HealthTrack

Informe de resultados

Automatización de Pruebas

Contenidos

1
2
3

- Análisis del estado actual de la plataforma
- Diseño y desarrollo de pruebas automatizadas
- Automatización del proceso de pruebas con CI/CD

Análisis actual de la plataforma

Descripción del error

La *plataforma HealthTrack presenta un error lógico en la funcionalidad de actualización del peso del usuario. El método* ***actualizarPeso(double nuevoPeso)*** resta 1 kilogramo al peso actual en lugar de asignar el nuevo valor ingresado, generando un resultado incorrecto cada vez que el usuario actualiza su peso.

```
public void actualizarPeso(double nuevoPeso){  no usages
    // ERROR: En lugar de asignar el nuevo peso, se está restando un 1Kg.
    this.peso -= 1;
}
```

Este método debería asignar directamente el nuevo peso ingresado por el usuario.

```
public void actualizarPeso(double nuevoPeso){
    this.peso = nuevoPeso;
}
```

Impacto del error

Experiencia de usuario

Cada vez que el usuario actualiza su peso, el valor registrado es incorrecto (reduce 1 Kg), lo que genera desconfianza y pérdida de fiabilidad

Salud y seguimiento médico

Dado que la plataforma está orientada a la salud, un dato mal registrado podría llevar a interpretaciones erróneas en diagnósticos o tratamientos

Despliegue sin pruebas

El error llegó a producción debido a la ausencia total de pruebas automatizadas (unitarias, de integración, funcionales y de regresión), y a la falta de un pipeline CI/CD que verifique la lógica antes de desplegar

Procesos de validación y pruebas en el desarrollo

Actualmente, no existen pruebas automatizadas ni procesos de validación previos al despliegue. Esto significa que cualquier error en el código puede llegar directamente a producción, lo que compromete la estabilidad y confiabilidad del sistema.

Esta situación pone en evidencia la necesidad de incorporar un pipeline de CI/CD con ejecución automática de pruebas y validación continua.

Diseño y Desarrollo de Pruebas Automatizadas

Pruebas Unitarias (Junit 5)

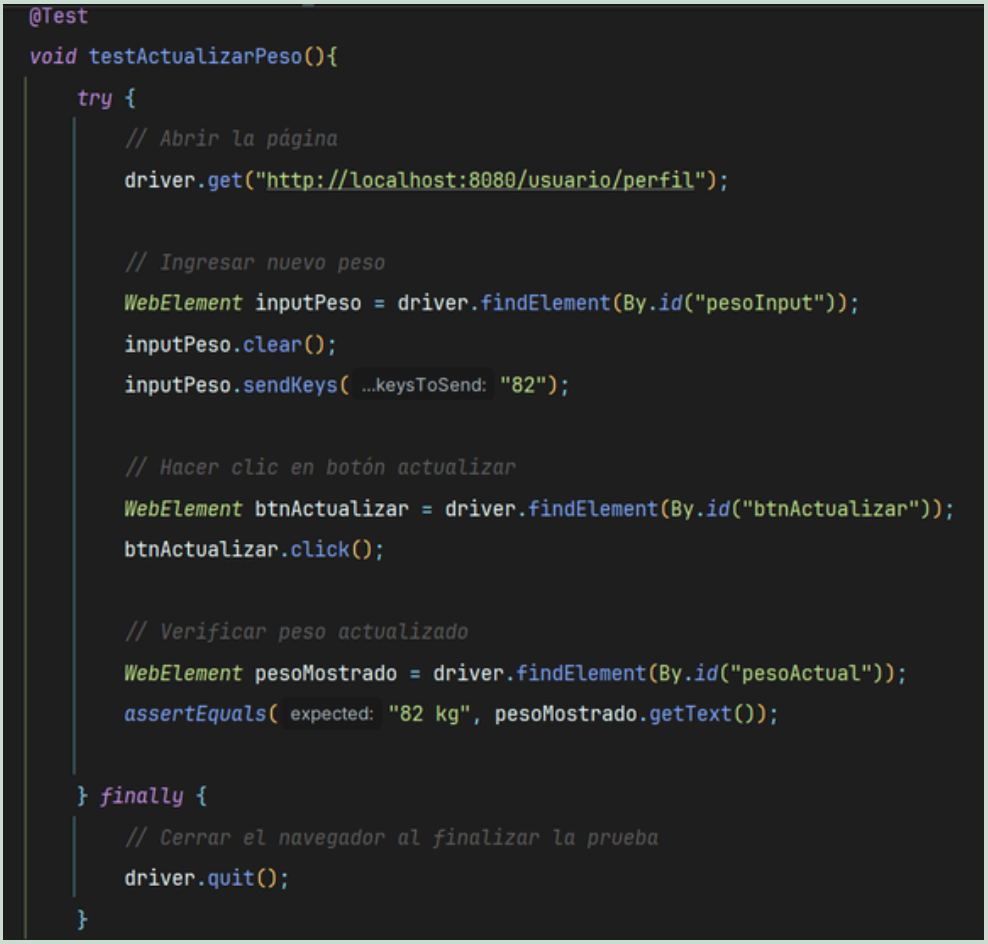
Se creó una clase ***UsuarioTest.java*** con pruebas que verifican la lógica del método ***actualizarPeso()*** , incluyendo casos de aumento o disminuición del peso

```java
class UsuarioTest {

    private Usuario usuario;  5 usages

    @BeforeEach
    void setUp(){
        usuario = new Usuario( nombre: "juan",  peso: 75.5);
    }

    @Test
    @DisplayName("El usuario aumenta su peso correctamente")
    void actualizarPeso_aumentoDePeso() {
        usuario.actualizarPeso( nuevoPeso: 78);
        assertEquals( expected: 78, usuario.getPeso());
    }

    @Test
    @DisplayName("El usuario disminuye su peso correctamente")
    void actualizarPeso_disminucionDePeso() {
        usuario.actualizarPeso( nuevoPeso: 70);
        assertEquals( expected: 70, usuario.getPeso());
    }
}
```

