INFORME

I. PORTADA

Tema:	Herramientas para la Calidad
Unidad de Organización Curricular:	PROFESIONAL
Nivel y Paralelo:	Sexto - A
Alumnos participantes:	Carrasco Paredes Kevin Andrés
	Chimborazo Guamán William Andrés
	Quishpe Lopez Luis Alexander
	Sailema Gavilánez Ismael Alexander
Asignatura:	Gestión de Calidad del software
Docente:	Ing. José Caiza, Mg.

II. INFORME DE GUÍA PRÁCTICA

1.1 Objetivos

General:

Fortalecer las competencias de los participantes en el uso de herramientas de calidad de software, mediante la comprensión de su propósito, la integración práctica de SonarQube, ESLint y Selenium, y la ejecución de un análisis estático funcional sobre un proyecto real que permita evaluar y mejorar la calidad del código.

Específicos:

- Diagnosticar los conocimientos iniciales del curso en herramientas de calidad.
- Explicar para qué sirven SonarQube, ESLint y Selenium y cómo se complementan.
- Dejar funcionando un análisis estático con SonarQube sobre un proyecto pequeño (.NET o JS) y revisar el primer reporte.

1.2 Modalidad

Presencia

1.3 Tiempo de duración

Presenciales: 4

No presenciales: 0

1.4 Instrucciones

- Verificar requisitos (Docker o Java, Git, entorno .NET o Node.js).
- Preparar un proyecto pequeño con algunas malas prácticas.
- Iniciar SonarQube localmente.
- Crear proyecto y generar token de acceso.
- Ejecutar el análisis estático del proyecto (.NET o JS).
- Revisar resultados en SonarQube: Issues, Métricas y Quality Gate.

- Ajustar reglas y parámetros del Quality Gate.
- Corregir tres problemas detectados y volver a analizar.
- Registrar evidencias: capturas, configuraciones y lista de correcciones.
- (Opcional) Integrar ESLint y considerar pruebas E2E con Selenium.

1.5 Listado de equipos, materiales y recursos

TAC (Tecnologías para el Aprendizaje y Conocimiento) empleados en la guía práctica:

☒Plataformas educativas

☐Simuladores y laboratorios virtuales

☒Aplicaciones educativas

☒Recursos audiovisuales

☐Gamificación

☐Inteligencia Artificial

Otros (Especifique): _____

1.6 Actividades por desarrollar

- Verifica que el entorno cumpla los requisitos (Docker o Java, Git, y un proyecto en .NET o JavaScript).
- Prepara un proyecto pequeño con algunas malas prácticas intencionales para generar hallazgos.
- Inicia el servicio de SonarQube localmente.
- Crea un proyecto dentro de SonarQube y genera un token de autenticación.
- Ejecuta el análisis estático del proyecto utilizando el escáner correspondiente.
- Revisa los resultados en la interfaz de SonarQube: métricas, incidencias y el estado del Quality Gate.
- Ajusta las reglas del análisis y los umbrales del Quality Gate según los objetivos del ejercicio.
- Corrige tres incidencias identificadas y repite el análisis para comparar los resultados.
- Reúne las evidencias: capturas del panel de resultados, configuraciones y lista de correcciones realizadas.
- (Opcional) Integra ESLint para análisis de estilo y Selenium para pruebas de interfaz complementarias.

1.7 Resultados obtenidos

Marco Teórico

El aseguramiento de la calidad del software requiere la integración de herramientas automatizadas que permitan detectar errores, vulnerabilidades y malas prácticas de programación de forma temprana. En este contexto, plataformas de análisis estático y

dinámico como SonarQube, ESLint y Selenium se han consolidado como pilares en los procesos de desarrollo ágil y de integración continua (CI/CD).

SonarQube

SonarQube es una plataforma de análisis estático de código que soporta múltiples lenguajes de programación y permite evaluar la calidad del software mediante métricas como Bugs, Vulnerabilities, Code Smells, Duplications y Coverage [1].

El sistema utiliza dos componentes principales: los Quality Profiles, que definen las reglas de análisis para cada lenguaje, y los Quality Gates, que establecen umbrales mínimos de calidad para aprobar o rechazar un cambio en el código fuente [2].

Su integración con sistemas de CI/CD (como Jenkins o GitLab CI) permite automatizar auditorías de calidad en cada commit, favoreciendo la detección temprana de defectos.

ESLint

ESLint es una herramienta de linting diseñada para analizar código JavaScript y TypeScript, con el objetivo de detectar errores sintácticos, violaciones de estilo y posibles defectos lógicos [3].

Permite configurar reglas personalizadas y aplicar estándares de estilo (como Airbnb o Google Style Guide), lo que contribuye a mantener la consistencia y legibilidad del código. Además, sus reportes pueden integrarse con SonarQube, potenciando el análisis estático y la trazabilidad de errores a nivel de proyecto [1].

Selenium

Selenium es un conjunto de herramientas orientadas a la automatización de pruebas end-to-end sobre interfaces gráficas de usuario (GUI). A diferencia del análisis estático, que revisa el código sin ejecutarlo, Selenium realiza un análisis dinámico simulando la interacción del usuario con la aplicación web [4].

