

DOCUMENTACIÓN FINAL PROYECTO CIVU

Alexis Diego Timaná Romero. 1870385
Luis Felipe Valencia Valencia. 1824494
Luis Eduardo Ruiz Castaño. 1831986
Laura Valentina Quintero Reyes. 1841533

Contenido

Contenido	2
Índice de figuras	4
Índice de tablas	4
Introducción	5
Descripción del cliente y usuarios finales	6
Descripción del problema	6
Sistema Propuesto	6
Propósito del sistema	6
Alcance del sistema	6
Sprint cero	7
Equipo de desarrollo, roles e integrantes	7
Ceremonias de Scrum	7
Herramientas y tecnologías usadas	8
Planeación inicial de los Sprints	8
Módulos principales del sistema	8
Ingeniería de Requerimientos	9
Requerimientos funcionales	9
Requerimientos no funcionales	9
Gestión de Proyectos de Software	10
Planeación y manejo de riesgos	10
Estimación de los primeros módulos del proyecto	10
Diseño de software	10
Arquitectura del proyecto	10
Diagramas de la arquitectura	11
Patrones de construcción de software	13
Patrones de comportamiento	13
Patrón creacional	14
Calidad de software	15
Atributos de calidad	15
Estándares de calidad	16
Diseño de casos de prueba	17
Pruebas unitarias	17
Pruebas funcionales	19
Pruebas no funcionales	20
Ejecución de casos de prueba	21
Pruebas unitarias	21

Pruebas funcionales	22
Pruebas no funcionales	29
Reporte de ejecución de pruebas	30
Resumen de pruebas y análisis	31
Referencias	32

Índice de figuras

Figura 1: relaciones en el patrón MVC.	11
Figura 2: Diagrama de arquitectura por capas.	11
Figura 3: Mapa de navegación de la interfaz de usuario.	12
Figura 4: Diagrama de módulos de la aplicación.	12
Figura 5: Diagrama de modelo relacional de datos.	12
Figura 6: Diagrama de clases del patrón observador.	13
Figura 7: Fragmento de código aplicando el patrón observador.	13
Figura 8: Diagrama de clases del patrón command.	14
Figura 9: Fragmento de código aplicando el patrón command.	14
Figura 10: Diagrama de clases del patrón Builder. página	15
Figura 11: Interfaces creadas aplicando el patrón Builder.	15
Figura 12: Método lógico.	17
Figura 13: Grafo de flujo.	18
Figura 14: Diagrama de flujo.	18
Figura 15: Construcción de escenarios para pruebas unitarias en java.	21
Figura 16: Métodos de ejecución de pruebas unitarias.	22
Figura 17: Resultados de las pruebas unitarias, Junit.	22
Figura 18: Prueba de validación para el campo usuario (Campo vacío).	23
Figura 19: Prueba de validación para el campo usuario (Ingreso de letras).	23
Figura 20: Prueba de validación para el campo contraseña (Campo vacío).	24
Figura 21: Fallo en prueba de validación para el campo contraseña (Campo vacío).	24
Figura 22: Prueba de validación para el campo nombre (Campo vacío).	25
Figura 23: Prueba de validación para el campo fecha (Campo vacío).	25
Figura 24: Prueba de validación para el campo id (Campo vacío).	26
Figura 25: Prueba de validación para el campo teléfono (Campo vacío).	26
Figura 26: Prueba de validación para el campo id (Ingreso de letras).	27
Figura 27: Prueba de validación para el campo nombre (Ingreso de números).	27
Figura 28: Fallo en prueba de validación para el campo teléfono (Ingreso de números extra).	28
Figura 29: Fallo en prueba de validación para el campo teléfono (Ingreso de letras).	28
Figura 30: Prueba de validación para casos válidos en todos los campos.	29
Figura 31: Registro exitoso con todos los campos validados.	29
Figura 32: Implementación y pruebas no funcionales en Java.	30

Índice de tablas

Tabla 1: Casos de prueba unitarios.	19
-------------------------------------	----

Introducción

Este documento tiene como objetivo presentar la documentación elaborada a lo largo de la ejecución del proyecto para la asignatura desarrollo de software II. El proyecto consiste en la elaboración de un sistema de software con varios módulos, utilizando una metodología ágil y sus respectivas prácticas ágiles desde la etapa inicial de diseño hasta la implementación, despliegue y mantenimiento.

Primero se realiza una descripción del proyecto, luego se define, describe e ilustra la arquitectura utilizada, se definen las tecnologías utilizadas para la elaboración, ejecución y evaluación del proyecto, así como también se describen los estándares de calidad aplicados y los atributos de calidad a evaluar para cumplir con dichos estándares. También se muestra el diseño, la ejecución y los resultados de las pruebas de software realizadas para evaluar el proyecto, esto mediante un reporte de pruebas.

Por último, el desarrollo del proyecto fue guiado en todo momento por las metodologías y herramientas vistas en clase.

Descripción del cliente y usuarios finales

Nuestro cliente es una pequeña empresa de venta de calzado llamada Calzado Eze, ubicada en Samaniego (Nariño), la cual cuenta con solo una sede y se dedica al comercio al por menor. Los usuarios finales serán el administrador de la empresa y los vendedores, quienes son usuarios entre principiantes e intermedios de sistemas de software.

Descripción del problema

El cliente necesita un producto de software que cuente con módulos para gestionar el inventario y de ventas del negocio principalmente. Esta empresa nunca ha contado con un sistema automatizado para llevar la contabilidad y el control de sus productos, estas actividades se llevan a mano con registros en papel. Adicionalmente, el equipo de desarrollo propone la implementación de un módulo para guardar y consultar datos de clientes y proveedores y un módulo para generar reportes referentes a las ventas, para terminar de complementar las funcionalidades solicitadas y brindar un mejor servicio.

Sistema Propuesto

Propósito del sistema

El sistema se llama CIVU (Controlador de inventarios, ventas y usuarios), será una aplicación que permita registrar las ventas, el inventario, los empleados, los clientes y entre otros datos para ser almacenados en una base de datos, de tal manera que se pueda generar reportes que permitan a la gerencia conocer el estado actualizado de la empresa en los aspectos mencionados anteriormente, facilitando la toma de decisiones. Para esto se contará con módulos que se comunicarán entre sí y con una base de datos remota para guardar y consultar los datos.

Alcance del sistema

El sistema CIVU será una aplicación que permita registrar:

- Ventas: Se registran los datos del vendedor, los artículos vendidos y datos básicos del cliente, así como un precio total a pagar y la forma de pago (de contado o por sistema de separado).
- Inventario: Se registran todos los detalles de los productos comprados por la empresa a sus proveedores (talla, color, marca, modelo, entre otros).
- Empleados: Se registran datos básicos del empleado como el nombre, número de contacto, cargo, un identificador y una contraseña para ingresar al sistema con su propio usuario.

- Clientes: Se registran datos básicos del cliente como el nombre, un número de contacto, su fecha de nacimiento y un identificador.
- Proveedores: Se registran datos básicos de los proveedores como el nombre, un número de contacto, la dirección y un identificador.

Todos los datos mencionados se guardan en una base de datos remota y esto permitirá generar reportes, tanto en tablas como en gráficas según se requiera. Por lo tanto es un sistema de software dirigido a empresas pequeñas cuya actividad principal sea el comercio al por menor de cualquier tipo de artículo, pues el sistema no está pensado para escalar a una empresa grande debido al servidor donde se aloja la base de datos.

Características no incluidas al proyecto:

- Gestión de sedes para varios almacenes.
- Pagos con diferentes medios (tarjetas débito, crédito).

Sprint cero

Equipo de desarrollo, roles e integrantes

Laura Quintero: Equipo de desarrollo (Frontend)

Luis Ruiz: Scrum master, equipo de desarrollo (Frontend y QA)

Luis Valencia: Equipo de desarrollo (Backend)

Alexis Timaná: Product owner, equipo de desarrollo (Backend/Tester)

Stakeholders (Cliente): Calzado Eze

Ceremonias de Scrum

- **Sprint planning (1 hora o menos).** Reunión realizada cada semana, al comienzo de cada sprint para analizar el backlog y que el equipo seleccione los ítems que se van a trabajar durante el siguiente sprint.

- **Spring review (1 hora o menos - Finalizando el sprint).** Reunión en la cual el product owner y el Development Team presentan a los stakeholders el incremento terminado para que sea aceptado o ajustar cambios. Se realiza cada semana al finalizar los sprint, prácticamente se une con el sprint planning.

- **"weekly" scrum meeting (Media hora - miércoles tarde/noche).** Reunión realizada cada semana en la que participan solamente los miembros del equipo de desarrollo para aclarar que se ha hecho en los sprints o si se presenta algún bloqueo.

- **Grooming (Media hora a una hora - miércoles):** Está práctica nos sirve para estar seguros de que el backlog siempre esté preparado. Con estas reuniones se pretende tener

un control del avance en el proyecto, unificar las distintas partes o módulos y plantear las tareas del siguiente sprint. Se realiza en conjunto con la reunión weekly.

Herramientas y tecnologías usadas

Para este proyecto se va a utilizar la herramienta github para el control de versiones de software ([Enlace](#)), mediante una única rama que será actualizada después de cada sprint. Además se va a utilizar la aplicación Jira para el product backlog y la asignación de tareas, y en general para la gestión del proyecto.

Será un software de escritorio.

Las herramientas de codificación serán:

- BackEnd: Java
- FrontEnd: Java y el plugin windowbuilder.
- Bases de datos: PostgreSQL, Elephant, PgAdmin.

Planeación inicial de los Sprints

La planeación de los sprints (sprint planning) se realizó en la herramienta Jira, teniendo en cuenta un número uniforme de puntos para realizar en cada sprint. Sin embargo, a medida que se marcan como realizadas las historias de usuario, la herramienta las elimina del product backlog, por lo cual se hizo una versión alternativa de éste en una hoja de cálculo.

