Testing

Ingeniería de software I

Kevin David Rodriguez Riveros

Frank Sebastian Pardo Amaya

Jorge Andrés Torres Leal

Departamento de Ingeniería de Sistemas e Industrial

Universidad Nacional de Colombia - Sede Bogotá Facultad de Ingeniería

27 de Febrero de 2025

Introducción

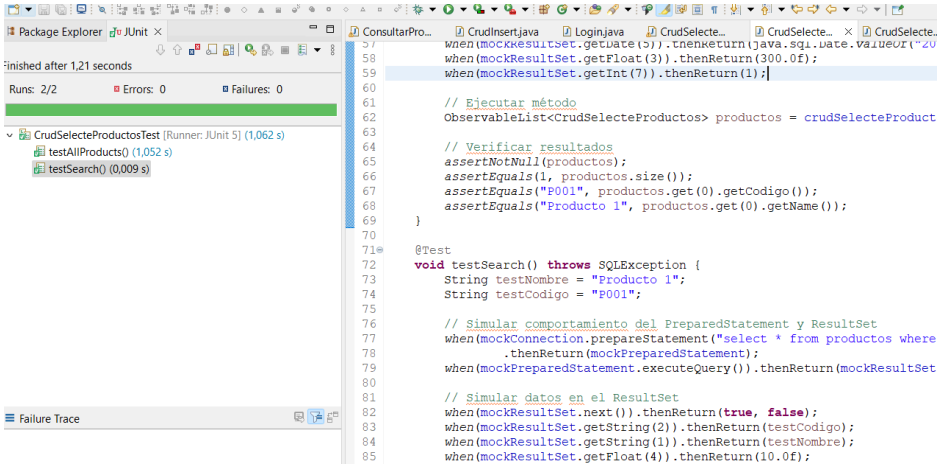
StockEase es un sistema de gestión de inventarios diseñado para pequeñas y medianas empresas con el objetivo de mejorar el control de productos, reducir errores y facilitar la administración de inventarios. La idea surge como una solución a la dependencia de hojas de cálculo y registros manuales, que suelen generar errores, pérdida de información y falta de precisión.

Resumen de los test:

1. **Test Jorge Andrés Torres Leal**
    a. Test Unitario
    b. En los tests unitarios que implementamos, estamos testeando el Modelo, específicamente la clase CrudSelecteProductos, que forma parte de la capa de acceso a datos (DAO - Data Access Object).
    c. Se utilizó JUnit y Mockito

d.

```java
package database;

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

@ExtendWith(MockitoExtension.class)
class CrudSelecteProductosTest {

    @Mock
    private Connection mockConnection;

    @Mock
    private Statement mockStatement;

    @Mock
    private PreparedStatement mockPreparedStatement;

    @Mock
    private ResultSet mockResultSet;

    @InjectMocks
    private CrudSelecteProductos crudSelecteProductos;

    @BeforeEach
    void setUp() throws SQLException {
        MockitoAnnotations.openMocks(this);
        crudSelecteProductos = new CrudSelecteProductos(mockConnection);
    }

    @Test
    void testAllProducts() throws SQLException {
        // Simular comportamiento del Statement y ResultSet
        when(mockConnection.createStatement()).thenReturn(mockStatement);
        when(mockStatement.executeQuery("select * from productos;")).thenReturn(mockResultSet);

        // Simular datos en el ResultSet
        when(mockResultSet.next()).thenReturn(true, false);
        when(mockResultSet.getString(2)).thenReturn("P001");
        when(mockResultSet.getString(1)).thenReturn("Producto 1");
        when(mockResultSet.getDate(5)).thenReturn(java.sql.Date.valueOf("2024-12-31"));
        when(mockResultSet.getFloat(3)).thenReturn(300.0f);
        when(mockResultSet.getInt(7)).thenReturn(1);

        // Ejecutar método
        ObservableList<CrudSelecteProductos> productos = crudSelecteProductos.allProducts();

        // Verificar resultados
        assertNotNull(productos);
        assertEquals(1, productos.size());
        assertEquals("P001", productos.get(0).getCodigo());
        assertEquals("Producto 1", productos.get(0).getName());
    }

    @Test
    void testSearch() throws SQLException {
        String testNombre = "Producto 1";
        String testCodigo = "P001";

        // Simular comportamiento del PreparedStatement y ResultSet
        when(mockConnection.prepareStatement("select * from productos where name=? or codigo=?;"))
                .thenReturn(mockPreparedStatement);
        when(mockPreparedStatement.executeQuery()).thenReturn(mockResultSet);

        // Simular datos en el ResultSet
        when(mockResultSet.next()).thenReturn(true, false);
        when(mockResultSet.getString(2)).thenReturn(testCodigo);
        when(mockResultSet.getString(1)).thenReturn(testNombre);
        when(mockResultSet.getFloat(4)).thenReturn(10.0f);
        when(mockResultSet.getFloat(6)).thenReturn(500.0f);
        when(mockResultSet.getDate(5)).thenReturn(java.sql.Date.valueOf("2024-12-31"));
        when(mockResultSet.getFloat(3)).thenReturn(300.0f);
        when(mockResultSet.getInt(7)).thenReturn(1);

        // Ejecutar método
        ObservableList<CrudSelecteProductos> productos = crudSelecteProductos.search(testNombre, testCodigo);

        // Verificar resultados
        assertNotNull(productos);
        assertEquals(1, productos.size());
        assertEquals(testCodigo, productos.get(0).getCodigo());
        assertEquals(testNombre, productos.get(0).getName());
    }
```