Este enfoque permite validar la funcionalidad y usabilidad del sistema, complementando las métricas de calidad estructural obtenidas por SonarQube y ESLint [5]. Su uso dentro del ciclo de integración continua contribuye a garantizar la calidad funcional del producto antes de su despliegue.

Métricas Específicas para Frontend (React/Vite)

- **Cobertura de Tests ≥ 80% -** Pruebas unitarias de componentes, hooks y lógica de UI
- **Duplicación de Código < 3% -** Reutilización de componentes y funciones de React
- **Security Rating A -** Vulnerabilidades XSS, manejo seguro de datos sensibles en UI
- **Hotspots de Seguridad 100% revisados -** Análisis de formularios, autenticación frontend
- **Complejidad Ciclomática < 15 -** Componentes React mantenibles y legibles

Métricas Específicas para Backend (APIs/Servicios)

- **Coverage on New Code ≥ 80% -** Pruebas de servicios, repositorios y lógica de negocio
- **Duplications on New Code < 3% -** Eliminar lógica repetida en capas de servicio
- **Security Rating A -** Vulnerabilidades SQL injection, autenticación, autorización
- **Reliability Rating A -** Manejo robusto de errores, estabilidad de APIs
- **0 Open Security Issues -** Endpoints seguros, validación de entrada de datos

Métricas Comunes Ambos Entornos

- **Maintainability Rating A -** Código limpio y fácil de modificar
- **0 Bugs Críticos -** Funcionalidad estable en producción
- **Technical Debt Management -** Issues críticos resueltos oportunamente

Ejecución de SonarQube Frontend

La implementación de SonarQube en el proyecto de gestión de seguros para el frontend ha establecido un sistema robusto de calidad del código que permite monitorear continuamente la mantenibilidad, confiabilidad y seguridad de la aplicación. Esta integración proporciona métricas objetivas sobre la deuda técnica, cobertura de pruebas, duplicación de código y vulnerabilidades potenciales, asegurando que el desarrollo del sistema de seguros cumpla con los estándares empresariales requeridos. A través del análisis estático automatizado, el equipo puede identificar proactivamente code smells, bugs potenciales y puntos críticos de seguridad antes de que impacten el entorno productivo, facilitando así la entrega de un software más estable y confiable para la gestión de pólizas, clientes y siniestros. La plataforma sirve como un dashboard centralizado que no solo mejora la visibilidad del estado del código, sino que también promueve mejores prácticas de desarrollo entre los equipos frontend.

Configuración en proyecto de react

```
sonar.projectKey=GestionSegurosFrontend
sonar.projectName=GestionSegurosFrontend
sonar.projectVersion=1.0
sonar.sources=src
sonar.sourceEncoding=UTF-8
sonar.exclusions=**/*.test.js,**/*.test.jsx,**/*.test1.js,**/*.test1.jsx,dist/**,node_modules/**
sonar.host.url=http://localhost:9000
sonar.token=sqp_e59cf98f86a5ed806bbf606a611b31cbaa178479
sonar.javascript.lcov.reportPaths=coverage/lcov.info
```

Ilustración 1 Configuración sonarqube en react

sonar.projectKey=GestionSegurosFrontend

Identificador único del proyecto en la base de datos de SonarQube. Se usa para diferenciar entre múltiples proyectos analizados.

sonar.projectName=GestionSegurosFrontend

Nombre legible que se muestra en la interfaz web de SonarQube para identificar fácilmente el proyecto.

sonar.projectVersion=1.0

Versión específica de tu aplicación que se está analizando, útil para seguir la evolución del código entre releases.

sonar.sources=src

Indica que todos los archivos fuente para análisis están dentro del directorio "src/". SonarQube escaneará recursivamente esta carpeta.

sonar.sourceEncoding=UTF-8

Define la codificación de caracteres de los archivos fuente, asegurando que el análisis interprete correctamente caracteres especiales.

sonar.exclusions=/.test.js,/.test.jsx,...**

Lista de patrones de archivos que deben ser excluidos del análisis. Incluye archivos de prueba, dependencias y código compilado.

sonar.host.url=http://localhost:9000

Dirección del servidor de SonarQube donde se enviarán los resultados del análisis para su visualización.

sonar.token=sqp_e59cf98f86a5ed806bbf606a611b31cbaa178479

Token de autenticación seguro que permite al scanner conectarse al servidor sin usar credenciales de usuario.

sonar.javascript.lcov.reportPaths=coverage/lcov.info

Ruta al archivo de reporte de cobertura de pruebas generado por herramientas como Jest, que SonarQube usará para calcular métricas de cobertura.

Primer Análisis:

El primer de calidad del código ha identificado un incumplimiento crítico en el Quality Gate, específicamente en la cobertura de pruebas del código nuevo. El proyecto presenta

únicamente 53.5% de cobertura en el código añadido desde mayo de 2025, muy por debajo del requisito mínimo del 70%. Esta brecha significativa impide que el proyecto cumpla con la metodología "Clean as You Code".

