



**Universidad de Guayaquil**

Facultad de Ciencias Matemáticas y Físicas

**Materia:**

Verificación y Validación.

**PROYECTO**

**Integrantes:**

* Aaron Álvarez Llamuca
* Fausto Torres Aspiazu
* Ivonne Roxana Minchala Plúas
* María Belén Vargas
* Milena Cruz Diaz
* Jordy Figueroa Martínez

**Ciclo:**

2023-2024 CI

**Contenido**

[**1.** **Testing Estático: Documentación.** 1](#_Toc143104760)

[ **Errores de redacción.** 1](#_Toc143104761)

[ **Requerimientos ambiguos o incompletos.** 1](#_Toc143104762)

[ **Casos de uso ambiguos o incompletos.** 2](#_Toc143104763)

[**2.** **Testing Estático: Código.** 2](#_Toc143104764)

[**3.** **Testing Dinámico de Caja Negra.** 7](#_Toc143104765)

[ **Clases de Equivalencia.** 7](#_Toc143104766)

[ **Valores Limites.** 8](#_Toc143104767)

[ **Diagrama de Transición de Estado.** 9](#_Toc143104768)

[ **Tabla de Decisión.** 11](#_Toc143104769)

[ **Casos de Uso.** 11](#_Toc143104770)

[**4.** **Testing Dinámico de Caja Blanca.** 17](#_Toc143104771)

[ **Criterios de caminos** 17](#_Toc143104772)

[ **Bucles** 18](#_Toc143104773)

[**5.** **Testing de Rendimiento del Software.** 18](#_Toc143104774)

[**6.** **Testing de Usabilidad del Software.** 21](#_Toc143104775)

[ **Software para testing de Usabilidad.** 21](#_Toc143104776)

[ **Testing con Optimal Worshop** 22](#_Toc143104777)

[ **Análisis del testing.** 25](#_Toc143104778)

[ **Conclusiones.** 25](#_Toc143104779)

[**7.** **Reporte de Incidencias.** 25](#_Toc143104780)

[**8.** **Repositorio.** 27](#_Toc143104781)

# **Testing Estático: Documentación.**

## **Errores de redacción.**

Documento **Diseño detallado de software.pdf**

**Visión del producto**

1. Se debe reemplazar “Para" a "los" para mayor claridad y concordancia con "viajeros".
2. Sustituir "NUESTRO PRODUCTO ofrece" a "**nuestro producto destaca al ofrecer**" para dar más énfasis y claridad al contraste con otros métodos.

Documento **Manual tecnico.pdf**

**Objetivos y alcances del sistema**

1. Agregar una coma después de "viaje" para mejorar la estructura y la fluidez del párrafo.
2. Sustituir "como lo es" por "tales como" para mejorar la fluidez y claridad de la enumeración.
3. Se debe reemplazar "para facilitar la planificación de los vuelos" a "Esto facilitará la planificación de los vuelos" para simplificar y darle más énfasis a la idea final.

## **Requerimientos ambiguos o incompletos.**

**Requerimientos Funcionales**

1. Componente De Actualización De Registros: No se especifica qué registros se deben actualizar ni qué proceso debe seguirse.
2. Componente Eliminación De Registros De Los Clientes: falta especificar en cuanto a qué registros se eliminarán y cómo se gestionará este proceso.

## **Casos de uso ambiguos o incompletos.**

**Actualizar reserva**: para mayor precisión, sería útil describir los pasos específicos involucrados en la actualización de una reserva, así como los datos que se pueden modificar.

**Actualizar cliente:** Se podría especificar qué campos específicos se pueden actualizar y cómo se llevará a cabo el proceso de actualización.

# **Testing Estático: Código.**

Con el propósito de garantizar la funcionalidad y confiabilidad de los métodos de lógica del negocio implementados en el sistema, se llevaron a cabo pruebas unitarias mediante la utilización de la herramienta JUnit5. Estas pruebas están diseñadas para verificar el comportamiento correcto de los distintos componentes funcionales del software, asegurando que cumplan con los requerimientos y objetivos establecidos.

El proceso de evaluación de los métodos de lógica del negocio se realizó en el entorno de desarrollo NetBeans, aprovechando la integración fluida de esta herramienta con JUnit5. Se procedió a ejecutar una serie de pruebas unitarias exhaustivas en cada uno de los métodos identificados, con el objetivo de validar su comportamiento en diferentes escenarios y condiciones.

CtrHotelTest: Conexión exitosa.

RenderTest: Prueba la creacion correcta de los objetos de la vista, Existoso.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

ConexionTest: Valida la conexión a la base desde el modelo, exitosa.

ConsultasClientesTest: Prueba que se puedan realizar operaciones en la tabla clientes.

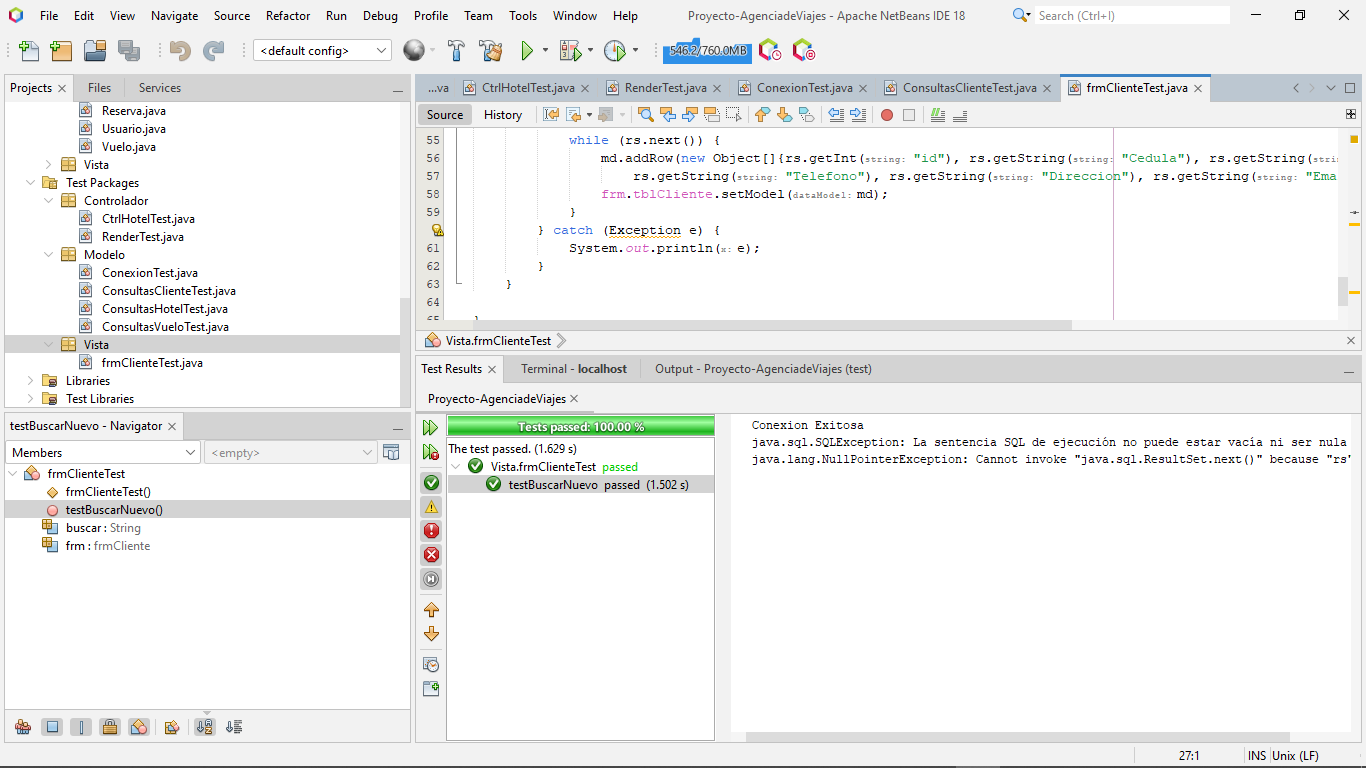
ConsultaHotelTest: Prueba que se puedan realizar operaciones en la tabla Hotel.

ConsultaVueloTest: Prueba que se puedan realizar operaciones en la tabla Vuelo.