e. 

2. Test Frank Sebastian Pardo Amaya

a. Test unitario

b. Se testea la funcionalidad para insertar usuarios sin hacer uso de la UI, además se realiza una conexión real con la base de datos.

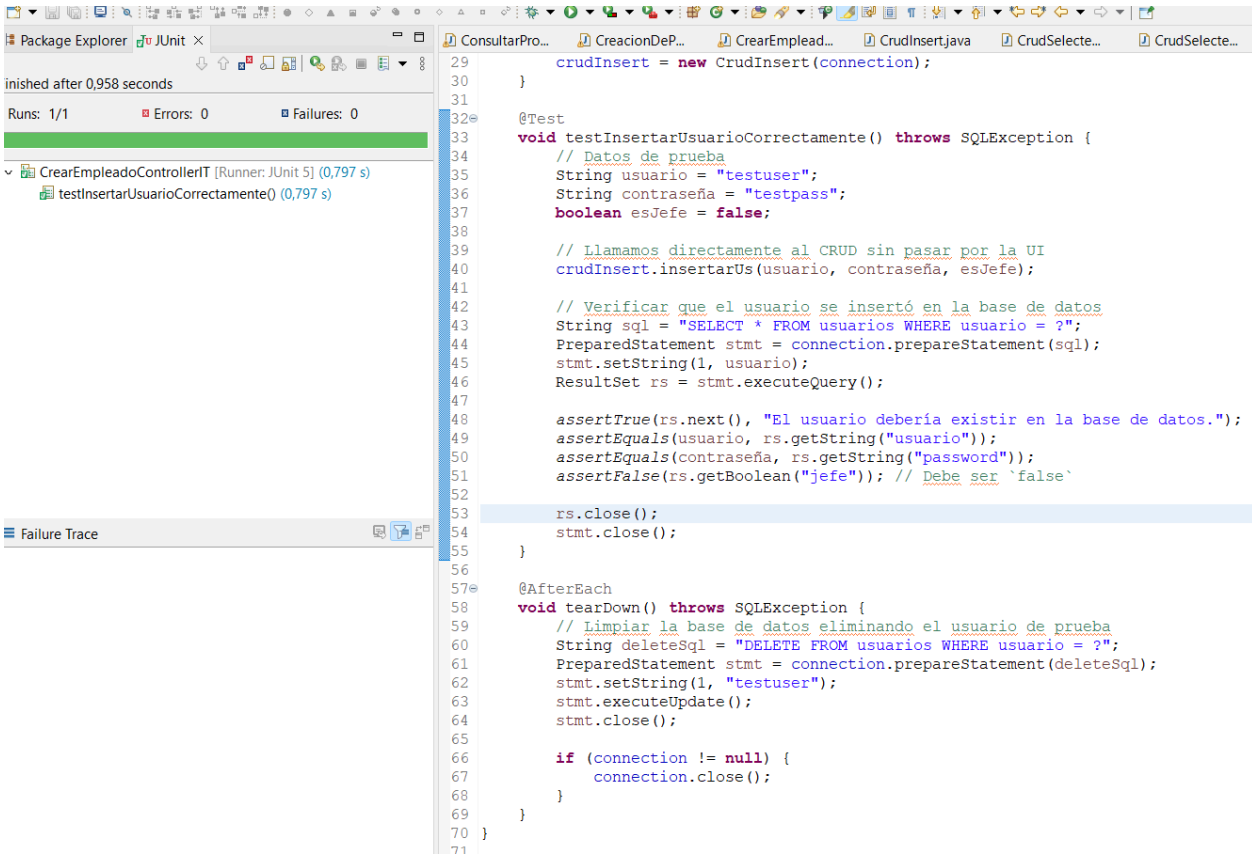
c. Se utiliza JUnit

d.

```java
package Sistema_administrativo_de_tienda.controlador;

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

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

import database.CrudInsert;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

class CrearEmpleadoControllerIT {

    private Connection connection;
    private CrudInsert crudInsert;

    @BeforeEach
    void setUp() throws SQLException {
        // Conexión a la base de datos de prueba
        String url = "jdbc:mysql://localhost:3306/pruebaingesoft?serverTimezone=UTC"; // BD de prueba
        String user = "root"; // Usuario de la BD
        String password = "957846Oso"; // Contraseña de la BD

        connection = DriverManager.getConnection(url, user, password);
        crudInsert = new CrudInsert(connection);
    }

    @Test
    void testInsertarUsuarioCorrectamente() throws SQLException {
        // Datos de prueba
        String usuario = "testuser";
        String contraseña = "testpass";
        boolean esJefe = false;

        // Llamamos directamente al CRUD sin pasar por la UI
        crudInsert.insertarUs(usuario, contraseña, esJefe);

        // Verificar que el usuario se insertó en la base de datos
        String sql = "SELECT * FROM usuarios WHERE usuario = ?";
```

```
