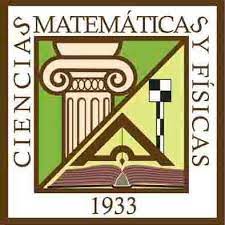
****

UNIVERSIDAD DE GUAYAQUIL

FACULTAD DE CIENCIAS MATEMÁTICAS Y FÍSICAS

PROYECTO SEGUNDO PARCIAL

Grupo:

Belén Alejandro Ambi María

Bermeo Varela Alisson Citlalli

Fajardo Cristopher Dustin Silva

Falcones Torres Kevin Josue

Panches Iñiguez Michael Adonis

Quimi Franco Washington Roberth

Docente:

Ing. Lilia Beatriz Santos Díaz

Materia:  
Verificación y Validación de Software

GUAYAQUIL

9 de agosto de 2023

[Testing estático 3](#_Toc142516865)

[Documentación 3](#_Toc142516866)

[SonarQube para Testing Estático **¡Error! Marcador no definido.**](#_Toc142516867)

[Testing dinámico – Caja Negra 6](#_Toc142516868)

[Partición de equivalencia y valores limite 6](#_Toc142516869)

[Diagramas de transición de estado 6](#_Toc142516870)

[Tablas de decisión 8](#_Toc142516871)

[Casos de uso 9](#_Toc142516872)

[Testing dinámico – Caja Blanca 10](#_Toc142516873)

[Criterios de caminos 10](#_Toc142516874)

[Comprobación de existencia de un nuevo cliente 10](#_Toc142516875)

[Bucles 11](#_Toc142516876)

[Testing Métodos Lógica de Negocios – JUnit 12](#_Toc142516877)

[Importancia de las Pruebas 12](#_Toc142516878)

[Descripción de las Pruebas 13](#_Toc142516879)

[Resultados de las Pruebas 22](#_Toc142516880)

[Conclusión 22](#_Toc142516881)

[Testing de Rendimiento 23](#_Toc142516882)

[Testing con JMeter 23](#_Toc142516883)

[Testing de Usabilidad 26](#_Toc142516884)

[Software para testing de Usabilidad 26](#_Toc142516885)

[Testing con Optimal Worshop 27](#_Toc142516886)

[Análisis del testing: 29](#_Toc142516887)

[Conclusiones: 30](#_Toc142516888)

[Reporte de Incidencias 31](#_Toc142516889)

[Plataforma Confluence 31](#_Toc142516890)

[Espacio creado 31](#_Toc142516891)

Repositorios

Proyecto a Evaluar:

[https://github.com/RubenCarton623/SALA5-CROWFUNDINGhttps://github.com/RubenCarton623/SALA5-CROWFUNDING](https://github.com/RubenCarton623/SALA5-CROWFUNDING)

Proyecto Evaludado:

<https://github.com/Micha3lP/Grupo-1-Evaluacion-Proyecto-del-Grupo-5>

Testing estático

Documentación

**Errores de Redacción**

1. Línea 8:`clientSeleccionado` debería ser `clienteSeleccionado`.

**Requerimientos Ambiguos o Incompletos**

1. Constructor (`TablaClientes`): El propósito y la funcionalidad exacta de la clase `TablaClientes` no están claramente definidos en el código. Se necesita una descripción más detallada de qué hace esta clase y cómo encaja en el sistema general.

**Modelos de Datos no Normalizados**

1. División de Datos en el Archivo: El código lee un archivo de datos donde los campos de un cliente están separados por comas. Esto podría ser propenso a errores si hay comas dentro de los valores de los campos. Sería más seguro usar un formato como JSON o CSV con un manejo de delimitadores más robusto.

**Casos de Uso Incompletos**

1. Casos de Uso de Interacción con los Clientes: No se describen los casos de uso relacionados con la interacción del usuario con los datos de los clientes, como agregar un nuevo cliente, editar un cliente existente, seleccionar un cliente de la tabla, entre otros.
2. Validación de Datos de Clientes: No se observa ninguna validación de datos de clientes antes de agregarlos o mostrarlos en la tabla. Sería importante incluir validaciones para garantizar que los datos sean consistentes y válidos.
3. Flujo de Trabajo del Formulario: No se proporciona un flujo de trabajo completo en la documentación que explique cómo interactuar con el formulario `TablaClientes`, qué acciones se pueden realizar y cómo se reflejarán en la interfaz.

**Conclusiones**

- El código actual podría beneficiarse de una mejor estructuración y separación de responsabilidades para mejorar la legibilidad y la mantenibilidad. La lógica de la carga de datos, la gestión de eventos y la manipulación de la interfaz de usuario podría modularizarse en diferentes métodos o clases.

- Sería útil documentar los atributos y métodos públicos con comentarios de estilo XML para que otros desarrolladores puedan comprender su propósito y cómo usarlos.

Testing dinámico – Caja Negra

Partición de equivalencia y valores limite

En esta hoja de cálculo, se han creado casos de prueba basados en las clases de equivalencia para la tabla de "Cliente". Se han identificado dos clases: "Casos Inválidos" (donde se prueban datos inválidos) y "Casos Válidos" (donde se prueban datos válidos). Los escenarios de prueba varían según estas clases de equivalencia.

|  |
| --- |
|  |
| Clase de Equivalencia | Escenario de Prueba | Datos de Entrada | Resultado Esperado | Resultado Real | Estado de la Prueba |
| Casos Inválidos | Crear cliente inválido | Nombre: "", Edad: -5 | Error: Datos inválidos |  |  |
| Casos Válidos | Crear cliente válido | Nombre: "John", Edad: 30 | Cliente creado exitosamente |  |  |
|  | Actualizar cliente con nombre válido | ID: 123, Nombre: "Jane" | Información del cliente actualizada exitosamente |  |  |
|  | Eliminar cliente válido | ID: 456 | Cliente eliminado exitosamente |  |  |

Diagramas de transición de estado

**Tabla de Cliente:**

***Diagrama de Transición de Estado***

| **Estado Actual** | **Evento** | **Estado Siguiente** | **Acción Esperada** | **Acción Real** | **Estado de la Prueba** |
| --- | --- | --- | --- | --- | --- |
| Sin cliente | Crear cliente | Cliente creado | Cliente creado exitosamente |  |  |
| Cliente creado | Actualizar | Cliente actualizado | Información actualizada |  |  |
| Cliente creado | Eliminar | Sin cliente | Cliente eliminado exitosamente |  |  |

**Tabla de Usuario:**

***Diagrama de Transición de Estado***

| **Estado Actual** | **Evento** | **Estado Siguiente** | **Acción Esperada** | **Acción Real** | **Estado de la Prueba** |
| --- | --- | --- | --- | --- | --- |
| Sin usuario | Crear usuario | Usuario creado | Usuario creado exitosamente |  |  |
| Usuario creado | Actualizar rol | Usuario actualizado | Rol de usuario actualizado |  |  |
| Usuario creado | Eliminar | Sin usuario | Usuario eliminado exitosamente |  |  |

En estos diagramas de transición de estado, puedes definir los estados posibles para las entidades (como "Sin cliente", "Cliente creado", etc.), los eventos que provocan transiciones de estado (como "Crear cliente", "Actualizar", etc.), y las acciones esperadas cuando ocurren esas transiciones.

Tablas de decisión

Presenta una tabla de decisiones para la tabla de "Cliente". Se han establecido condiciones lógicas basadas en las cuales se toman diferentes acciones. Las condiciones incluyen valores booleanos "Verdadero" y "Falso". Para cada combinación de condiciones, se espera una acción específica.

|  |
| --- |
|  |
| Caso | **Condición 1** | **Condición 2** | **Acción Esperada** | **Acción Real** | **Estado de la Prueba** |
| 1 | Verdadero | Verdadero | Crear cliente exitosamente |  |  |
| 2 | Verdadero | Falso | Mostrar error |  |  |
| 3 | Falso | Verdadero | Mostrar error |  |  |
| 4 | Falso | Falso | Mostrar error |  |  |

Casos de uso

se crean casos de uso para probar la tabla de "Usuario". Cada caso de uso describe un escenario específico, incluyendo los datos de entrada y el resultado esperado. Cada caso de uso abarca diferentes aspectos de la funcionalidad de usuario.

|  |
| --- |
|  |
| Caso de Uso | **Escenario de Prueba** | **Datos de Entrada** | **Resultado Esperado** | **Resultado Real** | **Estado de la Prueba** |
| Caso 1 | Crear un nuevo usuario con rol | Nombre: "user1", Rol: "Admin" | Usuario creado exitosamente |  |  |
| Caso 2 | Crear un nuevo usuario sin rol | Nombre: "user2" | Mostrar error |  |  |
| Caso 3 | Actualizar rol de usuario | ID: 123, Rol: "Editor" | Rol de usuario actualizado exitosamente |  |  |
| Caso 4 | Eliminar usuario | ID: 456 | Usuario eliminado exitosamente |  |  |

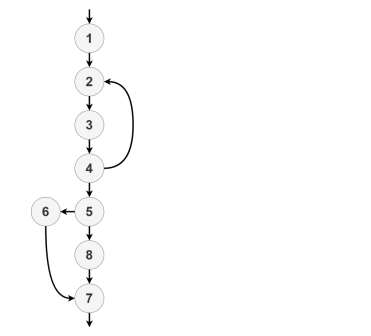
Testing dinámico – Caja Blanca

El testing de caja blanca se refiere a las pruebas realizadas en las que se tiene en cuenta el conocimiento interno de la estructura y el diseño del código. Esto implica probar las funcionalidades de un programa evaluando las decisiones tomadas en el código fuente. A menudo, estas pruebas se centran en garantizar que todas las rutas y condiciones lógicas del programa se prueben adecuadamente.

