# MANUAL DEL PROGRAMADOR

## “PaperLand”

# Tabla de contenido

## Contenido

## Introducción

El propósito de este manual del programador, es dar a conocer al lector los códigos fuentes del programa realizado. Para ello tratamos de la forma más concisa de explicar cada uno de los códigos, junto con la programación y configuración utilizada en el desarrollo del software, esto con el fin de que el usuario pueda modificar a gusto alguno de los valores y parámetros de las funciones que se encuentran expuestas en la programación de sistema de administración de papelería "PaperLand"

## Objetivo

El objetivo de este software es ayudar en la gestión de los productos, proveedores y agilizar el proceso de ventas,  así como de llevar un control exhaustivo de las ventas que se realizaron ya sea al día o la semana.

## Entorno de Desarrollo

Para trabajar con el proyecto se necesita tener instalados los siguientes programas y dependencias:

• Java JDK 8.

• NetBeans IDE

• Xampp/ MySql Workbench.

• Git Bash

## Obtención del código Fuente

Para el desarrollo de la aplicación se ha utilizado un repositorio Git hospedado en GitHub. Para obtener una copia de este hay que proceder de la siguiente manera:

1. Abrir la terminal Git Bash.
2. Desplazarse al directorio donde se desee copiar el repositorio (utilizando el comando cd).
3. Introducir el siguiente comando: git clone https://github.com/FranciscoOrantes/mantenimientopapeleria.git
4. Se iniciará la descarga del repositorio, cuando finalice se dispondrá de una copia completa de este.

## Programación

Este proyecto está construido bajo una arquitectura MVC el cual está dividido en cinco partes importantes para el funcionamiento del sistema, estos cinco paquetes funcionan de la siguiente manera:

1. Paquete Modelo
   Se encarga de realizar la conexión a la Base de Datos y hacer todo tipo de consultas que tengan que ver con la bd.
2. Paquete Vista
   En este paquete están guardadas todas las interfaces UI/UX que se muestran al usuario.
3. Paquete Controlador
   Este paquete funciona como el controlador de cada vista y es el encargado de conectar tanto al paquete modelo, cómo al paquete vista
4. Paquete Resources
   Este paquete almacena todos los estilos de diseño e imágenes que utiliza el sistema, archivos css, jpg, png etc.
5. Paquete Main
   Este paquete contiene la clase Main la cual se encarga de iniciar todo el sistema.

Estos paquetes tienen consigo una serie de clases que le dan el funcionamiento al sistema se explicara lo más detallado posible las clases más importantes.

## Librerías

En total fueron 2 librerías las que se utilizaron para el desarrollo de este sistema las cuales son:

• Com.mysql.jdbc_5.1.23.jar: Esta es la encargada de hacer la conexión de base de datos.

• JasperReports.6.0.0.jar: Esta es la encargada de generar los tickets y reportes de ventas.

## Revisión de Clases

### 1. Clase Conexión

La clase conexión, es la encargada de conectarse a la base de datos en la variable URL especificamos la dirección de la base de datos, en las variables username y password, nos sirven para escribir las credenciales de acceso de nuestra base de datos.

```
public class Conexion {
   Connection conectar=null;
   public static final String URL = "jdbc:mysql://localhost:3306/papeleriadb";
   public static final String USERNAME = "root";
   public static final String PASSWORD = "";

    public Connection conectate(){

      try {

         Class.forName("com.mysql.jdbc.Driver");
        conectar= (Connection) DriverManager.getConnection(URL, USERNAME, PASSWORD);

}catch(Exception e){
   System.err.println(e);
    Alert dialogoAlerta = new Alert(Alert.AlertType.ERROR);
   dialogoAlerta.setTitle("");
   dialogoAlerta.setHeaderText("Error con la BD");
   dialogoAlerta.initStyle(StageStyle.UTILITY);
   dialogoAlerta.showAndWait();
}
      return conectar;
    }
}
```

## 2. Clase Inicio de Sesión

La clase inicio de sesión hace una consulta a la base de datos, si el usuario y la contraseña existen permite al usuario avanzar a la siguiente ventana.

```
public class InicioSesion {
    private String usuario;
   private String password;
   static Stage ventanaInicio;
   static FXMLLoader loaderInicioAdmin;
   public void setUsuario(String usuario) {
      this.usuario = usuario;
   }
   public void setPassword(String password) {
      this.password = password;
   }
   public String getUsuario() {
      return usuario;
   }
   public String getPassword() {
      return password;
   }
   public void buscarUsuario() throws SQLException {
      Conexion con = new Conexion();
      Connection st = con.conectate();
      ResultSet rs;

      try {
         Statement execute2 = st.createStatement();
         PreparedStatement pst2 = st.prepareStatement(
               "SELECT * FROM usuario WHERE username= ? And password = ?");
         pst2.setString(1, this.getUsuario());
         pst2.setString(2, this.getPassword());
         rs = pst2.executeQuery();

         if (rs.next()) {
            Alert dialogoAlerta = new Alert(Alert.AlertType.INFORMATION);
            dialogoAlerta.setTitle("Inicio de sesion");
            dialogoAlerta.setHeaderText("Correcto");
            dialogoAlerta.setContentText("Ha iniciado sesión con éxito");
```

```
                dialogoAlerta.initStyle(StageStyle.UTILITY);
                dialogoAlerta.showAndWait();
                loaderInicioAdmin = new FXMLLoader(getClass().getResource("/Vista/ListaProductos.fxml"));
                    Parent root1 = (Parent) loaderInicioAdmin.load();
                    ventanaInicio = new Stage();
                    ventanaInicio.setScene(new Scene(root1));
                    ventanaInicio.setResizable(false);
                    ventanaInicio.show();
            } else {

                Alert dialogoAlerta = new Alert(Alert.AlertType.WARNING);
                dialogoAlerta.setTitle("Advertencia");
                dialogoAlerta.setHeaderText("Datos No Validos");
                dialogoAlerta.setContentText("Usuario o Contraseña incorrecta");
                dialogoAlerta.initStyle(StageStyle.UTILITY);
                dialogoAlerta.showAndWait();
                }
        } catch (Exception e) {
             Alert dialogoAlerta = new Alert(Alert.AlertType.WARNING);
            dialogoAlerta.setTitle("Error");
            dialogoAlerta.setHeaderText("Ha Ocurrido un error con la BD");
            dialogoAlerta.initStyle(StageStyle.UTILITY);
            dialogoAlerta.showAndWait();
        }
        con.conectate().close();
    }
}
```

### 3. Clase LoginController

Esta clase es la encargada de recibir los datos de la interface y comunicarla con la clase InicioSesion del paquete modelo para el inicio de sesión.

```
public class LoginController implements Initializable {
    @FXML
    TextField txtUsuario;
    @FXML
    PasswordField txtPassword;
    @FXML
    Button btnIniciar;
    private String usuario, password;
    static Stage ventanaInicio;
    static FXMLLoader loaderInicioAdmin;

    @Override
    public void initialize(URL url, ResourceBundle rb) {   }

    @FXML
    private void iniciarSesion(ActionEvent event) throws SQLException, IOException {
        usuario = txtUsuario.getText();
        password = txtPassword.getText();
        InicioSesion inicio = new InicioSesion();
        inicio.setUsuario(usuario);
        inicio.setPassword(password);
        if (inicio.buscarUsuario()) {
            loaderInicioAdmin = new FXMLLoader(getClass().getResource("/Vista/Menu.fxml"));
            Parent root1 = (Parent) loaderInicioAdmin.load();
            ventanaInicio = new Stage();
            ventanaInicio.setScene(new Scene(root1));
            ventanaInicio.setResizable(false);
            ventanaInicio.show();
            Node source = (Node) event.getSource();
            Stage stage = (Stage) source.getScene().getWindow();
            stage.close();

        };
    }
}
```

## 4. Clase Usuario

Esta clase es la encargada de hacer registros de nuevos usuarios y guardarlos en la tabla usuario.

```
public class Usuario {
   Conexion con = new Conexion();
   Connection st = con.conectate();
   public void Guardar(String username,String password,String nombre,String ap,String am,String
tipo) throws SQLException{
      Statement execute = st.createStatement();
      PreparedStatement pst = st.prepareStatement(
               "INSERT INTO
usuario(username,password,nombre,apellidoPaterno,apellidoMaterno,tipoUsuario)
VALUES(?,?,?,?,?,?)");
         pst.setString(1, username);
         pst.setString(2, password);
         pst.setString(3, nombre);
         pst.setString(4, ap);
         pst.setString(5, am);
         pst.setString(6, tipo);
           int res = pst.executeUpdate();
         if (res > 0) {
             Alert dialogoAlerta = new Alert(Alert.AlertType.INFORMATION);
   dialogoAlerta.setTitle("Exito");
   dialogoAlerta.setHeaderText("Se han guardado los Datos");
   dialogoAlerta.initStyle(StageStyle.UTILITY);
   dialogoAlerta.showAndWait();
         }
   }
 }
```