29          crudInsert = new CrudInsert(connection);
30      }
31
32      @Test
33      void testInsertarUsuarioCorrectamente() throws SQLException {
34          // Datos de prueba
35          String usuario = "testuser";
36          String contraseña = "testpass";
37          boolean esJefe = false;
38
39          // Llamamos directamente al CRUD sin pasar por la UI
40          crudInsert.insertarUs(usuario, contraseña, esJefe);
41
42          // Verificar que el usuario se insertó en la base de datos
43          String sql = "SELECT * FROM usuarios WHERE usuario = ?";
44          PreparedStatement stmt = connection.prepareStatement(sql);
45          stmt.setString(1, usuario);
46          ResultSet rs = stmt.executeQuery();
47
48          assertTrue(rs.next(), "El usuario debería existir en la base de datos.");
49          assertEquals(usuario, rs.getString("usuario"));
50          assertEquals(contraseña, rs.getString("password"));
51          assertFalse(rs.getBoolean("jefe")); // Debe ser `false`
52
53          rs.close();
54          stmt.close();
55      }
56
57      @AfterEach
58      void tearDown() throws SQLException {
59          // Limpiar la base de datos eliminando el usuario de prueba
60          String deleteSql = "DELETE FROM usuarios WHERE usuario = ?";
61          PreparedStatement stmt = connection.prepareStatement(deleteSql);
62          stmt.setString(1, "testuser");
63          stmt.executeUpdate();
64          stmt.close();
65
66          if (connection != null) {
67              connection.close();
68          }
69      }
70  }
71
```

e.