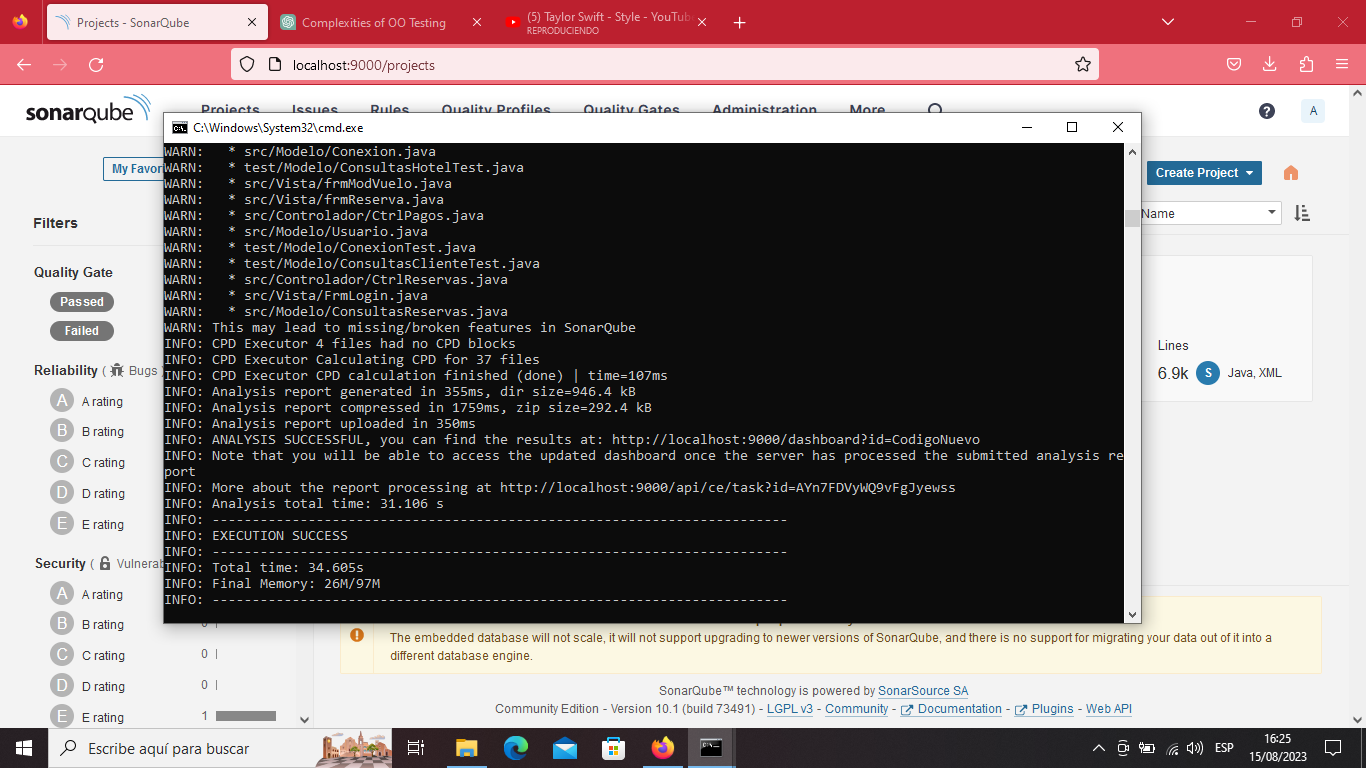
Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

frmClienteTest: Prueba que se pueda buscar un cliente en la base según los datos ingresados.



Posteriormente, se procedió a la evaluación del código fuente del proyecto mediante pruebas de testing estático sobre el código fuente del proyecto utilizando la herramienta SonarQube.



La finalidad de estas pruebas consistía en identificar posibles "code smells" y evaluar la calidad del código en términos de mantenibilidad, legibilidad y buenas prácticas de programación.

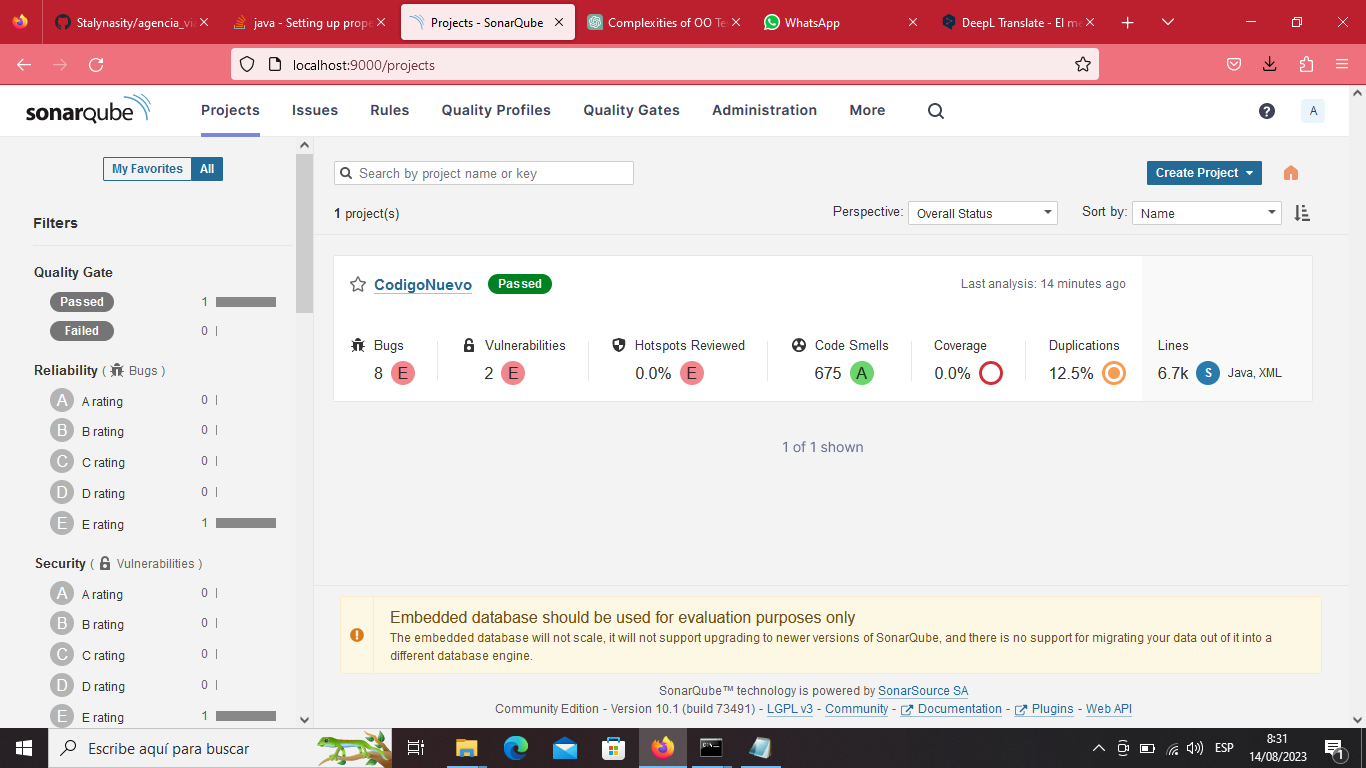


Figura 1. Captura de Pantalla del Primer Reporte.

Durante el proceso de evaluación con SonarQube, nuestro equipo se esforzó en abordar y reducir los "code smells" y bugs identificados en el código bajo análisis. Estos patrones de código problemáticos fueron atendidos en la medida de lo posible, con el objetivo de mejorar la calidad general y la legibilidad del código.

Se implementaron mejoras y correcciones en diversas áreas, lo que permitió obtener resultados más favorables en el informe de SonarQube en comparación con la primera evaluación. Sin embargo, es importante señalar que, dado que el código original ya estaba funcionando y formaba parte de un sistema existente, se tomó la decisión de no realizar modificaciones que pudieran comprometer el funcionamiento actual del sistema.

Algunas de las mejoras realizadas incluyeron la optimización de la lógica, la corrección de errores evidentes y la adopción de buenas prácticas de programación. Si bien no fue posible abordar todos los aspectos señalados por SonarQube debido a la necesidad de mantener la estabilidad del sistema evaluado, se lograron avances significativos que contribuyeron a una mayor calidad y legibilidad del código. A continuación, imagen del nuevo reporte:

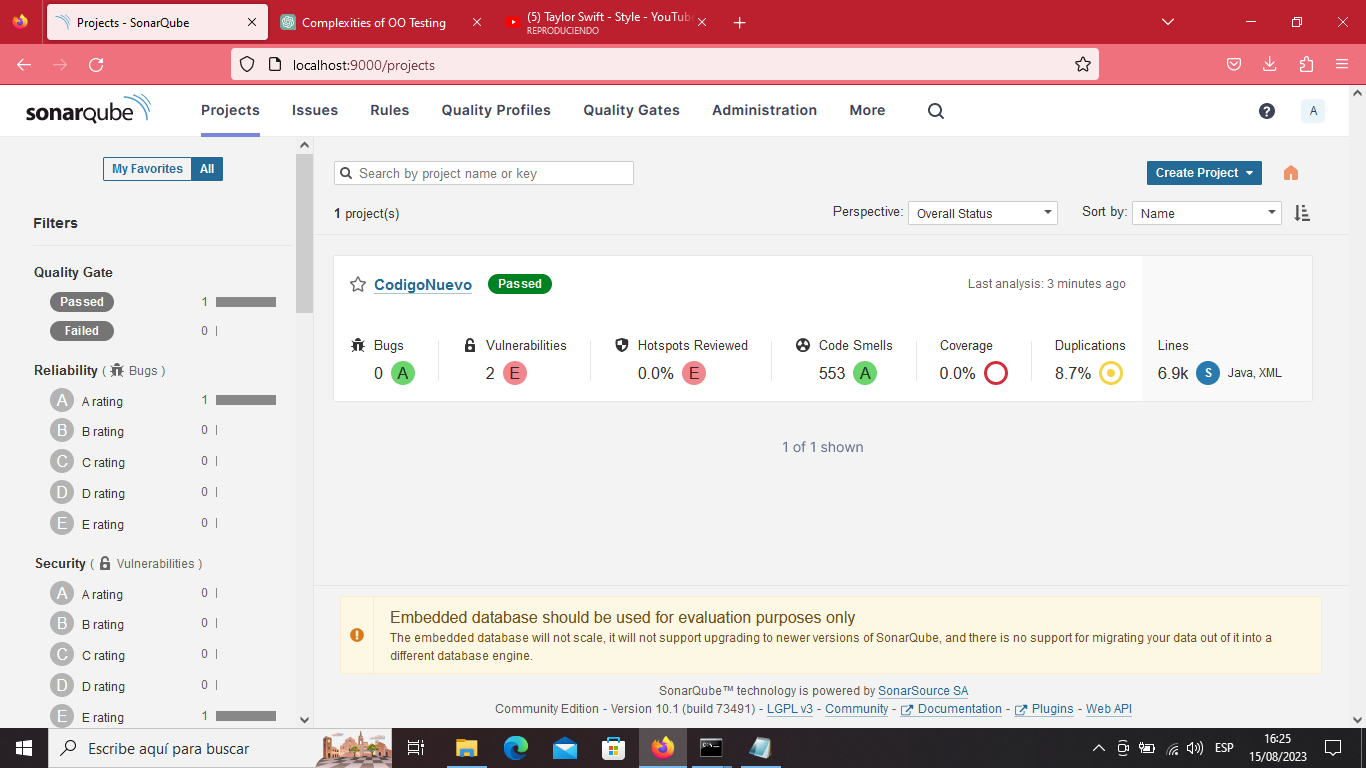


Figura 2. Captura de Pantalla del Nuevo reporte.

Uno de los problemas relevantes encontrados en este proceso se refiere a la cobertura de código y la ejecución de pruebas unitarias utilizando JUnit. A pesar de que se realizaron modificaciones en el proyecto para abordar este problema, se constató que la cobertura de código sigue siendo insuficiente debido a que JUnit no se carga adecuadamente en algunos módulos. Esta situación impacta negativamente en la capacidad de evaluar exhaustivamente la funcionalidad implementada.

En general, se observa que, si bien se han realizado esfuerzos para mejorar la calidad del código, aún persisten aspectos que requieren atención y optimización. La presencia de múltiples "code smells" y la problemática de cobertura de código señalan la necesidad de una revisión más profunda por parte del equipo evaluado, a fin de resolver de manera integral estas incidencias y lograr un estándar de calidad óptimo en el código del software evaluado.

# **Testing Dinámico de Caja Negra.**

En esta sección, nos adentraremos en el emocionante mundo del Testing Dinámico de Caja Negra, una metodología esencial para evaluar la robustez y el comportamiento de nuestro software desde una perspectiva funcional. A través de una serie de técnicas clave, exploraremos cómo poner a prueba diferentes aspectos de nuestro sistema sin requerir un profundo conocimiento de su implementación interna.

Empezaremos abordando las Clases de Equivalencia, una técnica que nos permite agrupar conjuntos de datos de entrada en categorías representativas. Identificaremos las condiciones límite y de frontera, y crearemos casos de prueba que abarquen escenarios diversos y relevantes.

## **Clases de Equivalencia.**

**El programa recibe como entrada un archivo, cuyo formato de registro de hotel es la siguiente:**

**nombre-Hotel:** campo alfanumérico de 30 caracteres

**ID-Hotel:** campo de enteros positivos de 3 dígitos (se excluye el 000)

**Numero-Habitaciones:** campo de enteros positivos de 1 dígitos (se incluye el 0)

**disponibilidad-Hotel:** campo de unos dos caracteres que puede contener si el hotel está Disponible o No si no lo es.

**El programa asigna una prima a cada empleado en base a las siguientes reglas:**

A: corresponde a los Hoteles con nombre de 30 o menos caracteres

B: corresponde a los no Disponibles con 2 o más habitaciones

C: corresponde a los disponibles con hasta 1 o más habitaciones

D: corresponde al ID del hotel con 3 dígitos menores que 1000

|  |  |  |
| --- | --- | --- |
| Condición de Entrada Analizada | Clases Válidas | Clases No Válidas |
| Nombre del Hotel (campo alfanumérico de 30 caracteres) | 1.- 30 caracteres | 2.- Mas de 30 caracteres |
| ID de Hotel (campo de enteros de 3 dígitos) | 3.- 001, 999 (cualquier valor positivo < 1000) | 4.- 000, 1000 (cualquier valor negativo) |
| Número de Habitaciones (Campo de un carácter) | 5.-1, 4 (cualquier valor positivo < 4) | 6.- 0, 5 (cualquier valor negativo) |
| Disponibilidad del hotel (campo de dos caracteres) | 7.- Solo los caracteres "Si" y "No" son válidos | 8.- Cualquier otro valor = "Ys", "SS", "N" |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Nombre del Hotel | ID de Hotel | Número de Habitaciones | Disponibilidad dl hotel | Clases Válidas Cubiertas | Clases No Válidas Cubiertas |
| HOTEL GUAYAQUIL | 22 | 2 | Si | 1 ,3,5,7 | -------- |
| HOTEL QUITO | 23 | 3 | NN | 1,3,5 | 8 |
| HOTEL QUITO | 1000 | 5 | No | 1,7 | 4,6 |

## **Valores Limites.**

Luego, nos sumergiremos en la técnica de Valores Límite, donde exploraremos cómo evaluar el comportamiento de nuestro software en los límites y alrededor de los valores extremos. Detectaremos posibles problemas que puedan surgir en condiciones críticas y aseguraremos un rendimiento consistente y confiable.

|  |  |  |  |
| --- | --- | --- | --- |
| Condición de Entrada Analizada | Regla | Clases Válidas | Clases No Válidas |
| Nombre del Hotel (campo alfanumérico de 30 caracteres) | La longitud del nombre del Hotel es igual a 30 caracteres | 9.- HOTEL GUAYAQUIL, HOTEL QUITO | 10.- HOTEL UNIVERSITARIO DEL SECTOR GUAYAQUIL |
| ID de Hotel (campo de enteros de 3 dígitos) | 001 ≤ ID-Hotel ≤ 999 | 11.- 001,999 | 12.- 000, 1000 |
| Número de Habitaciones (Campo de un carácter) | 1 ≤ número de habitaciones ≤ 4 | 1, 4 | 0, 5 |
| Disponibilidad del hotel (campo de dos caracteres) | Esta-disponible = "Si" OR Esta-disponible = "No" | “Si”, “No” | “Ys”, “N”, “SS” |

* **Diagrama de Transición de Estado.**

A continuación, utilizaremos los Diagramas de Transición de Estado para modelar y evaluar diferentes estados y transiciones dentro de nuestro software. Esta técnica nos permitirá verificar la lógica y la coherencia del flujo de nuestro sistema en respuesta a diversas entradas y eventos.

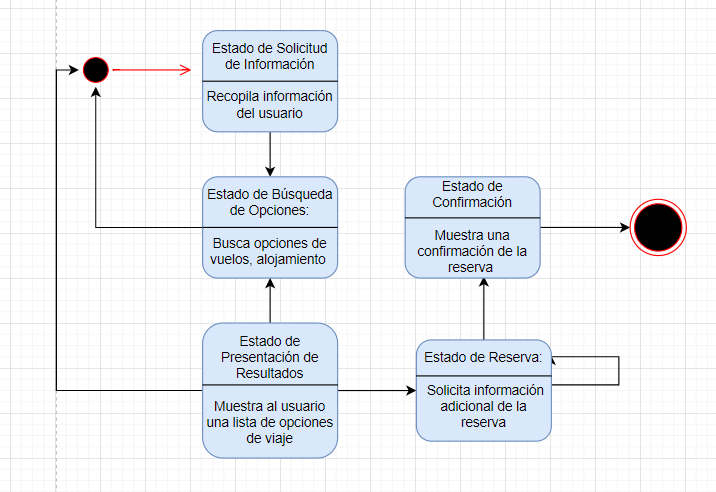


Imagen que contiene interior, foto, tabla, mucho

Descripción generada automáticamente

Diagrama

Descripción generada automáticamente

## **Tabla de Decisión.**

La Tabla de Decisión será nuestra siguiente herramienta, permitiéndonos mapear combinaciones complejas de condiciones y decisiones. Exploraremos cómo diseñar casos de prueba basados en escenarios lógicos y cómo asegurar que todas las posibles combinaciones sean probadas exhaustivamente.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Condiciones** | **1** | **2** | **3** | | **4** | **5** | | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** | **16** |
| ¿Nombre del hotel valido? | Y | Y | Y | Y | | | Y | Y | Y | Y | N | N | N | N | N | N | N | N |
| ¿Id de hotel valido? | Y | Y | Y | Y | | | N | N | N | N | Y | Y | Y | Y | N | N | N | N |
| ¿Es disponibilidad valido? | Y | Y | N | N | | | Y | Y | N | N | Y | Y | N | N | Y | Y | N | N |
| ¿Número de habitaciones valido? | Y | N | Y | N | | | Y | N | Y | N | Y | N | Y | N | Y | N | Y | N |
| **Acciones** |  |  |  |  | | |  |  |  |  |  |  |  |  |  |  |  |  |
| ¿Asignar hotel? | Y | Y | Y | Y | | | N | N | N | N | N | N | N | N | N | N | N | N |
| ¿Aprobar reserva? | Y | N | N | N | | | N | N | N | N | Y | N | N | N | Y | N | N | N |
| Mostrar mensaje de error | N | Y | Y | Y | | | Y | Y | Y | Y | Y | Y | U | Y | Y | Y | Y | Y |

## **Casos de Uso.**

Por último, trabajaremos con Casos de Uso, una técnica que nos ayuda a validar la funcionalidad del software desde la perspectiva del usuario final. Identificaremos los casos de uso clave y crearemos pruebas que simulen interacciones del mundo real, garantizando que nuestro software se comporte según lo previsto.

