

Universidad ORT Uruguay

Facultad de Ingeniería

Ingeniería de Software 1

Letra del Obligatorio 2

Alejandro Adorjan

Gerardo Quintana

Pablo Lucini

Martin Solari

Entregado como requisito de la materia Ingeniería de
Software 1

25 de noviembre de 2019

Declaraciones de autoría

Nosotros, Joaquin Lamela y Martín Gutman , declaramos que el trabajo que se presenta en esa obra es de nuestra propia mano. Podemos asegurar que:

- La obra fue producida en su totalidad mientras realizábamos Ingeniería de Software I
- Cuando hemos consultado el trabajo publicado por otros, lo hemos atribuido con claridad;
- Cuando hemos citado obras de otros, hemos indicado las fuentes. Con excepción de estas citas, la obra es enteramente nuestra;
- En la obra, hemos acusado recibo de las ayudas recibidas;
- Cuando la obra se basa en trabajo realizado conjuntamente con otros, hemos explicado claramente qué fue contribuido por otros, y qué fue contribuido por nosotros;
- Ninguna parte de este trabajo ha sido publicada previamente a su entrega, excepto donde se han realizado las aclaraciones correspondientes.

Resumen

En la actualidad, en el mundo en general se están produciendo grandes cambios ambientales. Esto debido a la falta de promoción de la reutilización en el mundo del comercio. Por lo cual mediante el esfuerzo impulsado por la universidad a contribuir con los cambios ambientales que están ocurriendo en Uruguay, se está viviendo un tiempo de cambio. De forma que tanto la universidad como los profesores, están haciendo que los alumnos tomen conciencia de esta situación. De manera que la documentación de esta investigación y desarrollo de la aplicación de un sistema, busca aplicar los conceptos aprendidos durante el desarrollo del curso, para poder realizar la implementación de una aplicación orientada a las practicas del Principio 3R, e intentar implementar una aplicación para mejorar esta situación. Por lo cual, algunos de los resultados obtenidos a través del análisis a informes científicos, información certera y conceptos aprendidos a lo largo del curso, enfatizan en la manera correcta de desarrollar software, aplicación correcta de reglas y principios básicos para el desarrollo del mismo. Además de tratar de concientizar a los alumnos de lo que esta sucediendo con el ámbito del reciclaje. De tal manera que estas campañas repercuten a la hora de realizar la especificación del sistema, de forma que se abren nuevos campos a desarrollar.

Índice general

1. Versionado	2
1.1. Repositorio Utilizado	2
1.1.1. Repositorio del obligatorio:	3
1.2. Criterios de versionado	3
1.3. Resumen del log de versiones	7
2. Codificación	10
2.1. Estándar de codificación	10
2.2. Pruebas Unitarias	19
2.3. Análisis de código	29
3. Interfaz de Usuario y Usabilidad	32
3.1. Criterios de interfaz de usuario	32
3.2. Evaluación de usabilidad	34
3.2.1. Heurísticas de Nielsen	36
3.2.2. Pruebas con Usuarios	41
4. Pruebas Funcionales	42
4.1. Técnicas de prueba aplicadas	42
4.2. Casos de prueba	43
4.3. Sesiones de ejecución de pruebas	49
5. Reporte de Defectos	50
5.1. Detección y documentación de defectos	50
5.2. Definición de categorías de defectos	52
5.3. Defectos encontrados por iteración	53
5.4. Estado de calidad global	59
6. Reflexión	62

1. Versionado

En esta sección del obligatorio se realizó el versionado para la construcción del obligatorio. En este caso el software de control de versiones utilizado fue Git. Se utilizaron herramientas como Git Bash, y la interfaz online de Git Hub.

Git, es un software de control de versiones, una de las partes principales del SCM Software configuration manager. Este programa es una herramienta de software que monitorea y gestiona cambios en un sistema de archivos. Este programa, también es una herramienta que ofrece la posibilidad de compartir e integrar dichos cambios al desarrollo de la aplicación. Además, de que al realizar dichas operaciones, el sistema de versionado monitoreará las acciones de adición, eliminación y modificación hechas en los archivos y directorios.

Algunas otras herramientas de control de versiones son Mercurial, SVN, Preforce, entre otras.

1.1. Repositorio Utilizado

Es evidente que, al estar hablando de una sección de versionado, este obligatorio este relacionado con la utilización de dicho software. De forma que efectivamente, en este obligatorio se utilizo el sistema de versionado Git y se utilizo además la creación del repositorio en la plataforma GitHub. De forma que GitHub, es un servicio que nos brinda la posibilidad de las gestión de proyectos. Dentro de la misma se nos ofrece la visualización de los cambios realizados en el proyecto, fecha en las cuales se realizaron y los cambios en el programa.

Además, considerando que lo importante es que GitHub, es una herramienta que nos permite tener nuestros repositorios de Git, se deberá hablar del concepto de repositorio, el cual se define como un lugar donde se guardan los códigos con los cuales estamos trabajando, compuesto por un sistema de carpetas y contenedores. De manera que en el mismo, podemos centralizar el contenido del repositorio para poder colaborar con otros miembros de nuestro proyecto.

También, nos ahorramos el mantenimiento de este repositorio pues está alojado en los servidores de Github aunque si lo deseamos podemos crear nuestros propios repositorios con Git en un VPS o para empresas de gran envergadura veremos que Github ofrece otras soluciones. Ellos se definen como una herramienta que te permite gestionar el ciclo de desarrollo completo, desde la planificación hasta el deploy.

1.1.1. Repositorio del obligatorio:

Una vez mencionado de que se trata un software de control de versiones y el servicio que utilizamos, hablaremos de nuestro repositorio. De manera que en nuestro caso, se adjunta un link al repositorio correspondiente: **<https://bit.ly/2pICMVY>** Dentro del mismo, se encuentra el código fuente JAVA FX del proyecto, junto con el ejecutable y las pruebas unitarias.

También dentro de dicho repositorio, se encuentra indexado dentro del código fuente el uso de una librería para la utilización de distintos componentes los cuales se utilizaron en el desarrollo de la interfaz. De forma, que se encuentra en forma relativa, para poder ser utilizado en todas las computadoras que cuenten con JAVA 8. En este caso estamos hablando de la librería Jphoenix, la cual es una librería de código abierto que implementa el Material Design haciendo uso de componentes de Java.

De forma como nombrábamos anteriormente, dicha librería ya se encuentra dentro del obligatorio de manera que el obligatorio funcione correctamente y no se produzcan errores por falta de las librerías ya especificadas.

1.2. Criterios de versionado

En esta sección se hablara de que criterios utilizamos para la organización de nuestro repositorio. De manera que lo que optamos por realizar 2 ramas distintas, a la rama principal (master). En este caso decidimos realizar la creación de una rama de Desarrollo y además la rama de Versiones. La rama de desarrollo la utilizamos para ir desarrollando en conjunto el BackEnd y FrontEnd. De forma que dentro de esta rama, se encontraran una gran cantidad de commits, esto debido a que es donde se realizo todo el desarrollo del código.

La rama de Versiones, la utilizamos para ir subiendo las versiones estables del obligatorio. De manera que pudiésemos ir teniendo versiones anteriores, por si ocurría algo inesperado a la hora del desarrollo.

Por otro lado la rama principal del proyecto, es la rama master, es aquella donde se encuentra el obligatorio finalizado. De forma que dentro de dicha rama, se encuentra la ultima versión realizada del mismo.

Por lo tanto, mostraremos el formato en el cual se realizaron los commits. De forma que todos los commits realizados durante el desarrollo del proyecto siguen el siguiente formato: En este caso estamos posicionados en la rama Desarrollo dentro del obligatorio y se puede observar que dicha rama cuenta con 39 commits. De manera que como mencionábamos anteriormente esta es la rama la cual cuenta con mayor cantidad de commits. Ver figura 1.1

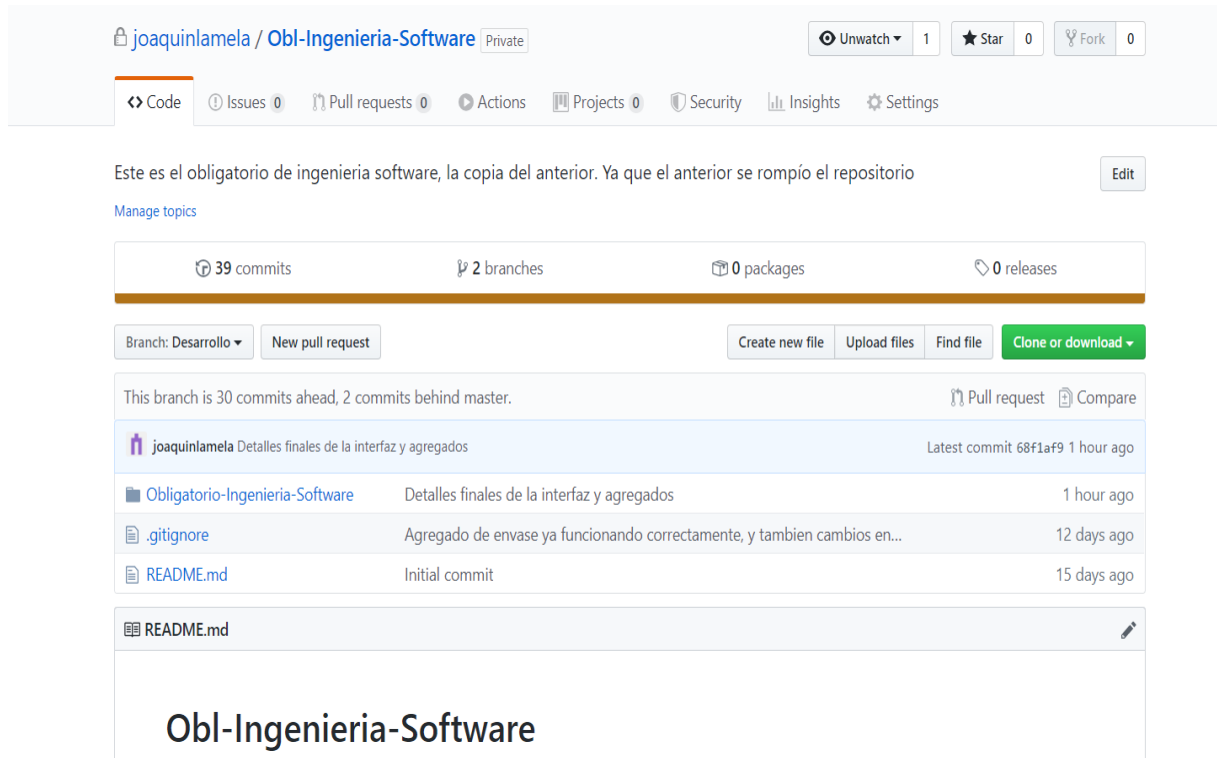


Figura 1.1: Branch Desarrollo

De forma que si nosotros clickeamos en donde dice commits, lo que se mostrara son todos los commits realizado a lo largo del desarrollo del proyecto, de forma que luego mostraremos el formato de los commits realizados durante la construcción del proyecto. Se muestra un ejemplo en la siguiente figura

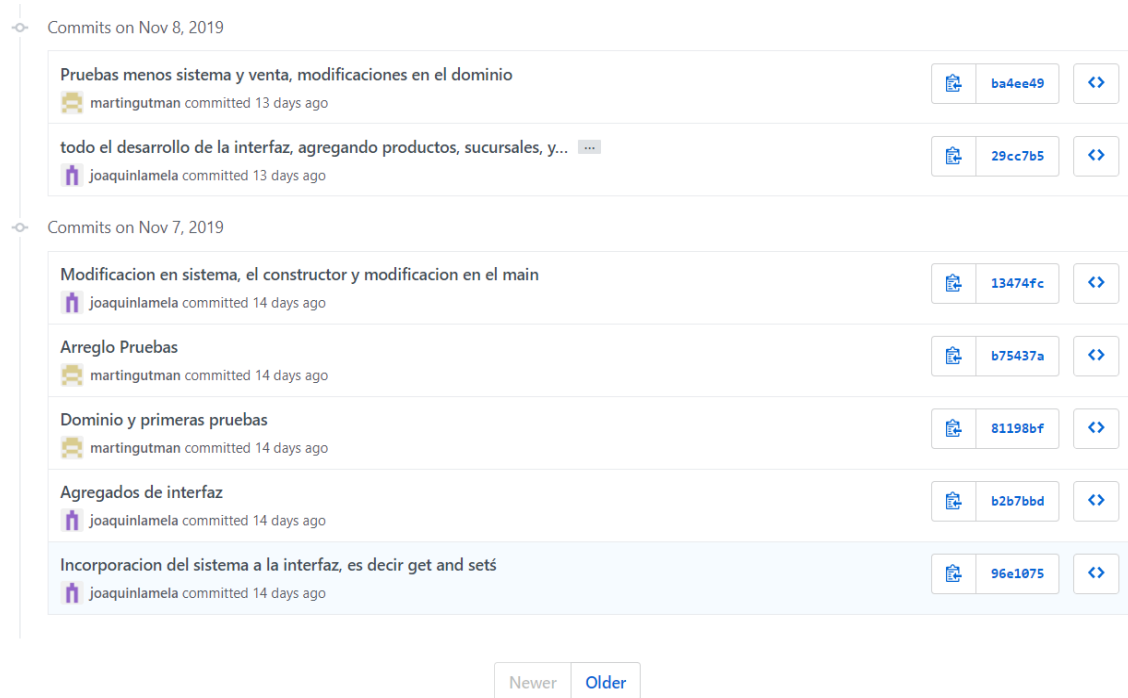


Figura 1.2: Commitments

Como se observa en dicha pantalla, se muestran los commits realizados por los colaboradores del proyecto, los cuales siguen el mismo formato:

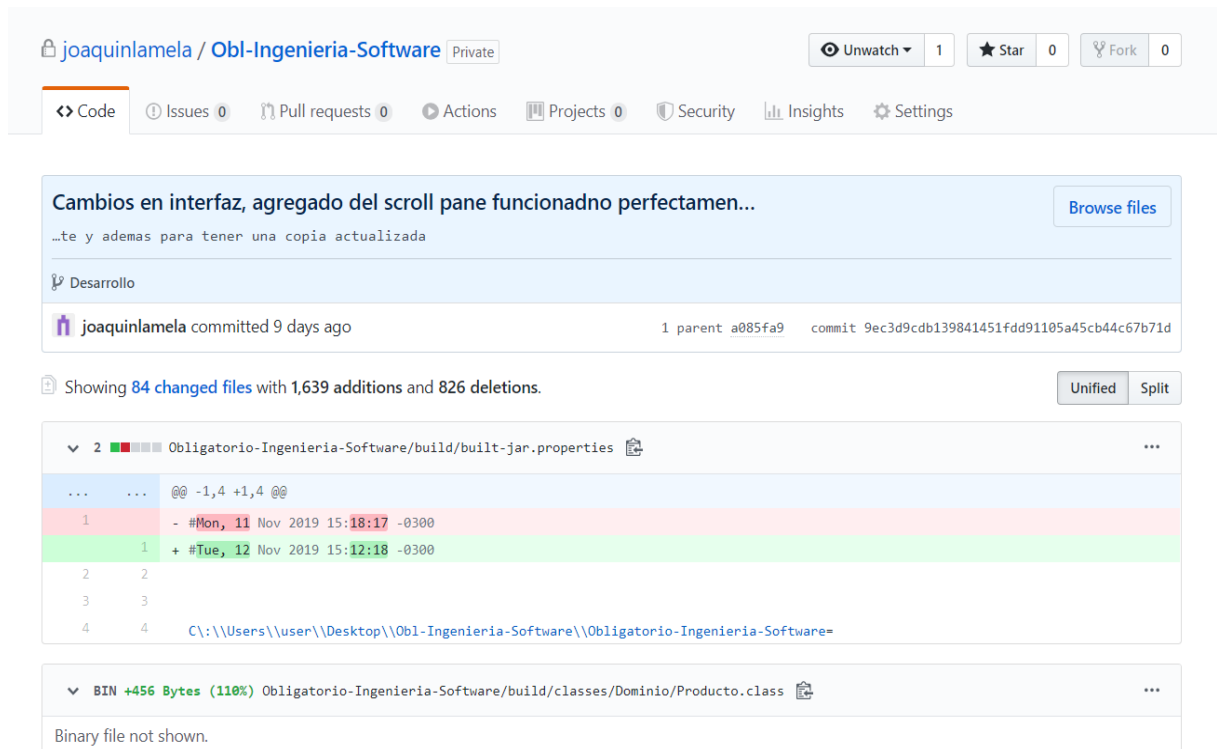


Figura 1.3: Commits de colaboradores

Como se muestra, el commit lleva la siguiente descripción y dentro de ella todas las modificaciones realizadas dentro del proyecto.

1.3. Resumen del log de versiones

En este caso, para esta sub-sección de la documentación del obligatorio, se hablara acerca del desarrollo del proyecto. De manera que uno de los colaboradores del proyecto cuenta con GitHub Pro, lo que va a poder observar además de lo básico ofrecido por el GitHub básico, va a poder observar un grafo con las inserciones y también la cantidad de commits hechos por cada uno de los miembros del proyecto. También esta funcionalidad esta abierta para los repositorios públicos, de manera que por la letra del obligatorio y por la seguridad del código de los integrantes del grupo, esta opción no se puede visualizar.

De forma que al inicio del proyecto, lo primero que se realizo fue la lectura de los comandos utilizados para la utilización correcta de Git. De manera casi inmediata, se procedió a la creación del repositorio, de forma que llegado este punto se comenzó el diagrama de clases de forma física. Lo que se utilizó para facilitar la creación del Dominio del proyecto, de forma que nos fuese mas sencillo a la construcción de la interfaz a través de SceneBuilder y JavaFX. Lo cual nos facilito, también la creación de los métodos y el cumplimiento de los requisitos propuestos por el obligatorio.

Por lo cual con lo primero que se comenzó a codificar, fue toda la realización del BackEnd, en este caso nos referimos dentro del repositorio al package "dominio". Esto debido a que es la parte mas importante de la codificación, debido a que allí se encuentra la clase Sistema, la cual contiene la gran mayoría de los métodos que se utilizan dentro de las interfaces, ejemplo de estos métodos son: ".agregarSucursal", ".agregarEnvase", entre otros.

Es decir que si nosotros eliminamos este package dentro del obligatorio, sería imposible la ejecución del mismo, debido a como se dijo anteriormente es la base de la codificación y de la aplicación en sí.

En este caso, la sincronización con el repositorio online, se realizo por medio del Git Bash. Los comandos utilizados inicialmente y por única vez fueron:

1. Creación de la carpeta donde se pondría el obligatorio
2. Allí haríamos click derecho y abriríamos el Bash.
3. Luego de realizado esto escribiríamos el comando (git clone URL). Es decir donde dice url pondríamos el url de nuestro repositorio de GitHub
4. Luego de esto crearíamos un rama (branch) llamada Desarrollo y otra llamada Versiones, esto con el comando git branch "Nombre de la rama", pero sin las comillas.
5. Nos cambiaríamos a la rama en la cual queremos subir los archivos modificados, con el comando git checkout "Nombre de la rama a la cual nos queremos cambiar"

Luego los comandos que se utilizaron a lo largo del obligatorio, para subir los archivos modificados y las versiones fueron los siguientes:

1. Luego de habernos posicionado en la rama en la cual queremos subir los cambios, haríamos el comando (`git add .`). Que básicamente lo que nos hace es añadirnos todos los archivos modificados al repositorio local.
2. Luego haríamos el commit, con el comando `git commit -m .E1 mensaje o descripción que deseásemos poner`. En nuestro caso el formato esta pre-establecido y se muestra en la sub-sección anterior
3. Y luego por ultimo haríamos el `git push` para subir todos los cambios al repositorio de GitHub (en el caso nuestro).

Por último, cuando quisiéramos subir todos los cambios a la rama master, lo que deberíamos hacer es estar sobre la rama master (si no lo estuviésemos haríamos el comando "`git checkout master`"), y allí escribir el comando de la rama que queremos mezclar con master (`git merge rama con la cual queremos poner los archivos actualizados en master`", ejemplo la rama Desarrollo).

Continuando, con lo que proseguimos a hacer fue la realización de la interfaz en la aplicación Adobe XD. Para luego poder llevarlo a SceneBuilder y hacer la creación correspondiente para JavaFX.

Mientras se realizaba la conexión entre las diferentes escenas del obligatorio (es decir interfaces), se realizaba también los JUnit test. De manera que se progresara tanto en la parte de diseño y codificación, como en las pruebas. Esto facilitó la terminación del obligatorio en menor tiempo debido a las progresiones sistemáticas que tenía el proceso de construcción.

sto gracias a la herramienta de Git y el repositorio online, de forma que avanzamos de manera continua tanto en el FrontEnd como en las pruebas unitarias al mismo tiempo. De manera que se verificase que lo anteriormente codificado del BackEnd estuviese correcto. Resumiéndose en que la base de la aplicación funcionase de la manera esperada y no se encontrasen errores finalizando la creación del misma.

En conjunto mientras se realizaba la finalización de las pruebas unitarias, se realizó la creación de los primeros códigos en los controladores de las interfaces. Esto se comenzó a realizar luego de haber finalizado correctamente con la conexión de las interfaces, de manera que dentro de la codificación de los controles, la codificación que se realizó básicamente fue programación defensiva.

¿Qué quiere decir esto? Básicamente, lo que se realizó fue la verificación de cada uno de los campos que contenían las interfaces donde el usuario podía realizar una acción, ya sea escribir el nombre de un producto hasta seleccionar una imagen. De forma que si no se cumplía con lo esperado, saltase una alerta mostrándole cual fue el error producido, orientándolo a poder utilizar el sistema de manera sencilla.

Básicamente las alertas generadas son tanto en el perfil de usuario como en el perfil de vendedor, alertándole a los mismos, ya sea de campos que han dejado sin completar, formato incorrecto, haberse olvidado de seleccionar los materiales, no haber seleccionado una foto, no haber agregado productos al carrito/pre-venta, fechas no validas, horarios no validos, entre otras alertas. Sin embargo, no todo en el sistema son alertas, sino que también cada vez que se hace una acción correcta en el sistema, se muestra que dicha acción se ha realizado de manera correcta, de forma que el usuario no piense que no ha realizado dicha acción.

Finalizado, cabe destacar que las versiones que se encuentran en la rama de versiones, son cada una de las que se ha hecho a lo largo del proceso de desarrollo del sistema, la primera versión es la mas básica hasta la versión final, la cual es la ultima que se ha desarrollado y es la entrega del obligatorio. Con todo el sistema funcionando correctamente.

2. Codificación

2.1. Estándar de codificación

En primera instancia definiremos los estándares de codificación que debe tener cualquier código, según lo dado en el curso. Entre los elementos básicos encontramos:

- – Comienzo y fin de estructuras
- – Agrupamiento de sentencias
- – Indentación, anidamiento
- – Nombres de elementos
- – Mayúsculas y minúsculas
- – Paréntesis
- – Espaciado de parámetros
- – Estructura de directorio y nombres de archivo

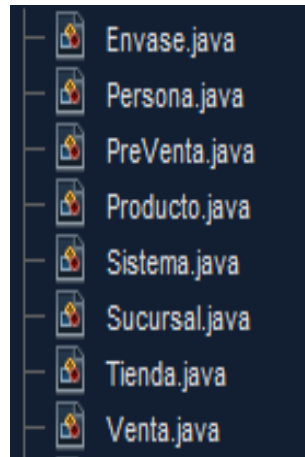
Más específicamente en Java, se debieron seguir los siguientes estándares:

- – Clases comienzan en mayúscula
- – Paquetes y métodos comienzan en minúscula
- – Uso de camelCase para diferenciar palabras
- – Constantes literales todo en mayúsculas
- – Llaves estilo Kernighan y Ritchie

Se exponen ejemplos del código generado que cumplen con esta características.

Clases en mayúsculas:

Tanto las clases en frontend como en el backend comienzan con mayúsculas:



Estilo Kernighan y Ritchie

Este estilo de codificación ha sido muy empleado en la construcción del código. se explica a continuación el criterio y la aplicación que se tuvo en la implementación del programa:

El criterio esta descripto en el libro: ^{E1} lenguaje de programación C”, en el que se explican los criterios de codificación que adoptaron Kernighan y Ritchie. En este caso, se explicará el criterio de llaves que ellos exponen en dicho libro, los cuales fueron utilizados en nuestro código para la mejora de la calidad del mismo.

También se tomaron como ejemplo los java code conventions, expuestos por un artículo de Oracle. El mismo establecía, (y se utilizó en el código):

- **Primeras líneas:** Las primeras líneas de código no comentadas deben ser la declaración del package, y luego del mismo, los imports que presenta la clase en cuestión.
- **Métodos:** Deberán estar agrupados por la funcionalidad, no alfabéticamente, ni por el acceso. Por ejemplo, un método privado puede estar entre dos métodos públicos, ya que el objetivo se basa en la facilitación de la legibilidad y el entendimiento del código.
- **Indentación:** La correcta indentación no está especificada, igualmente, la unidad de la indentación debe ser de 4 espacios.
- **Largo de líneas:** Se deben evitar líneas de más de 80 caracteres.
- **Indentación de métodos:** Se debe seguir una indentación específica para los métodos, que se compone de sentencias claras y en las cuales no haya desfasaje ni lugar a distintas interpretaciones.

[1] [2]

Se expone un ejemplo de una parte de una clase del dominio, en el que aparecen aplicadas las java code conventions expuestas en el artículo de Oracle:

Primeras líneas, definición del package, clase, atributos y constructores

Apreciar los saltos de línea que están definidos correctamente, además de la indentación en todo el código.

```
1. package dominio;
2.
3. import java.util.*;
4.
5. public class Venta {
6.
7.     //Atributos:
8.     private ArrayList<Producto> listaDeProductosAVender;
9.     private int precioTotal;
10.    private int[] cantidadesPorProducto;
11.    private Date fechaDeCompra;
12.    private Persona comprador;
13.    private Tienda echoShop;
14.    private ArrayList<Envase> listaDeEnvasesUtilizados;
15.    private int codigoIdentificadorDeVenta;
16.    private String direccionAEnviar;
17.
18.    //Constructor:
19.    public Venta( ArrayList<Producto> listaDeProductosAVender, int precioTotal, int[] cantidadesPorProducto, Date fechaDeCompra, Persona comprador, Tienda tienda, ArrayList<Envase> listaDeEnvases, int codigoIdentificador, String direccion) {
20.        this.setListaDeProductosAVender(listaDeProductosAVender);
21.        this.setPrecioTotal(precioTotal);
```



```

22.      this.setCantidadesPorProducto(cantidadesPorProducto);
23.      this.setFechaDeCompra(fechaDeCompra);
24.      this.setComprador(comprador);
25.      this.setEchoShop(tienda);
26.      this.setListaDeEnvasesUtilizados(listaDeEnvases);
27.      this.setCodigoIdentificadorDeVenta(codigoIdentificador);
28.      this.setDireccionAEnviar(direccion);
29.  }
30.
31.  public Venta() {
32.      this.setListaDeProductosAVender(new ArrayList<Producto>());
33.      this.setPrecioTotalGrindEQ1;
34.      this.setCantidadesPorProducto(new int[10]);
35.      this.setFechaDeCompra(new Date());
36.      this.setComprador(new Persona());
37.      this.setEchoShop(new Tienda());
38.      this.setListaDeEnvasesUtilizados(new ArrayList<Envase>());
39.      this.setCodigoIdentificadorDeVentaGrindEQ1;
40.      this.setDireccionAEnviar("Direccion");
41.  }
42.

