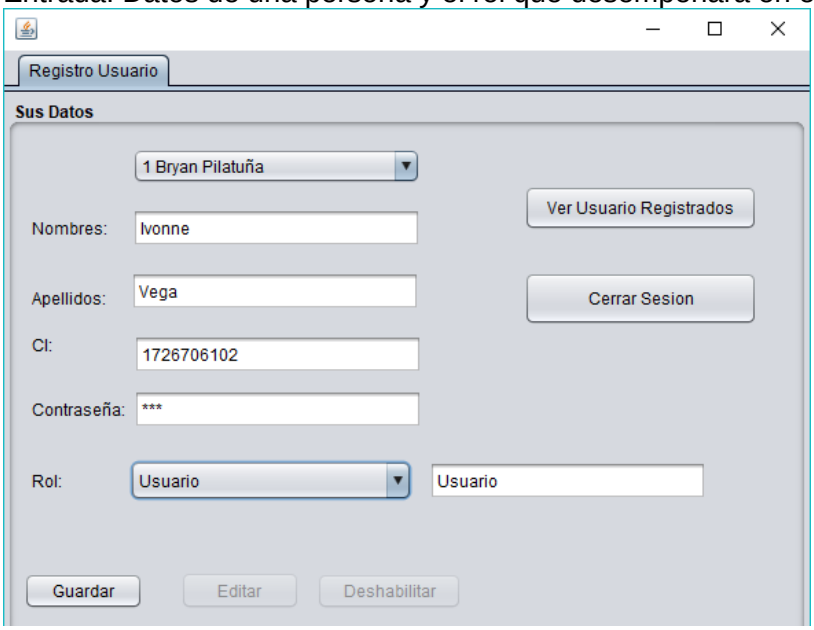
PRUEBAS DE CAJA BLANCA Y NEGRA

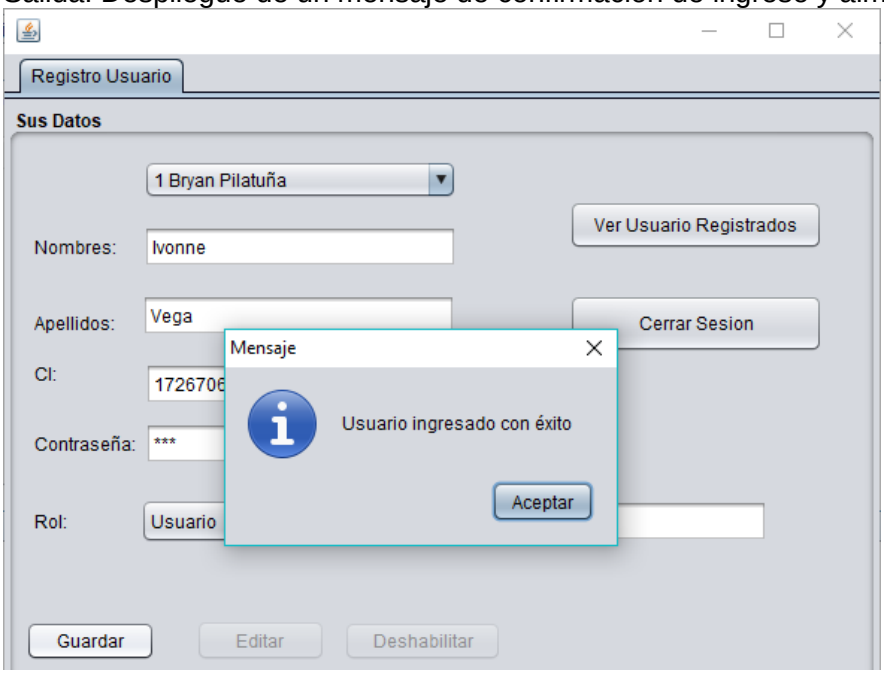
1. ***Pruebas de caja negra***

1.1 Administrador de caja chica

Entrada: Datos de una persona y el rol que desempeñará en el sistema.



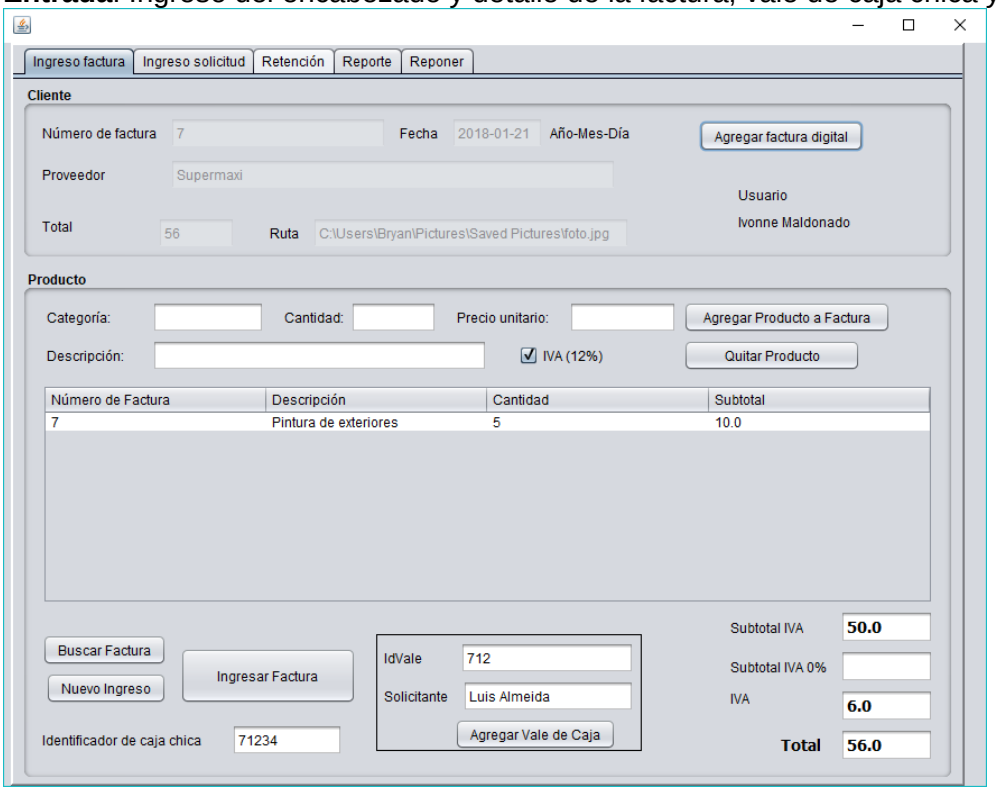
Salida: Despliegue de un mensaje de confirmación de ingreso y almacenamiento de datos.