[Product backlog y los sprints en jira](#)

[Product backlog en hoja de cálculo](#)

Módulos principales del sistema

- Módulo de inventario: Por este módulo será posible ver los productos que la empresa vende, la cantidad que se tiene en stock actualizada. Además se tendrán datos como el precio unitario, color del calzado, tallas, entre otros.
- Módulo de ventas: En este módulo se van a registrar los diferentes productos vendidos, el precio pagado en total y unitario, la cantidad de artículos vendidos y algunos datos del comprador y del vendedor.
- Módulo de gestión de usuarios (clientes y trabajadores): Aquí será posible tanto agregar, como editar datos o eliminar a un trabajador o cliente, se mostraran datos como nombre, documento de identidad, información de contacto, entre otros.
- Módulo de reportes: Con este módulo se puede acceder a una lista con todas las ventas realizadas con datos del vendedor y comprador, reportes sobre las ventas realizadas por empleado, mayor venta realizada en un periodo de tiempo, productos y cantidades que se vendieron en un periodo de tiempo.

- Módulo de validación de usuario: Con este módulo se pretende tener una forma de confirmar la identidad de los usuarios antes de ingresar al sistema, para evitar que personas ajenas a la empresa puedan utilizarlo.

Ingeniería de Requerimientos

Requerimientos funcionales

Los requerimientos funcionales están descritos en el product backlog, allí únicamente se encuentran cuestiones de funcionalidad del sistema.

[Enlace al product backlog y los sprint](#)

Requerimientos no funcionales

Los requerimientos no funcionales identificados dentro del sistema son del producto, restricciones en cuanto al uso por parte de los usuarios, puesto que no tenemos requerimientos de tipo externo u organizacional, ya que el cliente es una empresa ajena a la industria del software. Los requerimientos son:

- Al hacer click en alguna de las funciones de la aplicación, la transición entre las ventanas debe realizarse en menos de 2 segundos.
- El tiempo que tarda un usuario en aprender a manejar el sistema debe ser inferior a 4 horas.
- El software debe ser fácil de actualizar, de modo que solo se deba descargar una nueva versión y eliminar la anterior para obtener las características nuevas o correcciones aplicadas.

Restricciones:

- La información debe estar almacenada en una base de datos remota (en la nube).
- Las credenciales de acceso al sistema pueden ser otorgadas o modificadas únicamente por el administrador del sistema.
- El sistema de software debe funcionar en el sistema operativo Windows.
- El control de versiones se hace únicamente mediante un repositorio en github.

Gestión de Proyectos de Software

Planeación y manejo de riesgos

Se tuvo en cuenta los distintos tipos de riesgos que posiblemente afectarían el desarrollo del proyecto, y aplicando la matriz de riesgos se hizo una estimación de la posibilidad de ocurrencia, los posibles impactos y formas de solucionarlos o mitigarlos.

[Matriz de riesgos](#)

Estimación de los primeros módulos del proyecto

La estimación de tiempo y esfuerzo para realizar los sprints se encuentra en el [product backlog](#), y fue realizada mediante la técnica de planning poker. Aquí se tiene en cuenta el desarrollo de tareas individuales para aportar a la completitud de las historias de usuario.

La estimación de tamaño y recursos utilizados para nuestro producto se encuentra en el documento [Estimación CIVU](#), y fue realizada mediante métodos empíricos y la plantilla de estimación ME proporcionada por la profesora.

Diseño de software

Durante la tercera reunión grupal, finalmente el equipo decidió que se desarrollará una aplicación de escritorio utilizando el lenguaje de programación Java, esto teniendo en cuenta los requerimientos y funcionalidades solicitadas por el cliente, la manera en que se describió el uso que se daría a la aplicación en el ambiente laboral y los conocimientos de los programadores.

Arquitectura del proyecto

Para la arquitectura se está siguiendo el patrón arquitectónico conocido como modelo vista controlador (MVC) el cual separa los datos de la aplicación, la interfaz de usuario y la lógica de control en tres componentes distintos interconectados.

El “modelo” se encarga de acceder a la capa de almacenamiento de datos y de definir la funcionalidad del sistema. La “vista” o “interfaz de usuario” es la información que se le presenta al usuario de forma gráfica y los mecanismos de interacción con el programa. Por último, el “controlador” actúa como un intermediario entre el modelo y la vista gestionando el flujo de información entre ellos.

El siguiente diagrama refleja las relaciones existentes entre los componentes del Modelo, Vista y Controlador, y de éstos a su vez con el usuario, o cliente del sistema:

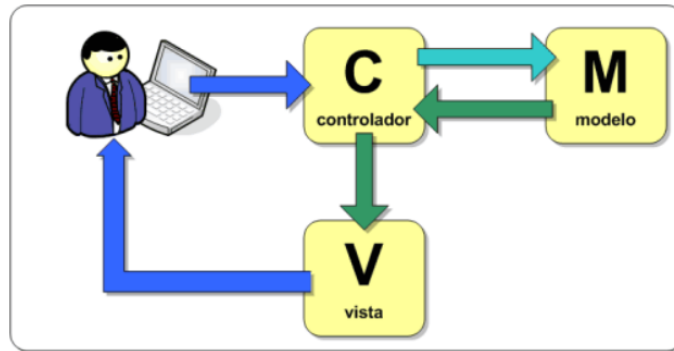


Figura 1: relaciones en el patrón MVC.

Las tecnologías usadas para el desarrollo del proyecto serán:

- Control de versiones mediante git, con repositorio en la nube GitHub.
- Manejo de bases de datos con PostgreSQL, gestor de bases pgadmin y base de datos remota en servidor de Elephant para acceso simultáneo de datos.
- Programación de la lógica y las ventanas del programa con java mediante el IDE Eclipse y el plugin window builder.
- Utilización de librerías como Jasper studio, swing, y Apache poi para el diseño tanto de la interfaz como de los reportes y recibos.

Diagramas de la arquitectura

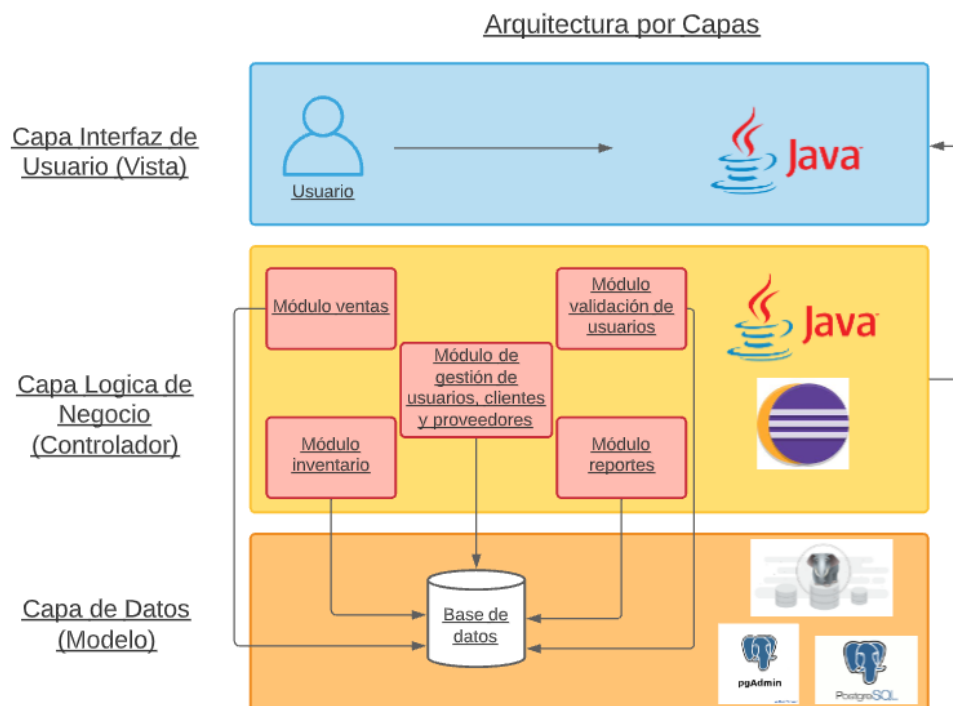


Figura 2: Diagrama de arquitectura por capas.

Diagrama de bloques: Presenta un diseño inicial de los módulos de la aplicación que representan el estilo arquitectural de la aplicación.

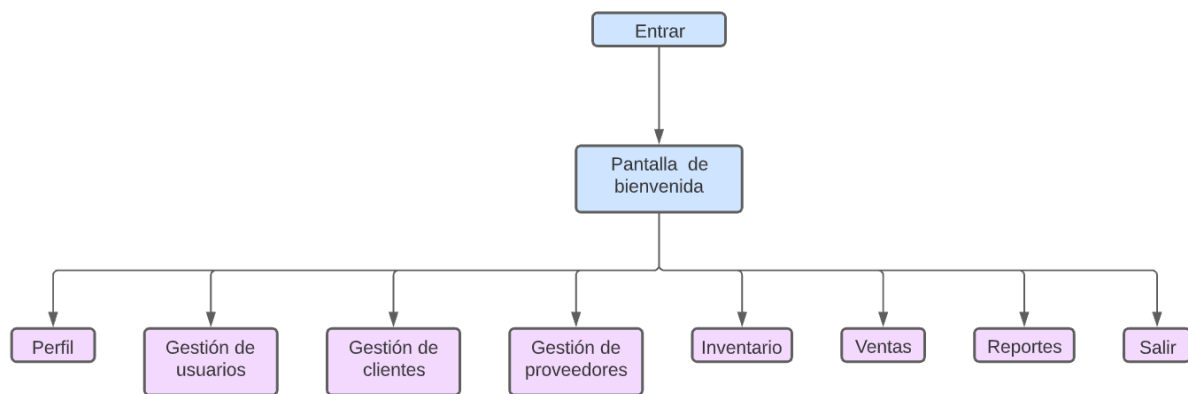


Figura 3: Mapa de navegación de la interfaz de usuario.

