuardaElNuevoUsuarioEnLaBaseDeDatos()
    And muestra un mensaje de confirmación para usuarios                                                   # UsuariosContext::muestraUnMensajeDeConfirmacionParaUsuarios()
      | Usuario creado correctamente

  Scenario: Visualizar Usuarios                                 # features\administrar_usuarios.feature:17
    Given el Personal navega a la página de administración de usuarios # UsuariosContext::elPersonalNavegaALaPaginaDeAdministracionDeUsuarios()
    Then el sistema muestra la lista de usuarios disponibles    # UsuariosContext::elSistemaMuestraLaListaDeUsuariosDisponibles()
      | Array
      | (
      |     [0] => Usuario1
      |     [1] => Usuario2
      | )

  Scenario: Actualizar Usuario                                  # features\administrar_usuarios.feature:21
    Given el Personal navega a la página de administración de usuarios # UsuariosContext::elPersonalNavegaALaPaginaDeAdministracionDeUsuarios()
    And selecciona un usuario para actualizar                   # UsuariosContext::seleccionaUnUsuarioParaActualizar()
    And modifica la información del usuario                     # UsuariosContext::modificaLaInformacionDelUsuario()
    And envía el formulario de usuarios                         # UsuariosContext::enviaElFormularioDeUsuarios()
    Then el sistema actualiza el usuario en la base de datos    # UsuariosContext::elSistemaActualizaElUsuarioEnLaBaseDeDatos()
    And muestra un mensaje de confirmación para usuarios        # UsuariosContext::muestraUnMensajeDeConfirmacionParaUsuarios()
      | Usuario creado correctamente

  Scenario: Eliminar Usuario                                    # features\administrar_usuarios.feature:29
    Given el Personal navega a la página de administración de usuarios # UsuariosContext::elPersonalNavegaALaPaginaDeAdministracionDeUsuarios()
    And selecciona un usuario para eliminar de la lista         # UsuariosContext::seleccionaUnUsuarioParaEliminarDeLaLista()
    And confirma la eliminación del usuario                     # UsuariosContext::confirmaLaEliminacionDelUsuario()
    Then el sistema elimina el usuario de la base de datos      # UsuariosContext::elSistemaEliminaElUsuarioDeLaBaseDeDatos()
    And muestra un mensaje de confirmación para usuarios        # UsuariosContext::muestraUnMensajeDeConfirmacionParaUsuarios()
      | Usuario creado correctamente
```

Feature: Administrar Ventas
As a Personal administrativo
I want to administrar ventas en el sistema
So that I can visualizar, actualizar y eliminar ventas

Scenario: Visualizar Ventas
Given el Personal navega a la página de administración de ventas
Then el sistema muestra la lista de ventas disponibles

Scenario: Actualizar Venta
Given el Personal navega a la página de administración de ventas
And selecciona una venta para actualizar
And modifica la información de la venta con datos incompletos
And envía el formulario de ventas
Then el sistema muestra un mensaje de error para ventas "Los campos del formulario no pueden estar vacíos"
And completa el formulario con la información correcta de la venta
And envía el formulario de ventas
Then el sistema actualiza la venta en la base de datos
And muestra un mensaje de confirmación para ventas actualizado

Scenario: Eliminar Venta
Given el Personal navega a la página de administración de ventas
And selecciona una venta para eliminar
And confirma la eliminación de la venta
Then el sistema elimina la venta de la base de datos
And muestra un mensaje de confirmación para ventas eliminado

Resultado del Escenario

```
Feature: Administrar Ventas
  As a Personal administrativo
  I want to administrar ventas en el sistema
  So that I can visualizar, actualizar y eliminar ventas

  Scenario: Visualizar Ventas                                        # features\administrar_ventas.feature:6
    Given el Personal navega a la página de administración de ventas # VentasContext::elPersonalNavegaALaPaginaDeAdministracionDeVentas()
    Then el sistema muestra la lista de ventas disponibles           # VentasContext::elSistemaMuestraLaListaDeVentasDisponibles()
      │ Array
      │ (
      │     [0] => Venta1
      │     [1] => Venta2
      │ )
      │

  Scenario: Actualizar Venta                                                                                      # features\administrar_ventas.feature:10
    Given el Personal navega a la página de administración de ventas                                              # VentasContext::elPersonalNavegaALaPaginaDeAdministracionDeVentas()
    And selecciona una venta para actualizar                                                                      # VentasContext::seleccionaUnaVentaParaActualizar()
    And modifica la información de la venta con datos incompletos                                                 # VentasContext::modificaLaInformacionDeLaVentaConDatosIncompletos()
    And envía el formulario de ventas                                                                             # VentasContext::enviaElFormularioDeVentas()
      │ Los campos del formulario no pueden estar vacíos
    Then el sistema muestra un mensaje de error para ventas "Los campos del formulario no pueden estar vacíos" # VentasContext::elSistemaMuestraUnMensajeDeErrorParaVentas()
      │ Los campos del formulario no pueden estar vacíos
    And completa el formulario con la información correcta de la venta                                            # VentasContext::completaElFormularioConLaInformacionCorrectaDeLaVenta()
    And envía el formulario de ventas                                                                             # VentasContext::enviaElFormularioDeVentas()
      │ Los campos del formulario no pueden estar vacíos
    Then el sistema actualiza la venta en la base de datos                                                        # VentasContext::elSistemaActualizaLaVentaEnLaBaseDeDatos()
    Then el sistema actualiza la venta en la base de datos                                                        # VentasContext::elSistemaActualizaLaVentaEnLaBaseDeDatos()
    Then el sistema actualiza la venta en la base de datos                                                        # VentasContext::elSistemaActualizaLaVentaEnLaBaseDeDatos()
    Then el sistema actualiza la venta en la base de datos                                                        # VentasContext::elSistemaActualizaLaVentaEnLaBaseDeDatos()
    Then el sistema actualiza la venta en la base de datos                                                        # VentasContext::elSistemaActualizaLaVentaEnLaBaseDeDatos()
    Then el sistema actualiza la venta en la base de datos                                                        # VentasContext::elSistemaActualizaLaVentaEnLaBaseDeDatos()
    Then el sistema actualiza la venta en la base de datos                                                        # VentasContext::elSistemaActualizaLaVentaEnLaBaseDeDatos()
    Then el sistema actualiza la venta en la base de datos                                                        # VentasContext::elSistemaActualizaLaVentaEnLaBaseDeDatos()
      │ Venta actualizada correctamente
    And muestra un mensaje de confirmación para ventas actualizado                                                # VentasContext::muestraUnMensajeDeConfirmacionParaVentasActualizado()
      │ Venta actualizada correctamente

  Scenario: Eliminar Venta                                           # features\administrar_ventas.feature:21
    Given el Personal navega a la página de administración de ventas # VentasContext::elPersonalNavegaALaPaginaDeAdministracionDeVentas()
    And selecciona una venta para eliminar                           # VentasContext::seleccionaUnaVentaParaEliminar()
    And confirma la eliminación de la venta                          # VentasContext::confirmaLaEliminacionDeLaVenta()
    Then el sistema elimina la venta de la base de datos             # VentasContext::elSistemaEliminaLaVentaDeLaBaseDeDatos()
      │ Venta eliminada correctamente
    And muestra un mensaje de confirmación para ventas eliminado     # VentasContext::muestraUnMensajeDeConfirmacionParaVentasEliminado()
      │ Venta eliminada correctamente