3. **Test Kevin David Rodriguez Riveros**
    a. Test Unitario
    b. Se testea el módulo de actualizaciones para varias funcionalidades como actualizar el estado o contraseña de un empleado, actualización de precios de productos, entre otros.
    c. Se utilizó JUnit

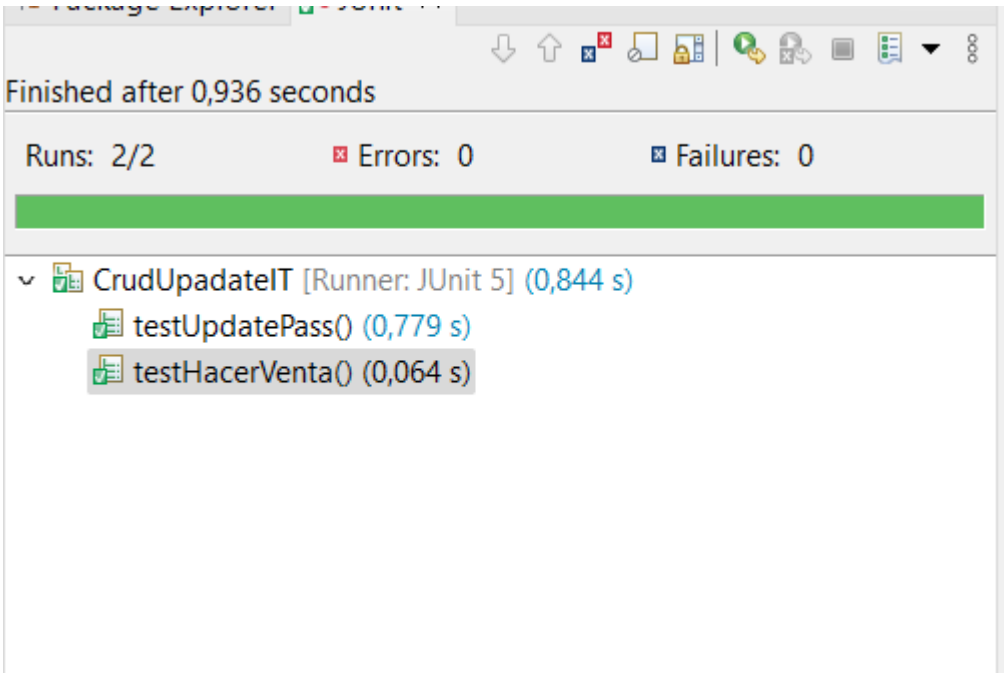
d.

```java
package database;

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

import java.sql.*;
import java.time.LocalDate;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import org.junit.jupiter.api.*;

class CrudUpadateIT {

    private Connection connection;
    private CrudUpadate crudUpadate;

    @BeforeEach
    void setUp() throws SQLException {
        // Conexión a la base de datos de prueba
        String url = "jdbc:mysql://localhost:3306/pruebaingesoft?serverTimezone=UTC";
        String user = "root";
        String password = "957846Oso";

        connection = DriverManager.getConnection(url, user, password);
        crudUpadate = new CrudUpadate(connection);

        // Insertar un usuario de prueba
        String insertUser = "INSERT INTO usuarios (usuario, password, jefe) VALUES (?, ?, ?)";
        PreparedStatement stmtUser = connection.prepareStatement(insertUser);
        stmtUser.setString(1, "testuser");
        stmtUser.setString(2, "oldpass");
        stmtUser.setBoolean(3, false);
        stmtUser.executeUpdate();
        stmtUser.close();

        // Insertar un producto de prueba (corregido)
        String insertProduct = "INSERT INTO productos (codigo, name, precioCompra, precio, cantidad, vencimi
        PreparedStatement stmtProduct = connection.prepareStatement(insertProduct);
        stmtProduct.setString(1, "P001");
        stmtProduct.setString(2, "Producto Test");
        stmtProduct.setFloat(3, 50.0f);
        stmtProduct.setFloat(4, 100.0f); // Corregido
        stmtProduct.setFloat(5, 20.0f); // Corregido
        stmtProduct.setDate(6, Date.valueOf(LocalDate.now().plusDays(30)));
```

```
        stmtProduct.setFloat(4, 100.0f); // Corregido
        stmtProduct.setFloat(5, 20.0f); // Corregido
        stmtProduct.setDate(6, Date.valueOf(LocalDate.now().plusDays(30)));
        stmtProduct.setInt(7, 1);
        stmtProduct.executeUpdate();
        stmtProduct.close();
    }

    @Test
    void testUpdatePass() throws SQLException {
        crudUpadate.updatePass("testuser", "newpass");

        String sql = "SELECT password FROM usuarios WHERE usuario = ?";
        PreparedStatement stmt = connection.prepareStatement(sql);
        stmt.setString(1, "testuser");
        ResultSet rs = stmt.executeQuery();

        assertTrue(rs.next(), "El usuario debería existir.");
        assertEquals("newpass", rs.getString("password"), "La contraseña debería haberse actualizado.");

        rs.close();
        stmt.close();
    }

    @Test
    void testHacerVenta() {
        ObservableList<CrudUpadate> venta = crudUpadate.hacerVenta("P001", 1, 2.0f, "Venta test");

        assertNotNull(venta);
        assertEquals(1, venta.size());
        assertEquals("Producto Test", venta.get(0).getName());
        assertEquals(200.0f, venta.get(0).getPrecio()); // 100 * 2
    }


    @AfterEach
    void tearDown() throws SQLException {
        String deleteVenta = "DELETE FROM ventas WHERE codigo = ?";
        PreparedStatement stmtVenta = connection.prepareStatement(deleteVenta);
        stmtVenta.setString(1, "P001");
        stmtVenta.executeUpdate();
        stmtVenta.close();

