

**UNIVERSIDAD TECNOLÓGICA DEL PERÚ**  
**FACULTAD DE INGENIERÍA DE SISTEMAS Y ELECTRÓNICA**



**CURSO INTEGRADOR I: SISTEMAS - SOFTWARE**

**TEMA:**

SISTEMA DE CONTROL DE STOCK

PROYECTO FINAL

**ELABORADO POR:**

- SOTO LUQUE, ELVIS 100%
- RIVERA GUTIERREZ, EMMANUEL 100%
- SARDON MARTINEZ, RALFPH 100%

**DOCENTE:**

MAG. JESAMIN ZEVALLOS

**Repositorio de GitHub:** <https://github.com/Emmanuel-Rivera-G/Proyecto-Final-Curso-Integrador-I-Sistemas-Software>

## INDICE

<b>1.</b>	<b>Introducción .....</b>	<b>4</b>
<b>2.</b>	<b>Problema .....</b>	<b>4</b>
<b>3.</b>	<b>Solución .....</b>	<b>5</b>
<b>4.</b>	<b>Pruebas de Software .....</b>	<b>6</b>
<b>4.1.</b>	<b>Pruebas Unitarias del Servicio para Usuario .....</b>	<b>6</b>
<b>4.1.1.</b>	<b>Prueba unitaria de Registrar Usuario .....</b>	<b>6</b>
<b>4.1.2.</b>	<b>Prueba unitaria de Editar Usuario .....</b>	<b>8</b>
<b>4.1.3.</b>	<b>Prueba unitaria de Eliminar Usuario por Documento .....</b>	<b>10</b>
<b>4.1.4.</b>	<b>Prueba unitaria de Buscar Usuario por Criterios .....</b>	<b>13</b>
<b>4.1.5.</b>	<b>Prueba unitaria de Obtener Usuarios.....</b>	<b>15</b>
<b>4.2.</b>	<b>Pruebas Unitarias del Servicio para Producto.....</b>	<b>18</b>
<b>4.2.1.</b>	<b>Prueba unitaria de Agregar Producto.....</b>	<b>18</b>
<b>4.2.2.</b>	<b>Prueba unitaria de Actualizar Producto .....</b>	<b>18</b>
<b>4.2.3.</b>	<b>Prueba unitaria de Eliminar Producto.....</b>	<b>21</b>
<b>4.2.4.</b>	<b>Prueba unitaria de Obtener Producto por Id .....</b>	<b>23</b>
<b>4.2.5.</b>	<b>Prueba unitaria de Obtener Todos los Productos .....</b>	<b>25</b>
<b>4.3.</b>	<b>Pruebas Unitarias de la clase Conexion.....</b>	<b>28</b>
<b>4.3.1.</b>	<b>Prueba unitaria de Get Connection .....</b>	<b>28</b>
<b>5.</b>	<b>Despliegue.....</b>	<b>30</b>
<b>5.1.</b>	<b>Procedimiento.....</b>	<b>30</b>
<b>5.2.</b>	<b>Link de Aplicación .....</b>	<b>31</b>
<b>6.</b>	<b>Monitoreo de logs .....</b>	<b>33</b>
<b>6.1.</b>	<b>Bibliotecas SLF4J y Logback.....</b>	<b>33</b>
<b>6.3.</b>	<b>Implementación .....</b>	<b>34</b>
<b>6.3.1.</b>	<b>Gestión de Loggers .....</b>	<b>34</b>
<b>6.3.2.</b>	<b>Clase CargadorProducto.java .....</b>	<b>35</b>
<b>6.3.3.</b>	<b>Clase ConverterProducto.java .....</b>	<b>36</b>
<b>6.3.4.</b>	<b>En la clase FiltradorProducto.java .....</b>	<b>37</b>
<b>6.4.</b>	<b>Gestión de Productos .....</b>	<b>38</b>

<b>6.5. Gestión de Usuarios .....</b>	40
<b>6.6. Clase DAOSalidaImpl .....</b>	45
<b>7. Mantenimiento .....</b>	47
<b>7.1. Tipos de mantenimiento:.....</b>	47
<b>7.1.1. Preventivo.....</b>	47
<b>7.1.2. Adaptable .....</b>	48
<b>7.1.3. Correctivo.....</b>	49
<b>7.1.4. Perfectivo o evolutivo .....</b>	50
<b>7.2. Ciclo de mantenimiento.....</b>	51
<b>7.3. Herramientas y procesos .....</b>	51
<b>7.3.1. Herramientas que se usarán .....</b>	51
<b>7.3.2. En qué procesos están involucrados .....</b>	52
<b>7.3.3. Describir al detalle cómo contribuyen al proceso.....</b>	53
<b>8. Control de Versiones .....</b>	54
<b>8.1. README(Instalación) .....</b>	54
<b>8.2. Instalación del proyecto.....</b>	55
<b>8.2.1. Requisitos Previos.....</b>	55
<b>8.2.2. Clonar el repositorio.....</b>	56
<b>8.2.3. Configuración del Proyecto en NetBeans .....</b>	56
<b>8.2.4. Compilar el Proyecto.....</b>	56
<b>8.2.5. Ejecutar el Proyecto .....</b>	56
<b>8.3. Ramas.....</b>	56
<b>8.4. Gráfico de barras .....</b>	57
<b>8.5. Commit de cada integrante .....</b>	59
<b>8.6. Documentación.....</b>	62
<b>9. Comparación de implementación con el diseño.....</b>	65

## **1. Introducción**

En el entorno actual, la gestión de inventarios juega un papel crítico en la operación eficiente de las empresas, especialmente en las pequeñas y medianas empresas (MyPEs y PyMEs), donde una mala administración de los recursos puede resultar en pérdidas significativas. La implementación de un sistema de control de stock se presenta como una solución estratégica para abordar las dificultades inherentes a los métodos tradicionales de control manual. Este documento tiene como objetivo definir los requisitos funcionales y no funcionales de un sistema de control de stock para desktop, el cual permitirá automatizar y agilizar los procesos de entrada y salida de productos en almacenes de diversos sectores, tales como materias primas, productos terminados y existencias generales. A través de este sistema, se busca optimizar el desempeño de los responsables del almacén y facilitar el análisis logístico necesario para una proyección adecuada de los flujos de inventario.

## **2. Problema**

Uno de los principales objetivos de toda empresa, es el poder generar la mayor cantidad de ventas posibles y que se tenga el mejor balance en stock disponible para no faltar a ninguna venta. Pero generalmente las MYPE's y algunas PYME's suelen presentar una ineficiencia en el control de stock, sobre todo en su etapa de inicios o incremento de actividades.

En la actualidad muchas empresas que no cuentan con la automatización digital de sus operaciones suelen tener dificultades en la gestión y administración de sus inventarios. Esto suele ser un desafío común en MYPE's y algunas PYME's, generando una serie de consecuencias negativas que impactan en la rentabilidad y competitividad de toda empresa que no se adapte a la tecnología.

Entre los principales problemas que se suelen generar ante la falta de un sistema de control de stock suelen ser:

- Gestión ineficiente de los inventarios: Los sistemas manuales suelen provocar desorganización en los registros de existencias, lo que dificulta el control adecuado de los productos en almacén y puede generar sobreabastecimiento o desabastecimiento.

- Errores humanos en el control físico: La falta de automatización en los procesos de entrada y salida de mercancías incrementa el riesgo de errores humanos, lo que puede traducirse en incongruencias entre el inventario físico y el inventario registrado.
- Falta de información en tiempo real para la toma de decisiones: La carencia de un sistema que ofrezca datos actualizados dificulta la capacidad de la empresa para realizar análisis efectivos y oportunos, afectando la toma de decisiones estratégicas.
- Ineficiencia en la generación de reportes: Al obtener realizar reportes manuales suele haber una mayor dificultad y demora en realizar los cálculos y verificar el flujo histórico de los productos, siendo este un proceso lento y tedioso.

Esta problemática puede llegar a afectar el nivel de ingresos que percibe la empresa. De aquí es donde nace la necesidad de poder contar con un sistema que facilite a los operarios almaceneros y analistas logísticos poder visualizar los productos disponibles en tiempo real, entradas y salidas de productos, rotación de inventarios y demás propio de un sistema de control de stock.

### **3. Solución**

La solución propuesta es un sistema de control de stock robusto y escalable diseñado para computadoras desktop, capaz de adaptarse a las necesidades de MyPEs y PyMEs que requieren un manejo eficiente de sus inventarios. Este software permitirá a las empresas registrar, organizar y monitorear el flujo de productos o elementos de manera automatizada y precisa, reduciendo significativamente los errores humanos que suelen ocurrir con los sistemas manuales.

Este sistema está diseñado para optimizar las operaciones de almacenamiento al proporcionar una gestión eficiente de las entradas y salidas de productos, facilitando la actualización en tiempo real del inventario disponible. Entre sus principales características se encuentran:

- Registro automatizado de inventario: Permite que cada producto sea ingresado al sistema mediante códigos únicos o escaneos, asegurando que las entradas y salidas sean reflejadas de manera precisa y sin demoras.

- Organización flexible del stock: El sistema ofrece una interfaz intuitiva para organizar los productos en diferentes categorías, almacenes y ubicaciones específicas dentro del depósito, optimizando el proceso de búsqueda y acceso a la información.
- Monitoreo en tiempo real: La plataforma proporciona datos actualizados sobre las existencias, lo que permite tomar decisiones informadas y evitar tanto el sobreabastecimiento como el desabastecimiento, asegurando un flujo óptimo de los productos.

Al integrar este servicio de software, las empresas no solo podrán automatizar sus procesos de control de stock, sino también mejorar la eficiencia operativa, facilitando una mejor toma de decisiones a nivel logístico y administrativo. Esto es particularmente relevante en las empresas que manejan múltiples productos o materiales, donde el control preciso y actualizado es esencial para garantizar la rentabilidad del negocio.

## **4. Pruebas de Software**

### **4.1. Pruebas Unitarias del Servicio para Usuario**

#### **4.1.1. Prueba unitaria de Registrar Usuario**

Verifica que el sistema registre correctamente un nuevo usuario con información válida. Asegura que los datos sean almacenados en la base de datos y que se devuelva un mensaje de éxito.

**Criterios de éxito o resultados esperados:**

- Se almacena toda la información del usuario en la base de datos.
- No se permite registrar usuarios con datos incompletos o duplicados.

```

    /**
     * Prueba para verificar el registro de un nuevo usuario. Se crea un
     * DTOUsuario y se llama al método registrarUsuario. Se comprueba que el
     * usuario se haya registrado correctamente.
     *
     * @throws SQLException Si ocurre un error en la base de datos.
     */
    @Test
    public void testRegistrarUsuario() throws SQLException {
        DTOUsuario usuario = new DTOUsuario();
        usuario.setDocumento("123456");
        usuario.setNombre("Usuario 1");
        usuario.setApellido("Apellido 1");
        usuario.setDireccion("Calle 1");
        usuario.setTelefono("9998456631");
        usuario.setCorreo("usuario1@example.com");
        usuario.setUsername("user1");
        usuario.setPassword("password123");
        usuario.setIdTipoUsuario(1);

        boolean registrado = serviceUsuario.registrarUsuario(usuario);
        assertTrue(registrado); // Verifica que el registro fue exitoso

        List<DTOUsuario> result = serviceUsuario.obtenerUsuarios();
        assertTrue(result.stream().anyMatch(u -> "123456".equals(u.getDocumento())));
    }

```

## Descripción de la prueba unitaria

Creación del objeto DTOUsuario:

Se instancia un nuevo objeto DTOUsuario.

Se establecen los atributos del usuario:

Documento: "123456"

Nombre: "Usuario 1"

Apellido: "Apellido 1"

Dirección: "Calle 1"

Teléfono: "9998456631"

Correo: "usuario1@example.com"

Nombre de usuario: "user1"

Contraseña: "password123"

Tipo de usuario: 1

## **Registro del usuario.**

Se llama al método registrarUsuario del servicio serviceUsuario pasando el objeto DTOUsuario como parámetro.

Se almacena el resultado del registro en una variable booleana registrado.

## **Verificación del registro.**

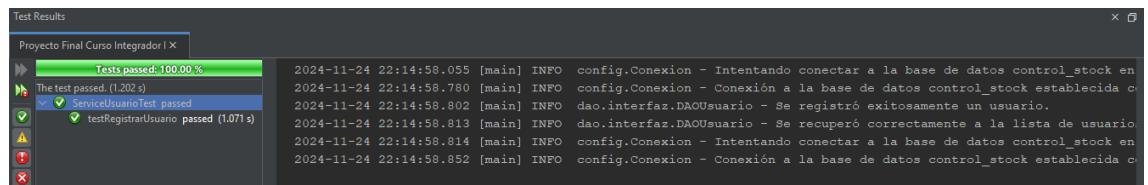
Se utiliza assertTrue para verificar que el valor de registrado sea verdadero, lo que indica que el registro fue exitoso.

## **Obtención y verificación de la lista de usuarios.**

Se llama al método obtenerUsuarios del servicio serviceUsuario para obtener la lista de usuarios registrados.

Se utiliza assertTrue junto con stream y anyMatch para verificar que la lista de usuarios contiene un usuario con el documento "123456".

## **Resultado:**



### **4.1.2. Prueba unitaria de Editar Usuario**

Comprueba que el sistema permita modificar la información de un usuario existente, garantizando que solo los datos válidos sean actualizados.

#### **Criterios de éxito o resultados esperados:**

- Los cambios realizados se reflejan correctamente en la base de datos.
- Se devuelve un valor de confirmación de la actualización.
- No se permite modificar usuarios inexistentes.

```

    /**
     * Prueba para verificar la edición de un usuario existente. Se registra un
     * usuario y luego se edita su nombre. Se verifica que el nombre se haya
     * actualizado correctamente.
     *
     * @throws SQLException Si ocurre un error en la base de datos.
     */
    @Test
    public void testEditarUsuario() throws SQLException {
        DTOUsuario usuario = new DTOUsuario();
        usuario.setDocumento("123456");
        usuario.setNombre("Usuario 1");
        usuario.setApellido("Apellido 1");
        usuario.setDireccion("Calle 1");
        usuario.setTelefono("9998456631");
        usuario.setCorreo("usuariol@example.com");
        usuario.setUsername("userl");
        usuario.setPassword("password123");
        usuario.setIdTipoUsuario(1);

        serviceUsuario.registrarUsuario(usuario);

        usuario.setNombre("Usuario Editado");
        boolean editado = serviceUsuario.editarUsuario(usuario);
        assertTrue(editado); // Verifica que la edición fue exitosa

        DTOUsuario result = serviceUsuario.buscarUsuarioPorCriteria("Usuario Editado", null, null, null).get(0);
        assertEquals("Usuario Editado", result.getNombre()); // Verifica que el nombre haya cambiado
    }
}

```

## Descripción de la prueba unitaria

Creación del objeto DTOUsuario:

Se instancia un nuevo objeto DTOUsuario.

Se establecen los atributos del usuario:

Documento: "123456"

Nombre: "Usuario 1"

Apellido: "Apellido 1"

Dirección: "Calle 1"

Teléfono: "9998456631"

Correo: "usuariol@example.com"

Nombre de usuario: "userl"

Contraseña: "password123"

Tipo de usuario: 1

### **Registro del usuario:**

Se llama al método registrarUsuario del servicio serviceUsuario pasando el objeto DTOUsuario como parámetro.

### **Edición del usuario:**

Se actualiza el nombre del usuario a "Usuario Editado".

Se llama al método editarUsuario del servicio serviceUsuario con el objeto DTOUsuario actualizado.

Se almacena el resultado de la edición en una variable booleana editado.

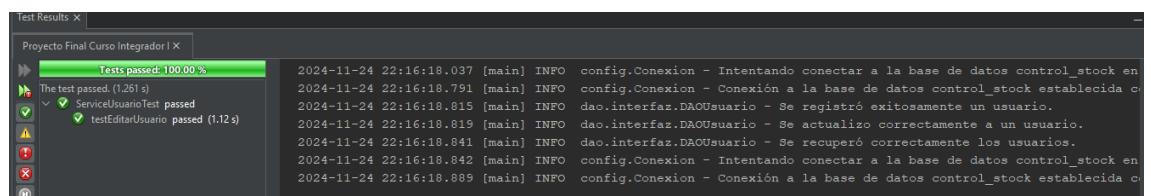
### **Verificación de la edición:**

Se utiliza assertTrue para verificar que el valor de editado sea verdadero, indicando que la edición fue exitosa.

Se llama al método buscarUsuarioPorCriterios del servicio serviceUsuario para obtener el usuario con el nombre "Usuario Editado".

Se verifica que el nombre del usuario en el resultado sea "Usuario Editado" utilizando assertEquals.

### **Resultado:**



The screenshot shows a 'Test Results' window from a Java IDE. The title bar says 'Test Results' and 'Proyecto Final Curso Integrador I'. The main area displays a green bar at the top indicating 'Tests passed: 100.00 %'. Below this, it shows a tree view with a single node 'ServiceUsuarioTest passed' which has a child node 'testEditarUsuario passed (1:12 s)'. To the left of the tree is a toolbar with icons for running tests, stopping them, and viewing logs. On the right side of the window, there is a log pane showing several INFO-level log entries related to database connections and user registration/editation. The log entries are as follows:

```
2024-11-24 22:16:18.037 [main] INFO config.Conexion - Intentando conectar a la base de datos control_stock en
2024-11-24 22:16:18.791 [main] INFO config.Conexion - Conexión a la base de datos control_stock establecida con éxito.
2024-11-24 22:16:18.819 [main] INFO dao.interfaz.DAOUsuario - Se registró exitosamente un usuario.
2024-11-24 22:16:18.819 [main] INFO dao.interfaz.DAOUsuario - Se actualizó correctamente un usuario.
2024-11-24 22:16:18.841 [main] INFO dao.interfaz.DAOUsuario - Se recuperó correctamente los usuarios.
2024-11-24 22:16:18.842 [main] INFO config.Conexion - Intentando conectar a la base de datos control_stock en
2024-11-24 22:16:18.889 [main] INFO config.Conexion - Conexión a la base de datos control_stock establecida con éxito.
```

### **4.1.3. Prueba unitaria de Eliminar Usuario por Documento**

Asegura que un usuario pueda ser eliminado del sistema al proporcionar un número de documento válido.