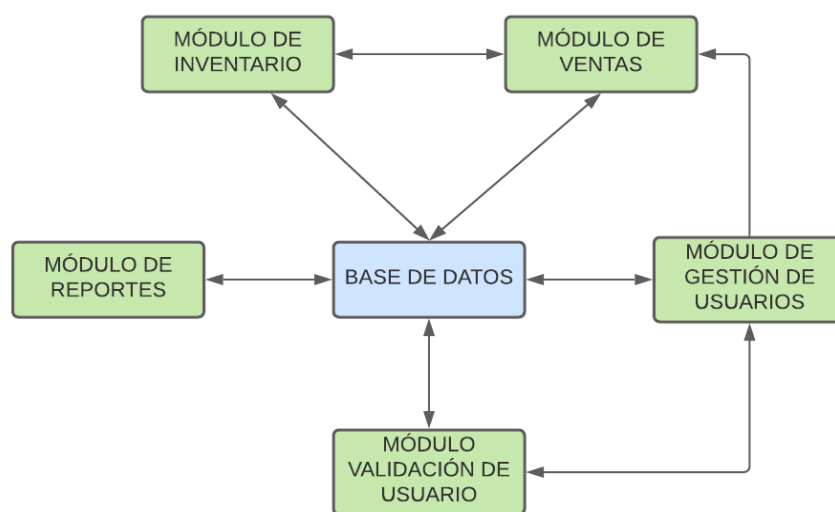


Figura 4: Diagrama de módulos de la aplicación.

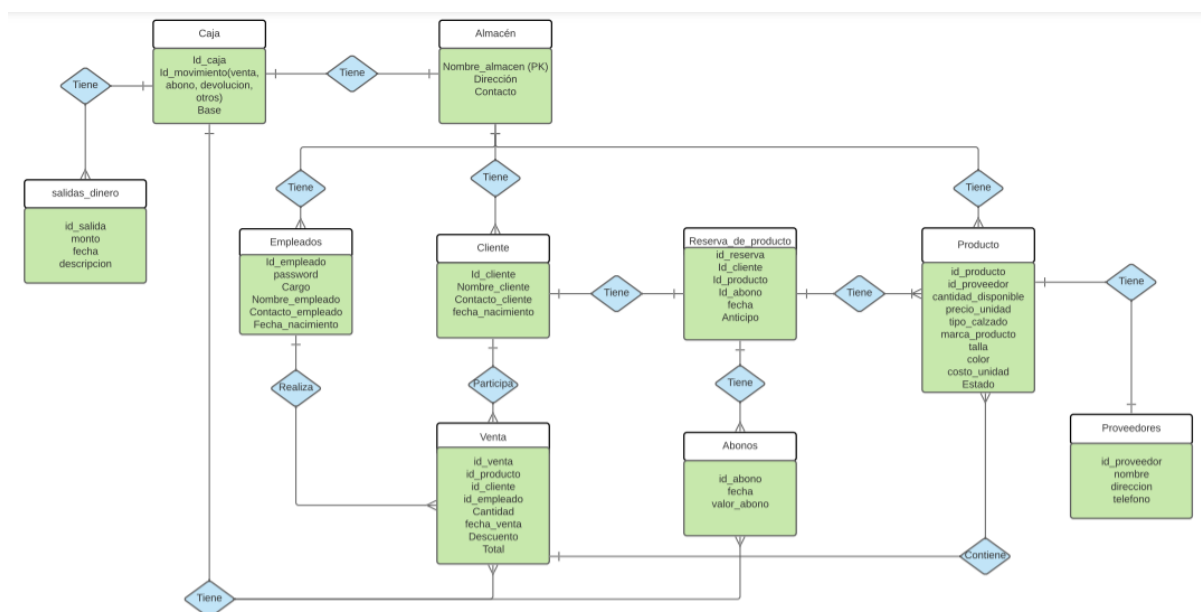


Figura 5: Diagrama de modelo relacional de datos.

[Para mejor calidad, click aquí.](#)

Patrones de construcción de software

En este apartado se explican los patrones de diseño de software aplicados en nuestro proyecto, cómo se usan, además se presenta su diagrama de clases y el código fuente de ejemplo.

Patrones de comportamiento

- **Observer:** Se necesita que los cambios que pueda tener un objeto puedan cambiar a otros objetos. Ej, GUI botones. El patrón Observer proporciona una forma de suscribirse y cancelar la subscripción a estos eventos para cualquier objeto que implemente una interfaz suscriptor. Un ejemplo del patrón en una biblioteca de Java que usaremos son todas las implementaciones de `java.util.EventListener` (prácticamente por todos los componentes Swing). Solución: Agregar un mecanismo para que los objetos interesados en conocer un estado de un objeto específico sean notificados.

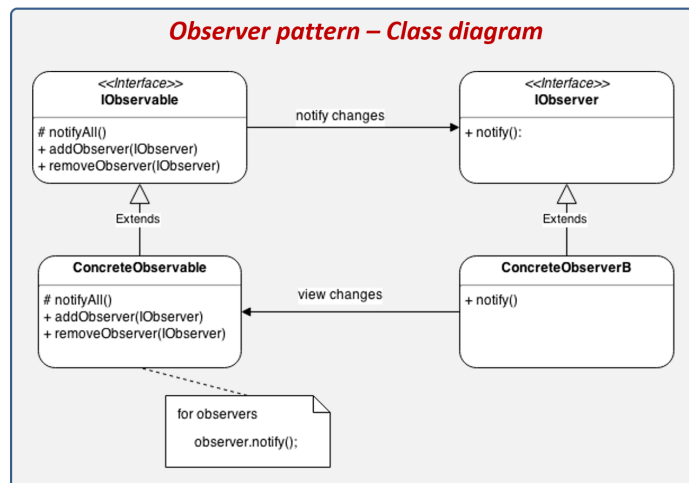


Figura 6: Diagrama de clases del patrón observador.

En nuestra aplicación se utiliza este patrón a la hora de realizar acciones con los botones. Un ejemplo particular es dentro de la función “actionPerformed” en la clase `ControladorGestionClientes`, pues se tiene una validación para el botón “modificar” en la ventana cliente, donde se notifica a 3 funciones más para que hagan algo como respuesta al utilizar el botón.

```
if(e.getSource() == ventanaGestionClientes.btnModificarCliente) {
    if(ventanaGestionClientes.validarCamposLlenos()) {
        ponerValoresEnModeloUsuario();
        consultaCliente.modificar(cliente);
        ventanaGestionClientes.limpiarCasillas();
    } else {
        JOptionPane.showMessageDialog(null, "Complete todos los campos con el formato correcto");
    }
}
```

Figura 7: Fragmento de código aplicando el patrón observador.

- **Command:** es un patrón de diseño de comportamiento que convierte una solicitud en un objeto independiente que contiene toda la información sobre la solicitud. El buen diseño de software casi siempre se basa en el principio de separación de responsabilidades, lo que suele tener como resultado la separación de la aplicación en capas, que es lo que nosotros aplicamos en nuestro proyecto. El ejemplo más habitual (y el que nosotros aplicamos con el modelo mvc), es tener una capa para la interfaz gráfica de usuario (GUI) y otra capa para la lógica de negocio.

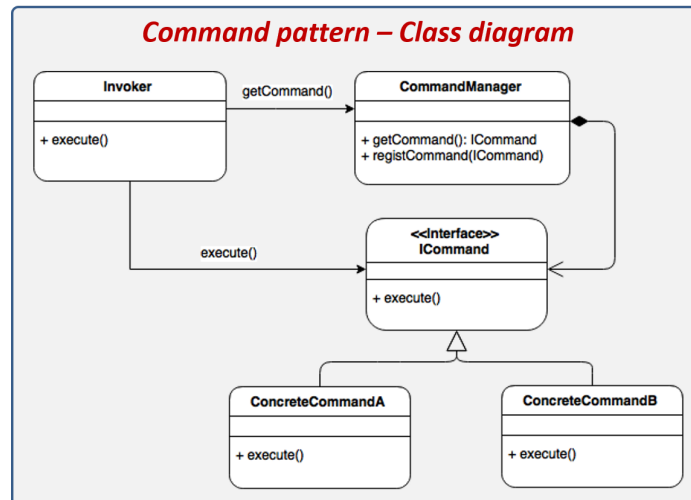


Figura 8: Diagrama de clases del patrón command.

Este patrón se implementa en nuestro código en varias clases. Un ejemplo particular es dentro de la función “actionPerformed” en la clase ControladorGestionClientes nuevamente, pues se tiene una validación para el botón “Registrar” en la ventana cliente, donde se hace la petición registrar a la clase “consultaCliente” que está en la lógica del programa (modelo).

```

if(e.getSource() == ventanaGestionClientes.btnRegistrarCliente) {
    if(ventanaGestionClientes.validarCamposLlenos()) {
        ponerValoresEnModeloUsuario();
        consultaCliente.registrar(cliente);
        ventanaGestionClientes.limpiarCasillas();

        ventanaGestionClientes.borrarElementosTabla();
        consultaCliente.poblarTabla(ventanaGestionClientes.table);
    }
} else {
    JOptionPane.showMessageDialog(null, "Complete todos los campos con el formato correcto");
}
  
```

Figura 9: Fragmento de código aplicando el patrón command.

Patrón creacional

- **Builder:** Problema: Se requiere definir varios "formatos" de venta, una venta puede ser con cliente registrado, sin cliente registrado, al contado, una separación de productos, entre otros. El patrón builder permite tener un "constructor abstracto", que recibe las órdenes básicas de cómo construir un objeto y además invoca unos constructores

específicos con unas instrucciones para cada variación del objeto, de manera que el constructor básico no es muy grande, ni se depende de una subclase para crear cada variación de objeto.

Solución: Crear constructores que contengan instrucciones específicas para cada parte de una posible venta.