## 5. Clase RegistroUsuariosController

Esta clase es la encargada de recibir los datos de la interface y comunicarla con la clase Usuario del paquete modelo para la creación de usuarios.

```
public class RegistroUsuariosController implements Initializable {
   private String username;
   private String nombre;
   private String apellidoP;
   private String apellidoM;
   private String contrasena;
   private String tipoUsuario;
   @FXML
   TextField user,name,ap,am,pass;
   @FXML
   ComboBox comboTipos;
   @FXML
   Button btnRegistrar;
   @Override
   public void initialize(URL url, ResourceBundle rb) {
      btnRegistrar.setDisable(true);
      comboTipos.getItems().addAll(
            "Administrador",
            "Cajero"
      );
      // TODO
   }
 public void seleccionarTipo(){
    try {
       tipoUsuario = comboTipos.getValue().toString();
       btnRegistrar.setDisable(false);
    } catch (Exception e) {
    }

 }
 public void registrar() throws SQLException{

if(name.getText().equals("")||ap.getText().equals("")||am.getText().equals("")||pass.getText().eq
uals(""))
   {
   Alert dialogoAlerta = new Alert(Alert.AlertType.WARNING);
   dialogoAlerta.setTitle("Advertencia");
```

```
    dialogoAlerta.setHeaderText("Campos No validos!");
    dialogoAlerta.initStyle(StageStyle.UTILITY);
    dialogoAlerta.showAndWait();
    }else{
        username =user.getText();
        nombre = name.getText();
        apellidoP=ap.getText();
        apellidoM = am.getText();
        contrasena = pass.getText();

        Usuario a= new Usuario();
        a.Guardar(username, contrasena, nombre, apellidoP, apellidoM, tipoUsuario);
        user.setText("");
        name.setText("");
        ap.setText("");
        am.setText("");
        pass.setText("");
        comboTipos.valueProperty().set(null);
        btnRegistrar.setDisable(true);
    }
}
}
```

## 6. Clase Producto

La clase producto es la encargada de hacer consultas a la base de datos en la tabla productos, esta tiene métodos de get, put, delete, post.

```
public class Producto {
 ArrayList<Integer> cantidades = new ArrayList<Integer>();
  public Producto(){  }

  public String getNombreProducto() {
     return nombreProducto;
  }

  public float getPrecioProducto() {
     return precioProducto;
  }

  public int getCantidadProducto() {
     return cantidadProducto;
  }

  public void setNombreProducto(String nombreProducto) {
     this.nombreProducto = nombreProducto;
  }

  public void setPrecioProducto(float precioProducto) {
     this.precioProducto = precioProducto;
  }

  public void setCantidadProducto(int cantidadProducto) {
     this.cantidadProducto = cantidadProducto;
  }
  public String nombreProducto;
  public  float precioProducto;
  public int cantidadProducto;
  public SimpleIntegerProperty id = new SimpleIntegerProperty();
  public SimpleIntegerProperty cantidad = new SimpleIntegerProperty();
  public SimpleStringProperty nombre = new SimpleStringProperty();
  public SimpleStringProperty descripcion = new SimpleStringProperty();
  public SimpleStringProperty precio = new SimpleStringProperty();
```

```
    public SimpleIntegerProperty folio = new SimpleIntegerProperty();
    public SimpleStringProperty tipo = new SimpleStringProperty();
    public SimpleStringProperty proveedor = new SimpleStringProperty();
    static String getDataFilter = "SELECT producto.id, producto.nombre,producto.cantidad,
producto.descripcion,producto.precio, producto.folio, tipoproducto.tipo,"
            + " proveedor.nombre FROM producto\n"
            + "INNER JOIN tipoproducto on tipoproducto.id = producto.tipo_id\n"
            + "INNER JOIN proveedor on proveedor.id = producto.proveedor_id ";

    public void setCantidad(int cantidad) {
        this.cantidad = new SimpleIntegerProperty(cantidad);
    }
    public Integer getId() {  return id.get();  }
    public Integer getCantidad() { return cantidad.get(); }
    public String getNombre() { return nombre.get(); }
    public String getDescripcion() { return descripcion.get(); }
    public String getPrecio() { return precio.get(); }
    public Integer getFolio() { return folio.get(); }
    public String getTipo() { return tipo.get(); }
    public String getProveedor() { return proveedor.get(); }

    public Producto(Integer id, String nombre, String descripcion, Integer cantidad, String precio,
Integer folio, String tipo, String proveedor) {
        this.id = new SimpleIntegerProperty(id);
        this.nombre = new SimpleStringProperty(nombre);
        this.descripcion = new SimpleStringProperty(descripcion);
        this.precio = new SimpleStringProperty(precio);
        this.folio = new SimpleIntegerProperty(folio);
        this.tipo = new SimpleStringProperty(tipo);
        this.proveedor = new SimpleStringProperty(proveedor);
        this.cantidad = new SimpleIntegerProperty(cantidad);
    }

    public Producto(Integer id, String nombre, String precio, Integer folio,Integer cantidad) {
        this.id = new SimpleIntegerProperty(id);
        this.nombre = new SimpleStringProperty(nombre);
        this.precio = new SimpleStringProperty(precio);
        this.folio = new SimpleIntegerProperty(folio);
        this.cantidad = new SimpleIntegerProperty(cantidad);
        }
    public Producto(String nombre, float precioProducto, int cantidadProducto) {
```

```
        this.nombreProducto = nombre;
        this.precioProducto = precioProducto;
        this.cantidadProducto = cantidadProducto;
        }
    public static void llenarVenta2(List<Producto> lista,Producto
producto,List<String>nombresProductos,List<Integer> cantidades,List<Float> precios){
         producto.setNombreProducto(producto.getNombre());
        producto.setPrecioProducto(Float.parseFloat(producto.getPrecio()));
        producto.setCantidadProducto(producto.getCantidad());
        lista.add(new Producto(
        producto.getNombreProducto(),
        producto.getPrecioProducto(),
        producto.getCantidadProducto()
        ));
        nombresProductos.add(producto.getNombreProducto());
        cantidades.add(producto.getCantidadProducto());
        precios.add(producto.getPrecioProducto());

    }
    public static void llenarVenta(ObservableList<Producto> lista,Producto producto){

        lista.add(new Producto(
        producto.getId(),
        producto.getNombre(),
        producto.getPrecio(),
        producto.getFolio(),
        producto.getCantidad()
        ));


    }
    public static void llenarInfoProductos(ObservableList<Producto> lista) {
        Conexion con = new Conexion();
        Connection st = con.conectate();
        ResultSet rs;

        try {
            Statement execute = st.createStatement();

            PreparedStatement pst = st.prepareStatement(
                "SELECT producto.id, producto.nombre, producto.descripcion,producto.cantidad,
producto.precio, producto.folio,tipoproducto.tipo, proveedor.nombre FROM producto"
```

```
            + " INNER JOIN tipoproducto on tipoproducto.id = producto.tipo_id"
            + " INNER JOIN proveedor on proveedor.id = producto.proveedor_id");
        rs = pst.executeQuery();
        while (rs.next()) {

          lista.add(
              new Producto(
                  rs.getInt("producto.id"),
                  rs.getString("producto.nombre"),
                  rs.getString("producto.descripcion"),
                  rs.getInt("producto.cantidad"),
                  rs.getString("producto.precio"),
                  rs.getInt("producto.folio"),
                  rs.getString("tipoproducto.tipo"),
                  rs.getString("proveedor.nombre")

              )
          );

        }

      } catch (Exception e) {
        System.err.println("excetpcion " + e);

      }
    }

    public static void filtradoPrincipal(String filtrado,ObservableList<Producto> lista,String valor)
throws SQLException {
      if (filtrado.equals("Tipo")) {
        filtradoCategoria(lista,valor);
      }
      if(filtrado.equals("Proveedor")){
        FiltradoProveedores(lista,valor);
      }
      if(filtrado.equals("Precio")){
        filtradoPrecio(lista,valor);
      }
    }

    public static void filtradoCategoria(ObservableList<Producto> lista,String categoria) throws
SQLException {
```

```
        Conexion con = new Conexion();
        Connection st = con.conectate();
        ResultSet rs;
        Statement execute = st.createStatement();
        PreparedStatement pst = st.prepareStatement(getDataFilter + "Where tipoproducto.tipo LIKE
'%" + categoria + "%'");
        rs = pst.executeQuery();
        while (rs.next()) {
           lista.add(
                   new Producto(
                           rs.getInt("producto.id"),
                           rs.getString("producto.nombre"),
                           rs.getString("producto.descripcion"),
                            rs.getInt("producto.cantidad"),
                           rs.getString("producto.precio"),
                           rs.getInt("producto.folio"),
                           rs.getString("tipoproducto.tipo"),
                           rs.getString("proveedor.nombre")
                   )
               );
        }
     }
     public static void filtradoPrecio(ObservableList<Producto> lista,String precioI) throws
SQLException {
        Conexion con = new Conexion();
        Connection st = con.conectate();
        ResultSet rs;
        Statement execute = st.createStatement();
        PreparedStatement pst = st.prepareStatement(getDataFilter + "where producto.precio LIKE
'%" + precioI +"%'");
        rs = pst.executeQuery();
        try {
           while (rs.next()) {
              lista.add(
                   new Producto(
                           rs.getInt("producto.id"),
                           rs.getString("producto.nombre"),
                           rs.getString("producto.descripcion"),
                            rs.getInt("producto.cantidad"),
                           rs.getString("producto.precio"),
                           rs.getInt("producto.folio"),
                           rs.getString("tipoproducto.tipo"),
```

```
                    rs.getString("proveedor.nombre")
                )
            );
          }
        } catch (SQLException e) {
        }
    }
    public static void FiltradoProveedores(ObservableList<Producto> lista,String proveedor) throws
SQLException {
         Conexion con = new Conexion();
        Connection st = con.conectate();
        ResultSet rs;
        Statement execute = st.createStatement();
        PreparedStatement pst = st.prepareStatement(getDataFilter + "Where proveedor.nombre LIKE
'%" + proveedor + "%'");
        rs = pst.executeQuery();
        while (rs.next()) {
            lista.add(
                new Producto(
                    rs.getInt("producto.id"),
                    rs.getString("producto.nombre"),
                    rs.getString("producto.descripcion"),
                     rs.getInt("producto.cantidad"),
                    rs.getString("producto.precio"),
                    rs.getInt("producto.folio"),
                    rs.getString("tipoproducto.tipo"),
                    rs.getString("proveedor.nombre")
                )
            );
          }

    }
    public void updateProduct(int id, String nombre, String descripcion, String precio,int cantidad,
int folio) throws SQLException {
        Conexion con = new Conexion();
        Connection st = con.conectate();

        try {

          Statement execute = st.createStatement();
          PreparedStatement pst = st.prepareStatement("UPDATE producto SET nombre = ?,
descripcion = ?, precio = ?, cantidad = ?, folio = ? WHERE id = ?");
```

```
        pst.setString(1, nombre);
        pst.setString(2, descripcion);
        pst.setString(3, precio);
        pst.setInt(4,cantidad);
        pst.setInt(5, folio);

        pst.setInt(6, id);

        int res = pst.executeUpdate();

        if (res > 0) {
            Alert dialogoAlerta = new Alert(Alert.AlertType.INFORMATION);
            dialogoAlerta.setTitle("Exito");
            dialogoAlerta.setHeaderText("Se han actualizado los Datos");
            dialogoAlerta.initStyle(StageStyle.UTILITY);
            dialogoAlerta.showAndWait();
        }

    } catch (Exception e) {
        System.err.println("EXCEPCION " + e);
        Alert dialogoAlerta = new Alert(Alert.AlertType.ERROR);
        dialogoAlerta.setTitle("Error");
        dialogoAlerta.setHeaderText("Ha ocurrido un error con la BD");
        dialogoAlerta.initStyle(StageStyle.UTILITY);
        dialogoAlerta.showAndWait();
    }
    st.close();
}
 public void updateCantidad(ObservableList<Producto>productos) throws SQLException {
     System.out.println("TAMAÑO DEL OBSERVABLE LIST " +productos.size());
     int contador = productos.size();
     int cantidadActualizar = 0;
     int res = 0;
    Conexion con = new Conexion();
    Connection st = con.conectate();
    ResultSet rs;
    try {
      PreparedStatement pstCantidad = st.prepareStatement(
              "SELECT producto.cantidad FROM producto WHERE id = ? ");
      for (Producto producto1 : productos) {
          pstCantidad.setInt(1, producto1.getId());
       rs = pstCantidad.executeQuery();
```

```
        while(rs.next()){
        cantidades.add(rs.getInt("cantidad"));
        }
       }



        Statement execute = st.createStatement();
        PreparedStatement pst = st.prepareStatement("UPDATE producto SET cantidad = ? WHERE
id = ?");
        for (int i = 0; i < contador; i++) {
           cantidadActualizar = cantidades.get(i)-productos.get(i).getCantidad();
            pst.setInt(1, cantidadActualizar);
         pst.setInt(2, productos.get(i).getId());
         res = pst.executeUpdate();
        }



        if (res > 0) {
           Alert dialogoAlerta = new Alert(Alert.AlertType.INFORMATION);
           dialogoAlerta.setTitle("Exito");
           dialogoAlerta.setHeaderText("Se han actualizado los Datos");
           dialogoAlerta.initStyle(StageStyle.UTILITY);
           dialogoAlerta.showAndWait();
        }

      } catch (Exception e) {
        System.err.println("EXCEPCION " + e);
        Alert dialogoAlerta = new Alert(Alert.AlertType.ERROR);
        dialogoAlerta.setTitle("Error");
        dialogoAlerta.setHeaderText("Ha ocurrido un error con la BD");
        dialogoAlerta.initStyle(StageStyle.UTILITY);
        dialogoAlerta.showAndWait();
      }
      cantidades.clear();
      productos.clear();
      st.close();
    }
     public void deleteProduct(int id) throws SQLException {

      try {
```

```
        Conexion con = new Conexion();
      Connection st = con.conectate();
      ResultSet rs;
      Statement execute = st.createStatement();
        String query1 = "delete from producto " + "where id=" + id;
        execute.executeUpdate(query1);
        System.out.println("Borrado con éxito");
      } catch (SQLException e) {
      }
   }
    public void registrarProducto(String nombreCampo,String descripcionCampo,String
precioCampo,int cantidadCampo,int folioCampo,int tipoProductoCampo,int tipoProvedorCampo){
      Conexion con = new Conexion();
      Connection st = con.conectate();
      try {

        Statement execute = st.createStatement();
        PreparedStatement pst =st.prepareStatement("INSERT INTO
producto(nombre,descripcion,precio,cantidad,folio,tipo_id,proveedor_id) VALUES(?,?,?,?,?,?,?)");
        pst.setString(1, nombreCampo);
        pst.setString(2, descripcionCampo);
        pst.setString(3, precioCampo);
        pst.setInt(4, cantidadCampo);
        pst.setInt(5, folioCampo);
        pst.setInt(6, tipoProductoCampo);
        pst.setInt(7, tipoProvedorCampo);
        pst.executeUpdate();
        Alert dialogoAlerta = new Alert(Alert.AlertType.INFORMATION);
          dialogoAlerta.setTitle("Registro de productos");
          dialogoAlerta.setHeaderText("Correcto");
          dialogoAlerta.setContentText("Ha registrado un producto con éxito");
          dialogoAlerta.initStyle(StageStyle.UTILITY);
          dialogoAlerta.showAndWait();
      } catch (Exception e) {
        System.err.println("Error "+e);
      }
      }
   public static void registrarVentas(ObservableList<Producto>ventas,String fecha,int usuario_id)
throws SQLException{
    Conexion con = new Conexion();
    SecureRandom random = new SecureRandom();
 String folio = new BigInteger(130, random).toString(32);
```

```
    float monto =0;
      Connection st = con.conectate();
      try {
        Statement execute = st.createStatement();
      PreparedStatement pst =st.prepareStatement("INSERT INTO
venta(producto_id,cantidad,monto,fecha,folio,usuario_id) VALUES(?,?,?,?,?,?)");
      for (Producto producto1 : ventas) {
        monto = producto1.getCantidad()*(Float.parseFloat(producto1.getPrecio()));
        System.out.println("Monto " + monto);
        pst.setInt(1, producto1.getId());
        pst.setInt(2, producto1.getCantidad());
        pst.setFloat(3, monto);
        pst.setString(4, fecha);
        pst.setString(5, folio);
        pst.setInt(6, usuario_id);
        pst.executeUpdate();
      }
      }catch(Exception e){
      System.err.println("ERROR " + e);
      }
      }
    public float sumaTotalDia(String fecha) throws SQLException {
      float total = 0;
      Conexion con = new Conexion();
      Connection st = con.conectate();
      ResultSet rs = null;
      try {
        PreparedStatement pstVenta = st.prepareStatement(
              "SELECT SUM(venta.monto)as suma FROM venta WHERE fecha = ? ");
       pstVenta.setString(1, fecha);
        rs = pstVenta.executeQuery();
        if(rs.next()){
        total = rs.getFloat("suma");
        }
      }catch(Exception e){

      }
      return total;
          }
    }
```