#### **Criterios de éxito o resultados esperados:**

- El usuario es eliminado de la base de datos.
- Se devuelve un valor de confirmación de eliminación.

```

    /**
     * Prueba para verificar la eliminación de un usuario por su documento. Se
     * registra un usuario y luego se elimina. Se comprueba que el usuario ya no
     * esté en la lista de usuarios.
     *
     * @throws SQLException Si ocurre un error en la base de datos.
     */
    @Test
    public void testEliminarUsuarioPorDocumento() throws SQLException {
        DTOUsuario usuario = new DTOUsuario();
        usuario.setDocumento("123456");
        usuario.setNombre("Usuario 1");
        usuario.setApellido("Apellido 1");
        usuario.setCorreo("usuario1@example.com");
        usuario.setDireccion("Calle 1");
        usuario.setTelefono("9998456631");
        usuario.setUsername("user1");
        usuario.setPassword("password123");
        usuario.setIdTipoUsuario(1);

        serviceUsuario.registrarUsuario(usuario);

        boolean eliminado = serviceUsuario.eliminarUsuarioPorDocumento("123456");
        assertTrue(eliminado); // Verifica que la eliminación fue exitosa

        List<DTOUsuario> result = serviceUsuario.obtenerUsuarios();
        assertFalse(result.stream().anyMatch(u -> "123456".equals(u.getDocumento())));
    }
}

```

## Descripción de la prueba unitaria

Creación del objeto DTOUsuario:

Se instancia un nuevo objeto DTOUsuario.

Se establecen los atributos del usuario:

Documento: "123456"

Nombre: "Usuario 1"

Apellido: "Apellido 1"

Correo: "usuario1@example.com"

Dirección: "Calle 1"

Teléfono: "9998456631"

Nombre de usuario: "user1"

Contraseña: "password123"

Tipo de usuario: 1

### **Registro del usuario:**

Se llama al método registrarUsuario del servicio serviceUsuario pasando el objeto DTOUsuario como parámetro.

### **Eliminación del usuario:**

Se llama al método eliminarUsuarioPorDocumento del servicio serviceUsuario con el documento "123456" del usuario que se desea eliminar.

Se almacena el resultado de la eliminación en una variable booleana eliminado.

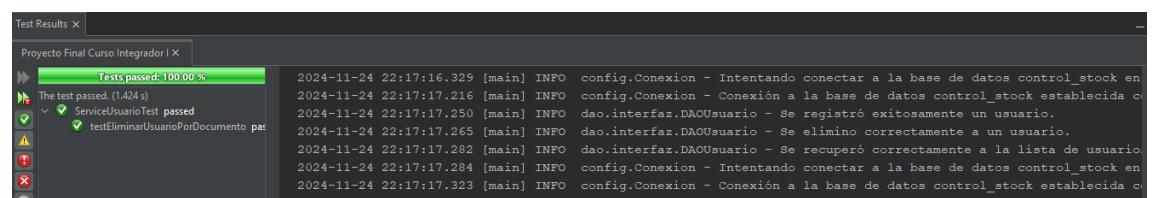
### **Verificación de la eliminación:**

Se utiliza assertTrue para verificar que el valor de eliminado sea verdadero, indicando que la eliminación fue exitosa.

Se llama al método obtenerUsuarios del servicio serviceUsuario para obtener la lista de usuarios registrados.

Se utiliza assertFalse junto con stream y anyMatch para verificar que la lista de usuarios no contiene un usuario con el documento "123456".

### **Resultado:**



The screenshot shows a 'Test Results' window from a Java IDE. The title bar says 'Test Results X' and 'Proyecto Final Curso Integrador I X'. The status bar at the top indicates 'Tests passed: 100.00 %'. Below this, there is a tree view of test results. It shows a single test named 'ServiceUsuarioTest passed' which contains a sub-test 'testEliminarUsuarioPorDocumento passed'. All tests are marked with a green checkmark icon. To the right of the tree view, there is a log window displaying the following log entries:

```
2024-11-24 22:17:16.329 [main] INFO config.Conexion - Intentando conectar a la base de datos control_stock en
2024-11-24 22:17:17.216 [main] INFO config.Conexion - Conexión a la base de datos control_stock establecida con éxito.
2024-11-24 22:17:17.250 [main] INFO dao.interfaz.DAOUsuario - Se registró exitosamente un usuario.
2024-11-24 22:17:17.265 [main] INFO dao.interfaz.DAOUsuario - Se eliminó correctamente a un usuario.
2024-11-24 22:17:17.282 [main] INFO dao.interfaz.DAOUsuario - Se recuperó correctamente a la lista de usuario.
2024-11-24 22:17:17.284 [main] INFO config.Conexion - Intentando conectar a la base de datos control_stock en
2024-11-24 22:17:17.323 [main] INFO config.Conexion - Conexión a la base de datos control_stock establecida con éxito.
```

#### 4.1.4. Prueba unitaria de Buscar Usuario por Criterios

Verifica que el sistema pueda buscar y retornar usuarios que coincidan con criterios específicos.

##### Criterios de éxito o resultados esperados:

- Se devuelven todos los usuarios que cumplen los criterios de búsqueda.
- La búsqueda es eficiente y no genera errores.
- Si no hay coincidencias, se devuelve un resultado vacío.

```
/*
 * Prueba para buscar un usuario por criterios específicos. Se registra un
 * usuario y luego se busca utilizando su nombre. Se verifica que el usuario
 * esté presente en los resultados.
 *
 * @throws SQLException Si ocurre un error en la base de datos.
 */
@Test
public void testBuscarUsuarioPorCriterios() throws SQLException {
    DTOUsuario usuario = new DTOUsuario();
    usuario.setDocumento("123456");
    usuario.setNombre("Usuario 1");
    usuario.setApellido("Apellido 1");
    usuario.setCorreo("usuario1@example.com");
    usuario.setDireccion("Calle 1");
    usuario.setTelefono("9998456631");
    usuario.setUsername("user1");
    usuario.setPassword("password123");
    usuario.setIdTipoUsuario(1);

    serviceUsuario.registrarUsuario(usuario);

    List<DTOUsuario> result = serviceUsuario.buscarUsuarioPorCriterios("Usuario 1", null, null, null);
    assertTrue(result.stream().anyMatch(u -> "123456".equals(u.getDocumento())));
}
```

##### Descripción de la prueba unitaria

Creación del objeto DTOUsuario:

Se instancia un nuevo objeto DTOUsuario.

Se establecen los atributos del usuario:

Documento: "123456"

Nombre: "Usuario 1"

Apellido: "Apellido 1"

Correo: [usuario1@example.com](mailto:usuario1@example.com)

Dirección: "Calle 1"

Teléfono: "9998456631"

Nombre de usuario: "user1"

Contraseña: "password123"

Tipo de usuario: 1

### **Registro del usuario:**

Se llama al método registrarUsuario del servicio serviceUsuario pasando el objeto DTOUsuario como parámetro.

### **Búsqueda del usuario:**

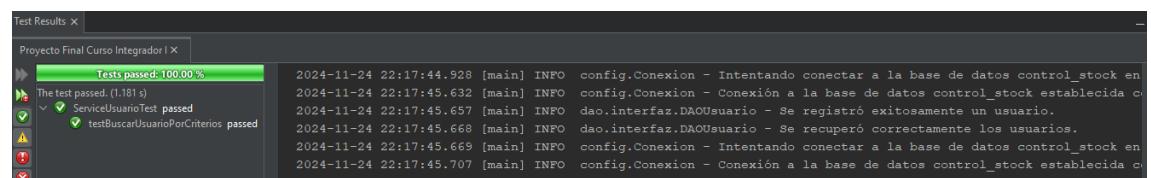
Se llama al método buscarUsuarioPorCriterios del servicio serviceUsuario utilizando "Usuario 1" como criterio de búsqueda para el nombre.

Se almacena el resultado de la búsqueda en una lista result de objetos DTOUsuario.

### **Verificación de la búsqueda:**

Se utiliza assertTrue junto con stream y anyMatch para verificar que la lista de resultados contiene un usuario con el documento "123456".

### **Resultado:**



```
Test Results x |  
Proyecto Final Curso Integrador I x  
Tests passed: 100.00 %  
The test passed. (1.181 s)  
✓ ServiceUsuarioTest passed  
  ✓ testBuscarUsuarioPorCriterios passed  
⚠  
⚠  
⚠  
⚠  
2024-11-24 22:17:44.928 [main] INFO config.Conexion - Intentando conectar a la base de datos control_stock en  
2024-11-24 22:17:45.632 [main] INFO config.Conexion - Conexión a la base de datos control_stock establecida con éxito.  
2024-11-24 22:17:45.657 [main] INFO dao.interfaz.DAOUsuario - Se registró exitosamente un usuario.  
2024-11-24 22:17:45.668 [main] INFO dao.interfaz.DAOUsuario - Se recuperó correctamente los usuarios.  
2024-11-24 22:17:45.669 [main] INFO config.Conexion - Intentando conectar a la base de datos control_stock en  
2024-11-24 22:17:45.707 [main] INFO config.Conexion - Conexión a la base de datos control_stock establecida con éxito.
```

#### 4.1.5. Prueba unitaria de Obtener Usuarios

Garantiza que el sistema pueda listar todos los usuarios almacenados en la base de datos.

##### Criterios de éxito o resultados esperados:

- Se devuelve un listado completo de usuarios.
- La respuesta incluye todos los campos relevantes para cada usuario.
- No se generan errores si la base de datos está vacía.

```
/*
 * Prueba para obtener todos los usuarios registrados en el sistema. Se
 * registran dos usuarios y se verifica que ambos estén presentes en la
 * lista de usuarios.
 *
 * @throws SQLException Si ocurre un error en la base de datos.
 */
@Test
public void testObtenerUsuarios() throws SQLException {
    DTOUsuario usuario1 = new DTOUsuario();
    usuario1.setDocumento("123456");
    usuario1.setNombre("Usuario 1");
    usuario1.setApellido("Apellido 1");
    usuario1.setDireccion("Calle 1");
    usuario1.setTelefono("9998456631");
    usuario1.setCorreo("usuario1@example.com");
    usuario1.setUsername("user1");
    usuario1.setPassword("password123");
    usuario1.setIdTipoUsuario(1);

    DTOUsuario usuario2 = new DTOUsuario();
    usuario2.setDocumento("654321");
    usuario2.setNombre("Usuario 2");
    usuario2.setApellido("Apellido 2");
    usuario2.setDireccion("Calle 1");
    usuario2.setTelefono("9998456631");
    usuario2.setCorreo("usuario2@example.com");
    usuario2.setUsername("user2");
    usuario2.setPassword("password456");
    usuario2.setIdTipoUsuario(2);

    serviceUsuario.registrarUsuario(usuario1);
    serviceUsuario.registrarUsuario(usuario2);

    List<DTOUsuario> result = serviceUsuario.obtenerUsuarios();
    assertTrue(result.stream().anyMatch(u -> "123456".equals(u.getDocumento())));
    assertTrue(result.stream().anyMatch(u -> "654321".equals(u.getDocumento())));
}
```

##### Descripción de la prueba unitaria

##### Creación del primer objeto DTOUsuario:

Se instancia un nuevo objeto DTOUsuario para el primer usuario.

Se establecen los atributos del primer usuario:

Documento: "123456"

Nombre: "Usuario 1"

Apellido: "Apellido 1"

Dirección: "Calle 1"

Teléfono: "9998456631"

Correo: "usuario1@example.com"

Nombre de usuario: "user1"

Contraseña: "password123"

Tipo de usuario: 1

### **Creación del segundo objeto DTOUsuario:**

Se instancia un nuevo objeto DTOUsuario para el segundo usuario.

Se establecen los atributos del segundo usuario:

Documento: "654321"

Nombre: "Usuario 2"

Apellido: "Apellido 2"

Dirección: "Calle 1"

Teléfono: "9998456631"

Correo: "usuario2@example.com"

Nombre de usuario: "user2"

Contraseña: "password456"

Tipo de usuario: 2

### **Registro de los usuarios:**

Se llama al método registrarUsuario del servicio serviceUsuario pasando el primer objeto DTOUsuario como parámetro.

Se llama nuevamente al método registrarUsuario del servicio serviceUsuario pasando el segundo objeto DTOUsuario como parámetro.

### **Obtención de la lista de usuarios:**

Se llama al método obtenerUsuarios del servicio serviceUsuario para obtener la lista de todos los usuarios registrados.

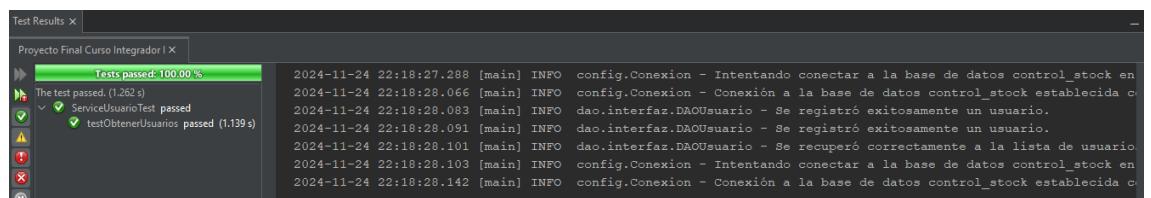
Se almacena el resultado de la obtención en una lista result de objetos DTOUsuario.

### **Verificación de la lista de usuarios:**

Se utiliza assertTrue junto con stream y anyMatch para verificar que la lista de resultados contiene un usuario con el documento "123456".

Se utiliza assertTrue nuevamente para verificar que la lista de resultados contiene un usuario con el documento "654321".

### **Resultado:**



The screenshot shows a 'Test Results' window from a Java IDE. The title bar says 'Test Results' and 'Proyecto Final Curso Integrador I'. A progress bar at the top indicates 'Tests passed: 100.00 %'. Below the bar, there's a tree view showing a single test named 'ServiceUsuarioTest passed' which includes a sub-test 'testObtenerUsuarios passed (1.139 s)'. To the right of the tree view is a log window displaying the following log entries:

```
2024-11-24 22:18:27.288 [main] INFO config.Conexion - Intentando conectar a la base de datos control_stock en
2024-11-24 22:18:28.066 [main] INFO config.Conexion - Conexión a la base de datos control_stock establecida c
2024-11-24 22:18:28.083 [main] INFO dao.interfaz.DAOUsuario - Se registró exitosamente un usuario.
2024-11-24 22:18:28.091 [main] INFO dao.interfaz.DAOUsuario - Se registró exitosamente un usuario.
2024-11-24 22:18:28.101 [main] INFO dao.interfaz.DAOUsuario - Se recuperó correctamente a la lista de usuario
2024-11-24 22:18:28.103 [main] INFO config.Conexion - Intentando conectar a la base de datos control_stock en
2024-11-24 22:18:28.142 [main] INFO config.Conexion - Conexión a la base de datos control_stock establecida c
```

## 4.2. Pruebas Unitarias del Servicio para Producto

### 4.2.1. Prueba unitaria de Agregar Producto

Verifica que el sistema registre correctamente un nuevo producto con información válida.

#### Criterios de éxito o resultados esperados:

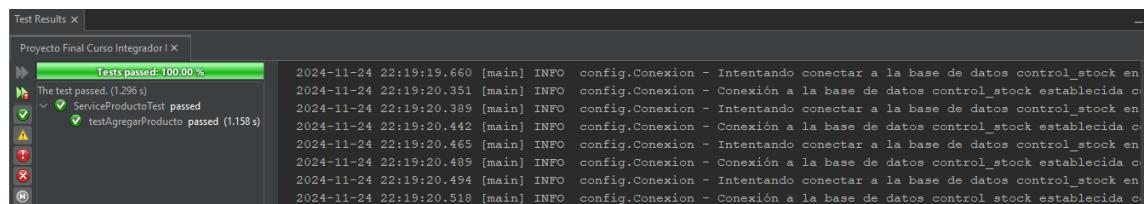
- El producto se almacena en la base de datos con todos sus datos.
- No se permiten registros duplicados ni productos con información incompleta.

```
/*
 * Test que verifica la funcionalidad de agregar un producto.
 * Se espera que el producto agregado esté presente en la lista
 * de productos del DAO simulado.
 */
@Test
public void testAgregarProducto() {
    DTOProducto dtoProducto = new DTOProducto();
    dtoProducto.setNombre("Producto de Prueba");
    dtoProducto.setIdCategoria(1);
    dtoProducto.setUndMedida("Unidad");
    dtoProducto.setStock(10);

    serviceProducto.agregarProducto(dtoProducto);

    List<DTOProducto> dtoProductos = daoProductoSimulado.obtenerTodosLosProductos();
    assertTrue(dtoProductos.stream().anyMatch(p -> "Producto de Prueba".equals(p.getNombre())));
}
```

#### Resultado:



### 4.2.2. Prueba unitaria de Actualizar Producto

Comprueba que los detalles de un producto puedan ser actualizados correctamente.

#### Criterios de éxito o resultados esperados:

- Los cambios realizados en el producto se reflejan en la base de datos.
- Se devuelve un mensaje de éxito al completar la actualización.
- No se permite actualizar productos inexistentes.

```

    /**
     * Test que verifica la funcionalidad de actualizar un producto.
     * Se espera que el nombre del producto se actualice correctamente
     * después de realizar la llamada al servicio de actualización.
     */
    @Test
    public void testActualizarProducto() {
        DTOProducto dtoProducto = new DTOProducto();
        dtoProducto.setNombre("Producto de Prueba");
        dtoProducto.setIdCategoria(1);
        dtoProducto.setUndMedida("Unidad");
        dtoProducto.setStock(10);

        serviceProducto.agregarProducto(dtoProducto);

        List<DTOProducto> productos = daoProductoSimulado.obtenerTodosLosProductos();
        int newProductId = productos.get(productos.size() - 1).getIdProducto();

        dtoProducto.setIdProducto(newProductId);
        dtoProducto.setNombre("Producto Actualizado");
        serviceProducto.actualizarProducto(dtoProducto);

        DTOProducto result = daoProductoSimulado.obtenerProductoPorId(newProductId);
        assertEquals("Producto Actualizado", result.getNombre());
    }
}

```

## Descripción de la prueba unitaria

Creación del objeto DTOProducto:

Se instancia un nuevo objeto DTOProducto.

