



Universidad de Colima

Bachillerato Técnico #16

Materia:

Programación IV

Evidencia:

Proyecto. Ferretería FerreCode

Integrantes del Equipo:

MICHEL MAGAÑA EDUARDO

RODRÍGUEZ ROBLES MATEO

MARMOLEDO JIMÉNEZ DAGOBERTO

PADILLA RUIZ ANA FERNANDA

CUEVA PABLO EMILIANO

ESPINOSA MAYA LEONARDO

Docente:

Ponce Suárez Hugo César

Fecha de Entrega:

Miércoles 28 de Mayo de 2025

Índice (Tabla de Contenidos)

1. Introducción.....	1
2. Análisis del problema.....	2
3. Detalles del proceso de programación.....	4
4. Roles de los integrantes del equipo.....	13
5. Descripción de los elementos del programa.....	14
6. Problemas experimentados y soluciones.....	38
7. Conclusiones (Individuales).....	39
● Conclusión Eduardo Michel.....	39
● Conclusión Mateo Rodríguez.....	40
● Conclusión Dagoberto Marmolejo.....	40
● Conclusión Ana Fernanda Padilla.....	41
● Conclusión Emiliano Cueva.....	41
● Conclusión Leonardo Espinosa.....	42

1. Introducción

El documento de reporte que se presentará a continuación describe a detalle cómo fue el proceso de implementación del sistema “Ferre - Code”, que fue diseñado por nuestro equipo como proyecto final de programación de sexto semestre.

Este sistema es la aplicación de una tienda de ferretería llamada FerreCode, que incluye todo lo necesario para el funcionamiento de esta como un punto de venta: un catálogo de usuarios y opciones para gestionar y vender el inventario que se tiene en la tienda.

Este programa se desarrolló con el objetivo de aplicar los conocimientos adquiridos por los estudiantes durante todo el semestre respecto al lenguaje de programación Java, al entorno eclipse id, y la “programación Orientada a Objetos”.

En este documento trataremos todo el proceso de planeación y codificación de nuestro programa. Hablaremos de todos los pasos que seguimos para llegar a una solución que cumpla con los requerimientos solicitados por el profesor. En el apartado de Análisis del Problema describiremos los requerimientos del programa y nuestro proceso de investigación y toma de decisiones para establecer los componentes que debía contener la aplicación de ferretería.

Posteriormente, se describirán los detalles del proceso de programación. En esta sección hablaremos de manera general sobre los pasos para la creación del proyecto. Además, presentaremos las herramientas y conocimientos nuevos que después de investigación y comprensión, decidimos implementar en la solución de nuestro proyecto para mejorarlo, tanto visualmente como en términos de optimización del código.

En el siguiente apartado (Descripción de los elementos del programa), explicaremos cada una de las partes y ventanas que componen nuestro proyecto, su funcionamiento y los métodos que se utilizaron en cada uno.

A manera de retroalimentación y futuras referencias, discutiremos algunos de los problemas enfrentados durante el proceso de programación, es decir, errores que tuvimos que resolver al crear el sistema. Estos problemas pueden referirse tanto a errores de compilación, dificultades en el front del programa o funcionalidades de la aplicación con las que tuvimos más problemas.

Finalmente, cada uno de los integrantes de nuestro equipo proporcionará una conclusión personal con lo aprendido durante la realización del proyecto: su experiencia al trabajar en equipo, los conocimientos nuevos adquiridos y la importancia que tuvo para ellos este proyecto de programación.

Esperamos que este documento sea de interés.

2. Análisis del problema

Cuando se presentó el proyecto se planteó ante nosotros la siguiente problemática: una tienda de Ferretería que requería el diseño y desarrolló de un programa en Java.

Los requisitos del programa fueron los siguientes. Debía contener Catálogo de Usuarios, Buscar, Agregar, Modificar, Eliminar y opciones necesarias para vender:

Después de analizar la problemática en equipo, llegamos a que nuestra aplicación contendría los siguientes apartados.

- **Catálogo de Usuarios:** Como para acceder al programa se requiere de un Inicio de Sesión, se necesita en la aplicación un apartado para gestionar los usuarios operativos del sistema. Se debe ofrecer Alta, Baja y Modificación de un Usuario.
- **Inventario:** Como se trata de una tienda, un apartado para que el usuario operativo que hace uso del sistema vea cuántos ejemplares se tienen en existencia y añada más si el proveedor trae productos y el usuario operativo tiene el permiso. Permite consultar la información de un producto.
- **Captura / Agregar Producto:** Un subapartado del apartado de Productos para que el usuario operativo pueda ingresar toda la información de un producto y este se agregue a la tienda.
- **Modificar Producto:** Un subapartado del apartado de Productos para que el usuario busque un producto y tenga la opción de editar los valores de los campos, es decir, la información del producto.
- **Eliminar Producto:** Un subapartado del apartado de Productos para que el usuario busque un producto y tenga la opción para eliminarlo de la tienda.
- **Vender Producto:** Funcionalidad principal de la aplicación. Permite que un usuario operativo realice una venta a un cliente. Esto lo hará mediante la búsqueda de productos y la adición de un carrito de compras.

Además, la aplicación contendrá apartados iniciales de bienvenida, donde se mostrará el logo de la aplicación FerreCode y otro de inicio de sesión, donde el usuario que quiera ingresar sesión deberá proporcionar al sistema su usuario y contraseña. Si el usuario comete 3 errores seguidos, el sistema se cerrará en automático para su seguridad.

Otros requisitos adicionales:

- **Permisos:** en los requerimientos del programa se menciona la creación de dos permisos. Admin y Cajero. El administrador podrá tener acceso a todos los apartados del sistema, mientras que el cajero únicamente podrá acceder a la aplicación con modo Lectura, es decir, solo podrá acceder a Vender y Buscar. Detallaremos nuestra manera de atacar el requisito.
- **Imágenes:** cuando se muestre la información de un producto seleccionado, se debe mostrar la imagen de este.

Sobre los Tipos de Permisos

Al analizar el problema, nos percatamos de que manejar únicamente dos tipos de permisos podría conllevar algunos problemas en la aplicación. Si solo existe un administrador, la carga sobre este sería gigante y más en tiendas de gran alcance, ya que por los requisitos este es el único usuario con permiso para Gestionar productos.

Por lo tanto, debería haber más de un administrador. Sin embargo, esto conlleva algunas complicaciones. Si todos estos administradores comparten el mismo permiso total, hay riesgos en el sistema y la seguridad. La toma de decisiones cruciales no sería tan clara al haber más de una persona con la misma autoridad total. Además, la ausencia de un único usuario con permiso total hace confusa la implementación del catálogo de usuarios. Un administrador no debería poder modificar o eliminar a otro administrador. Pero si no lo hace, la estancia de los administradores en el sistema es permanente a menos que se habilite un autoeliminado, el cual también traería errores.

Después de hacer este análisis, concluimos que lo mejor para nuestro programa sería implementar un sistema de roles con los siguientes tres roles: **Cajero, Admin y Propietario**. Las funciones y permisos de cada uno se describen a continuación.

- **Propietario:** Dueño único y gestor de la tienda de autopartes. Tiene permiso total en el sistema. Puede agregar, modificar y eliminar a todos los usuarios operativos. Su única restricción es la modificación de su permiso o eliminarse a sí mismo. En nuestra aplicación se tomó la decisión de que en todo momento existiera un propietario. Si se desea cambiar al propietario es suficiente con modificar la información del propietario en el apartado correspondiente.
- **Admin:** Tiene acceso a todos los apartados del sistema. Sin embargo, tiene restricciones en el apartado de Catálogo de Usuarios. Únicamente puede modificar su información y la de los cajeros. No puede modificar permisos y únicamente puede dar de alta cajeros.
- **Cajero:** Únicamente tiene acceso al sistema a manera de Lectura. Solo puede acceder a Vender y Buscar. El cajero no puede acceder al catálogo de usuarios.

3. Detalles del proceso de programación

Para programar este sistema en lenguaje Java utilizamos el Entorno de Desarrollo Integrado (IDE) de Eclipse. Esto debido a que es el Software con el que hemos estado trabajando durante todo el semestre y consideramos que nuestras habilidades en el manejo de Visual y en el conocimiento del lenguaje Java son buenas. Para la conexión a una base de datos utilizamos Access como sistema gestor de base de datos, por las mismas razones.

Herramientas adquiridas y utilizados por el equipo

- Bloques **try-catch**

Durante las clases que tuvimos durante el semestre aprendimos que cuando una aplicación en Java interactúa con una base de datos es necesario establecer una conexión cada vez que se quiera consultar, insertar, eliminar o actualizar registros de una tabla.

Para el caso de bases de datos Access, aprendimos a declarar una cadena de conexión (**ruta**), cargar el controlador correspondiente (**UcanaccessDriver**) mediante **Class.forName()** y establecer la conexión usando **DriverManager.getConnection()**.

Uno de los puntos más importantes que abordamos fue el uso del **bloque try-catch**, que es una estructura de control de excepciones, es decir un bloque de manejo de errores del lenguaje Java utilizada para manejar excepciones que pueden ocurrir durante la ejecución del programa. En nuestro sistema lo usamos en la función **abrirConexion()** para capturar cualquier error al intentar abrir una conexión con la base de datos, como fallos en el driver, errores en la ruta de acceso o problemas de permisos. Aquí está el código de la función para abrir una conexión.

```
public static void abrirConexion(){
    try {
        Class.forName("net.ucanaccess.jdbc.UcanaccessDriver");
        conn = DriverManager.getConnection(ruta);
    }
    catch(Exception e) {
        JOptionPane.showMessageDialog(null, "Ocurrió el error: "+ e);
    }
}
```

Gracias a esta estructura, conseguimos evitar que el programa se cierre inesperadamente, mejorar el control de flujo y depurar errores de forma más clara. Este manejo de errores se volvió una herramienta fundamental en el desarrollo de nuestra aplicación, especialmente al trabajar con operaciones como realizar ventas o gestionar productos que requieren conectarse a la base de datos.

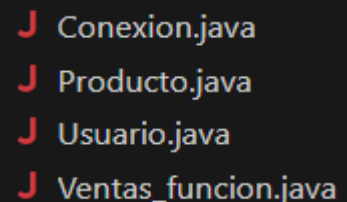
En nuestro programa, **ruta** es el string que contiene la cadena de conexión a la base de datos. Y **conn** es el objeto de la clase Connection para realizar la conexión.

Estos bloques los usa la función **abrirConexion()**, la cual la utilizamos en nuestro programa cada vez que se interactúa en una base de datos. También utilizamos estos bloques al trabajar con objetos ResultSet para consultar información de la base de datos.

- **Clases en Java, variables y métodos estáticos**

Al programar nuestra aplicación de Ferretería, utilizamos varias clases que contienen las funciones y variables que utiliza el sistema: **Conexión.java**, **Producto.java**, **Usuario.java** y **Ventas_funcion.java**. Podemos definir a una clase pública en Java como un elemento de código que contiene variables y métodos y que por su propiedad pública puede ser usado por cualquier otra clase o archivo del proyecto.

Utilizamos clases en lugar de programar en el código de cada Frame ya que esto nos permitió organizar mejor nuestro código. Al hacer esto, mantenemos todo el código que se relaciona en el mismo lugar. Fue una decisión que permitió tener nuestras líneas de programación organizadas y que sean fáciles de entender.



```
J Conexion.java
J Producto.java
J Usuario.java
J Ventas_funcion.java
```

Dentro de las clases, están variables y métodos estáticos. Una variable estática es aquella que mantiene su valor durante toda la ejecución del programa. Además, si se contiene en una clase (como es nuestro caso), decimos que la variable pertenece a la clase y no a un objeto, por lo que se puede acceder a ella desde cualquier parte del programa simplemente haciendo referencia a la clase y a la variable estática. Esto nos fue muy útil, ya que por ejemplo en **Conexion.java** declaramos una variable estática string con el nombre **Usuario**, en la que almacenamos el nombre del usuario que inicio sesión. Una vez hecho esto, si se quiere acceder al nombre de usuario que inicio sesión es suficiente con escribir **Conexion.Usuario**.

Funciona de manera similar con los métodos estáticos. Al ser un método estático en una clase pública podemos acceder a este desde cualquier parte del proyecto haciendo referencia a la clase. Por ser estático no se tienen que crear objetos de la clase para poder usarse, lo que ahorra mucha memoria y líneas de código.

El uso de clases nos permitió ahorrar recursos y reutilizar el código de funciones que se utilizarán en más de un formulario. Consideramos que fue una buena elección.

- **Estructura del Sistema y Herramientas Visuales**

Después de analizar la problemática, nos dimos cuenta de que la mejor manera de implementar el sistema era utilizando un JFrame principal inicial con el menú de opciones y un conjunto de instrucciones, y un JFrame por cada apartado en el menú que repita las opciones del menú principal solo que en JPanel del apartado se vean los controles relacionados a la funcionalidad de este apartado. Con esto, la **interfaz** de usuario queda **bien estructurada** y clara, lo que la hace sencilla y atractiva.

Este es un ejemplo de cómo se abriría la ventana del apartado específico al hacer click en el botón correspondiente en el menú principal inicial con instrucciones. Para este ejemplo al seleccionar Vender se abre el apartado de Ventas.

```
BtnVender_Menu.addActionListener(new ActionListener() {  
    public void actionPerformed(ActionEvent e) {  
        Ventas_visual ventas = new Ventas_visual();  
        ventas.setLocationRelativeTo(null);  
        ventas.setVisible(true);  
        dispose();  
    }  
});
```


Proceso de creación del sistema de Ferre - Code

1. Planeación

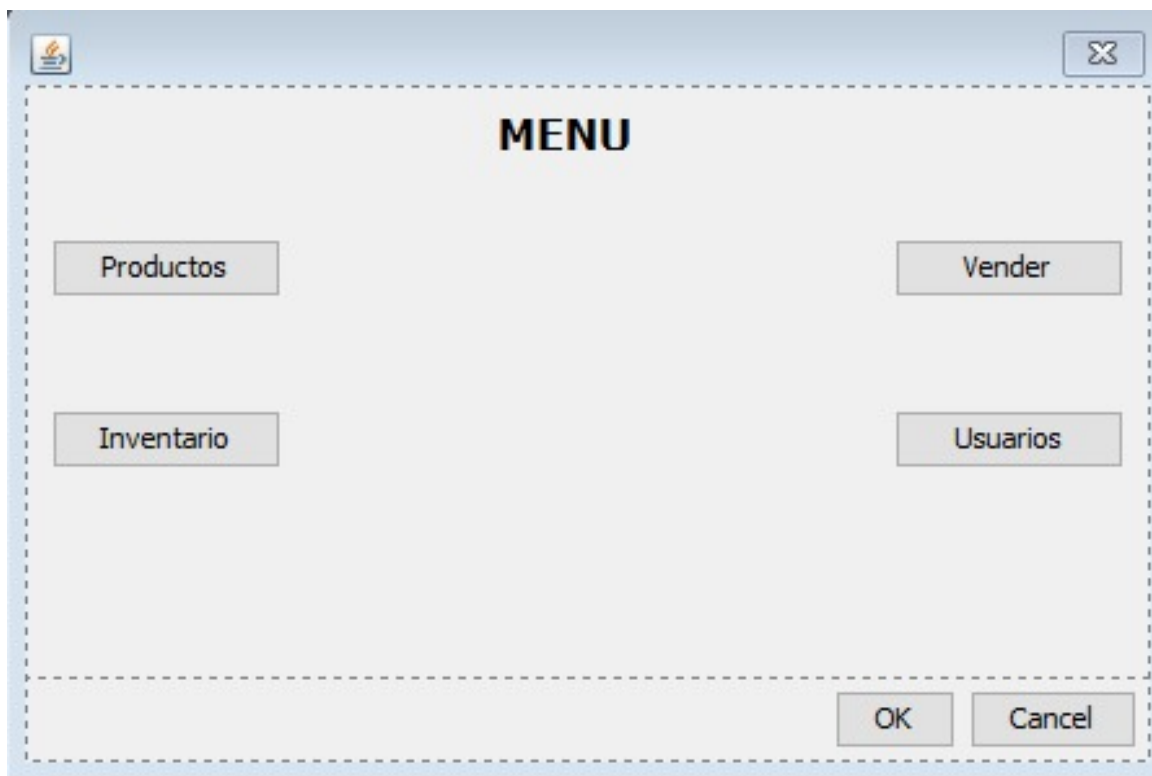
La primera parte consistió en la planeación de la aplicación, lo que incluye el bosquejo del diseño de la aplicación y la definición de las tablas y campos que tendrá la base de datos.

- **Planeación de la Aplicación**

Como se acaba de mencionar, después de analizar la problemática e investigar un poco sobre el tema, nuestro equipo decidió implementar una aplicación con una ventana principal que contenga un menú de opciones. Al presionar cualquier botón se carga la ventana correspondiente al apartado elegido.

Además, se debe tener un catálogo de usuarios. Después de discutir, determinamos que lo mejor era añadir como un botón en el menú principal la parte de “Catálogo de Usuarios”, botón que estará habilitado si el usuario es Propietario o Admin. También se planeó el diseño de los diferentes subapartados y apartados. A continuación mostramos algunos de los diseños preliminares de algunas ventanas.

Ventana Principal con Menú de Opciones



Ventana Apartado de Gestión de Productos

This window is titled "Ventana Apartado de Gestión de Productos". It features a sidebar on the left with a "Regresar" button at the top. Below it are four input fields labeled "Nombre producto", "Descripcion producto", "Proveedor Producto", and "Cantidad", each followed by a "<dynamic>" placeholder. A "Confirmar" button is positioned below these fields. The main area on the right contains a search bar labeled "Buscar..." with a dropdown arrow. Below the search bar are five buttons: "Alta", "Editar", "Baja", "Consultar", and "Consultar Todos". A large gray rectangular area displays the message "Producto no encontrado.". At the bottom right of the main area is a button labeled "Agregar Inventario".

Ventana Subapartado de Agregar Producto

This window is titled "AGREGAR PRODUCTOS". It contains several input fields for product information: "Nombre del Producto", "Descripcion del Producto", "Proveedor del producto", "Cantidad", and "Precio". To the right of these fields is a large, empty white rectangular area. At the bottom right of the window are two buttons labeled "OK" and "Cancel".

- **Planeación de la Base de Datos**

La siguiente parte fue planear la Base de Datos de nuestra aplicación. Primero establecimos el nombre de nuestra base de datos como **BDDS_Ferreteria**. Posteriormente establecimos las tablas, los campos y tipos de datos de acuerdo con los requerimientos de la aplicación. Estas son las tablas que creamos.

1. Tabla Usuarios_Operativos

La Base de Datos debe contener una tabla en la que se registren todos los usuarios operativos de la tienda FerreCode. Estos son los campos de la tabla Usuarios_Operativos.

- **Id (Autonumeración)**: Generar una clave única para cada usuario operativo.
- **Nombre_Completo (Texto corto)**: Para registrar el nombre completo del usuario
- **Usuario (Texto corto)**: Almacenar el nombre de usuario.
- **Password (Texto corto)**: Almacenar contraseña de cada usuario operativo.
- **Tipo (Texto corto)**: Para registrar el permiso del usuario. Este campo es controlado por nosotros, solamente permitiendo ingresar “Admin” o “Cajero”.

2. Tabla Productos

La principal tabla de la aplicación, la tabla de Productos. En esta tabla están contenidos todos los productos de la base de datos. Sobre esta tabla se llevan a cabo la mayoría de operaciones y consultas SQL. Estos son los campos de la tabla Productos.

- **Id (Autonumeración)**: Generar una clave única para cada producto de la tienda.
- **Nombre (Texto corto)**: Almacenar el nombre del producto en cuestión.
- **Descripcion (Texto largo)**: Guardar una descripción del producto (qué es).
- **Marca (Texto corto)**: Almacenar la marca del producto.
- **Precio (Número)**: Almacenar el costo por unidad de cada producto.
- **Cantidad_en_Stock (Número)**: Guardar la cantidad de ejemplares del producto.

3. Tabla Ventas

Considerado para nosotros como lo más importante de la aplicación, el apartado de Vender permite al usuario operativo hacer la venta de un producto a un cliente de la tienda. Estas ventas se guardarán en la tabla Ventas y sus campos son los siguientes:

- **IdVenta (Autonumeración)**: Generar una clave única para cada venta realizada.
- **Id_Usuario (Número)**: Guardar una clave única de usuario para cada venta.
- **Usuario (Texto largo)**: Registrar el usuario de la persona que hizo la venta.
- **Tipo (Texto corto)**: Guardar el permiso del usuario operativo que realizó la venta.
- **Id_Producto (Número)**: Almacenar el Id del producto que se vendió.

- **Nombre_Producto** (*Texto corto*): Guardar el nombre del producto que se vendió.
- **Precio_Producto** (*Número*): Para guardar el precio por unidad del producto.
- **Cantidad_Producto** (*Número*): Para la cantidad de unidades que se vendieron.
- **Total** (*Número*): Almacenar el total de la venta sin IVA.
- **Total_IVA** (*Número*): Almacenar el total de la venta con IVA.
- **Fecha_Hora** (*Fecha/Hora*): Almacenar la fecha y hora a la que se realizó la venta.

2. Desarrollo del Proyecto

Una vez definidos los roles en el equipo se comenzó con la elaboración del proyecto usando el lenguaje Java. Las partes de front y back se trabajaron en conjunto utilizando la herramienta de GitHub de trabajo colaborativo.

• GitHub

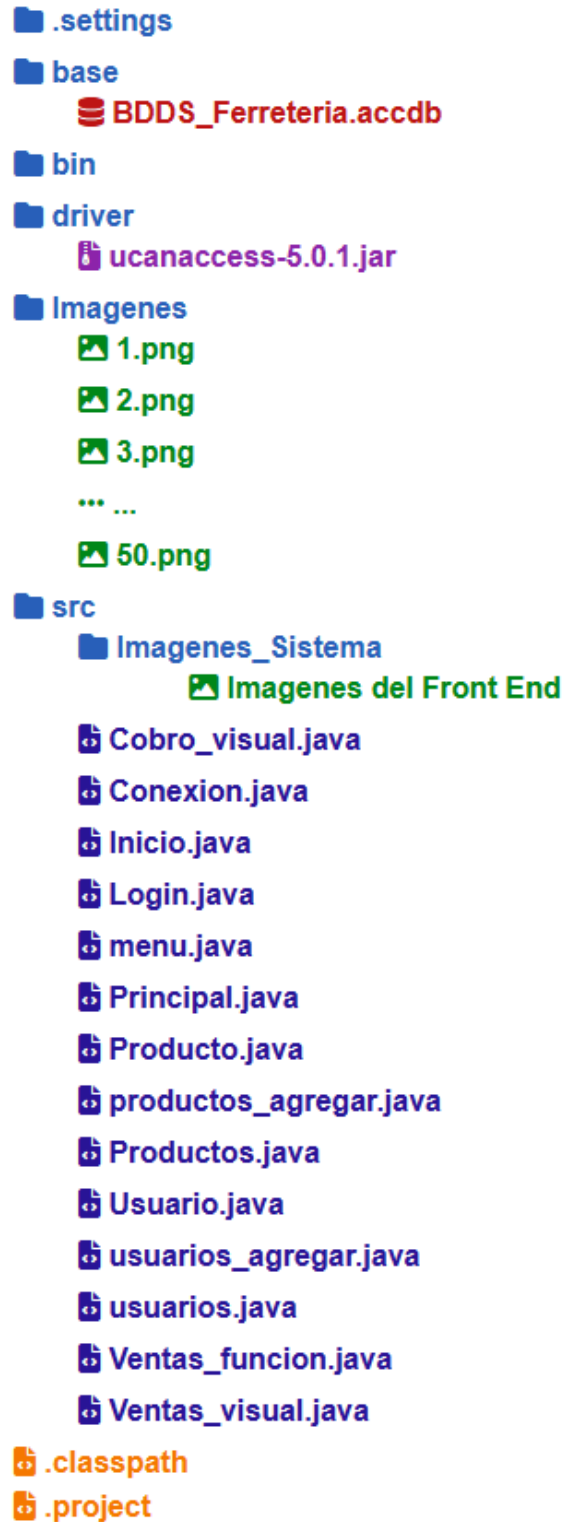
Decidimos utilizar GitHub para la realización de nuestro proyecto ya que es una herramienta de software que nos permite gestionar de manera práctica el código del proyecto y facilita que varias personas trabajen en equipo para una solución. Con GitHub pudimos hacer un seguimiento a todos los cambios que se realizaban en el proyecto, con lo que fue más sencillo encontrar y resolver errores.

Además, el tener una versión principal del proyecto (administrada por el jefe de equipo) permitió que todos los colaboradores del repositorio pudieran tener acceso a la versión más reciente del proyecto.



- **Consideraciones Adicionales**
- Diagrama de Árbol de la Carpeta con la Solución del Proyecto

Proyecto_Programacion_IV



3. Corrección de Errores, Validación y Mejora de Front

Por último, el equipo concentró sus esfuerzos en corregir los errores que se presentaban durante la programación de la aplicación de la Ferretería.

Además, se validó cada uno de los campos que puede ingresar un usuario operativo en todos los apartados. Después de investigar los caracteres permitidos para diferentes campos de texto y número hicimos las siguientes implementaciones.

- **Nombre_Completo:** solo permitimos ingresar letras, espacios y tecla backspace.
- **Usuario y Contraseña:** solo ingresar letras, números y caracteres especiales.
- **Nombre y Descripción de Producto:** letras, números y caracteres - / " (in) y .
- **Marca:** solo ingresar letras (ya sea mayúsculas o minúsculas).
- **Precio:** solo números enteros o decimales positivos.
- **Cantidad_en_Stock:** solo números enteros positivos.
- **Imagen:** ingreso obligatorio de una imagen en todo momento para cada producto.
- **DineroRecibo:** solo permitimos números enteros positivos.

En la mejora de front, para dar una mejor experiencia, alteramos las imágenes de los botones para generar versiones con mayor % de transparencia, para cuando estos botones se deshabiliten porque no se tengan permisos.

Finalmente continuamos con la prueba del sistema hasta que ya no hubo errores.

4. Roles de los integrantes del equipo

- **Eduardo Michelle (Jefe de Proyecto, Programador BackEnd y FrontEnd):**

Fue el responsable de la coordinación general del equipo durante el desarrollo del proyecto. Participó programando la lógica de algunos apartados del menú principal de la aplicación. Se encargó de conseguir que todas las ventanas de la aplicación fueran responsivas (que los controles cambien de tamaño al redimensionar la ventana) con el uso de GridBagLayout. Supervisó el cumplimiento de tareas y apoyó a los integrantes cuando fue necesario.

- **Mateo Rodríguez (Programador Backend y Base de Datos):**

Creó la base de datos en Access y la estructuró. Configuró las tablas de Usuarios Operativos, Productos y Ventas, así como los campos y tipos de datos que contienen. Desarrolló la lógica de programación en el apartado de Usuarios, asegurándose que se siguiera el sistema de roles. Programó el back de Ventas.

- **Dagoberto Marmolejo (Validación y Diseñador de la Interfaz):**

Validación de entradas de los campos de toda la aplicación: Usuario, Contraseña, Nombre Completo, Nombre de Producto, Descripción, Marca, Precio, Cantidad, etc. Utilizó las expresiones regulares (patrones) para evitar datos inválidos. Diseñó la interfaz gráfica de usuario (GUI) del apartado de productos de la aplicación haciendo uso de los controles que ofrece Eclipse.

- **Ana Padilla (Diseñador de la Interfaz Gráfica y Documentación):**

Diseñó la interfaz del menú principal de la aplicación haciendo uso de los controles que ofrece Eclipse. Además, diseñó los formularios de editar productos, eliminar productos y el menú del inventario. Creación de la presentación de diapositivas del proyecto. Documentó el proceso de programación de la aplicación.

- **Emiliano Cueva (Diseñador de la Interfaz Gráfica y Documentación):**

Diseñó la interfaz gráfica de usuario (GUI) del menú de catálogo de usuarios usando los controles de Eclipse. Además, diseñó los formularios de agregar usuarios y eliminar usuarios. Disposición y estética de botones y demás controles. Documentó el proceso de programación de la aplicación, así como los errores y problemas resueltos.

- **Leo Espinosa (Encargado de Pruebas y Diseñador de la Interfaz):**

Probó exhaustivamente cada uno de los apartados de la aplicación para encontrar posibles errores y reportarlos para su solución. Diseñó la versión preliminar del apartado de Usuarios y Ventas utilizando los controles de Eclipse. Disposición y estética de botones y demás controles.

5.Descripción de los elementos del programa

Librerías y paquetes utilizados (imports)

Los paquetes (o imports en inglés) son como carpetas o categorías que organizan y agrupan clases, métodos y otras funciones. Durante la creación de nuestra aplicación del sistema de la tienda FerreCode, utilizamos paquetes, con el objetivo de poder implementar y cumplir con los requerimientos. Los siguientes son los más notables, ya que tienen funcionalidades específicas.

1. `import java.sql.*;`

Nos proporciona clases para trabajar con bases de datos a través de JDBC (Java Database Connectivity). Utilizando este paquete podemos conectarnos a una base de datos como Access (que es la que utilizamos para este proyecto). Nos permite ejecutar cualquier comando SQL para gestionar los datos contenidos en las tablas, ya sea insertar, actualizar o eliminar, y también podemos leer el contenido de una tabla utilizando los objetos de las clases vistas en el semestre: Connection, PreparedStatement, Statement y ResultSet.

2. `import javax.swing.ImageIcon;` `import java.awt.Image;`

Nos proporciona las clases necesarias para trabajar con imágenes dentro de la interfaz gráfica. Utilizando estas librerías podemos cargar imágenes desde archivos, redimensionarlas y asignarlas como íconos en controles como lo son **JLabel** o **JButton**. **ImageIcon** permite cargar directamente una imagen mientras que **Image** se utiliza para manipular y escalar la imagen. Estas clases permiten, por ejemplo, mostrar la fotografía de un producto dentro del sistema.

3. `import java.awt.event.*`

Nos proporciona clases para manejar eventos que se producen en los componentes gráficos de la interfaz de usuario. Utilizando esta librería nosotros podemos detectar e interpretar acciones del usuario, como hacer clic en un botón, escribir en un campo de texto, mover o arrastrar el ratón, seleccionar una opción de una lista, marcar una casilla, o cuando un componente gana o pierde el foco.

Clases Públicas

Como ya se mencionó anteriormente, para la programación del proyecto se implementaron cuatro clases públicas con métodos y variables estáticas.

- **Conexion.java**

Aquí están contenidas todos los métodos y variables sobre la conexión con una base de datos. Métodos como **abrirConexion()** y **cerrarConexion()**. También tenemos todo lo relacionado con el inicio y término de una sesión, es decir, métodos como **iniciarSesion()**, **cerrarSesion()**, **mostrarError()** (que es para mostrar un mensaje de error cuando se inicia sesión) y variables que guardan los datos de quien inicia sesión.

- **Producto.java**

Aquí están contenidas todas los métodos y variables relacionadas con la gestión de productos, es decir, alta, baja y modificación de productos. También se incluyen los métodos utilizados para consultar productos y para agregar inventario.

- **Usuario.java**

Aquí están contenidas todas los métodos y variables que gestionan el catálogo de usuarios. Contiene funciones para Agregar, Modificar y Eliminar Usuarios.

- **Ventas_funcion.java**

En esta clase se encuentran los métodos y variables que tienen que ver con el apartado de ventas de la aplicación. Están los métodos que gestionan el carrito de compras, que muestran el cobro de un producto (para ingresar la cantidad con la que el cliente paga) y que finalizan una venta.

Variables globales estáticas

Descripción de la función de cada una de las variables...

- **String ruta:** cadena de conexión a la base de datos access.
- **Connection conn:** objeto de Connection para interactuar con la base de datos.
- **String NombreCompleto:** nombre de quien inicia sesión.
- **String IdUsuario:** Id del usuario que inició sesión. Para registrar ventas y corte.
- **String Usuario:** usuario de quien inicia sesión.
- **String TipoUsuario:** permiso de quien inicia sesión.
- **String PasswordBD:** contraseña de quien inicia sesión.
- **bool TienePermiso:** booleano que indica si el permiso es de solo lectura.
- **int ErroresInicioSesion:** cuenta los errores al iniciar sesión. Si hay 3, se cierra.
- **boolean Tipo_Actualizacion:** para gestionar correctamente la edición de la cantidad de productos a vender en el carrito.
- **bool SeCompletoOperacion:** para saber si se completó el cobro o no.
- **float PorPagar:** cantidad a pagar por carrito de compras con IVA.
- **bool SeAutoModifico:** indica si el usuario modificó su propia información.

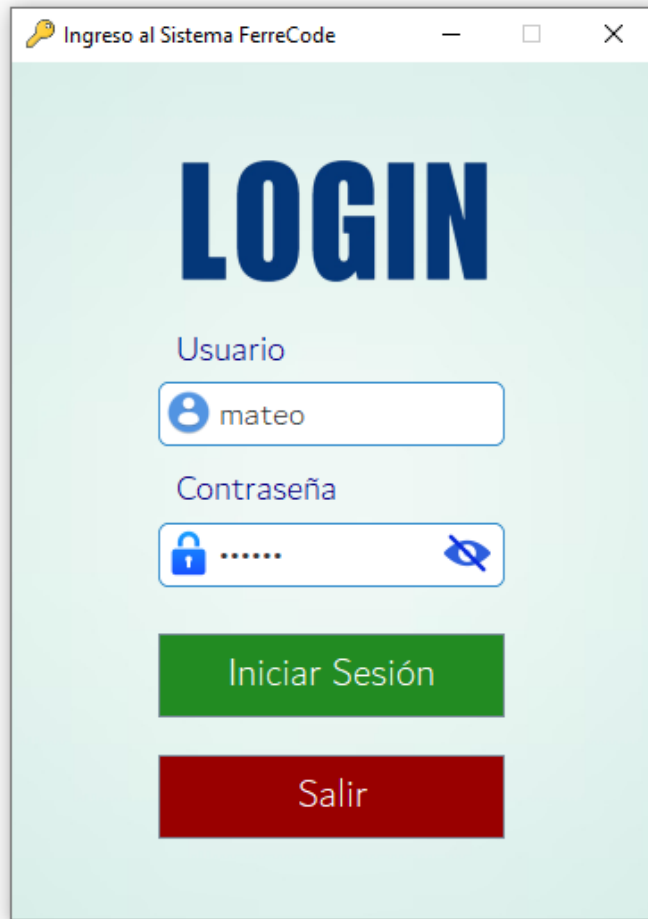
Inicio (Bienvenida al Sistema)



Primer Ventana
mostrada al iniciar la aplicación. Contiene un **Título** con el texto "Bienvenido" y el nombre de la ferretería. Incluye el **Logo** en el centro de la ventana y en la parte inferior un **botón** que **direcciona a** la ventana de **Inicio de Sesión**

Esta ventana solo requirió del diseño visual. Hasta el momento, no se implementa ninguna función, el botón con el texto "Ingresar" simplemente abre el formulario de Inicio de Sesión y se deshace del de inicio.


Inicio de Sesión





Ingreso al Sistema FerreCode

LOGIN

Usuario

 mateo

Contraseña

Iniciar Sesión

Salir

Segunda Ventana mostrada. Contiene un **Título** con el texto “Log In”. Incluye dos cajas de texto para ingresar los datos de inicio de sesión. Incluye un botón que si se presiona muestra la contraseña ingresada. Si se presiona el botón Iniciar Sesión se intentará un Inicio de Sesión. Si se presiona el botón Salir se cerrará Inicio Sesión y se mostrará la ventana de Bienvenida.

Esta ventana es la de Inicio de Sesión en la que el usuario operativo que desea ingresar al menú principal debe ingresar su usuario y contraseña. Para validar la existencia de la cuenta se debe hacer la conexión a la base de datos de la aplicación y buscar en la tabla de usuarios operativos. Si el usuario y contraseña se encuentran se cerrará esta ventana y se abrirá la ventana de inicio sesión. Si no se encuentran se contará un error. Si se cometen tres errores el sistema se cerrará por seguridad.

1. “void iniciarSesion” (Conexion.java)

Procedimiento de inicio de sesión. Recibe las cadenas de texto ingresadas en la caja de usuario y contraseña, las cajas de texto y la ventana actual en la que se inicio sesión. Abre la conexión a la base de datos y utiliza la consulta `"SELECT Nombre_Completo, Usuario, Password, Tipo FROM Usuarios_Operativos WHERE Usuario = ?"` para buscar la existencia de un registro con el usuario ingresado. Si no se encuentra registro se muestra mensaje de error y se añade 1 la variable de errores de inicio de sesión. Si se encuentra un registro con ese usuario asigna a las variables estáticas NombreCompleto, Usuario, PasswordBD y TipoUsuario los valores del registro encontrado. Posteriormente compara Usuario y PasswordBD con lo que el usuario ingresó en los TextBox.

Si ambos coinciden significa que el inicio de sesión fue exitoso y se obtiene el valor del booleano estático TienePermiso de acuerdo con el valor de TipoUsuario. Ahora se cierra la ventana actual de tipo Login y se abre una nueva ventana del menú principal (que contiene instrucciones). Finalmente se reinician variables.

En cambio, si alguno de los dos (Usuario o PasswordBD) se muestra el mensaje de error con “usuario y/o contraseña incorrectos” y se suma 1 a la variable de errores de inicio de sesión. Después de que se añade 1 a esta variable hay una validación de si los errores = 3. Si esto sucede, se cierra el sistema por seguridad.

Menú Principal de la Aplicación



Ventana de menú principal de la aplicación. En la esquina superior izquierda contiene la información (Usuario, Nombre Completo y Permiso) de la persona que inicio sesión. En el panel superior está el nombre de la tienda y en la esquina superior derecha un botón con X que te permite cerrar sesión y salir de la aplicación. En el panel de la izquierda están los 5 botones con las diferentes opciones que tiene el usuario operativo (Inventario, Productos, Vender, Usuarios, Cerrar Sesión).

Eso sí, cuando se carga el formulario de aplicación se consulta el valor de Permiso para saber si se tienen que deshabilitar botones. En el panel principal está cargada una imagen con un mensaje de bienvenida e instrucciones generales del uso de la aplicación. De fondo se observa el logo de la tienda Ferre - Code.

Si no se tiene permiso se deshabilitan los botones y se cambia la imagen de estos por la versión deshabilitada También se carga en las etiquetas de la esquina superior izquierda de la aplicación la información del usuario que inició sesión.

Funciones comunes en el programa

2. “void ReiniciarVariables” (Conexion.java)

Procedimiento utilizado para reasignar las variables estáticas de las diferentes clases usadas en la aplicación al valor con el que fueron inicializadas (básicamente reiniciarlas). Variables con la información del usuario que inició sesión y el booleano HayVentas.

Apartado Inventario

SISTEMA FERRETERIA FERRE-CODE

Gestión de Inventario

Actualiza el inventario del producto seleccionado

Cantidad:

Actualizar Inventario

No. Inv	Nombre	Descripción	Marca	Precio	Cantidad
1	Taladro Eléctrico	Taladro portátil de 500W con velocidad v...	Urrea	\$1500	40
2	MartillodeUa2	Martillo con mango de fibra y cabeza de a...	Truper	\$180	100
3	Destornillador Plano	Destornillador de punta plana con mango ...	Surtek	\$50	148
4	Destornillador de Cr...	Destornillador de estrella con punta magn...	Stanley	\$55	130
5	Cinta Métrica 5m	Cinta métrica retráctil de 5 metros.	Truper	\$90	200
6	Llave Inglesa 10"	Llave ajustable de acero cromado.	Urrea	\$180	78
7	Llave Allen Set	Juego de llaves Allen métricas.	Stanley	\$120	700
8	Segueta para Metal	Segueta con marco metálico y hoja de rep...	Surtek	\$85	60
9	Brochas de 2"	Brochas para pintura en interiores.	PintyPlus	\$25	300
10	Rodillo para Pintura	Rodillo con esponja para acabados lisos.	Surtek	\$60	1200
11	Tubo de PVC 1"	Tubo de PVC de alta presión de media pul...	Rotoplas	\$30	150
12	Codo de PVC 1"	Codo para instalación hidráulica.	Rotoplas	\$10	300
13	Pega PVC 125ml	Adhesivo para tuberías de PVC.	Tangit	\$45	100
14	Pintura Blanca 4L	Pintura vinílica para interiores.	Comex	\$280	50
15	Esmalte Negro 1L	Pintura esmalte para metal y madera.	Surtek	\$110	60
16	Resanador de Made...	Pasta para reparar muebles de madera.	Comex	\$40	90
17	Silicón Transparente	Sellador de silicón multiusos.	Sista	\$55	200
18	Epóxico 2 Compone...	Adhesivo epóxico de alta resistencia.	Resistol	\$95	75
19	Cinta Aislante	Cinta para aislar conexiones eléctricas.	Surtek	\$25	300

Productos Encontrados: 50

Control de usuario para consultar el inventario y agregarlo si se tiene el permiso. En la parte superior hay un título con “Agregar Inventario”. Un poco más abajo tenemos una caja de texto y los diferentes botones (Consultar y Consultar Todos) para hacer la búsqueda de productos. Debajo tenemos la tabla donde se cargan los registros encontrados de la búsqueda.

En la izquierda hay un panel que permitirá agregar unidades. En este panel hay una caja que permite al usuario ingresar la cantidad de unidades que desea añadir al inventario y tenemos un botón para realizar la adición de inventario.

Este apartado también permite consultar la información de un producto automáticamente, al mostrar la imagen del mismo en una etiqueta cada vez que se seleccione de la tabla. Aunque todos los usuarios operativos pueden ingresar al apartado de inventario, cuando este se carga, se valida el permiso del usuario. Si el usuario es un cajero el panel de Agregar Inventario se deshabilita, solo permitiendo la búsqueda y consulta de la información de los productos.

3. “void CargarProductos” (Producto.Java)

Procedimiento utilizado para cargar todos los registros de la tabla productos en un Jtable. Recibe un control JTable y un JLabel (etiqueta) en el que irá la cantidad de registros. Primero asigna a la variable que cuenta los productos el valor inicial de 0 y limpia el contenido de la tabla. Establece la conexión y la consulta SQL `"SELECT * FROM Productos"` con la que se consultan todos los registros. Obtenemos el modelo del JTable. Ahora se utiliza el ResultSet y un bucle while para leer el resultado de la consulta. Para cada registro que se obtenga, la variable ContarProductos aumenta en 1 y se carga en una fila toda la información del registro para posteriormente añadirlo a la tabla con `modelo.addRow(fila)`. Finalmente, una vez cerrada la conexión cambia el texto de la etiqueta contar registros usando la variable ContarProductos. Esta función se utiliza también en los demás apartados.

4. “void AgregarInventario” (Producto.java)

Procedimiento utilizado para agregar al inventario la cantidad de unidades del producto ingresado por el usuario. Recibe un control JTable con los productos, el textbox con el nombre del producto y el textbox de cantidad. Primero hace las siguientes validaciones. Si la cantidad de productos seleccionada es 0 indica con un mensaje de error que debe seleccionar un producto para poder agregar unidades y return. Si la CantidadIngresada es 0 indica con un mensaje de error que no se pueden agregar 0 unidades al inventario y return. Si pasa estas validaciones, muestra un mensaje de confirmación de agregado de unidades, en la que si el usuario dice que no se sale del método. Ahora, a CantidadStockSeleccionado se le suma el valor de CantidadIngresado para obtener la nueva cantidad en stock del producto. Obtiene el registro seleccionado con `getSelectedRow` y actualiza en este registro la columna de Cantidad en Stock por el CantidadStockSeleccionado. Posteriormente, establece la conexión y la consulta SQL `"UPDATE Productos SET Cantidad_en_Stock = ? WHERE Id = ?"` con la que se actualizará la cantidad de productos en stock del seleccionado. Como parámetros de búsqueda el primero es la nueva cantidad en stock y el segundo es el valor de IdSeleccionado (que cambia su valor conforme se selecciona un registro diferente). Se ejecuta el comando de actualización con `executeUpdate()`. Se quita la selección de un producto, las cajas producto y cantidad regresan a su contenido original (placeholder). Finalmente se muestra en MessageBox con el mensaje de agregado de inventario exitoso.

5. “void MostrarProducto” (Producto.java)

Procedimiento utilizado para mostrar la información de un producto seleccionado en el apartado de buscar, incluyendo su imagen. Se llama en el evento que controla cuando se da click al botón Mostrar Producto. Recibe un JTable con los productos. Primero valida si se ha seleccionado un registro, si no hay seleccionados mensaje de error y return. Si se selecciona uno, se obtiene el valor en la columna no Inv del producto. Ahora genera en un `String RutaImagenAbrir` la ruta del archivo de aplicación a la imagen en la carpeta Imágenes del producto seleccionado con ayuda del número de inventario del producto.

Finalmente carga un MessageBox en el que se muestra la información que está contenido en el JTable del producto, así como su la imagen del producto seleccionado.

Apartado Productos

Sistema Ferreteria Ferre-Code

mateo
Mateo Rodriguez
(Admin)

PRODUCTOS

VENDER

INVENTARIO

USUARIOS

CERRAR SESIÓN

Gestión de Productos

Nombre del Producto:

Descripción del Producto:

Marca del Producto:

Precio (\$):

Nombre	Descripción	Marca	Precio	Cantidad
Taladro Eléctrico	Taladro portátil de 500W con velocidad v...	Urrea	\$1500	40
MartillodeUa2	Martillo con mango de fibra y cabeza de ...	Truper	\$180	100
Destornillador Plano	Destornillador de punta plana con mango...	Surtek	\$50	148
Destornillador de C...	Destornillador de estrella con punta mag...	Stanley	\$55	130
Cinta Métrica 5m	Cinta métrica retráctil de 5 metros.	Truper	\$90	200
Llave Inglesa 10"	Llave ajustable de acero cromado.	Urrea	\$180	78
Llave Allen Set	Juego de llaves Allen métricas.	Stanley	\$120	700
Segueta para Metal	Segueta con marco metálico y hoja de re...	Surtek	\$85	60
Brochas de 2"	Brochas para pintura en interiores.	PintyPlus	\$25	300
Rodillo para Pintura	Rodillo con esponja para acabados lisos.	Surtek	\$60	1200
Tubo de PVC 1"	Tubo de PVC de alta presión de media pu...	Rotoplas	\$30	150
Codo de PVC 1"	Codo para instalación hidráulica.	Rotoplas	\$10	300
Pega PVC 125ml	Adhesivo para tuberías de PVC.	Tangit	\$45	100
Pintura Blanca 4L	Pintura vinílica para interiores.	Comex	\$280	50
Esmalte Negro 1L	Pintura esmalte para metal y madera.	Surtek	\$110	60
Resanador de Mad...	Pasta para reparar muebles de madera.	Comex	\$40	90

Productos Encontrados: 50

Ventana para realizar la gestión de los productos. En la parte superior tenemos una caja de texto y los botones Consultar y Consultar Todos para buscar un producto que se quiera gestionar. Debajo tenemos la tabla con los productos encontrados por la búsqueda.

En la parte izquierda tenemos cajas de texto con los diferentes campos de información de los productos, así como una etiqueta para mostrar una imagen. Esta parte se utilizará para editar la información de un producto seleccionado. Para esto, simplemente selecciona un producto, su información se carga en los campos de texto, se edita como se requiere y finalmente se selecciona el botón de editar para finalizar la actualización.

También tenemos entre la caja de búsqueda y la tabla un botón Eliminar Producto, el cual nos permitirá borrar un producto de la tienda. Para esto solo hace falta seleccionar el producto a eliminar en la tabla que se muestra y presionar el botón Eliminar.

A la izquierda de este botón tenemos uno que dice Agregar Producto. Este botón despliega el subapartado para añadir un producto del que hablaremos más adelante.

6. “void consultarProductos” (Producto.java)

Procedimiento utilizado para buscar productos en la base de datos de acuerdo con el texto ingresado por el usuario. Esta función es llamada generalmente cuando se activa el evento del botón de búsqueda. Recibe como parámetros el texto ingresado, la tabla donde se mostrarán los resultados (JTable tablaProductos) y la etiqueta donde se mostrará la cantidad de productos encontrados.

Primero abre la conexión con la base de datos, luego prepara una consulta SQL `"SELECT * FROM productos WHERE nombre LIKE ?"` con LIKE para realizar una búsqueda según el nombre del producto. El texto ingresado se envuelve entre símbolos de porcentaje (%) para que coincida con cualquier parte del nombre. Se ejecuta la consulta con `executeQuery()` y se obtiene el modelo actual de la tabla para poder modificarlo. Antes de cargar nuevos datos, se limpia la tabla utilizando `setRowCount(0)` y se inicializa un contador. Después, mientras haya resultados, se recorre el `ResultSet` y se agregan las filas correspondientes al modelo de la tabla con la información obtenida en la consulta de No. Inventario, Nombre, Marca, Precio (formateado con símbolo de peso) y Cantidad en Stock. Finalmente, se actualiza la etiqueta con el número total de registros encontrados, se cierran el `ResultSet` y la sentencia preparada, y se cierra la conexión a la base de datos.

7. “void ActualizarProducto” (Producto.java)

Función utilizada para modificar los datos de un producto existente en la base de datos. Se llama normalmente desde el evento del botón Modificar Producto. Recibe como parámetros el `JTable` con los productos, las cajas de texto correspondientes a nombre, descripción, marca, precio y cantidad (`JTextField`) y un `JLabel` o componente gráfico que represente la imagen del producto.

Luego se realizan validaciones: si no hay una fila seleccionada en la tabla, se muestra un mensaje indicando que se debe seleccionar un producto para poder modificarlo y el método termina. Si alguno de los campos está vacío o no hay imagen, también se muestra un mensaje de error y termina.

Se convierte el contenido de las cajas a cadenas y el precio y la cantidad usando `Double.parseDouble()` y `Integer.parseInt()`. Si el precio es cero, se muestra un mensaje de error indicando que no se puede modificar el producto con un precio de cero, y termina. Posteriormente se abre la conexión a la base de datos mediante la clase `Conexion`, y se ejecuta una primera consulta SQL: `"SELECT Nombre, Marca FROM Productos WHERE Nombre = ? AND Marca = ? AND Id <> ?"`. Esta consulta verifica si ya existe un producto con el mismo nombre y marca pero con un ID diferente, para evitar duplicados. Si la consulta devuelve un registro, el producto ya existe y se muestra un mensaje de advertencia. Esta validación es importante, ya que un mismo producto no puede ser registrado dos veces bajo la misma marca.

Si todo es válido, se muestra una ventana de confirmación para que el usuario decida si desea continuar con la modificación. Si responde negativamente, la función termina y retorna `false`. En caso de que el usuario confirme, se ejecuta la consulta SQL de actualización: `"UPDATE Productos SET Nombre = ?, Descripcion = ?, Marca = ?, Precio = ?, Cantidad_en_Stock = ? WHERE Id = ?"`.

Si el booleano `seModificoImagen` es verdadero, significa que el usuario seleccionó una nueva imagen. Entonces, se construye la ruta con el Id del producto y se copia el nuevo archivo construyendo una ruta a la carpeta de imágenes, sobrescribiendo el anterior si es necesario.

Una vez realizada la actualización en la base de datos, se limpia la imagen mostrada en el componente visual, se actualiza directamente el registro modificado en la tabla, se quita la selección de la fila activa, se restablecen las cajas de texto y se muestra un mensaje de éxito.

8. “void BorrarProducto” (`Producto.java`)

Este método se encarga de eliminar un producto de la tienda y se invoca al hacer clic en el botón **Borrar Producto**. Recibe como parámetros la tabla `JTable` que contiene los productos, la etiqueta `JLabel` que muestra la cantidad de registros actuales y el campo de texto `TextField` que contiene el término de búsqueda.

Primero, restablece el texto del campo de búsqueda al valor original guardado en su propiedad `Placeholder`. Luego, realiza una validación para asegurarse de que haya una fila seleccionada en la tabla: si no la hay, muestra un mensaje de advertencia indicando que se debe seleccionar un producto para borrarlo y termina la función con `return`. Si hay una fila seleccionada, se muestra una ventana de confirmación preguntando al usuario si realmente desea borrar el producto. Si el usuario elige **No**, se cancela la operación.

Si el usuario confirma la eliminación, se obtiene el **Id** del producto seleccionado desde la fila actual y se procede a borrar el registro de la base de datos. Se utiliza la clase `Conexion` para establecer una conexión con la base de datos y se prepara la consulta SQL: `"DELETE FROM Productos WHERE Id = ?"`. Esta consulta elimina el producto cuyo Id coincide con el seleccionado. El valor del Id se pasa como parámetro en la consulta. Después de eliminar el registro de la base de datos, se construye la ruta a la imagen correspondiente al producto y se elimina utilizando `File.delete(rutaImagen)`.

Luego se actualiza la tabla. Se elimina la fila correspondiente en el `JTable` utilizando su modelo (`DefaultTableModel`), y si la etiqueta de cantidad de productos mostraba un número mayor que cero, se actualiza disminuyéndolo en uno. Finalmente, se muestra un mensaje confirmando que el producto ha sido eliminado correctamente.

SubApartado Añadir Producto

Ventana para realizar el alta de un producto a la ferretería. Esta se despliega cuando se da click en el botón de Agregar Producto del apartado de Producto. Contiene un título en la parte superior con la leyenda "Agregar Producto".

Dentro se observan cajas de texto para ingresar la información del nuevo producto, así como un botón para seleccionar una imagen que se le asignará al producto. Esta imagen se carga en una etiqueta dentro del mismo subapartado. Para concluir el alta solo es necesario presionar en el botón Agregar.



9. "void altaProducto" (Producto.java)

Procedimiento utilizado para registrar un producto en la tienda. Se llama cuando se da click en el botón Agregar Producto. Recibe Nombre, Descripción, Marca ingresados, las cajas de texto (Nombre, Descripción, Marca, Precio, Cantidad) y el JLabel que contiene la imagen ingresada. Si alguno de los campos ingresados (incluida la imagen) está vacío muestra un mensaje de error y retorna para terminar la operación.

Convierte el precio del producto redondeado a dos decimales y la cantidad en stock usando `Double.parseDouble`, `Math.round` y `Integer.parseInt`. Si alguno de estos campos es igual a 0, mensaje de error indicando que no se puede ingresar 0 y regresa falso.

Posteriormente establece la conexión y la primera consulta SQL `"SELECT Nombre, Marca FROM Productos WHERE Nombre = ? AND Marca = ?"` con la que buscará si el producto ingresado con la marca ingresada ya existe en la tabla productos. Si esta consulta devuelve un registro significa que el producto ya existe, muestra un mensaje de advertencia y regresa falso. Hace esto porque es claro que un producto puede ser producido por varias marcas y que una marca puede tener varios productos, pero un mismo producto no puede ser producido dos veces por la misma marca. Si hasta ahora se pasaron todas las validaciones significa que el producto puede ingresar a la tienda.

Establece y ejecuta la segunda consulta `"INSERT INTO Productos (Nombre, Descripcion, Marca, Precio, Cantidad_en_Stock) VALUES (?, ?, ?, ?, ?)"` con la que se inserta el nuevo registro con los campos ingresados.

Establece y ejecuta la tercera consulta `"SELECT MAX(Id) FROM Productos"` para obtener el id del producto que acabamos de insertar. Este Id se almacenará en la variable `IdGenerado`. Ahora genera el string de `RutaDestino` que contendrá la ruta de la aplicación a la imagen en la carpeta de imágenes. Posteriormente utiliza `Files.copy(origen,RutaDestino,StandardCopyOption.REPLACE_EXISTING)` para insertar la imagen ingresada por el usuario en la carpeta `imagenes` del proyecto.

Finalmente restablecemos todas las cajas de texto a su contenido original (placeholder), y mostramos un mensaje de que se agregó el producto exitosamente .

Apartado Vender Producto

Sistema Ferreteria Ferre-Code

mateo
Mateo Rodriguez
(Admin)

SISTEMA FERRETERIA FERRE-CODE

Vender Productos

Consultar Consultar Todos

Nombre	Marca	Precio	Cantidad
Taladro Eléctrico	Urrea	\$1500	40
MartillodeUa2	Truper	\$180	100
Destornillador Plano	Surtek	\$50	148
Destornillador de Cruz	Stanley	\$55	130
Cinta Métrica 5m	Truper	\$90	200
Llave Inglesa 10"	Urrea	\$180	78
Llave Allen Set	Stanley	\$120	700
Segueta para Metal	Surtek	\$85	60
Brochas de 2"	PintyPlus	\$25	300
Rodillo para Pintura	Surtek	\$60	1200
Tubo de PVC 1"	Rotoplas	\$30	150
Codo de PVC 1"	Rotoplas	\$10	300
Pega PVC 125ml	Tangit	\$45	100
Pintura Blanca 4L	Comex	\$280	50
Esmalte Negro 1L	Surtek	\$110	60
Resanador de Madera	Comex	\$40	90
Silicón Transparente	Sista	\$55	200
Enchufe 2 Componentes	Resistol	\$95	75

Realizar Venta

Nombre	Marca	Precio	Cantidad	Total	Total IVA
--------	-------	--------	----------	-------	-----------

Monto a Pagar IVA (MXN)

\$0.00

Productos en Carrito: 0

Producto a Vender

Cantidad a Vender:

0 + -

Agregar a Carrito

Eliminar de Carrito

Vaciar Carrito

Productos Encontrados: 50

Ventana para realizar una venta en la tienda FerreCode. Contiene un título en la parte superior con el texto “Vender Producto”. A la izquierda tenemos las opciones para consultar un producto y la tabla con los productos. A la derecha se encuentra una tabla con los productos que han sido cargados en el carrito de compras. El carrito de compras contiene los campos de Id del Producto, Nombre, Marca, Precio por Unidad, Cantidad, Total y Total con IVA.

Debajo de esta tabla se encuentra el panel de Ventas. Hasta la izquierda en este panel tenemos la información del agregado al carrito, con una primera caja de texto deshabilitada que contiene el monto total de la venta realizada con IVA en pesos mexicanos. Esta caja cambia su valor conforme se modifica el carrito de compras.

Abajo de esta, hay una segunda caja de texto deshabilitada con el nombre del producto a vender que cambia su valor al nombre del producto seleccionado. Abajo hay una tercera caja que permite al usuario ingresar la cantidad de unidades que se venderán de ese producto. A la derecha de la caja cantidad hay unos botones de + y – que dan al usuario operativo la opción de aumentar o disminuir la cantidad mediante clicks.

En el lado derecho tenemos los botones para gestionar el carrito de compras. Hasta arriba se encuentra el botón de agregar al carrito, que al presionarlo carga el producto al carrito con los campos mencionados anteriormente. Debajo de este botón hay dos

botones, uno para eliminar un producto seleccionado del carrito y otro para vaciar por completo el contenido del carrito.

En la parte superior de la ventana, arriba del carrito de productos hay un botón que permite realizar la venta. Cuando se selecciona se abre una ventana para el cobro. Y si el cobro se produce con éxito la venta se realiza y se restablece la ventana de Vender Producto.

10. “void AgregarAlCarrito” (Ventas_funcion.java)

Método utilizado para agregar un producto al ListView del carrito de compras. Al iniciar, el método obtiene la fila seleccionada en la tabla de productos. Si no se ha seleccionado ninguna fila, muestra un mensaje de error que indica al usuario que debe seleccionar un producto para poder añadirlo al carrito, y detiene la ejecución. Si sí hay una selección, el siguiente paso es validar que la cantidad ingresada por el usuario sea un número entero mayor a cero. Si la cantidad es cero o negativa, también se muestra un mensaje de error y se detiene el procedimiento.

Posteriormente, se obtienen los datos del producto seleccionado: su identificador (`id`), nombre, y precio, el cual se limpia de símbolos como el signo de pesos y comas para poder convertirlo a tipo `float`. Se realizan los cálculos del total sin IVA (precio multiplicado por la cantidad) y el total con IVA (aplicando un 16% adicional).

Luego se recorre la tabla del carrito para verificar si el producto ya había sido agregado anteriormente. Si es así, se actualiza la cantidad, el total sin IVA y el total con IVA en la fila correspondiente. También se ajusta el total general de la venta (`total_venta`), restando primero el valor anterior del producto en el carrito y sumando el nuevo valor actualizado. Durante este proceso, se marca con una bandera booleana que el producto ya existía en el carrito.

En caso de que el producto aún no esté en el carrito, se agrega una nueva fila al modelo del carrito con los datos del producto, cantidad, total sin IVA y total con IVA. En este caso, simplemente se suma el nuevo total con IVA al monto total de la venta.

Después de agregar o actualizar el producto, se actualiza el campo de texto `txtPorPagar` con el total a pagar, formateado como moneda. También se actualiza la etiqueta que muestra la cantidad de productos en el carrito, basándose en la cantidad de filas en la tabla del carrito. Finalmente, el método limpia los controles relacionados: restablece el campo de cantidad a su valor original (almacenado en una propiedad personalizada llamada `placeholder`), limpia el campo de búsqueda de productos (`txtProductos`) y deselecta cualquier fila en la tabla de productos para que el usuario esté listo para agregar otro producto si lo desea.

11. “void VaciarElCarrito” (Ventas_funcion.java)

El método `vaciarCarrito` se encarga de eliminar todos los productos del carrito de compras representado por la `tablaCarrito`. Primero verifica si el carrito está vacío; si lo está, muestra un mensaje de error y termina la ejecución. Si contiene productos, solicita confirmación al usuario mediante un cuadro de diálogo. Si el usuario acepta, se eliminan todas las filas del carrito usando `setRowCount(0)`, lo que limpia la tabla.

Después, se restablecen los valores de los campos `txtPorPagar` y `lblCantidadRegistrosCarrito` a sus textos originales utilizando propiedades personalizadas (`placeholder`). Finalmente, reinicia la variable global `total_venta` a cero, dejando el sistema listo para una nueva operación de venta.

12. “void EliminarProductoCarrito” (Ventas_funcion.java)

El método permite eliminar uno o varios productos seleccionados del carrito de compras. Primero, verifica si el usuario ha seleccionado al menos una fila en la tabla; si no es así, muestra un mensaje de error y termina. Si hay selección, solicita confirmación para proceder.

Si el usuario confirma, el método recorre las filas seleccionadas desde la última hasta la primera, resta del total el valor correspondiente a cada producto (columna del total con IVA) y elimina la fila de la tabla. Al final, actualiza el total a pagar y la etiqueta que muestra la cantidad de productos en el carrito.

13. “bool VenderProductos” (Ventas_funcion.java)

La función es responsable de procesar la venta de productos añadidos al carrito de nuestra ferretería. Primero, valida si la tabla que representa el carrito contiene al menos un producto. Si está vacío, muestra un mensaje de error y termina la operación retornando `false`.

En caso de que haya productos, se abre una ventana de cobro (`Cobro_visual`) para que el usuario realice el pago. Si la operación de cobro no se completa (evaluada mediante la variable `seCompletoOperacion`), se cancela la venta retornando `false`.

Si el cobro se realizó exitosamente, la función procede a abrir una conexión a la base de datos y a preparar dos instrucciones SQL: una para registrar cada producto vendido en la tabla **Ventas** ("INSERT INTO Ventas (Id_Usuario, Usuario, Tipo, Id_Producto, Nombre_Producto, Precio_Producto, Cantidad_Producto, Total, Total_IVA, Fecha_Hora) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)"), y otra para actualizar la cantidad en stock en la tabla **Productos** ("UPDATE Productos SET Cantidad_en_Stock = Cantidad_en_Stock - ? WHERE Id = ?"). Por cada fila del carrito, se extraen los datos del producto (ID, nombre, precio, cantidad, total, etc.), se calcula el total con IVA y se ejecutan ambas consultas SQL para guardar la venta y disminuir el inventario.

Después de realizar todas las inserciones y actualizaciones, se limpia la tabla del carrito y se restauran los textos originales de los campos visuales como **txtPorPagar** y **lblCantidadCarrito**, usando valores almacenados como "placeholders". Finalmente, se muestra un mensaje de confirmación al usuario, se cierran la conexión y los objetos **PreparedStatement**, y la función retorna **true** indicando que la venta fue exitosa.

Subapartado de Cobro



Esta ventana se muestra cuando se da click en el botón de hacer venta del apartado de Ventas de la aplicación. Muestra un panel con el título “Ventana de Cobro”. En la izquierda del panel tenemos una caja de texto deshabilitado con el monto a pagar por la venta en pesos mexicanos ya habiéndole agregado el IVA (es la misma cantidad mostrada en el apartado de Ventas). En el centro tenemos una caja de texto en la que el usuario ingresa el dinero con el que va a pagar el cliente. Y a la derecha tenemos un botón para completar el pago y realizar la venta. Una vez que se presione el botón de cobrar se mostrará un mensaje con el cambio que tiene que entregar el usuario operativo y se cerrará la ventana de cobro. A continuación, describiremos la función de **HacerCobro** de este apartado.

14. “bool HacerCobro” (**Ventas_funcion.java**)

La función se encarga de validar que el monto de dinero recibido por el cliente sea adecuado para realizar una venta. Primero, establece que la operación no se ha completado, asignando **false** a la variable **seCompletoOperacion**. Luego verifica si el campo de texto que representa el dinero recibido está vacío; si es así, muestra un mensaje de error y termina la función.

Si el campo contiene un valor, lo convierte a tipo **float** y lo redondea a dos decimales. A continuación, compara ese valor con el total de la venta. Si el dinero recibido es menor, muestra un mensaje indicando que el monto no es suficiente. También valida que el cambio no exceda los \$1000 pesos; si lo hace, lanza una advertencia para evitar errores por montos inusuales.

Si todas las validaciones son correctas, calcula el cambio a devolver y lo muestra en un mensaje informativo. Finalmente, retorna **true** indicando que el cobro se puede realizar correctamente.

Apartado de Usuarios

Sistema Ferreteria FERRE-CODE

mateo
Mateo Rodriguez
(Admin)

PRODUCTOS
VENDER
INVENTARIO
USUARIOS
CERRAR SESIÓN

Gestión de Usuarios

Nombre Completo:

Usuario:

Contraseña:

Tipo de Usuario:

Nombre Completo	Usuario	Password	Rol
Mateo Rodriguez	mateo	*****	Admin
Dagoberto Marmoledo Jimenez	dagoberto	*****	Cajero
Ana Fernanda Padilla Ruiz	ana	*****	Cajero

Usuarios Encontrados: 3

Ventana para realizar la gestión de Usuarios. En la parte superior tenemos una caja de texto y los botones Consultar y Consultar Todos para buscar un usuario que se quiera gestionar. Debajo tenemos la tabla con los usuarios encontrados por la búsqueda. En esta tabla el campo de Password está oculto para proteger la información del usuario.

En la parte izquierda tenemos cajas de texto con los diferentes campos de información de los usuarios, así como una lista desplegable para seleccionar el permiso del usuario. Esta parte se utilizará para editar la información de un usuario seleccionado. Para esto, simplemente selecciona un usuario, su información se carga en los campos de texto y la lista desplegable, se edita como se requiere y finalmente se selecciona el botón de editar para finalizar la actualización.

También tenemos entre la caja de búsqueda y la tabla un botón Eliminar Usuario, el cual nos permitirá borrar un usuario del sistema. Para esto solo hace falta seleccionar el usuario a eliminar en la tabla que se muestra y presionar el botón Eliminar. No se permitirá eliminar el usuario que inició sesión.

A la izquierda de este botón tenemos uno que dice Agregar Usuario. Este botón despliega el subapartado para añadir un usuario del que hablaremos más adelante.

15. “void consultarUsuario” (Usuario.java)

El método realiza una búsqueda de usuarios en una base de datos a partir del texto ingresado en una caja de búsqueda. Si el campo de búsqueda está vacío, muestra un mensaje de advertencia y termina la ejecución.

Primero limpia la tabla visual donde se mostrarán los resultados y establece la conexión con la base de datos. Luego construye una consulta SQL que filtra por nombre o usuario utilizando **LIKE**, y dependiendo del tipo de usuario en sesión (**Admin** o **Propietario**), aplica diferentes restricciones: si es un Admin, solo puede ver su propio registro exacto (con distinción entre mayúsculas y minúsculas) o usuarios con permiso de Cajero; si es Propietario, puede ver todos los registros. Finalmente, la función agrega los resultados encontrados a la tabla, cuenta cuántos hay, y actualiza una etiqueta con el número de usuarios encontrados.

16. “void consultarTodosUsuarios” (Usuario.java)

El método se encarga de mostrar en una tabla (**JTable**) todos los usuarios registrados en la base de datos que el usuario en sesión tiene permiso de visualizar. Primero limpia la tabla para evitar duplicados y luego establece la conexión con la base de datos. A continuación, construye la consulta SQL dependiendo del tipo de usuario que inició sesión: si el usuario es "Propietario", puede ver todos los usuarios; si es "Admin", solo puede verse a sí mismo y a los cajeros. Esta validación se hace utilizando la función **StrComp** de Access para que la comparación entre nombres de usuario sea sensible a mayúsculas y minúsculas.

Luego se ejecuta la consulta, y por cada resultado encontrado se agregan los datos a la tabla. También se lleva un contador que suma cuántos usuarios fueron encontrados, y ese número se muestra en una etiqueta (**JLabel**) con el formato “Usuarios Encontrados: N”. Finalmente, se cierra la conexión con la base de datos.

17. “void ActualizarUsuario” (Usuario.java)

Este procedimiento se encarga de actualizar la información de un usuario en el catálogo de usuarios. Se invoca desde el evento **ActionListener** del botón "Editar Usuario". Recibe como parámetros el nuevo nombre (**nombreModificado**), nuevo nombre de usuario (**usuarioModificado**), la (**contraseñaModificada**), el **JTable** que contiene los usuarios, los campos de texto con la información editada, la lista desplegable con el permiso y el campo de texto usado para buscar usuarios.

El flujo del procedimiento es similar al que se utiliza en la función **actualizarProducto**, pero con diferencias clave. La primera diferencia es la validación que asegura que el nuevo nombre de usuario no esté ya registrado en la

base de datos (en la tabla **Usuarios_Operativos**). Para ello, se abre una conexión a la base de y se prepara la siguiente consulta SQL: **SELECT Usuario FROM Usuarios_Operativos WHERE Usuario = ? AND Id <> ?,**

Ahora, para cada registro obtenido se asigna el valor de usuario en base de datos a una variable temporal y se hace la comparación de esta temporal y el ingresado por el usuario. Si coinciden significa que el usuario ya existe, se muestra mensaje de error y se interrumpe el proceso con un **return**.

Si la validación se supera, se muestra un cuadro de confirmación para continuar con la edición. Luego, si el usuario que está iniciando sesión está modificando su propia información, se actualizan las variables estáticas globales que representan el nombre completo, nombre de usuario y tipo de usuario actual en ejecución (en la clase **Conexion**). Además, una variable booleana **seAutoModifico** se pone en **true**.

Finalmente, se realiza la actualización del registro en la base de datos usando otro **PreparedStatement**, tomando como criterio el ID del usuario seleccionado. Después se actualiza visualmente el registro en la tabla (**JTable**), se limpian los campos de edición y se deja la interfaz lista para futuras modificaciones.

18. “void BorrarUsuario” (**Usuario.java**)

Método utilizado para borrar un usuario del catálogo. Se llama al dar click al botón Borrar Usuario. Recibe el **JTable** con los usuarios, el **Label** con la Cantidad de Registros, y el **TextBox** con el texto a buscar. Primero se restablece el valor original de la caja buscar con la propiedad **tag**. Luego se continúa con la siguiente validación: si no hay usuario que esté seleccionado, mostramos mensaje de error indicando que se debe seleccionar un usuario para poder borrarlo y **return**.

Luego se compara el **id** del usuario seleccionado con el **id** actual en sesión, si coinciden significa que el usuario está intentando eliminarse a sí mismo, mostramos un mensaje de error y **return**. Si se pasan estas validaciones significa que el usuario operativo puede borrarse.


El procedimiento muestra un mensaje de confirmación de si el usuario operativo desea borrar el producto seleccionado. Si selecciona no, **return**. Si continuó con la eliminación se establece la conexión y la consulta SQL **"DELETE FROM Usuarios_Operativos WHERE Id = ?"** con la que borra el usuario de la base de datos. Se pasa como parámetro de búsqueda el **IdSeleccionado** y se ejecuta el comando.

Finalmente obtiene el registro seleccionado con **modelo.getSelectedRow()** y lo eliminamos del **JTable** con **modelo.removeRow()**. Actualizamos la etiquetas de usuarios encontrados con la cantidad de filas de **JTable**. El método termina mostrando un mensaje de “Usuario eliminado correctamente”.

SubApartado de Registrar Usuario

Ventana para realizar el alta de un usuario operativo al sistema de la ferretería. Esta se despliega cuando se da click en el botón de Agregar Usuario del apartado de Usuarios. Contiene un título en la parte superior con la leyenda “Registrar Usuario”.

Dentro se observan cajas de texto para ingresar la información del nuevo usuario, así como una lista desplegable para seleccionar el rol del nuevo usuario. Para concluir el alta solo es necesario presionar en el botón Agregar.



Agregar Usuario

Nombre Completo
Mateo Rodríguez

Nombre Usuario
Mateo123

Contraseña
NuevaContra

Tipo
Cajero

Agregar

20. “void AgregarUsuario” (Usuario.java)

La función `darDeAltaUsuario` permite registrar un nuevo usuario en la base de datos `Usuarios_Operativos`. Primero, obtiene el texto ingresado en las cajas de nombre, usuario y contraseña, y valida que ninguno de estos campos esté vacío. Si alguno lo está, muestra un mensaje de advertencia y termina la ejecución.

Luego, establece una conexión a la base de datos y realiza una consulta para verificar si ya existe un usuario con el mismo nombre utilizando `StrComp`, lo cual permite distinguir entre mayúsculas y minúsculas. Si el usuario ya existe, muestra un mensaje de error y detiene el proceso. Si el usuario no existe, se ejecuta una sentencia `INSERT` que agrega el nuevo usuario con su nombre completo, nombre de usuario, contraseña y tipo de permiso especificado. Finalmente, muestra un mensaje de éxito.

6. Problemas experimentados y soluciones

- **ERROR EN LÓGICA. CONSULTAS SQL EN MICROSOFT ACCESS NO DISTINGUEN ENTRE MAYÚSCULAS Y MINÚSCULAS**

Nos enfrentamos con este error por primera vez al hacer pruebas y validaciones para la ventana que ya habíamos programado de inicio de sesión. Teníamos un propietario con los datos de usuario y password como duenio y duenio, pero al ingresar DUENIO y DUENIO en el apartado de inicio de sesión se nos permitió el ingreso al sistema. Esto nos hizo cuestionarnos la funcionalidad de nuestro método de inicio de sesión e investigar porque sucedía esta anormalidad. Después de investigar, encontramos que, por default, las consultas SQL en Microsoft Access no son sensibles a mayúsculas y minúsculas. Esto significa que si realizamos consultas SQL con = en cadenas de texto no estaremos tomando en cuenta si lo ingresado está en mayúsculas y minúsculas.

Por ejemplo, la consulta **SELECT * FROM Usuarios WHERE Nombre = "Pedro"** devolverá los registros en los que el campo nombre sea Pedro, pedro, PEDRO o cualquier otra combinación de mayúsculas y minúsculas. Esta era la razón por la que se generaban problemas con el inicio de sesión. Propusimos dos soluciones:

1. Cambiar la lógica de la consulta SQL

Al parecernos lo más lógico, nuestra consulta previa era **SELECT * FROM Usuarios_Operativos WHERE Usuario = @UsuarioIngresado AND Password = @PasswordIngresado**, si esta consulta devolvía un registro entonces la cuenta existía y se iniciaba sesión. Pero ya vimos que esto generaba un error. Para solucionarlo nuestra consulta cambio a **SELECT * FROM Usuarios_Operativos WHERE Usuario = @UsuarioIngresado**, con esta consulta obtenemos todos los registros en los que el usuario ingresado coincide, y en lugar de hacer la comparación en la consulta, para cada registro obtenido cargamos la información de la base de datos en cadenas string y procedemos con la comparación de los dos campos. Si para algún campo los campos coinciden, se inicia sesión.

2. Utilizar función StrCmp

Al notar el error, nos dimos cuenta de que nuestro apartado de búsqueda de Usuarios también estaba erróneo, ya que no respetaba las minúsculas y mayúsculas ingresadas por el usuario en la caja de buscar. Después de investigar más sobre el tema, llegamos a una solución más práctica para corregir en el tipo de búsqueda exacta por nombre (el del error), que involucra el uso de StrCmp para comparar cadenas, esta función recibe 3 parámetros, las dos cadenas a comparar y el tipo de comparación. En nuestro caso el tipo de comparación se escribe con 0 indicando que la comparación sea sensible a mayúsculas y minúsculas. De tal manera que nuestra consulta previa **SELECT * FROM Usuarios_Operativos WHERE Usuario = @PorBuscar** se convirtió en **SELECT * FROM Usuarios_Operativos WHERE StrComp(Usuario, @PorBuscar, 0) = 0**

● ERRORES EN SINTAXIS

Podemos decir que los demás errores generados durante la programada fueron debido a errores del estudiante. El uso de variables que no eran las que se necesitaban, errores en consultas por sintaxis e inicialización incorrecta de variables cuando se llamaban a ciertas funciones fueron algunos de los errores más comunes que tuvimos. Como ejemplo podemos mencionar el error que se tenía en el apartado de ventas que no se guardaba correctamente en la base de datos la venta hecha. Se procedía con el cobro y este se realizaba correctamente pero nunca se mostraba el mensaje de venta exitosa. La razón de este error fue difícil de detectar, pero resultó que simplemente recaía en que la conversión de un objeto con el valor de la cantidad de productos a entero no se estaba haciendo bien. Errores de este estilo fueron los más frustrantes, pero pudimos resolverlos todos haciendo uso de algunos JOptionPane para detectar dónde estaba el error.

● COMPLICACIONES CON EL DISEÑO DE INTERFAZ

Aunque esto no sea como tal un error, fue una dificultad algo curiosa que se presentó a lo largo del proyecto. Esto se debió a que algunos integrantes del equipo tenían cierta dificultad a la hora de crear una interfaz sencilla para el usuario, mientras que otros lo hacían de una manera tan sencilla que dejaba de ser atractiva a la vista.

7. Conclusiones (Individuales)

● Conclusión Eduardo Michel

En lo personal el desarrollo de este proyecto fue mejor de lo que esperaba, ya que en un inicio, me aburría Java, más que nada por la parte visual, etc, sentía que era mas facil en C#, y pensamos en entregarlo solo funcional sin nada visual, sin embargo, debido a que buscamos cómo hacerlo responsivo, descubrí un tipo de layout el GridBagLayout, el cual si esta interesante, y al momento de realizar todos los apartados con esta herramienta, fue un poco tedioso se podría decir pero al mismo tiempo interesante, porque hubo veces en las cuales no sabía porque se mostraba así la app, y me quede buscando un rato hasta que lo resolvía, y además todo el tiempo a contrarreloj, ya que lo visual se realizó entre los días sábado y martes antes de entregar el proyecto.

Gracias al desarrollo de este proyecto y a las cosas que aprendí para realizarlo, siento que podría desarrollar más cosas que me proponga y me permitan avanzar en el desarrollo de lógica, etc.

El rol que desempeñe en este proyecto fue el de programador visual más que nada, si desarrolle lógica de back, como un 40%, pero me enfoque mas en la parte visual, ya que esta vez debido al proyecto de programación de internet, no estuve tan presente como lo tenía pensado para la repartición de las actividades o documentos, en el lado de la programación si, sin embargo en la parte de documentos, Mateo me ayudó en gran parte a organizarlos y poder sacar el proyecto en tiempo y forma.

● Conclusión Mateo Rodríguez

Personalmente, trabajar en este proyecto de Programación de la Ferretería fue de mi agrado. A pesar de que teníamos un plazo de entrega corto y que teníamos otro proyecto de programación con el que trabajar, considero que nuestro equipo gestionó bien los tiempos para poder entregar este proyecto a tiempo. Además, fue de gran ayuda la experiencia previa que teníamos del proyecto de quinto semestre, también sobre la creación de un sistema con interfaz visual.

Por esta razón, considero que fue fluido el trabajo en el proyecto y que no nos enfrentamos con tantos errores al programar o diseñar, y en caso de tenerlos fuimos capaces de resolverlos en poco tiempo porque eran problemas que ya habíamos enfrentado en el proyecto anterior.

Creo que el proyecto de final de bachillerato me permitió no sólo aplicar lo adquirido en las clases del semestre, sino que también me motivó a investigar más sobre el lenguaje de Java y cómo implementar todas las herramientas que ofrece para generar una solución lo más cercana posible a lo que nos encontraríamos en el sistema de una ferretería cerca de nuestro vecindario.

Mi rol en este proyecto fue el de subcapitán del equipo y programador de back. El capitán del proyecto fue Eduardo Michel, por lo que ahora me tocó experimentar el

desarrollo de un sistema desde el lado de uno de los integrantes del equipo y no como capitán. Con esto pude conocer la forma de trabajo en los equipos de Eduardo y aprendí mucho sobre como capitanear un equipo para hacer proyectos con eficiencia. Diría que esta experiencia de proyecto fue muy enriquecedora y completa.

Gracias a los proyectos de programación realizados en bachillerato, me siento preparado para ingresar a una carrera universitaria en Software.

● **Conclusión Dagoberto Marmolejo**

La manera en la que desarrollamos este proyecto fue muy interesante en lo personal, ya que aunque no teníamos tanto tiempo para poder organizarnos de mejor manera o estructurada, logramos dividir el trabajo de manera en la que todos estaríamos en nuestras “zonas de confort” teniendo mejor control del proyecto y tener registros más precisos sobre todo lo que hicimos.

Este proyecto final fue algo sencillo ya que en si gracias a los conocimientos previos, lo único complicado de este fue el adaptarlo a nuestro nuevo lenguaje de programación (Java), ya que sabíamos todo sobre la estructuración y la creación de ventanas y formularios adaptados al programa y se dio un resultado muy completo.

Fue un trabajo muy bien dividido y gracias a esto, logramos tener una mejor comunicación como equipo, asimismo nos facilitó la realización de nuestro proyecto, y asumimos responsabilidades en cada rol en los cuales nos habíamos adaptado.

● **Conclusión Ana Fernanda Padilla**

En conclusión de este proyecto, fue un proyecto grande y también puedo decir que digno de ser nuestro último proyecto para salir de el bachillerato, fue realmente interesante el crear el sistema de ferreteria, fue un poco pesado el estar trabajando con dos proyectos al mismo tiempo fue un reto difícil pero es necesario para nuestro entendimiento y crecimiento, realmente no tuvimos muchas clases de programación así que en si fue un reto de investigar un lenguaje de programación que no conocíamos mucho y aun así falta mucho que aprender.

A lo largo del proyecto el equipo enfrentó muchos desafíos como la gestión de las imágenes que eviten conflictos de acceso y la necesidad de adaptar las consultas SQL por la inestabilidad de access con las mayusculas y minusculas. Al final el tiempo fue el que nos evitó llegar aún más allá pero realmente quedamos satisfechos con el resultado que obtuvimos, de este proyecto salimos con grandes aprendizajes por experimentar, así que, fue una experiencia dura pero gratificante mejorando así las habilidades de programación y darnos una idea de lo que es trabajar en un proyecto de software complejo.

● Conclusión Emiliano Cueva

En mi conclusión sobre este proyecto, puedo decir que fue un reto bastante grande. A lo largo del semestre, tuvimos que lidiar con tiempos ajustados y, en muchos casos, los elementos del programa no estaban claros al principio. Esto nos llevó a investigar mucho por nuestra cuenta, probar y cometer errores, pero al final, con días de trabajo y colaboración, pudimos completar varias funciones importantes del programa.

Al principio, nos enfocamos en apartados clave como agregar, eliminar y comprar productos, que fueron los puntos de partida del proyecto. Luego, vino la parte más difícil: integrar la base de datos con el programa. Aún con las dificultades, logramos conectarlo todo y hacer que el programa de la tienda de autopartes funcionara.

Este fue posible gracias al trabajo en equipo. Cada idea, sin importar lo pequeña que fuera, era importante. Cada aportación nos dio una nueva perspectiva sobre cómo mejorar el programa. Además, la documentación fue algo también importante, porque todos pudimos compartir nuestras ideas, los problemas que enfrentamos y cómo los solucionamos. En resumen, creo que este proyecto no solo fue una experiencia de mucho aprendizaje, sino también una excelente oportunidad para mejorar nuestras habilidades de trabajo en equipo.

● Conclusión Leonardo Espinosa

El tiempo de trabajo de este proyecto fue algo corto y siento que por las prisas no se pudo trabajar de una forma más elaborada y colaborativa, pero también por el entorno en el que se hizo el proyecto se elaboró el proyecto fue un poco mas facil y rapido en general creo que el lenguaje de programación es más sencillo e intuitivo que otros como C#, en específico yo me encargue de elaborar las ventanas del menú de ventas y el de editar usuario y también hacer parte del documento, realmente no tuve muchos conocimientos nuevos ya que a mi no me tocó en sí programar y fuera de eso solo hubiera utilizado los conocimientos vistos en clase y sobre el trabajo en equipo estoy muy satisfecho ya que hubo muy buena organizacion y comunicacion todos trabajamos y cada quien escogio lo que quería hacer y creo que estuvo bien ya que gracias eso pudimos escoger las actividades que mejor nos salen con las habilidades que poseemos.

Para mi este proyecto en particular fue un poco menos complicado y complejo que el del semestre pasado ya que en lo personal se me dificulta un poco más el lenguaje de C# por que en mi opinion visual studio era más especial y delicado hablando de errores y por eso este proyecto se me hizo un poco menos complejo además de que me siento un poco más cómodo con el lenguaje Java. Y ya para concluir este proyecto me gusto porque hubo un buen trabajo en equipo y una buena organización y aparte los compañeros que se encargaron de la elaboración del programa hicieron un buen trabajo.

Declaratoria de uso de la Inteligencia Artificial Generativa

Declaramos como equipo, que el proyecto de fin de semestre asignado se ha realizado bajo las siguientes condiciones de uso de la Inteligencia Artificial Generativa (IAG):

☐ **No** se hizo uso de la IAG en el desarrollo del proyecto

Nota: al declarar que no hice uso de Inteligencia Artificial Generativa, acepto que, de ser necesario, nuestro profesor puede profundizar en la originalidad de este proyecto.

☒ **Sí**, hice uso de la IAG para:

- ☐ Redactar
- ☐ Procesar datos
- ☒ Crear imágenes
- ☒ Buscar información
- ☐ Crear presentaciones
- ☐ Corregir redacción y ortografía
- ☐ Obtener código de programación
- ☐ Otros (especificar): _____

La(s) aplicación(es) que usé fue (ron):

Ej. Copilot, Gemini, ChatGpt, Perplexity, etc.

ChatGpt y Perplexity

La

dirección electrónica y los prompts que se utilizaron son:

Ej.

<https://www.perplexity.ai/search/tengo-un-jframe-con-un-gbc-y-q-VrjpcTQTRoqzVv9PlhagKw#0>

<https://chatgpt.com/share/68374218-5cf0-8013-8e3b-f312acc7c162>

<https://chat.openai.com/share/f996b13c-a8e4-4926-af14-5ffe59821cf0>

De acuerdo con el código de programación entregado en el proyecto, señala circulando el porcentaje hecho con IAG.



Confirmamos que el equipo ha leído todo y ha verificado que la información es correcta y precisa.

☒ Sí
☐ No