17 scenarios (17 passed)
84 steps (84 passed)
0m1.09s (10.15Mb)
```

Para realizar el informe se usó la dependencia `emuse/BehatHTMLFormatter` es un formateador de informes para Behat que genera informes detallados en formato HTML.

Para instalar esta dependencia se utiliza el siguiente comando →

- **composer require --dev emuse/behat-html-formatter**

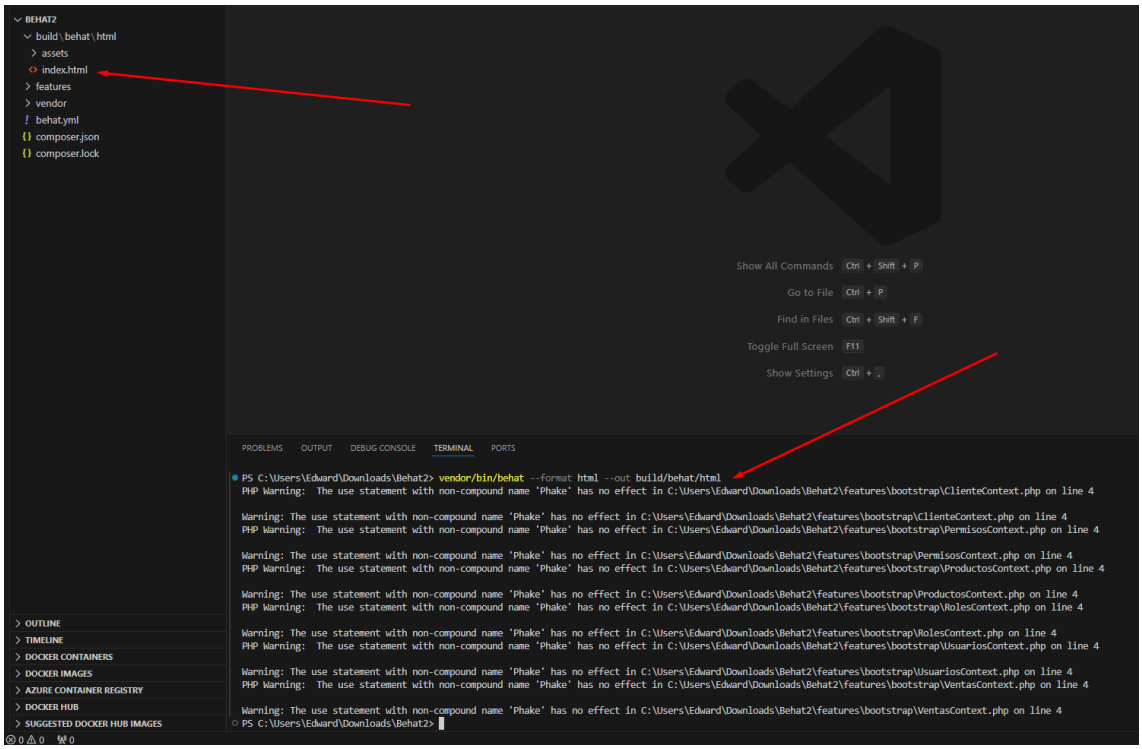
```
PS D:\CALIDAD\Behat2> composer require --dev emuse/behat-html-formatter
./composer.json has been updated
Running composer update emuse/behat-html-formatter
Loading composer repositories with package information
Updating dependencies
Lock file operations: 0 installs, 1 update, 0 removals
  - Upgrading emuse/behat-html-formatter (v0.2.0 => v1.0.0)
Writing lock file
Installing dependencies from lock file (including require-dev)
