PROYECTO FINAL

“Programa de Administración de Facturas con Base de Datos”

AUTOR (ES)

- **CORO ANDRÉS**
- **IZA JORGE**

TIPÁN JENNY

ESCUELA POLITÉCNICA NACIONAL

TECNOLOGÍA EN ANÁLISIS DE SISTEMAS INFORMÁTICOS

Jueves 15 de febrero del 2018

Quito – Ecuador

2017 – 2018

ÍNDICE GENERAL

ÍNDICE DE GRÁFICOS

ÍNDICE DE TABLAS

INFORME FINAL PROYECTO
JUEGO SPACE INVADERS

CORO ANDRÉS

IZA JORGE

TIPÁN JENNY

ESCUELA POLITÉCNICA NACIONAL DEL ECUADOR
TECNOLOGÍA EN ANÁLISIS DE SISTEMAS INFORMÁTICOS

1. INTRODUCCIÓN

El presente proyecto tiene como fin presentar la implementación de un sistema que administra las facturaciones de una tienda, dentro de este se detallará el proceso realizado y las herramientas utilizadas para desarrollar el programa. Es importante conocer las versiones de programas estables y compatibles con el equipo y con los demás programas que se vayan a utilizar antes de iniciar el desarrollo del sistema.

2. OBJETIVO GENERAL

Implementar una Interfaz gráfica en Java que me permita administrar un sistema de facturación que se encuentre conectada a una base de Datos en PhpMyAdmin que me permita almacenar los registros y datos de mi sistema, utilizando todas las herramientas vistas en clase y aplicando los conocimientos adquiridos en el presente semestre.

3. OBJETIVOS ESPECÍFICOS

- Descargar todos los programas necesarios para la implementación del sistema de facturación.
- Crear una base de datos en PhpMyAdmin
- Realizar una conexión de mi programa en Java (NetBeans) con la base de datos (PhpMyAdmin).
- Crear un Interfaz gráfica que me permita poder administrar mi sistema de facturación.

4. MARCO TEÓRICO

4.1 Lenguaje de Programación Java

Para el desarrollo del sistema de facturación se usó Java que es un lenguaje de programación de propósito general, concurrente, orientado a objetos, que fue diseñado específicamente para tener tan pocas dependencias de implementación como fuera posible [1]

4.2 Entorno de desarrollo integrado NetBeans

El desarrollo del programa lo hemos realizado NetBeans que es un entorno de desarrollo integrado libre, hecho principalmente para el lenguaje de programación Java.

Unas de las principales razones por la que hemos utilizado este IDE de NetBeans es que es mucho más fácil para crear y diseñar interfaces gráficas y además NetBeans cuenta con un conector que me permite la conexión con mi base de datos en PhpMyAdmin.

4.3 phpMyAdmin

Para la administración de Base de datos hemos utilizado phpMyAdmin que es una herramienta escrita en PHP con

la intención de manejar la administración de MySQL a través de páginas web, utilizando Internet.

Hemos escogido este administrador de base de datos ya que nos permite crear y eliminar Bases de Datos, crear, eliminar y alterar tablas, borrar, editar y añadir campos, ejecutar cualquier sentencia SQL en Netbeans administrar claves en campos

4.4 Servidor XAMPP

XAMPP es un servidor web de plataforma, software libre, que consiste principalmente en el sistema de gestión de bases de datos MySQL, el servidor web Apache.

La razón que instalamos este servidor es porque XAMPP nos permite conectarnos con la base de datos ya que para poder realizar alguna modificación o queramos obtener alguna información de la mismo no podremos hacerlo si no levantamos el servidor XAMPP.

4.5 Connector 5.1.45 Mysql

Mysql Connector es un driver creado por Mysql AB que te permitirá trabajar con Mysql desde programas escritos en Java.

Utilizamos este driver ya que lo necesitábamos para poder escribir sentencias SQL y que de esta forma nos permita interactuar con la base de datos en PhpMyAdmin.

5. MANUAL DE INSTALACIÓN

5.1 Instalación del servidor XAMPP

1. Lo primero que hacemos es ingresar a la página web de www.apachefriends.org para poder descargar el servidor xampp, en nuestro caso hemos escogido la versión 5.6.33 de 32bits.

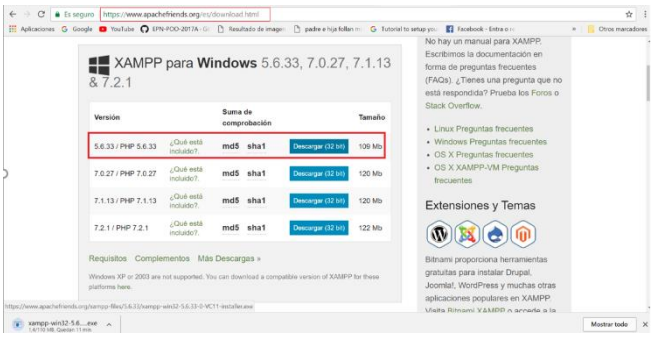


Imagen 1. Página de descarga del servidor XAMPP

2. Damos clic en el botón “Ejecutar” y nos aparecerá una ventana indicando que el antivirus en nuestro sistema operativo puede ralentizar o interferir la instalación, damos clic en el botón “Yes” para continuar la instalación.

3. Se inicia el asistente de instalación. Para continuar, hay que hacer clic en el botón "Next".

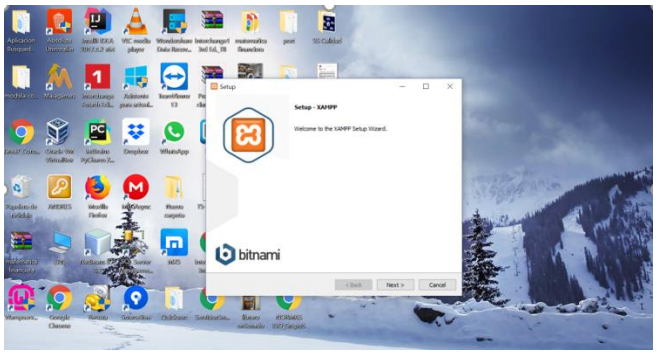


Imagen 2. Interfaz de instalación del servidor XAMPP

4. Los componentes mínimos que instala XAMPP son el servidor Apache y el lenguaje PHP, pero XAMPP también instala otros elementos.

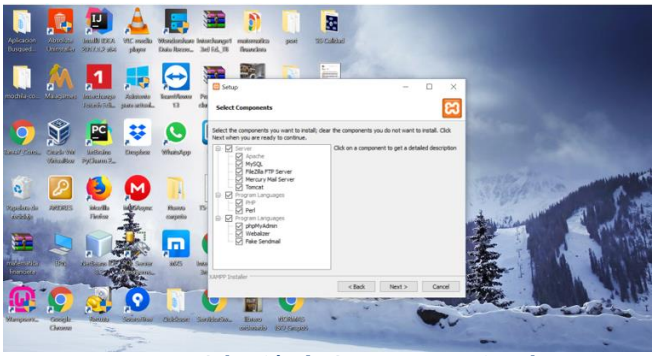


Imagen 3. Selección de Componentes a instalar

5. En la siguiente pantalla se puede elegir la carpeta de instalación de XAMPP. La carpeta de instalación predeterminada es C:\xampp. Para continuar la configuración de la instalación, hay que hacer clic en el botón "Next".

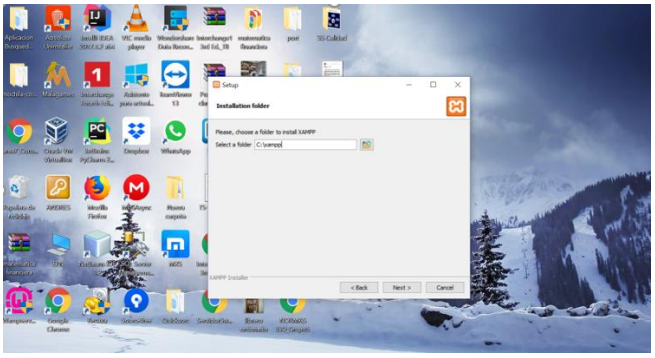


Imagen 4. Dirección donde se almacenará el servidor XAMPP

6. Se inicia el proceso de copia de archivos, que puede durar unos minutos.

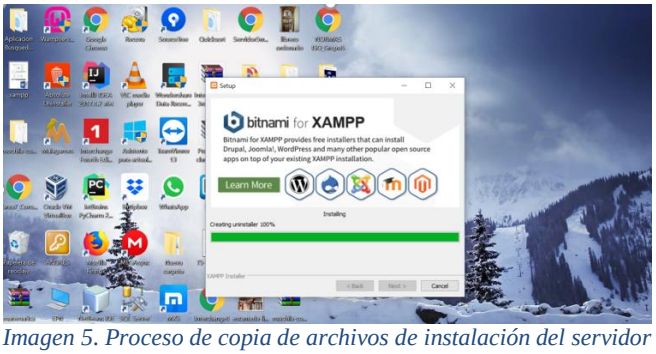


Imagen 5. Proceso de copia de archivos de instalación del servidor XAMPP

7. Seleccionamos el idioma con la que vamos a instalar el servidor xampp. En este caso seleccionaremos el idioma ingles

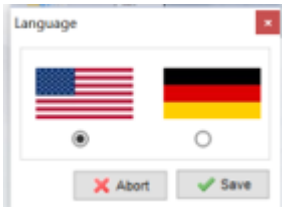


Imagen 6, Selección de idioma del Servidor XAMPP

8. Al finalizar la descarga se mostrará una ventana indicando esto, damos clic en el botón “Finish”. Una vez instalado podremos apreciar el panel de XAMPP listo para utilizar.