```

Ejemplo de definición de getters y setters

```
43.    //Getters && Setters
44.    public ArrayList<Producto> getListaDeProductosAVender() {
45.        return listaDeProductosAVender;
46.    }
47.
48.    public void setListaDeProductosAVender(ArrayList<Producto> listaDe-
    ProductosAVender) {
49.        this.listaDeProductosAVender = listaDeProductosAVender;
50.    }
51.
52.    public int getPrecioTotal() {
53.        return precioTotal;
54.    }
55.
56.    public void setPrecioTotal(int precioTotal) {
57.        if (precioTotal >= 1) {
58.            this.precioTotal = precioTotal;
59.        } else {
60.            throw new RuntimeException("PrecioMayorA0");
61.        }
62.    }
```

Ejemplo de definición de Métodos

```
63.    //Metodos
64.    public void agregarProducto(Producto producto) {
65.        if (!this.listaDeProductosAVender.contains(producto)) {
66.            this.listaDeProductosAVender.add(producto);
67.            int tamañoDeListaDeEnvases = producto.getPosiblesEnvasesRecomendados().size();
68.            while (tamañoDeListaDeEnvases > 0) {
69.                Envase envase = producto.getPosiblesEnvasesRecomendados().get(tamañoDe-
                ListaDeEnvases);
70.                tamañoDeListaDeEnvases--;
71.                this.getListaDeEnvasesUtilizados().add(envase);
72.            }
73.            this.precioTotal += producto.getPrecio();
74.        }
75.    }
76.
77.    public void eliminarProducto(Producto producto) {
78.        if (this.listaDeProductosAVender.contains(producto)) {
79.            this.listaDeProductosAVender.remove(producto);
80.            int tamañoDeListaDeEnvases = producto.getPosiblesEnvasesRecomendados().size();
81.            while (tamañoDeListaDeEnvases > 0) {
82.                Envase envase = producto.getPosiblesEnvasesRecomendados().get(tamañoDe-
                ListaDeEnvases);
83.                tamañoDeListaDeEnvases--;
84.                this.getListaDeEnvasesUtilizados().remove(envase);
85.            }
86.            this.precioTotal-= producto.getPrecio();
87.        }
88.    }
```

Ejemplo de redefinición de método toString y otros métodos

```
89.     @Override
90.     public String toString() {
91.         return "Venta{" + "listaDeProductosAVender==" + listaDeProductosA-
Vender
92.             + ", precioTotal==" + precioTotal
93.             + ", cantidadesPorProducto==" + cantidadesPorProducto
94.             + ", fechaDeCompra==" + fechaDeCompra
95.             + ", comprador==" + comprador
96.             + ", echoShop==" + echoShop
97.             + ", listaDeEnvasesUtilizados==" + listaDeEnvasesUtilizados
98.             + ", codigoIdentificadorDeVenta==" + codigoIdentificadorDeVenta
99.             + ", direccionAEnviar==" + direccionAEnviar + '}';
100.    }
101.
102.    public void agregarEnvase(Envase e){
103.        if (!this.listaDeEnvasesUtilizados.contains(e)) {
104.            this.listaDeEnvasesUtilizados.add(e);
105.        }
106.    }
107.
108.    public void eliminarEnvase(Envase e){
109.        if (this.listaDeEnvasesUtilizados.contains(e)) {
110.            this.listaDeEnvasesUtilizados.remove(e);
111.        }
112.    }
113.
114. }
```

Además de los aspectos ya mencionados, que se pudieron apreciar en la clase, existen otros aspectos, que también están implícitos en la clase, que contribuyeron a la mejora de la calidad del código. Entre ellos se destacan los siguientes:

- **Agrupamientos de líneas de código:** Como se pudo apreciar, los códigos están agrupados por comentarios tales como "Getter y Setter", "Métodos", por comentarios que describen métodos, y por espacios que definen los bloques de código que presenta el programa.
- **Comentarios:** En todo el código, se han usado comentarios, los mismos se intentaron hacer concisos y no ambiguos, y se usaron solo en casos necesarios y no obvios.
- **Nombres:** Se han usado palabras simples, y frases nemotécnicas para identificar a variables, métodos, clases, etc. Los nombres de métodos, packages y variables de instancia se nombraron comenzando con minúscula, y el nombre de la clase se nombró con mayúscula siguiendo el estándar con el que se ha trabajado.
- **Ocultamiento de información:** Como se pudo apreciar en la clase de ejemplo, los atributos de las clases son privadas, y su acceso se da tan solo por medio de getters y setters. Los elementos público son limitados y muy específicos, utilizados para alguna funcionalidad en concreto.
- **Otros:**
 1. Redefinición de los equals en todas las clases a excepción de la clase Sistema.
 2. Se evita la asignación en null, y el uso del mismo en general.
 3. Se utilizan puntos de retorno únicos.

2.2. Pruebas Unitarias

Para la codificación de las pruebas unitarias se siguieron los siguientes criterios:

- **Cobertura de las pruebas:**

1. Se probaron cursos normales y alternativos.
2. Se usan conjuntos de datos que prueba los casos de borde.

- **Complejidad:**

1. Independencia en las pruebas
2. Cada prueba verifica una sola función
3. Cada prueba contiene respuesta y evaluación

Además, se utilizaron los criterios ya expuestos que se usaron para todas las clases.

A continuación se expone un ejemplo de una sección de una prueba para una clase del dominio, generado con JUnit:

Definición de las primeras líneas y el package

1. `package` dominio;
- 2.
3. `import` org.junit.After;
4. `import` org.junit.AfterClass;
5. `import` org.junit.Before;
6. `import` org.junit.BeforeClass;
7. `import` org.junit.Test;
8. `import static` org.junit.Assert.*;
- 9.
10. `public class` SucursalTest {
- 11.
12. `public` SucursalTest() {
13. }
- 14.
15. @BeforeClass

```
16.  public static void setUpClass() {  
17.  }  
18.  
19.  @AfterClass  
20.  public static void tearDownClass() {  
21.  }  
22.  
23.  @Before  
24.  public void setUp() {  
25.  }  
26.  
27.  @After  
28.  public void tearDown() {  
29.  }  
30.
```

Se define un conjunto de pruebas que testea un bloque de funciones

```
31.    //Numero de Sucursal
32.    @Test(expected = RuntimeException.class)
33.    public void testGetNumeroSucursal1() {
34.
35.        Sucursal instance = new Sucursal();
36.        instance.setNumeroSucursal(0);
37.        int expResult = 0;
38.        int result = (int) instance.getNumeroSucursal();
39.        assertEquals(expResult, result);
40.
41.    }
42.
43.    @Test(expected = RuntimeException.class)
44.    public void testGetNumeroSucursal2() {
45.
46.        Sucursal instance = new Sucursal();
47.        instance.setNumeroSucursal(-1);
48.        int expResult = -1;
49.        int result = (int) instance.getNumeroSucursal();
50.        assertEquals(expResult, result);
51.
52.    }
53.
54.    @Test
55.    public void testGetNumeroSucursal3() {
56.
```



```

57.     Sucursal instance = new Sucursal();
58.     instance.setNumeroSucursalGrindEQ1;
59.     int expResult = 1;
60.     int result = (int) instance.getNumeroSucursal();
61.     assertEquals(expResult, result);
62.
63. }
64.
65. @Test
66. public void testGetNumeroSucursal4() {
67.
68.     Sucursal instance = new Sucursal();
69.     instance.setNumeroSucursalGrindEQ100;
70.     int expResult = 100;
71.     int result = (int) instance.getNumeroSucursal();
72.     assertEquals(expResult, result);
73.
74. }
75.
76. @Test
77. public void testGetNumeroSucursal5() {
78.
79.     Sucursal instance = new Sucursal();
80.     instance.setNumeroSucursal(100000);
81.     int expResult = 100000;
82.     int result = (int) instance.getNumeroSucursal();
83.     assertEquals(expResult, result);
84.

```

```

85.     }
86.
87.     @Test(expected = RuntimeException.class)
88.     public void testSetNumeroSucursal1() {
89.
90.         int precioDeSucursal = 0;
91.         Sucursal instance = new Sucursal();
92.         instance.setNumeroSucursal(precioDeSucursal);
93.
94.     }
95.
96.     @Test(expected = RuntimeException.class)
97.     public void testSetNumeroSucursal2() {
98.
99.         int precioDeSucursal = -8;
100.        Sucursal instance = new Sucursal();
101.        instance.setNumeroSucursal(precioDeSucursal);
102.
103.    }
104.
105.    @Test
106.    public void testSetNumeroSucursal3() {
107.
108.        int precioDeSucursal = 1;
109.        Sucursal instance = new Sucursal();
110.        instance.setNumeroSucursal(precioDeSucursal);
111.
112.    }

```

```

113.
114.     @Test
115.     public void testSetNumeroSucursal4() {
116.
117.         int precioDeSucursal = 100;
118.         Sucursal instance = new Sucursal();
119.         instance.setNumeroSucursal(precioDeSucursal);
120.
121.     }
122.
123.     @Test
124.     public void testSetNumeroSucursal5() {
125.
126.         int precioDeSucursal = 10000;
127.         Sucursal instance = new Sucursal();
128.         instance.setNumeroSucursal(precioDeSucursal);
129.
130.     }
131.

```

Para realizar estas clases se siguieron los mismos estándares que en el resto de las clases del proyecto. Por otro lado, se trabajó con los estándares de pruebas unitarias los cuales se aplicaron en las pruebas unitarias. Entre ellos se destacan:

- Trazabilidad de las pruebas
- Set de pruebas completas y probando casos de borde (estándares de pruebas de caja negra)
- Entradas en condicionales e iteraciones (estándares de prueba de caja blanca)
- Set de pruebas consistente

Cobertura de pruebas Unitarias

Para la verificación de las pruebas unitarias, se utilizó la herramienta JaCoCoverage. Se utilizó al descargarlo en Java y aplicarlo como un plugin.

Dentro de las clases cuyo JUnit asociado que fue probada fue Venta, la cual fue probada con la herramienta de JaCoCoverage, obteniendo la siguiente salida:





































● agregarProducto(Producto)		68%		75%	1	3	4	10	0	1
● eliminarProducto(Producto)		67%		50%	2	3	4	10	0	1
● hashCode()		0%		0%	2	2	2	2	1	1
● getCantidadesPorProducto()		0%	n/a	n/a	1	1	1	1	1	1
● static {...}		75%		50%	1	2	0	1	0	1
● Venta()		100%	n/a	n/a	0	1	0	11	0	1
● equals(Object)		100%		75%	1	3	0	6	0	1
● setDireccionAEnviar(String)		100%		100%	0	2	0	4	0	1
● setPrecioTotal(int)		100%		100%	0	2	0	4	0	1
● setCodigoIdentificadorDeVenta(int)		100%		100%	0	2	0	4	0	1
● agregarEnvase(Envase)		100%		100%	0	2	0	3	0	1
● eliminarEnvase(Envase)		100%		100%	0	2	0	3	0	1
● setListaDeProductosAVender(ArrayList)		100%	n/a	n/a	0	1	0	2	0	1
● setCantidadesPorProducto(int[])		100%	n/a	n/a	0	1	0	2	0	1
● setFechaDeCompra(Date)		100%	n/a	n/a	0	1	0	2	0	1
● setComprador(Persona)		100%	n/a	n/a	0	1	0	2	0	1
● setEchoShop(Tienda)		100%	n/a	n/a	0	1	0	2	0	1
● setListaDeEnvasesUtilizados(ArrayList)		100%	n/a	n/a	0	1	0	2	0	1
● getListaDeProductosAVender()		100%	n/a	n/a	0	1	0	1	0	1
● getPrecioTotal()		100%	n/a	n/a	0	1	0	1	0	1
● getFechaDeCompra()		100%	n/a	n/a	0	1	0	1	0	1
● getComprador()		100%	n/a	n/a	0	1	0	1	0	1
● getEchoShop()		100%	n/a	n/a	0	1	0	1	0	1
● getListaDeEnvasesUtilizados()		100%	n/a	n/a	0	1	0	1	0	1
● getCodigoIdentificadorDeVenta()		100%	n/a	n/a	0	1	0	1	0	1
● getDireccionAEnviar()		100%	n/a	n/a	0	1	0	1	0	1
Total	124 of 355	65%	7 of 26	73%	10	41	24	92	4	28

Figura 2.1: Pruebas iniciales JaCoCoverage

Se realizaron cambios, y se mejoraron las pruebas, para poder mejorar el resultado de la prueba, y poder hacerla más efectiva. Los mismos se exponen en las figuras siguientes:

```
public void testGetListaDeProductos4() {  
  
    Envase e = new Envase();  
    Venta instance = new Venta();  
    ArrayList<Producto> a = new ArrayList<>();  
    Producto p = new Producto();  
    p.agregarEnvase(e);  
    p.setCodigoIdentificador(18);  
    Producto f = new Producto();  
    Producto g = new Producto();  
    g = f;  
    instance.agregarProducto(f);  
    instance.agregarProducto(p);  
    instance.agregarProducto(g);  
    instance.eliminarProducto(f);  
    instance.eliminarProducto(p);  
  
    ArrayList<Producto> expResult = a;  
    ArrayList<Producto> result = instance.getListaDeProductosAVender();  
    assertEquals(expResult, result);  
}
```

Figura 2.2: Cambios recomendados por JaCoCoverage

Se pudo apreciar la mejora en los resultados del JaCoCoverage:














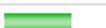
















agregarProducto(Producto)		100%		100%	0	3	0	10	0	1
eliminarProducto(Producto)		100%		75%	1	3	0	10	0	1
equals(Object)		100%		75%	1	3	0	6	0	1
setDireccionAEnviar(String)		100%		100%	0	2	0	4	0	1
setPrecioTotal(int)		100%		100%	0	2	0	4	0	1
setCodigoIdentificadorDeVenta(int)		100%		100%	0	2	0	4	0	1
agregarEnvase(Envase)		100%		100%	0	2	0	3	0	1
eliminarEnvase(Envase)		100%		100%	0	2	0	3	0	1
setListaDeProductosAVender(ArrayList)		100%		n/a	0	1	0	2	0	1
setCantidadesPorProducto(int[])		100%		n/a	0	1	0	2	0	1
setFechaDeCompra(Date)		100%		n/a	0	1	0	2	0	1
setComprador(Persona)		100%		n/a	0	1	0	2	0	1
setEchoShop(Tienda)		100%		n/a	0	1	0	2	0	1
setListaDeEnvasesUtilizados(ArrayList)		100%		n/a	0	1	0	2	0	1
getListaDeProductosAVender()		100%		n/a	0	1	0	1	0	1
getPrecioTotal()		100%		n/a	0	1	0	1	0	1
getFechaDeCompra()		100%		n/a	0	1	0	1	0	1
getComprador()		100%		n/a	0	1	0	1	0	1
getEchoShop()		100%		n/a	0	1	0	1	0	1
getListaDeEnvasesUtilizados()		100%		n/a	0	1	0	1	0	1
getCodigoIdentificadorDeVenta()		100%		n/a	0	1	0	1	0	1
getDireccionAEnviar()		100%		n/a	0	1	0	1	0	1
Total	98 of 357	73%	5 of 26	81%	8	41	16	92	4	28

Figura 2.3: Proceso de mejora de calidad del código JaCoCoverage

Al implementar mejoras en el equals y en el eliminar de Venta, se llegó al resultado esperado.































agregarProducto(Producto)		100%		100%	0	3	0	10	0	1
eliminarProducto(Producto)		100%		100%	0	3	0	10	0	1
equals(Object)		100%		100%	0	3	0	6	0	1
setDireccionAEnviar(String)		100%		100%	0	2	0	4	0	1
setPrecioTotal(int)		100%		100%	0	2	0	4	0	1
setCodigoIdentificadorDeVenta(int)		100%		100%	0	2	0	4	0	1
agregarEnvase(Envase)		100%		100%	0	2	0	3	0	1
eliminarEnvase(Envase)		100%		100%	0	2	0	3	0	1
setListaDeProductosAVender(ArrayList)		100%	n/a		0	1	0	2	0	1
setCantidadesPorProducto(int[])		100%	n/a		0	1	0	2	0	1
setFechaDeCompra(Date)		100%	n/a		0	1	0	2	0	1
setComprador(Persona)		100%	n/a		0	1	0	2	0	1
setEchoShop(Tienda)		100%	n/a		0	1	0	2	0	1
setListaDeEnvasesUtilizados(ArrayList)		100%	n/a		0	1	0	2	0	1
getListaDeProductosAVender()		100%	n/a		0	1	0	1	0	1
getPrecioTotal()		100%	n/a		0	1	0	1	0	1
getFechaDeCompra()		100%	n/a		0	1	0	1	0	1
getComprador()		100%	n/a		0	1	0	1	0	1
getEchoShop()		100%	n/a		0	1	0	1	0	1
getListaDeEnvasesUtilizados()		100%	n/a		0	1	0	1	0	1
getCodigoIdentificadorDeVenta()		100%	n/a		0	1	0	1	0	1
getDireccionAEnviar()		100%	n/a		0	1	0	1	0	1
Total	98 of 357	73%	3 of 26	88%	6	41	16	92	4	28

Figura 2.4: Avance completo de mejora de calidad del código JaCoCoverage

El proceso fue reiterado con todas las pruebas unitarias.

2.3. Análisis de código

Se utilizó la herramienta Find Bugs, para mejorar la calidad del código. Las mejoras que se debieron realizar fueron las siguientes:

1. Mejoras en el método equals de cada una de las clases en las que estaba implementado

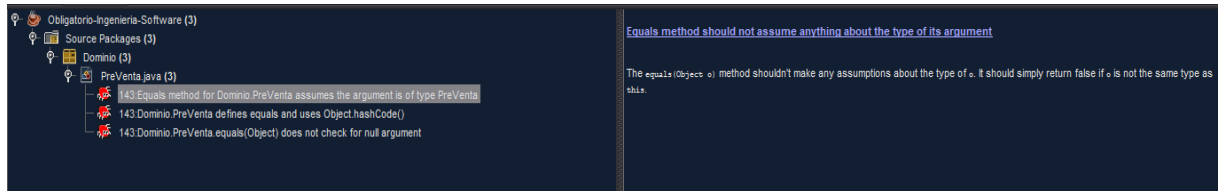


Figura 2.5: Aplicación de findbugs al proyecto

Este fue uno de los cambios que se debieron realizar en todas las clases del backend, debido a que la implementación del método equals no consideraba estos detalles. Un ejemplo de el método que debió de ser mejorado, se muestra a continuación:

```
@Override
public boolean equals(Object o) {
    PreVenta preVenta = (PreVenta) o;
    return this.getIdificadorDePreventa() ==
        preVenta.getIdificadorDePreventa();
}
```

Figura 2.6: Redefinición del método equals correctamente

Siguiendo las mejoras recomendadas por el FindBugs, se diseño el siguiente código:

```
@Override
public boolean equals(Object o) {
    boolean retorno = false;
    if (o instanceof PreVenta) {
        PreVenta preVenta = (PreVenta) o;
        retorno = this.getIdentificadorDePreventa()
            == preVenta.getIdentificadorDePreventa();
    }
    return retorno;
}

public int hashCode() {
    assert false : "hashCode not designed";
    return 1;
}
```

Figura 2.7: Redefinición del método equals correctamente

Otro de los cambios a realizar fue con respecto a la impresión de arrays. En una de las clases que se daba esto fue en la clase Venta, en la cual, se imprimía el array de cantidades por producto.

```
@Override
public String toString() {
    return "Venta{" + "listaDeProductosAVender=" + listaDeProductosAVender.toString()
        + ", precioTotal=" + precioTotal
        + ", cantidadesPorProducto=" + cantidadesPorProducto
        + ", fechaDeCompra=" + fechaDeCompra
        + ", comprador=" + comprador
        + ", echoShop=" + echoShop
        + ", listaDeEnvasesUtilizados=" + listaDeEnvasesUtilizados
        + ", codigoIdentificadorDeVenta=" + codigoIdentificadorDeVenta
        + ", direccionAEnviar=" + direccionAEnviar + '}';
}
```

Figura 2.8: Redefinición del método toString

La recomendación que propone findbugs fue la siguiente:

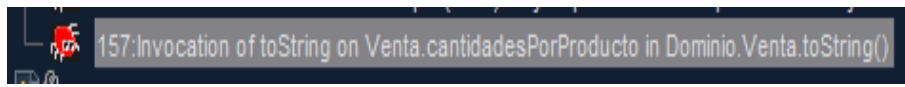


Figura 2.9: Redefinición del método toString

Se solucionó de la siguiente manera:

```
@Override
public String toString() {
    return "Venta{" + "listaDeProductosAVender=" + listaDeProductosAVender.toString()
        + ", precioTotal=" + precioTotal
        + ", cantidadesPorProducto=" + Arrays.toString(cantidadesPorProducto)
        + ", fechaDeCompra=" + fechaDeCompra
        + ", comprador=" + comprador
        + ", echoShop=" + echoShop
        + ", listaDeEnvasesUtilizados=" + listaDeEnvasesUtilizados
        + ", codigoIdentificadorDeVenta=" + codigoIdentificadorDeVenta
        + ", direccionAEnviar=" + direccionAEnviar + '}';
}
```

Figura 2.10: Redefinición del método toString

3. Interfaz de Usuario y Usabilidad

3.1. Criterios de interfaz de usuario

La interfaz es uno de los aspectos más importantes para todo sistema, es la forma de atraer una gran cantidad de usuarios, tan solo por medio de la vista, y de retener a los mismos por medio de la eficacia, eficiencia y satisfacción que la misma brinda

Cada interfaz puede tomar un rumbo, y un diseño completamente distinto a otras. Existen varios enfoques posibles que puede tomar una aplicación, entre ellos, se destaca la diferenciación en el estilo de interacción. Este parámetro se puede implementar de distintas maneras (con sus pros y contras):

- **Entrada de datos en consola:** Flexible, denso, aunque requiere entrenamiento y retención
- **Lenguaje natural:** Esfuerzos de aprendizaje casi nulo, caro de implementar, poco predecible aún con tecnologías del estado del arte

Otro de los parámetros que puede variar con respecto a la forma de la implementación es la tolerancia a errores, la cual se presenta en sus distintas variantes:

- **“Exploración segura”:** permitir que el usuario explore sin miedo a hacer daños
- **Permitir deshacer o revertir** el efecto de las acciones del usuario
- **Prevenir errores** lo más posible. E.g. deshabilitar opciones de menú que no aplican en el contexto en cuestión
- **Mensajes de error** constructivos

Otro parámetro que varía es la consistencia, que se puede presentar de las siguientes formas:

- A nivel de plataforma
- Dentro de la aplicación:
 1. Terminología consistente (“alta”, “ingreso”, “agregar”, “nuevo”, etc.)
 2. Colores
 3. Formatos
 4. Uso de mayúsculas
 5. Tono de voz en los mensajes al usuario
- A nivel de todos los sub-sistemas y módulos

Por último, el llamado de atención al usuario también se puede codificar de distintas maneras, algunas comunes son:

- **Intensidad**
- **Resaltado:** Subrayados, enmarcados, asteriscos, bullets, etc. son aceptables, pero usados de forma consistente y moderada
- **Tamaño:** Hasta cuatro tamaños de fuente
- **Estilo:** Hasta tres estilos de fuente

3.2. Evaluación de usabilidad

La usabilidad es un atributo de calidad que establece cuan fácil de usar es un sistema.

Se mide según tres dimensiones principales:

1. **Eficacia:** lograr el objetivo deseado por él usuario.
2. **Eficiencia:** el esfuerzo que se requiere para alcanzar el objetivo
3. **Satisfacción:** este punto es subjetivo debido a que es por parte de los usuarios

Esta definición, hace que la usabilidad sea uno de los parámetros que se debe tener más en cuenta a la hora de crear un sistema, ya que este, marcará el gusto del usuario por la aplicación, la facilidad para usarlo, tanto en cuanto al logro de utilizar una función, como al esfuerzo que se requiere para alcanzar la misma, que constituirá en gran parte, la felicidad del usuario.

Este parámetro se puede medir según las métricas de usabilidad, que están dadas por las siguientes características:

- **Tiempo de aprendizaje:** ¿Cuán fácil es para los usuarios realizar las tareas básicas la primera vez que utilizan el sistema?
- **Eficiencia o performance:** Es aquellos usuarios que aprendieron a interactuar con la interfaz ¿cuán rápido pueden realizar las tareas?
- **Tasa de errores:** ¿Cuántos errores cometen los usuarios? ¿Cuán severos son?
- **Retención o memorabilidad:** Cuando los usuarios vuelven a usar el sistema luego de un periodo sin uso. ¿Cuán fácilmente restablecen la habilidad?
- **Satisfacción:** ¿Cuán placentero es el diseño del sistema?

Además, la usabilidad está constituida por 3 desafíos, que constituyen a la creación de un programa excelente, y utilizable por todos los usuarios. Estos son los siguientes:

- Desafíos de diseñar:
 1. Foco en la tarea
 2. Foco en los usuarios
 3. Diversidad cultural
- Foco en los usuarios:
 1. Memoria a corto plazo
 2. Memoria a largo plazo
 3. Toma de decisiones y evaluaciones de riesgos

4. Memoria gráfica y sensorial
 5. Aprendizaje, desarrollo de habilidades, adquisición de conocimiento
- Habilidades físicas en juego:
 1. Accesibilidad universal
 2. Diversidad cultural e internacional:
 3. Caracteres especiales
 4. Idiomas
 5. Fecha y hora
 6. Divisas
 7. Unidades de medidas

Estos atributos, son los que constituyen la Usabilidad de un programa, que, como se pudo apreciar, son la base de un gran programa.

Estas características se pueden medir de forma empírica, utilizando varios métodos y bajo ciertos parámetros, lo cual nos permitirá catalogar si dichas características se están mostrando y ejecutando de manera adecuada, o si es preciso mejorar las mismas.

Existen dos formas para estudiar que se cumplan los parámetros de la Usabilidad, que son los siguientes:

3.2.1. Heurísticas de Nielsen

Estas pruebas, conducidos por diez parámetros descritos por Nielsen, constituyen un método de evaluación de la usabilidad de un sistema, conducido por expertos.

Usando este método, se analiza el sistema utilizando una serie de guías. Además de que más de una persona puede estar implicada en la aplicación de estas pruebas a cierto programa.

Las heurísticas de Usabilidad están dadas por los siguientes parámetros:

1. **Visibilidad del estatus del sistema** - El sistema siempre debe mantener al usuario informado sobre lo que está sucediendo, mostrando las operaciones en cada momento y los estados del sistema. Esto se debe realizar en un tiempo razonable y con un feedback apropiado, que pueda hacer que el usuario se de cuenta de estar en un camino correcto o erróneo y catalogar el error.
2. **Vínculo entre el sistema y el mundo real** - El sistema debe "hablar en el lenguaje del usuario", con palabras, frases y conceptos que le sean familiares, en vez de términos que estén netamente ligados al sistema. Se deben seguir convenciones de frases y palabras reales, haciendo que la información aparezca en un orden lógico y natural. Esto hace que el usuario pueda entender claramente las funciones de la aplicación.
3. **Control y libertad del Usuario** - El usuario elige a veces, funciones por error, por lo cual el sistema debe tener una "salida de emergencia" correctamente señalizada, de forma que se pueda abandonar el estado al cual se accedió por error, sin tener de por medio un diálogo extenso. Esta operación debe implementarse con el undo (deshacer un cambio) y el redo (rehacer cierta acción o cambio).
4. **Consistencia y estándares** - Los usuarios no deberían preocuparse ni siquiera por palabras diferentes, situaciones, o acciones que signifiquen lo mismo. no debe ocurrir en un sistema, que cierta funcionalidad se identifique con dos nombres distintos, ya que esto puede llegar a confundir al usuario. Por otro lado, el sistema debe seguir los estándares de la plataforma.
5. **Prevención de errores** - Mejor que mensajes de error correctos y bien ideados, un diseño minuciosamente estudiado es aquel que previene que los problemas ocurran en primera instancia.
6. **Reconocimiento en vez de memorización** - Hacer que los objetos, acciones, y opciones sean visibles. El usuario no debería tener que memorizar información de una parte de la interacción a otra. Las instrucciones para el uso del programa deben ser visibles o fácilmente reiterables cuando sea apropiado.
7. **Flexibilidad y eficiencia de uso** - Los aceleradores, generalmente hacen más rápida la interacción para los usuarios expertos, por lo cual, el sistema debe poder abastecer, tanto a usuarios experientes, como a los que carecen de la misma. Además, debe poder tener acciones predeterminadas, las cuales son

a las que se acceden de forma paulatina, facilitando el acceso a las mismas y su realización.

8. **Diseño minimalista** - La interacción no debe tener información irrelevante o que sea raramente utilizada. Esto se da, ya que toda información irrelevante en la interacción, compite con la relevante, y disminuyen de forma drástica su visibilidad.
9. **Ayudar a los usuarios a reconocer, diagnosticar, y recuperarse de los errores** - Los mensajes de error deben ser explicados en el cotidiano (no en el de código), y deben proponer una solución de forma constructiva.
10. **Ayuda y documentación** - Sin importar que es mejor que el sistema sea usado sin una documentación, en algunos casos puede ser necesario la asistencia por medio de los documentos adjuntos. En estos, todo tipo de información debe ser sencilla de encontrar, se debe enfocar en las tareas del usuario, listar pasos que se deben seguir en cada caso y no debe ser muy larga [3].

Aplicación a nuestra aplicación

Se destacan las heurísticas aplicadas, y si se cumplen o no los parámetros que se destacaron en la primera parte de la descripción de los parámetros de la interfaz.

Heurísticas de Nielsen:

Visibilidad del estatus del sistema: En toda la aplicación se muestran mensajes, tanto de error, como de buen manejo de las funcionalidades, indicando al usuario si se están realizando de manera correcta las acciones, o si debe tomar otro camino para alcanzar el objetivo deseado.

Se muestran algunos ejemplos a continuación.

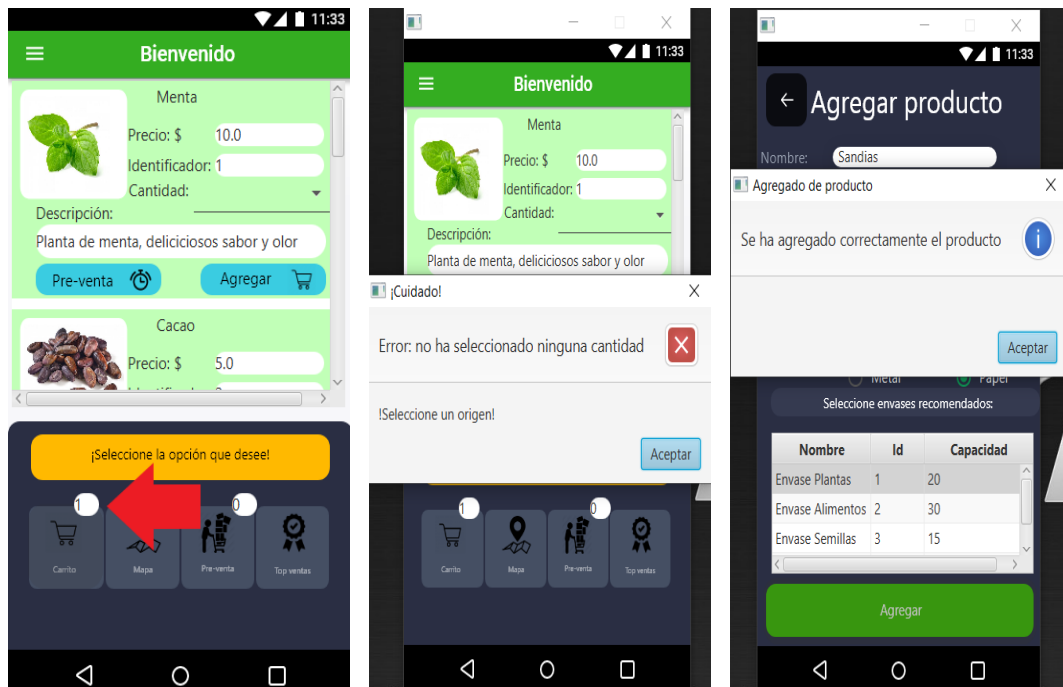


Figura 3.1: Visibilidad del estatus del sistema

Relación entre el sistema y el mundo real: En toda la aplicación se muestran alusiones al mundo real, tanto con la inclusión del carrito con la sección de beneficios que se muestran según la cantidad de ventas en un mes.

Se muestran algunos ejemplos a continuación.

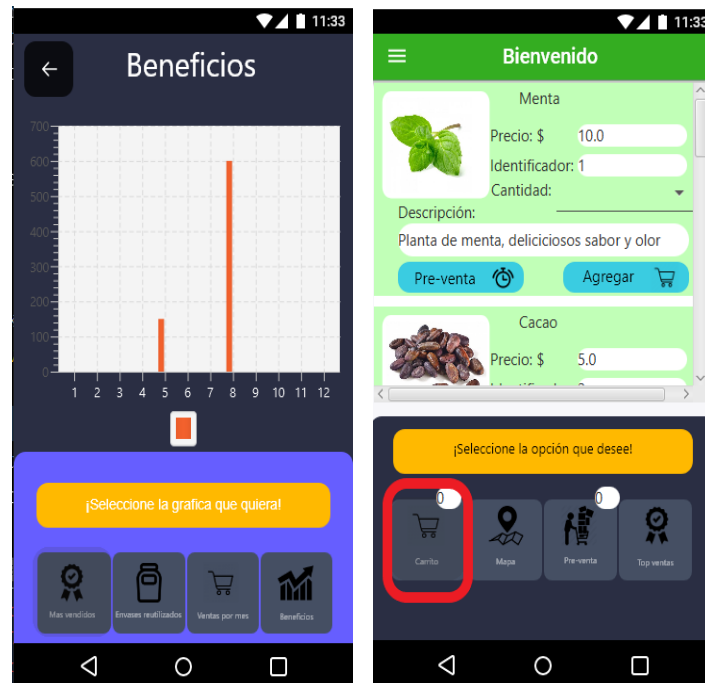


Figura 3.2: Relación entre el sistema y el mundo real

Consistencia y estándares: La aplicación esta programada de forma que no se encuentren palabras distintas que refieran a lo mismo, y se programó siguiendo los estándares dados en el curso.

Diseño minimalista: En el diseño de la interfaz, se puede apreciar que no hay información innecesaria, de forma que el usuario se puede centrar en las funcionalidades reales del sistema y no "distraerse", con el resto de cosas que no sirven (las cuales no se presentan en las stages).

Reconocimiento en vez de memorización: En la interfaz, no se requiere en ningún momento que el usuario memorice ningún tipo de información.

Ayuda a usuarios a diagnosticar, reconocer y recuperarse de errores:
En todas partes de la aplicación, en caso de que se inserte información incorrecta, se introducen avisos que ayudan a que el cliente se recupere de los mismos.

Se muestran ejemplos a continuación:

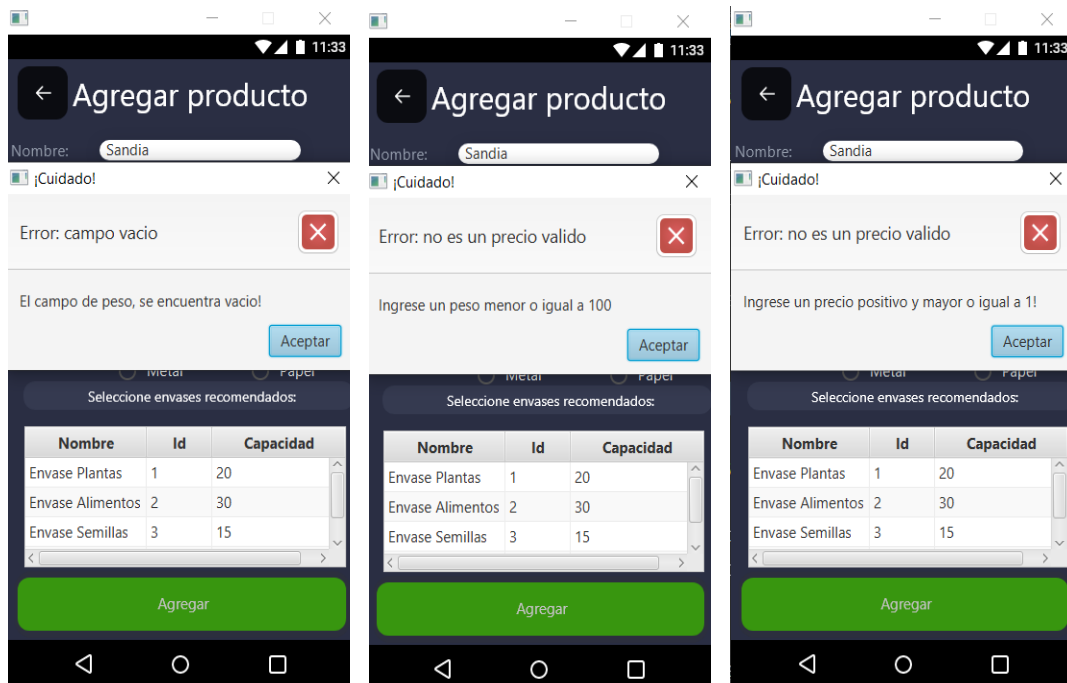


Figura 3.3: Diagnóstico y reconocimiento de errores

3.2.2. Pruebas con Usuarios

Se debe considerar en primera instancia que todo sistema es probado por los usuarios en algún momento. Esta prueba consiste en que un grupo de los usuarios, dentro de los stakeholders, testeen el avance de la aplicación (o la aplicación terminada), dando sus críticas y feedback correspondientes, además de verificar que tan atractiva es la interfaz y cuales son las funcionalidades más utilizadas. Esto se hace con el fin de que, un grupo dentro de los usuarios hacia los cuales está enfocado el sistema, lo prueben y den su opinión, sugiriendo mejoras para la aplicación.

En las pruebas con usuarios, el tester realiza la observación, interpretación, y medición correspondientes a las tareas realizadas por el usuarios.

Para estas pruebas se le pide al usuario que realice ciertas tareas, simulando un entorno de trabajo en situaciones reales, la misma puede realizarse en laboratorios de usabilidad utilizando herramientas específicas u observación directa.

Una forma de determinar encausar las observaciones y mediciones puede ser siguiendo los parámetros:

- Tiempo en la tarea
- Precisión
- Recordación
- Respuesta emocional

4. Pruebas Funcionales

El proceso de la realización de las pruebas está definido por Glen Myers, como el procesos de ejecutar el software con el objetivo de encontrar defectos.

Un prueba es efectiva en el caso de que la misma detecte errores en el software, y un buen caso de prueba es aquel con alta probabilidad de descubrir un error no encontrado hasta el momento. Esto se determina según lo que debe, o no debe hacer el programa, que se puede apreciar al comparar el resultado esperado, con el obtenido.

Además, se debe tener en cuenta que si la prueba no detecta errores, eso no significa que el programa no presente errores. Lo que se puede afirmar es, solamente, que bajo los casos probados, el resultado esperado coincide con el obtenido.

4.1. Técnicas de prueba aplicadas

Existen varios tipos de manejo de pruebas con el fin que las mismas presenten, entre ellos encontramos el TDD, Test-Driven Development, que consiste en la creación de las pruebas previo a la codificación y programación del software. Por otro lado, encontramos otros tipos de manejo de prueba, las cuales fueron utilizadas en el trabajo.

Se aplicó además, un sistema de pruebas de caja negra, siguiendo los cursos normales y alternativos de la funcionalidad de el agregado de productos al sistema.

Uno de los tipos de prueba que se aplicó en el proyecto, fue el sistema de pruebas unitarias, que es una forma de probar el funcionamiento de un módulo de código, antes de integrarlo al resto del sistema, lo cual permite detectar fallas tempranas, y arreglarlas antes del acoplamiento. Asegurando luego (Bajo el margen de error que tiene todo sistema de pruebas), que los errores no serán del módulo ya probado.

Para el caso de nuestro software, cada una de las clases presenta pruebas unitarias, programadas con JUnit, que permitieron probar todos los métodos de la clase, antes de que se pruebe con la interfaz. Las mismas se encuentran en la sección de codificación.

4.2. Casos de prueba

Se probó la funcionalidad del agregado de un producto
El diagrama asociado es el siguiente:

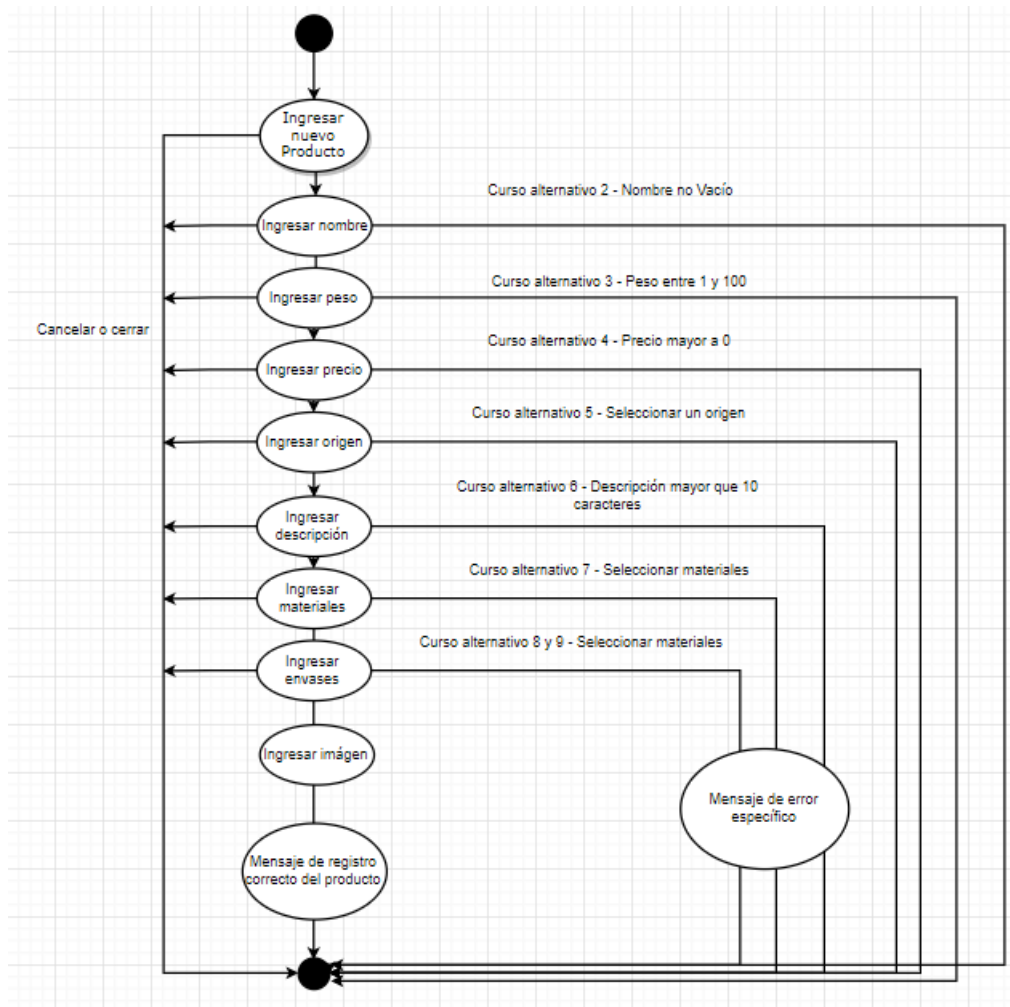


Figura 4.1: Diagrama de caso de uso de agregado de productos

Prueba funcional – Caja Negra – Agregar Productos

Escenarios posibles

Escenario	Nombre	Curso de comienzo	Curso alternativo
Escenario 1	Datos correctos	Camino básico	
Escenario 2	Verificación del nombre	Camino básico	CA 2.1
Escenario 3	Peso Incorrecto	Camino básico	CA 3.1
Escenario 4	Precio Incorrecto	Camino básico	CA 4.1
Escenario 5	Sin Origen	Camino básico	CA 5.1
Escenario 6	Descripción Incorrecta	Camino básico	CA 6.1
Escenario 7	Falta seleccionar Materiales	Camino básico	CA 7.1
Escenario 8	Falta seleccionar Envases	Camino básico	CA 8.1
Escenario 9	Envases no soportan el peso del producto	Camino básico	CA 9.1