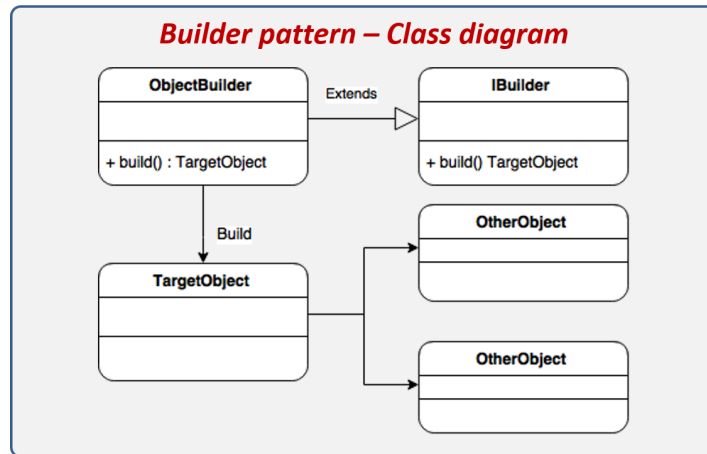


Figura 10: Diagrama de clases del patrón Builder.

Para mostrar el uso de este patrón en el proyecto se debería mostrar demasiado código, por lo tanto se opta por mostrar dos ventanas creadas a partir de un mismo modelo.

Figura 11: Interfaces creadas aplicando el patrón Builder.

Calidad de software

Aquí se describen los atributos y estándares de calidad aplicados en el proyecto.

Atributos de calidad

Usabilidad:

Este atributo de calidad evalúa que el producto de software sea de fácil entendimiento, aprendizaje y uso. El atributo tiene los siguientes criterios de calidad:

- Capacidad de aprendizaje: Capacidad del producto que permite al usuario aprender su aplicación.

- Capacidad de ser usado: Capacidad del producto que permite al usuario operarlo y controlarlo con facilidad.
- Protección contra errores del usuario: Capacidad del sistema para proteger a los usuarios “ de hacer errores” y de darle FeedBack sobre los mismos.
- Estética de la interfaz de usuario: Capacidad de la interfaz de usuario de agradar y satisfacer la interacción con el usuario.

Métricas para evaluar la usabilidad:

- Efectividad
- Eficiencia
- Satisfacción

Mantenibilidad:

Esta característica representa la capacidad del producto de software para ser modificado efectiva y eficientemente, debido a necesidades evolutivas, correctivas o perfectivas. El atributo tiene los siguientes criterios de calidad:

- Analizabilidad: Facilidad con la que se puede evaluar el impacto de un determinado cambio sobre el resto del software, diagnosticar las deficiencias o causas de fallos en el software o identificar las partes a modificar.
- Capacidad para ser modificado: Capacidad del producto que permite que sea modificado de forma efectiva y eficiente sin introducir defectos o degradar el desempeño.
- Capacidad para ser probado: Facilidad con la que se pueden establecer criterios de prueba para un sistema o componente y con la que se pueden llevar a cabo las pruebas para determinar si se cumplen dichos criterios.

Métricas para evaluar la Mantenibilidad:

- Código duplicado
- Número de errores de las pruebas unitarias

Estándares de calidad

Norma ISO/IEC 9126

La norma ISO/IEC 9126 de 1991, es la norma para evaluar los productos de software, esta norma nos indica las características de la calidad y los lineamientos para su uso, las características de calidad y sus métricas asociadas, pueden ser útiles tanto como para evaluar el producto como para definir los requerimientos de la calidad y otros usos. Esta norma está definida por un marco conceptual basado en los factores: Calidad del Proceso, Calidad del Producto del Software y Calidad en Uso. Según el marco conceptual, la calidad del producto, contribuye a su vez a mejorar la calidad en uso.

La norma ISO/IEC 9126 define la calidad en uso como la perspectiva del usuario de la calidad del producto software cuando éste es usado en un ambiente específico y un contexto de uso específico. Éste mide la extensión para la cual los usuarios pueden

conseguir sus metas en un ambiente particular, en vez de medir las propiedades del software en sí mismo. El modelo de la calidad en uso muestra un conjunto de 4 características: efectividad, productividad, integridad, y satisfacción.

Estándar ISO/IEC 14598

El estándar ISO/IEC 14598 es actualmente usado como base metodológica para la evaluación del producto software. En sus diferentes etapas, establece un marco de trabajo para evaluar la calidad de los productos de software proporcionando, además, métricas y requisitos para los procesos de evaluación de los mismos. La norma define las principales características del proceso de evaluación: Repetitividad. Reproducibilidad. Imparcialidad. Objetividad. Para estas características se describen las medidas concretas que participan: Análisis de los requisitos de evaluación. Evaluación de las especificaciones. Evaluación del diseño y definición del plan de evaluación. Ejecución del plan de evaluación. Evaluación de la conclusión.

Diseño de casos de prueba

Pruebas unitarias

Método lógico seleccionado: *Modificar*. Este método permite modificar la información de un cliente en la base de datos.

```
public boolean modificar(Cliente cliente){  
  
    int id = cliente.getId();  
    String nombre = cliente.getNombre();  
    String telefono = cliente.getTelefono();  
    Date fechaNacimiento = cliente.getFechaNacimiento();  
  
    Connection con = getConnection();  
  
    String sql = "UPDATE cliente SET cedula_cliente='"+id+"', nombre_cliente='"+nombre+"', contacto_cliente='"+  
        +telefono+"', fecha_nacimiento_cliente='"+fechaNacimiento+"' WHERE cedula_cliente='"+id+"'";  
  
    try{  
  
        Statement st = con.createStatement();  
        int ejecucion = st.executeUpdate(sql);  
  
        if(ejecucion != 0) {  
            JOptionPane.showMessageDialog(null, "Cambio exitoso");  
            return true;  
        }else {  
            JOptionPane.showMessageDialog(null, "Cambio fallido");  
            return false;  
        }  
  
    }catch(SQLException e) {  
        System.err.println(e);  
        return false;  
    } finally{  
        try{  
            con.close();  
        }catch (SQLException e){  
            System.err.println(e);  
        }  
    }  
}
```

Figura 12: Método lógico.

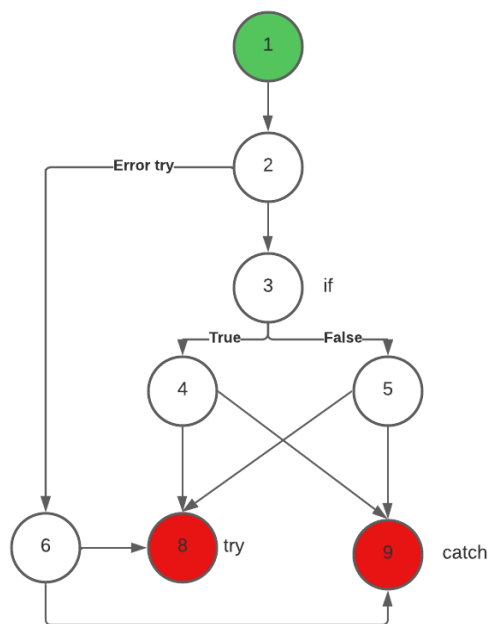


Figura 13: Grafo de flujo.

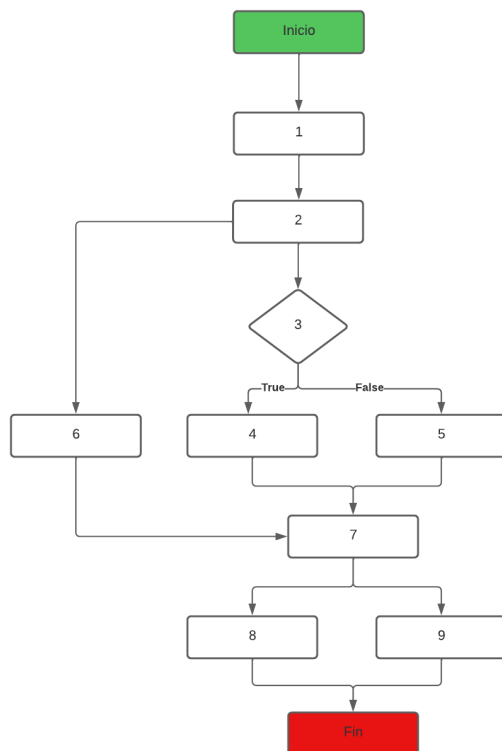


Figura 14: Diagrama de flujo.

Complejidad ciclomática:

$$V(G) = \text{Aristas} - \text{Nodos} + 2$$

$$V(G) = 11 - 9 + 2 = 4$$

o

$$V(G) = \text{Número de nodos predicado} + 1$$

$$V(G) = 3 + 1 = 4$$

Casos de prueba

Número	Caso de prueba	Camino independiente	Parámetros de entrada	Resultado
1	El cliente que se quiere modificar no se encuentra en la base de datos.	I-1-2-6-7-8-F	Cliente(104901939, "Miguel", "3112888308", 1999/03/03).	SQL Server Exception. Valor retornado: Falso
2	El cliente está en la base de datos e intentamos	I-2-3-5-7-8-F	Cliente(3216, "Ricardo Ocampo", "3254781236",	Mensaje "Cambio Fallido". Valor retornado:

	hacer un cambio pero en realidad no cambiamos nada.		1982/01/24)	False
3	El cliente se encuentra en la base de datos y sí realizamos un cambio.	I-2-3-4-7-8-F	Cliente(3216, "Ricardo Ocampo", "30023454", 1982/01/24)	Mensaje "Cambio exitoso". Valor retornado: True
4	Hay un error al intentar cerrar la conexión con la base de datos. No se ejecuta porque depende de factores externos.	I-2-3-4-7-9-F	Cliente(3216, "Ricardo Ocampo", "30023454", 1982/01/24)	IO Exception. Mensaje "Cambio exitoso". Valor retornado: True

Tabla 1: Casos de prueba unitarios.

Pruebas funcionales

Para las pruebas funcionales se utiliza el método de **partición de equivalencia**. Los requerimientos funcionales seleccionados son:

1. Como usuario requiero poder iniciar y cerrar sesión en el sistema para acceder a sus herramientas.

Se prueba el ingreso a la aplicación o login, por lo tanto se toma como rango el usuario y la contraseña.