Se establecen los atributos del producto:

Nombre: "Producto de Prueba"

ID de Categoría: 1

Unidad de Medida: "Unidad"

Stock: 10

### Registro del producto:

Se llama al método agregarProducto del servicio serviceProducto pasando el objeto DTOProducto como parámetro.

Obtención del ID del producto registrado:

Se llama al método obtenerTodosLosProductos del DAO daoProductoSimulado para obtener la lista de todos los productos registrados.

Se obtiene el ID del último producto registrado (el producto recién agregado).

#### **Actualización del producto:**

Se actualiza el nombre del producto a "Producto Actualizado".

Se establece el ID del producto con el ID obtenido previamente.

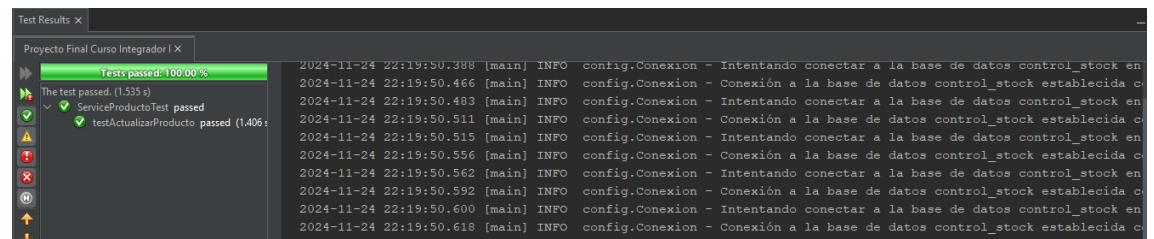
Se llama al método actualizarProducto del servicio serviceProducto con el objeto DTOProducto actualizado.

#### **Verificación de la actualización:**

Se llama al método obtenerProductoPorId del DAO daoProductoSimulado para obtener el producto con el ID del producto actualizado.

Se utiliza assertEquals para verificar que el nombre del producto en el resultado sea "Producto Actualizado".

#### **Resultado:**



The screenshot shows a 'Test Results' window from a Java IDE. The title bar says 'Proyecto Final Curso Integrador I x'. The main area displays a green bar indicating 'Tests passed: 100:00 %' and 'The test passed. (1.535 s)'. Below this, it shows a tree view with a green checkmark next to 'ServiceProductoTest passed' and 'testActualizarProducto passed (1.406 ms)'. To the right of the tree view, there is a log of INFO-level messages from the 'config.Conexion' class, showing multiple attempts to connect to the 'control\_stock' database at different timestamps on November 24, 2024.

```
2024-11-24 22:19:50.388 [main] INFO config.Conexion - Intentando conectar a la base de datos control_stock en
2024-11-24 22:19:50.466 [main] INFO config.Conexion - Conexión a la base de datos control_stock establecida c
2024-11-24 22:19:50.493 [main] INFO config.Conexion - Intentando conectar a la base de datos control_stock en
2024-11-24 22:19:50.511 [main] INFO config.Conexion - Conexión a la base de datos control_stock establecida c
2024-11-24 22:19:50.515 [main] INFO config.Conexion - Intentando conectar a la base de datos control_stock en
2024-11-24 22:19:50.556 [main] INFO config.Conexion - Conexión a la base de datos control_stock establecida c
2024-11-24 22:19:50.562 [main] INFO config.Conexion - Intentando conectar a la base de datos control_stock en
2024-11-24 22:19:50.592 [main] INFO config.Conexion - Conexión a la base de datos control_stock establecida c
2024-11-24 22:19:50.600 [main] INFO config.Conexion - Intentando conectar a la base de datos control_stock en
2024-11-24 22:19:50.618 [main] INFO config.conexion - Conexión a la base de datos control_stock establecida c
```

#### 4.2.3. Prueba unitaria de Eliminar Producto

Asegura que un producto pueda ser eliminado correctamente utilizando su identificador.

##### Criterios de éxito o resultados esperados:

- El producto se elimina de la base de datos.
- Se devuelve un mensaje de confirmación.
- No se permite eliminar productos inexistentes o con identificadores inválidos.

```
/*
 * Test que verifica la funcionalidad de eliminar un producto.
 * Se espera que el producto eliminado no esté presente en la lista
 * de productos después de realizar la llamada al servicio de eliminación.
 */
@Test
public void testEliminarProducto() {
    DTOProducto dtoProducto = new DTOProducto();
    dtoProducto.setNombre("Producto de Prueba");
    dtoProducto.setIdCategoria(1);
    dtoProducto.setUndMedida("Unidad");
    dtoProducto.setStock(10);

    serviceProducto.agregarProducto(dtoProducto);

    List<DTOProducto> productos = daoProductoSimulado.obtenerTodosLosProductos();
    int newProductId = productos.get(productos.size() - 1).getIdProducto();

    serviceProducto.eliminarProducto(newProductId);

    DTOProducto result = daoProductoSimulado.obtenerProductoPorId(newProductId);
    assertNull(result);
}
```

##### Descripción de la prueba unitaria

Creación del objeto DTOProducto:

Se instancia un nuevo objeto DTOProducto.

Se establecen los atributos del producto:

Nombre: "Producto de Prueba"

ID de Categoría: 1

Unidad de Medida: "Unidad"

Stock: 10

### **Registro del producto:**

Se llama al método agregarProducto del servicio serviceProducto pasando el objeto DTOProducto como parámetro.

### **Obtención del ID del producto registrado:**

Se llama al método obtenerTodosLosProductos del DAO daoProductoSimulado para obtener la lista de todos los productos registrados.

Se obtiene el ID del último producto registrado (el producto recién agregado).

### **Eliminación del producto:**

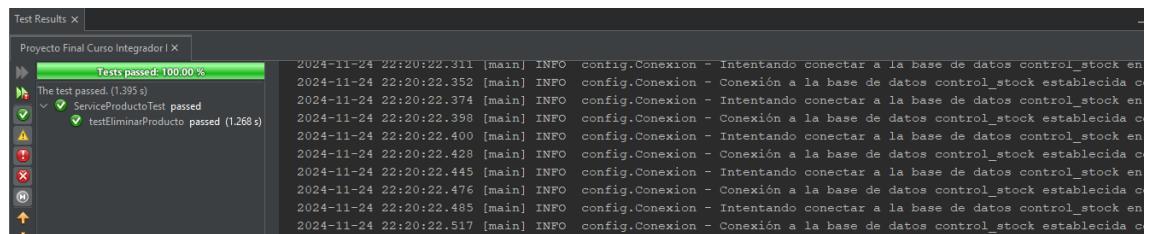
Se llama al método eliminarProducto del servicio serviceProducto con el ID del producto que se desea eliminar.

### **Verificación de la eliminación:**

Se llama al método obtenerProductoPorId del DAO daoProductoSimulado para intentar obtener el producto con el ID del producto eliminado.

Se utiliza assertNull para verificar que el producto no está presente en el sistema.

### **Resultado:**



The screenshot shows the 'Test Results' window from a Java IDE. The title bar says 'Test Results X' and 'Proyecto Final Curso Integrador | X'. The main area displays a green progress bar with the text 'Tests passed: 100.00 %' and 'The test passed. (1.395 s)'. Below this, there is a tree view showing a package named 'ServiceProductoTest' with a single test case 'testEliminarProducto' marked as 'passed' (green checkmark). To the right of the tree view is a log window showing several INFO-level log entries from the 'config.Conexion' class, indicating attempts to connect to the 'control\_stock' database at various timestamps on November 24, 2024.

```
2024-11-24 22:20:22.311 [main] INFO config.Conexion - Intentando conectar a la base de datos control_stock en
2024-11-24 22:20:22.352 [main] INFO config.Conexion - Conexión a la base de datos control_stock establecida co
2024-11-24 22:20:22.374 [main] INFO config.Conexion - Intentando conectar a la base de datos control_stock en
2024-11-24 22:20:22.398 [main] INFO config.Conexion - Conexión a la base de datos control_stock establecida co
2024-11-24 22:20:22.400 [main] INFO config.Conexion - Intentando conectar a la base de datos control_stock en
2024-11-24 22:20:22.428 [main] INFO config.Conexion - Conexión a la base de datos control_stock establecida co
2024-11-24 22:20:22.445 [main] INFO config.Conexion - Intentando conectar a la base de datos control_stock en
2024-11-24 22:20:22.476 [main] INFO config.Conexion - Conexión a la base de datos control_stock establecida co
2024-11-24 22:20:22.485 [main] INFO config.Conexion - Intentando conectar a la base de datos control_stock en
2024-11-24 22:20:22.517 [main] INFO config.Conexion - Conexión a la base de datos control_stock establecida co
```

#### 4.2.4. Prueba unitaria de Obtener Producto por Id

Verifica que el sistema pueda buscar y devolver los detalles de un producto utilizando su identificador único.

**Criterios de éxito o resultados esperados:**

- Se devuelven los detalles completos del producto solicitado.
- Si el producto no existe, se devuelve un error o un mensaje indicando su inexistencia.

```
/*
 * Test que verifica la funcionalidad de obtener un producto por su ID.
 * Se espera que el producto obtenido no sea nulo y que su nombre
 * coincida con el nombre del producto de prueba.
 */
@Test
public void testObtenerProductoPorId() {
    DTOProducto dtoProducto = new DTOProducto();
    dtoProducto.setNombre("Producto de Prueba");
    dtoProducto.setIdCategoria(1);
    dtoProducto.setUndMedida("Unidad");
    dtoProducto.setStock(10);

    serviceProducto.agregarProducto(dtoProducto);

    List<DTOProducto> productos = daoProductoSimulado.obtenerTodosLosProductos();
    int newProductId = productos.get(productos.size() - 1).getIdProducto();

    DTOProducto result = serviceProducto.obtenerProductoPorId(newProductId);
    assertNotNull(result);
    assertEquals("Producto de Prueba", result.getNombre());
}
```

#### Descripción de la prueba unitaria

Creación del objeto DTOProducto:

Se instancia un nuevo objeto DTOProducto.

Se establecen los atributos del producto:

Nombre: "Producto de Prueba"

ID de Categoría: 1

Unidad de Medida: "Unidad"

Stock: 10

### **Registro del producto:**

Se llama al método agregarProducto del servicio serviceProducto pasando el objeto DTOProducto como parámetro.

### **Obtención del ID del producto registrado:**

Se llama al método obtenerTodosLosProductos del DAO daoProductoSimulado para obtener la lista de todos los productos registrados.

Se obtiene el ID del último producto registrado (el producto recién agregado).

### **Búsqueda del producto por su ID:**

Se llama al método obtenerProductoPorId del servicio serviceProducto con el ID del producto registrado.

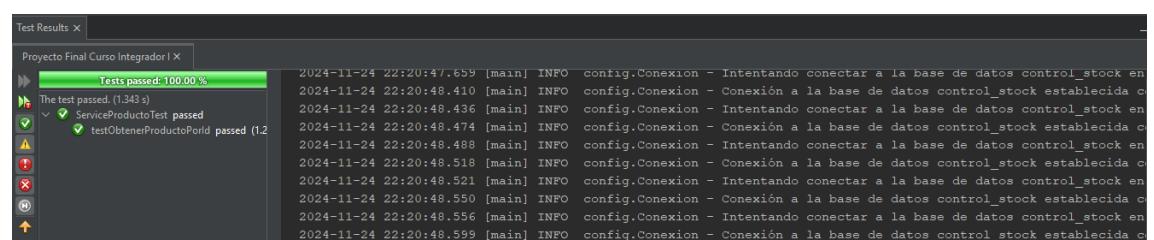
Se almacena el resultado de la búsqueda en un objeto DTOProducto llamado result.

### **Verificación de la obtención:**

Se utiliza assertNotNull para verificar que el objeto result no sea nulo.

Se utiliza assertEquals para verificar que el nombre del producto en el resultado sea "Producto de Prueba".

### **Resultado:**



The screenshot shows the 'Test Results' window from a Java IDE. The title bar says 'Test Results x' and 'Proyecto Final Curso Integrador I x'. The main area displays a tree view of test results. At the top, it says 'Tests passed: 100.00 %' and 'The test passed. (1.343 s)'. Below this, under 'ServiceProductoTest passed', there is a single test named 'testObtenerProductoPorId passed (1.2 s)'. To the right of the tree view, a large block of log output is shown, starting with '2024-11-24 22:20:47.659 [main] INFO config.Conexion - Intentando conectar a la base de datos control\_stock en' and continuing with several identical entries followed by '2024-11-24 22:20:48.556 [main] INFO config.Conexion - Intentando conectar a la base de datos control\_stock en' and '2024-11-24 22:20:48.599 [main] INFO config.conexion - Conexión a la base de datos control\_stock establecida c'.

#### 4.2.5. Prueba unitaria de Obtener Todos los Productos

Comprueba que el sistema pueda listar todos los productos almacenados.

##### Criterios de éxito o resultados esperados:

- Se retorna un listado completo de productos con toda su información.
- No se generan errores si la base de datos está vacía.

```
/*
 * Test que verifica la funcionalidad de obtener todos los productos.
 * Se espera que el tamaño de la lista de productos aumente en 2
 * después de agregar dos productos de prueba.
 */
@Test
public void testObtenerTodosLosProductos() {
    int initialSize = serviceProducto.obtenerTodosLosProductos().size();

    DTOProducto producto1 = new DTOProducto();
    producto1.setNombre("Producto de Prueba 1");
    producto1.setIdCategoria(1);
    producto1.setUndMedida("Unidad");
    producto1.setStock(10);

    DTOProducto producto2 = new DTOProducto();
    producto2.setNombre("Producto de Prueba 2");
    producto2.setIdCategoria(1);
    producto2.setUndMedida("Unidad");
    producto2.setStock(20);

    serviceProducto.agregarProducto(producto1);
    serviceProducto.agregarProducto(producto2);

    List<DTOProducto> productos = serviceProducto.obtenerTodosLosProductos();
    assertEquals(initialSize + 2, productos.size());
}
```

##### Descripción de la prueba unitaria

###### Obtención del tamaño inicial de la lista de productos:

Se llama al método obtenerTodosLosProductos del servicio serviceProducto para obtener la lista de todos los productos registrados.

Se almacena el tamaño inicial de la lista en una variable initialSize.

###### Creación del primer objeto DTOProducto:

Se instancia un nuevo objeto DTOProducto para el primer producto.

Se establecen los atributos del primer producto:

Nombre: "Producto de Prueba 1"

ID de Categoría: 1

Unidad de Medida: "Unidad"

Stock: 10

### **Creación del segundo objeto DTOProducto:**

Se instancia un nuevo objeto DTOProducto para el segundo producto.

Se establecen los atributos del segundo producto:

Nombre: "Producto de Prueba 2"

ID de Categoría: 1

Unidad de Medida: "Unidad"

Stock: 20

### **Registro de los productos:**

Se llama al método agregarProducto del servicio serviceProducto pasando el primer objeto DTOProducto como parámetro.

Se llama nuevamente al método agregarProducto del servicio serviceProducto pasando el segundo objeto DTOProducto como parámetro.

### **Obtención de la lista de productos:**

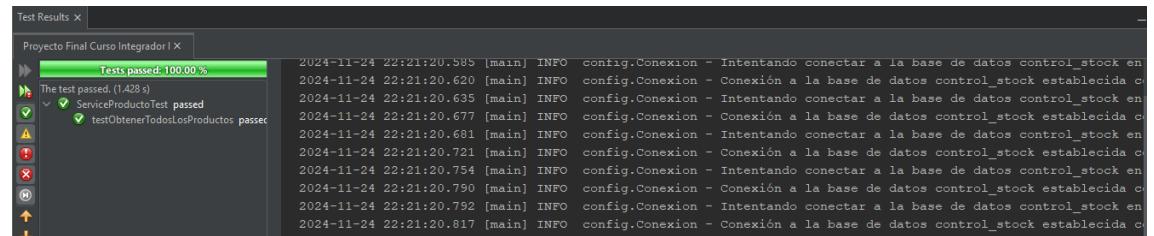
Se llama nuevamente al método obtenerTodosLosProductos del servicio serviceProducto para obtener la lista de todos los productos registrados después de haber agregado los nuevos productos.

Se almacena el resultado de la obtención en una lista productos de objetos DTOProducto.

### **Verificación de la lista de productos:**

Se utiliza assertEquals para verificar que el tamaño de la lista de productos ha aumentado en 2, comparándolo con el tamaño inicial incrementado en dos.

### Resultado:



The screenshot shows the 'Test Results' window from a Java IDE. The title bar says 'Test Results' and 'Proyecto Final Curso Integrador I X'. The main area displays a green bar at the top indicating 'Tests passed: 100.00 %'. Below this, it says 'The test passed. (1.428 s)'. Underneath, there is a tree view with a green checkmark next to 'ServiceProductoTest passed' and another green checkmark next to 'testObtenerTodosLosProductos passed'. To the left of the tree view is a vertical toolbar with icons for play, stop, refresh, and other test-related functions. On the right side of the window, a log of INFO-level messages from the 'config.Conexion' class is displayed, showing multiple attempts to connect to the 'control\_stock' database at different timestamps on November 24, 2024.

```
2024-11-24 22:21:20.585 [main] INFO config.Conexion - Intentando conectar a la base de datos control_stock en
2024-11-24 22:21:20.620 [main] INFO config.Conexion - Conexión a la base de datos control_stock establecida c
2024-11-24 22:21:20.635 [main] INFO config.Conexion - Intentando conectar a la base de datos control_stock en
2024-11-24 22:21:20.677 [main] INFO config.Conexion - Conexión a la base de datos control_stock establecida c
2024-11-24 22:21:20.681 [main] INFO config.Conexion - Intentando conectar a la base de datos control_stock en
2024-11-24 22:21:20.721 [main] INFO config.Conexion - Conexión a la base de datos control_stock establecida c
2024-11-24 22:21:20.754 [main] INFO config.Conexion - Intentando conectar a la base de datos control_stock en
2024-11-24 22:21:20.790 [main] INFO config.Conexion - Conexión a la base de datos control_stock establecida c
2024-11-24 22:21:20.792 [main] INFO config.Conexion - Intentando conectar a la base de datos control_stock en
2024-11-24 22:21:20.817 [main] INFO config.Conexion - Conexión a la base de datos control_stock establecida c
```

## 4.3. Pruebas Unitarias de la clase Conexion

### 4.3.1. Prueba unitaria de Get Connection

Garantiza que el sistema pueda establecer una conexión válida con la base de datos.

#### Criterios de éxito o resultados esperados:

- Se establece la conexión sin errores.
- Se devuelve un objeto de conexión válido

```
/**  
 * Test que verifica que se pueda obtener una conexión a la base de datos.  
 *  
 * Se espera que la conexión no sea nula.  
 * Si ocurre un error al intentar conectarse, se registrará un mensaje de error  
 * y la prueba fallará.  
 */  
@Test  
public void testGetConnection() {  
    try {  
        Connection con = conexion.getConnection();  
        assertNotNull("La conexión no debería ser null", con);  
        LOGGER.info("Test de conexión exitoso.");  
    } catch (Exception e) {  
        LOGGER.error("Test fallido: Error al conectar con la base de datos.", e);  
        fail("La conexión a la base de datos falló: " + e.getMessage());  
    }  
}
```

#### Descripción de la prueba unitaria

Se llama al método getConnection de la clase conexion para obtener una conexión a la base de datos.

Se almacena el resultado de la conexión en una variable con.

#### Verificación de la conexión:

Se utiliza assertNull para verificar que la conexión no sea nula. Esto asegura que se ha obtenido una conexión exitosa.

Se registra un mensaje de información indicando que la prueba de conexión fue exitosa utilizando LOGGER.info.

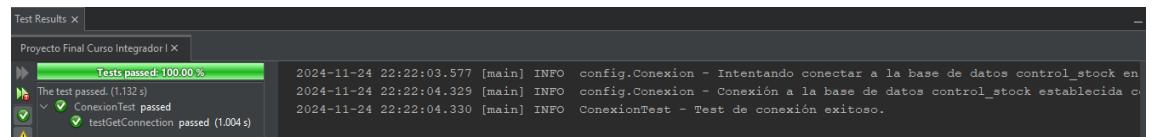
#### Manejo de excepciones:

Si ocurre una excepción al intentar obtener la conexión, se captura la excepción en un bloque catch.

Se registra un mensaje de error indicando que la prueba falló debido a un error al conectar con la base de datos utilizando LOGGER.error.

Se llama al método fail con un mensaje que incluye el detalle de la excepción para indicar que la prueba falló.

### Resultado:



## **5. Despliegue**

### **5.1. Procedimiento**

El despliegue del sistema de control de stock desarrollado con Java y Swing se realizó siguiendo un conjunto de pasos estructurados. A continuación, se describen los procedimientos llevados a cabo:

Herramienta Utilizada:

**Launch4j:** Es una herramienta gratuita que permite convertir archivos .jar de Java en ejecutables .exe para Windows. Facilita la configuración de entornos de ejecución al integrar opciones como especificar el JRE necesario, agregar iconos personalizados y definir parámetros de inicio, garantizando una distribución más profesional y amigable para los usuarios finales.

#### **1. Preparación del Ejecutable:**

- El proyecto fue compilado y empaquetado como un archivo **.jar** utilizando las herramientas de desarrollo de Java.
- Para facilitar la distribución y la ejecución en sistemas Windows, se utilizó la herramienta **Launch4j** para convertir el archivo **.jar** en un ejecutable **.exe**.
- Durante este proceso, se configuraron las dependencias necesarias y se especificaron los argumentos requeridos para garantizar el correcto funcionamiento del ejecutable.

#### **2. Incorporación del Entorno de Ejecución:**

- Para asegurar la portabilidad del sistema y evitar problemas relacionados con versiones de Java incompatibles, se incluyó el **Java Runtime Environment (JRE) 20** junto al ejecutable.
- La carpeta del JRE fue colocada junto al archivo **.exe**, eliminando la necesidad de que el usuario final tenga que instalar Java manualmente.

#### **3. Estructura del Proyecto Desplegado:**

- Se organizó el proyecto en una estructura de carpetas que incluye:
  - **jre/**: Carpeta que contiene el JRE 20 necesario para la ejecución del sistema.

- **libs/**: Carpeta que almacena todas las librerías adicionales requeridas en formato **.jar**. Estas librerías son utilizadas para funcionalidades específicas del sistema, como la conexión a la base de datos o la gestión de interfaces gráficas.
- **ProyectoFinal.exe**: Archivo ejecutable generado por Launch4j.

#### 4. Configuración de la Base de Datos:

- La base de datos del sistema fue desplegada en el servicio **Railway**, que ofrece soporte para bases de datos **MySQL**.
- Se utilizó la prueba gratuita de **5 USD** ofrecida por Railway, la cual proporciona recursos limitados pero suficientes para el correcto funcionamiento del sistema en un entorno de prueba o desarrollo.
- Las credenciales de acceso a la base de datos (host, puerto, usuario y contraseña) fueron configuradas dentro del sistema, garantizando que el software pudiera conectarse de forma remota a la base de datos alojada en Railway.

#### 5. Pruebas de Despliegue:

- Una vez ensamblado el proyecto, se realizaron pruebas exhaustivas para validar que:
  - El ejecutable funciona correctamente en entornos Windows sin necesidad de instalaciones adicionales.
  - La conexión con la base de datos remota alojada en Railway es estable y permite la ejecución de todas las operaciones del sistema, como consultas, inserciones y actualizaciones.
  - Las dependencias configuradas en la carpeta **libs/** se cargan adecuadamente durante la ejecución del programa.

#### 6. Distribución:

- El sistema fue empaquetado en un archivo comprimido que contiene la estructura de carpetas descrita.
- Se proporcionaron instrucciones claras al usuario final para descomprimir el archivo y ejecutar el archivo **ProyectoFinal.exe** sin complicaciones.

#### 5.2. Link de Aplicación

La aplicación puede descargarse desde el siguiente enlace de OneDrive:

**Descargar Aplicación - Sistema de Control de Stock:** [https://utpedupe-my.sharepoint.com/:f/g/personal/u22222056\\_utp\\_edu\\_pe/EsjOt0dWMaJKvLQajaYA6rMBr6o3O-3uR5n\\_T7KG0\\_dAew?e=7jVqSJ](https://utpedupe-my.sharepoint.com/:f/g/personal/u22222056_utp_edu_pe/EsjOt0dWMaJKvLQajaYA6rMBr6o3O-3uR5n_T7KG0_dAew?e=7jVqSJ)

**Nota:** Es necesario asegurarse de mantener la estructura de carpetas intacta al descomprimir los archivos para garantizar el correcto funcionamiento del sistema.

## **6. Monitoreo de logs**

### **6.1. Bibliotecas SLF4J y Logback**

Se hizo uso de las bibliotecas logback-core 1.5.11, logback-classic 1.5.11 y slf4j-api 2.0.16. Estas bibliotecas se caracterizan por ser ampliamente reconocidas en la industria debido a su flexibilidad, rendimiento y compatibilidad con diversos entornos de desarrollo. Además, permiten gestionar los logs de manera eficiente, incluso en aplicaciones con alta demanda de procesamiento, gracias a su diseño optimizado. Con ello también logramos guardar los logs en el archivo llamado application.log.

#### **¿Por qué se utilizó?**

- Más conocidas: Estas bibliotecas son referentes en el manejo de logs, lo que garantiza no solo confiabilidad, sino también una comunidad activa que respalda su uso, facilita la resolución de problemas y ofrece documentación completa. Este soporte asegura que cualquier inconveniente pueda abordarse rápidamente.
- Por su fácil implementación: SLF4J y Logback destacan por su integración intuitiva con múltiples frameworks y tecnologías. Su configuración es simple, lo que reduce tiempos de implementación y mantiene la flexibilidad para adaptarse a cambios futuros, como modificar el backend de almacenamiento o ajustar niveles de log según las necesidades del proyecto.

## 6.2. Archivo de Configuración

### Título: logback.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>

    <!-- Define un appender para consola -->
    <appender name="CONSOLE" class="ch.qos.logback.core.ConsoleAppender">
        <encoder>
            <!-- Agrega el hilo, clase y mejor formato -->
            <pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n%ex{full}</pattern>
        </encoder>
    </appender>

    <!-- Define un appender para archivo -->
    <appender name="FILE" class="ch.qos.logback.core.FileAppender">
        <file>logs/application.log</file>
        <append>true</append>
        <encoder>
            <pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n%ex{full}</pattern>
        </encoder>
    </appender>

    <!-- Nivel de logs para la aplicación -->
    <root level="info">
        <appender-ref ref="CONSOLE" />
        <appender-ref ref="FILE" />
    </root>

</configuration>
```

## 6.3. Implementación

### 6.3.1. Gestión de Loggers

El logger se utiliza para el control de la información de los productos, también el manejo de errores y la muestra de mensajes de éxito, se registrar las acciones realizadas en cada método para hacer un seguimiento de la información.

Clase principal para la fabricación de los loggers (UtilsLoggerManager)

```
package utils;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

/**
 * Esta clase implementa una forma resumida de obtener un registrador de tipo Logger de forma sencilla.
 * Utiliza el método de LoggerFactory, getLogger.
 *
 * @author Emmanuel
 */

public class UtilsLoggerManager {
    public static Logger getLogger(Class<?> clase) {
        return LoggerFactory.getLogger(clase);
    }
}
```

### 6.3.2. Clase CargadorProducto.java

Declaración del Logger (Variable: LOGGER):

```
private final static Logger LOGGER = UtilsLoggerManager.getLogger(CargadorProducto.class);
```

Para registro de errores generales y en valores no numéricos:

```
DTOProducto dtoProducto = new DTOProducto();
if (!ValidatorProducto.esNumerico(txtCodCategoria.getText())) {
    MessageUtils.mostrarMensajeError("El campo ID CATEGORIA deben ser numérico");
    LOGGER.error("Id Categoria no es numérico");
    return null;
}
if (!ValidatorProducto.esNumerico(txtStockProducto.getText())) {
    MessageUtils.mostrarMensajeError("El campo STOCK deben ser numérico");
    LOGGER.error("Stock no es numérico");
    return null;
}
try {
    dtoProducto.setNombre(txtNombreProducto.getText());
    dtoProducto.setIdCategoria(Integer.parseInt(txtCodCategoria.getText()));
    dtoProducto.setUndMedida(txtUndMedida.getText());
    dtoProducto.setStock(Integer.parseInt(txtStockProducto.getText()));
} catch (Exception e) {
    LOGGER.error("Error en la carga de datos: " + e.getMessage());
    return null;
}
```

En la clase ConversorProducto.java

Declaración del Logger (Variable: LOGGER):

```
private static final Logger LOGGER = UtilsLoggerManager.getLogger(ConversorProducto.class);
```

Para registro de errores o información necesario para el control

```
public static Object[][] toMatriz(List<Producto> productos) {
    try {
        if (productos == null || productos.isEmpty()) {
            LOGGER.info("La lista de productos está vacía o es nula.");
            return new Object[0][];
        }
        LOGGER.info("Iniciando conversión de lista de productos a matriz de objetos. Tamaño de la lista: {}", productos.size());
        Object[][] data = new Object[productos.size()][];
        for (int i = 0; i < productos.size(); i++) {
            Producto p = productos.get(i);
            data[i] = ConversorProducto.toArray(p);
        }
        LOGGER.info("Conversión de lista de productos completada con éxito.");
        return data;
    } catch (Exception e) {
        LOGGER.error("Error durante la conversión de lista de productos a matriz de objetos: {}", e.getMessage(), e);
        throw e;
    }
}
```

```

public static Object[] toArray(Producto producto) throws NullPointerException {
    try {
        if (producto == null) {
            LOGGER.error("El producto proporcionado es nulo.");
            throw new NullPointerException("El producto proporcionado es nulo.");
        }
        LOGGER.info("Convirtiendo producto con ID: {} a un array de objetos.", producto.getId());
        Object[] data = new Object[]{
            producto.getId(),
            producto.getNombre(),
            producto.getCategoría().getIdcat(),
            producto.getUndmedida(),
            producto.getStock()
        };
        LOGGER.info("Producto convertido con éxito a un array de objetos.");
        return data;
    } catch (Exception e) {
        LOGGER.error("Error durante la conversión del producto a array de objetos: {}", e.getMessage(), e);
        throw e;
    }
}

public static Object[] toArray(DTOProducto producto) throws NullPointerException {
    try {
        if (producto == null) {
            LOGGER.error("El DTOProducto proporcionado es nulo.");
            throw new NullPointerException("El DTOProducto proporcionado es nulo.");
        }
        LOGGER.info("Convirtiendo DTOProducto a Producto para posterior conversión.");
        Object[] data = ConversorProducto.toArray(ConverterProducto.toProducto(producto));
        LOGGER.info("DTOProducto convertido con éxito a un array de objetos.");
        return data;
    } catch (Exception e) {
        LOGGER.error("Error durante la conversión de DTOProducto a array de objetos: {}", e.getMessage(), e);
        throw e;
    }
}

```

### 6.3.3. Clase ConverterProducto.java

Declaración del Logger (Variable: LOGGER):

```
private static final Logger LOGGER = UtilsLoggerManager.getLogger(ConverterProducto.class);
```

Para definir información de las acciones realizadas

```

public static DTOProducto toDTOProducto(Producto producto) {
    LOGGER.info("Convirtiendo Producto a DTOProducto.");
    return new DTOProducto(producto);
}

public static Producto toProducto(DTOProducto dtoProducto) {
    LOGGER.info("Convirtiendo DTOProducto a Producto.");
    return dtoProducto.toProducto();
}

public static List<Producto> convertirDTOAProducto(List<DTOProducto> dtoProductos) {
    LOGGER.info("Convirtiendo lista de DTOProducto a lista de Producto.");
    return new ArrayList<>(CollectionUtils.collect(dtoProductos, DTOProducto::toProducto));
}

```

```

public static List<DTOProducto> convertirProductosADTO(List<Producto> productos) {
    LOGGER.info("Convirtiendo lista de Producto a lista de DTOProducto.");
    return new ArrayList<>(CollectionUtils.collect(productos, ConverterProducto::toDTOProducto));
}

```

### 6.3.4. En la clase FiltradorProducto.java

Declaración del Logger (Variable: LOGGER):

```

private static final Logger LOGGER = UtilsLoggerManager.getLogger(FiltradorProducto.class);

```

Para registrar las acciones realizadas en la filtración de productos

```

/**
 * Filtra una lista de DTOProducto por nombre con un prefijo específico.
 * @param productos Lista de productos a filtrar.
 * @param prefijo Prefijo del nombre a buscar.
 * @return Lista de productos cuyo nombre empieza con el prefijo.
 */
public static List<DTOProducto> filtrarPorNombreConPrefijo(List<DTOProducto> productos, String prefijo) {
    LOGGER.info("Filtrando productos por nombre con el prefijo: {}", prefijo);
    return filtrarPorVariosCriterios(productos, new Predicate<DTOProducto>() {
        @Override
        public boolean evaluate(DTOProducto producto) {
            return StringUtils.startsWithIgnoreCase(producto.getNombre(), prefijo);
        }
    });
}

/**
 * Filtra una lista de DTOProducto utilizando un criterio personalizado.
 * @param productos Lista de productos a filtrar.
 * @param criterio Criterio de filtrado.
 * @return Lista de productos que cumplen con el criterio.
 */
public static List<DTOProducto> filtrarPorVariosCriterios(List<DTOProducto> productos, Predicate<DTOProducto> criterio) {
    LOGGER.info("Filtrando productos con un criterio personalizado.");
    return new ArrayList<>(CollectionUtils.select(productos, criterio));
}

/**
 * Obtiene una lista de productos únicos entre dos listas de DTOProducto.
 * @param listal Primera lista de DTOProducto.
 * @param lista2 Segunda lista de DTOProducto.
 * @return Lista de productos únicos en la primera lista.
 */
public static List<DTOProducto> filtrarProductosUnicos(List<DTOProducto> listal, List<DTOProducto> lista2) {
    LOGGER.info("Obteniendo productos únicos de las dos listas proporcionadas.");
    return new ArrayList<>(CollectionUtils.subtract(listal, lista2));
}

/**
 * Filtra una lista de DTOProducto por ID con un prefijo específico.
 * @param productos Lista de productos a filtrar.
 * @param prefijo Prefijo del ID a buscar.
 * @return Lista de productos cuyo ID empieza con el prefijo.
 */
public static List<DTOProducto> filtrarPorIdConPrefijo(List<DTOProducto> productos, String prefijo) {
    LOGGER.info("Filtrando productos por ID con el prefijo: {}", prefijo);
    return filtrarPorVariosCriterios(productos, (producto) -> {
        return StringUtils.startsWith(String.valueOf(producto.getIdProducto()), prefijo);
    });
}

```

## 6.4.Gestión de Productos

El logger se utiliza para validar procesos y registrar eventos importantes entorno a los productos. Su implementación permite un control detallado en actividades como validación ids, creación de productos, edición de información, manejo de registros en la base de datos y gestión de productos.

En la clase DAOProductoImpl.java

Declaración del Logger (Variable: LOGGER):

```
private final Logger LOGGER = UtilsLoggerManager.getLogger(DAOProductoImpl.class);
```

Para el control de errores

```
        dtoProducto.setIdProducto(rs.getInt(1)); // referencia por la posicion de la columna tambien puede ir el nombre
        dtoProducto.setNombre(rs.getString(2));
        dtoProducto.setIdCategoria(rs.getInt(3));
        dtoProducto.setUndMedida(rs.getString(4));
        dtoProducto.setStock(rs.getInt(5));

        lista.add(dtoProducto); // aqui se carga todos los productos ARRAYLIST

    }
    cerrarConexion();

} catch (Exception e) {
    LOGGER.error("Error en listar en la tabla {} : {}", TABLA, e.getMessage(), ExceptionUtils.getStackTrace(e));
}