Package operations: 0 installs, 1 update, 0 removals
  - Downloading emuse/behat-html-formatter (v1.0.0)
  - Upgrading emuse/behat-html-formatter (v0.2.0 => v1.0.0): Extracting archive
Generating autoload files
19 packages you are using are looking for funding.
Use the `composer fund` command to find out more!
No security vulnerability advisories found.
Using version ^1.0 for emuse/behat-html-formatter
PS D:\CALIDAD\Behat2>
```

Luego modificamos el archivo .yml →

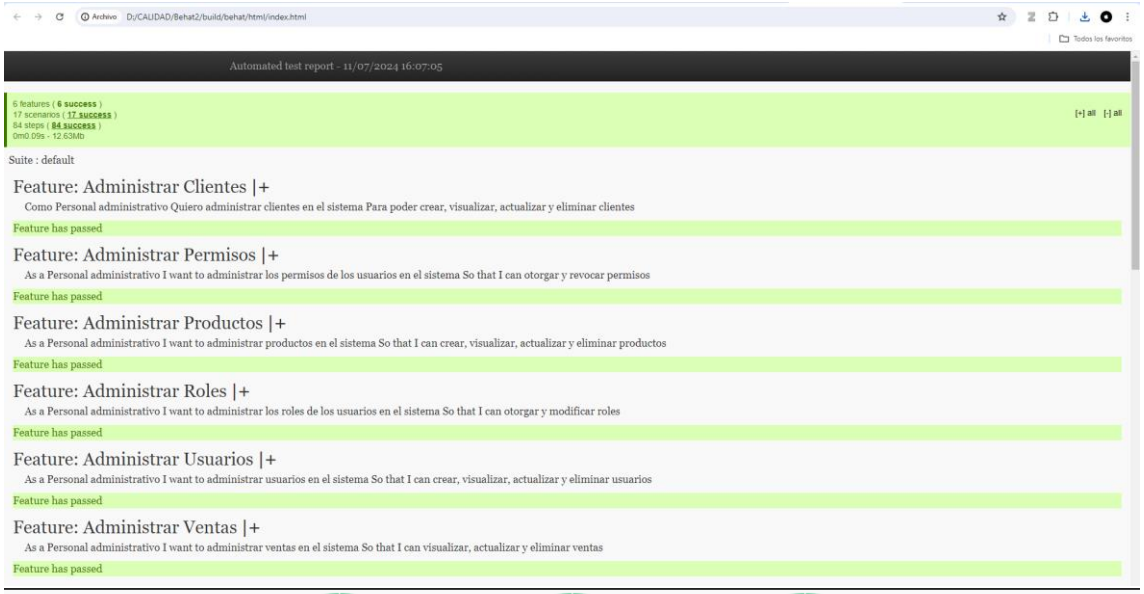
```yaml
default:
  suites:
    default:
      paths:
        - "%paths.base%/features"
      contexts:
        - ClienteContext
        - PermisosContext
        - ProductosContext
        - RolesContext
        - UsuariosContext
        - VentasContext
  # formatters:
  #   junit:
      # output_path: "build/junit"