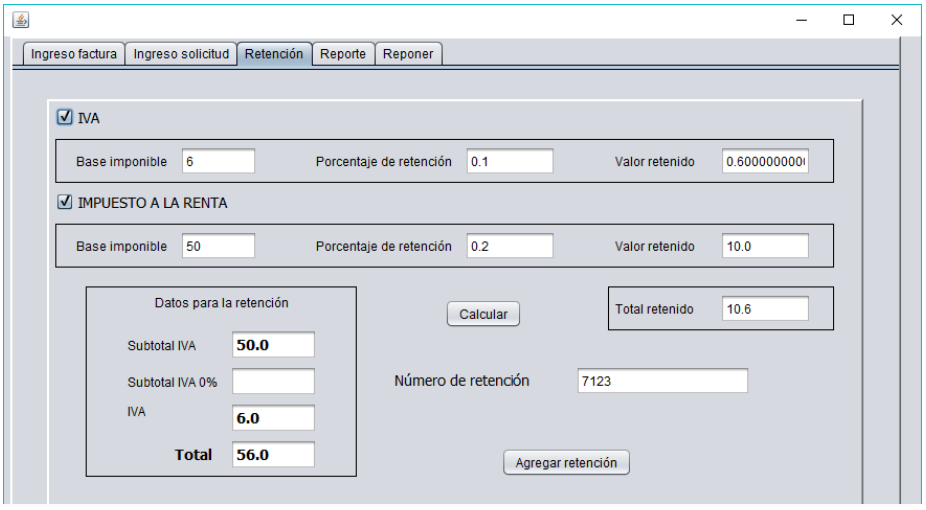


1.2 Usuario de caja chica

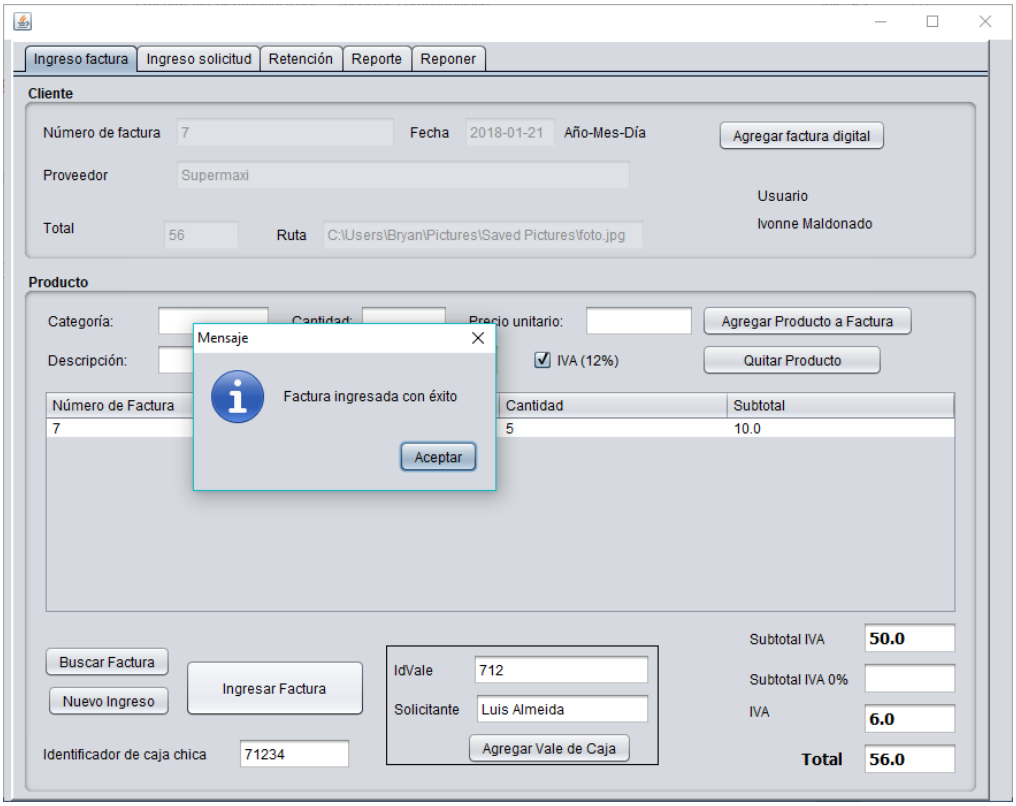
1.2.1 Factura

Entrada: Ingreso del encabezado y detalle de la factura, vale de caja chica y retención.