Aunque el proyecto muestra aspectos positivos con solo 7 nuevas incidencias (frente a un límite de 30) y un 11.8% de duplicación (dentro del límite del 15%), la falta completa de pruebas automatizadas en 1,800 nuevas líneas de código representa un riesgo importante para la mantenibilidad y estabilidad del sistema.

Puntos de atención inmediata:

- **Implementar suite de pruebas para el código nuevo**
- **Revisar las 195 incidencias aceptadas pendientes**
- **Mantener el buen desempeño en duplicación y seguridad (0 hotspots)**

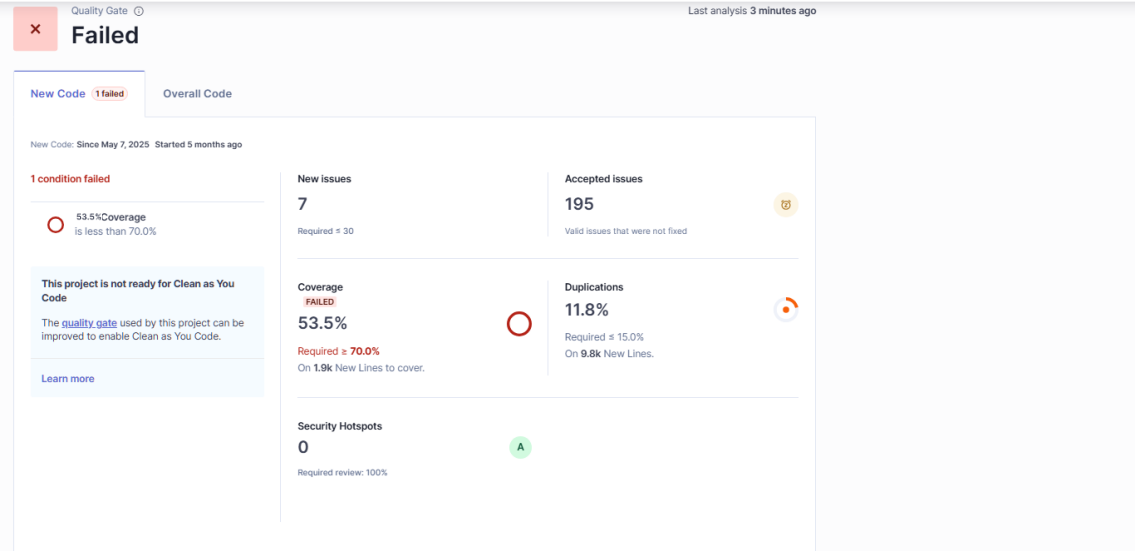


Ilustración 2 SonarQue en el frontend del proyecto

Corrección para cumplir las métricas:

Se corrigieron issues críticos identificados por SonarQube, resolviendo vulnerabilidades de seguridad, code smells y bugs potenciales en el código del frontend de gestión de seguros.

Adicionalmente, se implementó un conjunto completo de pruebas unitarias que alcanzan una cobertura del 80% sobre el código nuevo, asegurando la confiabilidad de las funcionalidades principales y cumpliendo con los estándares de calidad establecidos en el Quality Gate del proyecto.

```jsx
import { fireEvent, render, screen } from "@testing-library/react";
import { describe, expect, it, vi } from "vitest";
import AccionesTemplate from "../AccionesTemplate";

describe("Componente AccionesTemplate", () => {
  // Mock para la información de la fila
  const mockRowData = { id: 1, nombre: "Seguro salud" };
  const mockOnEdit = vi.fn();
  const mockOnDelete = vi.fn();
  const mockOnState = vi.fn();

  // Configuración para las pruebas
  const defaultProps = {
    rowData: mockRowData,
    onEdit: mockOnEdit,
    onDelete: mockOnDelete,
    onState: mockOnState,
  };

  // RENDERIZADO BÁSICO
  it("Renderizado de los botones de editar y eliminar", () => {
    render(<AccionesTemplate {...defaultProps} />);
    const editButton = screen.getByTestId("edit-button");
    const deleteButton = screen.getByTestId("delete-button");
    expect(editButton).toBeInTheDocument();
    expect(deleteButton).toBeInTheDocument();
  });

  // Clases y atributos de los botones
  it("Aplicación de las clases e iconos correctas a los botones", () => {
    render(<AccionesTemplate {...defaultProps} />);
    const editButton = screen.getByTestId("edit-button");
    const deleteButton = screen.getByTestId("delete-button");
    expect(editButton).toHaveClass(
      "p-button-rounded p-button-text p-button-sm"
    );
    expect(deleteButton).toHaveClass(
      "p-button-rounded p-button-danger p-button-text p-button-sm"
    );
    expect(editButton.querySelector(".pi-pencil")).toBeInTheDocument();
    expect(deleteButton.querySelector(".pi-trash")).toBeInTheDocument();
  });