Pruebas Funcionales (Selenium)

Simulación del flujo de usuario desde el navegador:

- Registro de nuevo usuario
- Ingreso de nuevo peso
- Verificación en la interfaz de que el valor se muestra correctamente



```java
@Test
void testActualizarPeso(){
    try {
        // Abrir la página
        driver.get("http://localhost:8080/usuario/perfil");

        // Ingresar nuevo peso
        WebElement inputPeso = driver.findElement(By.id("pesoInput"));
        inputPeso.clear();
        inputPeso.sendKeys( ...keysToSend: "82");

        // Hacer clic en botón actualizar
        WebElement btnActualizar = driver.findElement(By.id("btnActualizar"));
        btnActualizar.click();

        // Verificar peso actualizado
        WebElement pesoMostrado = driver.findElement(By.id("pesoActual"));
        assertEquals( expected: "82 kg", pesoMostrado.getText());

    } finally {
        // Cerrar el navegador al finalizar la prueba
        driver.quit();
    }
```

Pruebas de Regresión

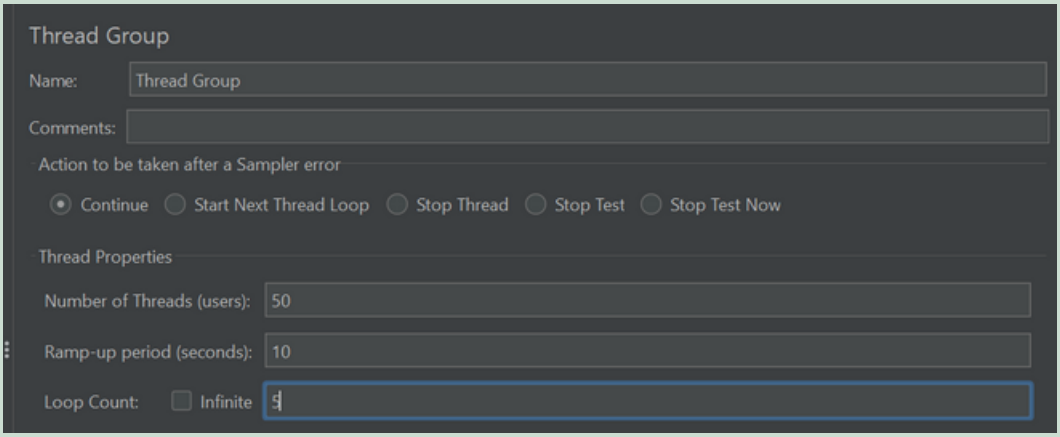
Las pruebas de regresión se diseñan para detectar si una funcionalidad previamente correcta deja de funcionar tras cambio en el código,