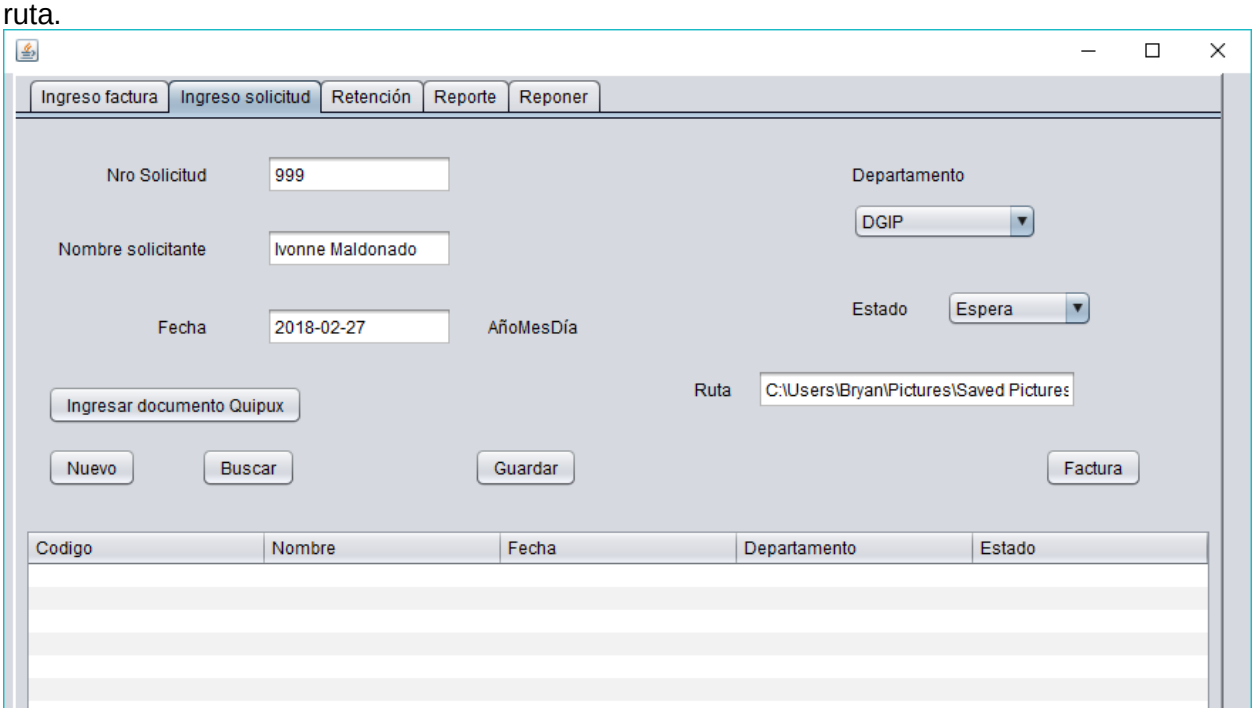


Salida: Ingreso en la base de datos y despliegue de un mensaje de ingreso.

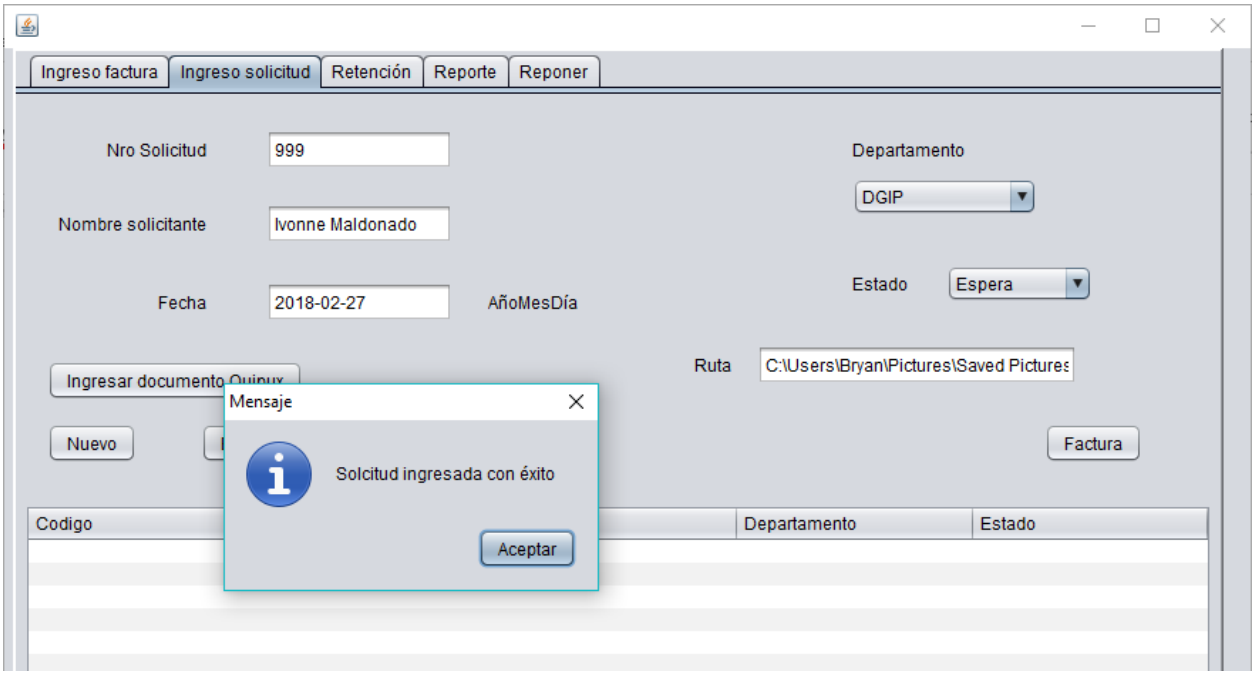


1.2.2 Solicitud

Entrada: Ingreso de los datos de la solicitud, el solicitante, el estado, la fecha, departamento y ruta.

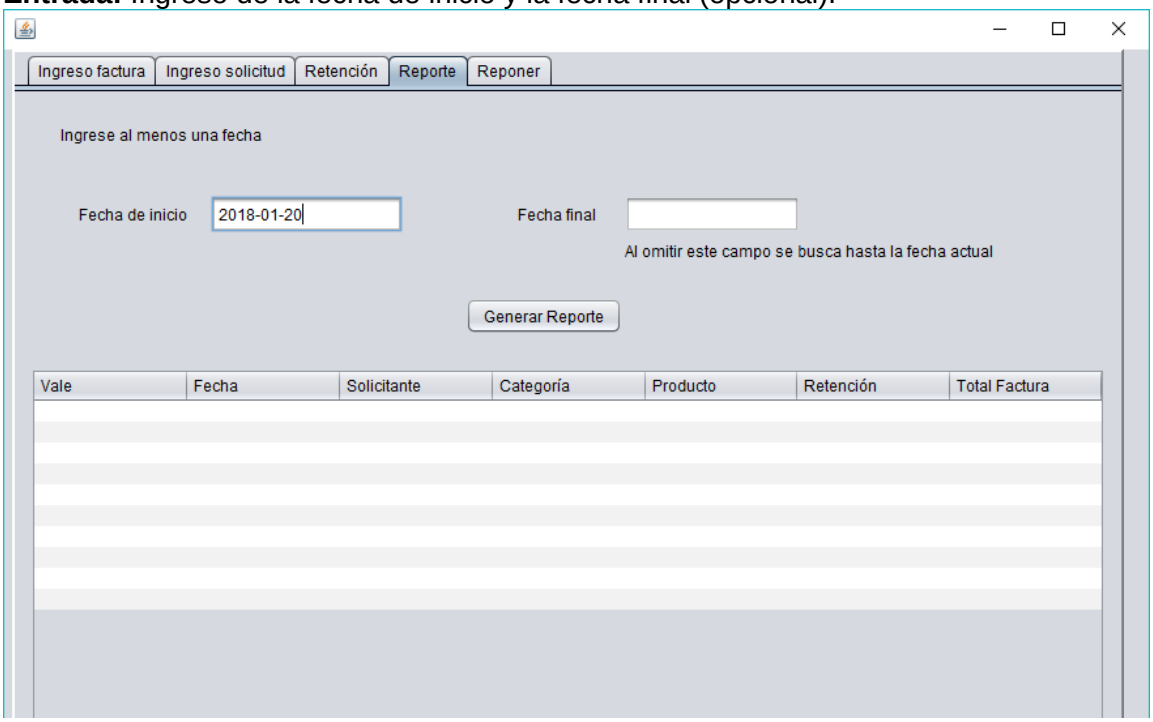


Salida: Mensaje de confirmación de ingreso y almacenamiento en la base de datos.

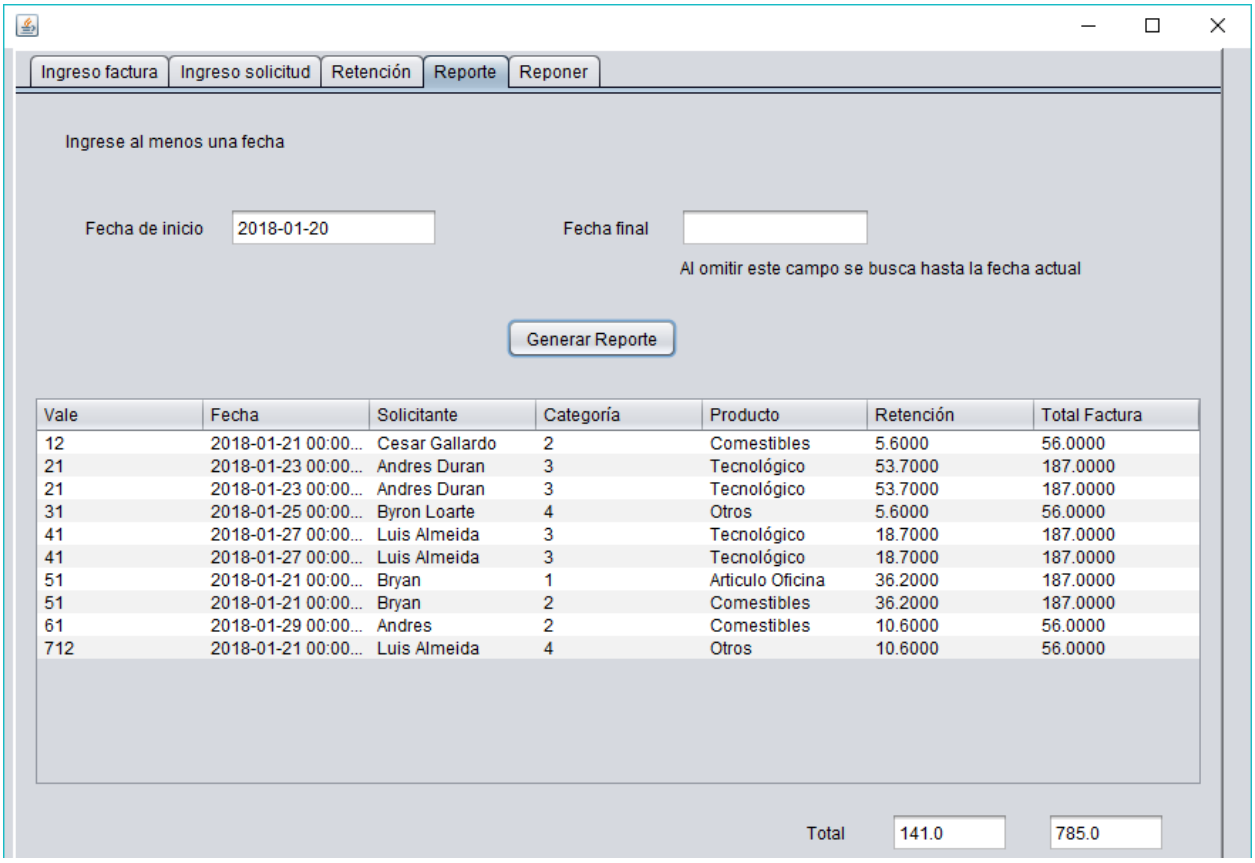


1.2.3 Reporte

Entrada: Ingreso de la fecha de inicio y la fecha final (opcional).

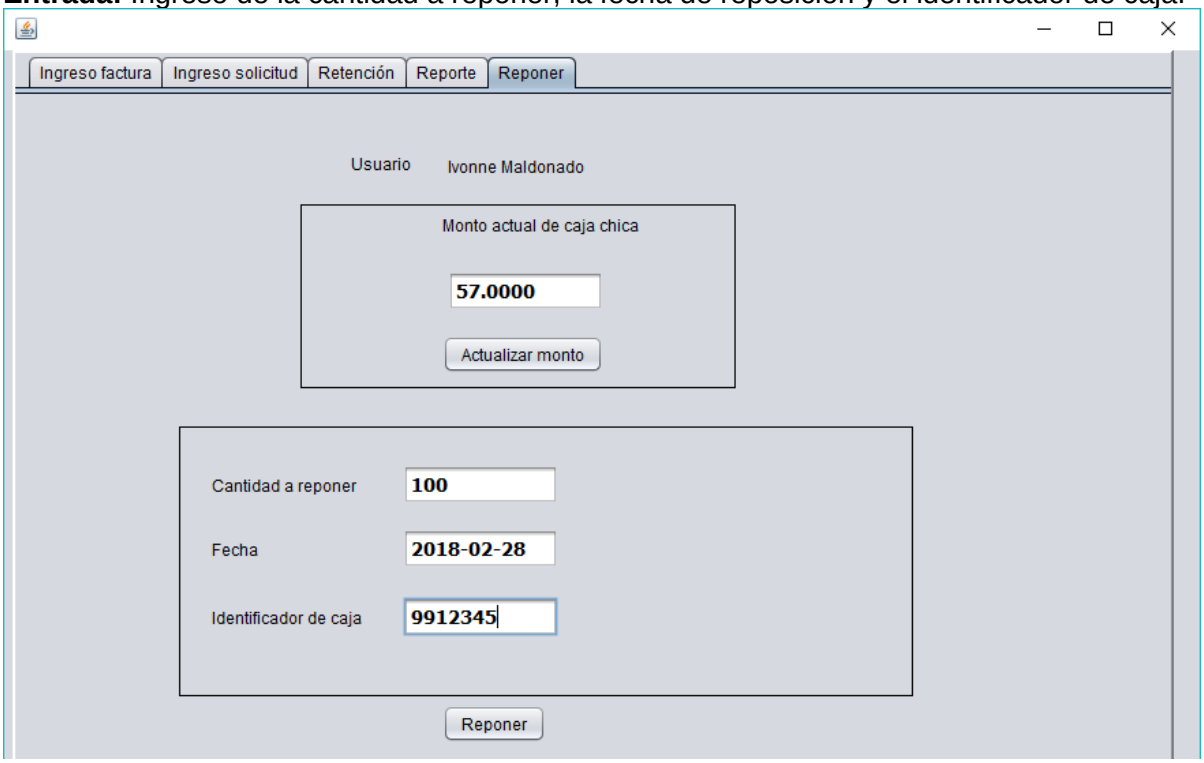


Salida: Detalle de las facturas ingresadas desde la fecha de inicio hasta la fecha final o en el caso de no haber llenado ese campo hasta la fecha actual, junto con la suma del total y la retención de todas las facturas.

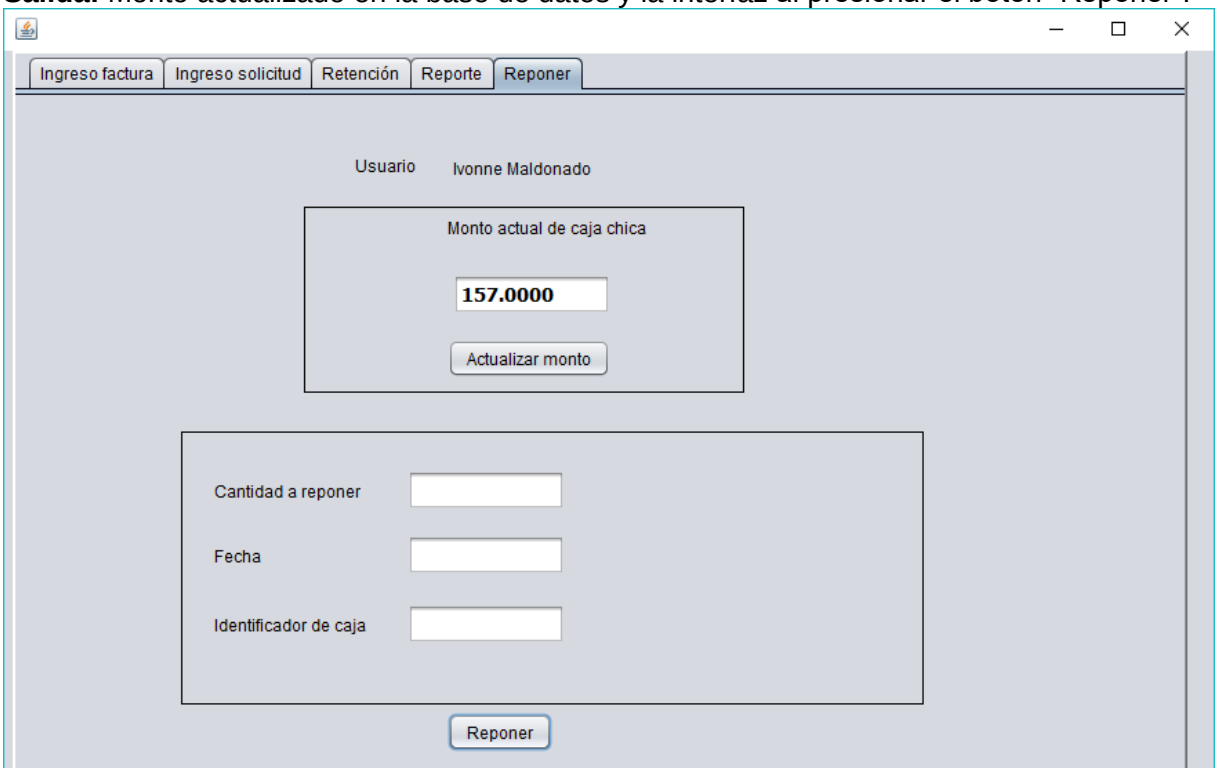


1.2.4 Reposición

Entrada: Ingreso de la cantidad a reponer, la fecha de reposición y el identificador de caja.