Usuario:

- Rango de valores: número de máximo 15 cifras.
- Variable de la entrada: usuario, cada dígito está dado por $(0 < x < 9)$.
- Clases:
 - Válida: $0 < x < 999999999999999$ → Por ejemplo 123456
 - Inválida: $x > 999999999999999$ → Por ejemplo 9999999999999921

Contraseña:

- Rango de valores: valor alfanumérico de máximo 10 caracteres.
- Variable de la entrada: contraseña, cada carácter puede ser entre 0 y 9 o una letra cualquiera del abecedario.
- Clases:
 - Válida: abcde45
 - Inválida: 12345hola6789

[Enlace prueba Login](#)

2. Como vendedor/administrador requiero poder registrar un cliente en el sistema para guardar información de los clientes en la base de datos.

Se prueba el registro de clientes en el sistema, por lo tanto el rango está conformado por los campos que conforman un cliente en la base de datos.

Id:

- Rango de valores: número de máximo 15 cifras.
- Variable de la entrada: id, cada dígito está dado por $(0 < x < 9)$.
- Clases:
 - Válida: $0 < x < 999999999999999$ → Por ejemplo 1007681934
 - Inválida: $x > 999999999999999$ → Por ejemplo 19999999999999921

Nombre:

- Rango de valores: cadena de caracteres de longitud máxima 40.
- Variable de la entrada: nombre, cada caracter puede ser una letra cualquiera del abecedario o un espacio en blanco. El campo no puede quedar vacío.
- Clases:
 - Válida: Camilo Rodríguez
 - Inválida: " " (Campo vacío)
Camilo49

Teléfono:

- Rango de valores: número con máximo de 10 cifras.
- Variable de la entrada: teléfono (celular en Colombia), cada dígito está dado por $(0 < x < 9)$.
- Clases:
 - Válida: 3118597090
 - Inválida: 311859709023548

Para la fecha de nacimiento no es necesario hacer pruebas ya que se selecciona de un campo desplegable que contiene todos los posibles valores perteneciente a la clase válida. Por lo tanto es imposible insertar un dato inválido.

[Enlace prueba registro cliente](#)

Pruebas no funcionales

Para esta parte se realizará una prueba de rendimiento, enfocada al tiempo de respuesta de la aplicación cuando se pasa entre los diferentes menús de los módulos y el login.

Entorno de pruebas:

- Sistema operativo: Windows 10
- Ambiente de prueba: Local
- Base de datos: En la nube con Elephant.
- Herramientas: Método del sistema Java. `currentTimeMillis()`.

Criterios de aceptación:

El tiempo de respuesta aceptable para nuestro requerimiento no funcional será de 2 segundos.

[Diseño de las pruebas y resultados.](#)

Ejecución de casos de prueba

Pruebas unitarias

Herramienta seleccionada: se trabaja con JUnit test. Según Wikipedia, JUnit es un conjunto de clases (framework) que permite realizar la ejecución de clases Java de manera controlada, para poder evaluar si el funcionamiento de cada uno de los métodos de la clase se comporta como se espera. Es decir, en función de algún valor de entrada se evalúa el valor de retorno esperado; si la clase cumple con la especificación, entonces JUnit devolverá que el método de la clase pasó exitosamente la prueba; en caso de que el valor esperado sea diferente al que regresó el método durante la ejecución, JUnit devolverá un fallo en el método correspondiente.

Definición de los diferentes escenarios:

```
public class Tests extends TestCase{

    private ConsultaCliente consulta;
    private Cliente cliente;

    public void escenarioUno() {
        consulta = new ConsultaCliente();
        @SuppressWarnings("deprecation")
        Date fecha = new Date(99, 2, 3);
        cliente = new Cliente();
        cliente.setNombre("Miguel");
        cliente.setId(104901939);
        cliente.setFechaNacimiento(fecha);
        cliente.setTelefono("3112888308");
    }

    public void escenarioDos() {
        consulta = new ConsultaCliente();
        @SuppressWarnings("deprecation")
        Date fecha = new Date(82, 01, 24);
        cliente = new Cliente();
        cliente.setNombre("Ricardo Ocampo");
        cliente.setId(3216);
        cliente.setFechaNacimiento(fecha);
        cliente.setTelefono("3254781236");
    }

    public void escenarioTres() {
        consulta = new ConsultaCliente();
        @SuppressWarnings("deprecation")
        Date fecha = new Date(82, 01, 24);
        cliente = new Cliente();
        cliente.setNombre("Ricardo Ocampo");
        cliente.setId(3216);
        cliente.setFechaNacimiento(fecha);
        cliente.setTelefono("30023454");
    }
}
```

Figura 15: Construcción de escenarios para pruebas unitarias en java.

Métodos que ejecutan las pruebas.

```
//caso de prueba en el que el cliente que queremos modificar no se encuentra en la base de datos
//esperamos que no se logre modificar los datos del cliente
public void testCasoUno(){
    escenarioUno();
    assertTrue(consulta.modificar(cliente)==false);
}

//caso de prueba en el que el cliente que queremos modificar se encuentra en la base de datos pero
//por algún error le pasamos los mismos datos que ya tenía. O sea, no hay cambios.
//esperamos que no se logre modificar los datos del cliente
public void testCasoDos(){
    escenarioDos();
    assertTrue(consulta.modificar(cliente)==false);
}

//caso de prueba en el que el cliente que queremos modificar se encuentra en la base de datos y
//hacemos algún cambio en los datos.
//esperamos que se logre modificar los datos del cliente
public void testCasoTres(){
    escenarioTres();
    assertTrue(consulta.modificar(cliente)==true);
}
```

Figura 16: Métodos de ejecución de pruebas unitarias.

Resultados

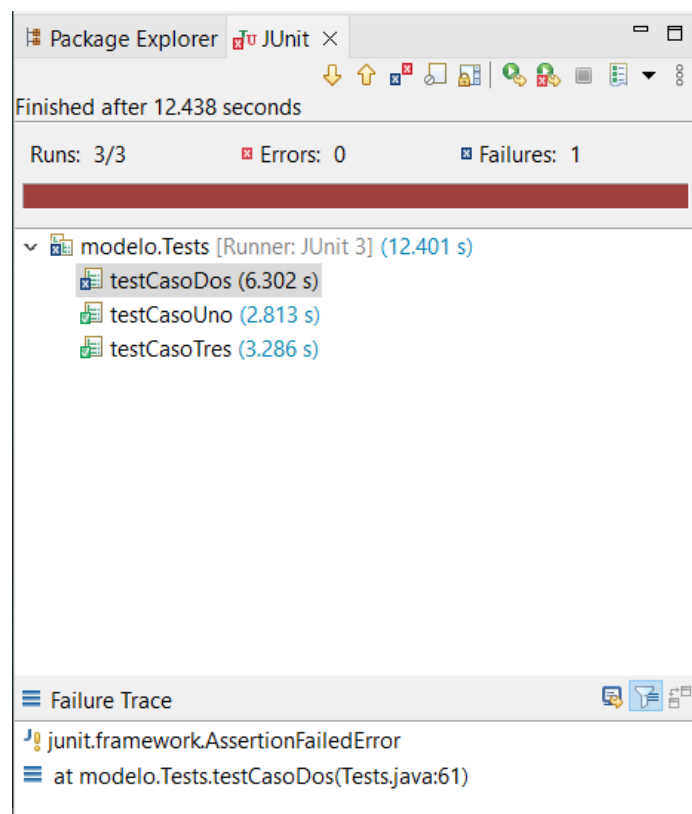


Figura 17: Resultados de las pruebas unitarias, JUnit.

Pruebas funcionales

Estas pruebas se realizaron manualmente, por lo tanto a continuación se muestran capturas de pantalla con los resultados obtenidos.

Primero se llevan a cabo las pruebas en el Login. Para los campos de usuario y contraseña se tienen distintas restricciones y por tanto se deben mostrar mensajes acordes para que el usuario tenga una ayuda a la hora de ingresar en caso de que haya alguna confusión o equivocación.

Pruebas de usuario:

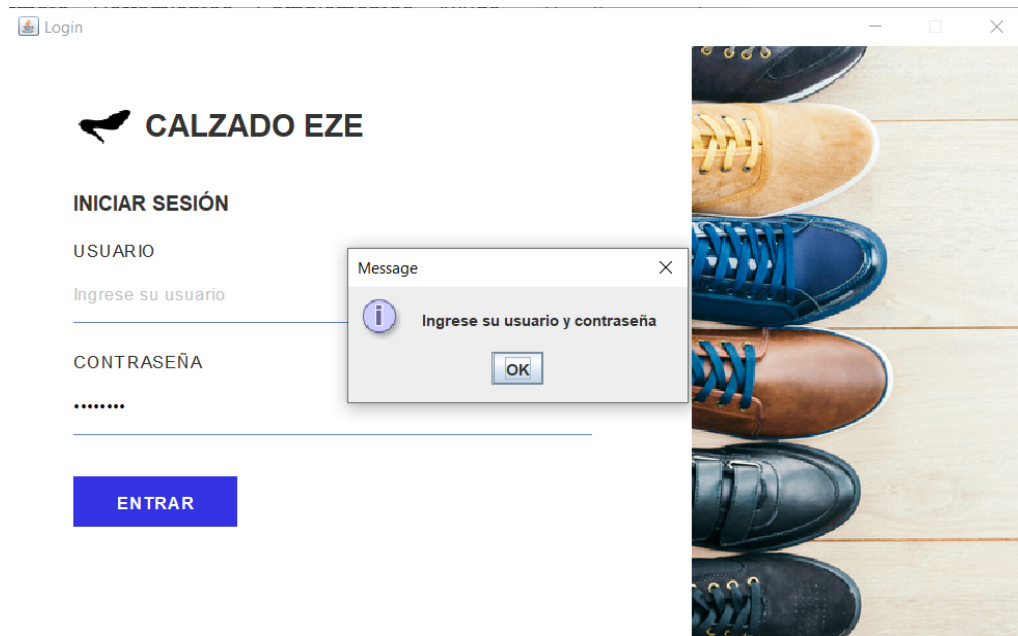


Figura 18: Prueba de validación para el campo usuario (Campo vacío).