1. Actualizar Reserva.

Diagrama

Descripción generada automáticamente

1. Diagrama

   Descripción generada automáticamenteHacer reserva.
2. Desconexión a la base de datos

Imagen que contiene Texto

Descripción generada automáticamente

1. Texto, Carta

   Descripción generada automáticamenteCerrar sesión
2. Diagrama

   Descripción generada automáticamenteBuscar vuelos
3. Diagrama

   Descripción generada automáticamenteBuscar clientes
4. Iniciar sesión

Diagrama

Descripción generada automáticamente

1. Eliminar registros de vuelos

Diagrama

Descripción generada automáticamente

1. Eliminar registros de hoteles

Diagrama

Descripción generada automáticamente

1. Eliminar registros de clientes

Diagrama

Descripción generada automáticamente

1. Actualizar cliente

Diagrama

Descripción generada automáticamente

1. Modificar registros de hoteles

Diagrama

Descripción generada automáticamente

1. Conexión a la base de datos

Diagrama

Descripción generada automáticamente

1. Modificar registros de vuelos

Diagrama

Descripción generada automáticamente

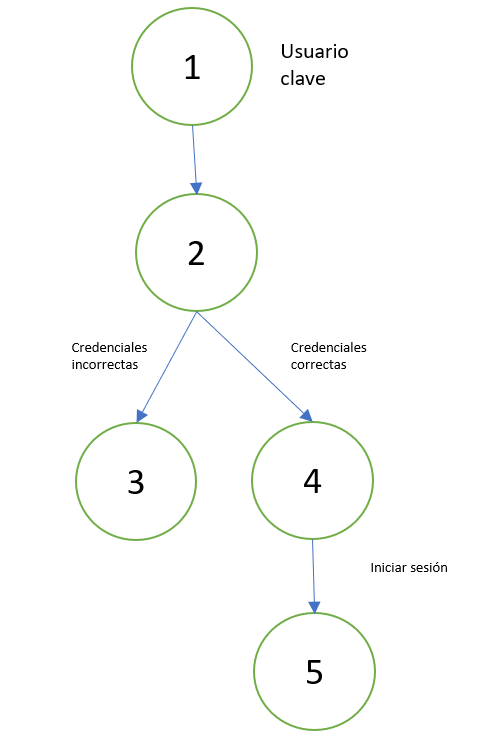
Este enfoque en el testing funcional nos ha permitido explorar diferentes aspectos de nuestro software, identificar posibles problemas y verificar que cumple con los requerimientos y expectativas establecidos. Al aplicar estas técnicas, hemos obtenido una visión más completa de cómo nuestro software se comporta en diversas situaciones y cómo interactúa con los usuarios.

# **Testing Dinámico de Caja Blanca.**

## Criterios de caminos

Estos criterios se utilizan para medir qué tan bien diseñados y accesibles son los flujos de interacción dentro de una interfaz. Evalúan aspectos como la claridad de la ruta que un usuario debe seguir para realizar una tarea específica, la coherencia de las acciones y opciones disponibles en cada paso del camino, la cantidad de pasos necesarios para completar una tarea, la eficiencia de la navegación y la ausencia de obstáculos innecesarios.

Comprobación de inicio de sesión:



Todos los caminos:

[1 ,2 ,3]

[1 ,2, 4, 5]

## Bucles

Es una sentencia que se realiza repetidas veces a un trozo aislado de código, hasta que la condición asignada a dicho bucle deje de cumplirse. Es utilizado para hacer una acción repetida sin tener que escribir varias veces el mismo código, lo que ahorra tiempo deja el código más claro y facilita su modificación en el futuro.