Criterios de caminos

Los criterios de caminos son una técnica utilizada en el testing de caja blanca para garantizar que todas las posibles rutas y condiciones en el código sean examinadas durante las pruebas. Esto implica identificar todas las rutas posibles a través del código y diseñar casos de prueba para cubrir cada una de ellas. Los criterios comunes incluyen la cobertura de sentencias, la cobertura de decisiones, la cobertura de condiciones, etc.

Comprobación de existencia de un nuevo cliente



Un test *T* satisface este criterio si todos los caminos desde el nodo inicial hasta el nodo final del grafo son atravesados

Todos los caminos:

[1 ,2 ,3 ,4 ,5 ,6 ]

[1 ,2 ,3 ,4 ,7 ,6 ]

Bucles

Los bucles son estructuras de control en la programación que permiten ejecutar un bloque de código repetidamente mientras se cumple una condición. En términos de testing, es esencial probar los bucles de manera adecuada, considerando diferentes situaciones: el bucle nunca se ejecuta, se ejecuta una vez, múltiples veces y hasta el máximo permitido. Además, se deben tener en cuenta casos extremos y los posibles puntos de entrada y salida del bucle.

Los bucles son casos especiales. Se deberían cubrir los siguientes casos, cuando sea posible:

* Ejecutar 0 veces el bucle, o el mínimo número de veces.
* Ejecutar el caso máximo posible de veces, si es determinable.
* Ejecutar un número promedio de veces, si es un valor calculable

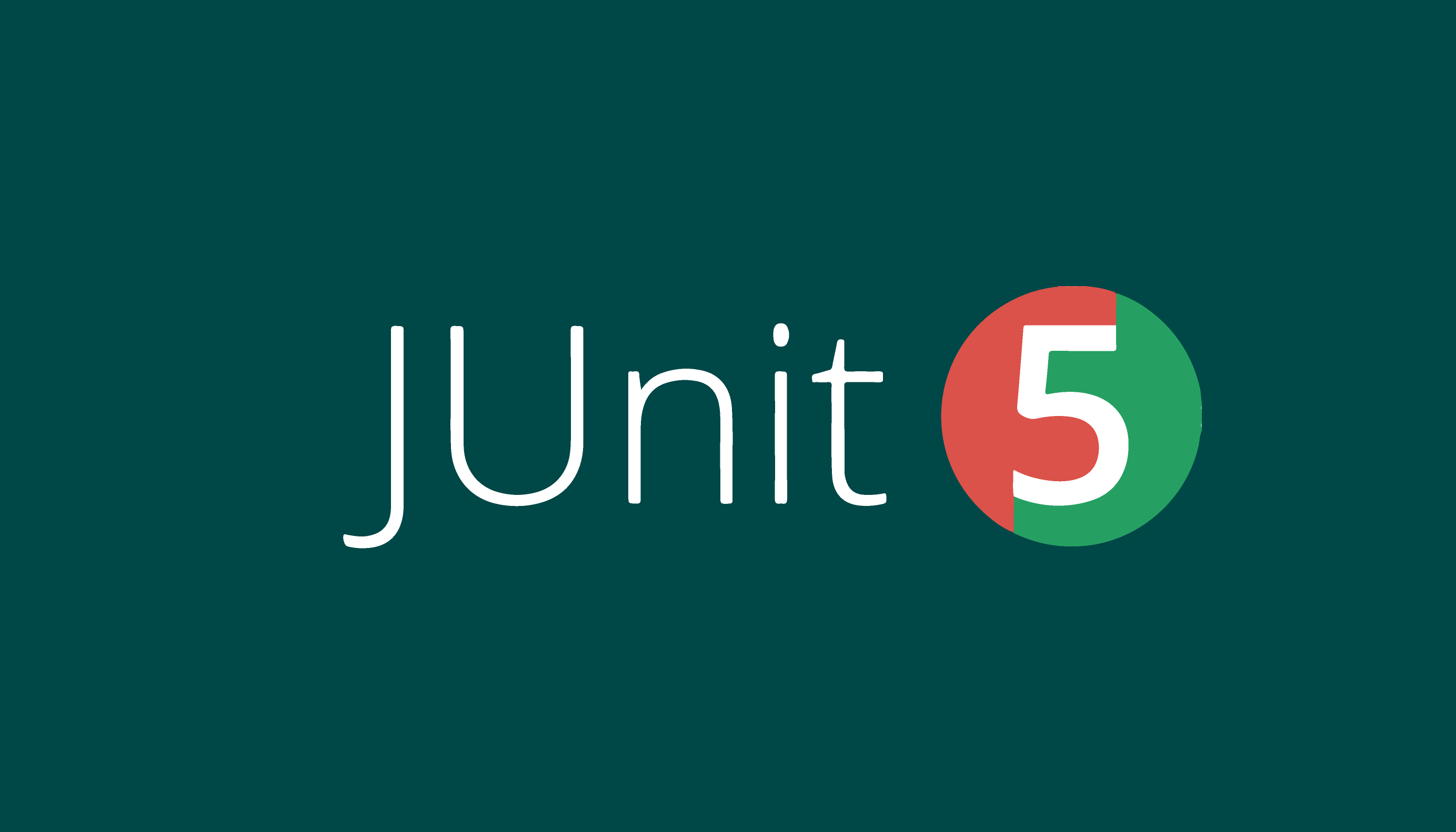
Los casos 2 y 3 no siempre son posibles. Por ejemplo, si el bucle es un for, sería posible, pero con un while va a ser más complicado

Testing Métodos Lógica de Negocios – JUnit

Las pruebas de lógica de negocio son una parte esencial en el desarrollo de software, ya que permiten verificar que las funcionalidades principales de una aplicación se comporten según lo esperado. En este documento, se presentarán y explicarán las pruebas realizadas utilizando JUnit para el proyecto de Crowdfunding descrito, destacando su importancia y resaltando los resultados exitosos de estas pruebas.

Importancia de las Pruebas

Las pruebas de lógica de negocio aseguran que el código subyacente de una aplicación funcione correctamente y cumpla con los requisitos y objetivos del sistema. Estas pruebas son especialmente relevantes en sistemas como el de Crowdfunding, donde la interacción entre los usuarios, la gestión de proyectos y la manipulación de datos son críticos para el funcionamiento exitoso del sistema.



JUnit es un popular marco de pruebas unitarias para Java, proporciona una estructura que permite diseñar y ejecutar pruebas automatizadas. Estas pruebas ayudan a identificar y resolver problemas en la lógica de negocio antes de que lleguen a la etapa de producción, lo que reduce costos y riesgos.

Descripción de las Pruebas

A continuación, se detallan las pruebas realizadas utilizando JUnit en el proyecto de Crowdfunding, junto con una descripción de cada método probado y su objetivo.

Prueba de Obtención de Icono Válido (testGetIconImageWithValidImage)

    @Test

    public void **testGetIconImageWithValidImage**() {

        System.out.**println**("testGetIconImageWithValidImage");

        menuprinc instance = new **menuprinc**();

        Image validImage = null;

        try {

            validImage = ImageIO.**read**(new **File**("D:\\Documentos\\NetBeansProjects\\PROYECTO CORREGIDO\\CrowdfundinGO\\src\\res\\ico.png"));

        } catch (IOException e) {

            e.**printStackTrace**();

        }

        Image result = instance.**getIconImage**();

**assertTrue**(validImage.**getWidth**(null) == result.**getWidth**(null) &&

                   validImage.**getHeight**(null) == result.**getHeight**(null));

    }

Esta prueba verifica si el método getIconImage de la clase menuprinc carga una imagen de icono válida. Se carga una imagen desde un archivo y se compara con la imagen obtenida del método.

Prueba de Obtención de Icono (testGetIconImage)

    @Test

    public void **testGetIconImage**() {

        System.out.**println**("getIconImage");

        menuprinc instance = new **menuprinc**();

        Image result = instance.**getIconImage**();

**assertNotNull**("Icon image should not be null", result);

**assertTrue**("Icon image should be an instance of Image", result instanceof Image);

    }

Esta prueba verifica si el método getIconImage de la clase menuprinc retorna una imagen de icono no nula y del tipo correcto. El objetivo es confirmar que se obtenga una imagen válida para ser utilizada como icono.

Prueba de Método main con Argumentos Nulos (testMainWithNullArgs)

    @Test

    public void **testMainWithNullArgs**() {

        System.out.**println**("testMainWithNullArgs");

        String[] args = null;

        menuprinc.**main**(args);

    }

Esta prueba verifica si el método main de la clase menuprinc se ejecuta correctamente cuando se le pasan argumentos nulos.

Prueba de Método main con Argumentos (testMainWithArgs)

    @Test

    public void **testMainWithArgs**() {

        System.out.**println**("testMainWithArgs");

        String[] args = new String[]{"arg1", "arg2"};

        menuprinc.**main**(args);

    }

Esta prueba verifica si el método main de la clase menuprinc se ejecuta correctamente cuando se le pasan argumentos.

Prueba de Validación de Campos (testValidar)

    @Test

    public void **testValidar**() {

        System.out.**println**("testValidar");

        Login instance = new **Login**();

        instance.user.**setText**("NombreUsuario");

        instance.contra.**setText**("Contraseña");

*// Verificar que NOMBRESG se haya establecido correctamente después de la validación.*

**assertEquals**( "", instance.**getNOMBRESG**());

    }

Esta prueba verifica si el método validar de la clase Login valida correctamente los campos de usuario y contraseña.

Prueba de Obtención de Valor de NOMBRESG (testGetNOMBRESG)

    @Test

    public void **testGetNOMBRESG**() {

        System.out.**println**("testGetNOMBRESG");

        Login instance = new **Login**();

*// Establece un valor esperado para el resultado de getNOMBRESG.*

        String expResult = ""; *// Cambia esto al valor esperado.*

*// Llama al método que deseas probar.*

        String result = instance.**getNOMBRESG**();

*// Compara el resultado con el valor esperado utilizando el método assertEquals.*

**assertEquals**(expResult, result);

    }

Esta prueba verifica si el método getNOMBRESG de la clase Login devuelve el valor esperado para la variable NOMBRESG. Se establece un valor esperado y se compara con el valor devuelto por el método.

Prueba de Llenado de Tabla de Proyectos (testLlenarTablaProy)

 @Test

    public void **testLlenarTablaProy**() throws Exception {

        System.out.**println**("llenarTablaProy");

        listusuarios instance = new **listusuarios**();

        instance.ProyTables.**setModel**(new javax.swing.table.**DefaultTableModel**(

            new Object [][] {

                {1, "Usuario1", "Apellido1", "Cuenta1", "Banco1", "Tipo1"},

                {2, "Usuario2", "Apellido2", "Cuenta2", "Banco2", "Tipo2"},

            },

            new String [] {

                "ID", "NOMBRES", "APELLIDOS", "CUENTA", "BANCO", "TIPO"

            }

        ));

**assertNotNull**(instance.ProyTables.**getModel**());

**assertEquals**(2, instance.ProyTables.**getModel**().**getRowCount**());

**assertEquals**(6, instance.ProyTables.**getModel**().**getColumnCount**());

    }

Esta prueba verifica si el método llenarTablaProy de la clase listusuarios llena la tabla de proyectos correctamente con datos simulados. Se crea una instancia de la clase, se establece un modelo simulado para la tabla y se verifican las dimensiones y valores de la tabla después del llenado.

Prueba de Validación de Campos Vacíos (testValidarCamposVacios)

    @Test

    public void **testValidarCamposVacios**() throws Exception {

        System.out.**println**("testValidarCamposVacios");

        modcuenta instance = new **modcuenta**();

        instance.username.**setText**("");

        instance.contra.**setText**("");

        try {

            instance.**validar**();

        } catch (Exception e) {

**fail**("Se esperaba una excepción para campos vacíos.");

        }

    }

Esta prueba verifica si el método validar de la clase modcuenta detecta correctamente los campos vacíos y lanza una excepción en consecuencia.

Prueba de Validación de Contraseña Vacía (testValidarContraseniaVacia)

    @Test

    public void **testValidarContraseniaVacia**() throws Exception {

        System.out.**println**("testValidarContraseniaVacia");

        modcuenta instance = new **modcuenta**();

        instance.username.**setText**("nombre\_de\_usuario");

        instance.contra.**setText**("");

        try {

            instance.**validar**();

        } catch (Exception e) {

**fail**("Se esperaba una excepción para contraseña vacía.");

        }

    }

Esta prueba verifica si el método validar de la clase modcuenta detecta correctamente la contraseña vacía y lanza una excepción.

Prueba de Validación de Campos Llenos (testValidarCamposLlenos)

  @Test

    public void **testValidarCamposLlenos**() throws Exception {

        System.out.**println**("testValidarCamposLlenos");

        modcuenta instance = new **modcuenta**();

*// Simular campos llenos*

        instance.username.**setText**("nombre\_de\_usuario");

        instance.contra.**setText**("contraseña\_segura");

        try {

*//conexion BD*

        } catch (Exception e) {

**fail**("No se esperaba ninguna excepción para campos llenos.");

        }

    }

Esta prueba verifica si el método validar de la clase modcuenta no lanza excepciones cuando todos los campos están llenos. Se establecen campos con valores y se espera que el método no lance ninguna excepción.

Prueba de Selección de Imagen (testElegirimg)

    @Test

    public void **testElegirimg**() throws FileNotFoundException {

        System.out.**println**("elegirimg");

        creaProy instance = new **creaProy**();

        instance.**elegirimg**();

**assertFalse**(creaProy.narchivo.**isEmpty**());

    }

Esta prueba verifica si el método elegirimg de la clase creaProy selecciona una imagen correctamente y asigna un valor a la variable narchivo. Se verifica si la variable no está vacía después de la selección de la imagen.

Prueba de Copiado de Imagen (testCopiarimg)

    @Test

    public void **testCopiarimg**() {

        System.out.**println**("copiarimg");

        String directorio = "D:/Imágenes/"; *// Cambia esta ruta a una válida en tu sistema*

        creaProy instance = new **creaProy**();

        creaProy.origen = "D:/Documentos/";

        creaProy.narchivo = "foto.jpg";

        try {

            instance.**copiarimg**(directorio);

        } catch (Exception ex) {

**fail**("La prueba de copiarimg falló: " + ex.**getMessage**());

        }

    }

Esta prueba verifica si el método copiarimg de la clase creaProy copia una imagen desde una ubicación de origen a una ubicación de destino correctamente. Se establece una ubicación de origen simulada y se verifica si la copia se realiza exitosamente.

Prueba de Llenado de Tabla de Proyectos

    @Test

    public void **testLlenarTablaProy**() throws Exception {

        System.out.**println**("llenarTablaProy");

*// Crear datos quemados para simular el ResultSet*

        Object[][] data = {

            {1, "Proyecto 1", 1000},

            {2, "Proyecto 2", 1500},

            {3, "Proyecto 3", 800}

        };

        String[] columnNames = {"ID", "TITULO", "DINERO"};

        DefaultTableModel mockModel = new **DefaultTableModel**(data, columnNames);

*// Crear una instancia de listdir y establecer el modelo de tabla simulado*

        listdir instance = new **listdir**();

        instance.ProyTables.**setModel**(mockModel);

    }

Esta prueba verifican si el método llenarTablaProy de la clase listdir llena la tabla de proyectos correctamente con datos simulados. Se crean instancias de la clase y se establecen modelos simulados para la tabla.

Prueba de Generación de Números Aleatorios (testRand)

    @Test

    public void **testRand**() {

        System.out.**println**("rand");

        int min = 0;

        int max = 10; *// Define a suitable range for your test*

        String result = MostraProy.**rand**(min, max);

**assertNotNull**(result);

    }

Esta prueba verifica si el método rand genera un número aleatorio dentro de un rango válido. Se establece un rango y se verifica si el resultado es no nulo.

Pruebas de Configuración de Atributos (testSetTitulovar, testSetLemavar, testSetDinerovar, testSetDescripcionvar, testSetLinkvar)

 @Test

    public void **testSetTitulovar**() {

        System.out.**println**("setTitulovar");

        String titulovar = "Mi Proyecto";

        MostraProy instance = new **MostraProy**();

        instance.**setTitulovar**(titulovar);

**assertEquals**(titulovar, instance.titulo.**getText**());

    }

    @Test

    public void **testSetLemavar**() {

        System.out.**println**("setLemavar");

        String lemavar = "Lema de mi proyecto";

        MostraProy instance = new **MostraProy**();

        instance.**setLemavar**(lemavar);

**assertEquals**(lemavar, instance.lema.**getText**());

    }

    @Test

    public void **testSetDinerovar**() {

        System.out.**println**("setDinerovar");

        String dinerovar = "5000";

        MostraProy instance = new **MostraProy**();

        instance.**setDinerovar**(dinerovar);

**assertEquals**("$" + dinerovar, instance.money.**getText**());

    }

    @Test

    public void **testSetDescripcionvar**() {

        System.out.**println**("setDescripcionvar");

        String descripcionvar = "Descripción de mi proyecto";

        MostraProy instance = new **MostraProy**();

        instance.**setDescripcionvar**(descripcionvar);

**assertEquals**("<html>" + descripcionvar + "<html>", instance.desc.**getText**());

    }

    @Test

    public void **testSetLinkvar**() {

        System.out.**println**("setLinkvar");

        String linkvar = "mi\_link.jpg";

        MostraProy instance = new **MostraProy**();

        instance.**setLinkvar**(linkvar);

**assertEquals**(linkvar, instance.**getLinkvar**());

    }

Estas pruebas verifican si los métodos de configuración de atributos en la clase MostraProy establecen correctamente los valores de los atributos. Se establecen valores y se verifica si los atributos se han configurado correctamente.

Prueba de Donación (testDonacion)

 @Test

    public void **testDonacion**() {

        System.out.**println**("donacion");

        int don = 100;

        int ID = 1;

        MostraProy instance = new **MostraProy**();

        try {

            int dineroInicial = 600;

            instance.**setDinerovar**(String.**valueOf**(dineroInicial));

            int dineroDespuesDeDonacion = Integer.**parseInt**(instance.money.**getText**().**substring**(1)) + don;

**assertEquals**(dineroInicial + don, dineroDespuesDeDonacion);

        } catch (Exception ex) {

            System.out.**println**("An exception occurred: " + ex.**getMessage**());

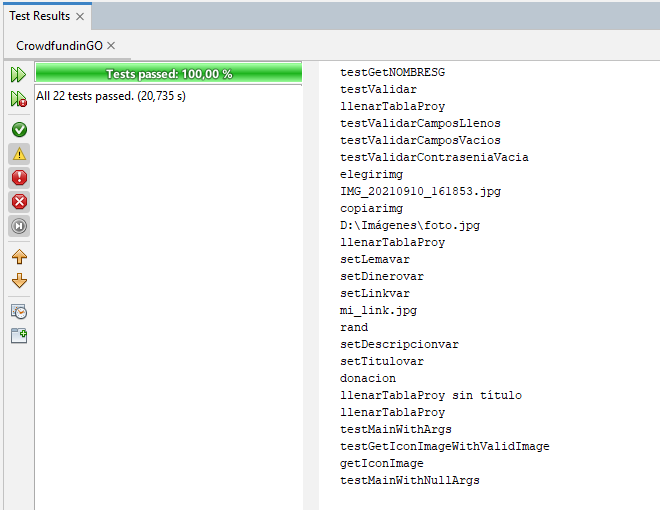
**fail**("An exception occurred: " + ex.**getMessage**());

        }

    }

Esta prueba verifica si el método donacion de la clase MostraProy actualiza correctamente la cantidad de dinero después de una donación. Se simula una donación y se verifica si el valor actualizado coincide con el valor esperado.

Resultados de las Pruebas



Todas las pruebas de lógica de negocio en el proyecto de Crowdfunding se realizaron exitosamente, lo que indica que la funcionalidad principal de la aplicación está implementada correctamente y que la lógica de negocio se comporta según lo esperado. Cada prueba confirmó que los métodos probados cumplen con su funcionalidad.

Conclusión

Las pruebas de lógica de negocio desempeñan un papel crucial en el desarrollo de software, ya que garantizan que las funcionalidades centrales de una aplicación funcionen correctamente. Utilizando JUnit, se pueden diseñar y ejecutar pruebas automatizadas para verificar la lógica subyacente de la aplicación, identificar problemas potenciales y asegurar la calidad del software. En el caso del proyecto de Crowdfunding, las pruebas de lógica de negocio demostraron que el sistema está en buen estado y que la lógica detrás de sus funciones es sólida y confiable.

Testing de Rendimiento

Testing con JMeter



Apache JMeter se puede usar para probar el rendimiento tanto en recursos estáticos como dinámicos, aplicaciones dinámicas web.  
Se puede usar para simular una carga pesada en un servidor, grupo de servidores, red u objeto para probar su fuerza o para analizar el rendimiento general bajo diferentes tipos de carga.

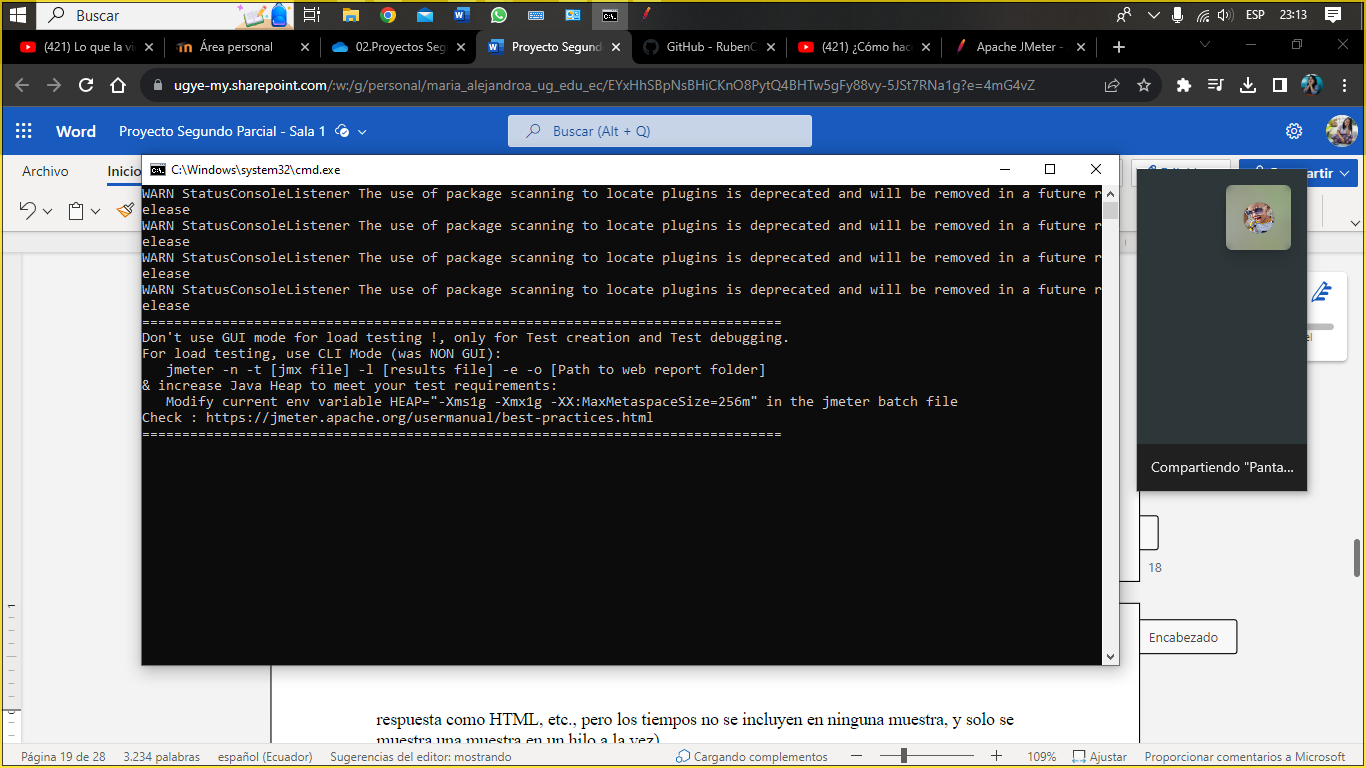
Las características de Apache JMeter incluyen:

Capacidad para cargar y probar el rendimiento de muchas aplicaciones/servidores/tipos de protocolos diferentes:

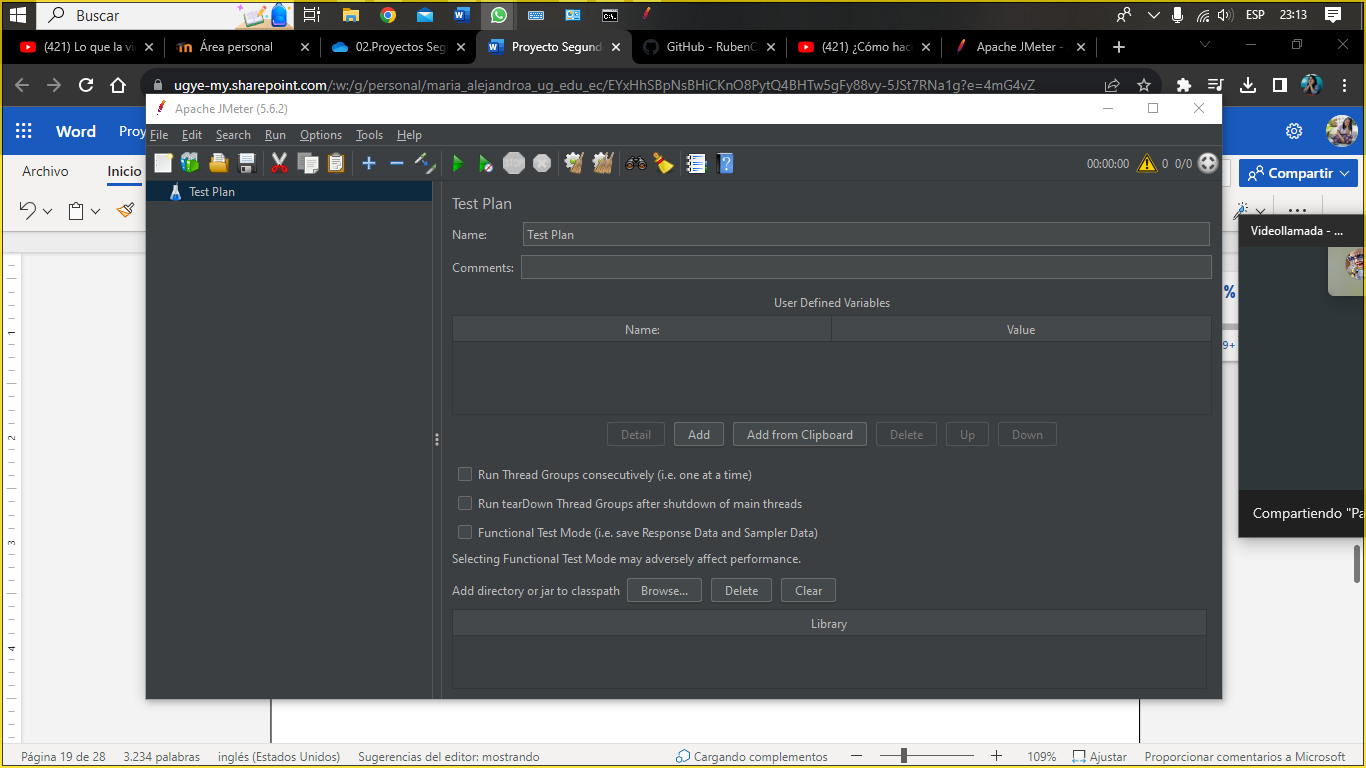
* Web: HTTP, HTTPS (Java, NodeJS, PHP, ASP.NET, …)
* Servicios web SOAP/REST
* FTP
* Base de datos a través de JDBC
* LDAP
* Middleware orientado a mensajes (MOM) a través de JMS
* Correo - SMTP(S), POP3(S) e IMAP(S)
* Comandos nativos o scripts de shell
* TCP
* Objetos Java
* IDE de prueba con todas las funciones que permite la grabación rápida del plan de prueba (desde navegadores o aplicaciones nativas), compilación y depuración .
* [Modo CLI (modo de línea de comandos (anteriormente llamado Non GUI) / modo sin cabeza)](https://jmeter.apache.org/usermanual/get-started.html#non_gui) para cargar la prueba desde cualquier sistema operativo compatible con Java (Linux, Windows, Mac OSX, …)
* Un [informe HTML dinámico completo y listo para presentar](https://jmeter.apache.org/usermanual/generating-dashboard.html)
* Fácil correlación a través de la capacidad de extraer datos de los formatos de respuesta más populares, [HTML](https://jmeter.apache.org/usermanual/component_reference.html#CSS/JQuery_Extractor) , [JSON](https://jmeter.apache.org/usermanual/component_reference.html#JSON_Extractor) , [XML](https://jmeter.apache.org/usermanual/component_reference.html#XPath_Extractor) o [cualquier formato de texto](https://jmeter.apache.org/usermanual/component_reference.html#Regular_Expression_Extractor)
* Portabilidad completa y pureza 100% Java .
* El marco completo de subprocesos múltiples permite el muestreo concurrente por muchos subprocesos y el muestreo simultáneo de diferentes funciones por grupos de subprocesos separados.
* Almacenamiento en caché y análisis/reproducción fuera de línea de los resultados de las pruebas.