## 7. Clase ListaProductosController

Esta clase se encarga de mostrar los productos, así como también de gestionar las actualizaciones y las búsquedas del mismo.

```
public class ListaProductosController implements Initializable {
   // Declaramos los botones
   @FXML private Button anadirBT;
   @FXML private Button modificarBT;
   @FXML private Button eliminarBT;
    // Declaramos los textfileds
   @FXML private TextField nombreTF;
   @FXML private TextField descripcionTF;
   @FXML private TextField precioTF;
   @FXML private TextField folioTF;
   @FXML private TextField tipoTF;
   @FXML private TextField proveedorTF;
   @FXML private TextField indiceBusqueda;
    // Declaramos la tabla y las columnas
   @FXML private TableView<Producto> tablaProducto;
   @FXML private TableColumn nombreCol;
   @FXML private TableColumn descripcionCol;
   @FXML private TableColumn precioCol;
   @FXML private TableColumn folioCol;
   @FXML private TableColumn tipoCol;
   @FXML private TableColumn proveedorCol;
   @FXML private ComboBox filtrado;
   private ObservableList<Producto> productos;
   private int posicionProductoEnTabla;
   private String opcionFiltro,nombre,precio,descripcion;
   private int id,folio;
   static Stage ventanaInicio;
   static FXMLLoader loaderInicioAdmin;
   @FXML private void anadir(ActionEvent event) throws IOException{

      loaderInicioAdmin = new
FXMLLoader(getClass().getResource("/Vista/RegistroProducto.fxml"));
            Parent root1 = (Parent) loaderInicioAdmin.load();
            ventanaInicio = new Stage();
            ventanaInicio.setScene(new Scene(root1));
            ventanaInicio.setResizable(false);
            ventanaInicio.show();
```

```
}
@FXML private void eliminar(ActionEvent event) throws SQLException{
   Producto product = new Producto();
   product.deleteProduct(id);
   inicializarTablaProductos();

}
@FXML private void modificar(ActionEvent event) throws SQLException{
      nombre = nombreTF.getText();
      precio = precioTF.getText();
      descripcion = descripcionTF.getText();
      folio = Integer.valueOf(folioTF.getText());
   Producto product = new Producto();

   product.updateProduct(id,nombre,descripcion,precio,folio);
   nombreTF.clear();
  descripcionTF.clear();
  precioTF.clear();
  folioTF.clear();
   inicializarTablaProductos();
}
private final ListChangeListener<Producto> selectorTablaProductos =
      new ListChangeListener<Producto>() {
         @Override
         public void onChanged(ListChangeListener.Change<? extends Producto> c) {
            ponerProductoSeleccionada();
         }
      };

public Producto getTablaProductosSeleccionada() {
   if (tablaProducto != null) {
      List<Producto> tabla = tablaProducto.getSelectionModel().getSelectedItems();
      if (tabla.size() == 1) {
         final Producto competicionSeleccionada = tabla.get(0);
         return competicionSeleccionada;
      }
   }
   return null;
}
 public void ponerProductoSeleccionada() {
```

```
        final Producto producto = getTablaProductosSeleccionada();
        posicionProductoEnTabla = productos.indexOf(producto);

        if (producto != null) {

            // Pongo los textFields con los datos correspondientes
            nombreTF.setText(producto.getNombre());
            descripcionTF.setText(producto.getDescripcion());
            precioTF.setText(String.valueOf(producto.getPrecio()));
            folioTF.setText(String.valueOf(producto.getFolio()));
            id=producto.getId();

            // Pongo los botones en su estado correspondiente
            modificarBT.setDisable(false);
            eliminarBT.setDisable(false);
            anadirBT.setDisable(true);
            nombreTF.setEditable(true);
            descripcionTF.setEditable(true);
            precioTF.setEditable(true);
            folioTF.setEditable(true);
        }
    }
     public void inicializarTablaProductos() {
         productos = FXCollections.observableArrayList();
         Producto.llenarInfoProductos(productos);
         tablaProducto.setItems(productos);
        nombreCol.setCellValueFactory(new PropertyValueFactory<Producto, String>("nombre"));
        descripcionCol.setCellValueFactory(new PropertyValueFactory<Producto,
String>("descripcion"));
        precioCol.setCellValueFactory(new PropertyValueFactory<Producto, Float>("precio"));
        folioCol.setCellValueFactory(new PropertyValueFactory<Producto, Integer>("folio"));
        tipoCol.setCellValueFactory(new PropertyValueFactory<Producto, String>("tipo"));
        proveedorCol.setCellValueFactory(new PropertyValueFactory<Producto, String>("proveedor"));

        nombreTF.setEditable(false);
       descripcionTF.setEditable(false);
       precioTF.setEditable(false);
       folioTF.setEditable(false);

    }

    @Override
```

```
    public void initialize(URL url, ResourceBundle rb) {
      productos = FXCollections.observableArrayList();
      this.inicializarTablaProductos();
      modificarBT.setDisable(true);
      eliminarBT.setDisable(true);
       final ObservableList<Producto> tablaProductoSel =
tablaProducto.getSelectionModel().getSelectedItems();
       tablaProductoSel.addListener(selectorTablaProductos);
       filtrado.getItems().addAll(
            "Tipo",
            "Proveedor",
            "Precio"
       );
    }
    public void limpiarSeleccion(){
    tablaProducto.getSelectionModel().clearSelection();
    modificarBT.setDisable(true);
      eliminarBT.setDisable(true);
      anadirBT.setDisable(false);
      nombreTF.setEditable(false);
      descripcionTF.setEditable(false);
      precioTF.setEditable(false);
      folioTF.setEditable(false);
      nombreTF.clear();
      descripcionTF.clear();
      precioTF.clear();
      folioTF.clear();
    }
    public void getFilter(){
       opcionFiltro = filtrado.getValue().toString();
    }
    public void buscarProducto() throws SQLException {
       String valor= indiceBusqueda.getText();
         productos = FXCollections.observableArrayList();
         Producto.filtradoPrincipal(opcionFiltro,productos,valor);
         tablaProducto.setItems(productos);
       nombreCol.setCellValueFactory(new PropertyValueFactory<Producto, String>("nombre"));
       descripcionCol.setCellValueFactory(new PropertyValueFactory<Producto,
String>("descripcion"));
       precioCol.setCellValueFactory(new PropertyValueFactory<Producto, String>("precio"));
       folioCol.setCellValueFactory(new PropertyValueFactory<Producto, Integer>("folio"));
       tipoCol.setCellValueFactory(new PropertyValueFactory<Producto, String>("tipo"));
```

```
        proveedorCol.setCellValueFactory(new PropertyValueFactory<Producto, String>("proveedor"));

        nombreTF.setEditable(false);
        descripcionTF.setEditable(false);
        precioTF.setEditable(false);
        folioTF.setEditable(false);
        filtrado.valueProperty().set(null);

    }
}
```

## 8. Clase ProductoController

Esta clase se encarga de conectarse a la base de datos para registrar productos y para obtener los datos del proveedor y los tipos de productos.

```
public class ProductoController implements Initializable {

    private String nombreCampo;
    private String descripcionCampo;
    private String precioCampo;
    private int cantidadCampo;
    private int folioCampo;
    private int tipoProductoCampo;
    private int tipoProvedorCampo;
    ArrayList<String> lista1 = new ArrayList<String>();
    ArrayList<String> lista2 = new ArrayList<String>();
    @FXML
    private TextField nombre, descripcion, precio, cantidad, folio;
    @FXML
    private ComboBox tipoProducto, tipoProvedor;

    /**
     * Initializes the controller class.
     */
    @Override
    public void initialize(URL url, ResourceBundle rb) {

        llenarDatosProvedor();
        llenarDatosTiposProductos();

    }

    @FXML
    private void registrarProducto(ActionEvent event) {
        nombreCampo = nombre.getText();
        descripcionCampo = descripcion.getText();
        precioCampo = precio.getText();
        cantidadCampo = Integer.valueOf(cantidad.getText());
        folioCampo = Integer.valueOf(folio.getText());
        String var = (String) tipoProducto.getValue();
        String[] dato = var.split(" ");

        String var1 = (String) tipoProvedor.getValue();
        String[] dato1 = var1.split(" ");
        tipoProductoCampo = Integer.valueOf(dato[0]);
```

```
        tipoProvedorCampo = Integer.valueOf(dato1[0]);
        Producto a = new Producto();

        a.registrarProducto(nombreCampo, descripcionCampo, precioCampo, cantidadCampo,
folioCampo, tipoProductoCampo, tipoProvedorCampo);
        nombre.setText("");
        descripcion.setText("");
        folio.setText("");
        precio.setText("");
        cantidad.setText("");
        Node source = (Node) event.getSource();
        Stage stage = (Stage) source.getScene().getWindow();
        stage.close();

    }

    public void llenarDatosProvedor() {
        Conexion con = new Conexion();
        Connection st = con.conectate();

        ResultSet rs;

        try {
            Statement execute = st.createStatement();
            PreparedStatement pst = st.prepareStatement("SELECT * FROM proveedor");
            rs = pst.executeQuery();
            while (rs.next()) {
                lista1.add(rs.getString("id") + " " + rs.getString("nombre"));
            };
            tipoProvedor.getItems().addAll(lista1);
        } catch (Exception e) {
            System.err.println("Error Llenar datos");
        }

    }

    public void llenarDatosTiposProductos() {
        Conexion con = new Conexion();
        Connection st = con.conectate();

        ResultSet rs;

        try {
            Statement execute = st.createStatement();
            PreparedStatement pst = st.prepareStatement("SELECT * FROM tipoproducto");
            rs = pst.executeQuery();
            while (rs.next()) {
                lista2.add(rs.getString("id") + " " + rs.getString("tipo"));
```

```
            };
            tipoProducto.getItems().addAll(lista2);
        } catch (Exception e) {
            System.err.println("Error Llenar datos");
        }
    }

    @FXML
    private void cerrar(ActionEvent event) {

        Node source = (Node) event.getSource();
        Stage stage = (Stage) source.getScene().getWindow();
        stage.close();
    }
}
```

## 9. Clase Proveedor

La clase Proveedor es la encargada de hacer consultas a la base de datos en la tabla proveedor, esta tiene métodos de get, put, delete y post, así también los métodos de filtrado de proveedores por nombre, teléfono o empresa.

```
public class Proveedor {

   public Proveedor() {
   }

   public SimpleIntegerProperty id = new SimpleIntegerProperty();
   public SimpleStringProperty nombre = new SimpleStringProperty();
   public SimpleStringProperty apellidoPaterno = new SimpleStringProperty();
   public SimpleStringProperty apellidoMaterno = new SimpleStringProperty();
   public SimpleStringProperty telefono = new SimpleStringProperty();
   public SimpleStringProperty direccion = new SimpleStringProperty();
   public SimpleStringProperty correo = new SimpleStringProperty();
   public SimpleStringProperty empresa = new SimpleStringProperty();
   static String getDataFilter = "SELECT * FROM proveedor ";

   public Integer getId() { return id.get();  }
   public String getNombre() { return nombre.get(); }
   public String getApellidoPaterno() {  return apellidoPaterno.get(); }
   public String getApellidoMaterno() {  return apellidoMaterno.get();  }
   public String getTelefono() { return telefono.get();  }
   public String getDireccion() { return direccion.get();  }
   public String getCorreo() {  return correo.get(); }
   public String getEmpresa() { return empresa.get(); }

   public Proveedor(Integer id, String nombre, String apellidoPaterno, String apellidoMaterno,
String telefono, String direccion, String correo, String empresa) {
      this.id = new SimpleIntegerProperty(id);
      this.nombre = new SimpleStringProperty(nombre);
      this.apellidoPaterno = new SimpleStringProperty(apellidoPaterno);
      this.apellidoMaterno = new SimpleStringProperty(apellidoMaterno);
      this.telefono = new SimpleStringProperty(telefono);
      this.direccion = new SimpleStringProperty(direccion);
      this.correo = new SimpleStringProperty(correo);
      this.empresa = new SimpleStringProperty(empresa);
   }

   public static void llenarInfoProveedores(ObservableList<Proveedor> lista) {
      Conexion con = new Conexion();
      Connection st = con.conectate();
      ResultSet rs;
```

```
        try {
            Statement execute = st.createStatement();
            PreparedStatement pst = st.prepareStatement(
                    "SELECT * FROM proveedor");
            rs = pst.executeQuery();
            while (rs.next()) {
                lista.add(
                        new Proveedor(
                                rs.getInt("proveedor.id"),
                                rs.getString("proveedor.nombre"),
                                rs.getString("proveedor.apellidoPaterno"),
                                rs.getString("proveedor.apellidoMaterno"),
                                rs.getString("proveedor.telefono"),
                                rs.getString("proveedor.direccion"),
                                rs.getString("proveedor.correo"),
                                rs.getString("proveedor.empresa")
                        )
                );
            }

        } catch (Exception e) {
            System.err.println("excetpcion " + e);
        }
    }

    public static void filtradoPrincipal(String filtrado, ObservableList<Proveedor> lista, String valor)
throws SQLException {
        System.out.println("el valor fue " + filtrado);
        if (filtrado.equals("Nombre")) {
            filtradoNombre(lista, valor);
        }
        if (filtrado.equals("Telefono")) {
            filtradoTelefono(lista, valor);
        }
        if (filtrado.equals("Empresa")) {
            filtradoEmpresa(lista, valor);
        }
    }

    public static void filtradoNombre(ObservableList<Proveedor> lista, String nombre) throws
SQLException {
        Conexion con = new Conexion();
        Connection st = con.conectate();
        ResultSet rs;
        Statement execute = st.createStatement();
        PreparedStatement pst = st.prepareStatement(getDataFilter + "WHERE proveedor.nombre
LIKE '%" + nombre + "%'");
        rs = pst.executeQuery();
```

```
        while (rs.next()) {
            lista.add(
                    new Proveedor(
                            rs.getInt("proveedor.id"),
                            rs.getString("proveedor.nombre"),
                            rs.getString("proveedor.apellidoPaterno"),
                            rs.getString("proveedor.apellidoMaterno"),
                            rs.getString("proveedor.telefono"),
                            rs.getString("proveedor.direccion"),
                            rs.getString("proveedor.correo"),
                            rs.getString("proveedor.empresa")
                    )
            );
        }
    }

    public static void filtradoTelefono(ObservableList<Proveedor> lista, String telefonoI) throws
SQLException {
        Conexion con = new Conexion();
        Connection st = con.conectate();
        ResultSet rs;
        Statement execute = st.createStatement();
        PreparedStatement pst = st.prepareStatement(getDataFilter + "where proveedor.telefono LIKE
'%" + telefonoI + "%'");
        rs = pst.executeQuery();
        try {
            while (rs.next()) {
                lista.add(
                        new Proveedor(
                                rs.getInt("proveedor.id"),
                                rs.getString("proveedor.nombre"),
                                rs.getString("proveedor.apellidoPaterno"),
                                rs.getString("proveedor.apellidoMaterno"),
                                rs.getString("proveedor.telefono"),
                                rs.getString("proveedor.direccion"),
                                rs.getString("proveedor.correo"),
                                rs.getString("proveedor.empresa")
                        )
                );
            }
        } catch (SQLException e) {
        }
    }

    public static void filtradoEmpresa(ObservableList<Proveedor> lista, String empresa) throws
SQLException {
        Conexion con = new Conexion();
        Connection st = con.conectate();
```

```
        ResultSet rs;
        Statement execute = st.createStatement();
        PreparedStatement pst = st.prepareStatement(getDataFilter + "Where proveedor.empresa
LIKE '%" + empresa + "%'");
        rs = pst.executeQuery();
        while (rs.next()) {
            lista.add(
                new Proveedor(
                    rs.getInt("proveedor.id"),
                    rs.getString("proveedor.nombre"),
                    rs.getString("proveedor.apellidoPaterno"),
                    rs.getString("proveedor.apellidoMaterno"),
                    rs.getString("proveedor.telefono"),
                    rs.getString("proveedor.direccion"),
                    rs.getString("proveedor.correo"),
                    rs.getString("proveedor.empresa")
                )
            );
        }
    }

    public void updateProveedor(int id, String nombre, String apellidoPaterno, String
apellidoMaterno, String telefono, String direccion, String correo, String empresa) throws
SQLException {
        Conexion con = new Conexion();
        Connection st = con.conectate();

        try {
            Statement execute = st.createStatement();
            PreparedStatement pst = st.prepareStatement("UPDATE proveedor SET nombre = ?,
apellidoPaterno = ?, apellidoMaterno = ?, telefono = ?, direccion = ?, correo = ?, empresa = ?
WHERE id = ?");

            pst.setString(1, nombre);
            pst.setString(2, apellidoPaterno);
            pst.setString(3, apellidoMaterno);
            pst.setString(4, telefono);
            pst.setString(5, direccion);
            pst.setString(6, correo);
            pst.setString(7, empresa);
            pst.setInt(8, id);

            int res = pst.executeUpdate();

            if (res > 0) {
                Alert dialogoAlerta = new Alert(Alert.AlertType.INFORMATION);
                dialogoAlerta.setTitle("Exito");
                dialogoAlerta.setHeaderText("Se han actualizado los Datos");
```

```
            dialogoAlerta.initStyle(StageStyle.UTILITY);
            dialogoAlerta.showAndWait();
          }

      } catch (Exception e) {
          System.err.println("EXCEPCION " + e);
          Alert dialogoAlerta = new Alert(Alert.AlertType.ERROR);
          dialogoAlerta.setTitle("Error");
          dialogoAlerta.setHeaderText("Ha ocurrido un error con la Base de Datos");
          dialogoAlerta.initStyle(StageStyle.UTILITY);
          dialogoAlerta.showAndWait();
      }
      st.close();
   }

   public void deleteProveedor(int id) throws SQLException {
      try {
          Conexion con = new Conexion();
          Connection st = con.conectate();
          ResultSet rs;
          Statement execute = st.createStatement();
          String query1 = "delete from proveedor " + "where id=" + id;
          execute.executeUpdate(query1);
          System.out.println("Borrado con éxito");
      } catch (SQLException e) {
      }
   }

   public void registrarProveedor(String nombreCampo, String apellidoPaternoCampo, String
apellidoMaternoCampo, String telefonoCampo, String empresaCampo, String correoCampo, String
direccionCampo) {
      Conexion con = new Conexion();
      Connection st = con.conectate();
      try {
          Statement execute = st.createStatement();
          PreparedStatement pst = st.prepareStatement("INSERT INTO
proveedor(nombre,apellidoPaterno,apellidoMaterno,telefono,direccion,correo,empresa)
VALUES(?,?,?,?,?,?,?)");
          pst.setString(1, nombreCampo);
          pst.setString(2, apellidoPaternoCampo);
          pst.setString(3, apellidoMaternoCampo);
          pst.setString(4, telefonoCampo);
          pst.setString(5, direccionCampo);
          pst.setString(6, correoCampo);
          pst.setString(7, empresaCampo);
          pst.executeUpdate();
          Alert dialogoAlerta = new Alert(Alert.AlertType.INFORMATION);
          dialogoAlerta.setTitle("Registro de productos");
```

```
            dialogoAlerta.setHeaderText("Correcto");
            dialogoAlerta.setContentText("Ha registrado un proveedor con éxito");
            dialogoAlerta.initStyle(StageStyle.UTILITY);
            dialogoAlerta.showAndWait();
        } catch (Exception e) {
            System.err.println("Error " + e);
        }
    }
}
```

## 10. Clase ListaProveedoresController

Esta clase se encarga de mostrar los proveedores, así como también de gestionar las actualizaciones y las búsquedas del mismo.

```
public class ListaProveedoresController implements Initializable {

    // Declaramos los botones
    @FXML
    private Button anadirBT;
    @FXML
    private Button modificarBT;
    @FXML
    private Button eliminarBT;
    // Declaramos los textfileds
    @FXML
    private TextField nombreTF;
    @FXML
    private TextField apellidoPTF;
    @FXML
    private TextField apellidoMTF;
    @FXML
    private TextField telefonoTF;
    @FXML
    private TextField direccionTF;
    @FXML
    private TextField correoTF;
    @FXML
    private TextField empresaTF;
    @FXML
    private TextField indiceBusqueda;
    // Declaramos la tabla y las columnas
    @FXML
    private TableView<Proveedor> tablaProveedor;
    @FXML
    private TableColumn nombreCol;
    @FXML
    private TableColumn apellidoPaternoCol;
    @FXML
    private TableColumn apellidoMaternoCol;
    @FXML
    private TableColumn telefonoCol;
    @FXML
    private TableColumn direccionCol;
    @FXML
```

```
    private TableColumn correoCol;
    @FXML
    private TableColumn empresaCol;
    @FXML
    private ComboBox filtrado;
    private ObservableList<Proveedor> proveedores;
    private int posicionProveedorEnTabla;
    private String nombre, apellidoP, apellidoM, telefono, direccion, correo, empresa;
    private String opFiltro = "";
    private int id;
    static Stage ventanaInicio;
    static FXMLLoader loaderInicioAdmin;

    @FXML
    private void anadir(ActionEvent event) throws IOException {

        loaderInicioAdmin = new
FXMLLoader(getClass().getResource("/Vista/RegistroProveedor.fxml"));
        Parent root1 = (Parent) loaderInicioAdmin.load();
        ventanaInicio = new Stage();
        ventanaInicio.setScene(new Scene(root1));
        ventanaInicio.setResizable(false);
        ventanaInicio.show();
    }

    @FXML
    private void eliminar(ActionEvent event) throws SQLException {
        Proveedor proveedor = new Proveedor();
        proveedor.deleteProveedor(id);
        inicializarTablaProveedor();

    }

    @FXML
    private void modificar(ActionEvent event) throws SQLException {
        nombre = nombreTF.getText();
        apellidoP = apellidoPTF.getText();
        apellidoM = apellidoMTF.getText();
        telefono = telefonoTF.getText();
        direccion = direccionTF.getText();
        correo = correoTF.getText();
        empresa = empresaTF.getText();

        Proveedor proveedor = new Proveedor();

        proveedor.updateProveedor(id, nombre, apellidoP, apellidoM, telefono, direccion, correo,
empresa);
        nombreTF.clear();
```

```
    apellidoPTF.clear();
    apellidoMTF.clear();
    telefonoTF.clear();
    direccionTF.clear();
    correoTF.clear();
    empresaTF.clear();
    inicializarTablaProveedor();
}
private final ListChangeListener<Proveedor> selectorTablaProveedores
        = new ListChangeListener<Proveedor>() {
    @Override
    public void onChanged(ListChangeListener.Change<? extends Proveedor> c) {
        ponerProveedorSeleccionado();
    }
};

public Proveedor getTablaProveedorSeleccionado() {
    if (tablaProveedor != null) {
        List<Proveedor> tabla = tablaProveedor.getSelectionModel().getSelectedItems();
        if (tabla.size() == 1) {
            final Proveedor competicionSeleccionado = tabla.get(0);
            return competicionSeleccionado;
        }
    }
    return null;
}

public void ponerProveedorSeleccionado() {

    final Proveedor proveedor = getTablaProveedorSeleccionado();
    posicionProveedorEnTabla = proveedores.indexOf(proveedor);

    if (proveedor != null) {

        // Pongo los textFields con los datos correspondientes
        nombreTF.setText(proveedor.getNombre());
        apellidoPTF.setText(proveedor.getApellidoPaterno());
        apellidoMTF.setText(proveedor.getApellidoMaterno());
        telefonoTF.setText(proveedor.getTelefono());
        direccionTF.setText(proveedor.getDireccion());
        correoTF.setText(proveedor.getCorreo());
        empresaTF.setText(proveedor.getEmpresa());
        id = proveedor.getId();

        // Pongo los botones en su estado correspondiente
        modificarBT.setDisable(false);
        eliminarBT.setDisable(false);
        anadirBT.setDisable(true);
```

```
            nombreTF.setEditable(true);
            apellidoPTF.setEditable(true);
            apellidoMTF.setEditable(true);
            telefonoTF.setEditable(true);
            direccionTF.setEditable(true);
            correoTF.setEditable(true);
            empresaTF.setEditable(true);

        }
    }

    public void inicializarTablaProveedor() {
        proveedores = FXCollections.observableArrayList();
        Proveedor.llenarInfoProveedores(proveedores);
        tablaProveedor.setItems(proveedores);
        nombreCol.setCellValueFactory(new PropertyValueFactory<Proveedor, String>("nombre"));
        apellidoPaternoCol.setCellValueFactory(new PropertyValueFactory<Proveedor,
String>("apellidoPaterno"));
        apellidoMaternoCol.setCellValueFactory(new PropertyValueFactory<Proveedor,
String>("apellidoMaterno"));
        telefonoCol.setCellValueFactory(new PropertyValueFactory<Proveedor, String>("telefono"));
        direccionCol.setCellValueFactory(new PropertyValueFactory<Proveedor, String>("direccion"));
        correoCol.setCellValueFactory(new PropertyValueFactory<Proveedor, String>("correo"));
        empresaCol.setCellValueFactory(new PropertyValueFactory<Proveedor, String>("empresa"));

        nombreTF.setEditable(false);
        apellidoPTF.setEditable(false);
        apellidoMTF.setEditable(false);
        telefonoTF.setEditable(false);
        direccionTF.setEditable(false);
        correoTF.setEditable(false);
        empresaTF.setEditable(false);

    }

    @Override
    public void initialize(URL url, ResourceBundle rb) {

        proveedores = FXCollections.observableArrayList();
        this.inicializarTablaProveedor();
        modificarBT.setDisable(true);
        eliminarBT.setDisable(true);
        final ObservableList<Proveedor> tablaProveedorSel =
tablaProveedor.getSelectionModel().getSelectedItems();
        tablaProveedorSel.addListener(selectorTablaProveedores);
        filtrado.getItems().addAll(
                "Nombre",
                "Telefono",
```

```
            "Empresa"
        );
    }

    public void limpiarSeleccion() {
        tablaProveedor.getSelectionModel().clearSelection();
        modificarBT.setDisable(true);
        eliminarBT.setDisable(true);
        anadirBT.setDisable(false);
        nombreTF.setEditable(false);
        apellidoPTF.setEditable(false);
        apellidoMTF.setEditable(false);
        telefonoTF.setEditable(false);
        direccionTF.setEditable(false);
        correoTF.setEditable(false);
        empresaTF.setEditable(false);
        nombreTF.clear();
        apellidoPTF.clear();
        apellidoMTF.clear();
        telefonoTF.clear();
        direccionTF.clear();
        correoTF.clear();
        empresaTF.clear();

    }

    public void getFilter() {

        try {
            opFiltro = filtrado.getValue().toString();
        } catch (Exception e) {

        }
    }

    public void buscarProveedor() throws SQLException {
        String valor = indiceBusqueda.getText();
        proveedores = FXCollections.observableArrayList();
        Proveedor.filtradoPrincipal(opFiltro, proveedores, valor);
        tablaProveedor.setItems(proveedores);
        nombreCol.setCellValueFactory(new PropertyValueFactory<Proveedor, String>("nombre"));
        apellidoPaternoCol.setCellValueFactory(new PropertyValueFactory<Proveedor,
String>("apellidoPaterno"));
        apellidoMaternoCol.setCellValueFactory(new PropertyValueFactory<Proveedor,
String>("apellidoMaterno"));
        telefonoCol.setCellValueFactory(new PropertyValueFactory<Proveedor, String>("telefono"));
        direccionCol.setCellValueFactory(new PropertyValueFactory<Proveedor, String>("direccion"));
        correoCol.setCellValueFactory(new PropertyValueFactory<Proveedor, String>("correo"));
```

```
        empresaCol.setCellValueFactory(new PropertyValueFactory<Proveedor, String>("empresa"));

        nombreTF.setEditable(false);
        apellidoPTF.setEditable(false);
        apellidoMTF.setEditable(false);
        telefonoTF.setEditable(false);
        direccionTF.setEditable(false);
        correoTF.setEditable(false);
        empresaTF.setEditable(false);
        filtrado.valueProperty().set(null);

    }
}
```

## 11. Clase ProveedorController

Esta clase se encarga de obtener los datos para el registro de proveedores.

```
public class ProveedorController {
 private String nombreCampo, apellidoPaternoCampo, apellidoMaternoCampo, direccionCampo,
correoCampo, empresaCampo;
   private String telefonoCampo;
   @FXML
   private TextField nombre, apellidoPaterno, apellidoMaterno, telefono, empresa, correo,
direccion;

   Proveedor proveedor = new Proveedor();
   private void delete(int id) throws SQLException {
      proveedor.deleteProveedor(id);
   }

   @FXML
   private void registrarProveedor(ActionEvent event) {
      nombreCampo = nombre.getText();
      apellidoPaternoCampo = apellidoPaterno.getText();
      apellidoMaternoCampo = apellidoMaterno.getText();
      telefonoCampo = telefono.getText();
      empresaCampo = empresa.getText();
      correoCampo = correo.getText();
      direccionCampo = direccion.getText();
      Proveedor a = new Proveedor();
      a.registrarProveedor(nombreCampo, apellidoPaternoCampo, apellidoMaternoCampo,
telefonoCampo, empresaCampo, correoCampo, direccionCampo);
      limpiar();
   }

   public void limpiar() {
      nombre.setText("");
      apellidoPaterno.setText("");
      apellidoMaterno.setText("");
      telefono.setText("");
      empresa.setText("");
      correo.setText("");
      direccion.setText("");
   }

   @FXML
   private void cerrar(ActionEvent event) {
      Node source = (Node) event.getSource();
      Stage stage = (Stage) source.getScene().getWindow();
      stage.close();
   } }
```

## 12. Clase Reporte

La clase Reporte es la encargada de hacer consultas a la base de datos en la tabla reportes, para generar ticket de venta y realizar el corte de caja .

```
public class Reporte {

   public void generarTicketVenta(String fecha, float total, List<String> ventas2, List<Integer>
cantidades, List<Float> precios) {
      JRDataSource jr = new JREmptyDataSource();
      try {
         JasperReport ticket = null;
         String path = "/reportes/ticket.jasper";

         Map parametro = new HashMap();
         parametro.put("fecha", fecha);
         parametro.put("total", total);
         parametro.put("productos", ventas2);
         parametro.put("cantidades", cantidades);
         parametro.put("precios", precios);
         parametro.put("nombreCajero", VentasController.nombreCajero);
         ticket = (JasperReport) JRLoader.loadObject(getClass().getResource(path));

         JasperPrint jprint = JasperFillManager.fillReport(ticket, parametro, jr);

         JasperViewer view = new JasperViewer(jprint, false);

         view.setDefaultCloseOperation(DISPOSE_ON_CLOSE);

         view.setVisible(true);
      } catch (Exception e) {
         System.err.println("Heey " + e);

      }
   }

   public void generarCorteCaja(String fecha, float total) {
      Conexion con = new Conexion();
      Connection st = con.conectate();
      try {
         JasperReport ticket = null;
         String path = "/reportes/cortecaja.jasper";

         Map parametro = new HashMap();
         parametro.put("fecha", fecha);
         parametro.put("total", total);
```

```
            parametro.put("nombreCajero", VentasController.nombreCajero);
            ticket = (JasperReport) JRLoader.loadObject(getClass().getResource(path));

            JasperPrint jprint = JasperFillManager.fillReport(ticket, parametro, st);

            JasperViewer view = new JasperViewer(jprint, false);

            view.setDefaultCloseOperation(DISPOSE_ON_CLOSE);

            view.setVisible(true);
        } catch (Exception e) {
            System.err.println("Heey " + e);

        }
    }
}
```