Imagen 7. Panel del servidor XAMPP

5.2 Instalación del connector 5.1.45 Mysql

1. Procederemos con la instalación del driver que nos va a permitir la conexión de nuestra base de datos con NetBeans para ello ingresaremos a la página de dev.mysql/ donde descargaremos la versión 5.1.45 de3.6 M “Conector J 5.1.45”.

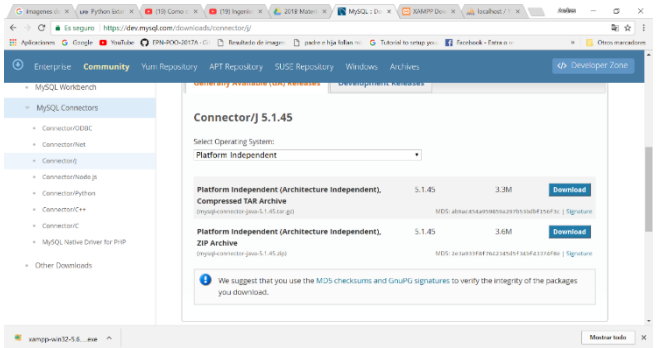


Imagen 8. Página para descargar connector 5.,1,45 Mysql

2. El driver se descargará en un archivo .rar lo cual deberemos descomprimirlo y de preferencia ubicarlos dentro de la carpeta donde se encuentre el

proyecto que realizaremos en Netbeans.

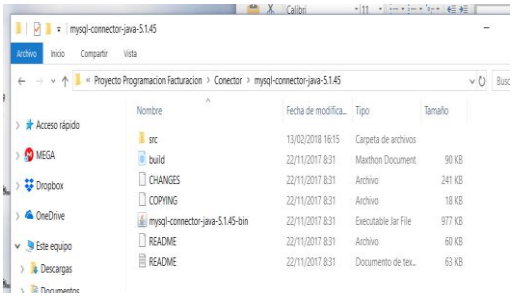


Imagen 9. Archivos que tiene la carpeta del conector 5.1,45 Mysql

6. IMPLEMENTACIÓN PROGRAMA

6.1 Creación de la base de datos y las tablas en PhpMyAdmin.

1. Procedemos a crear la base de datos en "PhpMyAdmin" el cual la llamaremos "facturación" y crearemos las tablas que vamos a utilizar en esta base como son: productos, clientes, cab_fact (cabecera de facturas), det_fact (Detalle de la factura).

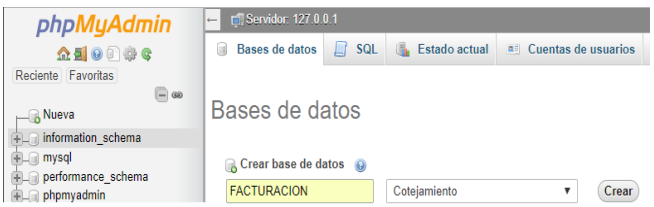


Imagen 10. Creación de base de datos "facturación" en PjpMyAdmin

Una vez creada la base de datos "facturación" dentro de phpMyadmin procederemos a crear las tablas que vamos a utilizar.

En nuestro caso lo haremos directamente en PhpMyAdmin utilizando la consola SQL que trae integrada, y escribiendo el código de creación de una tabla dentro de la base de datos "facturación".

2. Creación mediante código sql de la tabla "cliente" dentro de la base de datos "facturación".

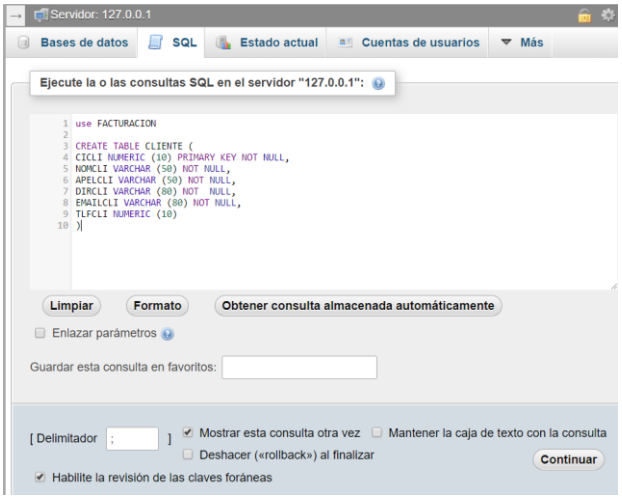


Imagen 11. Código de creación de la tabla cliente

3. Creación mediante código sql de la tabla "cab_fact" dentro de la base de datos "facturación".

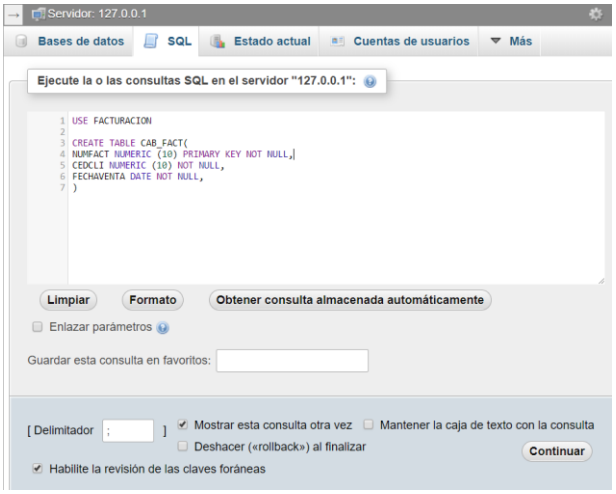


Imagen 12. Código de creación de la tabla cab_fact

4. Creación mediante código sql de la tabla "det_fact" dentro de la base de datos "facturación".

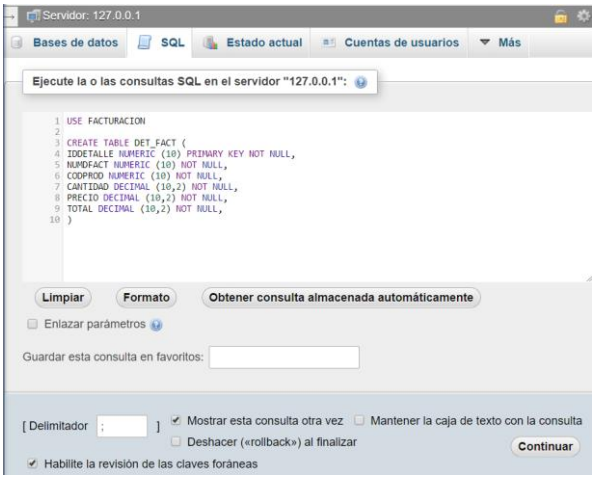


Imagen 13. Código de creación de la tabla det_fact

5. Creación mediante código sql de la tabla “productos” dentro de la base de datos “facturación”.

Imagen 14. Código de creación de tabla productos

Como se puede observar en la imagen ya tenemos creada toda nuestra base de datos con las columnas que necesita cada tabla, es decir declarando cuales van hacer nuestras “primary_key”, cuales son los datos y de qué tipo vamos a poder ingresar en la base de datos.

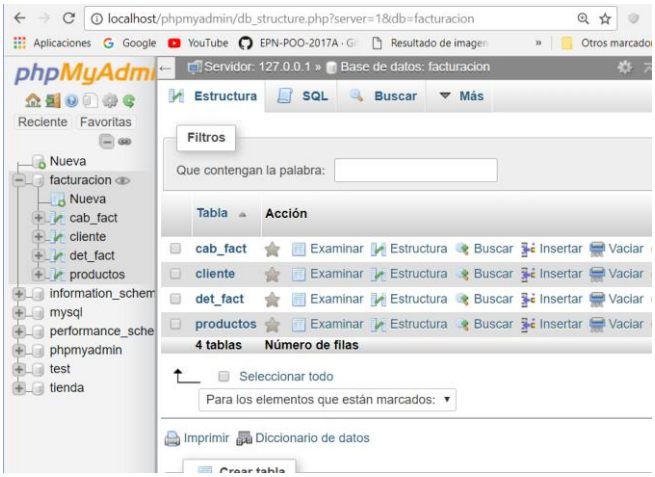


Imagen 15. Visualización de las tablas en la base de datos facturación

6.2 Creación del proyecto “Facturacion” y de los paquetes que vamos a utilizar dentro del programa.

1. Creamos un proyecto de java en Netbeans en este caso llamado “facturación”, e indicaremos la dirección donde queremos que se guarde el proyecto.