  extensions:
    emuse\BehatHTMLFormatter\BehatHTMLFormatterExtension:
      name: html
      renderer: "Twig,Behat2"
      file_name: index
      print_args: true
      print_outp: true
      loop_break: true
```

Por ultimo ejecutamos el siguiente comando →

vendor/bin/behat --format html --out build/behat/html

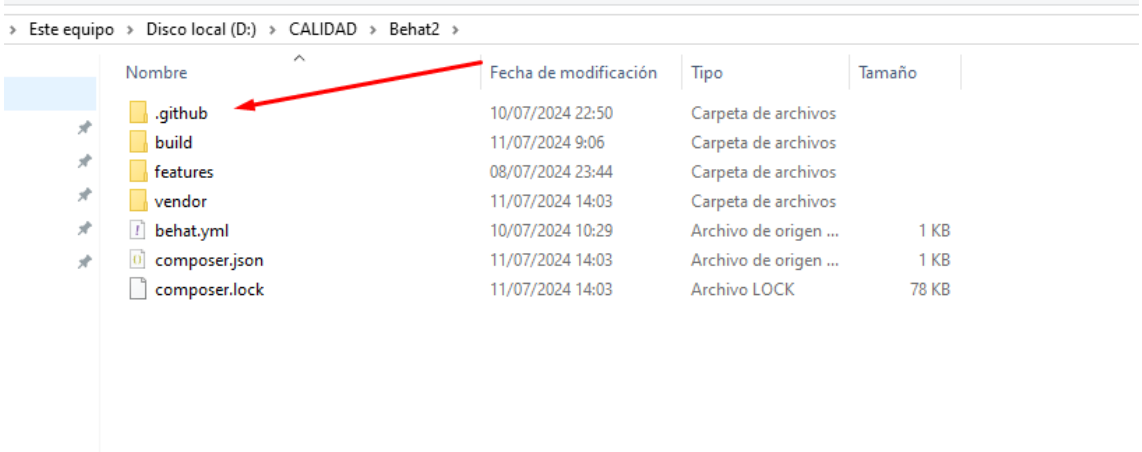


Resultado del Escenario

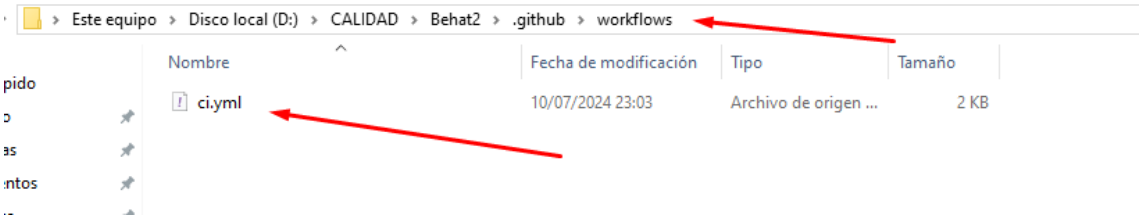


Desplegar las pruebas automatizadas en una solución de gestión de pruebas, pudiendo utilizar como Azure Devops, AWS Codepipeline, Google Code Build, Github actions, etc.

Creamos la parte .github



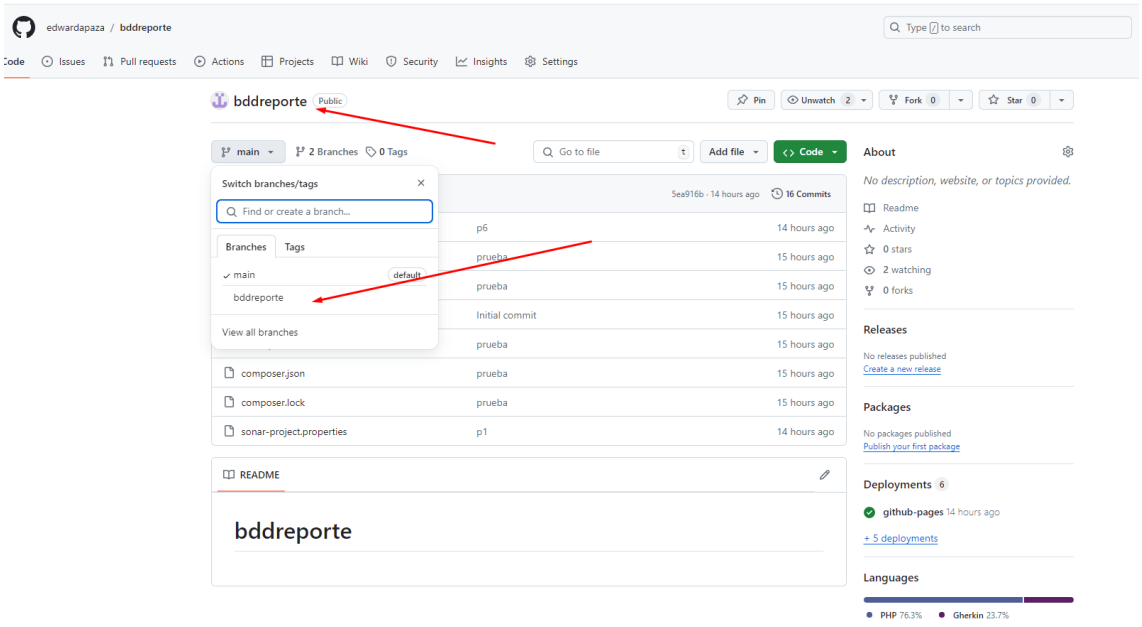
dentro otra carpeta workflows y ahi el archivo ci.yml →



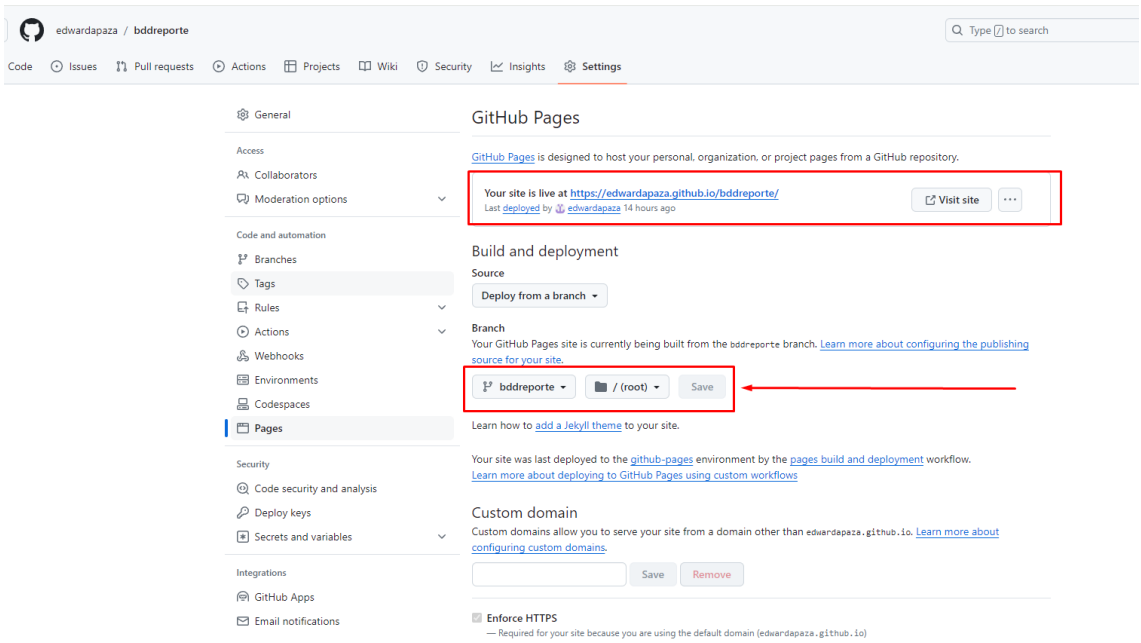
Dentro del archivo ci.yml →