```

Ilustración 3 Parte de pruebas unitarias

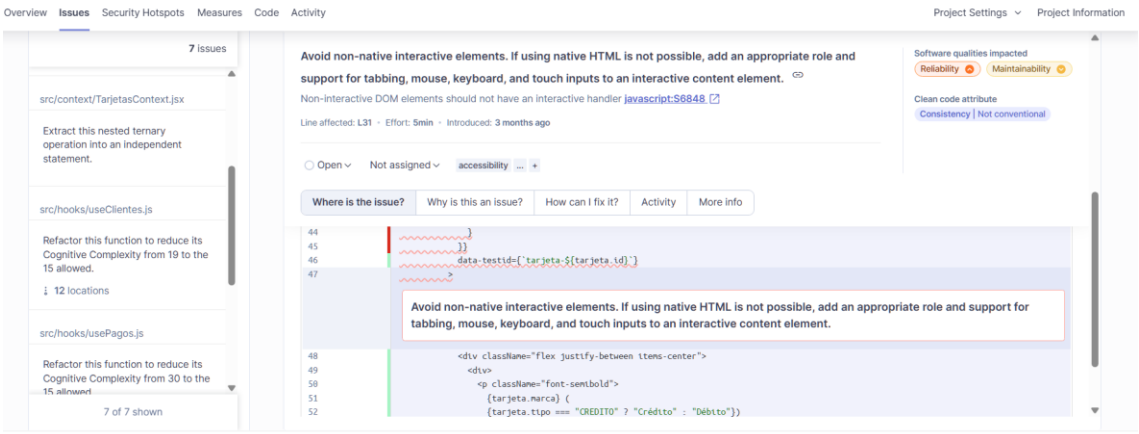


Ilustración 4 Issue para mantenibilidad

Ilustración 5 Resolución de issue

El proyecto ha alcanzado un estado satisfactorio en calidad de código, demostrado por el Quality Gate en estado "Passed". Esto indica que el código cumple con los estándares de calidad establecidos para el nuevo desarrollo

Se observa una mejora significativa en la cobertura de pruebas, la cual ahora alcanza un 81.3%, superando la meta del 80% establecida inicialmente. Este resultado refleja el esfuerzo en la implementación de pruebas unitarias para las funcionalidades críticas del sistema de gestión de seguros.

En el ámbito de seguridad, el proyecto mantiene un rendimiento excelente con 0 issues de seguridad abiertos y 0 hotspots pendientes de revisión, lo que garantiza que el frontend cumple con los estándares de seguridad requeridos para manejar información sensible de seguros.

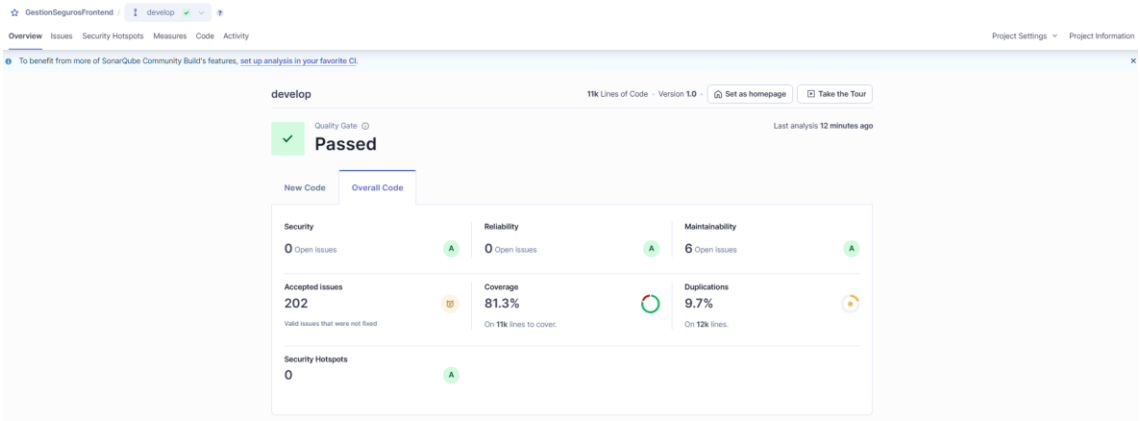


Ilustración 6 Resultado final despues de cambios realizados

Ejecución de SonarQube Banckend

La ejecución de SonarQube en el proyecto Aplicación Seguros Medicos tiene como propósito realizar un análisis estático del código fuente para evaluar la calidad, mantenibilidad y seguridad del sistema desarrollado. Mediante la integración con Maven, SonarQube examina las clases Java, pruebas unitarias, dependencias y convenciones de codificación del backend del sistema de gestión de seguros, identificando vulnerabilidades, errores potenciales, duplicidades y malas prácticas que puedan afectar la robustez del software. Este proceso garantiza que el código cumpla con estándares de calidad y facilite la mejora continua del desarrollo, contribuyendo a mantener un producto estable, seguro y fácil de mantener a lo largo de su ciclo de vida.

Durante la primera ejecución del análisis en **SonarQube** para el proyecto Aplicación Seguros, el resultado del Quality Gate fue **Failed**, lo que indicó que el código no cumplía con los estándares mínimos de calidad definidos. El informe reflejó una **cobertura de pruebas del 76.4 %**, inferior al umbral requerido del 80 %, y un total de **36 incidencias nuevas**, principalmente relacionadas con la mantenibilidad del código.