## 13. Clase VentasController

Esta clase es la encargada de recibir los datos de la interface para realizar el llenado de la tabla de ventas y mostrarlo en la tabla de ventas.

```
public class VentasController implements Initializable {

    @FXML
    private TableView<Producto> tablaProducto;
    @FXML
    private TableColumn nombreCol;
    @FXML
    private TableColumn precioCol;
    @FXML
    private TableColumn folioCol;

    @FXML
    private TableView<Producto> tablaVenta;
    @FXML
    private TableColumn nombreVenta;
    @FXML
    private TableColumn precioVenta;
    @FXML
    private TableColumn folioVenta;
    @FXML
    private TableColumn cantidadVenta;
    @FXML
    private Button anadir;
    @FXML
    private Button btnEliminar;
    @FXML
    private Button btnEndVenta;
    @FXML
    private Label fechaVenta;
    @FXML
    private Label totalVenta;
    public static String nombreCajero;
    public static int idCajero;

    Date fecha;

    private ObservableList<Producto> productos;
    private ObservableList<Producto> ventas;
    private List<Producto> ventas2;
    private List<String> nombresProductos;
```

```
private List<Integer> cantidadProductos;
private List<Float> precioProductos;

@Override
public void initialize(URL url, ResourceBundle rb) {
   SimpleDateFormat objSDF = new SimpleDateFormat("dd-MMM-yyyy");

   fecha = new Date();
   productos = FXCollections.observableArrayList();
   ventas = FXCollections.observableArrayList();
   ventas2 = new ArrayList<Producto>();
   nombresProductos = new ArrayList<String>();
   cantidadProductos = new ArrayList<Integer>();
   precioProductos = new ArrayList<Float>();
   btnEliminar.setDisable(true);
   btnEndVenta.setDisable(true);
   this.inicializarTablaProductos();

   fechaVenta.setText(objSDF.format(fecha).toString());

   // TODO
}

public void inicializarTablaProductos() {
   productos = FXCollections.observableArrayList();
   Producto.llenarInfoProductos(productos);
   tablaProducto.setItems(productos);
   nombreCol.setCellValueFactory(new PropertyValueFactory<Producto, String>("nombre"));
   precioCol.setCellValueFactory(new PropertyValueFactory<Producto, Float>("precio"));
   folioCol.setCellValueFactory(new PropertyValueFactory<Producto, Integer>("Folio"));
}

public Producto getTablaProductosSeleccionada() {
   if (tablaProducto != null) {
      List<Producto> tabla = tablaProducto.getSelectionModel().getSelectedItems();
      if (tabla.size() == 1) {
         final Producto competicionSeleccionada = tabla.get(0);
         //JOptionPane.showMessageDialog(null, competicionSeleccionada.id);
         return competicionSeleccionada;
      }
   }
   return null;
}

public void vaciarTablaVenta() {
   final Producto producto = tablaVenta.getSelectionModel().getSelectedItem();
   System.out.println("ANTES DE ELIMINAR " + producto.getNombre());
```

```
        ventas.remove(producto);
        System.out.println("DESPUES DE ELIMINAR " + ventas.size());
        verTablaVenta();
        calcularTotal();
        if (ventas.size() == 0) {
            btnEliminar.setDisable(true);
            btnEndVenta.setDisable(true);
        }
    }

    public void calcularTotal() {
        float res = 0;
        for (Producto producto1 : ventas) {
            res = res + (producto1.getCantidad() * Float.parseFloat(producto1.getPrecio()));
            System.out.println("Productoo " + producto1.getCantidad());

        }
        totalVenta.setText(String.valueOf(res));
    }

    public void verTablaVenta() {
        tablaVenta.setItems(ventas);
        nombreVenta.setCellValueFactory(new PropertyValueFactory<Producto, String>("nombre"));
        precioVenta.setCellValueFactory(new PropertyValueFactory<Producto, Float>("precio"));
        folioVenta.setCellValueFactory(new PropertyValueFactory<Producto, Integer>("Folio"));
        cantidadVenta.setCellValueFactory(new PropertyValueFactory<Producto,
Integer>("cantidad"));
    }

    public void llenarTablaVenta() {
        btnEliminar.setDisable(false);
        btnEndVenta.setDisable(false);
        TextInputDialog dialog = new TextInputDialog("");
        dialog.setTitle("Ventas");
        dialog.setContentText("Ingresa la cantidad:");
        Optional<String> result = dialog.showAndWait();
        if (result.isPresent()) {
            final Producto producto = getTablaProductosSeleccionada();

            if (Integer.valueOf(Integer.valueOf(result.get())) > producto.getCantidad()) {

                Alert alert = new Alert(AlertType.WARNING);
                alert.setTitle("Advertencia");
                alert.setHeaderText("Cantidades no validas");
                alert.setContentText("El producto no cuenta con esa cantidad, por favor ingrese una
cantidad válida ");

                alert.showAndWait();
```

```
        } else {

            producto.setCantidad(Integer.valueOf(result.get()));
            Producto.llenarVenta(ventas, producto);
            tablaVenta.setItems(ventas);
            nombreVenta.setCellValueFactory(new PropertyValueFactory<Producto,
String>("nombre"));
            precioVenta.setCellValueFactory(new PropertyValueFactory<Producto, Float>("precio"));
            folioVenta.setCellValueFactory(new PropertyValueFactory<Producto, Integer>("Folio"));
            cantidadVenta.setCellValueFactory(new PropertyValueFactory<Producto,
Integer>("cantidad"));
            calcularTotal();
            Producto.llenarVenta2(ventas2, producto, nombresProductos, cantidadProductos,
precioProductos);

        }

    }
  }

  public void endVenta() {
    Alert alert = new Alert(Alert.AlertType.CONFIRMATION);
    alert.setTitle("Confirmar");
    alert.setHeaderText("");
    alert.setContentText("Esta seguro de continuar?");

    Optional<ButtonType> result = alert.showAndWait();
    if (result.get() == ButtonType.OK) {
      Reporte reporte = new Reporte();
      Producto productoActualizable = new Producto();
      try {

        System.out.println("Productos " + productos.get(0).getCantidad() + " ventas " +
ventas.get(0).getCantidad());

        reporte.generarTicketVenta(fechaVenta.getText(), Float.parseFloat(totalVenta.getText()),
nombresProductos, cantidadProductos, precioProductos);
        //INSERTAR

        Producto.registrarVentas(ventas, fechaVenta.getText(), idCajero);
        productoActualizable.updateCantidad(ventas);
      } catch (Exception e) {
        System.err.println(e);
      }
    } else {
    }
  }
}
```

## 14. Clase MenuController

Esta clase es la encargada de abrir las diferentes ventanas del sistema tales como: la ventana de ventas, de usuarios, de proveedores, del inventario y la de reportes.

```
public class MenuController implements Initializable {
    @FXML
    private ImageView btnVentas;
    @FXML
    private ImageView btnProductos;
    @FXML
    private ImageView btnProveedores;
    @FXML
    private ImageView btnReportes;
    @FXML
    private ImageView btnUsuarios;
    @FXML
    private Text txtVentas;
    @FXML
    private Text txtProductos;
    @FXML
    private Text txtProveedores;
    @FXML
    private Text txtReportes;
    @FXML
    private Text txtUsuarios;
    static Stage opcionMenu;
    static FXMLLoader loaderMenu;
    public static String tipoUsuario;
    float total;
    Date fecha;
    String fechaString;

    @Override
    public void initialize(URL url, ResourceBundle rb) {
        SimpleDateFormat objSDF = new SimpleDateFormat("dd-MMM-yyyy");

        fecha = new Date();
        fechaString = objSDF.format(fecha).toString();
        if (tipoUsuario.equals("Cajero")) {
            btnReportes.setVisible(false);
            btnUsuarios.setVisible(false);
            txtReportes.setVisible(false);
            txtUsuarios.setVisible(false);
        }
```

```
}
    public void abrirVentas() throws IOException {
        loaderMenu = new FXMLLoader(getClass().getResource("/Vista/Ventas.fxml"));
        Parent root1 = (Parent) loaderMenu.load();
        opcionMenu = new Stage();
        opcionMenu.setScene(new Scene(root1));
        opcionMenu.setResizable(false);
        opcionMenu.show();
    }

    public void abrirUsuarios() throws IOException {
        loaderMenu = new FXMLLoader(getClass().getResource("/Vista/RegistroUsuarios.fxml"));
        Parent root1 = (Parent) loaderMenu.load();
        opcionMenu = new Stage();
        opcionMenu.setScene(new Scene(root1));
        opcionMenu.setResizable(false);
        opcionMenu.show();
    }

    public void abrirProveedores() throws IOException {
        //DIRIGIRSE A LISTA PROVEEDORES
        loaderMenu = new FXMLLoader(getClass().getResource("/Vista/ListaProveedores.fxml"));
        Parent root1 = (Parent) loaderMenu.load();
        opcionMenu = new Stage();
        opcionMenu.setScene(new Scene(root1));
        opcionMenu.setResizable(false);
        opcionMenu.show();
    }

    public void abrirInventario() throws IOException {
        loaderMenu = new FXMLLoader(getClass().getResource("/Vista/ListaProductos.fxml"));
        Parent root1 = (Parent) loaderMenu.load();
        opcionMenu = new Stage();
        opcionMenu.setScene(new Scene(root1));
        opcionMenu.setResizable(false);
        opcionMenu.show();
    }

    public void abrirReportes() throws SQLException {
        System.err.println("FALTA REPORTES");
        Producto producto = new Producto();
        total = producto.sumaTotalDia(fechaString);
        Reporte reporte = new Reporte();
        reporte.generarCorteCaja(fechaString, total);
    }
}
```