UNIVERSIDAD MARIANO GÁLVEZ DE GUATEMALA
FACULTAD DE INGENIERIA EN SISTEMAS DE INFORMACIÓN
LICENCIATURA EN INGENIERÍA EN SISTEMAS DE INFORMACIÓN
CENTRO UNIVERSITARIO DE CHIMALTENANGO

CUARTO CICLO
PROGRAMACION II

Manual De Tecnico

Daniel Angel Ambrocio Coj 1990-22-13443

ESTUDIANTE

CHIMALTENANGO, SEPTIEMBRE DE 2023

Introducción

En Puzzle Haven, nos enorgullece ser parte de la emocionante comunidad de entusiastas de los cubos de Rubik, proporcionando una amplia gama de rompecabezas y recursos para satisfacer todas tus necesidades. Este Manual Técnico es tu guía esencial para comprender y dominar nuestro avanzado Sistema de Gestión de Inventario, el motor que impulsa nuestra operación de venta de cubos de Rubik de manera eficiente y efectiva.

Los cubos de Rubik han cautivado a personas de todas las edades y niveles de habilidad en todo el mundo durante décadas, y en Puzzle Haven, estamos comprometidos a brindar una experiencia excepcional a nuestros clientes. Nuestro Sistema de Gestión de Inventario es una parte fundamental de esta misión, ya que garantiza que tengamos los productos adecuados disponibles en el momento adecuado para satisfacer tus necesidades y deseos.

Te invitamos a sumergirte en las páginas de este Manual Técnico y descubrir cómo Puzzle Haven continúa siendo un destino confiable y apasionante para los amantes de los cubos de Rubik. ¡Bienvenido a Puzzle Haven, donde el desafío se convierte en diversión y la satisfacción del cliente es nuestra prioridad número uno!

Objetivos

El sistema tiene como objetivo principal permitir a los usuarios gestionar el inventario de cubos de Rubik de manera eficiente. Los objetivos específicos incluyen:

- Ver el inventario de cubos de Rubik.
- Eliminar productos del inventario.
- Agregar nuevos productos al inventario.
- Modificar la información de los productos en el inventario.
- Buscar productos por nombre.

2. Arquitectura del Sistema

El sistema de Puzzle Haven consta de las siguientes partes principales:

- Interfaz de Usuario: Proporciona una interfaz gráfica para que los usuarios interactúen con el sistema.
- Base de Datos MySQL: Almacena la información del inventario de cubos de Rubik.
- Lógica de Aplicación: Maneja las operaciones de agregar, modificar, eliminar y buscar productos en la base de datos.
- Conexión a la Base de Datos: Establece y gestiona la conexión a la base de datos MySQL.

3. Requisitos del Sistema

Para ejecutar el sistema de Puzzle Haven, se requieren los siguientes componentes:

- Hardware:
  - Un servidor o computadora con capacidad para ejecutar una base de datos MySQL.
- Software:
  - Sistema operativo compatible.
  - Java Runtime Environment (JRE).

- Servidor MySQL.

Requerimientos	**Descripción**
Navegadores	Un buen navegador de internet actualizado permite el acceso óptimo a la plataforma. En este caso se recomienda el uso de google Chrome.
Netbeans	Se recomienda el uso de NetBeans 8.0 que fue la versión donde se programó el sistema de información.
XAMPP	Se recomienda el uso de la versión 3.2.2 de Xampp para no tener dificultades con los puertos de apache y mysql.

Documentación de Código

Seguridad

Las medidas de seguridad se establecieron con un Login en donde solicita un usuario y contraseña decodificados en el código fuente.

```
private void iniciarActionPerformed(java.awt.event.ActionEvent evt) {


    String user, pwd;

    user = txtuser.getText();
    pwd = pass.getText();

    if (user.equals("cubo") && pwd.equals("cubo2023")) {
        JOptionPane.showMessageDialog(null, " !Bienvenido! ");
        Ventana ven = new Ventana();
        ven.setVisible(true);
    } else {
        JOptionPane.showMessageDialog(null, "Usuario o Password Incorrectas!!!");
    }


}
```