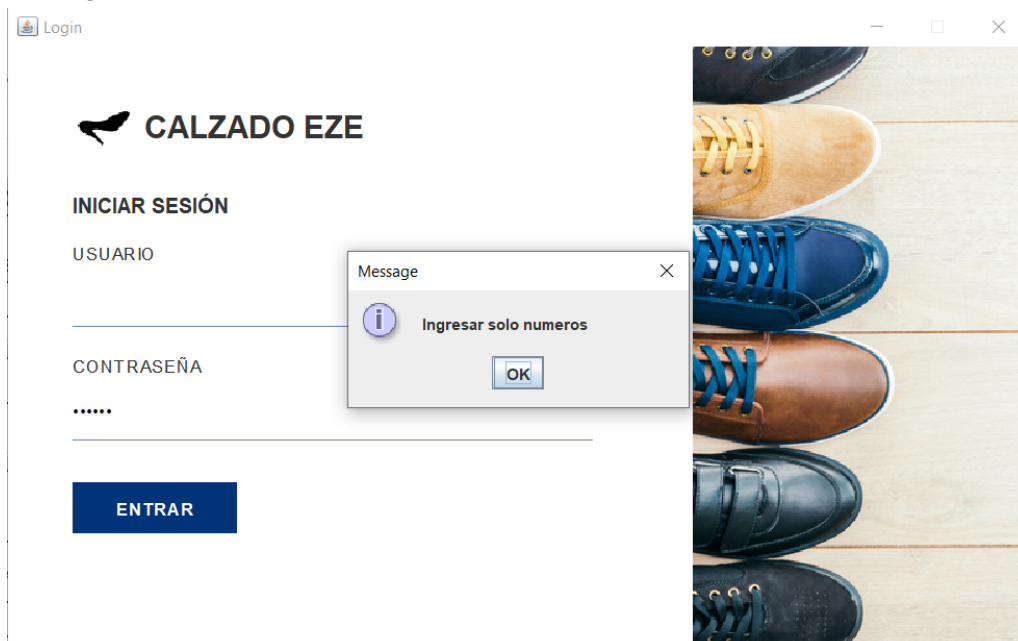


Figura 19: Prueba de validación para el campo usuario (Ingreso de letras).

Pruebas de contraseña:

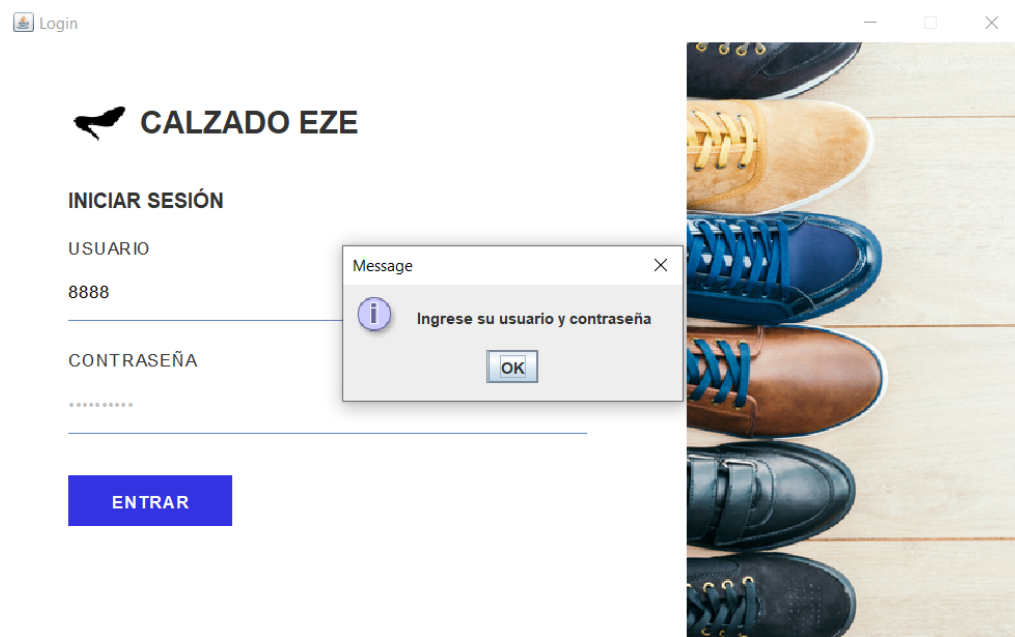


Figura 20: Prueba de validación para el campo contraseña (Campo vacío).

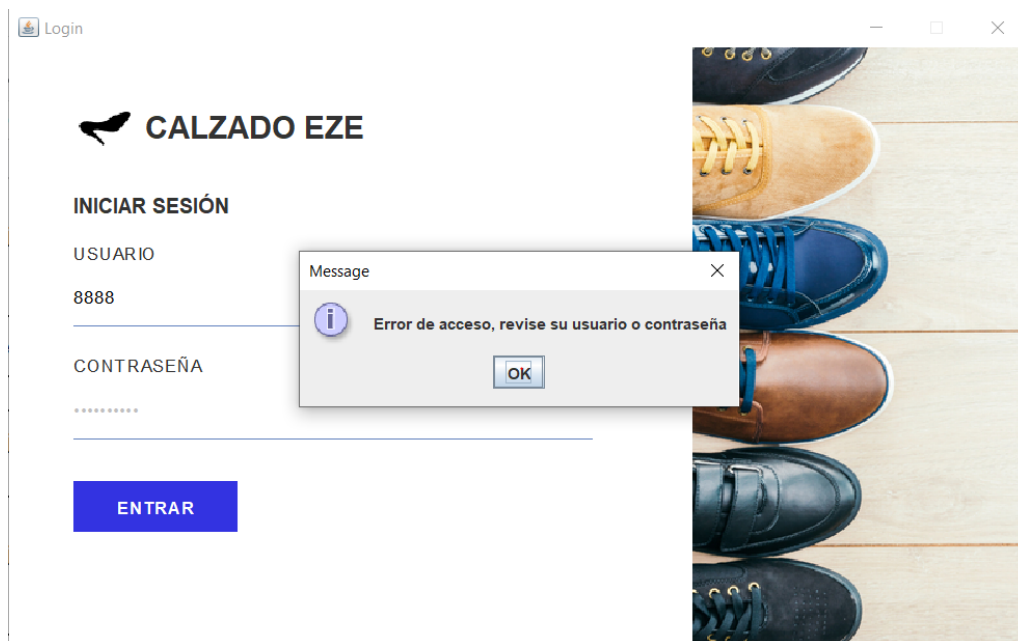


Figura 21: Fallo en prueba de validación para el campo contraseña (Campo vacío).

Pruebas del registro de cliente: Primero se prueba el caso donde se deja vacío alguno de los campos del cliente.

The screenshot shows the 'Gestion de Clientes' form in a web application. The form has fields for 'Id', 'Nombre', 'Telefono', and 'Fecha de Nacimiento (YYYY-MM-DD)'. The 'Id' field contains '123456789', 'Telefono' contains '3008481950', and 'Fecha de Nacimiento' is set to '1990-07-02'. The 'Nombre' field is empty. A modal message box is displayed over the form, stating 'Complete todos los campos con el formato correcto' (Complete all fields with the correct format). Below the form, there is a table of existing clients.

ID	Nombre	Telefono	FechaNacimiento
3578	Julio Gutierrez	3452136568	1994-11-03
2245	Roberto Mata	3053430542	1980-05-28
5461	Marcela Galindo	7714876365	1991-10-30
4566	Salome Valencia	4162336677	2004-03-24
123456	Ricardo Cruz	3216549871	1990-01-01
3216	Ricardo Ocampo	30023454	1982-02-24

Figura 22: Prueba de validación para el campo nombre (Campo vacío).

The screenshot shows the 'Gestion de Clientes' form in a web application. The form has fields for 'Id', 'Nombre', 'Telefono', and 'Fecha de Nacimiento (YYYY-MM-DD)'. The 'Id' field contains '123456789', 'Telefono' contains '3008481950', and 'Fecha de Nacimiento' is empty. A modal message box is displayed over the form, stating 'Complete todos los campos con el formato correcto' (Complete all fields with the correct format). Below the form, there is a table of existing clients.

ID	Nombre	Telefono	FechaNacimiento
3578	Julio Gutierrez	3452136568	1994-11-03
2245	Roberto Mata	3053430542	1980-05-28
5461	Marcela Galindo	7714876365	1991-10-30
4566	Salome Valencia	4162336677	2004-03-24
123456	Ricardo Cruz	3216549871	1990-01-01
3216	Ricardo Ocampo	30023454	1982-02-24

Figura 23: Prueba de validación para el campo fecha (Campo vacío).

Menu Principal

Calzado ...

Perfil

Gestión de Usuarios

Gestión de Clientes

Gestión de Proveedores

Inventario

Ventas

Reportes

Salir

CIVU

Usuario: Luis Ruiz

Hoy es Marzo 4 del año 2022

Gestion de Clientes

Id

Nombre

Telefono

Fecha de Nacimiento (YYYY-MM-DD)

1990 - 07 - 02

REGISTRAR MODIFICAR ELIMINAR

Message

Complete todos los campos con el formato correcto

OK

BUSCAR

ID	Nombre	Telefono	FechaNacimiento
3578	Julio Gutierrez	3452136568	1994-11-03
2245	Roberto Mata	3053430542	1980-05-28
5461	Marcela Galindo	7714876365	1991-10-30
4566	Salome Valencia	4162336677	2004-03-24
123456	Ricardo Cruz	3216549871	1990-01-01
3216	Ricardo Ocampo	30023454	1982-02-24

SELECCIONAR LISTA DE CLIENTES

Figura 24: Prueba de validación para el campo id (Campo vacío).

Menu Principal

Calzado ...