JMeter no es un navegador, funciona a nivel de protocolo. En lo que respecta a los servicios web y los servicios remotos, JMeter parece un navegador (o más bien, múltiples navegadores); sin embargo, JMeter no realiza todas las acciones compatibles con los navegadores. En particular, JMeter no ejecuta el Javascript que se encuentra en las páginas HTML. Tampoco representa las páginas HTML como lo hace un navegador (es posible ver la respuesta como HTML, etc., pero los tiempos no se incluyen en ninguna muestra, y solo se muestra una muestra en un hilo a la vez).

Bueno cuando se empieza por abrir el JMeter nos aparece lo siguiente, pero toca esperar unos minutos...



Una vez cargada se muestra lo siguiente luego de unos minutos...

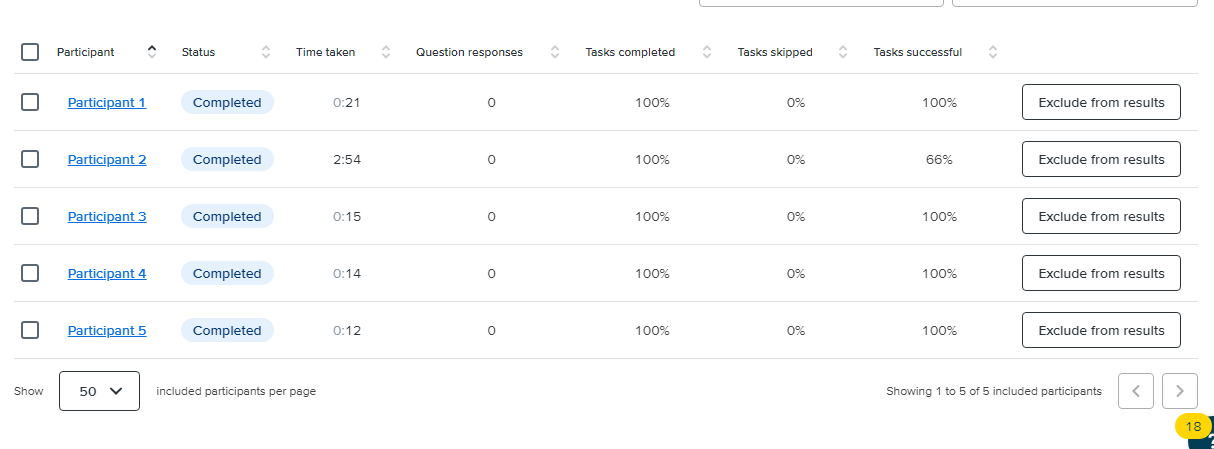


Luego ya se abre el documento que desiemos testear ya tenemos la ejecucion y los resultados.

Testing de Usabilidad

Software para testing de Usabilidad

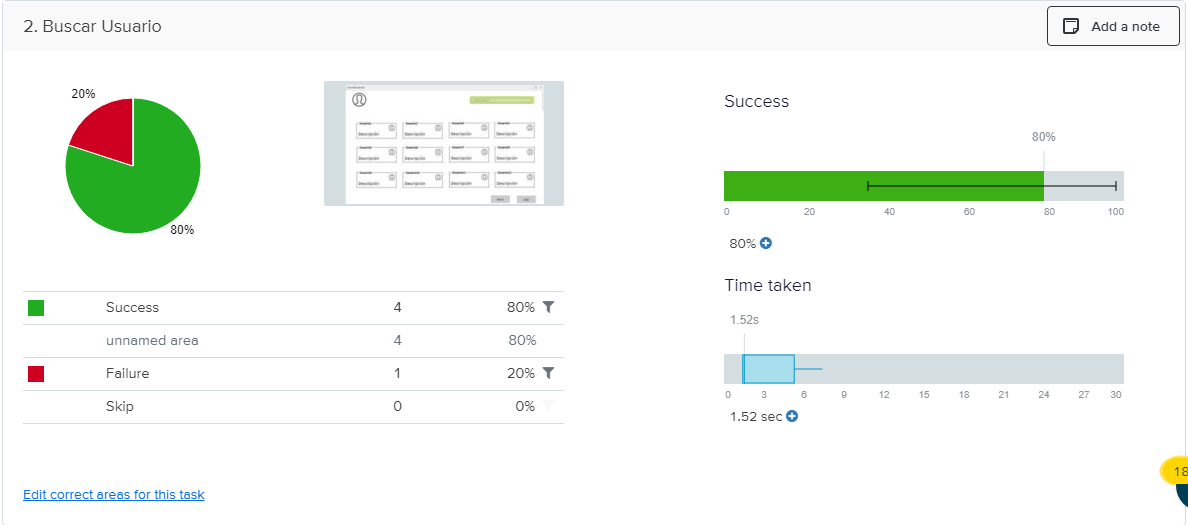
Para el testing de usabilidad se optó por usar la herramienta Optimal Worshop. Optimal Workshop es una plataforma de investigación y diseño centrado en la experiencia del usuario que proporciona herramientas y soluciones para ayudar a los equipos de diseño y desarrollo a mejorar la usabilidad y la efectividad de sus productos digitales.