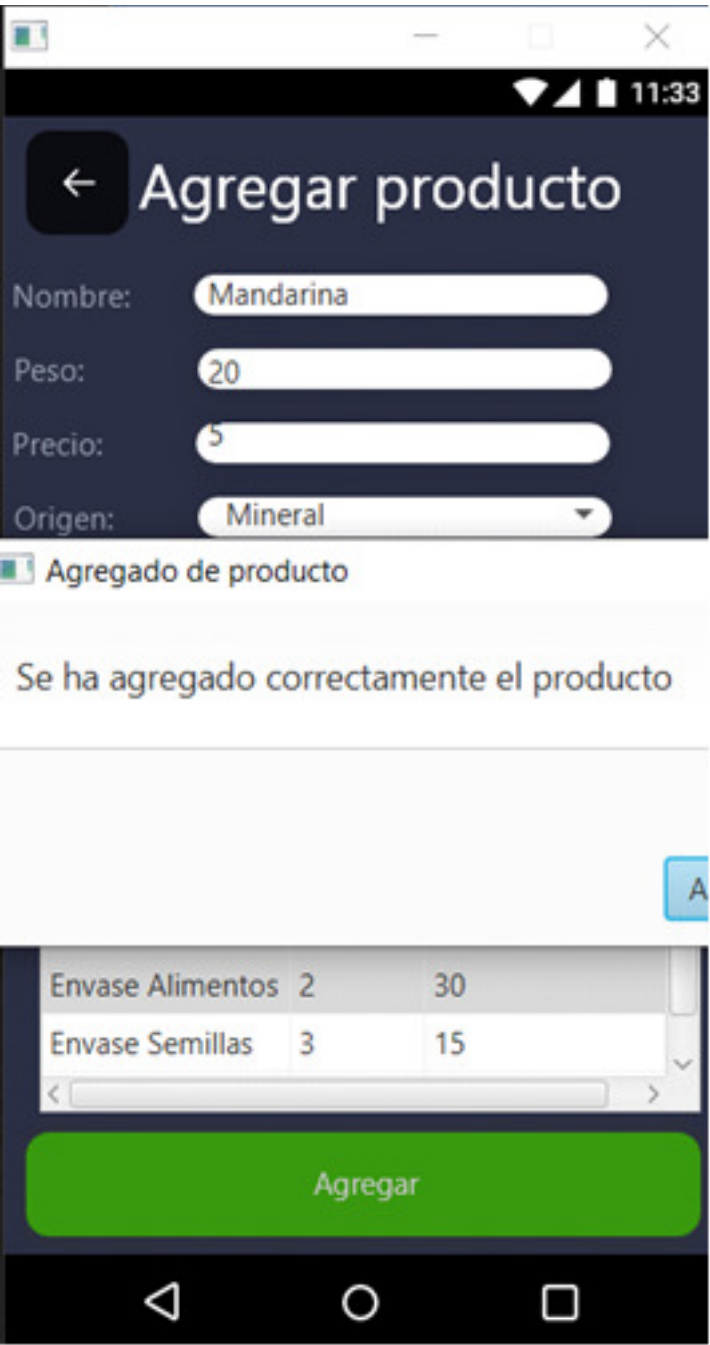
Caso de prueba	Escenario	Nombre	Peso	Precio	Origen	Descripción	Materiales	Envases	Resultado esperado
CP 1.1	Escenario 1	V	V	V	V	V	V	V	Registro del cliente
CP 2.1	Escenario 2	NV	V	V	V	V	V	V	Nombre incorrecto
CP 3.1	Escenario 3	V	NV	V	V	V	V	V	Peso incorrecto
CP 4.1	Escenario 4	V	V	NV	V	V	V	V	Precio incorrecto
CP 5.1	Escenario 5	V	V	V	NV	V	V	V	Sin Origen
CP 6.1	Escenario 6	V	V	V	V	NV	V	V	Descripción incorrecta
CP 7.1	Escenario 7	V	V	V	V	V	NV	V	Seleccionar Materiales
CP 8.1	Escenario 8	V	V	V	V	V	V	NV	Seleccionar Materiales
CP 9.1	Escenario 9	V	V	V	V	V	V	V	No hay envases que soporten el peso del producto



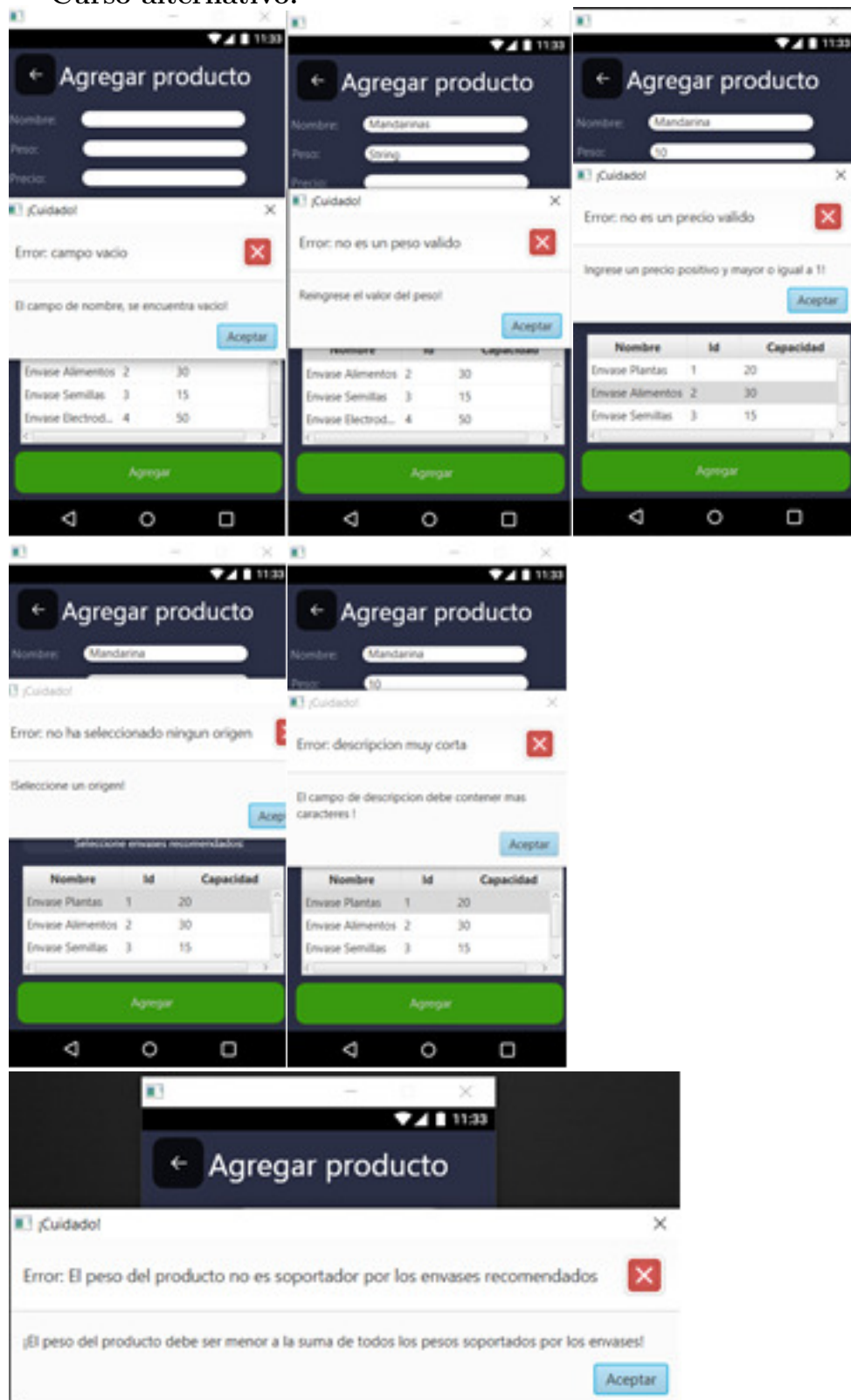
Condición	Clases válidas	Clases no válidas
Nombre	Cualquier <u>String</u> no vacío	
Peso	Valor entero entre 1 y 100	<u>String</u> no numérico
Precio	Valor entero mayor a 1	<u>String</u> no numérico
Origen	Uno de los cinco valores presentes	
Descripción	<u>String</u> con 10 o más caracteres	
Materiales	Uno de los cuatro valores presentes	
Envases	Uno o más envases presentes. El peso del producto, debe ser menor al mayor peso soportado de los envases seleccionados	

Caso de prueba	Escenario	Nombre	Peso	Precio	Origen	Descripción	Materiales	Envases	Resultado esperado
CP 2.1	Escenario 2	"	V	V	V	V	V	V	Mensaje: "Error, Campo Vacío"
CP 2.2	Escenario 2	" "	V	V	V	V	V	V	Mensaje: "Error, Campo Vacío"
CP 3.1	Escenario 3	V	"String"	V	V	V	V	V	Peso incorrecto
CP 3.2	Escenario 3	V	"0"	V	V	V	V	V	Peso incorrecto
CP 3.3	Escenario 3	V	"101"	V	V	V	V	V	Peso incorrecto
CP 3.4	Escenario 3	V	"-1"	V	V	V	V	V	Peso incorrecto
CP 3.5	Escenario 3	V	"100"	V	V	V	V	V	Peso correcto
CP 3.6	Escenario 3	V	"1"	V	V	V	V	V	Peso correcto
CP 3.7	Escenario 3	V	"50"	V	V	V	V	V	Peso correcto
CP 3.8	Escenario 3	V	"99"	V	V	V	V	V	Peso correcto
CP 4.1	Escenario 4	V	V	"0"	V	V	V	V	Precio incorrecto
CP 4.2	Escenario 4	V	V	"-1"	V	V	V	V	Precio incorrecto
CP 4.3	Escenario 3	V	V	"String"	V	V	V	V	Precio incorrecto
CP 4.4	Escenario 3	V	V	"	V	V	V	V	Precio incorrecto
CP 4.5	Escenario 4	V	V	"1"	V	V	V	V	Precio correcto
CP 4.6	Escenario 4	V	V	"100"	V	V	V	V	Precio correcto
CP 5.1	Escenario 5	V	V	V	NV	V	V	V	Sin Origen
CP 5.2	Escenario 5	V	V	V	V	V	V	V	Origen correcto
CP 6.1	Escenario 6	V	V	V	V	"	V	V	Descripción incorrecta
CP 6.2	Escenario 6	V	V	V	V	"Mandarina"	V	V	Descripción incorrecta
CP 6.2	Escenario 6	V	V	V	V	"Mandarinas muy ricas"	V	V	Descripción correcta
CP 7.1	Escenario 7	V	V	V	V	V	NV	V	Seleccionar Materiales
CP 7.2	Escenario 7	V	V	V	V	V	V	V	Materiales correctos
CP 8.1	Escenario 8	V	V	V	V	V	V	NV	Envases incorrectos
CP 8.2	Escenario 8	V	V	V	V	V	V	V	Envases correctos
CP 9.1	Escenario 9	V	90	V	V	V	V	Peso: 50 Peso: 20	No hay envases que soporten el peso del producto
CP 9.2	Escenario 9	V	90	V	V	V	V	Peso: 89 Peso: 20	No hay envases que soporten el peso del producto
CP 9.3	Escenario 9	V	20	V	V	V	V	Peso: 89 Peso: 20	Escenario correcto

Capturas:
Curso normal:



Curso alternativo:



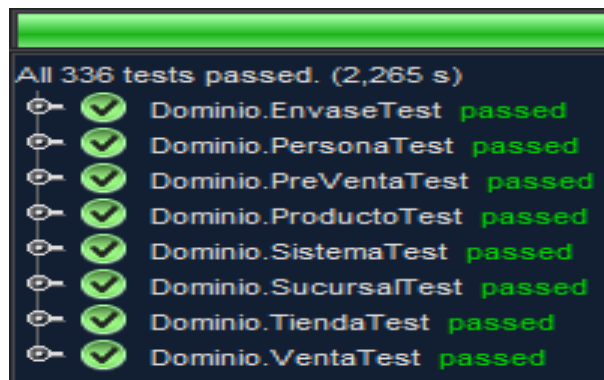
4.3. Sesiones de ejecución de pruebas

16 hs, 10/11/2019 Prueba completa Backend.1:

Se realizaron las pruebas del BackEnd completo utilizando las pruebas unitarias.