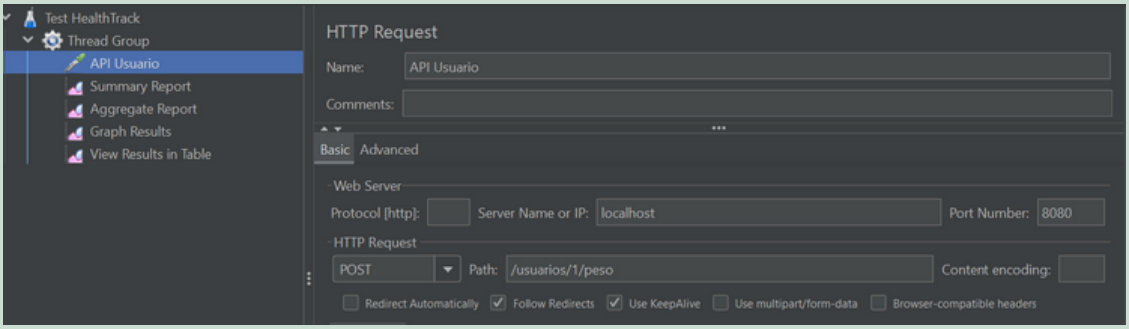
- Las pruebas unitarias de ***acutalizarPeso()*** se ejecutan automáticamente en cada push.
- Se usan anotaciones como ***@BeforeEach*** para inicializar datos consistentes.
- Se garantiza que los atributos nombre y peso no cambien accidentalmente.
- Se incluye validación para verificar que ***mostrarInformacion()*** no altere el estado del objeto.
- A futuro, automatizar flujos funcionales con Selenium en interfaz web.

Pruebas de Rendimiento (JMeter)

Se diseño un script en JMeter para generara 250 request actualizando su peso.

- Number of Threads: 50 usuarios concurrentes
- Ramp-up Period: Los usuarios se conectan cada 10 seg.
- Loop Count: Cada usuario repite la solicitud 5 veces





- **Summary Report**: Da métricas promedio, mínimo, máximo y tasas de errores.
- **Aggregate Report**: Agrupa datos por request y da tiempos promedios, desviación y throughput.
- **Graph Results**: Representación visual del tiempo de respuesta vs números de usuarios.
- **View Results in Table**: Muestra cada solicitud con su tiempo y respuesta.

Automatización del Proceso de Pruebas con CI/CD

- **Pipeline en GitHub Actions**

```yaml
name: CI - HealthTrack

on:
  push:
    branches: [ main ]

jobs:
  build-test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3

      - name: Instalar Java
        uses: actions/setup-java@v3
        with:
          distribution: 'temurin'
          java-version: '21'

      - name: Compilar proyecto
        run: mvn clean install -DskipTests

      - name: Ejecutar pruebas unitarias
        run: mvn test

      - name: Generar reporte de cobertura JaCoCo
        run: mvn jacoco:report

      - name: Subir reporte JaCoCo
        uses: actions/upload-artifact@v4
        with:
          name: cobertura
          path: target/site/jacoco/index.html
```

Configuración de resultados

- Se usa JaCoCo para generar reporte de cobertura.
- Se almacena como artefactos y se accede vía GitHub Actions.
- Se visualiza cobertura por clase y método.

Alertas de fallos

- El pipeline marca errores en rojo si alguna prueba falla.
- Se puede integrar con Slack, Discord o email para alertar automáticamente.
- Mecansimo de validación antes del merge: se bloquea si el pipeline falla.

Validación y calidad del código

Para asegurar la calidad y mantenibilidad del código de la plataforma HealthTrack, se utilizó SonarQube como herramienta de análisis estático. SonarQube permitió detectar problemas como: