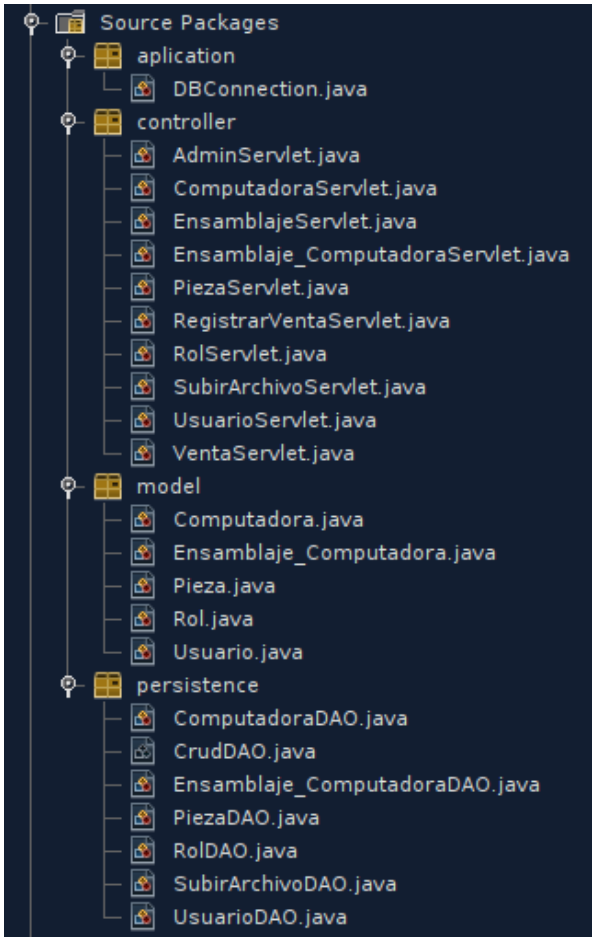
UNIVERDAD DE SAN CARLOS DE GUATEMALA
CENTRO UNIVERSITARIO DE OCCIDENTE DIVISIÓN DE CIENCIAS DE LA INGENIERÍA
INTRODUCCIÓN A LA PROGRAMACION Y COMPUTACION 2
PROYECTO 1
PRIMER SEMESTRE 2025
Andy Jefferson González Fuentes 202231338

MANUALES TECNICO

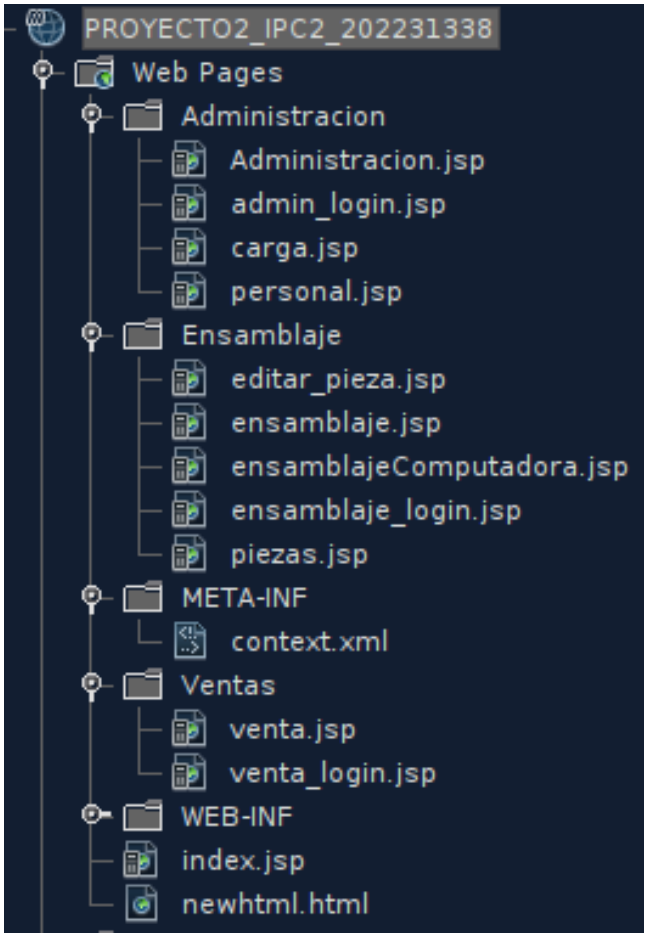
Sistema Operativo Ubuntu
Nombre del dispositivo DESKTOP-QMQJJV4
Procesador AMD Ryzen 7 3700U with Radeon Vega Mobile Gfx
RAM instalada 8.00 GB (5.88 GB utilizable)
IDE NetBens Lenguaje de programación Java
mysql-connector-j_9.2.0-1ubuntu22.04_all
apache-tomcat-9.0.99
8.0.41-0ubuntu0.22.04.1 (Ubuntu)