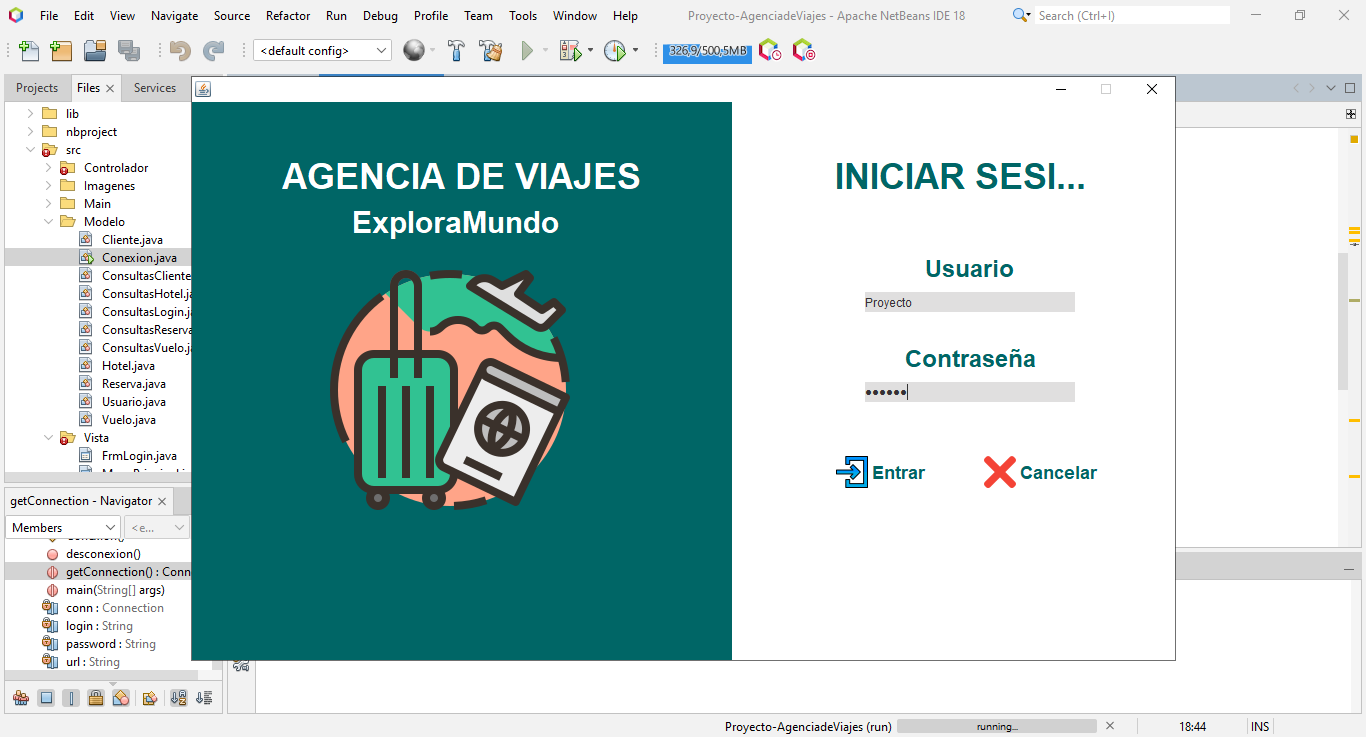
Validar usuario

Usuario y clave registrados: mostrar mensaje de bienvenido

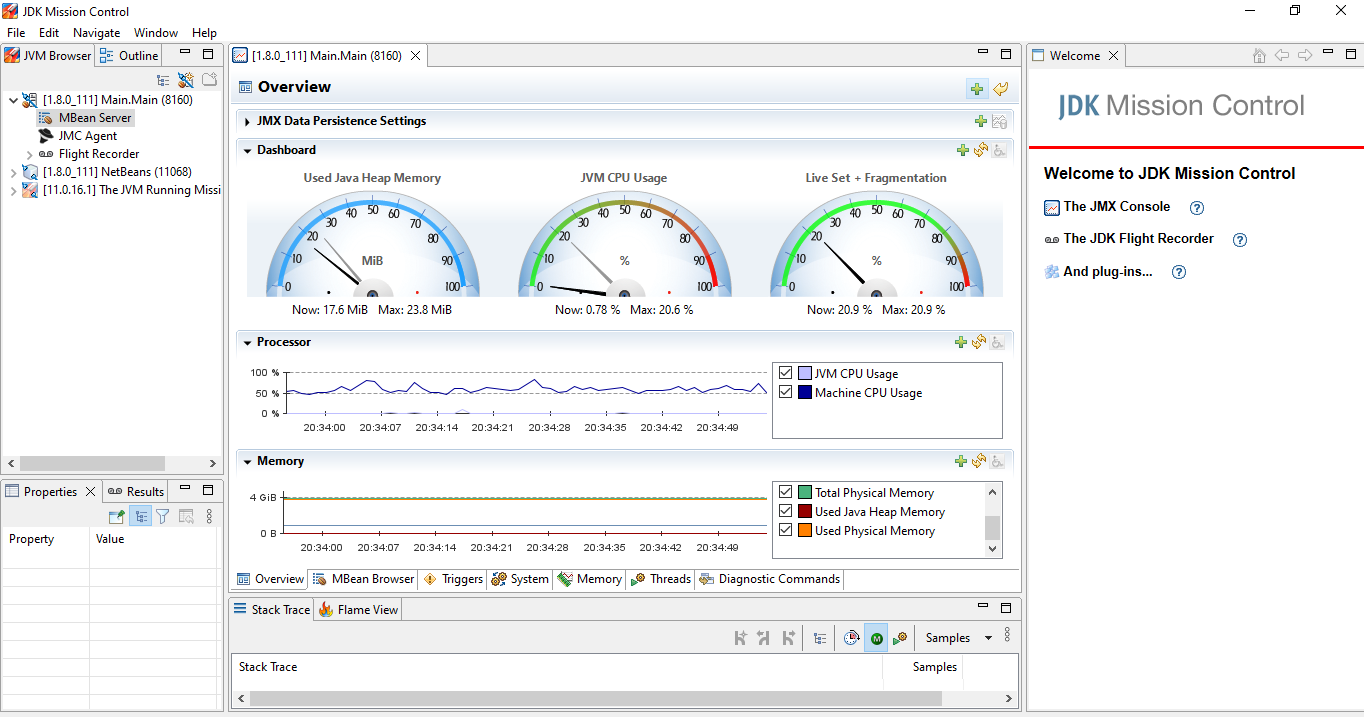
Usuario y claves no registrados: mostrar mensaje de credenciales incorrectas

# **Testing de Rendimiento del Software.**

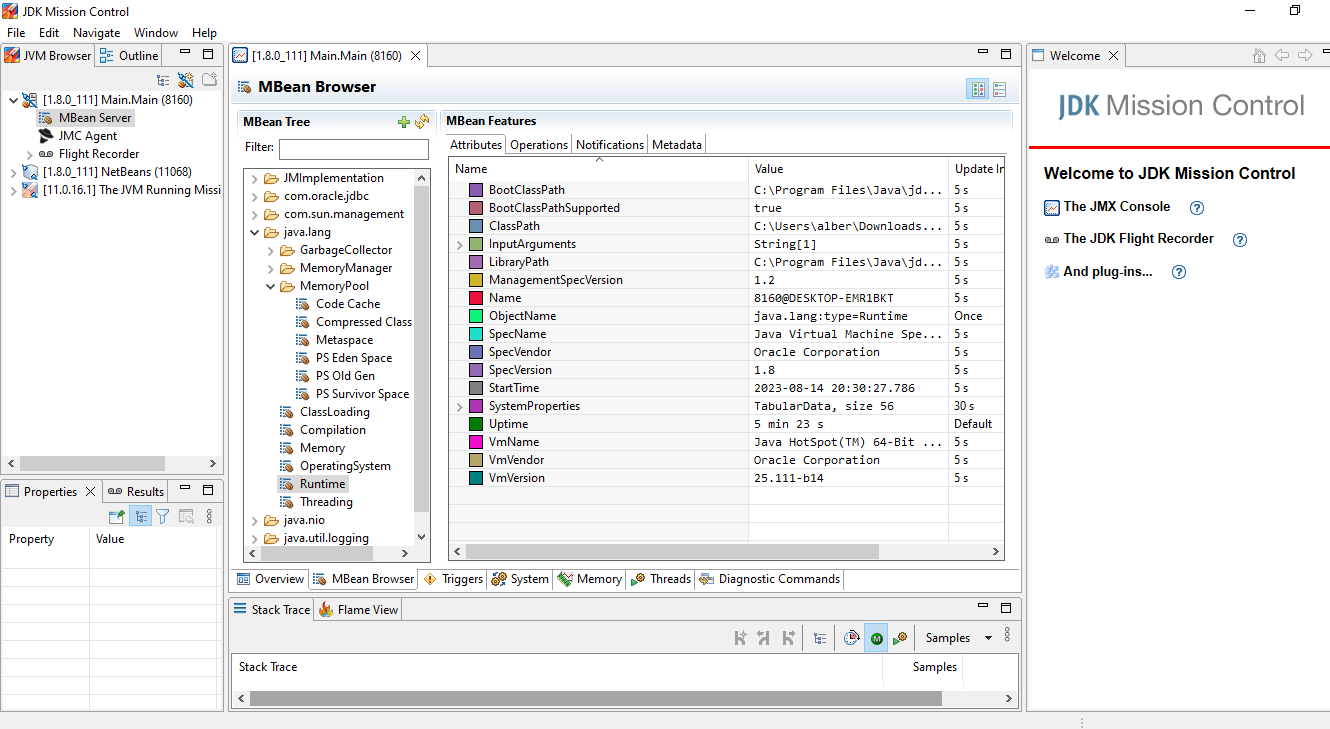
En esta sección, abordaremos el crucial aspecto del rendimiento del software, evaluando su capacidad para manejar cargas y tareas bajo diversas condiciones. Mediante herramientas y técnicas especializadas, se analizarán y medirán los tiempos de respuesta, la eficiencia de los recursos y la escalabilidad del sistema.

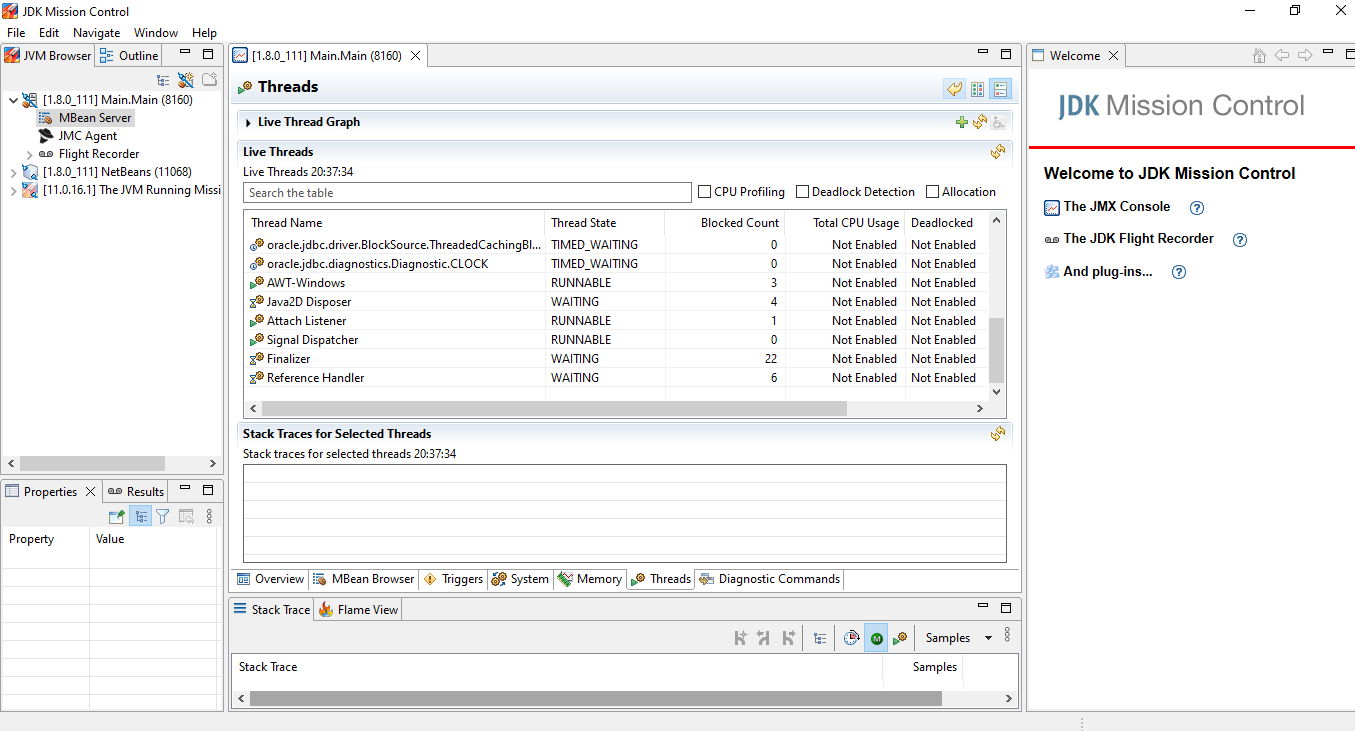


La siguiente imagen muestra los medidores de rendimiento en la interfaz de JDK Mission Control. Estos medidores proporcionan visualizaciones en tiempo real de indicadores clave de rendimiento, como la carga de la CPU, la utilización de memoria y otros recursos críticos. Esta información nos permite identificar rápidamente posibles cuellos de botella y áreas de mejora en el rendimiento del sistema.



Estos MBeans proporcionan información detallada sobre diversos aspectos internos del sistema, lo que nos ayuda a identificar posibles problemas y optimizar el rendimiento.





En esta sección dedicada al Testing de Rendimiento del Software, hemos explorado a fondo la evaluación de uno de los aspectos más cruciales en el desarrollo de aplicaciones y sistemas: su capacidad para funcionar eficientemente bajo diferentes condiciones de carga y uso. A lo largo de este análisis, hemos empleado herramientas especializadas y técnicas específicas para medir y evaluar diversos aspectos del rendimiento.

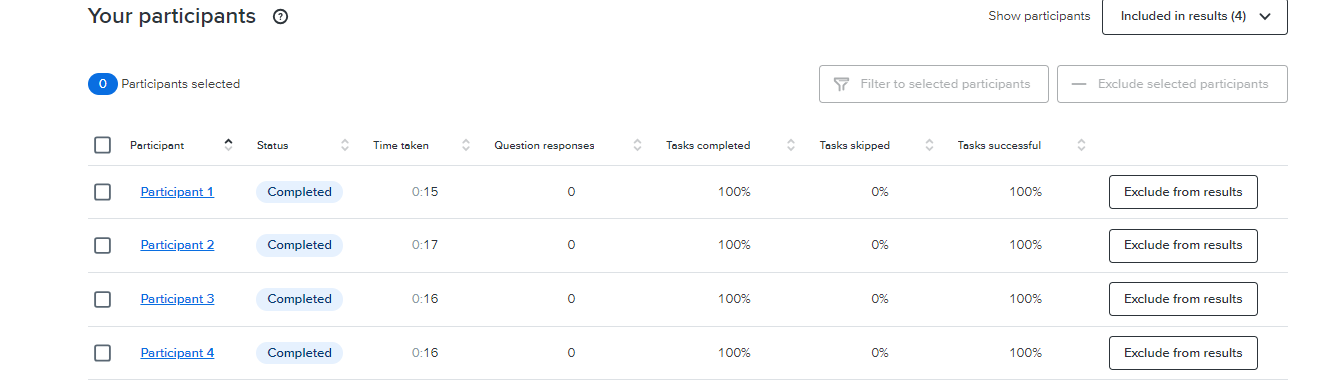
Hemos examinado detalladamente imágenes que capturan diferentes etapas del proceso de evaluación de rendimiento, desde la ejecución normal del software hasta la utilización de la herramienta JDK Mission Control para analizar métricas críticas. Las imágenes han proporcionado una visión visual de cómo monitorear y mejorar el rendimiento de nuestro software.

En la siguiente sección, abordaremos la evaluación de usabilidad del software, lo que nos permitirá asegurarnos de que la aplicación no solo sea eficiente desde el punto de vista técnico, sino también cómoda y satisfactoria para los usuarios finales.

# **Testing de Usabilidad del Software.**

## **Software para testing de Usabilidad.**

**Para el testing de usabilidad se optó por usar la herramienta Optimal Workshop. Optimal Workshop se trata de una plataforma enfocada en la investigación y el diseño orientado hacia la experencia del usuario. Ofrece recursos y resoluciones destinados a asistir a equipos de diseño y desarrollo en la optimización de la facilidad de uso y el rendimiento de sus productos digitales.**

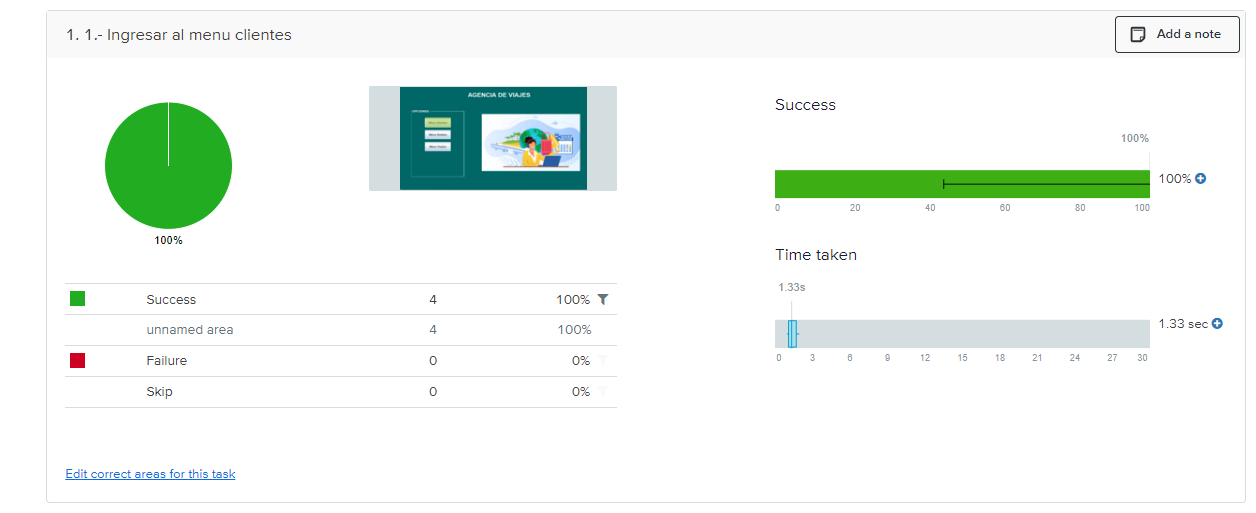
****

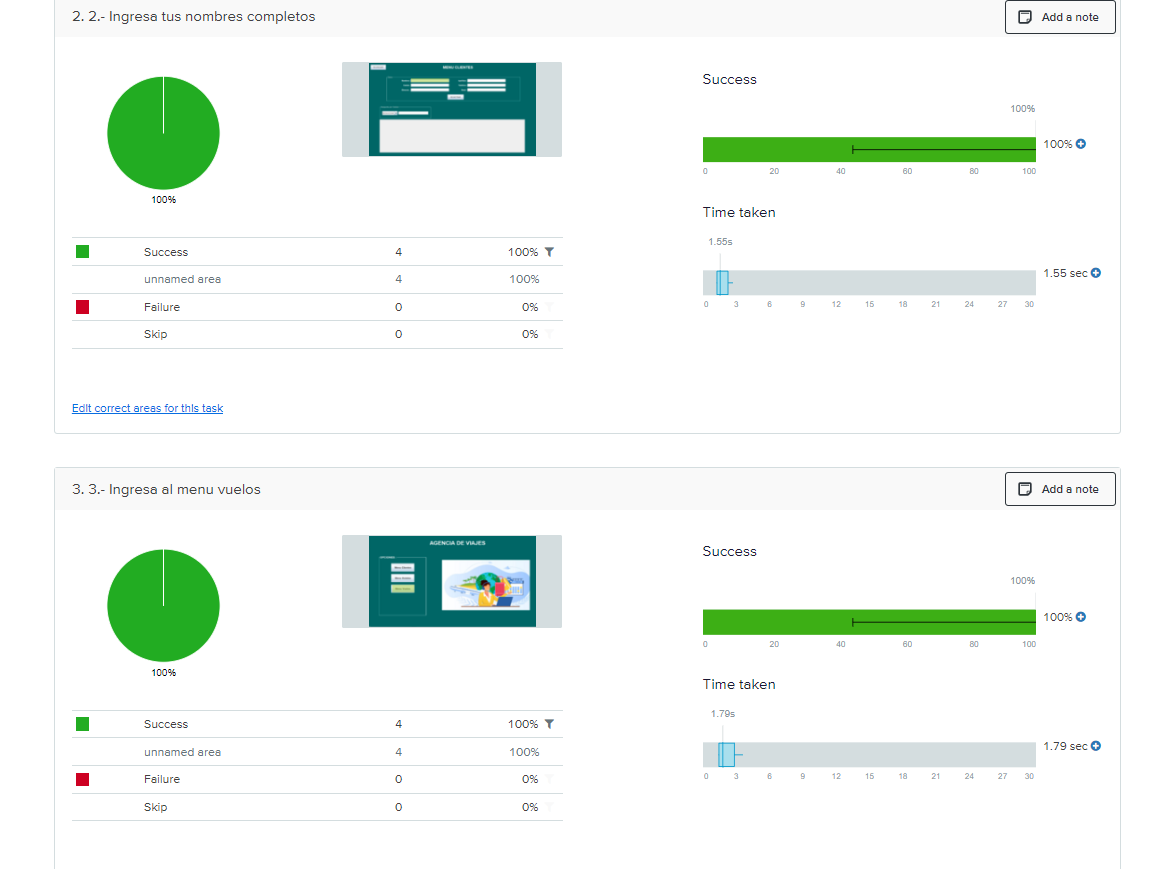
**Sus herramientas de pruebas de usabilidad están diseñadas para ayudar a investigadores, diseñadores y arquitectos de la información a mejorar la experiencia del usuario de sus productos.**

## **Testing con Optimal Worshop**

**Para el testing se usaron 4 participantes que tenían que cumplir con 3 tareas, ingresar al menú clientes, ingresa tus nombres y la registrar el cliente.**

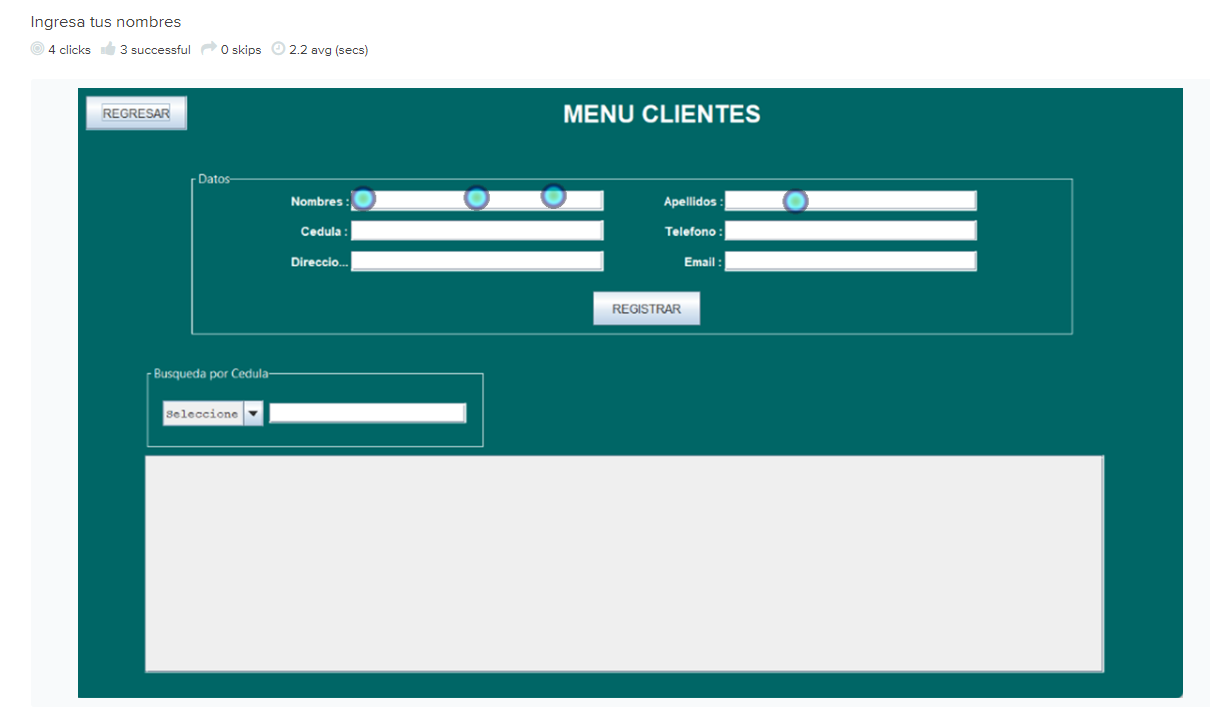
**Luego de que los clientes completaron las tareas se obtienen los siguientes resultados:**

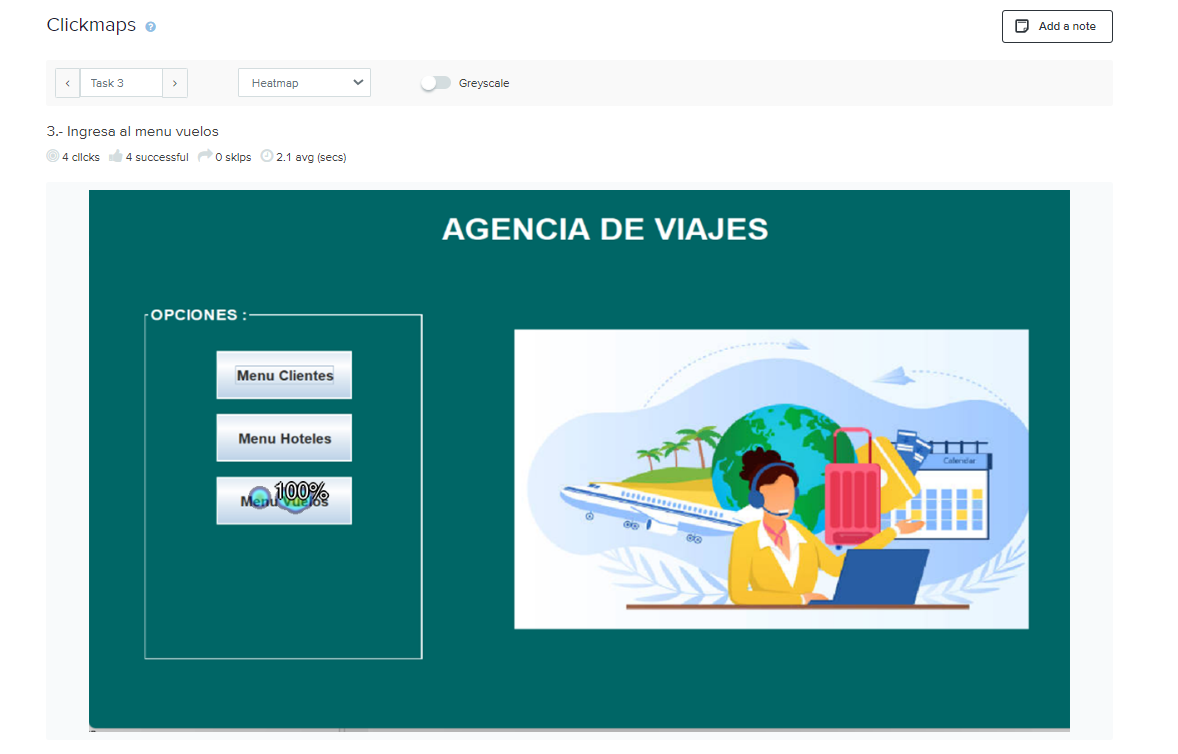
****

****

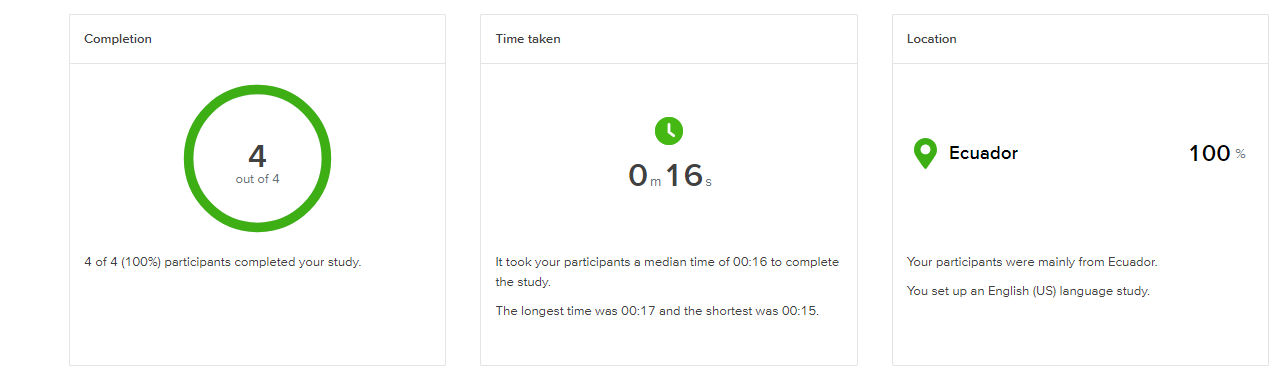
**Mapas de calor que indican las zonas en las que el usuario realizo interacción:**

****

****

****

**Resultados generales**



## **Análisis del testing.**

En la etapa de pruebas y evaluación podemos observar resultados excelentes. De manera específica, al poder analizar los resultados del testing en el que estuvieron involucrados los cuatro usuarios, podemos llegar a la conclusión de que todos los usuarios pudieron culminar todas las tareas que fueron asignadas, en un intervalo de tiempo de estimado de 16 segundos.

## **Conclusiones.**

Con estos resultados llegamos a la conclusión de que la interfaz tiene eficiencia y usabilidad para los usuarios. Lo que se puede llegar a notar es que posee intuición siendo así que facilita la comprensión de las tareas establecidas.

El resultado obtenido refleja que el enfoque ha sido pensado especialmente para los usuarios, al poder mostrar elementos visuales que muestren una forma de navegación lógica y fluida, lo que nos da a entender que se ha creado un entorno que no solo cumple con las expectativas del usuario, sino que además supera las necesidades tanto como de accesibilidad y rendimiento.

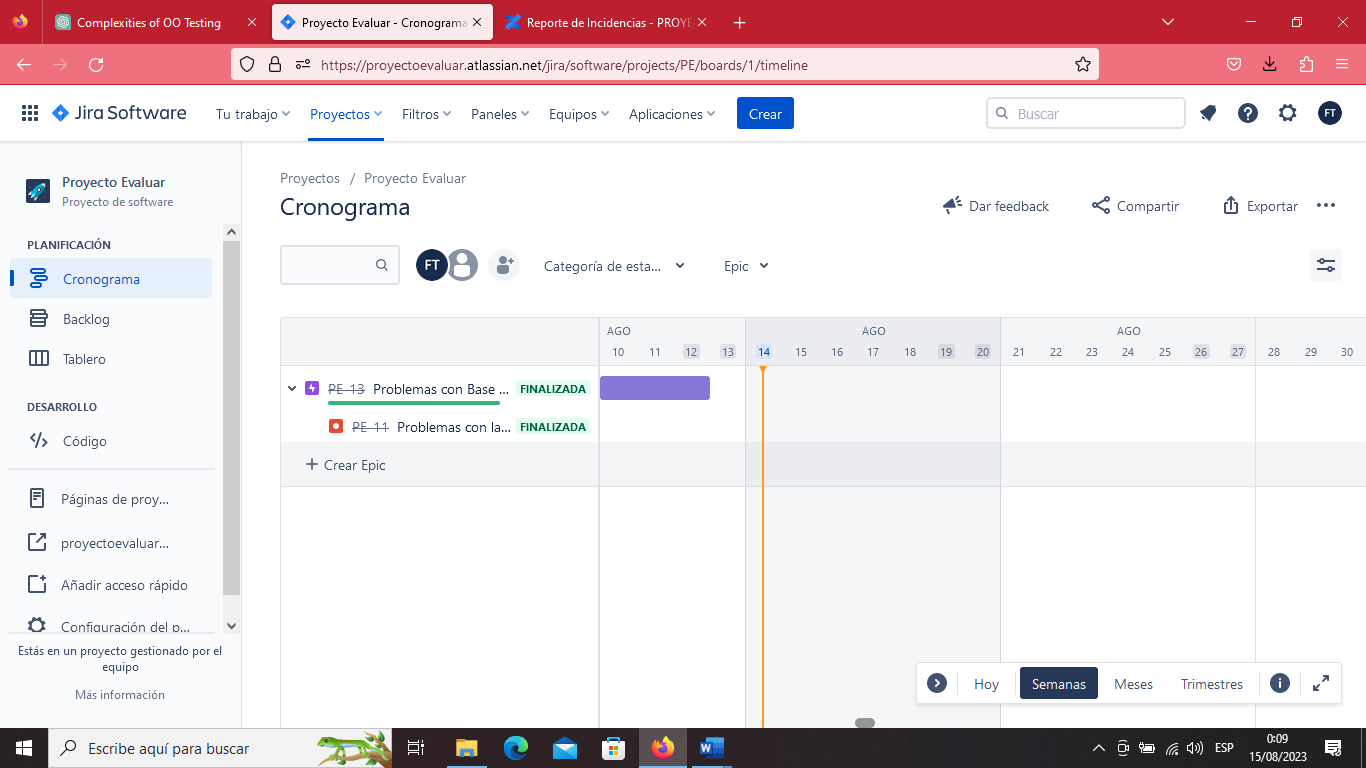
# **Reporte de Incidencias.**

|  |  |
| --- | --- |
| Colaboradores | * Aaron Álvarez Llamuca * Fausto Torres Aspiazu * Ivonne Roxana Minchala Plúas * María Belén Vargas * Milena Cruz Diaz * Jordy Figueroa Martínez |
| Objetivo | El objetivo principal de la evaluación y testing es garantizar la calidad y funcionalidad del sistema software a través de enfoques de evaluación exhaustivos y pruebas rigurosas. |
| Fecha de vencimiento | 15 de agosto del 2023 |
| Principales resultados | * Documentación detallada de todas las incidencias encontradas. * Pruebas dinámicas realizadas según las técnicas de caja negra y caja blanca especificadas. * Resultados de pruebas de rendimiento y usabilidad. |
| Estado | completado |

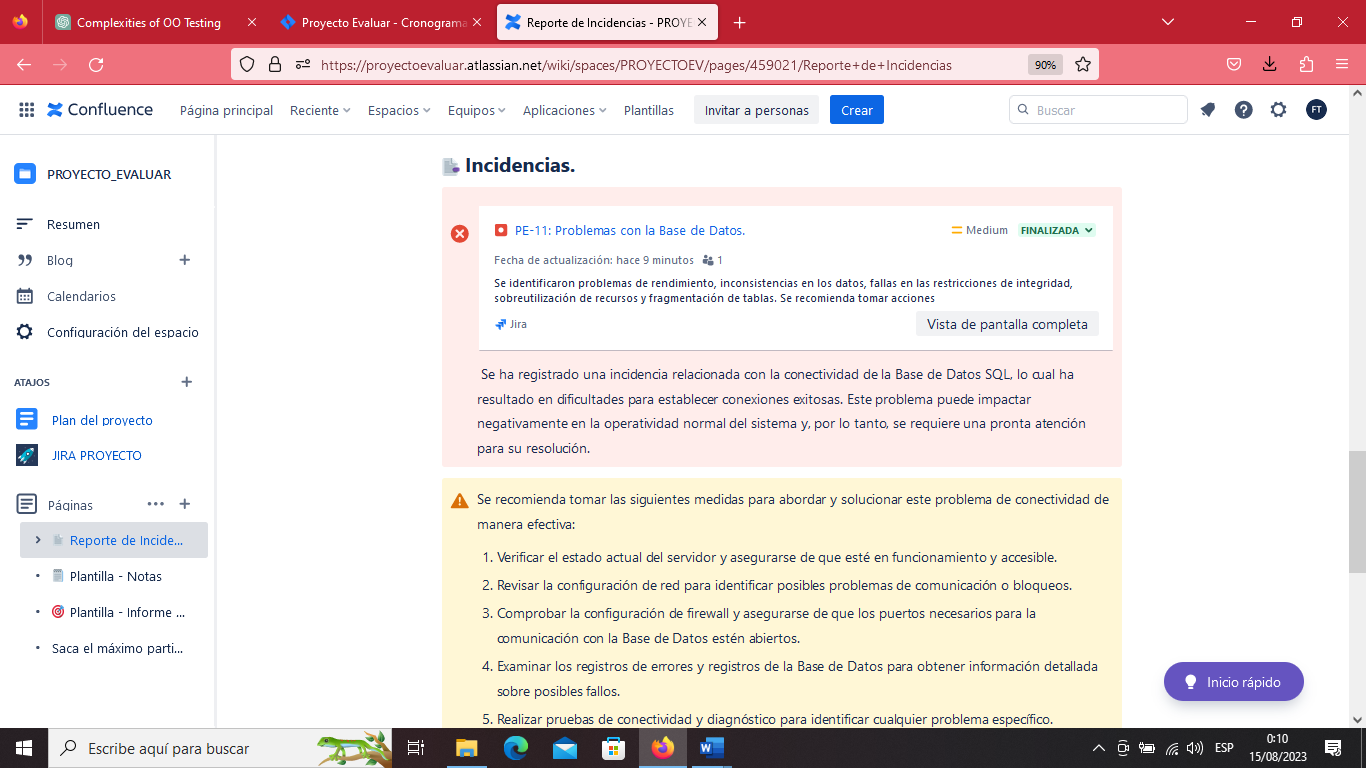
**Alcance**

|  |  |
| --- | --- |
| **Debe tener:** | * Descripción precisa de cada incidencia, incluyendo pasos para reproducirla y detalles relevantes. * Clasificación de las incidencias según su gravedad o impacto en el sistema. * Identificación de los módulos o componentes del software afectados por cada incidencia. |
| **Podría tener:** | * Capturas de pantalla o evidencia visual que respalde y clarifique la descripción de cada incidencia. * Recomendaciones preliminares para resolver o mitigar las incidencias identificadas. |
| **Fuera del alcance:** | * Aspectos no relacionados directamente con las incidencias encontradas durante la ejecución del sistema. * Inclusiones de cambios o mejoras no vinculados a las incidencias actuales. * Detalles sobre aspectos del sistema que no se han evaluado en el contexto de esta ejecución. |

En esta imagen, se muestra una captura de pantalla del calendario en la plataforma Jira, donde se registra la incidencia. Se puede observar la fecha y hora en la que se detectó el problema de conectividad con la Base de Datos SQL.



Esta imagen presenta un extracto del reporte de incidencia en la herramienta Jira. En la captura se resalta la descripción detallada del error de conectividad con la Base de Datos SQL. Se incluyen datos relevantes como la fecha y hora de ocurrencia, los pasos para reproducir el problema y el impacto en el sistema. Esta captura documenta el proceso inicial de identificación y registro de la incidencia para su análisis y resolución.



# **Repositorio.**

Enlace: <https://github.com/Fausto2002/Proyecto_Evaluar-Grupo2>