```yaml
name: Tarea Automatizada de ejecución de pruebas

env:
  SONAR_ORG: 'edwardapaza'                # Nombre de la organización en SonarCloud
  SONAR_PROJECT: 'edwardapaza_bddreporte'   # Key ID del proyecto en SonarCloud

on:
  push:
    branches: [ "main" ]
  workflow_dispatch:

jobs:
  build:
    name: Build and Analyze
    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v3

    - name: Set up PHP
      uses: shivammathur/setup-php@v2
      with:
        php-version: '8.2' # Ajusta la versión de PHP según tus necesidades
        extensions: mbstring, intl, pdo_mysql

    - name: Install Composer dependencies
      run: composer install

    - name: Create reports directory
      run: mkdir -p features/bootstrap/reports

    - name: Give execute permission to Behat
      run: chmod +x vendor/bin/behat

    - name: Run Behat tests
      run: vendor/bin/behat --format=html --out=features/bootstrap/reports/index.html

    - name: SonarCloud Scan
      uses: SonarSource/sonarcloud-github-action@master
      env:
        GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }} # Needed to get PR information, if any
        SONAR_TOKEN: ${{ secrets.SONAR_TOKEN }}

    - name: Upload Behat HTML report
      if: always()
      uses: actions/upload-artifact@v2
      with:
        name: behat-report
        path: features/bootstrap/reports/index.html

    - name: Deploy Behat HTML report to GitHub Pages
      if: always()
      uses: peaceiris/actions-gh-pages@v3
      with:
        personal_token: ${{ secrets.PAT_TOKEN }}
        publish_branch: bddreporte
        publish_dir: features/bootstrap/reports
```

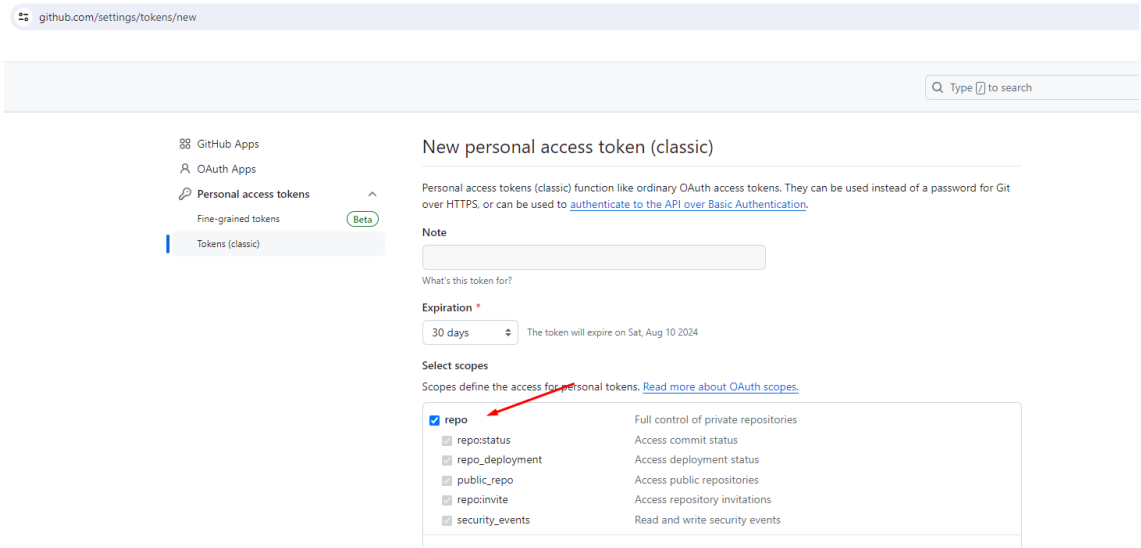
Creamos un repositorio con el nombre bddreporte y la rama bddreporte →



Nos dirigimos a settings → Pages → cambiamos la rama a bddreporte y se generara el web page site →

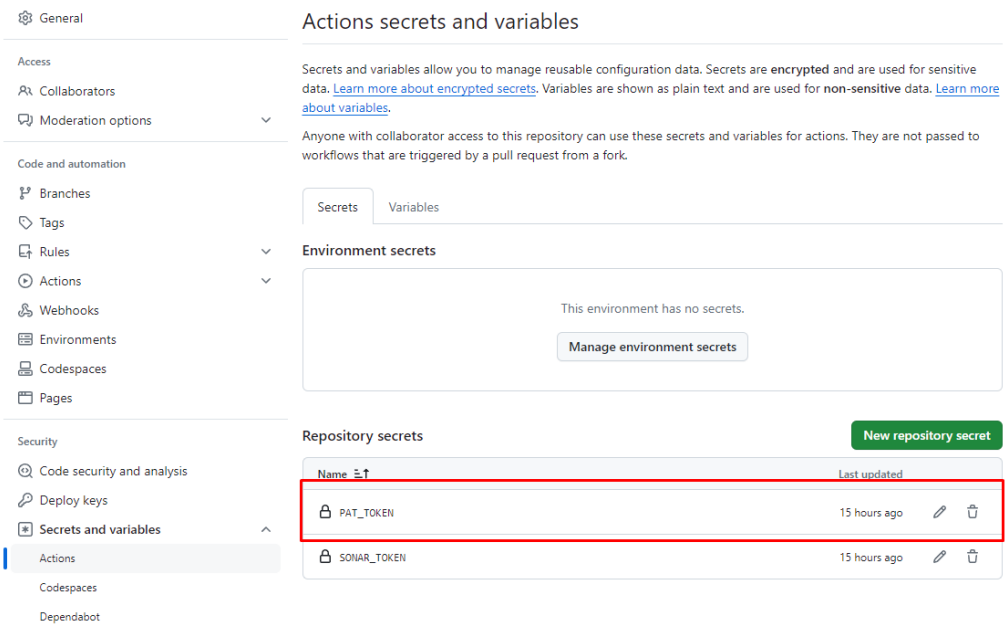


Luego generamos el PAT_TOKEN → nos dirigimos a esta url →