```

```
59          assertEquals("newpass", rs.getString("password"), "La contraseña debería haberse actualizado.");
60
61          rs.close();
62          stmt.close();
63      }
64
65      @Test
66      void testHacerVenta() {
67          ObservableList<CrudUpadate> venta = crudUpadate.hacerVenta("P001", 1, 2.0f, "Venta test");
68
69          assertNotNull(venta);
70          assertEquals(1, venta.size());
71          assertEquals("Producto Test", venta.get(0).getName());
72          assertEquals(200.0f, venta.get(0).getPrecio()); // 100 * 2
73      }
74
75
76      @AfterEach
77      void tearDown() throws SQLException {
78          String deleteVenta = "DELETE FROM ventas WHERE codigo = ?";
79          PreparedStatement stmtVenta = connection.prepareStatement(deleteVenta);
80          stmtVenta.setString(1, "P001");
81          stmtVenta.executeUpdate();
82          stmtVenta.close();
83
84          String deleteProducto = "DELETE FROM productos WHERE codigo = ?";
85          PreparedStatement stmtProducto = connection.prepareStatement(deleteProducto);
86          stmtProducto.setString(1, "P001");
87          stmtProducto.executeUpdate();
88          stmtProducto.close();
89
90          String deleteUser = "DELETE FROM usuarios WHERE usuario = ?";
91          PreparedStatement stmtUser = connection.prepareStatement(deleteUser);
92          stmtUser.setString(1, "testuser");
93          stmtUser.executeUpdate();
94          stmtUser.close();
95
96          if (connection != null) {
97              connection.close();
98          }
99      }
100 }
101
```

e.



Lecciones aprendidas y dificultades: Reflexión grupal sobre la experiencia de testear

Fue realmente duro, especialmente ya que todos fueron test unitarios y realmente algunos parecían depender de otros, sin embargo, al momento de intentar hacer los test de Integración no pudimos terminar de entender la estructura y se complicó bastante, además, el código final que teníamos se borró así que las pruebas fueron sobre código “spaguetti” anterior bajo el cual basamos nuestro proyecto, eso realmente nos mostró la importancia de tener una buena estructura de base en el código que nos permitiera entender la interdependencia de los distintos módulos para así mismo poder tener una mejor automatización de pruebas, de hecho al testear la aplicación nos dimos cuenta de cómo el código que teníamos mostraba ciertos errores no manejados en cuanto a lógica, algunos de ellos eran por ejemplo intentar actualizar un producto que no estaba siquiera en stock, así pues entendimos que debíamos añadir un mayor manejo de este tipo de casos donde es necesario que el usuario sea “detenido” antes de enviar datos que no darán una respuesta o que provocarán errores internos de la aplicación.