//Metodo agregar
@Override
public void agregarProducto(DTOProducto producto) {
    String sql = "INSERT INTO " + TABLA + " (nombre,idCategoria,undMedida,Stock) VALUES (?, ?, ?, ?)";
    try {
        con = conexion.getConnection();
        Preconditions.checkNotNull(con, "La conexión no debe de ser nula.");
        ps = con.prepareStatement(sql);

        ps.setString(1, producto.getNombre());
        ps.setInt(2, producto.getIdCategoria());
        ps.setString(3, producto.getUndMedida());
        ps.setInt(4, producto.getStock());

        ps.executeUpdate(); // EJECUTA LA CONSULTA O INSERCIÓN SQL UPDATE
        cerrarConexion();

    } catch (SQLException e) {
        LOGGER.error("Error en agregar en la tabla {} : {}", TABLA, e.getMessage(), ExceptionUtils.getStackTrace(e));
    }
} //TERMINA EL METODO AGREGAR
```

```

@Override
public void actualizarProducto(DTOProducto producto) {
    String sql = "UPDATE " + TABLA + " SET nombre = ?, idCategoria = ?, undMedida = ?, Stock = ? WHERE id = ?";
    try {
        con = conexion.getConnection();
        Preconditions.checkNotNull(con, "La conexión no debe de ser nula.");
        ps = con.prepareStatement(sql);

        ps.setString(1, producto.getNombre());
        ps.setInt(2, producto.getIdCategoria());
        ps.setString(3, producto.getUndMedida());
        ps.setInt(4, producto.getStock());

        ps.setInt(5, producto.getIdProducto());

        ps.executeUpdate();
        cerrarConexion();

    } catch (SQLException e) {
        LOGGER.error("Error en actualizar la tabla {} : {}", TABLA, e.getMessage(), ExceptionUtils.getStackTrace(e));
    }
}

@Override
public void eliminarProducto(int idProducto) {
    String sql = "DELETE FROM " + TABLA + " WHERE id = ?";
    try {
        con = conexion.getConnection();
        Preconditions.checkNotNull(con, "La conexión no debe de ser nula.");
        ps = con.prepareStatement(sql);

        ps.setInt(1, idProducto);

        ps.executeUpdate();
        cerrarConexion();
    } catch (SQLException e) {
        LOGGER.error("Error en eliminar de la tabla {} : {}", TABLA, e.getMessage(), ExceptionUtils.getStackTrace(e));
    }
}

ps.setString(1, param);
rs = ps.executeQuery();

if (rs.next()) {
    dtoProducto = new DTOProducto();
    dtoProducto.setIdProducto(rs.getInt("id"));
    dtoProducto.setNombre(rs.getString("nombre"));
    dtoProducto.setIdCategoria(rs.getInt("idCategoria"));
    dtoProducto.setUndMedida(rs.getString("undMedida"));
    dtoProducto.setStock(rs.getInt("Stock"));
}
} catch (SQLException e) {
    LOGGER.error("Error en obtener un producto por nombre. ", ExceptionUtils.getStackTrace(e));
}

```

```

        while (rs.next()) {
            dtoProducto = new DTOProducto()
                .setIdProducto(rs.getInt("id"))
                .setNombre(rs.getString("nombre"))
                .setIdCategoria(rs.getInt("idCategoria"))
                .setUndMedida(rs.getString("undMedida"))
                .setStock(rs.getInt("Stock"));
        }
    } catch (SQLException e) {
        LOGGER.error("Error en obtener un producto por Id. ", ExceptionUtils.getStackTrace(e));
    } finally {
        cerrarConexion();
    }
    return dtoProducto;
}

private void cerrarConexion() {
    try {
        con.close();
        ps.close();
    } catch (SQLException e) {
        LOGGER.error("Error en cerrar conexión. ", ExceptionUtils.getStackTrace(e));
    }
}

```

## 6.5.Gestión de Usuarios

El logger se utiliza en diversas clases del proyecto para rastrear errores, validar procesos y registrar eventos importantes entorno al usuario. Su implementación permite un control detallado en actividades como validación de credenciales, creación de documentos, inicio de sesión, manejo de registros en la base de datos y gestión de usuarios (búsqueda, edición y eliminación).

Declaración del Logger (Variable: LOGGER):

En la clase ViewRegistroUsuario.java

```
private final org.slf4j.Logger LOGGER = UtilsLoggerManager.getLogger(ViewRegistroUsuario.class);
```

Uso en el constructor de la clase:

```

    /**
     * Constructor de la clase viewRegistroUsuario. Inicializa el panel con un
     * color de fondo específico y carga los componentes.
     */
    public ViewRegistroUsuario() {
        this.setBackground(Color.decode("#000511"));
        initComponents();
        List<DTOUsuario> usuarios = null;
        try {
            usuarios = new ControllerUsuario().obtenerUsuarios();
        } catch (SQLException ex) {
            Logger.getLogger(ViewRegistroUsuario.class.getName()).log(Level.SEVERE, null, ex);
        }
        mostrarUsuariosPorCriterios(usuarios);
    }
}

```

Para control de errores en la validación de credenciales del usuario:

```

try {
    validarUsername(username);
    validarDocumento(documento);
    validarNombre(nombre);
} catch (IllegalArgumentException e) {
    String errorMessage = e.getMessage();
    JOptionPane.showMessageDialog(null, errorMessage, "Error de Validación", JOptionPane.ERROR_MESSAGE);
    LOGGER.error("Error de validación de entrada: " + errorMessage);
    return; // Salir si hay un error de validación
}
if (mensajeError != null) {
    JOptionPane.showMessageDialog(null, mensajeError);
    LOGGER.error("Error inesperado");
    return;
}

```

Para el control de la creación y nombrado del documento Excel:

```

try {
    ControllerUsuario controllerUser = new ControllerUsuario();

    if (selecArchivo.showDialog(null, "Exportar") == JFileChooser.APPROVE_OPTION) {
        archivo = selecArchivo.getSelectedFile();
        String fileName = archivo.getName().toLowerCase();

        if (fileName.endsWith("xls") || fileName.endsWith("xlsx")) {
            String resultado = controllerUser.exportarExcel(archivo, tablaUsuarios);
            JOptionPane.showMessageDialog(null, resultado + "\nFormato ." + fileName.substring(fileName.lastIndexOf(".") + 1));
            LOGGER.info("Formato elegido correctamente. (" + fileName.substring(fileName.lastIndexOf(".") + 1) + ")");
        } else {
            JOptionPane.showMessageDialog(null, "Elija un formato válido.");
            LOGGER.error("Formato de exportación incorrecto.");
        }
    }
} catch (Exception e) {
    JOptionPane.showMessageDialog(null, "Ocurrió un error durante la exportación: " + e.getMessage());
}

```

En la clase DAOUsuarioImpl.java

Definir el logger:

```
private final Logger LOGGER = UtilsLoggerManager.getLogger(DAOUsuario.class);
```

Control de errores y éxitos en el inicio de sesión y registro con la base de datos:

```

@Override
public boolean login(String usuario, String contrasena, int tipoUsuario) throws SQLException {
    boolean resultado = false;
    String sql = "SELECT * FROM usuario WHERE username=? AND password_usuario=? AND id_tipo_usuario=?";
    try (PreparedStatement ps = connection.prepareStatement(sql)) {
        ps.setString(1, usuario);
        ps.setString(2, contrasena);
        ps.setInt(3, tipoUsuario);
        ResultSet rs = ps.executeQuery();
        resultado = rs.next();
        rs.close();
    } catch (SQLException e) {
        LOGGER.error("Hubo un problema en la autenticación del usuario. " + e.getMessage(), e);
    }
    return resultado;
}

@Override
public boolean registrarUsuario(DTOUsuario usuario) throws SQLException {
    boolean registroExitoso = false;
    String sql = "INSERT INTO usuario (id_usuario, nombre_usuario, apellido_usuario, documento_usuario, "
            + "VALUES (NULL, ?, ?, ?, ?, ?, ?, ?, ?, ?)";
    try (PreparedStatement ps = connection.prepareStatement(sql)) {
        ps.setString(1, usuario.getNombre());
        ps.setString(2, usuario.getApellido());
        ps.setString(3, usuario.getDocumento());
        ps.setString(4, usuario.getDireccion());
        ps.setString(5, usuario.getTelefono());
        ps.setString(6, usuario.getCorreo());
        ps.setInt(7, usuario.getIdTipoUsuario());
        ps.setString(8, usuario.getUsername());
        ps.setString(9, usuario.getPassword());

        int rowsAffected = ps.executeUpdate();
        registroExitoso = rowsAffected > 0;
        LOGGER.info("Se registró exitosamente un usuario.");
    } catch (SQLException e) {
        LOGGER.error("Hubo un problema en el registro de usuario. " + e.getMessage(), e);
    }
    return registroExitoso;
}

```

Para el control de errores y aciertos en la búsqueda de usuario:

```

try (PreparedStatement ps = connection.prepareStatement(sql.toString())) {
    // Asignar los parámetros al PreparedStatement
    for (int i = 0; i < parametros.size(); i++) {
        ps.setObject(i + 1, parametros.get(i));
    }

    ResultSet rs = ps.executeQuery();

    while (rs.next()) {
        DTOUsuario usuario = new DTOUsuario(
            rs.getInt("id_usuario"),
            rs.getString("nombre_usuario"),
            rs.getString("apellido_usuario"),
            rs.getString("documento_usuario"),
            rs.getString("direccion_usuario"),
            rs.getString("telefono_usuario"),
            rs.getString("correo_usuario"),
            rs.getInt("id_tipo_usuario"),
            rs.getString("username"),
            rs.getString("password_usuario")
        );
        listaUsuarios.add(usuario);
    }
    rs.close();
    LOGGER.info("Se recuperó correctamente los usuarios.");
} catch (SQLException e) {
    LOGGER.error("Hubo un problema en recuperar los usuarios por criterio. " + e.getMessage(), e);
}

return listaUsuarios;

```

Para eliminar usuario:

```

@Override
public boolean eliminarUsuarioPorDocumento(String documento) throws SQLException {
    boolean eliminado = false;
    String sql = "DELETE FROM usuario WHERE documento_usuario = ?";

    try (PreparedStatement ps = connection.prepareStatement(sql)) {
        ps.setString(1, documento);
        int rowsAffected = ps.executeUpdate();
        eliminado = rowsAffected > 0;
        LOGGER.info("Se elimino correctamente a un usuario.");
    } catch (SQLException e) {
        LOGGER.error("Hubo un problema en eliminar un usuario por documento. " + e.getMessage(), e);
    }

    return eliminado;
}

```

Para editar usuarios:

```
 /**
 * Override
 */
public boolean editarUsuario(DTOUsuario usuario) throws SQLException {
    boolean actualizado = false;
    String sql = "UPDATE usuario SET nombre_usuario = ?, apellido_usuario = ?, direccion_usuario = ?, t";

    try (PreparedStatement ps = connection.prepareStatement(sql)) {
        ps.setString(1, usuario.getNombre());
        ps.setString(2, usuario.getApellido());
        ps.setString(3, usuario.getDireccion());
        ps.setString(4, usuario.getTelefono());
        ps.setString(5, usuario.getCorreo());
        ps.setInt(6, usuario.getIdTipoUsuario());
        ps.setString(7, usuario.getUsername());
        ps.setString(8, usuario.getPassword());
        ps.setString(9, usuario.getDocumento());

        int rowsAffected = ps.executeUpdate();
        actualizado = rowsAffected > 0;
        LOGGER.info("Se actualizo correctamente a un usuario.");
    } catch (SQLException e) {
        LOGGER.error("Hubo un problema en actualizar un usuario en especifico. " + e.getMessage(), e);
    }

    return actualizado;
}
```

## 6.6.Clase DAOSalidaImpl

El logger se utiliza para validar procesos y registrar eventos importantes relacionados con las salidas. Su implementación permite un control detallado en actividades como la validación de IDs, creación de salidas, edición de información, manejo de registros en la base de datos y gestión de las salidas.

En la clase DAOSalidaImpl.java

Declaración del Logger (Variable: LOGGER):

```
private static final Logger LOGGER = UtilsLoggerManager.getLogger(DAOSalida.class);
```

Para manejo y registro de información y errores

```
public DAOSalidaImpl(Connection connection) {
    this.connection = connection;
    LOGGER.info("Instancia de DAOSalidaImpl creada con conexión a la base de datos.");
}

@Override
public List<DTOSalida> obtenerTodasLasSalidas() {
    LOGGER.info("Obteniendo todas las salidas de la base de datos.");
    List<DTOSalida> salidas = new ArrayList<>();
    String sql = "SELECT * FROM Salida";
    try (Statement stmt = connection.createStatement()) {
        ResultSet rs = stmt.executeQuery(sql));
        while (rs.next()) {
            salidas.add(mapResultSetToSalida(rs));
        }
        LOGGER.info("Se obtuvieron {} salidas correctamente.", salidas.size());
    } catch (SQLException e) {
        LOGGER.error("Error al obtener todas las salidas: {}", e.getMessage(), e);
    }
    return salidas;
}

@Override
public DTOSalida obtenerSalidaPorId(int idSalida) {
    LOGGER.info("Obteniendo la salida con ID: {}", idSalida);
    String sql = "SELECT * FROM Salida WHERE idSalida = ?";
    try (PreparedStatement stmt = connection.prepareStatement(sql)) {
        stmt.setInt(1, idSalida);
        ResultSet rs = stmt.executeQuery();
        if (rs.next()) {
            LOGGER.info("Se encontró la salida con ID: {}", idSalida);
            return mapResultSetToSalida(rs);
        } else {
            LOGGER.warn("No se encontró ninguna salida con el ID: {}", idSalida);
        }
    } catch (SQLException e) {
        LOGGER.error("Error al obtener la salida con ID {}: {}", idSalida, e.getMessage(), e);
    }
    return null;
}
```

```

@Override
public int guardarSalida(DTOSalida dtoSalida) {
    LOGGER.info("Insertando una nueva salida con ID: {}", dtoSalida.getIdSalida());
    String sql = "INSERT INTO Salida (idSalida, idProducto, cantidad, valorUnitario, valorTotal, fechaSalida) VALUES (?, ?, ?, ?, ?, ?)";
    try (PreparedStatement stmt = connection.prepareStatement(sql, Statement.RETURN_GENERATED_KEYS)) {
        stmt.setInt(1, dtoSalida.getIdSalida());
        stmt.setInt(2, dtoSalida.getDtoProducto().getIdProducto());
        stmt.setInt(3, dtoSalida.getCantidad());
        stmt.setDouble(4, dtoSalida.getValorUnitario());
        stmt.setDouble(5, dtoSalida.getValorTotal());
        stmt.setTimestamp(6, Timestamp.valueOf(dtoSalida.getFechaSalida()));
        int affectedRows = stmt.executeUpdate();
        if (affectedRows > 0) {
            ResultSet generatedKeys = stmt.getGeneratedKeys();
            if (generatedKeys.next()) {
                int generatedId = generatedKeys.getInt(1);
                LOGGER.info("Salida insertada correctamente con ID generado: {}", generatedId);
                return generatedId;
            }
        }
    } catch (SQLException e) {
        LOGGER.error("Error al insertar la salida: {}", e.getMessage(), e);
    }
    return -1;
}

@Override
public boolean actualizarSalida(DTOSalida dtoSalida) {
    LOGGER.info("Actualizando la salida con ID: {}", dtoSalida.getIdSalida());
    String sql = "UPDATE Salida SET idProducto = ?, cantidad = ?, valorUnitario = ?, valorTotal = ?, fechaSalida = ? WHERE idSalida = ?";
    try (PreparedStatement stmt = connection.prepareStatement(sql)) {
        stmt.setInt(1, dtoSalida.getDtoProducto().getIdProducto());
        stmt.setInt(2, dtoSalida.getCantidad());
        stmt.setDouble(3, dtoSalida.getValorUnitario());
        stmt.setDouble(4, dtoSalida.getValorTotal());
        stmt.setTimestamp(5, Timestamp.valueOf(dtoSalida.getFechaSalida()));
        stmt.setInt(6, dtoSalida.getIdSalida());
        boolean updated = stmt.executeUpdate() > 0;
        if (updated) {
            LOGGER.info("Salida con ID {} actualizada correctamente.", dtoSalida.getIdSalida());
        } else {
            LOGGER.warn("No se encontró ninguna salida para actualizar con el ID: {}", dtoSalida.getIdSalida());
        }
        return updated;
    } catch (SQLException e) {
        LOGGER.error("Error al actualizar la salida con ID {}: {}", dtoSalida.getIdSalida(), e.getMessage(), e);
    }
    return false;
}

@Override
public boolean eliminarSalida(int idSalida) {
    LOGGER.info("Eliminando la salida con ID: {}", idSalida);
    String sql = "DELETE FROM Salida WHERE idSalida = ?";
    try (PreparedStatement stmt = connection.prepareStatement(sql)) {
        stmt.setInt(1, idSalida);
        boolean deleted = stmt.executeUpdate() > 0;
        if (deleted) {
            LOGGER.info("Salida con ID {} eliminada correctamente.", idSalida);
        } else {
            LOGGER.warn("No se encontró ninguna salida para eliminar con el ID: {}", idSalida);
        }
        return deleted;
    } catch (SQLException e) {
        LOGGER.error("Error al eliminar la salida con ID {}: {}", idSalida, e.getMessage(), e);
    }
    return false;
}

```

```

private DTOSalida mapResultSetToSalida(ResultSet rs) throws SQLException {
    DTOSalida salida = new DTOSalida();
    salida.setIdSalida(rs.getInt("idSalida"));

    DTOProyecto producto = new DTOProyecto();
    producto.setIdProducto(rs.getInt("idProducto"));
    salida.setDtoProducto(producto);

    DTOUsuario usuario = new DTOUsuario();
    usuario.setIdUsuario(rs.getInt("idUsuario"));
    salida.setDtoUsuario(usuario);

    salida.setCantidad(rs.getInt("cantidad"));
    salida.setValorUnitario(rs.getDouble("valorUnitario"));
    salida.setValorTotal(rs.getDouble("valorTotal"));
    salida.setFechaSalida(rs.getTimestamp("fechaSalida").toLocalDateTime());