https://github.com/settings/tokens/new

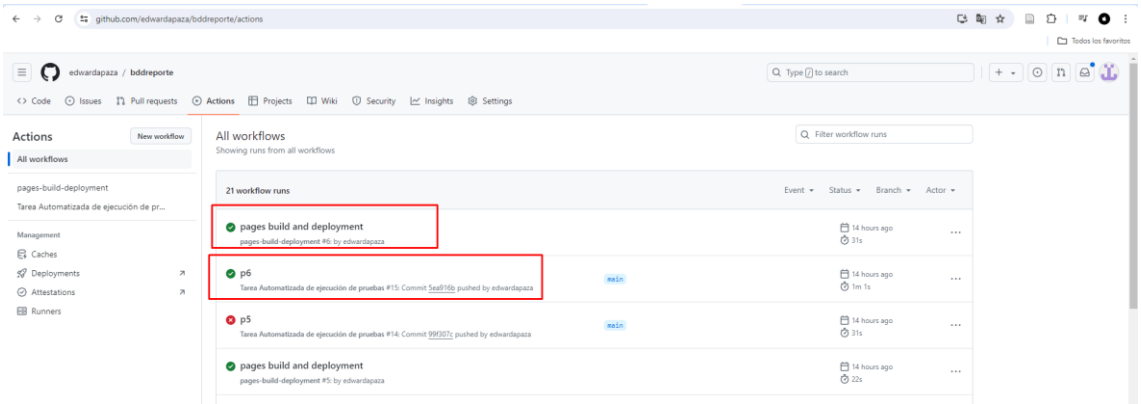


Le damos en click en →

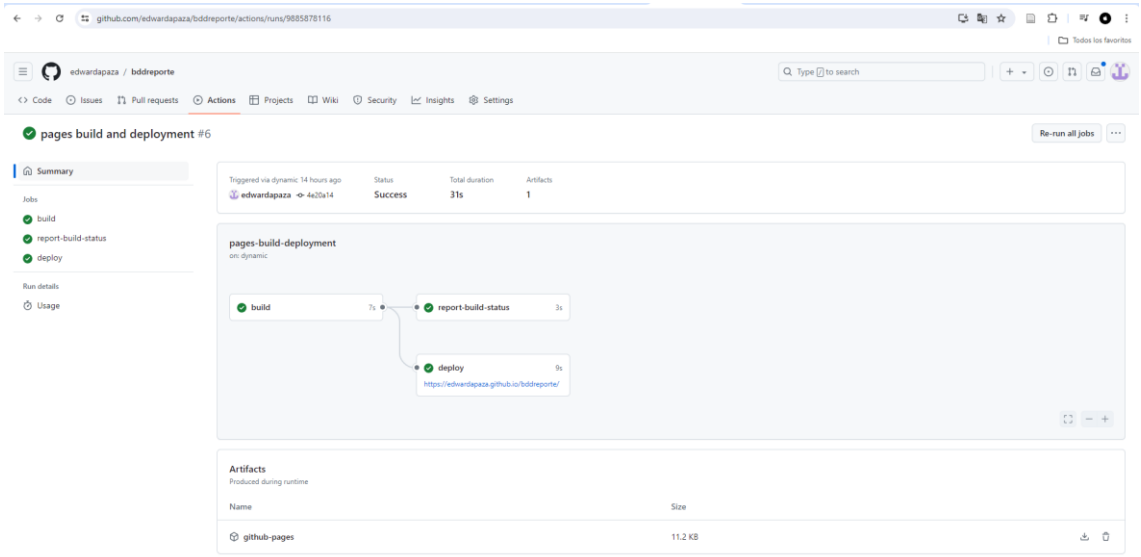
Luego nos dirigimos a settings → settings and variables → actions → creamos New repositoy secret → PAT_TOKEN y agregamos la key que nos genero anteriormente



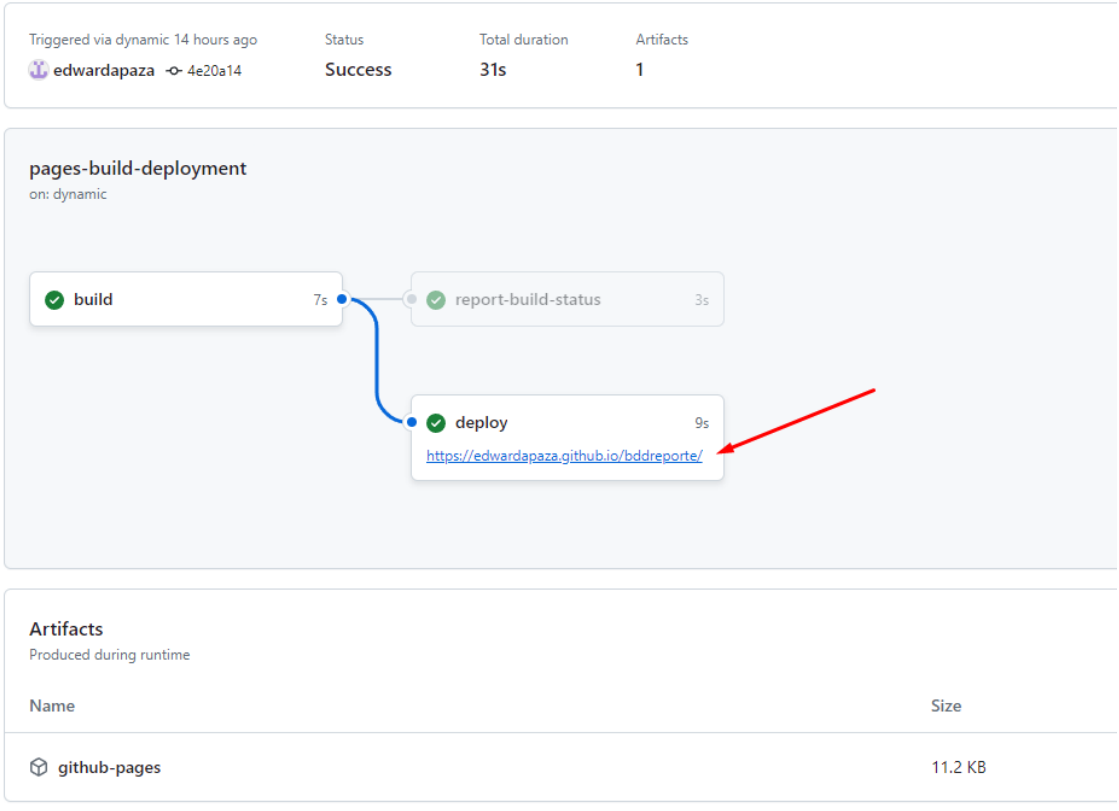
Luego haremos el su respectivo git add . → git commit → git push →



Nos mostrara el diagrama → pages-build-deployment →



Hacemos click →



Nos mostrará el reporte bdd →