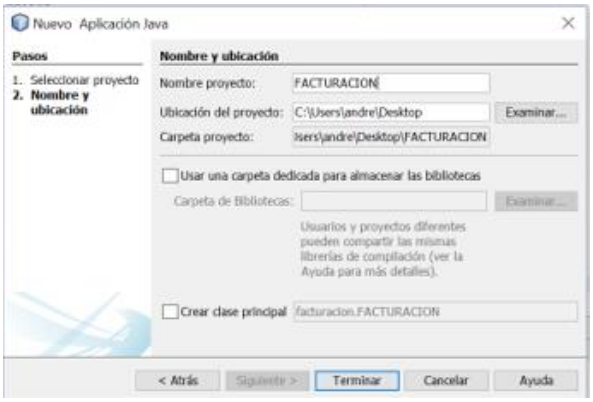


Imagen 16. Creación de proyecto en Netbeans

2. Creamos un paquete dentro del proyecto de “facturación” llamado “Clases” que es donde vamos a colocar todas las clases que necesitemos para poder programar y gestionar la base de datos.

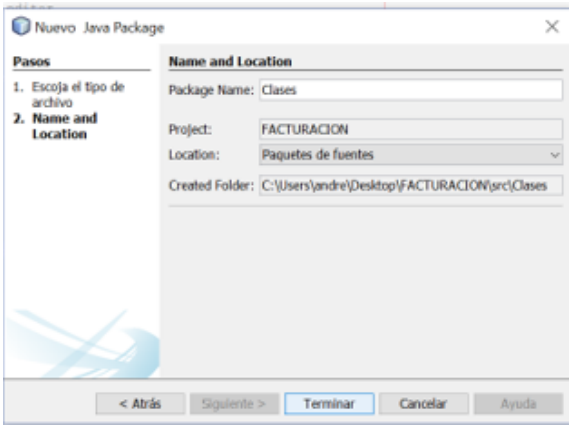


Imagen 17. Creación de paquete “Clases” dentro del proyecto facturación.

3. Creamos un paquete dentro del proyecto de “facturación” llamado “Interfaces” que es donde vamos a colocar todas las interfaces que vamos a utilizar para gestionar nuestro sistema de facturación.

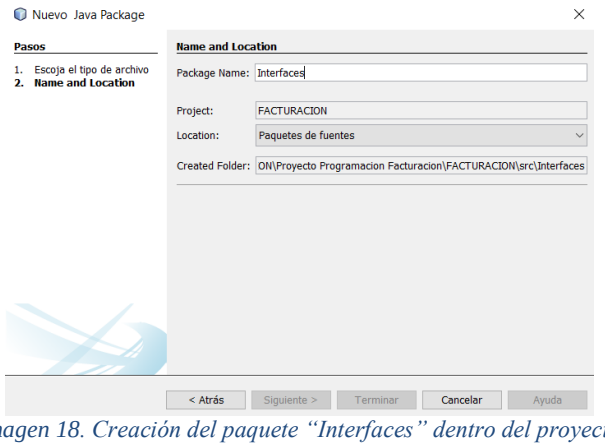


Imagen 18. Creación del paquete “Interfaces” dentro del proyecto facturación.

4. Creamos un paquete dentro del proyecto de "facturación" llamado "Imágenes" que es donde vamos a colocar todas las imágenes que vamos a colocar en los botones o interfaces para que sea mucho más interactivo e intuitivo nuestro sistema de facturación.

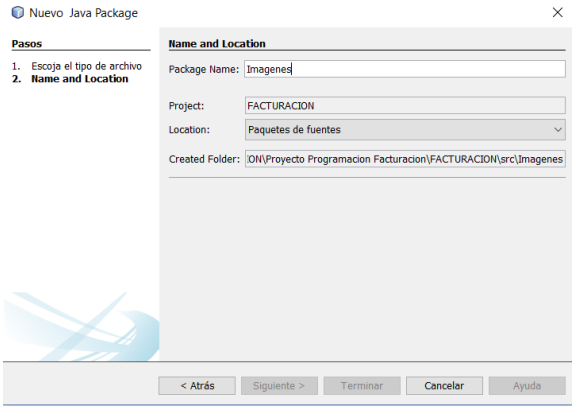


Imagen 19. Creación del paquete "Imágenes" dentro del proyecto facturación.

6.3 Conexión de la base de datos con netbeans

1. Para realiazar la conexión tendresmos que activar los servicios del servidor XAMPP que son "Apache" y "MySQL", esto se lograra dando clic en el boton "Start" con ello crearemos la conexión que va a tener nuestra Base de datos con Netbeans.

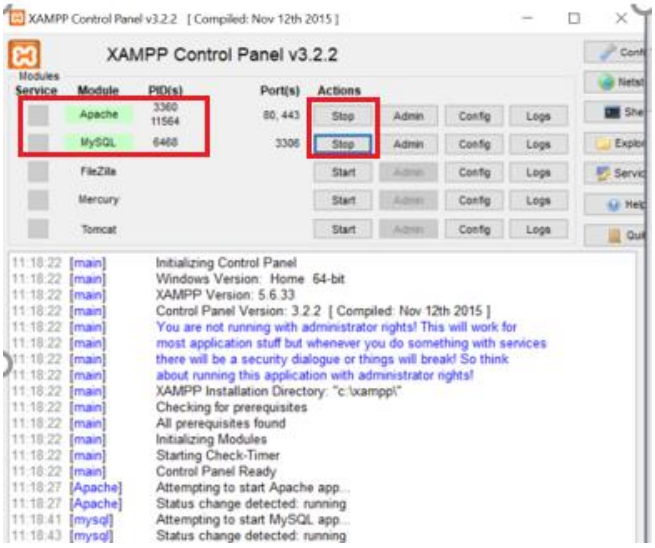


Imagen 20. Levantar el servicio de "Apache" y "Mysql" del servidor XAMPP

2. Antes de comenzar a programar el código para proceder con la conexión de la base de datos lo que deberemos hacer es ir dentro de la carpeta "Biblioteca" y dar clic derecho sobre ella y seleccionar la opción de "Agregar archivo JAR" esto quiere decir que vamos a ubicar el archivo "Conector J 5.1.45." que el driver necesario que nos permitirá hacer posible la conexión con nuestra base de datos mediante el servidor Xampp.

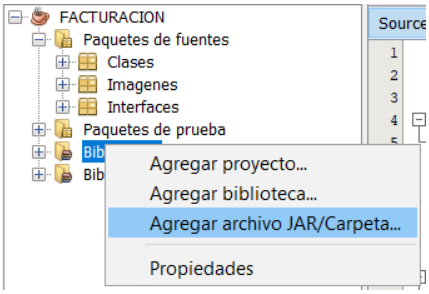


Imagen 21. Importar el driver Conector J 5.1.45." a nuestro proyecto de facturación en Netbeans.

3. Seleccionaremos la ubicación donde se encuentra almacenado el driver de conexión "Conector J 5.1.45." y listo.

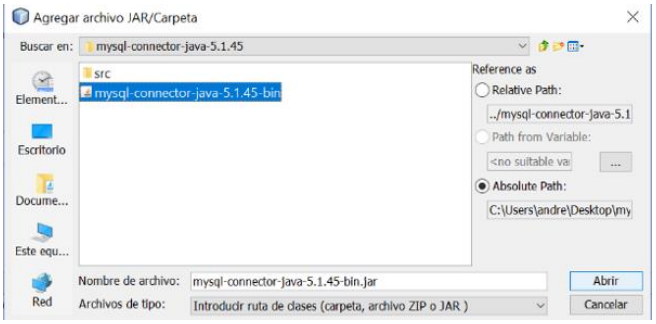


Imagen 22. Ubicación del driver de conexión "Conector J 5.1.45."

4. Después lo que haremos es crear la clase "ClsConectar" el cual me permitirá realizar la conexión de mi base de datos con mi proyecto de facturación para ello importaremos las librerías

```
import java.sql.Connection;
import java.sql.DriverManager;
```

Imagen 23. Librerías utilizadas para la conexión de la base de datos y Netbeans

con las que me permitirán trabajar con el driver de conexión "Conector J 5.1.45."

```
Salida | ClsConectar.java
Source | History
1  package Clases;
2
3  import java.sql.Connection;
4  import java.sql.DriverManager;
5
6  public class ClsConectar {
7
8      Connection conectar = null;
9
10     public Connection conexion(){
11         try{
12             Class.forName("com.mysql.jdbc.Driver");
13             conectar=DriverManager.getConnection("jdbc:mysql://localhost/facturacion","root","");
14
15         }catch(Exception e){
16             System.out.print(e.getMessage());
17         }
18
19         return conectar;
20     }
21
22 }
23
24
```

Imagen 24. Creación de clase "ClsConectar"

Dentro de esta clase especificaremos la ubicación de la base de datos, la clave de acceso si la tiene y el nombre de usuario para ingresar a la base de datos que se encuentra en PhpMyAdmin y donde crearemos una variable con la que podamos identificar al driver de conexión,

```
Class.forName("com.mysql.jdbc.Driver");
conectar=DriverManager.getConnection("jdbc:mysql://localhost/facturacion","root",
```

Imagen 25. Código de acceso a la base de datos y acceso a driver de conexión.

6.4 Creación de la clase principal.

Lo primero que hemos hecho es crear una clase principal desde el cual al momento de ejecutar el programa en Netbeans sea la primera instrucción que lea que es crear una ventana o interfaz llamada “FrmLogin” El cual tiene como pedir al usuario que ingrese una contraseña y usuario de acceso para que pueda ingresar a administrar nuestro sistema de facturación.

```
package Clases;

import Interfaces.*;
import Clases.*;

public class ClasePrincipal {
    public static void main(String[] args) {
        FrmLogin venlogin = new FrmLogin();
        venlogin.setSize(500, 550);
        venlogin.setResizable(false);
        venlogin.setDefaultCloseOperation(venlogin.EXIT_ON_CLOSE);
        venlogin.setLocation(500, 100);
        venlogin.setVisible(true);
    }

}
```

Imagen 26. Creación de la clase principal “ClsPrincipal”

Como observamos en la imagen este método llama a la ventana “FrmLogin”, definimos el tamaño de la ventana y cuál será su localización.

6.5 Creación de la interfaz “FrmLogin”

1. Después lo que hemos hecho es crear una interfaz que me permita acceder a gestionar la base datos ingresando un usuario y una contraseña.

Imagen 27 Ventana de acceso al programa de Facturacion “FrmLogin”

como se puede observar en la imagen hemos creado un método que lea el usuario y la contraseña que estamos ingresando, si la contraseña y el usuario son a las que se encuentran definidas pues lo que hará es abrir una ventana que es interfaz de menús, donde se puede escoger, registrar la factura, registrar cliente, registrar producto, o ver el historial de factura, todo depende de la opción que ingresemos.

```
public void iniciarSecion() {

    String usuario = txtUsuLogin.getText();
     String pasword = String.valueOf(txtPassword.getPassword());

    if (!usuario.equals("") || !pasword.equals("")) {
        if (usuario.equals("admin") && password.compareTo("admin") == 0) {

            FrmMenu venMenu = new FrmMenu();
            venMenu.setSize(425, 450);
            venMenu.setResizable(false);
            venMenu.setDefaultCloseOperation(venMenu.EXIT_ON_CLOSE);
            venMenu.setLocation(500, 100);
            venMenu.setVisible(true);
            this.dispose();

        } else {
            JOptionPane.showMessageDialog(null, "Usuario o contraseña no valida", "error", JO
        }

    } else {
        JOptionPane.showMessageDialog(null,"INGRESE DATOS","ERROR!!",JOptionPane.INFORMATION_M
    }

}
```

Imagen 28. Imagen 29. Código pare validar un usuario y contraseña para acceder al programa de admiración de facturas.

De igual manera como vemos en la imagen en caso de que al

momento de ingresar las contraseña y la clave no ingresemos alguno de estos dos datos o ingresemos incorrectamente a los definidos pues nos aparecerá un mensaje indicando que falta llenar un dato o de que los datos ingresados están incorrectos.

Imagen 30. Mensaje informativo de contraseña o usuario incorrectos.

Como podemos observar este es el mensaje que nos apareces ara en el caso de que no ingresemos correctamente el usuario o la contraseña.

6.6 Creación de la interfaz "FrmMenu"

Después de que en la ventana anterior "FrmLogin" hayamos ingresado correctamente el usuario y la contraseña se abrirá esta nueva ventana llamada "FrmMenu" en el cual se le da al usuario varias opciones que puede realizar dentro del programa de administración de facturas como son:

1. **Registrar un Factura:** En esta opción lo que se hace generar una factura siempre y cuando el cliente se encuentre registrado en nuestra base de datos, en caso de que no deberemos primero registrar al cliente.

2. **Registrar Cliente:** como su nombre lo indica ingresando a esta opción podremos registrar en la base de datos a un cliente nuevo, con el fin de poder emitir una nueva factura.

3. **Registrar Producto:** Si ingresamos a esta opción podremos registrar un nuevo producto que vende nuestra tienda, en este caso los productos son electrónicos y de computación para que puedan mostrarse al momento de vender y generar una factura.

4. **Historial de facturas:** Ingresando a esta opción lo que podremos hacer es visualizar la lista de facturas generadas en mi tienda, y poder realizar consultas como son ver las facturas de un cliente en específico, de una fecha en específico o si queremos visualizarlos ordenadas ya se por fecha, valor o cliente.

5. **Salir:** Si ingresamos a esta opción lo que aremos es cerrar el programa

Imagen 31 Interfaz "FrmMenu"

El código que se encuentra dentro de "FrmMenu" es simplemente que cada botón lo que hace es abrir una nueva ventana como por ejemplo en el caso de que presionemos el botón "REGISTRAR CLIENTE" lo que haremos es abrir la venta de "FrmRegistroCliente" y así igualmente con los demás botones.

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

    FrmRegistroCliente venCliente = new FrmRegistroCliente();
    venCliente.setSize(590, 700);
    venCliente.setResizable(false);
    venCliente.setDefaultCloseOperation(venCliente.EXIT_ON_CLOSE);
    venCliente.setLocation(400, 100);
    venCliente.setVisible(true);
    this.dispose();

}

private void btnRegistrarFacturaActionPerformed(java.awt.event.ActionEvent evt) {

    FrmRegistroFactura venFactura = new FrmRegistroFactura();
    venFactura.setSize(1200, 1300);
    venFactura.setResizable(false);
    venFactura.setDefaultCloseOperation(venFactura.EXIT_ON_CLOSE);
    venFactura.setLocation(350, -10);
    venFactura.setVisible(true);
    this.dispose();

}

private void btnRegistrarProductosActionPerformed(java.awt.event.ActionEvent evt) {

    FrmRegistroProductos venProductos = new FrmRegistroProductos();
    venProductos.setSize(565, 800);
    venProductos.setResizable(false);
    venProductos.setDefaultCloseOperation(venProductos.EXIT_ON_CLOSE);
    venProductos.setLocation(500, 100);
    venProductos.setVisible(true);
    this.dispose();
}

private void btnHistoriaFacturasActionPerformed(java.awt.event.ActionEvent evt) {

    FrmHistorialFacturas venHistorialFacturas = new FrmHistorialFacturas();
    venHistorialFacturas.setSize(725, 800);
    venHistorialFacturas.setResizable(false);
    venHistorialFacturas.setDefaultCloseOperation(venHistorialFacturas.EXIT_ON_CL
    venHistorialFacturas.setLocation(500, 100);
    venHistorialFacturas.setVisible(true);
    this.dispose();
```

Imagen 32, Código de funcionamiento "FrmMenu"

En el caso del botón salir lo único que esta implementado al presionar este botón es:

```
private void btnSalirActionPerformed(java.awt.event.ActionEvent evt) {
    System.exit(0);
}
```

Imagen 33. Método implementado en el botón salir de "FrmMenu"

Como vemos ese comando lo que hace es cerrar el programa

6.7 Creación de la interfaz "FrmRegistrarCliente"

Imagen 34. Interfaz "FrmRegistroClientes"

Como podemos observar en esta ventana lo que hacemos es registrar a un cliente nuevo, como ya lo mencionamos anteriormente no podremos generar una factura que en si es registrar una venta a un cliente que no se encuentre registrado previamente.

En esta opción lo que pedimos es que se ingrese los datos de Cedula, nombre, apellido, dirección, email y teléfono del cliente. Por tal motivo esta validado para que se ingrese 10 dígitos de la cedula y que todos los campos este completamente llenos para poder registrar a un cliente en la base de datos de la tienda.

```
ClsConectar cc = new ClsConectar();
Connection cn = cc.conexion();
```

Imagen 35 Definición de variables para acceder a la base de datos.

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

    try{
        PreparedStatement pst = cn.prepareStatement("INSERT INTO CLIENTE (CICLI,NOMCLI,APELCLI,DIRCLI,EMAILCLI,TLFCLI) "
                + "VALUES (?,?,?,?,?,?)");

        String ciCli = txtCedulaCli.getText();
        String nomCli = txtNomCli.getText();
        String apellCli = txtApelCli.getText();
        String dirCli = txtDirCli.getText();
        String emailCli = txtEmailCli.getText();
        String tlfCLi = txtTlfCli.getText();
```

Imagen 36. Código para leer datos de los campos de texto de la interfaz "Registro de Clientes"

Como vemos en la imagen la variable pst lo que hacemos es definir en qué tabla de mi base de datos vamos a guardar, en este caso decimos que vamos a insertar los datos que el usuario llene en los campos en la tabla "Cliente".

```
if (txtCedulaCli.getText() == null || nomCli.equals("") ||
        apellCli.equals("") || dirCli.equals("") || emailCli.equals("")
        || txtTlfCli.getText() == null) {

    JOptionPane.showMessageDialog(null, "!! FALTA LLENAR CAMPOS!! ");

} else {

    int i;
    for(i=1; i<ciCli.length();i++){
        System.out.println("i " +i);
    }

    if(i==10){
        pst.setString(1, ciCli);
        pst.setString(2, nomCli);
        pst.setString(3, apellCli);
        pst.setString(4, dirCli);
        pst.setString(5, emailCli);
        pst.setString(6, tlfCli);
        pst.executeUpdate();
        mostrarClientesTabla();
        JOptionPane.showMessageDialog(null, " !! CLIENTE REGISTRADO EXITOSAMENTE !! ");
    }else{
        JOptionPane.showMessageDialog(null, "  !!  INGRESE LOS 10 DIGITOS DE LA CEDULA ");
    }
}

atch (Exception ex) {

JOptionPane.showMessageDialog(null, " !! ERROR EN EL INGRESO DE DATOS !! ");
```

Imagen 37. Código para almacenar los datos en la base de datos de PhpMysql

Como vemos en la imagen para saber a qué columna vamos a ingresar el dato lo que hacemos es usar la **variable pst.setString(1,ciCli)** lo cual me pide que indiquemos el número de columna donde vamos a guardar y deberemos pasarle el dato que vamos a almacenar.

```
public void mostrarClientesTabla(){

    DefaultTableModel modelo = new DefaultTableModel(){
        @Override
        public boolean isCellEditable(int row, int column) {
            return false; //declarar que las filas y columnas no sean editable
        }
    };

    modelo.addColumn("CI");
    modelo.addColumn("NOMBRE");
    modelo.addColumn("APELLIDO");
    modelo.addColumn("DIRECCION");
    modelo.addColumn("EMAIL");
    modelo.addColumn("TELEFONO");
    tblRegistroCliente.setModel(modelo);
    String [] datosCliente = new String [6];


    try{
        Statement st = cn.createStatement();
        ResultSet rs = st.executeQuery("SELECT * FROM CLIENTE");
        while(rs.next()){

            datosCliente[0] = rs.getString(1);
            datosCliente[1] = rs.getString(2);
            datosCliente[2] = rs.getString(3);
            datosCliente[3] = rs.getString(4);
            datosCliente[4] = rs.getString(5);
            datosCliente[5] = rs.getString(6);
            modelo.addRow(datosCliente);
        }

        tblRegistroCliente.setModel(modelo);

    }catch(SQLException ex){

    }
}
```

Imagen 38. Código para llenar los clientes ya registrados ya en la base de datos en una tabla dentro de la interfaz de registrar Cliente

Para mostrar los clientes registrados en mi base de datos dentro de una tabla hemos implementado el método de "mostrarClienteTabla" lo que hacemos es crear un modelo de tabla y obtenemos los datos que ya se encuentran guardado en la base de datos mandando la tabla de donde vamos a sacar los datos y estos datos que obtenemos lo guardamos dentro de un arreglo. Después de obtener todos los datos lo que hacemos es pasar el arreglo al modelo de la tabla.

6.8 Creación de la interfaz "FrmRegistrarProducto"

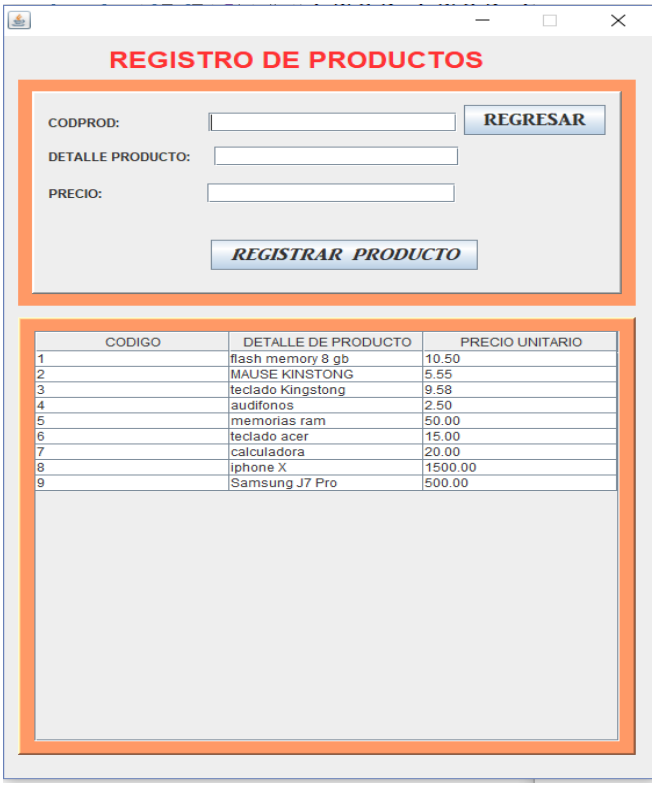


Imagen 39. Interfaz "FrmRegistroProductos"

En el caos de la ventana de registro de productos, es similar a la de registro de cliente ya que lo que hacemos es pedir al usuario que ingrese los datos del producto como son el código del producto, el detalle o nombre del producto y el precio del producto.

De la misma manera se encuentra validado que para registrar un producto se ingresen todos los campos que se pide al usuario.

6.9. Creación de interfaz "FrmRegistroFactura"

Imagen 40. Interfaz "FrmRegistrofactura"

Como podemos observar en la imagen se muestra la interfaz de una factura en el cual se registra la venta de productos de la tienda a un cliente mediante una factura.

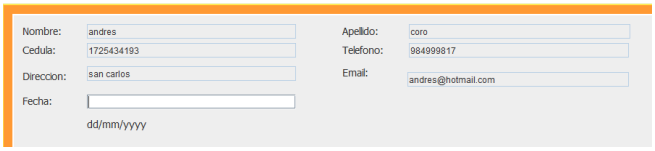


Imagen 41. Datos del cliente registrados en la ventana "FrmRegistro de Factura"

Para obtener los datos del cliente en la factura lo que hacemos es pedir que el usuario ingrese un numero de cedula, si el cliente ya se encuentra registrado en nuestra base de datos se cargaran en los campos correspondientes, caso contrario nos aparecerá un mensaje indicando que el usuario que intentamos ingresar no se encuentra registrado.

```
try {

    String sql = "SELECT * FROM CLIENTE WHERE CICLI = '" + valor + "'";
    Statement st = cn.createStatement();
    ResultSet rs = st.executeQuery(sql);

    String datos[] = new String[6];

    while (rs.next()) {
        datos[0] = rs.getString(1);
        datos[1] = rs.getString(2);
        datos[2] = rs.getString(3);
        datos[3] = rs.getString(4);
        datos[4] = rs.getString(5);
        datos[5] = rs.getString(6);

    }
    if(datos[0] != null ){

        // lleno los datos del cliente en los combobox correspondientes

        txtNomCli.setText(datos[1]);
        txtApelCli.setText(datos[2]);
        txtDirCli.setText(datos[3]);
        txtEmailCli.setText(datos[4]);
        txtTlfCli.setText(datos[5]);

        // activo los campos de fecha y numero de factura

        txtFechaFact.setEditable(true);
        txtNumeroFactura.setEditable(true);
        txtCedCli.setEditable(false);

        btnBuscarCli.setEnabled(false);
        btnLimpiarDatos.setEnabled(true);
        btnCrearFactura.setEnabled(true);


    }else{
        JOptionPane.showMessageDialog(null, "!!    CLIENTE NO REGISTRADO    !!");
        inicializarCampos();
        txtCedCli.setText("");
    }
```

Imagen 42. Código para mostrar los datos en los campos correspondientes en la factura

Lo que hacemos es capturar y guardar en una variable la cedula que nos ingresa el usuario, después lo que hacemos es buscar los datos del cliente que se encuentran guardado en la base de datos del cliente y la comparamos con la cedula que nos han ingresado. Después lo que hacemos es mostrar los campos del cliente en los campos de texto correspondientes.

Imagen 43. Datos del cliente, número y fecha de la factura

Una vez que hayamos ingresado una cedula valida de un cliente ya registrado lo que deberemos hacer es ingresar una fecha en formato dd/mm/yyyy y un numero de factura, el programa esta validado que si la fecha esta en otro formato o incorrecta salta un mensaje indicándole el error, o de igual manera si ingresa un numero de factura que ya se encuentra registrado.

```
String fecha = txtFechaFact.getText();
String numeroFact = txtNumeroFactura.getText();
String cedulaCli = txtCedCli.getText();

boolean fechaValidar = validarFecha(fecha);

if(!fecha.equals("") && !numeroFact.equals("")){

    if (fechaValidar == true) {

        System.out.println("si entro");

        PreparedStatement pst = cn.prepareStatement("INSERT INTO CAB_FACT (NUMFACT,CEDCLI,FECHAVENTA) "
                + "VALUES (?,?,?)");


        pst.setString(1, numeroFact );
        pst.setString(2, cedulaCli );
        pst.setString(3, FechaFact);
        pst.executeUpdate();

        //JOptionPane.showMessageDialog(null, "    !!  FACTURA CREADA EXITOSAMENTE  !! ");

        btnCrearFactura.setEnabled(false);
        btnLimpiarDatos.setEnabled(false);
        btnRegresar.setEnabled(false);
        btnAgregarProdFact.setEnabled(true);

        txtFechaFact.setEditable(false);
        txtNumeroFactura.setEditable(false);
        txtCantProdFact.setEditable(true);

        cmbListaProdFact.setEnabled(true);

    } catch (SQLException ex) {
        JOptionPane.showMessageDialog(null, "  !!  NUMERO DE FACTURA YA REGISTRADO  !!  ");
    }
}else{
    JOptionPane.showMessageDialog(null, "  !!  FORMATO DE FECHA INCORRECTA dd/mm/yyyy  !!  ");
}
```

Imagen 44 Código para validar el formato de fecha.

En este código validamos que el formato de la fecha sea correcto y que el campo de fecha y el número de factura no estén vacios.

Al dar clic en el botón de crear factura lo que hara después es activarse el campo de seleccionar producto.