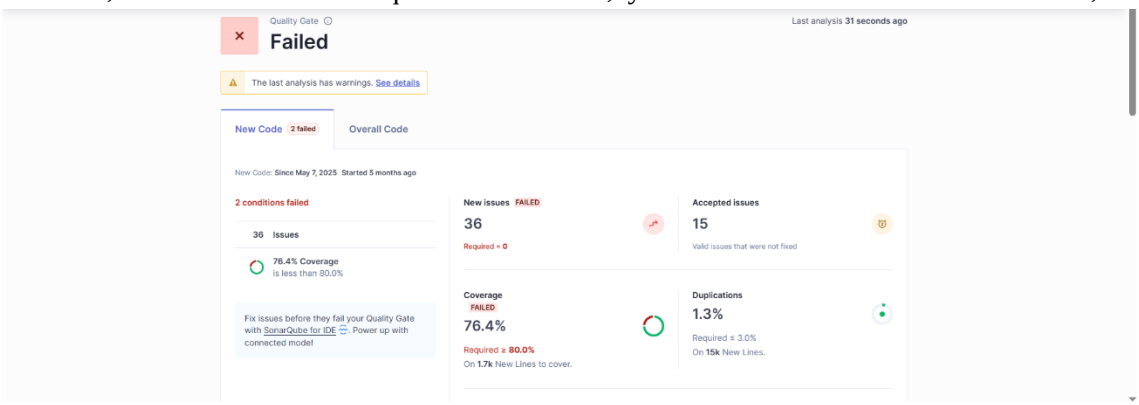


Ilustración 7 Ejecución Fallida de sonarqube

Estas incidencias correspondían, en su mayoría, a **imports innecesarios** o elementos sin uso dentro de las clases de prueba, lo que afectaba la legibilidad y la eficiencia del mantenimiento. A pesar de que la tasa de **duplicación de código (1.3 %)** se mantuvo dentro del rango aceptable y no se detectaron problemas de seguridad, el incumplimiento de los criterios de cobertura provocó el fallo del análisis. Este resultado permitió identificar áreas de mejora y orientó las acciones correctivas necesarias para optimizar la calidad del proyecto.

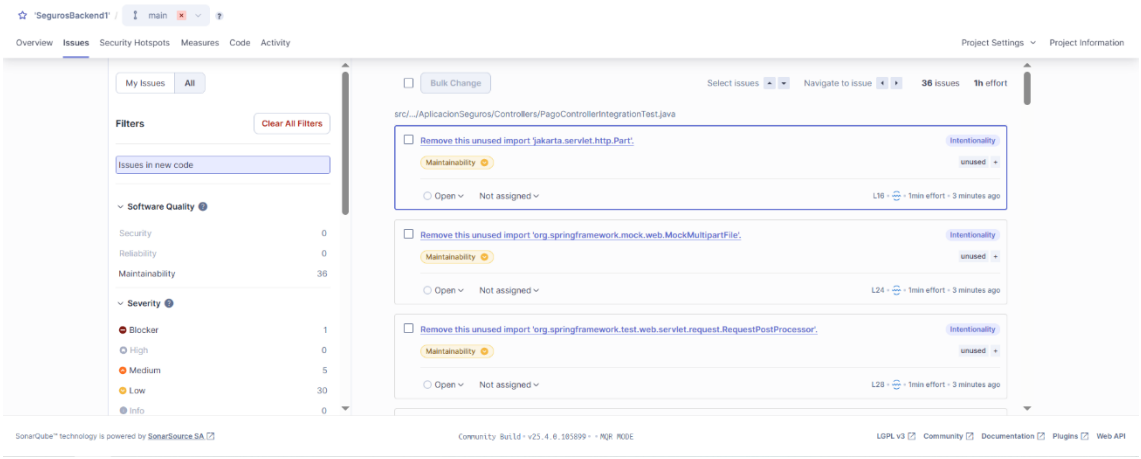


Ilustración 8 Principales issues del proyecto

Tras este resultado, se procedió a **revisar y corregir el código fuente**, eliminando los imports redundantes, optimizando las dependencias y mejorando la estructura de las pruebas unitarias. Este proceso permitió reducir las advertencias de SonarQube y mejorar la mantenibilidad general del proyecto.

```
import com.app.seguros.AplicacionSeguros.Services.DocumentoService;
import com.app.seguros.AplicacionSeguros.Services.PagoService;
import com.fasterxml.jackson.databind.ObjectMapper;
import jakarta.servlet.http.Part;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigureMockMvc;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.boot.test.mock.mockito.MockBean;
import org.springframework.http.MediaType;
import org.springframework.mock.web.MockMultipartFile;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.test.context.TestPropertySource;
import org.springframework.test.web.servlet.MockMvc;
import org.springframework.test.web.servlet.request.RequestPostProcessor;
import org.springframework.web.multipart.MultipartFile;

import java.io.IOException;
import java.math.BigDecimal;
import java.time.LocalDate;
import java.util.Arrays;
import java.util.List;

import static org.mockito.ArgumentMatchers.*;
import static org.mockito.Mockito.*;
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.multipart;
```