Salida: Monto actualizado en la base de datos y la interfaz al presionar el botón “Reponer”.



2. ***Pruebas de caja blanca***

2.1 Administrador de caja chica

2.1.1 Validar contraseña:

Prueba 1: Se ingresó varios campos numéricos de diferentes longitudes logrando obtener con éxito el resultado deseado, en este caso un valor verdadero o falso dependiendo si la cédula es válida.
Prueba 2: Se ingresó letras, y el método dio un error, este error se cubre con otro método que detecta si el campo ingresado es numérico.

```
private boolean validarContrasena(String cedula) {
    boolean validar = false;
    int size = cedula.length();
    if (esNumerico(cedula) && size == 10) {
        int provincia = Integer.parseInt(cedula.substring(0, 2));
        int digito3 = Integer.parseInt(cedula.substring(2, 3));
        int diez = Integer.parseInt(cedula.substring(9, 10));
        int mul, arreglo, res, digver;
        String ced = "212121212";
        int sum = 0;
        for (int i = 0; i < 9; i++) {
            mul = Integer.parseInt(cedula.substring(i, i + 1));
            arreglo = Integer.parseInt(ced.substring(i, i + 1));
            res = mul * arreglo;
            if (res >= 10) {
                res = res - 9;
            }
            sum = sum + res;
        }
        int suma = sum;
        while (suma % 10 != 0) {
            suma++;
        }
        digver = suma - sum;
        if (provincia >= 1 && provincia <= 24) {
            if (digito3 < 6) {
                if (digver == diez) {
                    validar = true;
                }
            }
        }
    }
 private boolean esNumerico(String cedula) {
     try {
         Integer.parseInt(cedula);
         return true;
     } catch (NumberFormatException nfe) {
         return false;
     }
 }
```

2.1.2 Cédula repetida

Este método al estar validado con el método numérico siempre recibe una cadena de texto con un número de cédula.

Prueba 1: Se ingresó un número de cédula anterior y determinó que la cédula ya se encontraba en la base de datos prohibiendo así el ingreso.

```
private boolean cedulaRepetida(String cedul) {
    String cedula;
    boolean cedulaRep = false;
    ResultSet res = Conexiones.Conexion.Consulta("select * from USUARIO");
    try {
        while (res.next()) {
            cedula = res.getString("NICK_USUARIO");
            if (cedula.equals(cedul)) {
                cedulaRep = true;
            }
        }

    } catch (SQLException e) {
    }
    return cedulaRep;
}
```

2.1.3 Llenar combo con usuarios

Prueba 1: Se eliminó todos los usuarios de la base e igual seguía funcionando el programa con la diferencia que el combo está vacío.
Prueba 2: Se agregaron varios usuarios a la base y se cargaban los mismos en el combo.

```
public void llenarCombo() {
    String n, i;
    usuarios.clear();
    try {
        ResultSet res = Conexiones.Conexion.Consulta("select * from USUARIO order by ID_USUARIO");
        while (res.next()) {
            n = res.getString("NOMBRE_USUARIO");
            i = res.getString("ID_USUARIO");
            System.out.println(i + " " + n);
            usuarios.add(i + " " + n);
        }
        System.out.println(usuarios);

    } catch (SQLException e) {
    }
    cmbUs.setModel(new javax.swing.DefaultComboBoxModel(usuarios.toArray()));
}
```

2.1.4 Registro de usuario

Para guardar el usuario en la base de datos se hizo uso de los anteriores métodos mencionados, ya que los posibles errores se capturaron con las sentencias condicionales, no se requirió realizar demasiadas pruebas, igualmente para mostrar los usuarios y editar los mismos.

```
private void bgActionPerformed(java.awt.event.ActionEvent evt) {
    if (tnom.getText().isEmpty() || tapel.getText().isEmpty() || tc.getText().isEmpty()
            || pc.getText().isEmpty()/*||trol.getText().isEmpty()*/) {
        //if para validar que los campos no esten vacíos
        JOptionPane.showMessageDialog(null, "Uno o más campos están vacíos");
    } else if (!validarContrasena(tc.getText())) {
        //if para validar que la cédula sea verdadera
        JOptionPane.showMessageDialog(null, "Cédula inválida");
    } else if (cedulaRepetida(tc.getText())) {
        //Se valida que no se ingrese una cédula repetida tomando datos de SQL
        JOptionPane.showMessageDialog(null, "Existe un usuario registrado con la misma cédula");
    } else {
        String nombreUser = tnom.getText() + " " + tapel.getText();
        String idLastUser = "";
        try {
            //Se consulta la BD para asignar el id del nuevo usuario
            ResultSet res = Conexiones.Conexion.Consulta("select * from USUARIO order by ID_USUARIO");
            while (res.next()) {
                idLastUser = res.getString("ID_USUARIO");
            }
            //Se transforma de string a número para aumentar el id en uno
            int idNextUser = Integer.parseInt(idLastUser);
            idNextUser++;
            //Se vuelve a transformar a string el id para pasar al método "EntradaUsuario",
            //el cual solo acepta strings
            String id = Integer.toString(idNextUser);
            System.out.println("number" + id);
            Procedimientos.EntradaUsuario(id, nombreUser, tc.getText(), pc.getText());
            JOptionPane.showMessageDialog(null, "Usuario ingresado con éxito");
            limpiar();
            //Se actualiza el combo con el nombre del nuevo Usuario
            llenarCombo();
```

2.2 Usuario de caja chica

Dentro la interfaz de usuario también se usó el método para validar que un campo sea numérico para los campos numéricos respectivos.
Además de métodos para vaciar ciertos campos al guardar una factura, solicitud o monto.

2.2.1 Determinar si los campos de la factura están vacíos

Prueba: Cada vez que se dejaba un campo vacío la función retornaba falso.

```
private boolean camposFacturaVacios(){
    boolean vacios=true;
    if(tNumeFact.getText().equals("")||tProveedor.getText().equals("")||tTotal.getText().equals("")
        ||tRuta.getText().equals("")){
        JOptionPane.showMessageDialog(null, "Uno o más campos del cliente están vacíos");
    }else{
        if(tCateg.getText().equals("")||tCantidad.getText().equals("")||tPrecio.getText().equals("")
        ||tDescrip.getText().equals("")){
        JOptionPane.showMessageDialog(null, "Uno o más campos del producto están vacíos");
        }else{
            vacios=false;
        }
    }
    return vacios;
}
```

2.2.2 Buscar solicitud

Prueba 1: Se dejó el campo vació y se mostró un mensaje de que no se ingresó datos.
Prueba 2: Se ingresó caracteres numéricos y salió el mensaje de la solicitud no se encuentra en la base de datos.
Prueba 3: Se ingresó un número de una solicitud válido y se desplegó la información en una tabla.

```
private void bBuscarSolActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    //Buscar la solicitud en la base de datos por medio del id
    String num="";
    String solicitud=JOptionPane.showInputDialog(null, "Ingrese el número de solicitud que busca:");
    if(solicitud.equals("")){
        JOptionPane.showMessageDialog(null, "No se ingresó datos");
    }else{
    String solicitante="";
    String fecha="";
    String departamento="";
    String estado="";
    ResultSet resul = Conexiones.Conexion.Consulta("select * from SOLICITUD where ID_SOLICITUD="+solicitud);
    try {
        while (resul.next()) {
                num=resul.getString("ID_SOLICITUD");//System.out.println()
        }
            if(num.equals(solicitud)){
                //NOMBRE_SOLICITANTE,FECHA_SOLICITUD,DEPARTAMENTO,ESTADO
        resul = Conexiones.Conexion.Consulta("select * from SOLICITUD where ID_SOLICITUD="+solicitud);
        while (resul.next()) {
            //solicitud=resul.getString("ID_SOLICITUD");//System.out.println();
            solicitante=resul.getString("NOMBRE_SOLICITANTE");
            fecha=resul.getString("FECHA_SOLICITUD");
            departamento=resul.getString("DEPARTAMENTO");
            estado=resul.getString("ESTADO");
        }
        DefaultTableModel model = (DefaultTableModel) TablaSolicitud.getModel();
        //Ya que solo existe una solicitud solo se ingresa y se remueve la primera fila
        model.removeRow(0);
        Object solic[] = {solicitud, solicitante,fecha,departamento,estado};
```

2.2.3 Guardar factura

Esta fue la parte que tomó más tiempo ya que se encontraba implícito el guardado de vale de caja chica, la retención, el encabezado de la factura, detalle de factura, el monto y el identificador de caja chica.
Prueba 1: Igual que en los anteriores, se intentó dejar los campos vacíos, pero no se pudo.
Prueba 2: Se intentó guardar sin ingresar el vale y la retención, igual se denegó el ingreso.
Prueba 3: Se intentó un ingresar una factura con el mismo id de otra factura de la base de datos, de igual forma se denegó el ingreso en la base.

```
private void bGuardarFactActionPerformed(java.awt.event.ActionEvent evt) {
    //Guardar la factura en la base de datos
    //Primero se comprueba que el total ingresado en la parte del cliente sea igual al total de la factura
    if(saldo!=Double.parseDouble(tTotal.getText())){
        //System.out.println(saldo);
        JOptionPane.showMessageDialog(null, "El total ingresado no corresponde con el total de la factura");
    }else{
    //Si no se ingresa la retencion, el vale y el identificador de caja chica no se puede ingresar la factura
    if(retencion==false||valecaja==false||IdentificadorCaja.getText().equals("")){
        JOptionPane.showMessageDialog(null, "La retención,el vale de caja y el identificador de caja son campos obligatorios");
    }else{
        String numf="";
        ResultSet resul = Conexiones.Conexion.Consulta
        ("select * from FACTURA where ID_FACTURA="+tNumeFact.getText());
        try {
        while (resul.next()) {
            numf=resul.getString("ID_FACTURA");//System.out.println()
        }
        //Si existe una factura con el mismo id en la base de datos no se puede guardar la factura
        if(numf.equals(tNumeFact.getText())){
        JOptionPane.showMessageDialog(null, "La factura que se quiere ingresar ya existe");
        }else{
        //Para  guardar la factura se comienza ingresando sin los campos not null
        Procedimientos.IngresarFactura(tNumeFact.getText(), tFecha.getText(),tProveedor.getText(),tTotal.getText());
        //Se ingresa el vale de caja
        Procedimientos.IngresarVale(IdVale.getText(), tNumeFact.getText(),Solicitante.getText());
        String idDetalle;
        //Se ingresa la retención del iva y el impuesto a la renta
        Procedimientos.IngresarRetencion(idRetencion.getText(), tNumeFact.getText(),TotalRet.getText());
        //Se agregra los campos not null que no se ingresaron en el inicio de la tabla factura los cuales
        //son la retencion, el vale de caja y el usuario
        Procedimientos.ActualizarFactura(tNumeFact.getText(),idRetencion.getText(),id_usuario
```