PAQUETES DEL PROGRAMA

BACKEND



FRONTED

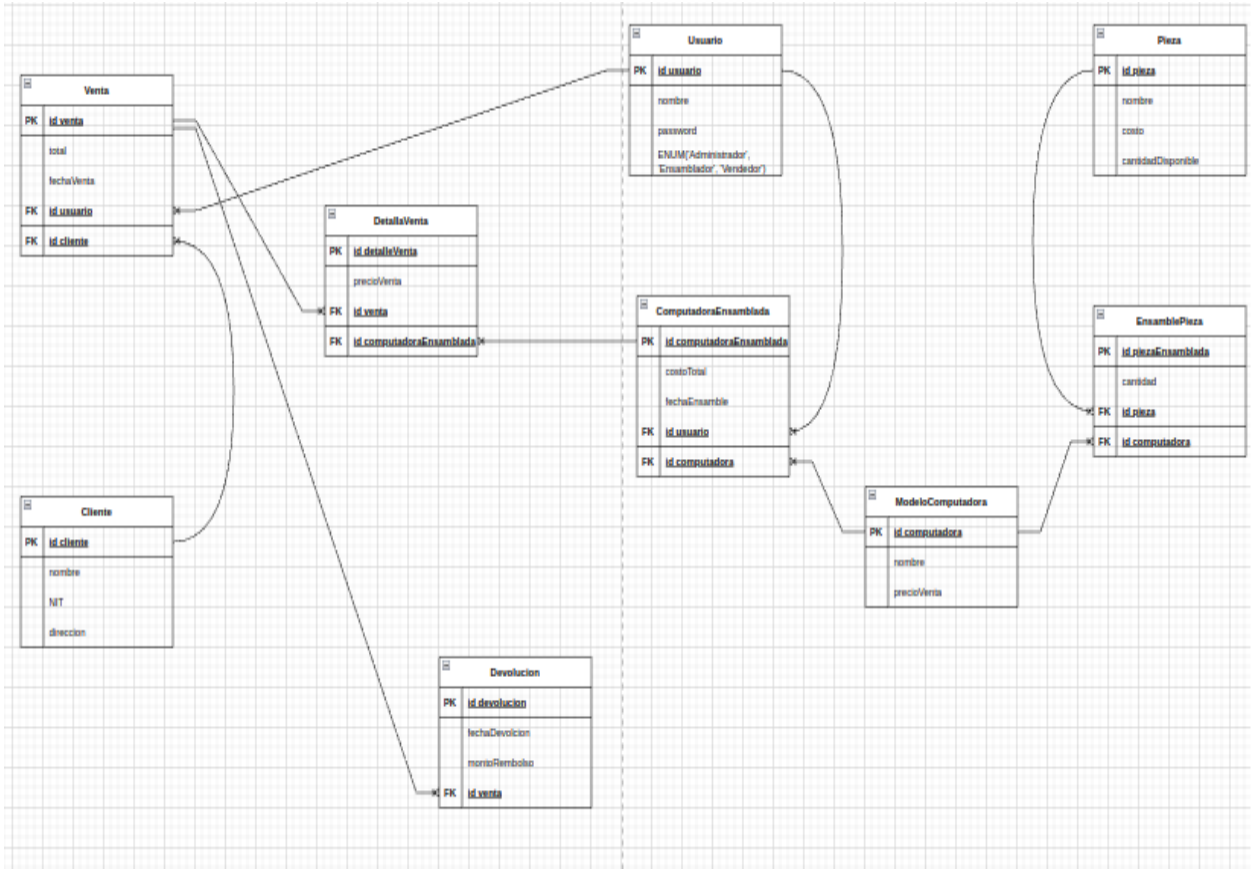


CONEXIÓN A LA BASE DE DATOS

Clase creada con el objetivo de evitar estar llamando recursos cada que se quiere realizar una consulta a la base de datos.

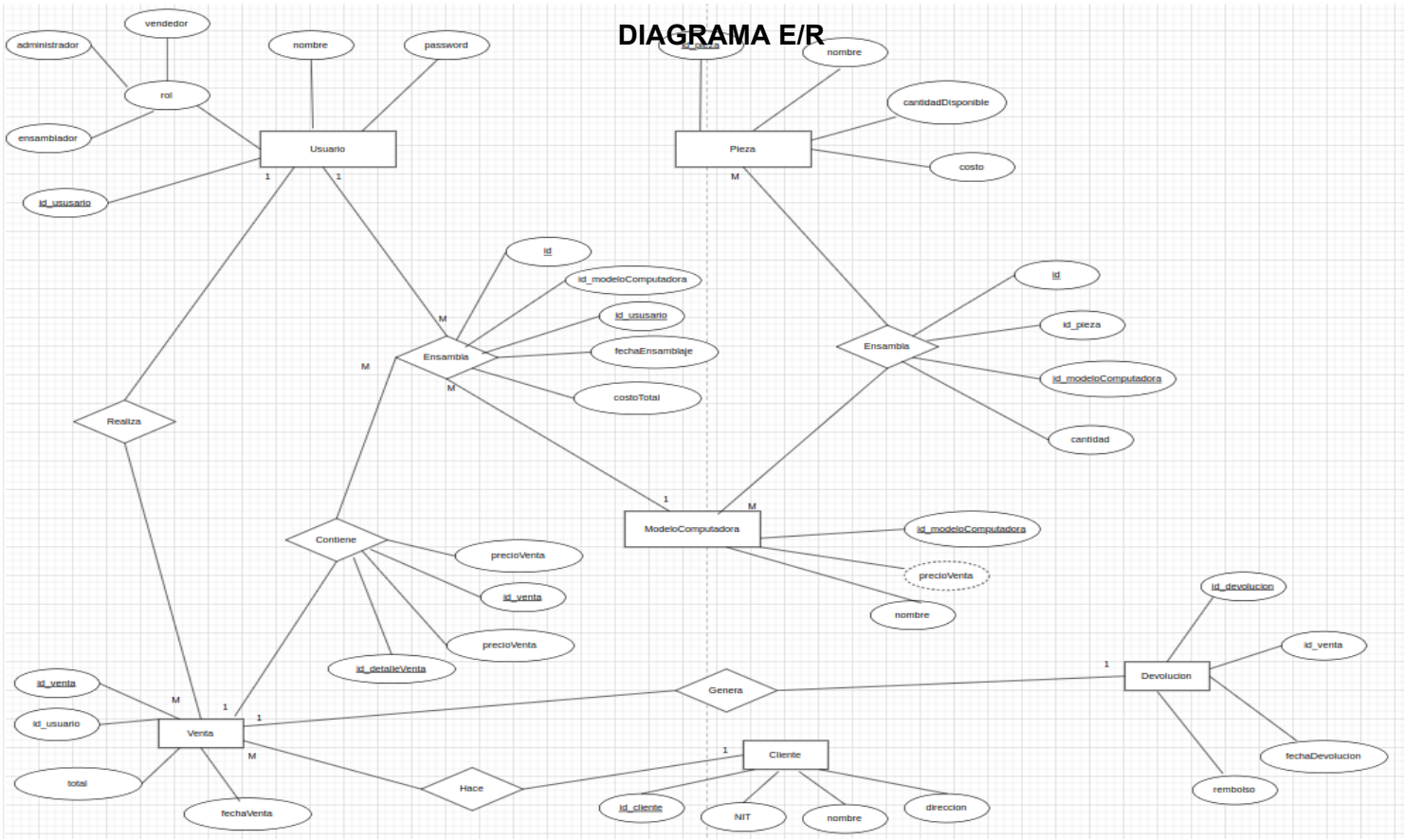
```
public class DBConnection {

    private static final String URL = "jdbc:mysql://localhost:3306/proyecto1";
    private static final String USER = "admin";
    private static final String PASSWORD = "admin";


    private static Connection connection;

    static {
        try {

            Class.forName("com.mysql.cj.jdbc.Driver");

        } catch (ClassNotFoundException e) {
            throw new RuntimeException("Error al cargar el driver JDBC ", e);
        }
    }

    private DBConnection() {}

    public static Connection getConnection() throws SQLException {
        if (connection == null || connection.isClosed()) {
            synchronized (DBConnection.class) {
                if (connection == null || connection.isClosed()) {
                    connection = DriverManager.getConnection(URL, USER, PASSWORD);
                }
            }
        }
        return connection;
    }
}
```