Conexión con MYSQL

Para lograr la conexión con mysql se utilizó la siguiente clase que esta e un paquete conection.

```
Connection con;


public conection (){

    try {
    Class.forName("com.mysql.jdbc.Driver");
    con=DriverManager.getConnection("jdbc:mysql://localhost:3306/bdnegocio","root","");



    } catch (ClassNotFoundException | SQLException e){
        System.err.println(" No se pudo conectar a la base de datos. Error: " +e);

    }



}

public Connection getConnection() {
    return con;

}
```

Ver Inventario

Para mostrar los datos del inventario a la tabla, utilizamos arreglos para mostrar los datos de cada casia.

```
    void consultar() {
        String sql = "Select * from producto";

        try {
            conet = con1.getConnection();
            st = conet.createStatement();
            rs = st.executeQuery(sql);
            Object[] producto = new Object[5];
            modelo = (DefaultTableModel) Tabla.getModel();

            while (rs.next()) {
                producto[0] = rs.getString("Codigo_Producto");
                producto[1] = rs.getString("Nombre_Producto");
                producto[2] = rs.getFloat("Precio_Unitario");
                producto[3] = rs.getInt("Cantidad_Producto");
                producto[4] = rs.getString("Edad_de_Uso");

                modelo.addRow(producto);
            }
            Tabla.setModel(modelo);

        } catch (Exception e) {

        }
```

Buscar

Para buscar se utiliza el nombre del producto. Se utiliza un filtro para filtrar la información.

```
    private void buscarActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        busqueda.addKeyListener(new KeyAdapter() {

            @Override
            public void keyReleased(final KeyEvent e) {

                String cadena = busqueda.getText();
                busqueda.setText(cadena);
                repaint();
                filtro();

            }

        });
```

TablaMouseClicked

Utilizamos MouseClicked para el momento de seleccionar un producto en la tabla lo muestre en los datos.

```
private void TablaMouseClicked(java.awt.event.MouseEvent evt) {
    // TODO add your handling code here:
    int fila = Tabla.getSelectedRow();
    if (fila == -1) {
        JOptionPane.showMessageDialog(null, "No se selcciono la fila");

    } else {
        String codigo = (String) Tabla.getValueAt(fila, 0);
        String nombre = (String) Tabla.getValueAt(fila, 1);
        float pre = (float) Double.parseDouble((String) Tabla.getValueAt(fila, 2).toString())
        int canti = Integer.parseInt((String) Tabla.getValueAt(fila, 3).toString());
        String ed = (String) Tabla.getValueAt(fila, 4);

        ide.setText("" + codigo);
        nom.setText("" + nombre);
        precio.setText("" + pre);
        cantidad.setText("" + canti);
        edad.setText("" + ed);

    }
```

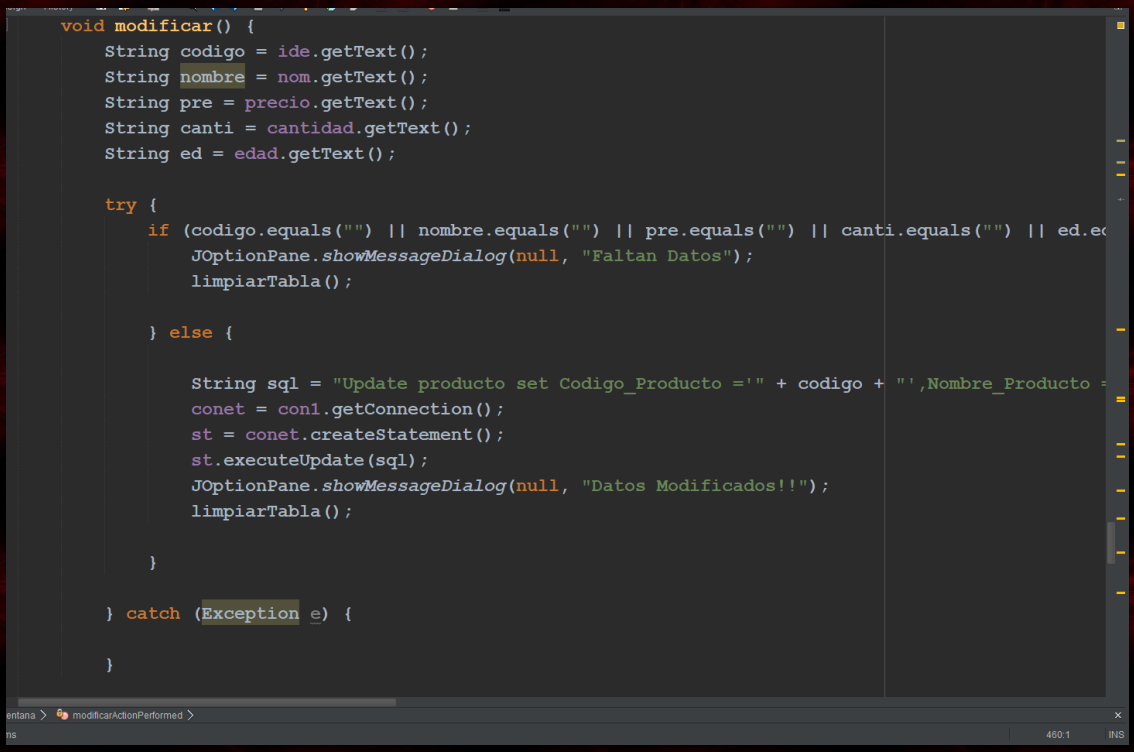
Agregar

Para la función agregar utilizamos la variables que se escriben en los jfieltext le asignamos una variable y se añade ala base de datos.

```
void agregar() {
    String codigo = ide.getText();
    String nombre = nom.getText();
    String pre = precio.getText();
    String canti = cantidad.getText();
    String ed = edad.getText();

    try {
        if (codigo.equals("") || nombre.equals("") || pre.equals("") || canti.equals("") || ed.e
            JOptionPane.showMessageDialog(null, "Faltan Datos");
            limpiarTabla();

        } else {

            String sql = "insert into producto(Codigo_Producto,Nombre_Producto,Precio_Unitario,(

            conet = con1.getConnection();
            st = conet.createStatement();
            st.executeUpdate(sql);
            JOptionPane.showMessageDialog(null, "Datos Guardados Correctamente!!");
            limpiarTabla();

        }
```

Modificar

Para la función modificar utilizamos nombrevaribale.equals donde sea iguales lo modificara según la nueva variable que se ingreso en el Jfieltext



```
void modificar() {
    String codigo = ide.getText();
    String nombre = nom.getText();
    String pre = precio.getText();
    String canti = cantidad.getText();
    String ed = edad.getText();

    try {
        if (codigo.equals("") || nombre.equals("") || pre.equals("") || canti.equals("") || ed.ec
            JOptionPane.showMessageDialog(null, "Faltan Datos");
            limpiarTabla();

        } else {

            String sql = "Update producto set Codigo_Producto ='" + codigo + "',Nombre_Producto =
            conet = con1.getConnection();
            st = conet.createStatement();
            st.executeUpdate(sql);
            JOptionPane.showMessageDialog(null, "Datos Modificados!!");
            limpiarTabla();

        }

    } catch (Exception e) {

    }
```

Nuevo

Para la función nuevo se limpia los campos para realizar una nueva acción.

```
            }

        } catch (Exception e) {
        }

    }

    void nuevo() {
        ide.setText("");
        nom.setText("");
        precio.setText("");
        cantidad.setText("");
        edad.setText("");

    }
```

JasperReport

Utilizamos la función JasperReport para crear un reporte de la mercadería disponible en la base de datos.

```
}

void doc() {
    String informeOrigen = "C:\\Users\\HYMA 57\\Documents\\NetBeansProjects\\Puzzle_Haven\\sr

    try {
        //compilamos el archivo jrkml
        JasperReport jasperReport = JasperCompileManager.compileReport(informeOrigen);

        //crear informe
        JasperPrint jasperPrint = JasperFillManager.fillReport(jasperReport, null, conet);

        //System.out.println("Gennerando informe");
        // exportar el formato que se quiere del pdf
        JasperExportManager.exportReportToPdf(jasperPrint);

        // visualizar el archivo con JasperWiewer
        JasperViewer.viewReport(jasperPrint);

        // En caso de erro se nos mandara por consola
    } catch (JRException e) {
        JOptionPane.showMessageDialog(null, "No se pudo generar el documento!!!");

    }
```