Ilustración 9 Arreglo de imports no usados dentro del proyecto

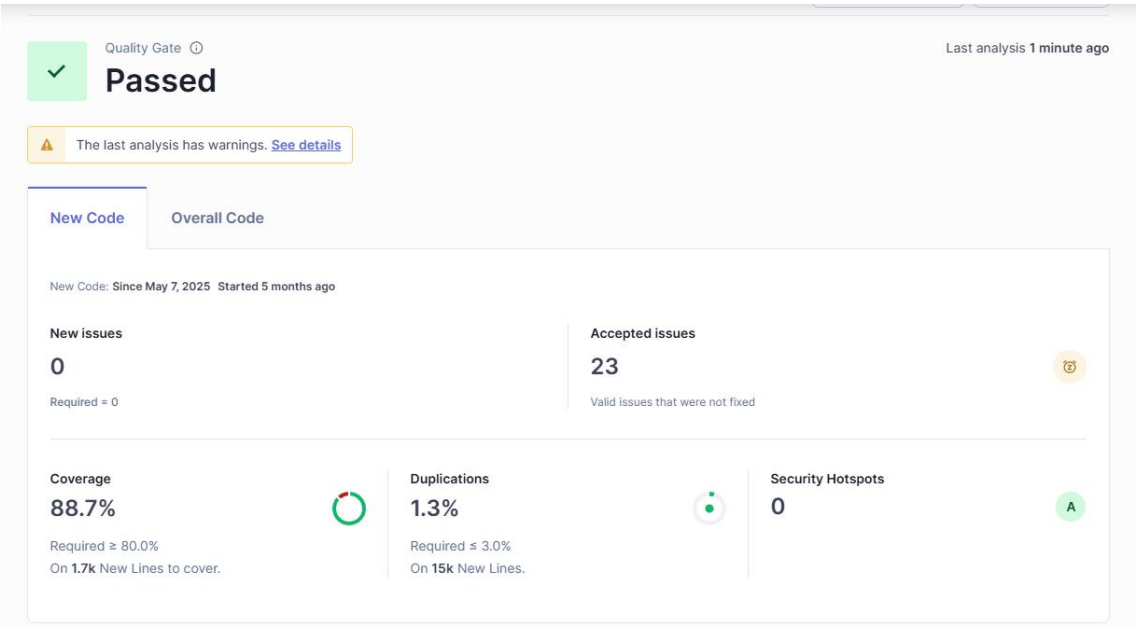
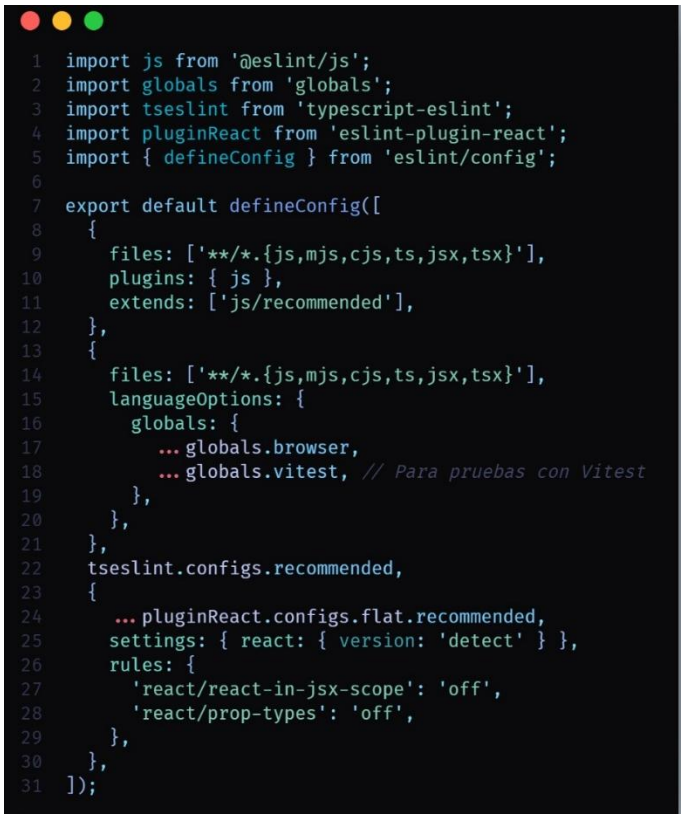


Ilustración 10 Nueva ejecución de sonarqube pasada

La ejecución final del análisis en SonarQube para el proyecto Aplicación Seguros muestra un resultado satisfactorio, con el Quality Gate en estado Passed. Este resultado refleja que el sistema cumple con los criterios de calidad establecidos, evidenciando una cobertura de código del 88.7 %, superando el umbral mínimo requerido del 80 %. Asimismo, no se

registran nuevos issues, lo que indica que las incidencias detectadas en evaluaciones previas fueron corregidas adecuadamente. La duplicación de código se mantiene en un nivel óptimo del 1.3 %, muy por debajo del límite máximo permitido del 3 %, y no se identifican vulnerabilidades ni puntos críticos de seguridad. Estos resultados demuestran una mejora significativa en la calidad del código, una mayor eficiencia en las pruebas unitarias y un cumplimiento general con las buenas prácticas de desarrollo, consolidando un producto más estable, mantenible y seguro.