Una vez que la fecha y el número de factura estén correctos se activara el campo de añadir producto

Imagen 45 campo de selección de producto

Como vemos en el combobox se encuentran cargado los productos que ya registrados anteriormente.

```
public void mostrarProductosComboProd(){

    String [] datosProd = new String [3];

    try{
        Statement st = cn.createStatement();
        ResultSet rs = st.executeQuery("SELECT * FROM PRODUCTOS");
        while(rs.next()){

            datosProd[0] = rs.getString(1);
            datosProd[1] = rs.getString(3);
            datosProd[2] = rs.getString(2);

            cmbListaProdFact.addItem(datosProd[1]);

        }

    }catch(SQLException ex){

    }
```

Imagen 46. Código para cargar los productos a el combobox

Este es el código del método que usamos para cargar los productos en el combobox, como vemos ingresamos a la base de datos a la tabla de productos y cogemos solo la columna del detalle de producto.

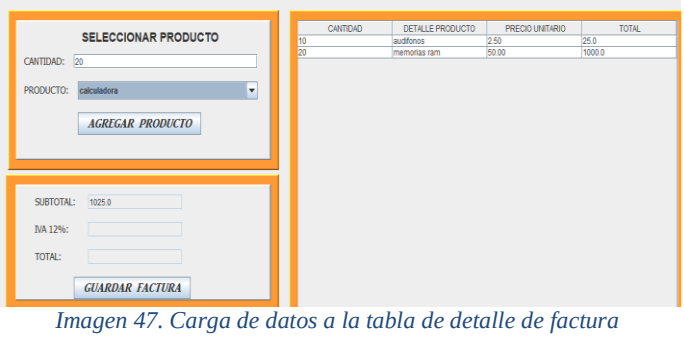


Imagen 47. Carga de datos a la tabla de detalle de factura

Como observamos al dar clic en agregar producto se va cargando a una tabla en donde se muestra la cantidad, el detalle de factura, el precio unitario y el total.

```
contador = contador+1;
totalPorProducto = 0;
Statement st = cn.createStatement();

PreparedStatement pst = cn.prepareStatement("INSERT INTO DET_FACT (IDDETALLE,NUMFACT,CODPROD,CANTIDAD,PRECIO
        + " VALUES (?,?,?,?,?,?)");

ResultSet rs = st.executeQuery("SELECT * FROM PRODUCTOS");

while (rs.next()) {

    datosProdDetalleFact[0] = txtCantProdFact.getText();
    datosProdDetalleFact[1] = rs.getString(3); //detalleProd
    datosProdDetalleFact[2] = rs.getString(2); //precioProd
    datosProdDetalleFact[4] = rs.getString(1); //codg producto

    if (detalleProdSelect.equals(datosProdDetalleFact[1])) {

        total = Float.parseFloat(datosProdDetalleFact[2]) * Integer.parseInt(txtCantProdFact.getText());

        totalPorProducto = Float.parseFloat(datosProdDetalleFact[2]) * Integer.parseInt(txtCantProdFact.getT

        totalFacturaSinIVA = totalFacturaSinIVA + total;
        txtSubtotalFact.setText(String.valueOf(totalFacturaSinIVA));
        datosProdDetalleFact[3] = String.valueOf(total);

        //agrego a la tabla DET_FACT en my base de datos los productos que compro con esta factura

        pst.setString(1,String.valueOf(contador) ); //agrego a DET_FACT el numero de detalle de factura
        pst.setString(2, txtNumeroFactura.getText()); // agrego a DET_FACT el numero de factura
        pst.setString(3, datosProdDetalleFact[4]); // agrego a DET_FAT el codigo de producto
        pst.setString(4, txtCantProdFact.getText()); // agrego a DET_FAT la cantidad
        pst.setString(5, datosProdDetalleFact[2]); // agrego a DET_FAT el precio del producto
        pst.setString(6, String.valueOf(totalPorProducto)); // agrefo a DET_FAT el total de la factura
        System.out.println("ya pase ");
        System.out.println("c");
        modelo.addRow(datosProdDetalleFact);
        System.out.println("numero "+contador);

        pst.executeUpdate();
```

Imagen 48. Código para la carga de datos a la tabla de detalle de factura

Podemos observar el código donde guardamos los productos que compra el cliente en la tabla de det_fact, el cual paso a la tabla, el número de id del detalle de factura, agrego el número de factura, el código de producto, la cantidad y el precio.

De la misma manera tomamos los datos y procedemos a cargar en la tabla de factura.

6.10. Creación de interfaz "FrmHistorialFacturas"

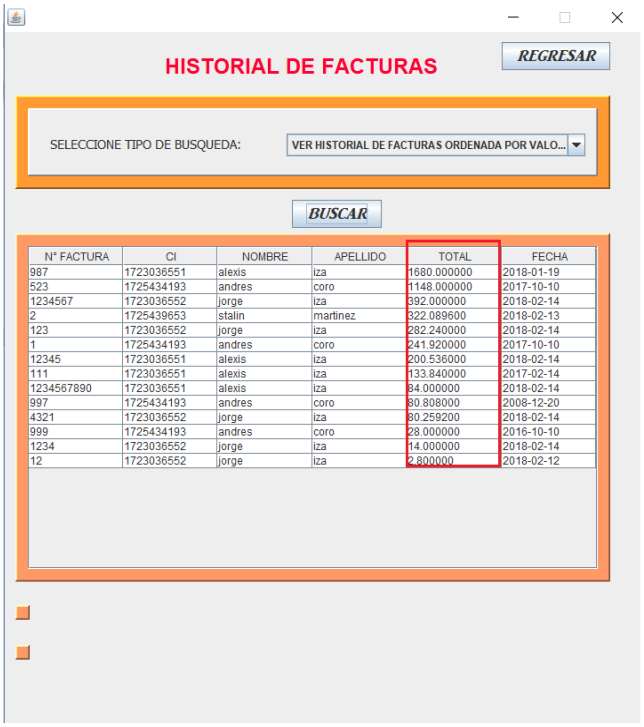
Imagen 49. Interfaz historial de facturas

En el historial de facturas podemos vuisualizar las facturas generadas, podremos ordenarlas ya sea por el cliente, por la fecha de venta que fue emitida la factura, por el valor de mayor a menor costo, de igual manera podemos buscarla por una fecha o cliente especifico.

```
public void mostrarHistorialFacturas(){

    DefaultTableModel modelo = new DefaultTableModel(){
        @Override
        public boolean isCellEditable(int row, int column) {
            return false; //declarar que las filas y columnas no sean editable
        }

    };

    modelo.addColumn("N° FACTURA");
    modelo.addColumn("CI");
    modelo.addColumn("NOMBRE");
    modelo.addColumn("APELLIDO");
    modelo.addColumn("TOTAL");
    modelo.addColumn("FECHA");
    tblHistorialFacturas.setModel(modelo);

    String [] datosCliente = new String [6];

    try{
        String SQL = "SELECT DET_FACT.NUMFACT, CAB_FACT.CEDCLI,NOMCLI,APELCLI,((SUM(TOTAL)*12)/100)+SUM(TOTAL),F
                + "FROM CLIENTE, CAB_FACT, DET_FACT "
                + "WHERE CLIENTE.CICLI = CAB_FACT.CEDCLI "
                + "AND DET_FACT.NUMFACT = CAB_FACT.NUMFACT "
                + "GROUP BY DET_FACT.NUMFACT "
                + "ORDER BY APELCLI ASC";

        Statement st = cn.createStatement();
        ResultSet rs = st.executeQuery(SQL);
        while(rs.next()){

            datosCliente[0] = rs.getString(1);
            datosCliente[1] = rs.getString(2);
            datosCliente[2] = rs.getString(3);
            datosCliente[3] = rs.getString(4);
            datosCliente[4] = rs.getString(5);
            datosCliente[5] = rs.getString(6);
            modelo.addRow(datosCliente);
        }

        tblHistorialFacturas.setModel(modelo);

    }catch(SQLException ex){
```

Imagen 50. Código para visualizar las facturas generadas

Lo que hacemos es escribir el código de búsqueda SQL donde decimos que de la tabla cliente, cab_fact y det_fact muestre los datos número de factura, nombre del cliente, apellido del cliente, costo total más IVA y la fecha agrupando por número de factura y ordenando el apellido de forma ascendente.