Perfil

Gestión de Usuarios

Gestión de Clientes

Gestión de Proveedores

Inventario

Ventas

Reportes

Salir

CIVU

Usuario: Luis Ruiz

Hoy es Marzo 4 del año 2022

Gestion de Clientes

Id

Nombre

Telefono

Fecha de Nacimiento

1990 - - -

REGISTRAR MODIFICAR ELIMINAR LIMPIAR

Message

Complete todos los campos con el formato correcto

OK

BUSCAR

ID	Nombre	Telefono	FechaNacimiento
3578	Julio Gutierrez	3452136568	1994-11-03
2245	Roberto Mata	3053430542	1980-05-28
5461	Marcela Galindo	7714876365	1991-10-30
4566	Salome Valencia	4162336677	2004-03-24
123456	Ricardo Cruz	3216549871	1990-01-01
3216	Ricardo Ocampo	30023454	1982-02-24
123456789	Camilo Rodríguez	3008481950	1990-07-02

SELECCIONAR LISTA DE CLIENTES

Figura 25: Prueba de validación para el campo teléfono (Campo vacío).

Luego se prueban los casos inválidos de cada campo:

The screenshot shows the 'Gestion de Clientes' form in the CIVU application. The 'Id' field contains '123456789L'. A modal message box is displayed with the text 'Ingresar solo numeros' (Enter only numbers). The form includes fields for 'Id', 'Nombre', 'Telefono', and 'Fecha de Nacimiento'. Below the form is a table of existing clients.

ID	Nombre	Telefono	FechaNacimiento
3578	Julio Gutierrez	3452136568	1994-11-03
2245	Roberto Mata	3053430542	1980-05-28
5461	Marcela Galindo	7714876365	1991-10-30
4568	Salome Valencia	4162336677	2004-03-24
123456	Ricardo Cruz	3216549871	1990-01-01
3216	Ricardo Ocampo	30023454	1982-02-24

Figura 26: Prueba de validación para el campo id (Ingreso de letras)

The screenshot shows the 'Gestion de Clientes' form in the CIVU application. The 'Nombre' field contains 'Camilo49 Rodriguez'. A modal message box is displayed with the text 'Ingresar solo letras' (Enter only letters). The form includes fields for 'Id', 'Nombre', 'Telefono', and 'Fecha de Nacimiento'. Below the form is a table of existing clients.

ID	Nombre	Telefono	FechaNacimiento
3578	Julio Gutierrez	3452136568	1994-11-03
2245	Roberto Mata	3053430542	1980-05-28
5461	Marcela Galindo	7714876365	1991-10-30
4568	Salome Valencia	4162336677	2004-03-24
123456	Ricardo Cruz	3216549871	1990-01-01
3216	Ricardo Ocampo	30023454	1982-02-24

Figura 27: Prueba de validación para el campo nombre (Ingreso de números).

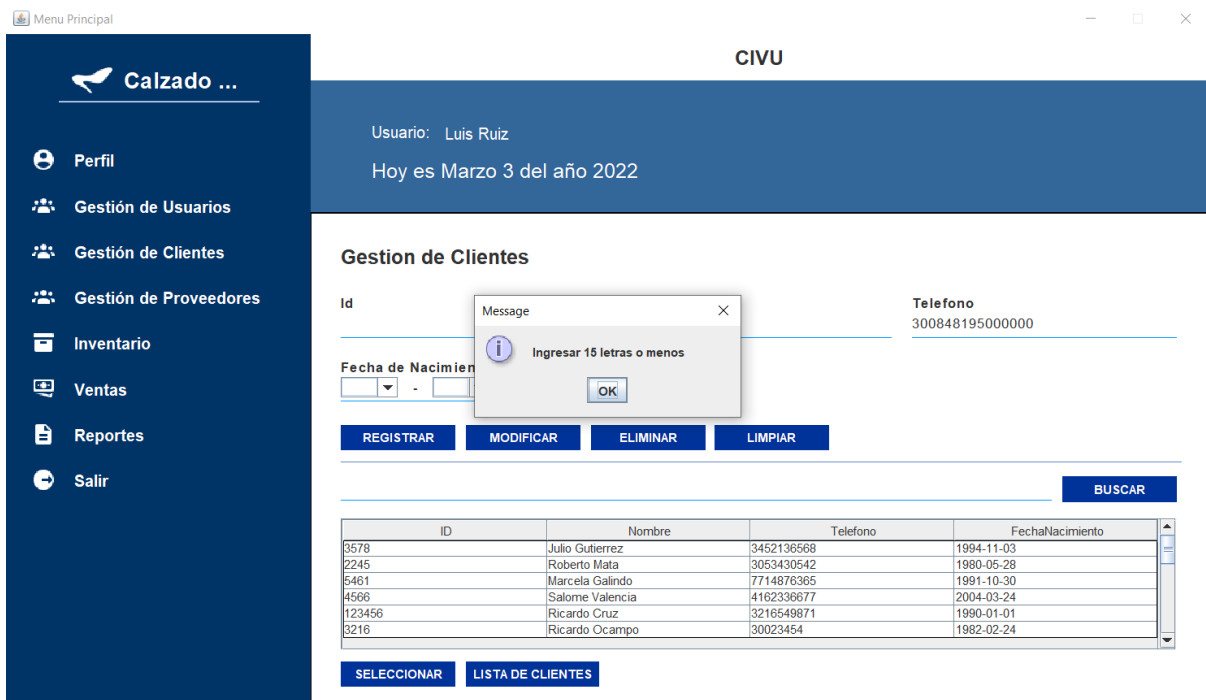


Figura 28: Fallo en prueba de validación para el campo teléfono (Ingreso de números extra).

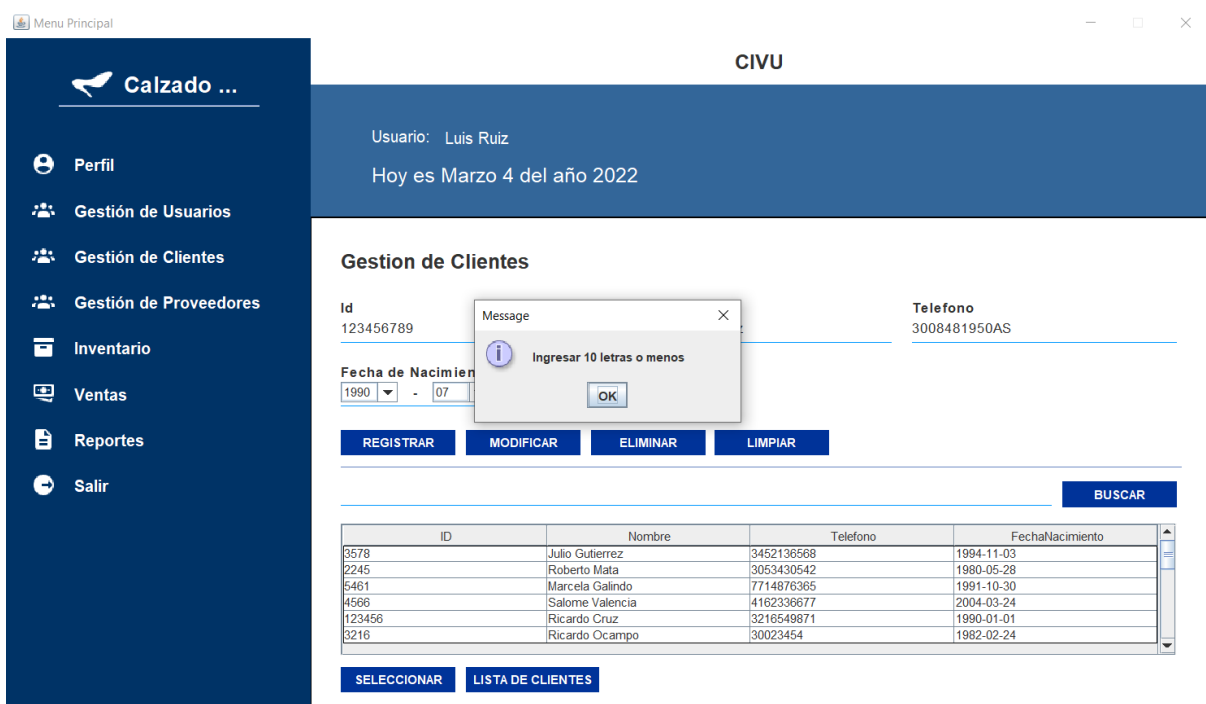


Figura 29: Fallo en prueba de validación para el campo teléfono (Ingreso de letras).

Finalmente se prueba aplicando los casos válidos en todos los campos:

The screenshot shows the 'Gestion de Clientes' interface. A modal message box titled 'Message' is displayed in the center, containing an information icon, the text 'Registro exitoso', and an 'OK' button. The background interface includes a sidebar with navigation options like 'Perfil', 'Gestión de Usuarios', 'Gestión de Clientes', 'Gestión de Proveedores', 'Inventario', 'Ventas', 'Reportes', and 'Salir'. The main content area has a header with 'CIVU', user information 'Usuario: Luis Ruiz', and the date 'Hoy es Marzo 4 del año 2022'. Below this, the 'Gestion de Clientes' section contains input fields for 'Id' (123456789), 'Nombre' (Camilo Rodríguez), 'Telefono' (3008481950), and 'Fecha de Nacimiento' (1990-07-02). There are buttons for 'REGISTRAR', 'MODIFICAR', 'ELIMINAR', and 'LIMPIAR'. A 'BUSCAR' button is also present. At the bottom, there is a table with columns 'ID', 'Nombre', 'Telefono', and 'FechaNacimiento', listing several clients. Below the table are buttons for 'SELECCIONAR' and 'LISTA DE CLIENTES'.

Figura 30: Prueba de validación para casos válidos en todos los campos.

This screenshot shows the same 'Gestion de Clientes' interface as Figure 30, but after the successful registration. The modal message box is no longer present. The input fields for 'Id', 'Nombre', 'Telefono', and 'Fecha de Nacimiento' are now empty. The 'REGISTRAR' button is still visible. The table at the bottom now includes the newly registered client '123456789 Camilo Rodríguez 3008481950 1990-07-02' in its list. The rest of the interface, including the sidebar and header, remains the same.