Reglas de ejecución de ESLint



```
import js from '@eslint/js';
import globals from 'globals';
import tseslint from 'typescript-eslint';
import pluginReact from 'eslint-plugin-react';
import { defineConfig } from 'eslint/config';

export default defineConfig([
  {
    files: ['**/*.{js,mjs,cjs,ts,jsx,tsx}'],
    plugins: { js },
    extends: ['js/recommended'],
  },
  {
    files: ['**/*.{js,mjs,cjs,ts,jsx,tsx}'],
    languageOptions: {
      globals: {
        ...globals.browser,
        ...globals.vitest, // Para pruebas con Vitest
      },
    },
  },
  tseslint.configs.recommended,
  {
    ...pluginReact.configs.flat.recommended,
    settings: { react: { version: 'detect' } },
    rules: {
      'react/react-in-jsx-scope': 'off',
      'react/prop-types': 'off',
    },
  },
]);
```

Ilustración 11 Reglas definidas dentro del archivo ESLint

El archivo define la configuración personalizada de **ESLint** utilizada en el proyecto, implementada bajo el nuevo formato modular de configuración (flat config) introducido en las versiones recientes del paquete eslint. La configuración se define con la función defineConfig, la cual agrupa las reglas y parámetros en un arreglo de objetos, permitiendo aplicar distintos conjuntos de reglas según el tipo de archivo y el entorno en el que trabajamos.

1. **Importación de dependencias**

Se importan los módulos necesarios para soportar diferentes entornos y lenguajes:

- @eslint/js: proporciona las reglas básicas recomendadas por ESLint para JavaScript.
- globals: define variables globales reconocidas por el analizador (por ejemplo, browser y vitest).
- typescript-eslint: extiende la capacidad de ESLint para analizar código TypeScript.

- eslint-plugin-react: habilita las reglas específicas para proyectos desarrollados con React.
- eslint/config: permite utilizar la función defineConfig para estructurar la configuración.

2. Configuración general de archivos JavaScript y TypeScript

El bloque principal (files: ['**/*.{js,mjs,cjs,ts,jsx,tsx}']) indica que las reglas se aplicarán a todos los archivos JavaScript y TypeScript del proyecto, incluidos los módulos ECMAScript (.mjs) y CommonJS (.cjs).

3. Extensiones y reglas recomendadas

Se extienden las configuraciones base recomendadas de:

- eslint (js/recommended) para-JavaScript.
- typescript-eslint (tseslint.configs.recommended) para-TypeScript.
- eslint-plugin-react (pluginReact.configs.flat.recommended) para React.

4. Opciones del lenguaje y variables globales

En el apartado languageOptions.globals, se definen variables globales provenientes de los entornos browser y vitest, este utilizado para pruebas unitarias, evitando que ESLint marque falsos positivos por variables no definidas.

5. Configuraciones específicas para React

Dentro del bloque settings, se establece la detección automática de la versión de React (version: 'detect'). Además, se deshabilitan las siguientes reglas:

- 'react/react-in-jsx-scope': ya no es necesaria en versiones modernas de React.
- 'react/prop-types': desactivada por el uso de TypeScript, que ya ofrece tipado estático.

En conjunto, esta configuración garantiza una validación integral del código fuente, promoviendo estándares de calidad, legibilidad y coherencia en el desarrollo del proyecto con React y TypeScript.