N° FACTURA	CI	NOMBRE	APELLIDO	TOTAL	FECHA
987	1723036551	alexis	iza	1680.000000	2018-01-19
523	1725434193	andres	coro	1148.000000	2017-10-10
1234567	1723036552	jorge	iza	392.000000	2018-02-14
2	1725439653	stalin	martinez	322.089600	2018-02-13
123	1723036552	jorge	iza	282.240000	2018-02-14
1	1725434193	andres	coro	241.920000	2017-10-10
12345	1723036551	alexis	iza	200.536000	2018-02-14
111	1723036551	alexis	iza	133.840000	2017-02-14
1234567890	1723036551	alexis	iza	84.000000	2018-02-14
997	1725434193	andres	coro	80.808000	2008-12-20
4321	1723036552	jorge	iza	80.259200	2018-02-14
999	1725434193	andres	coro	28.000000	2016-10-10
1234	1723036552	jorge	iza	14.000000	2018-02-14
12	1723036552	jorge	iza	2.800000	2018-02-12

Imagen 51. Historial de facturas ordenadas por la fecha de factura

```
public void mostrarHistorialFacturasValor(){

    DefaultTableModel modelo = new DefaultTableModel(){
        @Override
        public boolean isCellEditable(int row, int column) {
            return false; //declarar que las filas y columnas no sean editable
        }

    };

    modelo.addColumn("N° FACTURA");
    modelo.addColumn("CI");
    modelo.addColumn("NOMBRE");
    modelo.addColumn("APELLIDO");
    modelo.addColumn("TOTAL");
    modelo.addColumn("FECHA");
    tblHistorialFacturas.setModel(modelo);

    String [] datosCliente = new String [6];

    try{
        String SQL = "SELECT DET_FACT.NUMFACT, CAB_FACT.CEDCLI,NOMCLI,APELCLI,((SUM(TOTAL)*12)/100)+SUM(TOTAL)
                + "FROM CLIENTE, CAB_FACT, DET_FACT "
                + "WHERE CLIENTE.CICLI = CAB_FACT.CEDCLI "
                + "AND DET_FACT.NUMFACT = CAB_FACT.NUMFACT "
                + "GROUP BY DET_FACT.NUMFACT "
                + "ORDER BY VALOR DESC";

        Statement st = cn.createStatement();
        ResultSet rs = st.executeQuery(SQL);
        while(rs.next()){

            datosCliente[0] = rs.getString(1);
            datosCliente[1] = rs.getString(2);
            datosCliente[2] = rs.getString(3);
            datosCliente[3] = rs.getString(4);
            datosCliente[4] = rs.getString(5);
            datosCliente[5] = rs.getString(6);
            modelo.addRow(datosCliente);
        }

        tblHistorialFacturas.setModel(modelo);

    }catch(SQLException ex){
```

Imagen 52 Código visualizar las facturas ordenadas según la fecha de forma descendente

Lo que hacemos es escribir el código de búsqueda SQL donde decimos que de la tabla cliente, cab_fact y det_fact muestre los datos número de factura, nombre del cliente, apellido del cliente, costo total más IVA y la fecha agrupando por número de factura y ordenando el valor total de la factura pero en forma descendente.

Imagen 53 Visualización de la facturas ordenadas de forma descendente por la fecha de emisión de la factura

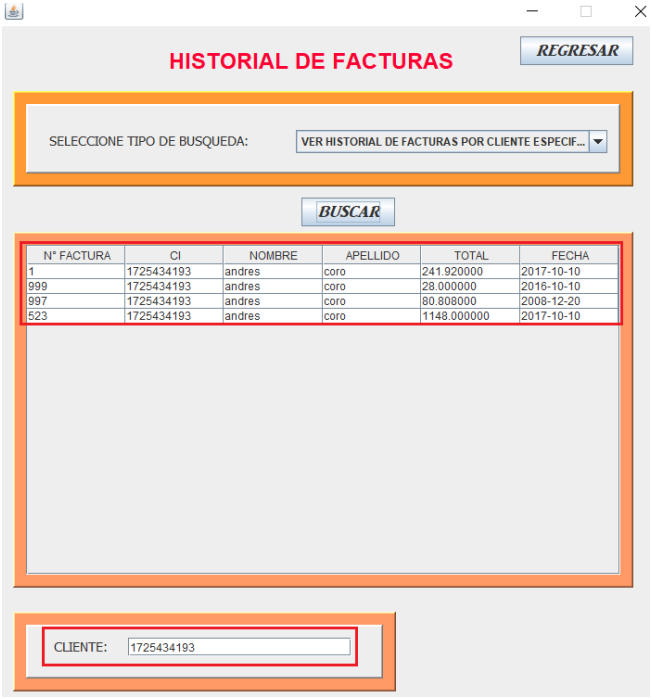
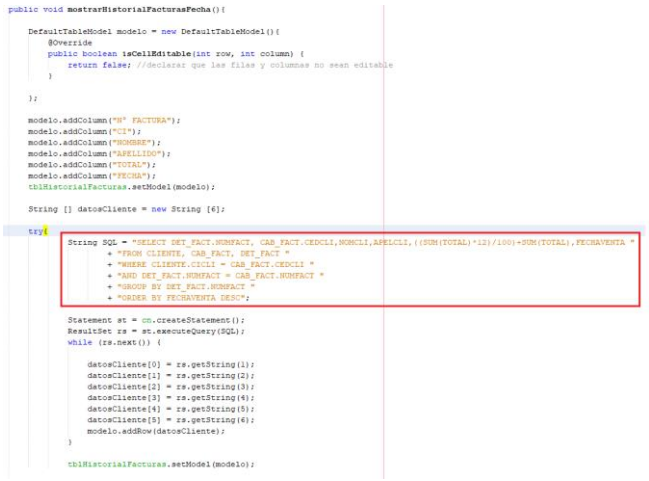
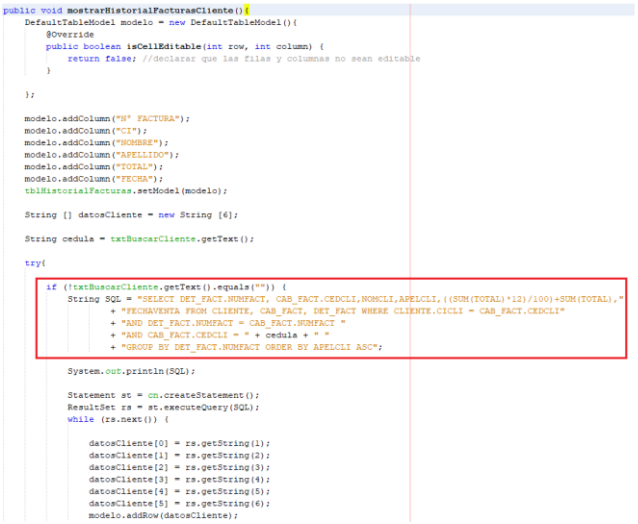


Imagen 55 Visualización de las facturas emitidas de un cliente en especifico



```
public void mostrarHistorialFacturasFecha(){

    DefaultTableModel modelo = new DefaultTableModel(){
        @Override
        public boolean isCellEditable(int row, int column) {
            return false; //declarar que las filas y columnas no sean editable
        }

    };

    modelo.addColumn("N° FACTURA");
    modelo.addColumn("CI");
    modelo.addColumn("NOMBRE");
    modelo.addColumn("APELLIDO");
    modelo.addColumn("TOTAL");
    modelo.addColumn("FECHA");
    tblHistorialFacturas.setModel(modelo);

    String [] datosCliente = new String [6];

    try{
        String SQL = "SELECT DET_FACT.NUMFACT, CAB_FACT.CEDCLI,NOMCLI,APELCLI,((SUM(TOTAL)*12)/100)+SUM(TOTAL),FECHAVENTA "
                + "FROM CLIENTE, CAB_FACT, DET_FACT "
                + "WHERE CLIENTE.CICLI = CAB_FACT.CEDCLI "
                + "AND DET_FACT.NUMFACT = CAB_FACT.NUMFACT "
                + "GROUP BY DET_FACT.NUMFACT "
                + "ORDER BY FECHAVENTA DESC";

        Statement st = cn.createStatement();
        ResultSet rs = st.executeQuery(SQL);
        while (rs.next()) {

            datosCliente[0] = rs.getString(1);
            datosCliente[1] = rs.getString(2);
            datosCliente[2] = rs.getString(3);
            datosCliente[3] = rs.getString(4);
            datosCliente[4] = rs.getString(5);
            datosCliente[5] = rs.getString(6);
            modelo.addRow(datosCliente);
        }

        tblHistorialFacturas.setModel(modelo);
```

Imagen 54. Código para visualizar las facturas ordenadas de forma descendente por la fecha de emisión de la factura.



```
public void mostrarHistorialFacturasCliente(){
    DefaultTableModel modelo = new DefaultTableModel(){
        @Override
        public boolean isCellEditable(int row, int column) {
            return false; //declarar que las filas y columnas no sean editable
        }

    };

    modelo.addColumn("N° FACTURA");
    modelo.addColumn("CI");
    modelo.addColumn("NOMBRE");
    modelo.addColumn("APELLIDO");
    modelo.addColumn("TOTAL");
    modelo.addColumn("FECHA");
    tblHistorialFacturas.setModel(modelo);

    String [] datosCliente = new String [6];

    String cedula = txtBuscarCliente.getText();

    try{
      if (!txtBuscarCliente.getText().equals("")) {
        String SQL = "SELECT DET_FACT.NUMFACT, CAB_FACT.CEDCLI,NOMCLI,APELCLI,((SUM(TOTAL)*12)/100)+SUM(TOTAL),"
                + "FECHAVENTA FROM CLIENTE, CAB_FACT, DET_FACT WHERE CLIENTE.CICLI = CAB_FACT.CEDCLI"
                + "AND DET_FACT.NUMFACT = CAB_FACT.NUMFACT "
                + "AND CAB_FACT.CEDCLI = " + cedula + " "
                + "GROUP BY DET_FACT.NUMFACT ORDER BY APELCLI ASC";

        System.out.println(SQL);

        Statement st = cn.createStatement();
        ResultSet rs = st.executeQuery(SQL);
        while (rs.next()) {

            datosCliente[0] = rs.getString(1);
            datosCliente[1] = rs.getString(2);
            datosCliente[2] = rs.getString(3);
            datosCliente[3] = rs.getString(4);
            datosCliente[4] = rs.getString(5);
            datosCliente[5] = rs.getString(6);
            modelo.addRow(datosCliente);
```

Imagen 56 Código para la visualización de las facturas emitidas de un cliente en especifico

Lo que hacemos es escribir el código de búsqueda SQL donde decimos que de la tabla cliente, cab_fact y det_fact muestre los datos número de factura, nombre del cliente, apellido del cliente, costo total más IVA y la fecha agrupando por número de factura y ordenando la fecha en forma descendente.