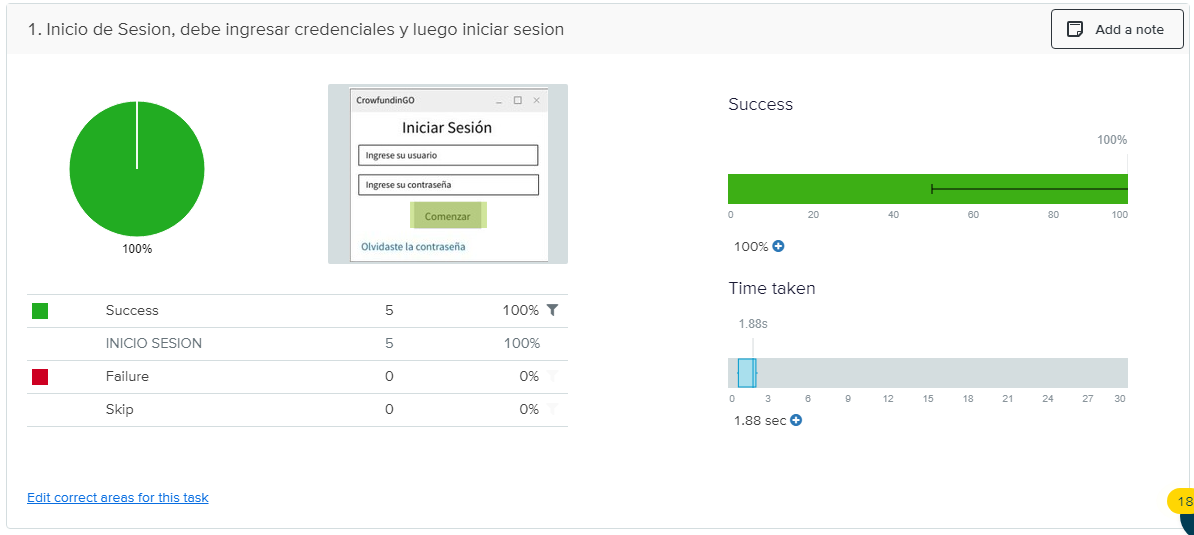


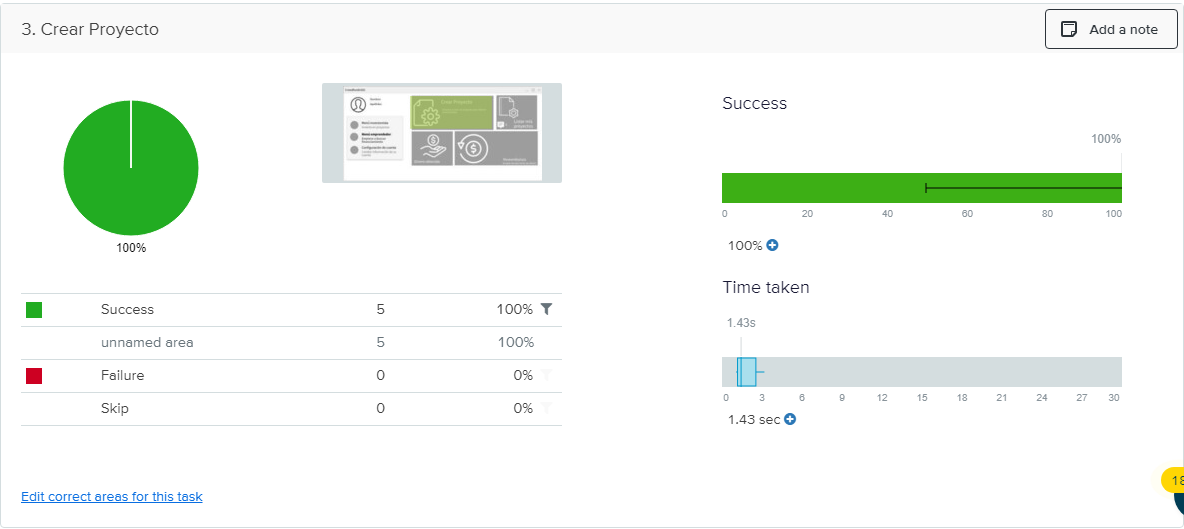
Esta plataforma ofrece una variedad de herramientas y métodos que permiten a los profesionales de UX (experiencia de usuario) y a los diseñadores de interacción llevar a cabo investigaciones, pruebas y análisis de la usabilidad de sus productos.

Testing con Optimal Worshop

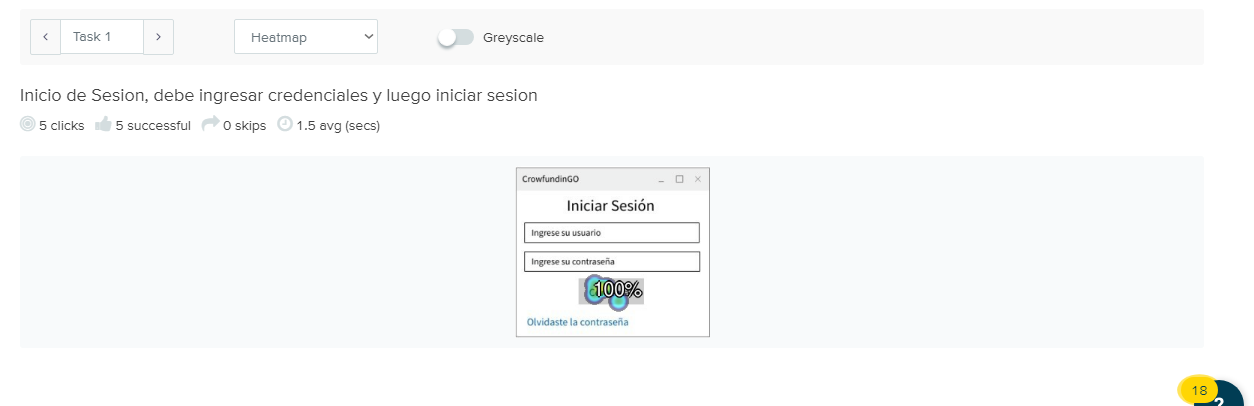
Para el testing se usaron a 5 participantes que tenían que cumplir con 3 tareas, el inicio sesión, la búsqueda de usuarios y la creación de proyectos:



Luego de que se completaron las tareas, se obtuvieron los siguientes resultados:  


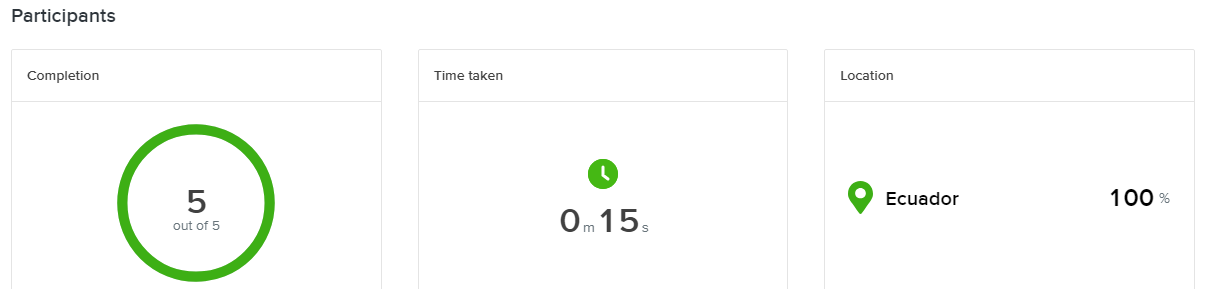


Mapas de calor donde se indican las zonas con las que el usuario tuvo interacción:







Vista General:  
  


Análisis del testing:

En el contexto de la fase de pruebas y evaluación, es notable destacar que los resultados obtenidos de las pruebas realizadas son sumamente alentadores. De manera específica, al analizar los datos emergentes del proceso de testing en el que participaron los cinco involucrados, se puede discernir que todos los participantes lograron completar las tareas encomendadas en un intervalo de tiempo promedio de 15 segundos, presentándose incluso un impresionante mínimo de 12 segundos en un caso particular.

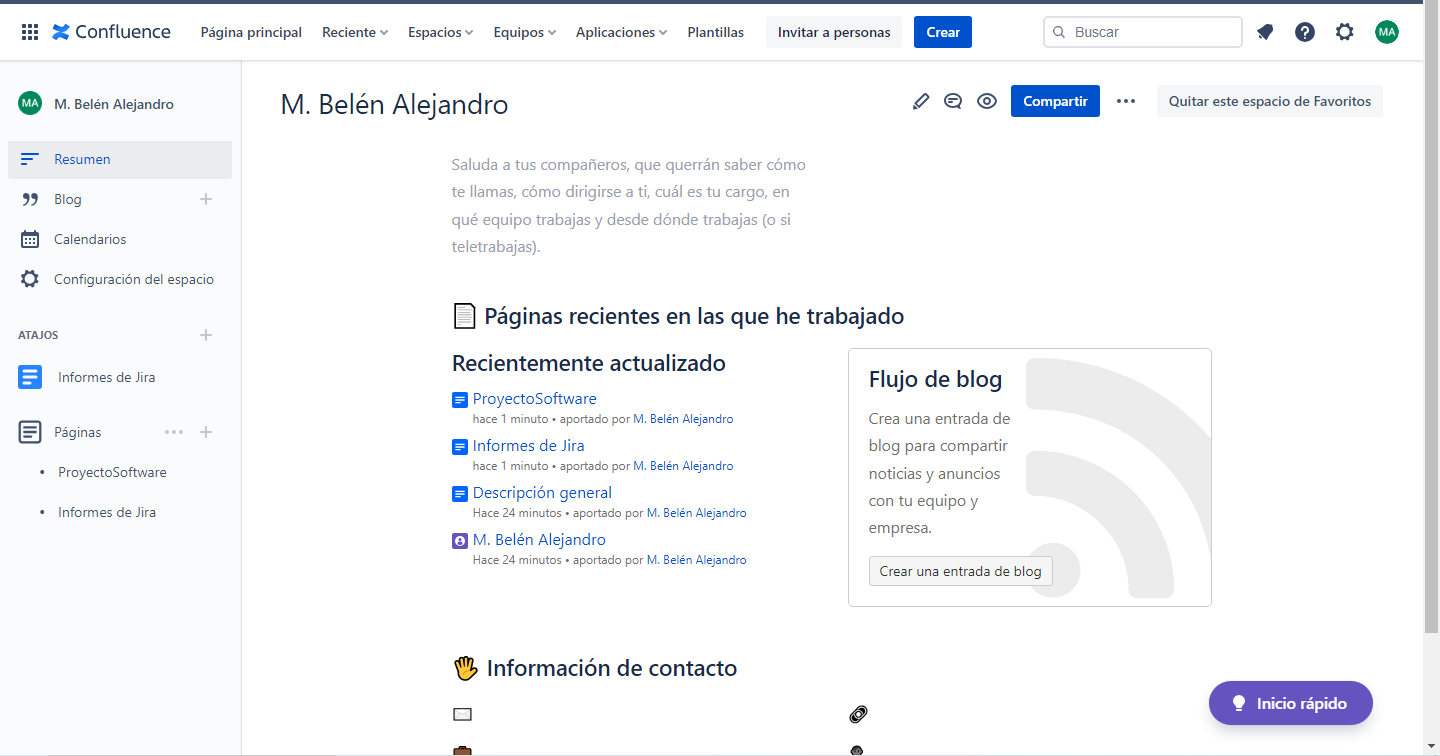
Conclusiones:

Estos resultados aportan un valioso indicio acerca de la eficiencia y la usabilidad de la interfaz que ha sido desarrollada. Emerge claramente que el sistema en cuestión exhibe un nivel de dinamismo e intuición que facilita la comprensión y la ejecución de las tareas planteadas. Este logro es de particular relevancia, ya que resalta la capacidad de la interfaz para guiar a los usuarios de manera efectiva y eficiente hacia la consecución de sus objetivos dentro del sistema.