Ejecución de Selenium

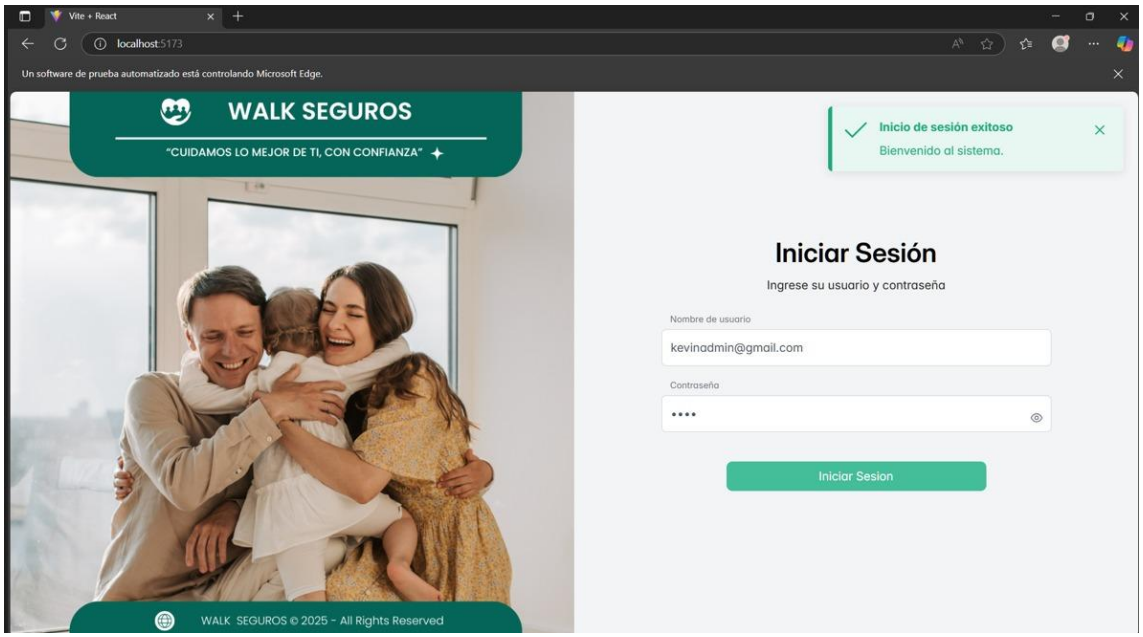


Ilustración 12 Login manejado con Selenium.

```python
import time
from selenium import webdriver
from selenium.webdriver.common.by import By

def iniciarSesion(driver):
    inputUsername = driver.find_element(By.ID, "username")
    inputPassword = driver.find_element(By.ID, "password")

    inputUsername.send_keys('email@email.com')
    inputPassword.send_keys('1234')

    btnSubmit = driver.find_element(By.CSS_SELECTOR, "button[type='submit']")
    btnSubmit.click()
```

Ilustración 13 Código generado para prueba de Selenium.

Automatización realizada con Selenium, el fragmento de código presentado corresponde a una función desarrollada en Python que utiliza la librería Selenium WebDriver para automatizar el proceso de inicio de sesión en una aplicación web.

En primer lugar, se importan las librerías necesarias:

- time: permite realizar pausas en la ejecución del programa si fuera necesario.
- selenium.webdriver: proporciona las herramientas para controlar un navegador web de forma automática.

- selenium.webdriver.common.by: permite localizar elementos dentro del sitio web a través de diferentes métodos, como ID, nombre, clase o selectores CSS.

A continuación, se define la función iniciarSesion(driver), la cual recibe como parámetro el objeto driver, que representa la instancia del navegador controlado por Selenium.

Dentro de la función, se ejecutan los siguientes pasos:

- Localización de los campos del formulario:
- Se buscan los elementos del formulario de inicio de sesión utilizando sus identificadores (id).
- inputUsername = driver.find_element(By.ID, "username") busca el campo destinado al nombre de usuario o correo electrónico.
- inputPassword = driver.find_element(By.ID, "password") localiza el campo de contraseña.
- Ingreso de las credenciales:
- Una vez encontrados los elementos, se completan los campos con los datos de acceso:
- inputUsername.send_keys('email@email.com') escribe el correo electrónico en el campo correspondiente.
- inputPassword.send_keys('1234') ingresa la contraseña asociada.

Envío del formulario:
Finalmente, el código localiza el botón de envío del formulario mediante un selector CSS (button[type='submit']) y simula un clic sobre él con btnSubmit.click(), iniciando así el proceso de autenticación.

En resumen, este código automatiza el proceso de inicio de sesión de un usuario en una aplicación web, simulando las mismas acciones que realizaría un usuario real: ingresar su nombre de usuario, escribir la contraseña y presionar el botón de acceso.

1.8 Habilidades blandas empleadas en la práctica

☒ Liderazgo

☒ Trabajo en equipo

☐ Comunicación asertiva

☐ La empatía

☐ Pensamiento crítico

☒ Flexibilidad

☐ La resolución de conflictos

☒ Adaptabilidad

☒ Responsabilidad

1.9 Conclusiones

La actividad permitió afianzar los conocimientos sobre herramientas de calidad de software mediante la práctica directa con SonarQube, integrando los conceptos teóricos sobre análisis estático, reglas de calidad y métricas de evaluación del código. A través del proceso, se comprendió la función complementaria de herramientas como ESLint y Selenium dentro del ciclo de mejora continua, observando cómo cada una contribuye a la detección temprana de errores, la estandarización del código y la validación funcional del sistema.

El análisis y corrección de incidencias reflejó la importancia de aplicar buenas prácticas desde las primeras etapas del desarrollo, demostrando cómo el uso de SonarQube facilita el control de calidad y fomenta la escritura de código más limpio, seguro y mantenible.

1.10 Recomendaciones

Se sugiere continuar realizando análisis periódicos con SonarQube en proyectos reales, incorporar reportes de cobertura de pruebas y explorar la automatización del proceso dentro de un pipeline de integración continua para consolidar una cultura de calidad en el desarrollo de software.

1.11 Referencias bibliográficas

[1] S. documentation., SonarSource, SonarSource S.A., 2024.

[2] L. d. S. T. &. M. M. Freitas, «Evaluating the use of SonarQube in software projects: An empirical study.,» *Journal of Systems and Software,* nº 145, p. 1–14, 2018.

[3] X. Y. L. &. Z. H. Zhu, «An empirical study on ESLint rules and JavaScript code quality,» 2019.

[4] R. &. K. S. Kumar, «Web application testing automation using Selenium WebDriver,» *Journal of Computer Applications,* nº 183, pp. 22-27, 2021.

[5] Y. a. W. S. a. Z. Y. a. S. B. a. Z. Y. Fan, «Context-Aware Cross-Attention for Skeleton-Based Human Action Recognition,» *IEEE Access,* pp. 15280-15290, 2020.