A diferencia del resto de las otras consultas aquí pido que el usuario ingrese un numero de cedula y mando a buscar la consulta SQL pero solo las facturas que sean iguales a IDCLI = al número que ingreso.

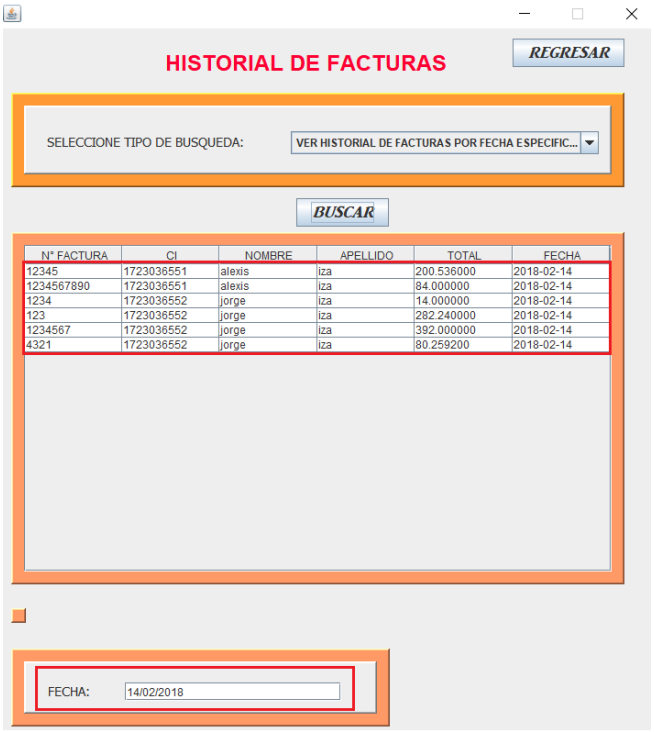


Imagen 57. Visualización de las facturas mediante una fecha especifica

```
public void mostrarHistorialFacturasFechaEspecifica(){

    DefaultTableModel modelo = new DefaultTableModel(){
        @Override
        public boolean isCellEditable(int row, int column) {
            return false; //declarar que las filas y columnas no sean editable
        }

    };

    modelo.addColumn("N° FACTURA");
    modelo.addColumn("CI");
    modelo.addColumn("NOMBRE");
    modelo.addColumn("APELLIDO");
    modelo.addColumn("TOTAL");
    modelo.addColumn("FECHA");
    tblHistorialFacturas.setModel(modelo);

    String [] datosCliente = new String [6];

    try{

        if(!txtFechaBusqueda.getText().equals("")){
            String fecha = txtFechaBusqueda.getText();
            boolean fechaValidar = validarFecha(fecha);

            if(fechaValidar == true){

                System.out.println("FECHA FACT "+FechaFact);

            String SQL = "SELECT DET_FACT.NUMFACT, CAB_FACT.CEDCLI,NOMCLI,APELCLI,((SUM(TOTAL)*12)/100)+SUM(TOTAL),FECHAVENTA "
                    + "FROM CLIENTE, CAB_FACT, DET_FACT "
                    + "WHERE CLIENTE.CICLI = CAB_FACT.CEDCLI "
                    + "AND DET_FACT.NUMFACT = CAB_FACT.NUMFACT "
                    + "AND CAB_FACT.FECHAVENTA = '%' + FechaFact + "') "
                    + "GROUP BY DET_FACT.NUMFACT "
                    + "ORDER BY FECHAVENTA DESC";

            Statement st = cn.createStatement();
            ResultSet rs = st.executeQuery(SQL);
            while (rs.next()) {

                datosCliente[0] = rs.getString(1);
                datosCliente[1] = rs.getString(2);
                datosCliente[2] = rs.getString(3);
                datosCliente[3] = rs.getString(4);
                datosCliente[4] = rs.getString(5);
                datosCliente[5] = rs.getString(6);
                modelo.addRow(datosCliente);
            }

            tblHistorialFacturas.setModel(modelo);
            }else{
                JOptionPane.showMessageDialog(null, "    FORMATO DE FECHA INCORRECTO dd/mm/yyyy   ");
                txtFechaBusqueda.setText("");
            }
        }else{
            JOptionPane.showMessageDialog(null, "    INGRESE UNA FECHA PARA BUSCAR   ");
        }
    } catch (SQLException ex){

    }

}
```

Imagen 58 Código para la visualización de las facturas mediante una fecha especifica

En cambio, en esta consulta pido que el usuario ingrese una fecha y mando a buscar la consulta SQL pero solo las facturas que sean iguales a FECHAVENTA = a la fecha que el usuario ingreso.

REGISTRO DE ACTIVIDADES

Tabla 1 Registro de actividades por cada integrante del grupo

MIEBROS DEL GRUPO	ACTIVIDADES		
JENNY TIPÁN	*Instalación del servidor XAMPP*	*ELABORACIÓN DEL INFORME*	*PRESENTACIÓN EN POWER POINT*
	Conexión de la base de datos PhpMyAdmin con el proyecto de Netbeans		
	Creación de tablas en PhpMyAdmin		
	Creación de la interfaz de login		
JORGE IZA	*Creación de la interfaz y funcionamiento de la interfaz menú*		
	Creación y funcionamiento de la interfaz registrar Producto		
	Creación y funcionamiento de la interfaz registrar Cliente		
	Unión general del código		
JENNY TIPÁN	*Creación y funcionamiento de la interfaz registrar productos*		
	Creación y funcionamiento de la interfaz historial de facturas		
	Creación de la búsqueda de facturas según un cliente en especifico		
	Creación de la búsqueda de facturas según una fecha en especifico		

CONCLUSIONES

Luego de realizar la implementación de este programa hemos sacado las siguientes conclusiones:

- *Mediante este proyecto hemos aprendido a administrar una base de datos creada en PhpMyAdmin con el fin de poder interactuar mediante interfaces graficas es decir hemos podido registrar datos, obtener datos de la base y sobre todo lo más importante pudimos realizar consultas utilizando lenguaje Java y SQL*
- *Es muy importante realizar validaciones como por ejemplo en nuestro caso que en la columna de cedula de la tabla cab_fact se registre por lo menos los 10 dígitos necesarios para que nuestro programa sea consistente.*
- Es muy importante descargar todos los programas necesarios para la implementación del sistema de facturación, ya que como sabemos el driver que se utiliza para realizar la conexión no siempre funciona y esto depende de la versión de netbeans
- *Trabajar con una cuenta de GitHub nos proporciona ayuda para compartir las actualizaciones que realizamos de nuestro código.*

PROBLEMAS ENCONTRADOS

- *Uno de los problemas que más me demoro en solucionarlo es que al momento de crear un Id en la tabla de detalle de factura sea autoincrementadle ya que se borraba un elemento el generador duplicaba un valor y no era posible guardar es detalle de la compra en la factura*
- *Para realizar una búsqueda mediante una fecha que ingrese el cliente es importante definir primero en la tabla que tipo de formato vamos a dar al momento de crear la tabla y la columna fecha ya que cada tipo de fecha tiene un formato y si no se da dicho formato se produce un error*
- *Al momento de cargar los productos en un comboBox no le vi muy factible colocar solo el código del producto por lo que decidimos colocar el detalle del producto es decir el nombre pero vimos un problema de que se podían duplicar los nombre, por lo que decidimos colocar al detalle del producto como un campo UNIQUE lo que nos permitió que cada nombre que se almacene de un producto deberá se único.*

RECOMENDACIONES

- *Verificar la compatibilidad entre las versiones de los programas que se van a utilizar, ya que al final se pueden presentar problemas que son un tanto complicados de resolver, como cambiar de versión de Python lo que indica que se tiene que cambiar la versión de todos los programas utilizados.*
- *Usar una cuenta de GitHub para compartir archivos, ya que en este se puede ir conservando versiones anteriores de los mismos, que serían útiles en caso de que se quiera trabajar con una versión anterior.*

RESULTADOS

Ya implementado el código podemos observar que el programa funciona correctamente. Se cumple con el objetivo de que mediante interfaces graficas podamos administrar la base de datos añadiendo datos, obteniendo datos y realizado consultas SQL en Netbeans utilizando lenguaje Java.

REFERENCIA

[1]https://computerhoy.com/paso-a-paso/software/como-instalar-servidor-web-tu-ordenador-xampp-48132

[2] https://es.wikipedia.org/wiki/NetBeans

[3] https://www.softonic.com/articulos/que-es-java

[4]http://evilnapsis.com/2016/03/29/conectar-una-base-de-datos-mysql-con-java-y-netbeans/

[5] https://www.mysql.com/products/connector/