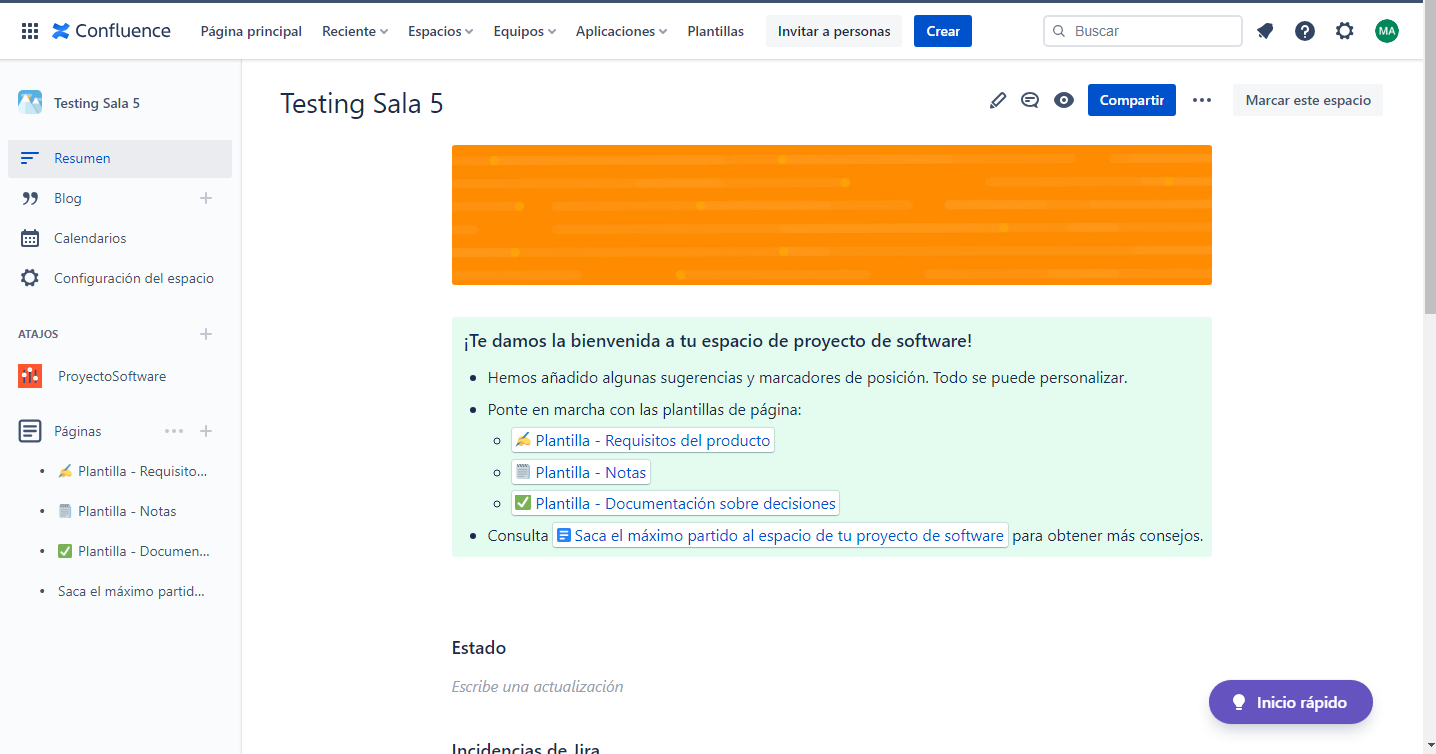
Este hallazgo no solo refleja la solidez del diseño y la implementación de la interfaz, sino que también valida el enfoque orientado al usuario que ha sido incorporado en todas las etapas del proceso de desarrollo. La combinación de elementos visuales atractivos con una estructura de navegación fluida y lógica ha resultado en un entorno que no solo se amolda a las expectativas de los usuarios, sino que también supera sus requisitos en términos de accesibilidad y desempeño.

Reporte de Incidencias

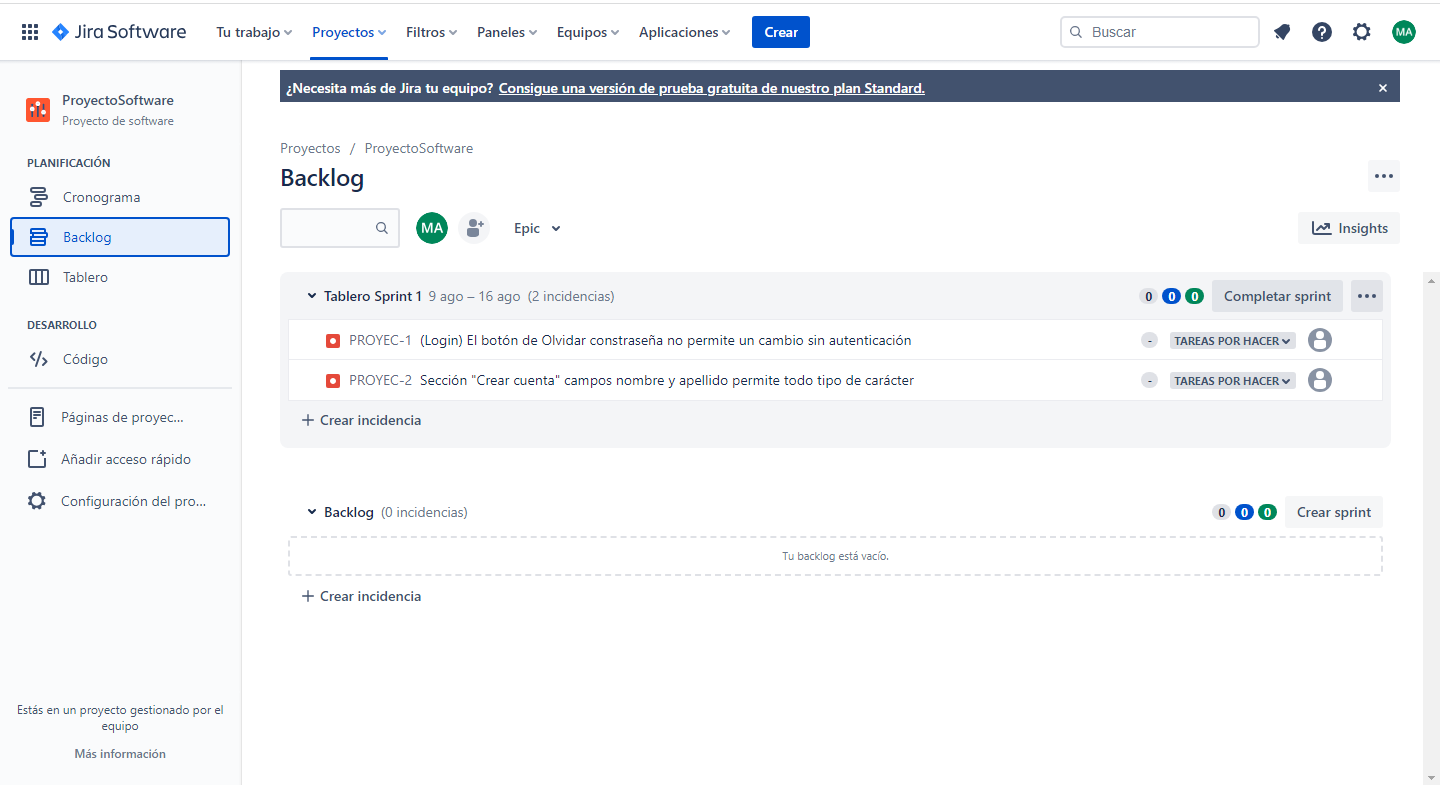
Plataforma Confluence



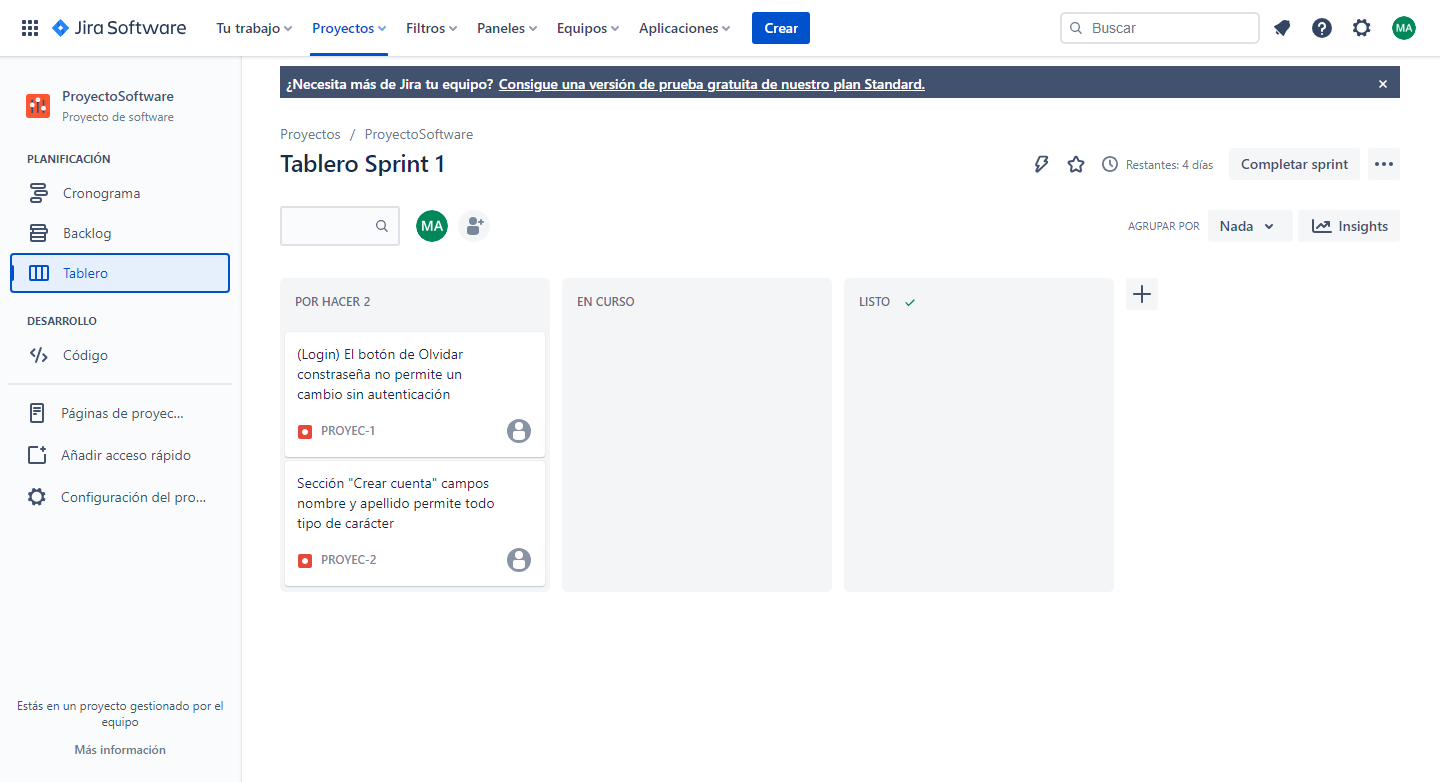
Espacio creado



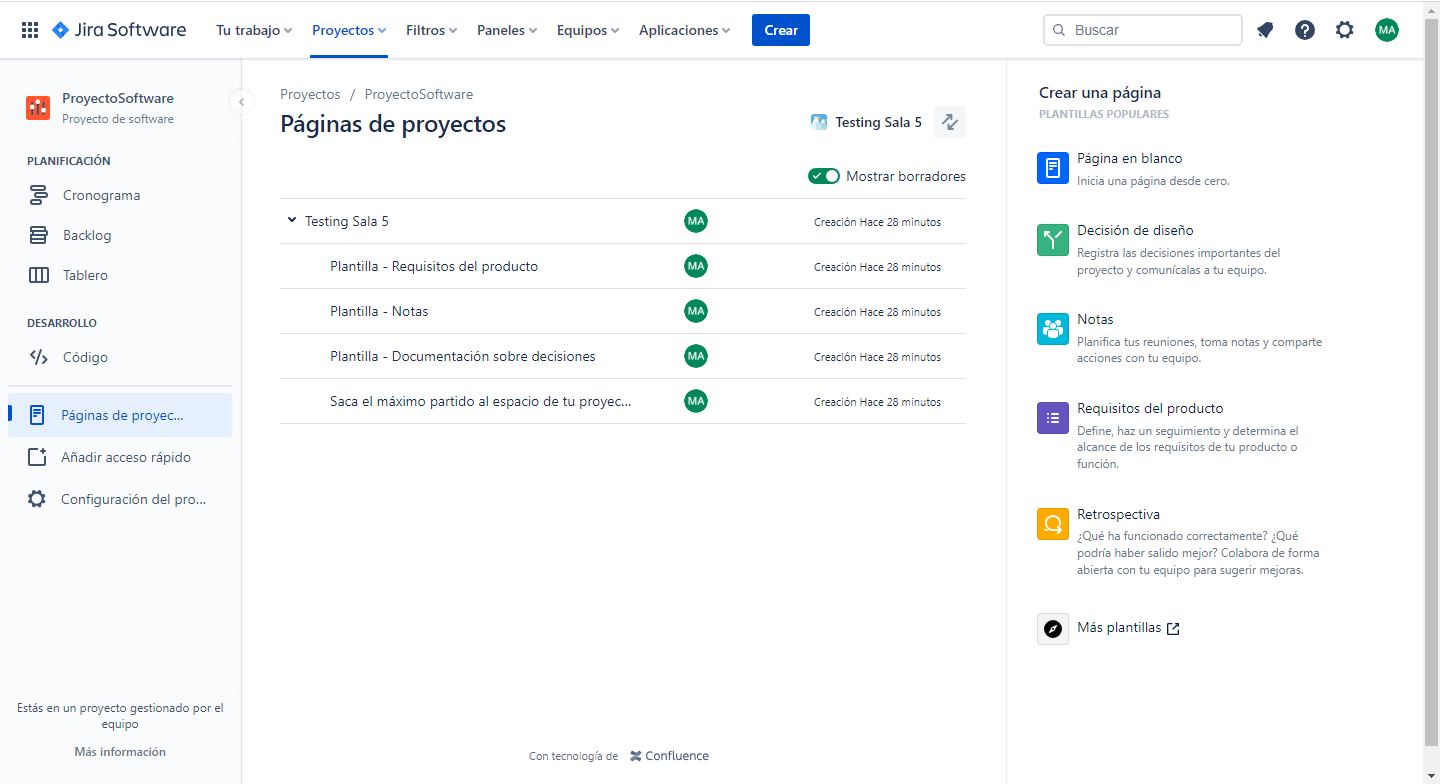
**Plataforma Jira**



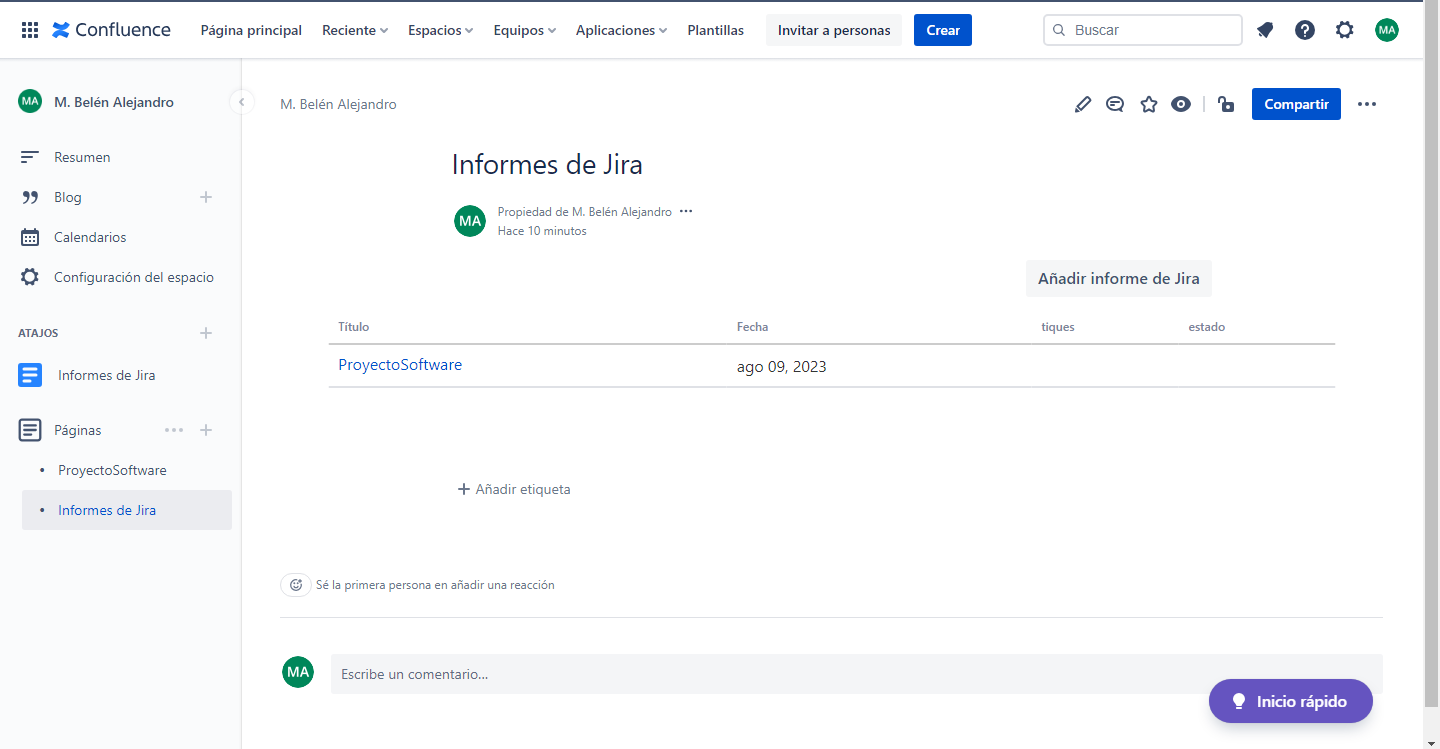
**Tablero Spring en Jira**



**Conexión Confluence – Jira**



**Informe incidencias Jira**



**Contenido informe incidencias Jira desde Confluence**