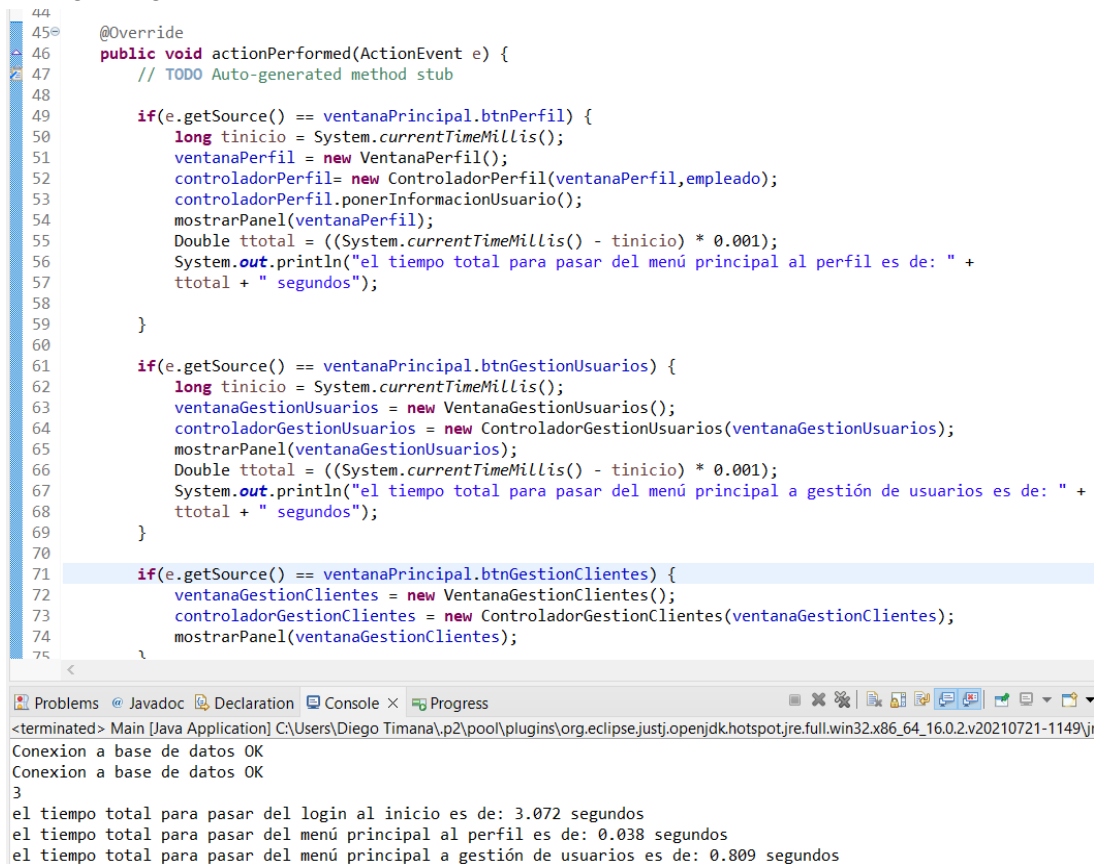
Figura 31: Registro exitoso con todos los campos validados.

Pruebas no funcionales

Para estas pruebas de rendimiento se utiliza un método del sistema, para obtener la hora actual del sistema en milisegundos , el método es `System.currentTimeMillis()`. Obteniendo así un tiempo inicial. Luego, para obtener el tiempo total de ejecución, llamamos al método

de nuevo, luego de que se ejecute lo que queremos, y le restamos el tiempo inicial. Finalmente hacemos la conversión de milisegundos a segundos.

En la imagen siguiente se muestra lo que se hizo:



```
44  
45  
46 @Override  
47 public void actionPerformed(ActionEvent e) {  
48     // TODO Auto-generated method stub  
49  
50     if(e.getSource() == ventanaPrincipal.btnPerfil) {  
51         long tinicio = System.currentTimeMillis();  
52         ventanaPerfil = new VentanaPerfil();  
53         controladorPerfil = new ControladorPerfil(ventanaPerfil, empleado);  
54         controladorPerfil.ponerInformacionUsuario();  
55         mostrarPanel(ventanaPerfil);  
56         Double tttotal = ((System.currentTimeMillis() - tinicio) * 0.001);  
57         System.out.println("el tiempo total para pasar del menú principal al perfil es de: " +  
58             tttotal + " segundos");  
59     }  
60  
61     if(e.getSource() == ventanaPrincipal.btnGestionUsuarios) {  
62         long tinicio = System.currentTimeMillis();  
63         ventanaGestionUsuarios = new VentanaGestionUsuarios();  
64         controladorGestionUsuarios = new ControladorGestionUsuarios(ventanaGestionUsuarios);  
65         mostrarPanel(ventanaGestionUsuarios);  
66         Double tttotal = ((System.currentTimeMillis() - tinicio) * 0.001);  
67         System.out.println("el tiempo total para pasar del menú principal a gestión de usuarios es de: " +  
68             tttotal + " segundos");  
69     }  
70  
71     if(e.getSource() == ventanaPrincipal.btnGestionClientes) {  
72         ventanaGestionClientes = new VentanaGestionClientes();  
73         controladorGestionClientes = new ControladorGestionClientes(ventanaGestionClientes);  
74         mostrarPanel(ventanaGestionClientes);  
75     }  
76 }
```

Problems | Javadoc | Declaration | Console | Progress

<terminated> Main [Java Application] C:\Users\Diego Timana\p2\pool\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64_16.0.2.v20210721-1149\j

Conexion a base de datos OK
Conexion a base de datos OK
3
el tiempo total para pasar del login al inicio es de: 3.072 segundos
el tiempo total para pasar del menú principal al perfil es de: 0.038 segundos
el tiempo total para pasar del menú principal a gestión de usuarios es de: 0.809 segundos

Figura 32: Implementación y pruebas no funcionales en Java.

Diseño de las pruebas y resultados.

Reporte de ejecución de pruebas

En cuanto a las pruebas unitarias, vemos que hay una fallida. Investigamos al respecto pero lo que encontramos es que los formatos de la base de datos sql y la lógica del programa en java difieren, razón por la cual la base de datos registra los mismos datos 2 veces como máximo pues el campo fecha lo toma como diferente. Consideramos que este problema puede solucionarse gracias a una validación por el campo de ID, pues este número sería un número de identidad único como la cédula. Es un problema que se corregirá en el sprint final. El resto de pruebas unitarias fueron ejecutadas con éxito.

Pasando ahora a las pruebas funcionales, se aplicaron pruebas a 2 requerimientos funcionales, uno relacionado al ingreso a la aplicación y el otro relacionado al registro de clientes a la aplicación. En el primer requerimiento se hicieron validaciones para los campos de usuario y contraseña mediante el método partición de equivalencias. Se aplicaron 6 validaciones de las cuales fallaron 2, consideradas fallas por mostrar mensajes diferentes de los esperados, y evidenciando fallas leves en la lógica del programa. Las dos fallas

fueron corregidas mediante un cambio en la función validarUsuario de la clase ControladorLogin. Por otra parte, en el segundo requerimiento puesto a prueba se aplicaron 11 validaciones para los campos de Id, nombre, teléfono y fecha. De estas validaciones falló una sola, la cual estaba relacionada con la validación del teléfono. Se consideró error porque mostró un mensaje no esperado 2 veces, la segunda ocasión fue debida a que se corrigió parcialmente el error inicial en la lógica del programa. La corrección se hizo en la función VentajaGestionClientes perteneciente a la clase llamada del mismo modo. El resto de validaciones fueron ejecutadas con éxito y una vez corregidos los errores, las pruebas funcionales fueron ejecutadas con éxito en su totalidad.

Para finalizar se hizo una prueba a un requerimiento no funcional, relacionado con los tiempos de respuesta de la aplicación. Se probó el tiempo de respuesta al hacer ingreso en la aplicación y al pasar entre el menú de bienvenida y dos de las herramientas del sistema. La prueba se realizó en 2 computadores diferentes de los miembros del equipo de desarrollo, arrojando resultados diferentes. En el primer caso se presentó una única falla entre las 3 pruebas hechas, lo que se atribuyó a la propia aplicación debido a la cantidad de elementos gráficos que se deben cargar. En el segundo caso fallaron 2 de las 3 pruebas; el primer fallo es el mismo que ocurrió en el otro computador, pues tardó más tiempo del esperado en cargar. El segundo fallo ocurrido se atribuyó también a la cantidad de elementos gráficos y además a la cantidad de procesos actualmente activos en el equipo donde se llevó a cabo la prueba.

Para concluir, se tiene en cuenta que los fallos encontrados en el sistema son pocos y realmente no afectan al funcionamiento del sistema. Ciertos fallos se pudieron arreglar pues estaban relacionados a la lógica del programa, pero otros se deben a cuestiones de optimización, objetivo que no es perseguido en el curso actual, y que no se corrige por cuestiones de tiempo y dado el poco impacto en el funcionamiento de la aplicación. Será un próximo paso a mejorar en el desarrollo futuro de la aplicación.

Resumen de pruebas y análisis

[Reporte de pruebas.](#)

Referencias

- <https://refactoring.guru/es/design-patterns/command>
- <https://refactoring.guru/es/design-patterns/observer>
- <https://refactoring.guru/es/design-patterns/builder>
- https://platzi.com/tutoriales/1248-pro-arquitectura/5498-atributos-de-calidad-de-un-producto-de-software/?utm_source=google&utm_medium=cpc&utm_campaign=12915366154&utm_adgroup=&utm_content=&gclid=CjwKCAiAx8KQBhAGEiwAD3EiPxOqMDIh4les-hhLaSy9rpovUgv1MGJBwGUNPHlr-pTnPdUK4KXVJhoCEXAQAvD_BwE&gclsrc=aw.ds
- http://fcaenlinea.unam.mx/anexos/1728/Unidad_2/u2_act2_1.pdf
- https://www.scielo.cl/scielo.php?script=sci_arttext&pid=S0718-33052020000400654&lng=es&nrm=iso
- Presentaciones de la profesora sobre requerimientos, arquitectura de software, patrones de diseño, pruebas funcionales y no funcionales, entre otras.