    LOGGER.debug("Mapeado ResultSet a DTOSalida: {}", salida);
    return salida;
}

```

## • Ejecutar Pruebas

El proyecto contiene pruebas unitarias (JUnit), puedes ejecutarlas en NetBeans:

Haz clic **derecho** en el proyecto y selecciona Test.

## 7. Mantenimiento

### 7.1. Tipos de mantenimiento:

#### 7.1.1. Preventivo

Tipo de mantenimiento	Estrategia	Descripción
Mantenimiento preventivo	Update y/o actualizaciones	Se ha considerado llevar a cabo actualizaciones software y hardware de manera esporádica para corregir vulnerabilidades y mejora del sistema.
	Pruebas y testing del sistema	Se harán pruebas unitarias de manera periódica con el objetivo de mantener en funcionamiento del sistema de control de stock.

<b>Documentación</b>	Se cuenta con java-doc que describe y/o comenta los procesos que realiza cada método y líneas de código dentro del software y se mantendrá que esta práctica conforme se actualice el sistema a futuro.
----------------------	---

### 7.1.2. Adaptable

Tipo de mantenimiento	Estrategia	Descripción
<b>Mantenimiento adaptativo</b>	<b>Cambios en los requerimientos del software</b>	El sistema desarrollado podrá adaptarse a cambios como parte de actualizaciones y/o nuevos requerimientos por lo mismo que el sistema considera buenas prácticas y metodología como el patrón de diseño mvc, dao y dto lo que facilitaría si se requiere añadir nuevos procedimientos de control de inventario. Así mismo podría adaptarse a cualquier otro sistema como parte de su integración.
	<b>Nuevas tecnologías</b>	Se estará a la vanguardia de nuevas tecnologías para este tipo de sistema de escritorio. Así mismo también se puede adaptar o convertir a un sistema de control de stock

		web con almacenamiento en la nube por lo que mejorar tanto su funcionalidad y rendimiento, facilitando a los usuarios poder contar y/o consultar registro desde su smartphone o cualquier portátil que cuente con internet.
--	--	---

### 7.1.3. Correctivo

Tipo de mantenimiento	Estrategia	Descripción
Mantenimiento correctivo	<b>Control de incidentes</b>	Los usuarios podrán acceder a un sistema de tickets para reportar y solucionar problemas de forma rápida y eficiente, así mismo podrán hacer uso de un bot que recepcione los tickets desde whatsapp.
	<b>Respaldos periódicos-copia de seguridad</b>	Se tendrán backups que serán realizados de manera periódica por lo que la base de datos bd contara con un respaldo ante fallas o ciberataques que vulneren la bd.
	<b>Monitoreos periódicos</b>	Se contará con un sistema de monitoreo periódico para la detección de fallas bugs que puedan presentarse en el sistema.

#### 7.1.4. Perfectivo o evolutivo

Tipo de mantenimiento	Estrategia	Descripción
Mantenimiento perfectivo	<b>Nuevas funcionalidades para el sistema de control de stock</b>	Se contará con asistencia de línea abierta para que los usuarios y/o empresa que contrató el servicio del sistema considere implementar nuevas funcionalidades que se adapten a sus necesidades o integrar a algún otro sistema como la integración de un erp.
	<b>Optimización de código fuente</b>	Se tiene una política de mejora continua por lo que se busca la optimización de algoritmos que optimice el rendimiento del sistema eliminando malas prácticas de programación.
	<b>Usabilidad ux-ui</b>	Se considera siempre que el software – sistema de control de stock sea lo más intuitivo posible para que el usuario pueda interactuar con el sistema sin ninguna dificultad aun así se está abierto a realizar ajustes en la interfaz de usuario periódicamente según especificaciones o mejora de interfaz.

## 7.2. Ciclo de mantenimiento



## 7.3. Herramientas y procesos

### 7.3.1. Herramientas que se usarán

#### 1. Cron Jobs:

- Para la automatización de tareas periódicas, como la creación de respaldos y la ejecución de scripts de mantenimiento.

#### 2. Sistema de Backups:

- Herramientas de respaldo automatizado para asegurar la integridad y disponibilidad de los datos, como scripts personalizados de MySQL o servicios en la nube.

#### 3. Monitores de Logs (Ejemplo: Logwatch):

- Para el seguimiento continuo de errores, eventos y estadísticas del sistema.

#### 4. Controladores de Actualizaciones (Git y Github):

- Para gestionar versiones del código y aplicar actualizaciones futuras con seguridad.

### **7.3.2. En qué procesos están involucrados**

#### **1. Identificación de Necesidades y Funcionalidades (Fase 1):**

- Uso de monitores de logs para registrar errores y recopilar solicitudes de nuevas funcionalidades.

#### **2. Planificación (Fase 2):**

- Herramientas como Git y Github para organizar las actualizaciones necesarias según prioridades.
- Cron Jobs para planificar tareas automatizadas como respaldos y verificaciones.

#### **3. Ejecución (Fase 3):**

- Uso de scripts programados con Cron Jobs para ejecutar las tareas de mantenimiento.
- Implementación de las mejoras y corrección de errores detectados.

#### **4. Verificación (Fase 4):**

- Validación de los respaldos generados para garantizar su recuperación.
- Monitoreo de logs para asegurar que los procesos automatizados funcionen correctamente.

#### **5. Evaluación (Fase 5):**

- Análisis de los resultados obtenidos con las herramientas utilizadas, asegurando que se cumplan los requerimientos planteados.

#### **6. Seguimiento (Fase 6):**

- Actualización de las tareas automatizadas según los nuevos requerimientos y mantenimiento continuo con Cron Jobs.

### **7.3.3. Describir al detalle cómo contribuyen al proceso.**

#### **1. Cron Jobs:**

- Facilitan la ejecución programada de tareas críticas como respaldos automáticos y limpieza de logs antiguos.
- Ayudan a minimizar la intervención manual, mejorando la eficiencia del mantenimiento.

#### **2. Sistema de Backups:**

- Asegura la protección de datos ante fallos del sistema o errores humanos mediante copias periódicas.
- Contribuye a la rápida recuperación de información, minimizando el impacto en la operación del sistema.

#### **3. Monitores de Logs:**

- Detectan errores en tiempo real y generan alertas automáticas para la pronta resolución de problemas.
- Proveen información detallada sobre el rendimiento del sistema, ayudando a identificar áreas de mejora.

#### **4. Git (o Github):**

- Permite rastrear cambios en el código, realizar pruebas antes de implementar actualizaciones y restaurar versiones anteriores en caso de fallos.
- Mejora la organización del trabajo y asegura la integridad del sistema a lo largo del tiempo.

## 8. Control de Versiones

### 8.1. README(Instalación)

The screenshot shows a GitHub repository page for a project titled "Proyecto-Final-Curso-Integrador-I-Sistemas-Software | Sistema Control de Stock". The README file is open, displaying the following content:

Este proyecto implementa un sistema de control de stock versátil, adecuado para diversas aplicaciones y entornos.

Para poder instalar el proyecto en un entorno de desarrollo se tiene el siguiente documento como referencia: [Guía de Instalación](#)

#### Fase de Desarrollo (100%)

En esta fase se alcanzó un 100% del desarrollo del proyecto. Se han iniciado los trabajos y se han implementado varios casos de uso clave. Además, se han aplicado metodologías y patrones de diseño para asegurar un desarrollo sólido.

#### Total de Casos de Uso

- CU01 Registrar Usuario
- CU02 Autenticar Usuario
- CU03 Iniciar Sesión
- CU04 Registrar productos
- CU05 Modificar productos
- CU06 Eliminar productos
- CU07 Visualizar productos
- CU08 Registrar entrada de stock
- CU09 Registrar salida de stock
- CU10 Visualizar estadísticas totales

## Diagrama de casos de uso del Sistema de control de Stock



## 8.2.Instalación del proyecto

### 8.2.1. Requisitos Previos

#### Instalación del Proyecto

Este documento describe cómo instalar y ejecutar el proyecto Java utilizando Apache Ant y NetBeans.

#### Requisitos Previos

Asegúrate de tener instalados los siguientes elementos en tu sistema antes de continuar con la instalación del proyecto:

- Java Development Kit (JDK) versión 20
  - [Descargar JDK 20](#)
- NetBeans IDE versión 12 o superior (recomendado versión 20)
  - [Descargar NetBeans](#)

### 8.2.2. Clonar el repositorio

#### Clonar el Repositorio

Primero, clona el repositorio en tu máquina local:

```
git clone https://github.com/Emmanuel-Rivera-G/Proyecto-Final-Curso-Integrador-I-Sistemas-Software.git
```

Luego, navega a la carpeta del proyecto:

```
cd Proyecto-Final-Curso-Integrador-I-Sistemas-Software
```

### 8.2.3. Configuración del Proyecto en NetBeans

#### Configuración del Proyecto en NetBeans

1. Abre NetBeans IDE.
2. Selecciona `File > Open Project....`
3. Navega a la carpeta donde clonaste el repositorio y selecciona el proyecto.
4. Haz clic en `Open Project` para cargar el proyecto en NetBeans. **Nota:** Asegurate de tener las dependencias en la carpeta `lib`.

### 8.2.4. Compilar el Proyecto

#### Compilar el Proyecto

Para compilar el proyecto en NetBeans:

1. En el panel `Projects`, haz clic derecho sobre el proyecto.
2. Selecciona `Build` o `Clean and Build` para compilar todo el proyecto.

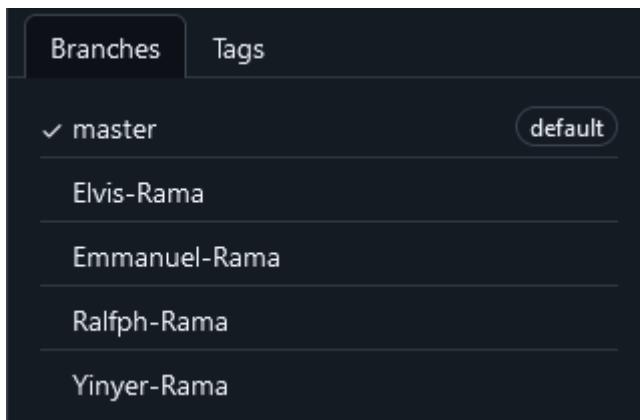
### 8.2.5. Ejecutar el Proyecto

#### Ejecutar el Proyecto

Para ejecutar el proyecto en NetBeans:

1. Haz clic derecho en el proyecto desde el panel `Projects`.
2. Selecciona `Run` o utiliza el atajo de teclado `F6`. **Nota:** La clase principal es `ViewInicioSesion.java` ubicada en `view/`.

### 8.3.Ramas



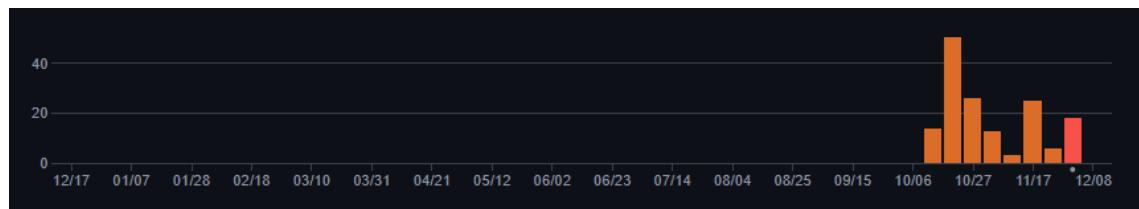
Issues del proyecto realizados según los casos de uso

<input type="checkbox"/>	<a href="#">CU10: Visualizar estadísticas totales</a> <small>(back-end documentation front-end)</small>	<small>#65 by Emmanuel-Rivera-G was closed last week</small>	<small>↑↑ 1</small>	
<input type="checkbox"/>	<a href="#">CU09 - Registro de salida de Stock (FrontEnd)</a> <small>(front-end)</small>	<small>#54 by Emmanuel-Rivera-G was closed last month</small>	<small>↑↑ 1</small>	
<input type="checkbox"/>	<a href="#">CU08 - Registrar entrada de Stock (FrontEnd)</a> <small>(front-end)</small>	<small>#53 by Emmanuel-Rivera-G was closed last month</small>	<small>↑↑ 1</small>	
<input type="checkbox"/>	<a href="#">CU07 Visualizar Productos (FrontEnd)</a> <small>(front-end)</small>	<small>#48 by Emmanuel-Rivera-G was closed on Nov 6</small>	<small>↑↑ 1</small>	
<input type="checkbox"/>	<a href="#">Aregar documentación de Javadoc para el proyecto.</a> <small>(documentation)</small>	<small>#25 by Emmanuel-Rivera-G was closed 2 days ago</small>	<small>↑↑ 1</small>	
<input type="checkbox"/>	<a href="#">Aregar documentación de Markdown necesaria</a> <small>(documentation)</small>	<small>#21 by Emmanuel-Rivera-G was closed 3 days ago</small>	<small>↑↑ 1</small>	
<input type="checkbox"/>	<a href="#">Aregar las librerías de Apache Commons y Apache POI</a> <small>(dependency)</small>	<small>#20 by Emmanuel-Rivera-G was closed 5 days ago</small>	<small>↑↑ 2</small>	
<input type="checkbox"/>	<a href="#">Aregar la librería para el uso de LogBack y Corrección de errores</a> <small>(back-end bug)</small>	<small>#18 by Emmanuel-Rivera-G was closed last week</small>	<small>↑↑ 1</small>	
<input type="checkbox"/>	<a href="#">CU01 Registrar Usuario (Backend)</a> <small>(back-end)</small>	<small>#15 by Emmanuel-Rivera-G was closed last week</small>	<small>↑↑ 1</small>	
<input type="checkbox"/>	<a href="#">CU02 - 03 Autenticar usuario e Iniciar Sesión (Backend)</a> <small>(back-end)</small>	<small>#14 by Emmanuel-Rivera-G was closed last week</small>	<small>↑↑ 1</small>	
<input type="checkbox"/>	<a href="#">Caso de Uso 4-5-6: Implementar el modelo de Producto y el controlador de este ControllerProducto (Backend)</a> <small>(back-end)</small>	<small>#10 by Emmanuel-Rivera-G was closed last week</small>	<small>↑↑ 2</small>	
<input type="checkbox"/>	<a href="#">Caso de Uso 4-5-6: Implementar el DTOProducto y el Service del mismo para su manejo (Backend)</a> <small>(back-end)</small>	<small>#9 by Emmanuel-Rivera-G was closed 2 weeks ago</small>	<small>↑↑ 1</small>	
<input type="checkbox"/>	<a href="#">Caso de Uso 4-5-6: Implementar la Conexión con la base de datos y el DAOProducto necesario (Backend)</a> <small>(back-end)</small>	<small>#8 by Emmanuel-Rivera-G was closed 2 weeks ago</small>	<small>↑↑ 1</small>	
<input type="checkbox"/>	<a href="#">Caso de Uso 4-5-6: Registrar-Modificar-Eliminar Producto (Frontend)</a> <small>(front-end)</small>	<small>#4 by Emmanuel-Rivera-G was closed 2 weeks ago</small>	<small>↑↑ 1</small>	
<input type="checkbox"/>	<a href="#">Caso de Uso 3: Iniciar Sesión (Frontend)</a> <small>(front-end)</small>	<small>#3 by Emmanuel-Rivera-G was closed 2 weeks ago</small>	<small>↑↑ 1</small>	
<input type="checkbox"/>	<a href="#">Caso de Uso 1: Registrar Usuario (Frontend)</a> <small>(front-end)</small>	<small>#2 by Emmanuel-Rivera-G was closed 2 weeks ago</small>	<small>↑↑ 1</small>	
<input type="checkbox"/>	<a href="#">Especificar casos de uso 40% README</a> <small>(documentation)</small>	<small>#1 by Emmanuel-Rivera-G was closed 2 weeks ago</small>	<small>↑↑ 1</small>	

## 8.4. Gráfico de barras



## Commits a lo largo del tiempo



## Frecuencia de código



## 8.5.Commit de cada integrante

Figura

Commit de Ralphy Sardon

The screenshot shows a GitHub commit history for a repository. At the top, there's a dropdown for 'Ralphy-Rama' and a date range selector for 'All time'. Below this, a single commit is highlighted from December 3, 2024, with the message: 'Resuelve el caso de uso CU10: Visualizar estadísticas totales #65. Se implementó el front-end y back-end para mostrar estadísticas clave diferenciadas para administradores y empleados. Se diseñó un...'. This commit was made by 'ralphyg committed last week' and has a hash '8d88e3c'. Below this, there are several other commit entries grouped by date: November 27, 2024; November 24, 2024; November 12, 2024; November 6, 2024; and October 29, 2024. Each group contains one or more commits with their respective messages, dates, authors, and hash codes.

- o- Commits on Dec 3, 2024
  - Resuelve el caso de uso CU10: Visualizar estadísticas totales #65. Se implementó el front-end y back-end para mostrar estadísticas clave diferenciadas para administradores y empleados. Se diseñó un... 8d88e3c
- o- Commits on Nov 27, 2024
  - Creación de vista principal para usuario normal (empleado). Actualización de vistas de inicio de sesión, registro y listado de salidas para manejo de roles. Además, se añadió la documentación para... 27107dd
- o- Commits on Nov 24, 2024
  - Ajustes menores en las pruebas de ServiceUsuarioTest.java 7d45e7c
- o- Commits on Nov 12, 2024
  - Resuelve el issue #54: Registrar salida de Stock en FrontEnd. Asimismo, se agrega un panel de listado de salidas y el panel de registro de proveedores. c99f0ce
  - Resuelve el issue #53: Registrar entrada de Stock en FrontEnd. Además, se modifica el panel principal para asegurar la visualización del nuevo panel. 0afffd9
- o- Commits on Nov 6, 2024
  - Implementación de gráficos circulares y dinámicos para mostrar información relevante al administrador (gerente, jefe, etc.). Se agregó la librería DesktopTiming, así como nuevos paquetes y clases p... 685c78f
- o- Commits on Oct 29, 2024
  - Implementación de google Guava, para el registro de usuarios 271da85

The screenshot shows a GitHub commit history for a project. The commits are organized by date:

- Commits on Oct 28, 2024:
  - Pequeños cambios dentro de las interfaces** (773bbc7) - A commit by ralf8g 13 hours ago.
- Commits on Oct 26, 2024:
  - Resuelve una parte del issue #25 - Documentación en Javadoc: Se agregaron y actualizaron comentarios en las siguientes clases:** (c801c61) - A commit by ralf8g 3 days ago.
- Commits on Oct 23, 2024:
  - Implementacion de metodo exportarExcel, usando la libreria apache POI** (04facd6) - A commit by ralf8g 5 days ago.
  - Issue #20 - Agregacion de libreria apache POI** (80e1945) - A commit by ralf8g 5 days ago.
- Commits on Oct 20, 2024:
  - Resuelve el issue #15: CU01 Registrar Usuario (Backend) - Implementación de la lógica de registro de usuarios, eliminación, modificación, búsqueda y guardado.** (e4b8802) - A commit by ralf8g last week.
  - Resuelve el issue #14: CU02 - 03: Autenticar usuario e Iniciar Sesión (Backend) - Implementación del login y autenticación de usuarios.** (8e74c96) - A commit by ralf8g last week.
- Commits on Oct 15, 2024:
  - Resuelve el issue #14: CU02 - 03: Autenticar usuario e Iniciar Sesión (Backend) - Implementación del login y autenticación de usuarios.** (70b8211) - A commit by ralf8g last week.
  - Resuelve el issue #2 - Crear interfaz grafica para la administracion de usuarios del sistema** (c4c3e55) - A commit by ralf8g 2 weeks ago.
  - Resuelve el issue #4 - Crear interfaz grafica para la gestion de Productos** (1d8706b) - A commit by ralf8g 2 weeks ago.
  - Resuelve el issue #3 - Crear interfaz gráfica para el Inicio de Sesión** (266627a) - A commit by ralf8g 2 weeks ago.

Figura

Commit de Emmanuel Rivera

## Commits

Emmanuel-Rama	All time
Commits on Dec 6, 2024	
<b>Ejecutable</b> Emmanuel-Rivera-G committed 5 days ago	22ee0c0
<b>Actualizar db</b> Emmanuel-Rivera-G committed 5 days ago	0787e32
<b>Actualizar todas la características para correcto funcionamiento.</b> Emmanuel-Rivera-G committed 5 days ago	365a5a4
<b>Merge origin/Emmanuel-Rama into Emmanuel-Rama</b> Emmanuel-Rivera-G committed 5 days ago	b6f13e2
<b>Actualizar todas la características para correcto funcionamiento.</b> Emmanuel-Rivera-G committed 5 days ago	a74e4ae

Commits on Dec 4, 2024	
<b>Agregar librería</b> Emmanuel-Rivera-G committed last week	50dc76f
<b>Merge pull request #68 from Emmanuel-Rivera-G/Emmanuel-Rama</b> Emmanuel-Rivera-G authored last week	<span>Verified</span> a51191b
<b>Configuración de la librería.</b> Emmanuel-Rivera-G committed last week	475b384
<b>Cambios y agregación de métodos y funcionamiento Salida</b> Emmanuel-Rivera-G committed last week	9e3f1d4
<b>Agregar métodos y controladores para el funcionamiento de la interfaz.</b> Emmanuel-Rivera-G committed last week	ffa4ac0
<b>Actualizar el script de la base de datos con salidas</b> Emmanuel-Rivera-G committed last week	254717a
<b>Agregando una librería</b> Emmanuel-Rivera-G committed last week	34a8d74

Actualización de DAOSalidalimpl para registro y control. Emmanuel-Rivera-G committed last week	f9f6779
Actualización en las utilidades para los productos. Emmanuel-Rivera-G committed last week	5fc373d
Actualización del script para el uso de la base de datos. Emmanuel-Rivera-G committed last week	0eb01a0

Commits on Oct 14, 2024	
<b>Creación de archivo .gitignore</b> Emmanuel-Rivera-G committed 2 weeks ago	ccfb24a
<b>Creación de la estructura de carpetas en base a los modelos MVC, DAO, SOLID</b> Emmanuel-Rivera-G committed 2 weeks ago	62b7ef0
<b>Creación de la estructura inicial de carpetas del Proyecto</b> Emmanuel-Rivera-G committed 2 weeks ago	e07b551
<b>Initial commit</b> Emmanuel-Rivera-G authored 2 weeks ago	<span>Verified</span> 589a03b

docs: Actualizar README con especificaciones de la primera fase Emmanuel-Rivera-G committed 2 weeks ago	8a331b3
--	---------

Resuelve el issue #9 - Crear la clase DTOProducto y ServiceProducto para el manejo de objetos Emmanuel-Rivera-G committed 2 weeks ago	2dbd7cb
--	---------

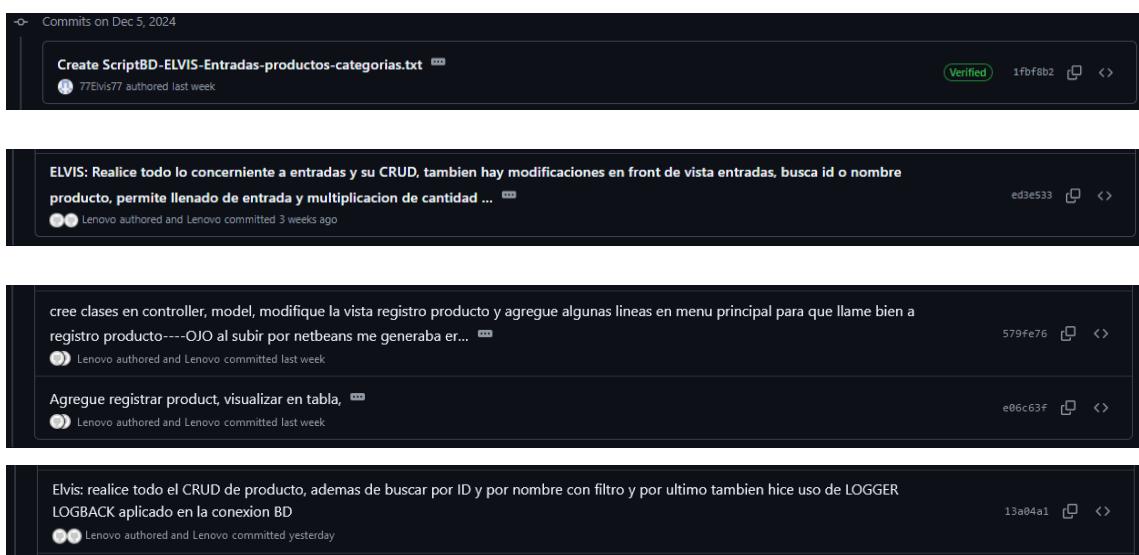
Añadiendo librerías para los tests y para los logs Emmanuel-Rivera-G committed last week	f89234a
---	---------

Añadir un test para la conexión con la base de datos. Emmanuel-Rivera-G committed last week	a54b82b
--	---------



Figura

## Commit de Elvis



## 8.6.Documentación

### Generar Javadoc

Para generar la documentación de Javadoc:

1. Abre NetBeans.
2. Haz clic derecho en el proyecto y selecciona Generate Javadoc.

La documentación generada estará disponible en la carpeta dist/javadoc.

# Contribuyendo al Proyecto

Gracias por tu interés en contribuir a este proyecto en Java. Este documento describe las pautas para contribuir al desarrollo del proyecto, incluyendo reglas de estilo, instrucciones de configuración y buenas prácticas. ¡Cualquier contribución, grande o pequeña, es bienvenida!

## Tabla de Contenidos

1. [Cómo empezar](#)
2. [Reportar problemas](#)
3. [Solicitudes de extracción \(Pull Requests\)](#)
4. [Estándares de Codificación](#)
5. [Pruebas](#)
6. [Documentación](#)

## Cómo empezar

Para empezar a contribuir, sigue estos pasos:

1. Haz un fork del repositorio en GitHub.
2. Clona tu fork en tu máquina local.