2.2.4 Guardar solicitud

Para esta sección solo se restringió que los campos no estuviesen vacíos.

```
private void bGuardarSolActionPerformed(java.awt.event.ActionEvent evt) {
    //Guarda la solicitud de requerimiento de material
    String departamento=(String)cDepar.getSelectedItem();
    String estado=(String) cEstado.getSelectedItem();

    if(tNumSol.getText().equals("")||tNomSol.getText().equals("")||tFechaSol.getText().equals("")
        ||estado.equals("")||departamento.equals("")||tRutaQuipux.getText().equals("")){
        JOptionPane.showMessageDialog(null, "Uno o más campos están vacíos");
    }else{
        String solicitud=tNumSol.getText();
        String solicitante=tNomSol.getText();
        String fecha=tFechaSol.getText();
        String rutaQuipux=tRutaQuipux.getText();
        String idSol="";
        ResultSet resul = Conexiones.Conexion.Consulta
        ("select * from SOLICITUD where ID_SOLICITUD="+solicitud);
        try {
        while (resul.next()) {
            idSol=resul.getString("ID_SOLICITUD");//System.out.println()
        }
        if(solicitud.equals(idSol)){
           JOptionPane.showMessageDialog(null, "La solicitud que se quiere ingresar ya existe");
        }else{
            Procedimientos.IngresarSolicitud(solicitud, solicitante,fecha,departamento,estado);
            JOptionPane.showMessageDialog(null, "Solcitud ingresada con éxito");
            limpiarSolicitud();
        }
        } catch (SQLException ex) {
            Logger.getLogger(InterfazUsuario.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}
```

2.2.5 Generar reportes

Prueba 1: No se ingresó una fecha y el sistema solicitó que se ingrese al menos la primera fecha.
Prueba 2: Se ingresó la fecha inicial y el reporte se generó hasta la fecha actual.
Prueba 3: Se ingresó la fecha inicial y final y el reporte se generó en ese rango de las fechas.

```
private void GenerarReporteActionPerformed(java.awt.event.ActionEvent evt) {
    //Se realiza una consulta de la base de datos uniendo varias tablas para mostrar
    //los campos deseados segun la fecha que el usuario ingrese para mostrar los datos
    //en la tabla de reporte de netbeans
    //También se calcula el total de factura del reporte y total de retencion del reporte
    ResultSet resul;
    String fechai=fechaInicial.getText();
    String fechaf=fechaFinal.getText();
    if(fechai.equals("")){
        JOptionPane.showMessageDialog(null, "Debe ingresar al menos la fecha de inicio");
    }else{
    if(fechaf.equals("")){
        resul = Conexiones.Conexion.Consulta("select  FACTURA.ID_VALE_CAJA_CHICA,"
        + "FACTURA.ID_FACTURA,FECHA_FACTURA, NOMBRE_ENCARGADA,\n" +
        "CATEGORIA_PARTIDA_PRESUPUESTARIA.CODIGO_CATEGORIA,CONCEPTO_PRODUCTO,VALOR_RETENCION,TOTAL\n" +
        "from VALE_CAJA_CHICA, RETENCION_FACTURA,CATEGORIA_PARTIDA_PRESUPUESTARIA,DETALLE_FACTURA,FACTURA\n" +
        "where \n" +
        "DETALLE_FACTURA.ID_FACTURA=FACTURA.ID_FACTURA  \n" +
        "and VALE_CAJA_CHICA.ID_FACTURA=FACTURA.ID_FACTURA \n" +
        "and RETENCION_FACTURA.ID_FACTURA=FACTURA.ID_FACTURA\n" +
        "and CATEGORIA_PARTIDA_PRESUPUESTARIA.CODIGO_CATEGORIA=DETALLE_FACTURA.CODIGO_CATEGORIA\n" +
        "and FECHA_FACTURA>'"+fechai+"' order by FACTURA.ID_FACTURA");
    }else{
        resul = Conexiones.Conexion.Consulta("select  FACTURA.ID_VALE_CAJA_CHICA,"
        + "FACTURA.ID_FACTURA,FECHA_FACTURA, NOMBRE_ENCARGADA,\n" +
        "CATEGORIA_PARTIDA_PRESUPUESTARIA.CODIGO_CATEGORIA,CONCEPTO_PRODUCTO,VALOR_RETENCION,TOTAL\n" +
        "from VALE_CAJA_CHICA, RETENCION_FACTURA,CATEGORIA_PARTIDA_PRESUPUESTARIA,DETALLE_FACTURA,FACTURA\n" +
        "where \n" +
        "DETALLE_FACTURA.ID_FACTURA=FACTURA.ID_FACTURA  \n" +
        "and VALE_CAJA_CHICA.ID_FACTURA=FACTURA.ID_FACTURA \n" +
        "and RETENCION_FACTURA.ID_FACTURA=FACTURA.ID_FACTURA\n" +
```

2.2.6 Reponer monto

Se agregó la restricción de que todos los campos no estén vacíos.

```
private void btReponerActionPerformed(java.awt.event.ActionEvent evt) {
    //Repone el monto de caja chica
    if(txfReponer.getText().equals("")||txfIdentificadorCaja.getText().equals("")
            ||txfFecha.getText().equals("")){
        JOptionPane.showMessageDialog(null, "Uno o más campos están vacíos");
    }else{
            try {
                Procedimientos.Reponer(txfIdentificadorCaja.getText(),txfReponer.getText(), txfFecha.getText());
                ResultSet resul = Conexiones.Conexion.Consulta
                ("select TOP 1 MONTO from caja_chica order by FECHA_MONTO desc");
                while(resul.next()){
                  SaldoCaja.setText(resul.getString("MONTO"));
                }
                txfReponer.setText("");
                txfIdentificadorCaja.setText("");
                txfFecha.setText("");
            } catch (SQLException ex) {
                Logger.getLogger(InterfazUsuario.class.getName()).log(Level.SEVERE, null, ex);
            }

    }
}
```