**Evaluación Módulo 4: Automatización de Pruebas en una Plataforma de Salud**

**🎯 Contexto:**

La empresa **HealthTrack** ha desarrollado una plataforma web para el monitoreo del peso de los usuarios. La aplicación permite a los usuarios registrarse y actualizar su peso cada 48 horas. Sin embargo, la plataforma tiene un error crítico: cada vez que un usuario actualiza su peso, el sistema le resta automáticamente 1 kg en lugar de registrar el valor ingresado.

El problema surge porque no se han implementado pruebas unitarias, de integración, de regresión ni de rendimiento. Además, la empresa no tiene un pipeline de CI/CD que asegure la validación automática del código antes de su despliegue.

Como especialistas en automatización de pruebas, los estudiantes deberán evaluar el estado actual de la plataforma y proponer soluciones utilizando estrategias de pruebas unitarias, de integración, funcionales y de rendimiento. También deberán estructurar un pipeline de CI/CD para asegurar que estas pruebas se ejecuten automáticamente.

**1. Análisis del Estado Actual de la Plataforma**

**El error en la lógica del código**

El codigo en el método actualizarPeso es incorrecta:

java

CopiarEditar

public void actualizarPeso(double nuevoPeso) {

this.peso -= 1;

}

En lugar de almacenar el nuevo peso ingresado, el sistema **resta siempre 1 kg**, lo que genera inconsistencias en la data histórica del usuario.

**Impacto del error en la experiencia del usuario**

**Impacto en la salud**: Los usuarios reciben un valor de peso incorrecto cada vez que actualizan su peso.

**Falta de confianza**: La plataforma pierde credibilidad al registrar datos erróneos.

**Riesgos regulatorios**: Una app de salud debe cumplir estándares de precisión. Datos incorrectos pueden derivar en sanciones o demandas.

**Falta de procesos de validación y pruebas**

**No hay pruebas unitarias**: Errores básicos en la lógica pasan inadvertidos.

**No existen pruebas de integración ni regresión**: Cada cambio es riesgoso.

**No hay pruebas de rendimiento**: Desconocemos cómo escala la app con muchos usuarios.

**Sin CI/CD**: El código se despliega sin ninguna validación automatizada.

**2. Diseño y Desarrollo de Pruebas Automatizadas**

**Pruebas Unitarias (JUnit)**

**Objetivo:** Validar la lógica del método actualizarPeso.

**Corrección en el código**

Debes corregir el método así:

java

public void actualizarPeso(double nuevoPeso) {

this.peso = nuevoPeso;

}

**Test en JUnit**

src/test/java/com/healthtrack/UsuarioTest.java

java

import static org.junit.jupiter.api.Assertions.\*;

import org.junit.jupiter.api.Test;

public class UsuarioTest {

@Test

public void testActualizarPeso() {

Usuario usuario = new Usuario("Pedro", 70.0);

usuario.actualizarPeso(72.5);

assertEquals(72.5, usuario.getPeso(), 0.001);

}

@Test

public void testPesoInicial() {

Usuario usuario = new Usuario("Ana", 65.0);

assertEquals(65.0, usuario.getPeso(), 0.001);

}

}

**Pruebas Funcionales (Selenium)**

**Objetivo:** Simular un usuario que:

1. Ingresa a la plataforma.
2. Actualiza su peso.
3. Visualiza el peso actualizado.

Ejemplo con Selenium (Java):

java

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.chrome.ChromeDriver;

import org.junit.jupiter.api.\*;

public class HealthTrackFunctionalTest {

private WebDriver driver;

@BeforeEach

public void setUp() {

System.setProperty("webdriver.chrome.driver", "/path/to/chromedriver");

driver = new ChromeDriver();

}

@Test

public void testActualizarPeso() {

driver.get("http://localhost:8080/login");

driver.findElement(By.id("username")).sendKeys("testuser");

driver.findElement(By.id("password")).sendKeys("1234");

driver.findElement(By.id("loginButton")).click();

driver.findElement(By.id("pesoInput")).clear();

driver.findElement(By.id("pesoInput")).sendKeys("75");

driver.findElement(By.id("btnActualizar")).click();

WebElement resultado = driver.findElement(By.id("pesoActual"));

Assertions.assertEquals("75 kg", resultado.getText());

}

@AfterEach

public void tearDown() {

driver.quit();

}

}

**Pruebas de Regresión**

**Objetivo:** Asegurar que cambios futuros no rompan la lógica corregida.

**Estrategia:**

* **Crear suite de regresión**:
  + Tests unitarios → lógica interna.
  + Tests funcionales → UI.
* Etiquetas JUnit:

java

@Tag("regression")

* Ejecutar suite completa en cada build.

**Pruebas de Rendimiento (JMeter)**

**Objetivo:** Medir tiempo de respuesta al actualizar el peso para múltiples usuarios simultáneos.

**Pasos:**

* Configurar Thread Group:
  + 50 usuarios simultáneos.
  + Ramp-up de 10 segundos.
  + 5 iteraciones.
* Configurar sampler HTTP:

POST /actualizarPeso

Body: {"idUsuario": 123, "peso": 72.5}

* Listener → Summary Report:
  + Latencia promedio < 500 ms.
  + Throughput mínimo: 20 req/s.
  + Error rate < 1%.

**Ejecución en consola (ideal para CI/CD):**

bash

jmeter -n -t tests/actualizarPeso.jmx -l results/resultados.jtl -e -o results/html

**Automatización del Proceso de Pruebas con CI/CD**

**Pipeline en GitHub Actions**

Archivo: .github/workflows/ci.yml

yaml

name: HealthTrack CI Pipeline

on:

push:

branches: [ "main" ]

pull\_request:

branches: [ "main" ]

jobs:

build:

runs-on: ubuntu-latest

steps:

- name: Checkout Code

uses: actions/checkout@v3

- name: Set up JDK 17

uses: actions/setup-java@v3

with:

java-version: '17'

- name: Build and Run Unit Tests

run: mvn clean test

- name: Run Selenium Tests

run: mvn verify -Pselenium-tests

- name: Run JMeter Tests

run: |

jmeter -n -t tests/actualizarPeso.jmx -l results/resultados.jtl -e -o results/html

- name: Run SonarQube Scan

uses: sonarsource/sonarqube-scan-action@v2

env:

SONAR\_TOKEN: ${{ secrets.SONAR\_TOKEN }}

Beneficios del pipeline:

* Ejecuta unit tests (JUnit).
* Ejecuta Selenium.
* Ejecuta JMeter.
* Publica reportes de cobertura (SonarQube).
* Bloquea despliegues si fallan las pruebas.

**Conclusiones**

**Problema detectado:**

* Lógica incorrecta en actualización de peso.

**Solución técnica:**

* Corregir método actualizarPeso.
* Implementar pruebas unitarias, funcionales y de rendimiento.
* Integrar pipeline CI/CD.

**Beneficios esperados:**

* Evitar errores en producción.
* Incrementar confianza en la plataforma.
* Mejorar la experiencia del usuario y cumplir requisitos regulatorios.