```
git clone https://github.com/tu-usuario/Proyecto-Final-Curso-Integrador-I-Sistemas-Software.git
```

3. Crea una rama nueva para tu contribución.

```
git checkout -b nombre-de-la-rama
```

4. Realiza los cambios en tu rama, asegúrandote de seguir las Pautas de Codificación.

5. Realiza commits descriptivos de tus cambios.

6. Realiza pruebas para asegurar que tus cambios no rompan nada.

7. Abre un pull request en el repositorio original y describe los cambios realizados.

## Reportar problemas

Si encuentras un error o tienes una sugerencia de mejora, sigue estos pasos:

1. Asegúrate de que el problema o sugerencia no haya sido reportado anteriormente.
2. Si el problema es nuevo, abre un issue en el repositorio.
3. Describe el problema con el mayor detalle posible, proporcionando el contexto necesario para entenderlo y reproducirlo; si es posible, incluye el código fuente e imágenes de muestra.

## Formato de Issues

Para agilizar la gestión de los issues, proporciona la siguiente información:

- **Descripción del problema:** Explica el error o la sugerencia.
- **Pasos para reproducir:** Describe los pasos para replicar el problema.
- **Sistema y versión:** Indica el sistema operativo y la versión de Java utilizada.
- **Logs y capturas de pantalla:** Incluye cualquier log o captura que pueda ser útil para identificar el problema.

## Solicitudes de extracción (Pull Requests)

Para realizar una solicitud de extracción (Pull Request), sigue estas pautas:

1. Asegúrate de que tu código sigue los Estándares de Codificación y que no introduce errores en el proyecto.
2. Si es un cambio significativo, abre un issue primero para discutir la propuesta.
3. Incluye una descripción clara de los cambios realizados.
4. Agrega pruebas que cubran la nueva funcionalidad o los cambios realizados.
5. Asocia tu Pull Request a un issue existente, si aplica.
6. Asegúrate de que tu código pase todas las pruebas antes de enviar el Pull Request.

## Estándares de Codificación

Por favor, sigue estos estándares de estilo:

- **Formato de Código:** Utiliza spaces o tabs según como en el resto del proyecto.
- **JavaDoc:** Añade JavaDoc en todos los métodos y clases públicas, además de las clases o métodos privados más utilizados.
- **Nombres de Clases y Métodos:** Usa nombres descriptivos y sigue la convención de CamelCase para clases y camelCase para métodos y variables; la estructura de los nombres de clase tiene que ser primero su el *general* y luego el *específico* (ejemplo: ControllerModelo).
- **Estructura de Paquetes:** Organiza el código en paquetes según la funcionalidad y asegúrate de que cada clase esté en el paquete correspondiente (No modificar la estructura de paquetes del proyecto).
- **Limpieza de Código:** No dejes código comentado o en desuso, asegurate de que no haya errores de estilo y elimina comentarios de plantillas.
- **Control de Errores:** Maneja todas las excepciones correctamente y usa logs (logback) para reportar errores, en lugar de System.out.println.

```
/*
 * Realiza una operación específica en el proyecto.
 *
 * @param parametro Un parámetro necesario para el método.
 * @return Resultado de la operación.
 * @throws MiExcepcion si ocurre un error durante la operación.
 */
public Resultado miMetodo(Tipo parametro) throws MiExcepcion {
    // Implementación
}
```

## Pruebas

Este proyecto utiliza JUnit para las pruebas unitarias. Asegúrate de que tus cambios pasen todas las pruebas antes de enviar un Pull Request. Si agregas una nueva funcionalidad, incluye pruebas unitarias para cubrirla.

Para ejecutar las pruebas, utiliza:

```
ant test
```

O

En netbeans presiona *click derecho* y ejecutar Test (Alt + F6).

### Especificaciones de pruebas

- **Cobertura:** Asegúrate de que la cobertura de las pruebas sea adecuada, especialmente en los métodos públicos.
- **Nombres Descriptivos:** Nombra los métodos de prueba de forma descriptiva para reflejar la funcionalidad que están validando.
- **Organización:** Organiza las clases de prueba siguiendo la estructura de paquetes del proyecto.

## Documentación

Documenta cada cambio de funcionalidad y asegúrate de actualizar los archivos README o cualquier otro archivo relevante si la funcionalidad cambia. Toda nueva clase pública debe tener su JavaDoc.

Para generar el JavaDoc completo del proyecto (en NetBeans con Ant):

1. Navega a la carpeta del proyecto.

2. Ejecuta:

```
ant javadoc
```

O

Presiona *click derecho* en la carpeta del proyecto y ejecutar JavaDoc.

Los archivos generados estarán en la carpeta *dist/javadoc*.

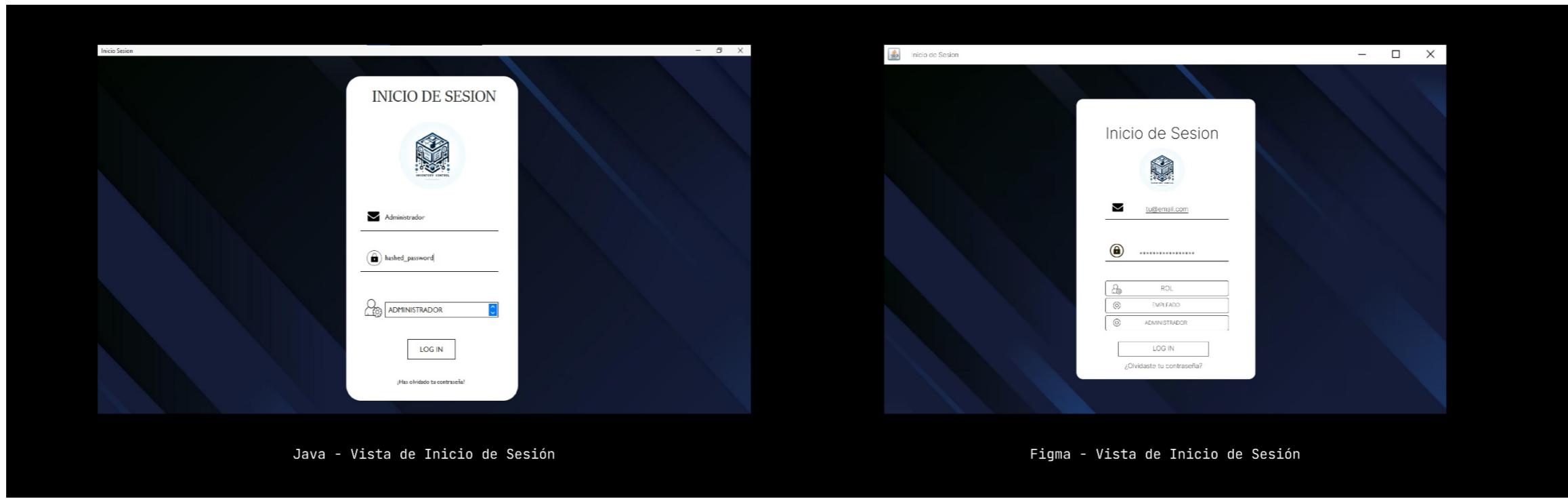
## **9. Comparación de implementación con el diseño**

Link Interactividad en Figma:

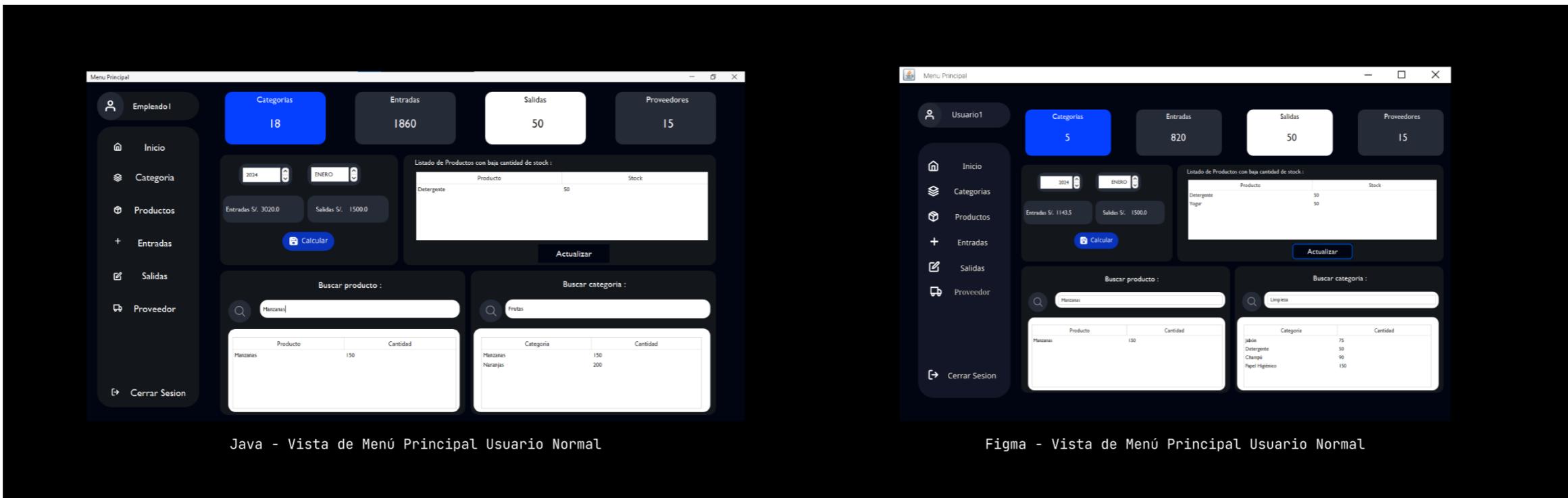
<https://www.figma.com/proto/xrxaX3t2tCiKt8jKjrFx4e/Integrador-Control-De-Stock-Prototipos?node-id=1-1292&node-type=canvas&t=9Iii9MjhVbHvFWin-0&scaling=scale-down&content-scaling=fixed&page-id=0%3A1&starting-point-node-id=15%3A1392>

## Figura

Comparación de la implementación de la vista de inicio de sesión del proyecto del sistema de control de stock en Java y Figma

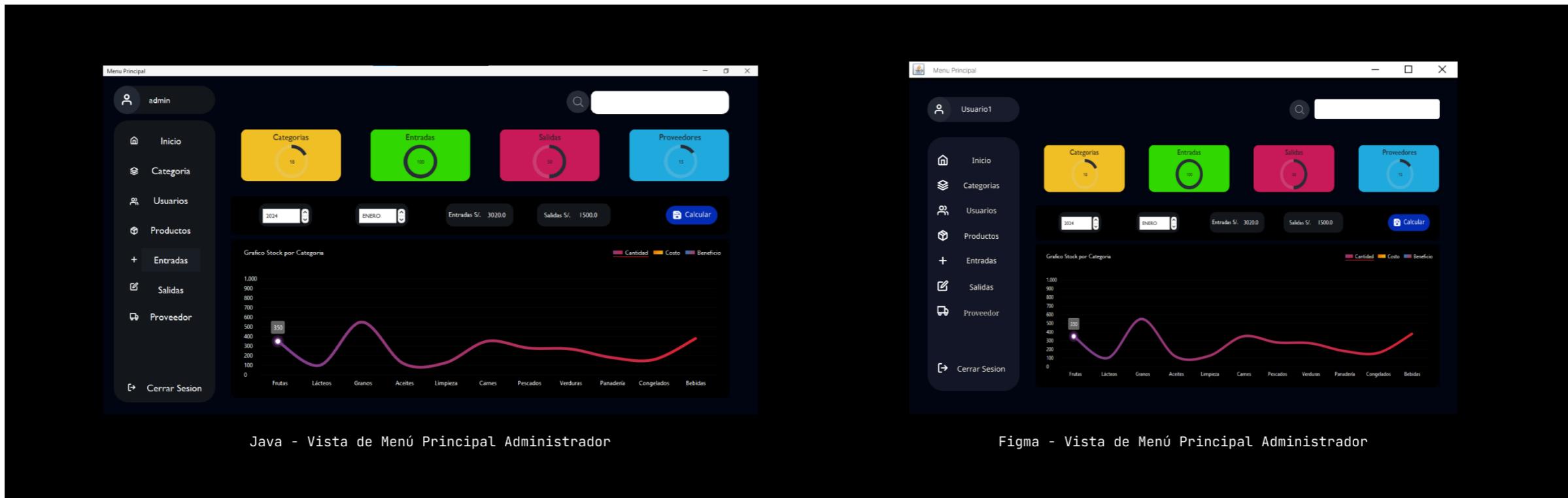


Comparación de la implementación de la vista de menú principal para el Usuario Normal o Empleado del proyecto del sistema de control de stock en Java y Figma

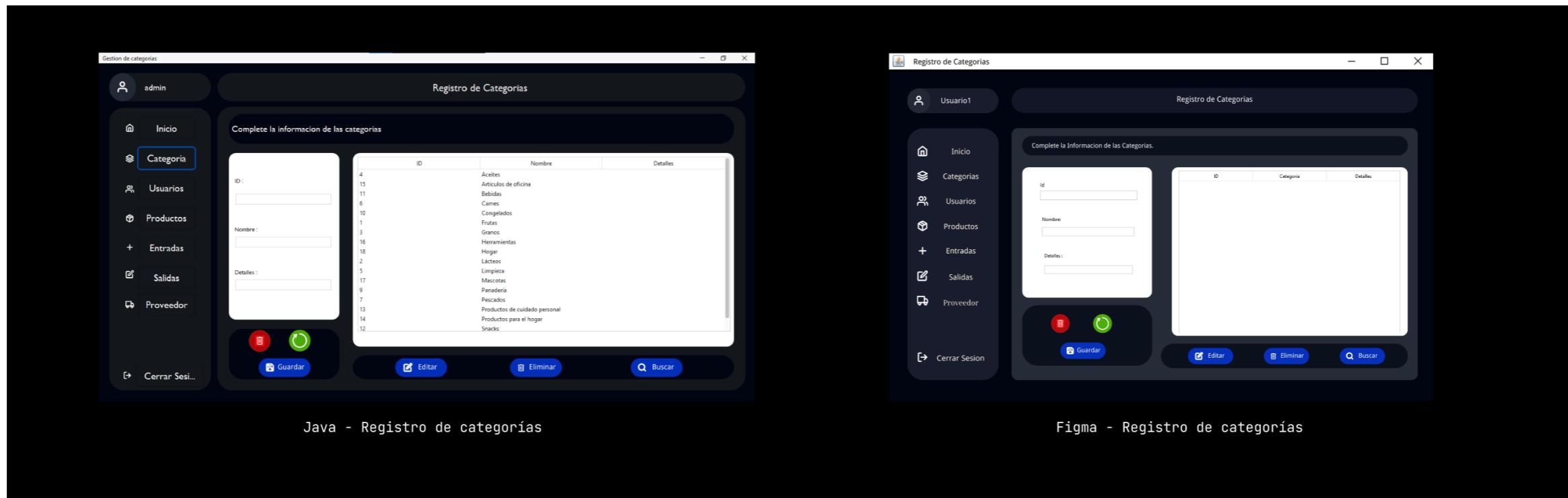


## Figura

Comparación de la implementación de la vista de menú principal para el Administrador del proyecto del sistema de control de stock en Java y Figma

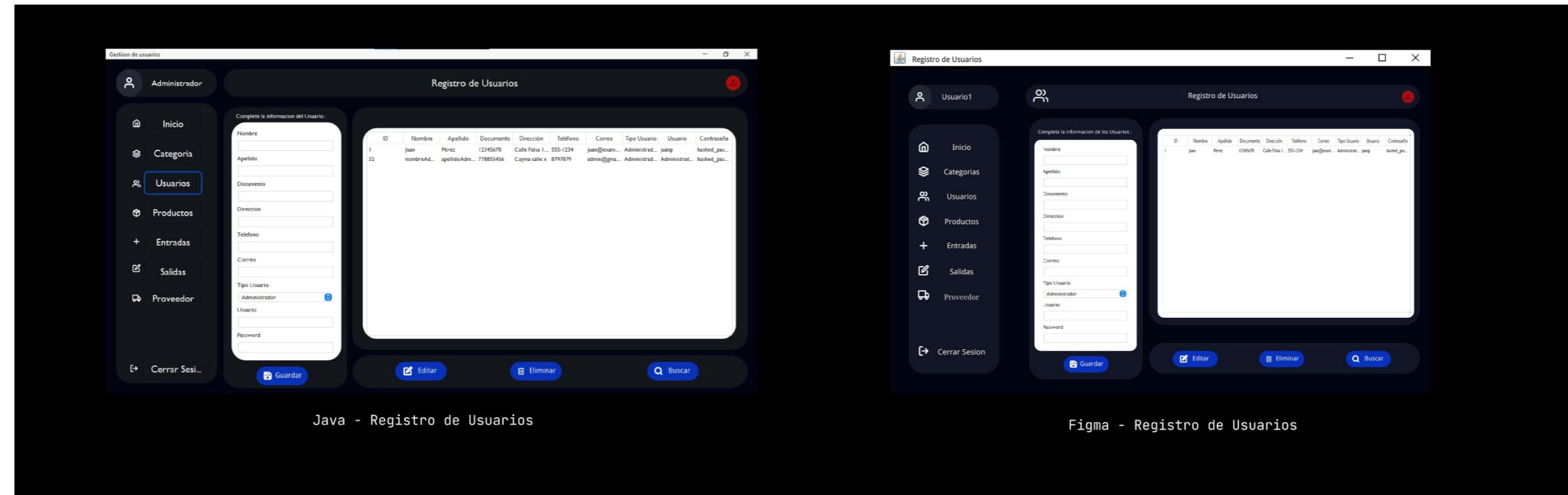


Comparación de la implementación de la vista de registro de categorías del proyecto del sistema de control de stock en Java y Figma

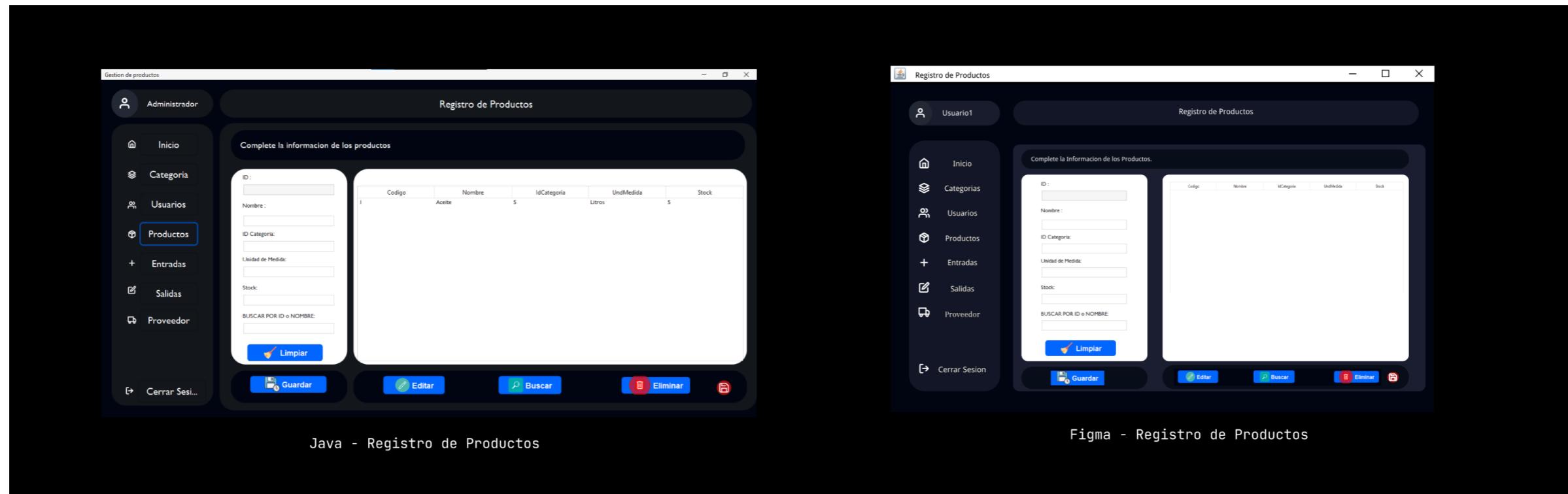


## Figura

Comparación de la implementación de la vista de registro de usuarios del proyecto del sistema de control de stock en Java y Figma

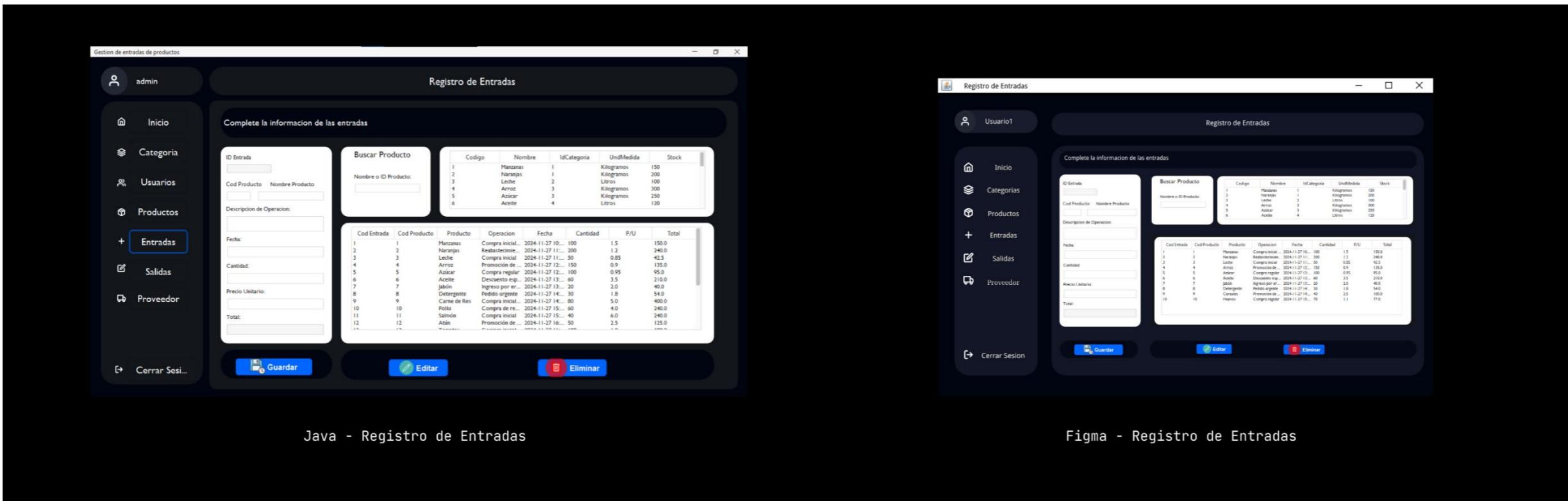


Comparación de la implementación de la vista de registro de productos del proyecto del sistema de control de stock en Java y Figma

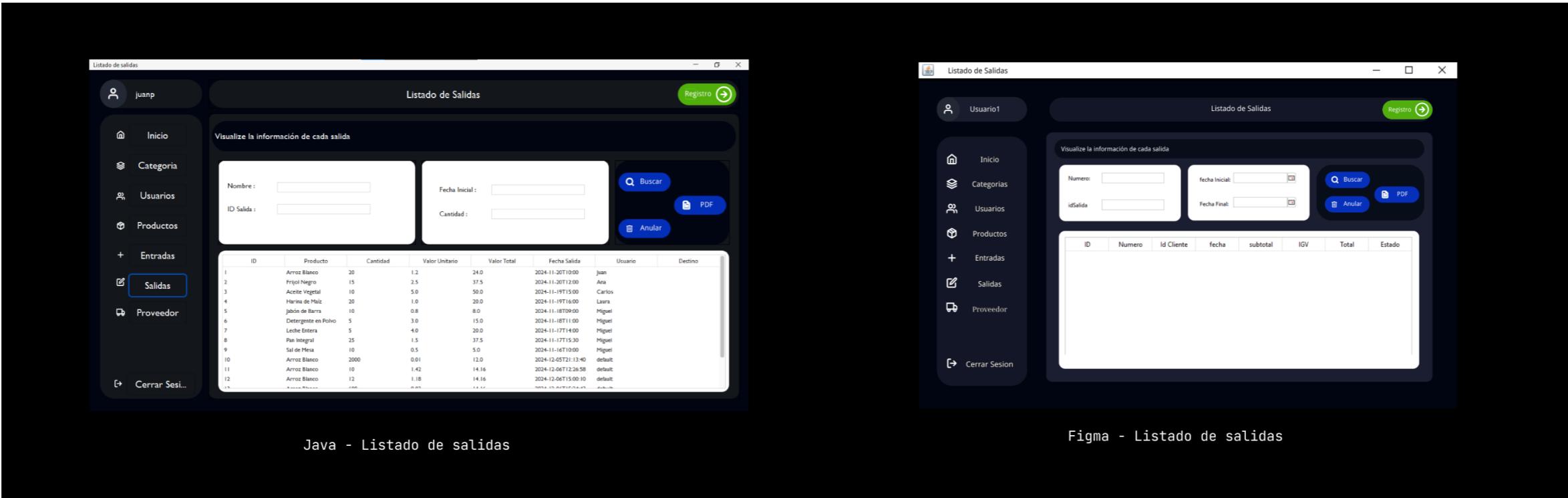


**Figura**

**Comparación de la implementación de la vista de registro de entradas del proyecto del sistema de control de stock en Java y Figma**

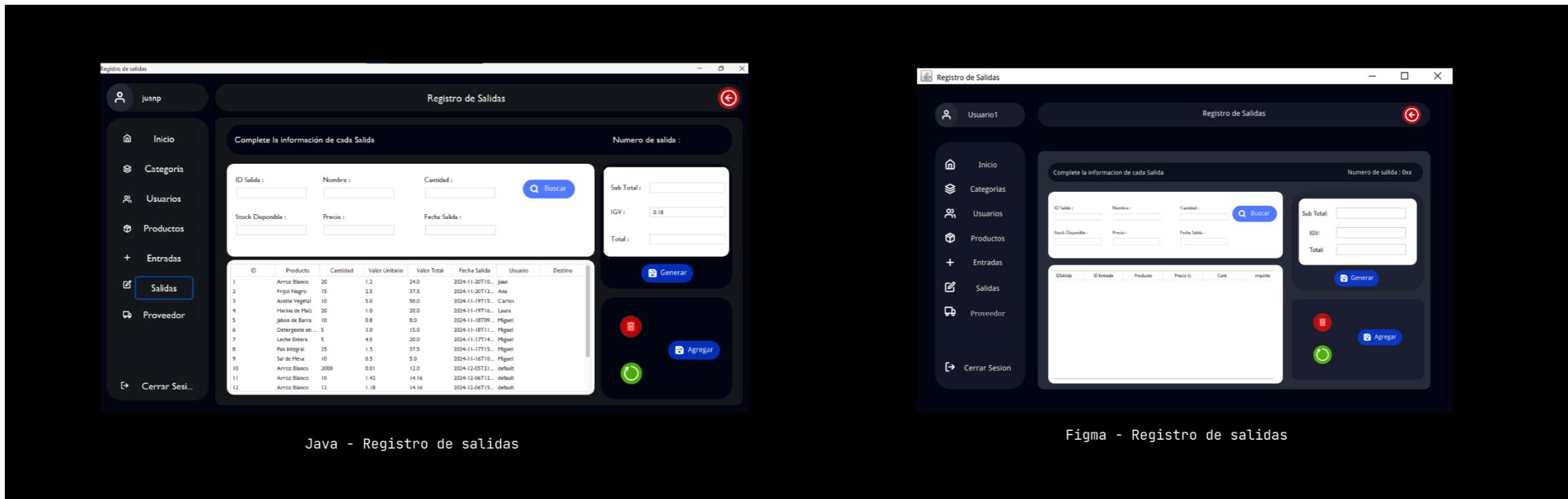


**Comparación de la implementación de la vista de listado de salidas del proyecto del sistema de control de stock en Java y Figma**



## Figura

Comparación de la implementación de la vista de registro de salidas del proyecto del sistema de control de stock en Java y Figma



Comparación de la implementación de la vista de registro de proveedores del proyecto del sistema de control de stock en Java y Figma