12:30 hs, 19/11/2019 Prueba completa Backend.2:

Se realizaron las pruebas del BackEnd completo utilizando las pruebas unitarias. Para esta prueba, se modificaron los JUnit, en correspondencia con los cambios que se realizaron en backend.



10:30 hs, 25/11/2019 Prueba exploratoria de la interfaz:

Se realizó una prueba exploratoria completa de la interfaz de usuario. Se adjunta el link del video debajo.

11:00 hs, 25/11/2019 Realizada por Martín Gutman

Prueba exploratoria de las funciones del cliente

En este caso se realizo en el sistema operativo Windows 10.

Se adjunta un link al video correspondiente:<https://youtu.be/QqUh5LVIZFw>

11:00 hs, 25/11/2019 Realizada por Joaquin Lamela

En este caso se realizo en el sistema operativo Windows 10.

Prueba exploratoria de las funciones del vendedor

Se adjunta un link al video correspondiente:<https://bit.ly/2QOvefB>

5. Reporte de Defectos

5.1. Detección y documentación de defectos

Una de las formas de detectar un defecto en el sistema es siguiendo el estándar que propone Mozilla.

Los lineamientos a seguir que se proponen son los siguientes.

Preliminares:

1. Asegurarse de que el software se encuentre actualizado: Realizar pruebas en desarrollo para ver si el bug ya ha sido solucionado. Si se produce de manera ocasionales, pero no después de seguir pasos específicos. Por último, en caso de que no se pueda reproducir el problema, es probable que no se reporte, excepto que se provea información a cerca de lo ocurrido [4].
2. Búsqueda en reportes anteriores, para ver si el bug ya había sido reportado [4].
3. Reportar el bug con el uso de un formulario específico [4].

Pasos para reproducir un bug:

Los pasos para reproducir un bug son la parte más importante de cualquier reporte. Si el desarrollador está listo para reproducir el bug, el bug es muy probable que sea corregido. Si este paso no está tan claro, no será posible saber si el bug ha sido corregido.

Describe tu método de interacción con Firefox además de la atención de cada paso.

- Impreciso: Abrir Gmail en otra ventana.
- Preciso: Presiona Cmd+N para abrir una nueva ventana en el navegador, luego escriba <https://mail.google.com/> en la barra de direcciones y presione Enter.

Después de tus pasos, describe con precisión los resultados observados y el resultado esperado. Hechos claramente separados (observaciones) de especulaciones.

- Impreciso: Esto no trabaja
- Preciso: En lugar de mostrar mi bandeja de entrada, se muestra el mensaje 'Tu navegador no soporta cookies (error -91)'.

Si el bug parece enorme, podría ser algo inusual en la configuración necesaria de los pasos para reproducir el bug. Ver si puedes reproducir el bug en un nuevo perfil Firefox. Si el bug solo sucede en tu perfil existente, trata de averiguar que ajustes, extensiones, o archivos en tu perfil se necesitan para reproducir el bug.

Resumen claro del bug:

Un buen resumen podría rápida y únicamente identificar un reporte. Esto debe exponer el problema, no tu solución sugerida [4].

- Bien: Cancelando el diálogo de copia de archivo falla el Administrador de Archivos [5]
- Mal: Falla en el software [5].
- Bien: El desplazamiento con la flecha abajo no trabaja en `textarea` con estilo `overflow:hidden` [5].
- Mal: El navegador debería trabajar con mi sitio web [5].

5.2. Definición de categorías de defectos

El estándar ya visto de Mozilla, también incluye varios tipos de bugs, que los categorizan según varios parámetros encontrados en la figura de debajo [6].

▼ Detailed Bug Information Narrow results by the following fields: Comments, URL, Whiteboard, Keywords, Bug Numbers, Version, Target Milestone, Type, Severity, Priority, Hardware, OS

Comment: contains all of the strings ☐ Description (initial comment) only

URL: contains all of the strings

Whiteboard: contains all of the strings

Keywords: contains all of the words

Bugs numbered should be only included in the results
(comma-separated list)

Only bugs with at least: votes

Version: 0.7 0.9 1.0 1.0 Branch 1.1 1.4 Branch 1.5 1.5.0.x Branch

Target Milestone: 0.2 0.2a1 0.2b1 0.2b2 0.3 0.3a1 0.3a2 0.3b1 0.4

Type: defect enhancement task

Severity: blocker critical major normal minor trivial enhancement

Priority: P1 P2 P3 P4 P5 --

Hardware: Unspecified All Desktop ARM ARM64 DEC HP PowerPC

OS: Unspecified All Windows Windows 95 Windows 98 Windows ME Windows NT Windows 2000

Figura 5.1: Reporte de bugs de Mozilla [6].

Entre ellos se deben destacar las categorías que aplican a nuestro programa:

Tipo del defecto:

1. Defecto: Regresión, crasheo, causa de colgamiento, vulnerabilidad de seguridad y otro problemas reportados [7].
2. Mejora: nuevas funcionalidades, mejoras en interfaz, etc. Y cualquier otro pedido por los usuarios para la mejora del producto [7].
3. Tarea: Refactorear, remover, reemplazar, habilitar, deshabilitar funcionalidades y cualquier otra tarea de ingeniería [7].

Severidad: Indica que tan severo es el problema, desde un "blocker", que deja a la aplicación en desuso, hasta un error trivial, que indica un problema con menor relevancia [5].

Prioridad: Marca la prioridad de los bugs, la marcan los managers de los proyectos, o alguien con la postestad y el conocimiento para saber como priorizar [5].

5.3. Defectos encontrados por iteración

Los defectos que fueron encontrados son los siguientes:

1. Error encontrado en el repositorio:

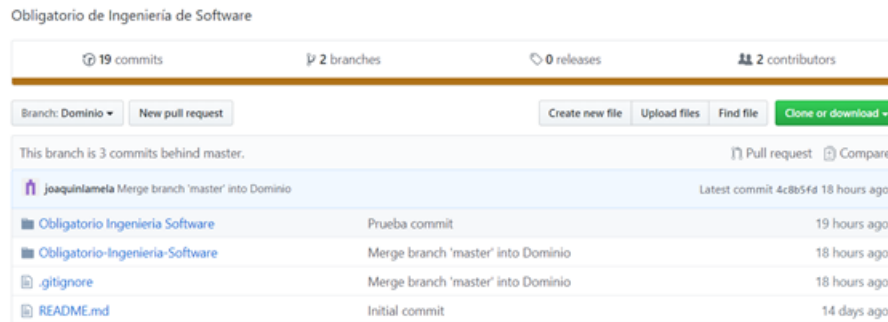


Figura 5.2: Bug reportado 1

Descripción: Se reportó un problema en el repositorio, al intentar fusionar dos branches. Dichas Branches eran Dominio, una branch auxiliar que se utilizaba para subir códigos prematuros. En cuanto master era la branch principal. Luego de una modificación de clases iguales en las dos branches, se intentó hacer el merge, y se detectó un conflicto al intentar realizar dicha acción. Ese conflicto fue solucionado de manera errónea, y se hizo el merge igualmente. Una vez realizado esto, se fusionaron las branches, y las clases quedaron de una manera extraña como se nota en la fotografía. Se creó un archivo, en el cual, la carpeta parecía haberse duplicado. Lo que había sucedido es que las clases se habían dividido en los dos archivos mostrados (Figura 1).

Tipo: Se reportó como un defecto, debido a que se podía diagnosticar como un problema de regresión, lo cual lo incluye en la categoría de defectos.

Seriedad: Alta

Solución: Se creó un nuevo repositorio, y se subió el último respaldo que se tenía del proyecto.

Imágenes

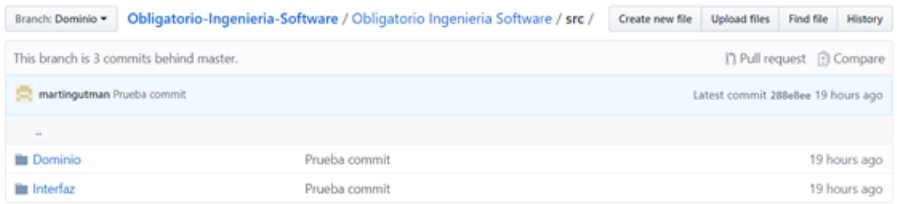


Figura 5.3: Bug reportado 1

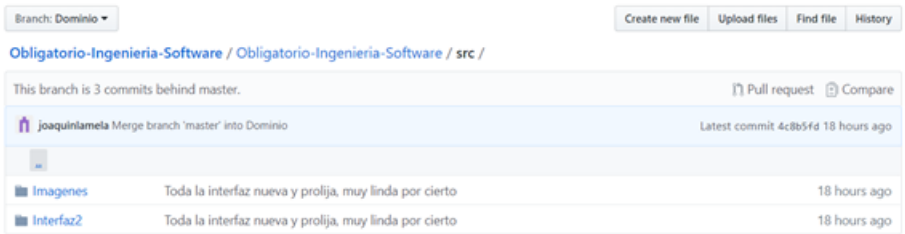


Figura 5.4: Bug reportado 1

2. Error encontrado luego del registro de la compra:

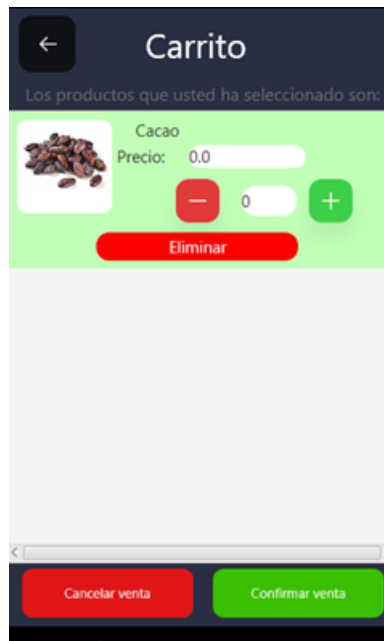


Figura 5.5: Bug reportado 2

Descripción: Luego del registro de la venta de más de un elemento, al entrar luego al carrito, aparece un elemento, con 0 unidades, y precio 0.

Tipo: Se reportó como un defecto, debido a que se podía diagnosticar como un problema de crasheo, ya que al quedar en esa instancia, se provocaba que al intentar registrar otra venta, se caía el programa.

Seriedad: Baja

Solución: Se verificaron los métodos en FrontEnd y en BackEnd, este error ya fue solucionado.

Figuras:

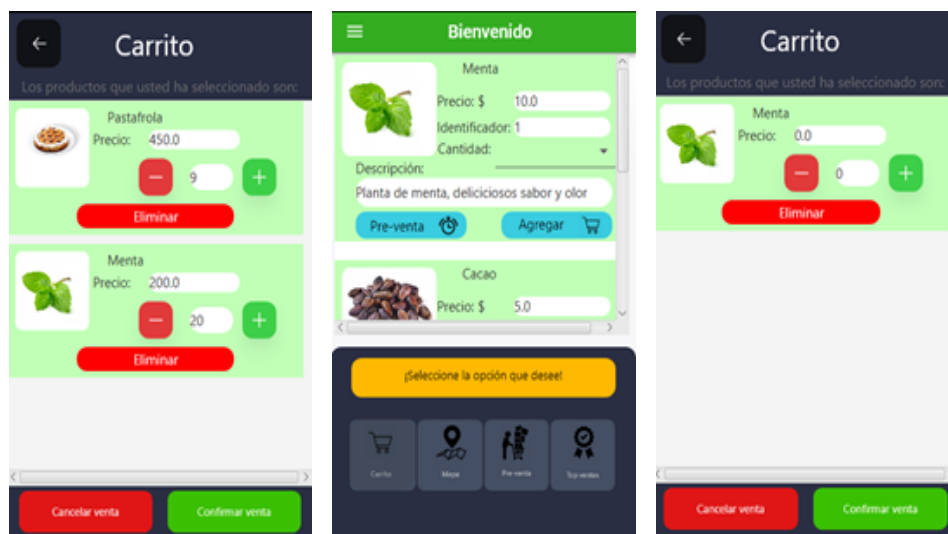
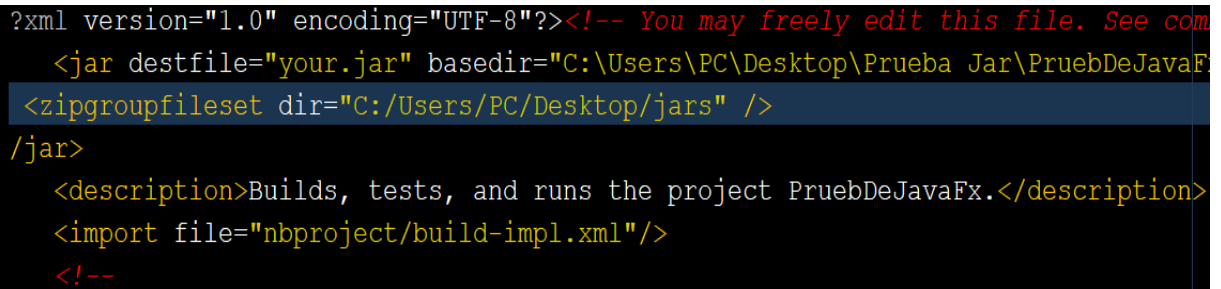


Figura 5.6: Error en carrito

3. Error encontrado en la creación del archivo .jar:

A screenshot of a text editor showing an XML file. The XML is a manifest for a Java JAR file. It starts with a declaration: `?xml version="1.0" encoding="UTF-8"?><!-- You may freely edit this file. See com`. Then it has a `<jar destfile="your.jar" basedir="C:\Users\PC\Desktop\Prueba Jar\PruebDeJavaF` tag. Below that is a `<zipgroupfileset dir="C:/Users/PC/Desktop/jars" />` tag. Then a `/jar>` closing tag. Next is a `<description>Builds, tests, and runs the project PruebDeJavaFx.</description>` tag. Then an `<import file="nbproject/build-impl.xml"/>` tag. Finally, it ends with `<!--`.

```
?xml version="1.0" encoding="UTF-8"?><!-- You may freely edit this file. See com
<jar destfile="your.jar" basedir="C:\Users\PC\Desktop\Prueba Jar\PruebDeJavaF
<zipgroupfileset dir="C:/Users/PC/Desktop/jars" />
/jar>
<description>Builds, tests, and runs the project PruebDeJavaFx.</description>
<import file="nbproject/build-impl.xml"/>
<!--
```

Figura 5.7: Error en la creación del Jar

Descripción: Se reportó un problema en el proyecto. Al intentar crear un archivo con el formato .jar, para ser ejecutado externamente, si necesidad de abrir el proyecto. Este problema estuvo arraigado a la inclusión de las librerías JPhoenix, y las imágenes, para que la aplicación fuese totalmente portable. Esto no fue posible, ya que la creación de este tipo de ejecutable con java fx presenta una dificultad superlativa, y no se pudo llegar a realizar. Para intentar realizar esta operación, se dedicaron más de 8 horas, y no se logró realizarlo.

Tipo: Se reportó como un defecto, debido a que se podía diagnosticar como un problema en el cual no hubo posibilidad de implementación.

Seriedad: Media

Solución: No se ha encontrado solución que mejore totalmente la portabilidad, por el momento, el .jar se puede ejecutar junto a una carpeta lib, que contiene las librerías y una carpeta que contiene las imágenes.

4. Error encontrado al correr el mapa en las computadoras de la facultad (dentro del jar):

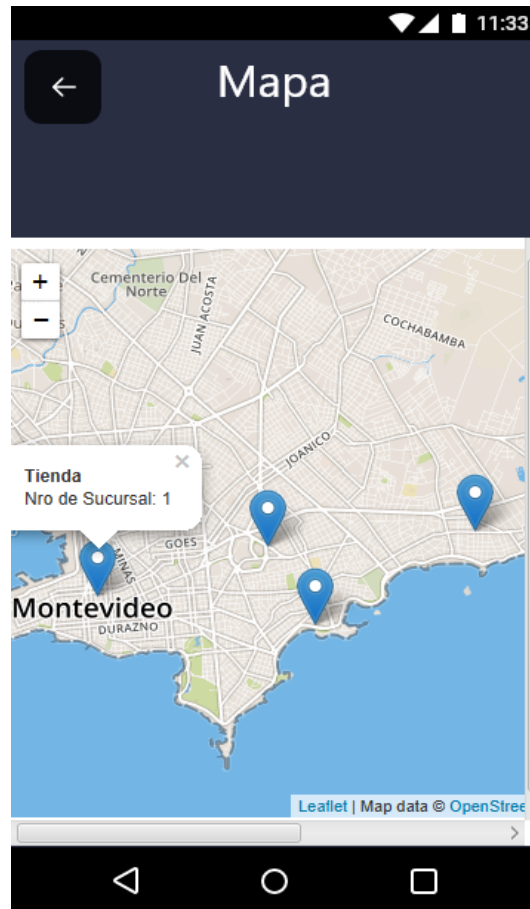


Figura 5.8: Error en el cargado de mapa

Descripción: Se reportó un problema en el proyecto. Al intentar correrlo en las computadoras de la facultad, dentro del jar generado, al abrir la pestaña de mapa, el mismo no logra ser cargado.

Tipo: Se reportó como un defecto, debido a que se podía diagnosticar como un problema en el cual no hubo posibilidad de implementación ni mejora.

Seriedad: Baja

Solución: No se ha encontrado solución para este problema.

5.4. Estado de calidad global

En esta sección se hablara acerca del estado global tanto de la interfaz del proyecto como la calidad de la codificación. De forma que el primer punto que se destaca son las diferentes funcionalidades con las que cuenta el proyecto. De manera que dentro de ellas las funcionalidades con las que cuenta la aplicación en este caso son, la elección entre el perfil vendedor o cliente. Luego a partir de la pestaña de la pestaña elegida, las diferentes funcionalidades que se presentan:

Funcionalidades para vendedor:

- Agregado de producto: se puede agregar un producto al sistema, incluyendo fotos.
- Agregado de sucursal: se puede agregar una sucursal al sistema.
- Agregado de envase: se puede agregar un envase a la lista de envases disponibles del sistema, incluyendo fotos de los mismos.
- Eliminación de productos: se puede eliminar los productos disponibles en el sistema
- Top ventas: se puede visualizar los productos mas vendidos del sistema
- Ventas por mes: el vendedor puede visualizar seleccionando un mes correspondiente, las ventas que se han realizado en dicho mes.
- Estadísticas: en este caso se muestran en forma de gráficas de barras, las estadísticas de ventas realizadas por mes, envases reutilizados, productos vendidos y además las ganancias que ha obtenido la empresa.

Funcionalidades para cliente:

- Carrito: el cliente puede visualizar el carrito con los productos que ha seleccionado para agregar al carrito y desde allí confirmar la venta. De manera que la diferencias entre venta y pre-venta es que en el caso de la venta, los productos se le envían a la dirección solicitada.
- Mapa: el cliente puede visualizar las localizaciones de las sucursales, de manera que puede visualizar donde se encuentran los productos para poder ir a levantar los productos.
- Pre-Venta: el cliente puede visualizar los productos que ha agregado a la pre-venta, de manera que luego clickee el botón de pre-venta pueda visualizar los productos agregados a la preventa.
- Top ventas: el cliente puede visualizar los productos mas vendidos del sistema

Por otra parte otro punto a destacar, es el diseño de la interfaz. En este caso lo que sucede con el diseño, es que antes de comenzar a realizar el desarrollo de la aplicación se pre-diseño en la aplicación Adobe XD. De manera que a la hora de realizar la codificación nos fue mas sencillo debido a que teníamos un diseño ya realizado. Esto nos facilito a la hora de realizar el diseño en Scene Builder y nos ahorro tiempo. Además de que también nos brinda la posibilidad de realizar una interfaz de mayor prolijidad, brindando de manera mas sencilla una gran paleta de colores y distintos componentes los cuales facilita la usabilidad a los usuarios. Una gran critica a la interfaz es la gran variedad de colores utilizados dentro de la misma.

De forma que es un diseño minimalista, innovador, llamativo. Pero como nombrábamos anteriormente, la paleta de colores no es exactamente la adecuada para la idea de un EchoShop, el reuso y el reciclaje, de manera que a futuro una nueva versión a actualizar seria realizar un nuevo diseño de interfaz, la cual contenga una paleta de colores adecuada a dichas ideas. En este caso los colores ideales serían un verde claro y un anaranjado.

Finalizando con los comentarios acerca de la interfaz y las funcionalidades lo que se trato de hacer con este proyecto fue seguir con las Heurísticas de Nielsen, las cuales son las mencionadas anteriormente.

Para continuar, hablaremos de la calidad del código. De manera que podemos decir que el primer punto con el que cumple el obligatorio es que no contiene la repetición de código. Lo cual produce que la lectura del obligatorio se realice de manera directa, como si se estuviera leyendo una escritura. Este punto claramente se habla en la guía de Clean Code, de manera que .allí se dice que el código duplicado hace que nuestro código sea más difícil de mantener y comprender además de generar posibles inconsistencias” [8].

Otro punto a destacar de nuestra codificación es el principio de ”Least Surprise” [8] de manera que las funciones o clases deben hacer lo que (razonablemente) se espera de ellas. Es decir, una función o una clase debe tener, en función de su nombre, un comportamiento obvio para cualquier programador, sin que este tenga la necesidad de sumergirse en su código. Por lo cual a lo hora de la codificación fue donde hicimos mas énfasis de manera, que cualquiera que leyera el código entendiera con tan solo leer el nombre que debería hacer dicho método.

Por otra parte una regla esencial, es aquella que esta basada en las reglas de los Scout. ”deja el campamento más limpio de como te lo encontraste” [8]. Destacándose el cumplimiento de esta regla tan esencial. Esto debido a que en primera parte facilita la lectura, la prevención de errores y el arreglo de los mismos. Una ilustración de esto, es el caso de que muchas veces, revisando código, nos encontramos con que el nombre de una variable no es demasiado intuitivo o con un fragmento de código duplicado. Resolviendo este tipo de situaciones, estaremos aplicando la regla de los Scouts.

Por otro lado también, en el caso del obligatorio necesariamente debíamos cumplir la regla de FIRST. De manera que esta regla se basa en los test del proyecto. De manera que no es suficiente con escribirlos, sino que deben cumplir determinadas reglas:

- **Fast:** los test deben correr rápido. Deben tardar poco en ejecutarse. El no ejecutar los test con relativa frecuencia puede hacer que tardemos más en encontrar los problemas que vayan surgiendo.
- **Independent:** los test deben ser independientes unos de otros. El resultado de un test no debe condicionar el de los siguientes. Deben poder ejecutarse en el orden que se quiera. De lo contrario, un fallo en el primer test, puede desencadenar un fallo en cascada de los demás, haciendo complejo el diagnóstico del sistema.
- **Repeatable:** los test deben poder ejecutarse en cualquier entorno (desarrollo, pre-producción, producción).
- **Self-Validating:** los test deben devolver una respuesta booleana.
- **Timely:** los test deben ser escritos antes que el código de producción. De no ser así, el código de producción será difícil de testear. Aunque en este proyecto, esto no fue necesario debido a que generaría una complejidad extra.

Otro principio con el cual se cumple del libro de Clean Code, básicamente es el principio de responsabilidad única, el cual nos dice que: «una clase debe tener una y solo una razón para cambiar». Debe tener una única responsabilidad.

En forma de critica constructiva uno de los principios que debería seguir el proyecto, es aquel que esta basado en Open/Closed Principle. El cual nos dice que: «una clase debe estar abierta a extensiones pero cerrada a modificaciones». O lo que es lo mismo, el comportamiento de dicha clase debe ser alterado sin tener que modificar su código fuente. De lo contrario podría desencadenar efectos colaterales[9].

6. Reflexión

A modo de resumen y reflexión acerca de este proyecto. Se debe destacar en primera instancia que fue un proyecto bastante abierto y que se abarcó de una manera ambiciosa, el cual incluyó una fase previa de aprendizaje de un nuevo lenguaje JavaFx. Cuya complejidad es alta, pero a un costo de una interfaz mucho más usable, ya sea en cuanto a eficiencia, como en cuanto a visibilidad.

Esto hizo que el proyecto tuviera una demanda de tiempo importante para ser finalizado en tiempo y forma, de manera de que gracias a la constancia en el trabajo, se pudieron alcanzar los objetivos y construir la anhelada aplicación.

La gratificación que se tuvo realizar y poder terminar este proyecto, fue alta, ya que, como se explica anteriormente, la dificultad también era muy importante.

En cuanto a la construcción en general, sentimos que pasamos por todas las etapas de la construcción y creación de un software. Desde relevamiento de requerimientos, por medio de la consigna del obligatorio especificada, el diseño de la aplicación por medio de UML, y diagramas de caso de uso. Además de la codificación, edificación y el probado del sistema, además de la mejora en la calidad del código, que constituyeron la construcción de un código muy usable, estable, entendible, y desde nuestro punto de vista, muy funcional.

Otro punto a destacar, dentro del desarrollo del sistema, fue el desafío de comenzar a aprender un lenguaje de programación desconocido hasta el momento por los miembros del equipo de desarrollo. En este caso, se habla de JavaFX, el cual fue fundamental para el desarrollo de todo el proyecto, dando un gran cambio visualmente. Mejorando de manera drástica con respecto a Java Swing, brindándonos la posibilidad de desarrollar una interfaz de excelencia y prolijidad.

Destacándose también, dentro del desarrollo y codificación de la aplicación, el uso de programación defensiva. Se hace énfasis en este punto, debido a que no era algo solicitado por parte de la letra del proyecto, por lo cual creemos que haberlo realizado también aporta a la hora de la usabilidad, de manera que si un usuario realiza una acción incorrecta tenga un feedback correspondiente por parte de la aplicación. Es decir, mostrarle que ha realizado correctamente una acción o que la ha realizado de manera incorrecta. Es por esto que creemos que es un gran aporte para el desarrollo de la misma, de manera que es un punto extra a destacar dentro del sistema.

Por último, se destaca que se pudieron abarcar todos los conceptos dados en el curso para la creación de el software, y se pudo seguir (en caso de que se aplicara al obligatorio), las recomendaciones que se nos fueron brindando en el estudio de cada etapa de creación de un programa.

Bibliografía

- [1] B. W. Kernighan and D. M. Ritchie, *El lenguaje de programación C*. Pearson Educación, 1991.
- [2] Oracle. (1997) jccOracle. [Online]. Available: <https://www.oracle.com/technetwork/java/codeconventions-150003.pdf>
- [3] J. Nielsen, “10 usability heuristics for user interface design,” *Nielsen Norman Group*, vol. 1, no. 1, 1995.
- [4] T. Dhaliwal, F. Khomh, and Y. Zou, “Classifying field crash reports for fixing bugs: A case study of mozilla firefox,” in *2011 27th IEEE International Conference on Software Maintenance (ICSM)*. IEEE, 2011, pp. 333–342.
- [5] MDN Contributors. (2019) ReporteDefectos. [Online]. Available: https://developer.mozilla.org/es/docs/QA/Bug_writing_guidelines
- [6] Mozilla Firefox. (2019) ErroresMozilla. [Online]. Available: <https://bugzilla.mozilla.org/home>
- [7] ——. (2019) ErroresMozilla2. [Online]. Available: <https://mozilla.github.io/bug-handling/task-defect-enhancement>
- [8] R. C. Martin, *Clean code: a handbook of agile software craftsmanship*. Pearson Education, 2009.
- [9] ———, *Clean Code-Refactoring, Patterns, Testen und Techniken für sauberen Code: Deutsche Ausgabe*. MITP-Verlags GmbH & Co. KG, 2013.
- [10] R. S. Pressman, *Software Engineering: A Practitioner’s Approach*, eight ed. McGraw-Hill, 2014.
- [11] I. Sommerville, *Software Engineering*, 10th ed. Pearson, 2015.
- [12] Universidad ORT Uruguay. (2013) Documento 302 - Facultad de Ingeniería. [Online]. Available: <http://www.ort.edu.uy/fi/pdf/documento302facultaddeingenieria.pdf>