TABLAS DE LA BASE DE DATOS

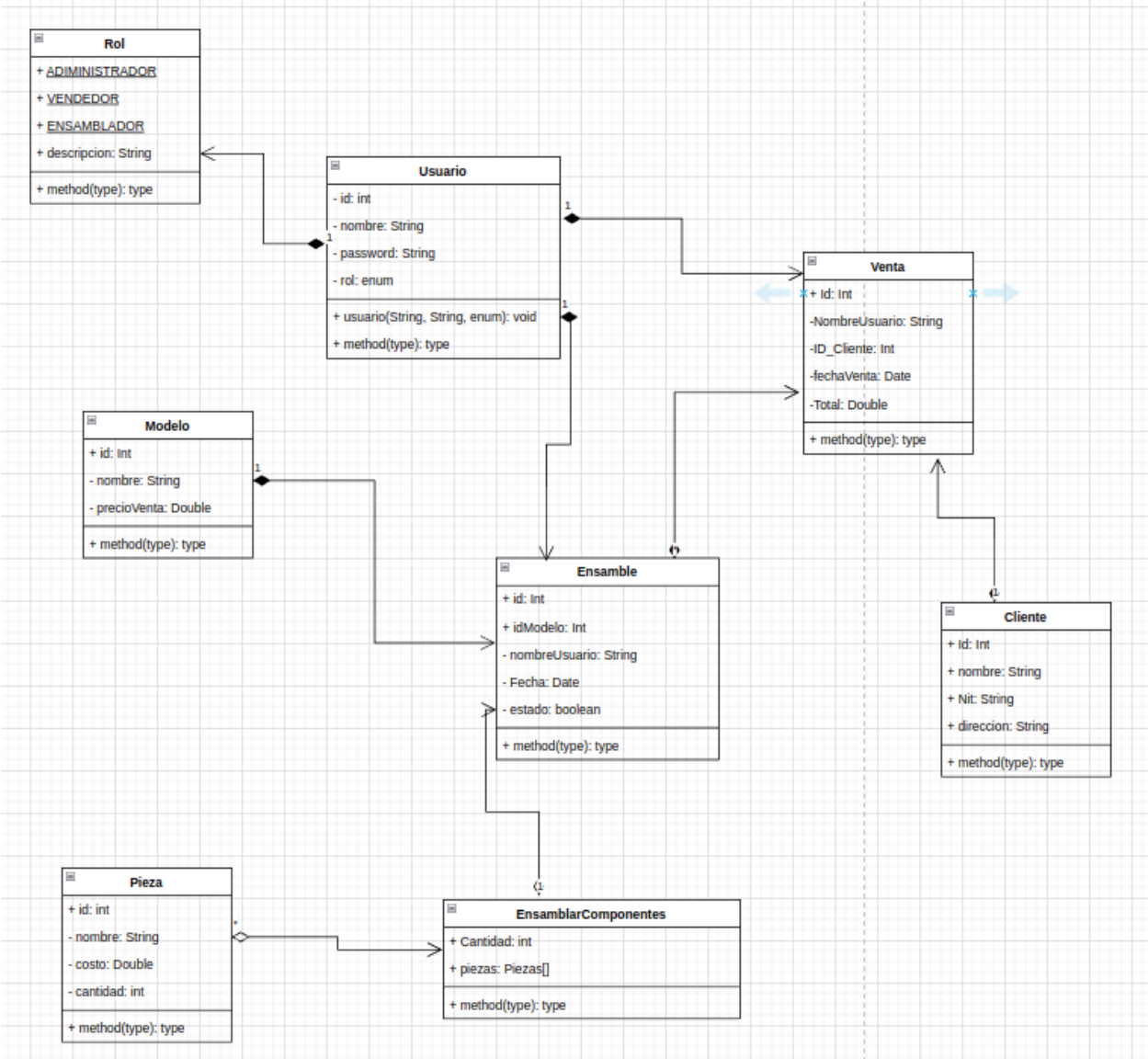


https://drive.google.com/file/d/1SxBpc-rIgCl4Mib2NrBWJXV45RpqzfnS/view?usp=sharing

DIAGRAMA E/R



https://drive.google.com/file/d/11GOEn9iQ-rQO0JbaWKGh8p62aBS5sbH-/view?usp=sharing



CLASES IMPORTANTES DENTRO DEL PROYECTO

Método abstracto del cual todas mis clases DAO heredaran las funciones para realizar consultas a la base de datos.

```
public abstract class CrudDAO<T>{
    public abstract T insert(T entity) throws SQLException;
    public abstract T findById(String id) throws SQLException;//string en id
    public abstract List<T> findAll() throws SQLException;
    public abstract void update(T entity) throws SQLException;
    public abstract void delete(String id) throws SQLException;//string en id
```

Clase en la que se realiza el recibir un archivo para lectura.

```
@WebServlet(urlPatterns = {"/SubirArchivoServlet"})
@MultipartConfig(
    fileSizeThreshold = 1024 * 1024 * 2, // 2MB
    maxFileSize = 1024 * 1024 * 10,      // 10MB
    maxRequestSize = 1024 * 1024 * 50    // 50MB
)
public class SubirArchivoServlet extends HttpServlet {

    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");

    }

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        processRequest(request, response);
    }

    private SubirArchivoDAO subirArchivoDAO = new SubirArchivoDAO();
    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {

        Part filePart = request.getPart("archivo");
        if (filePart == null || filePart.getSize() == 0) {
            request.setAttribute("mensaje", "Error: No se ha subido ningún archivo.");
            //request.getRequestDispatcher(request.getContextPath() +
"/Administracion/carga.jsp").forward(request, response);
            return;
        }

        try (InputStream fileContent = filePart.getInputStream()) {
            String resultado = subirArchivoDAO.procesarArchivo(fileContent);
            request.setAttribute("mensaje", "Se ha subido a la base de datos" + resultado);
        } catch (SQLException ex) {
            Logger.getLogger(SubirArchivoServlet.class.getName()).log(Level.SEVERE, null, ex);
        }
        request.getRequestDispatcher("/Administracion/carga.jsp").forward(request, response);
    }
}
```

Este `Pattern` está diseñado para reconocer comandos con el siguiente formato:
USUARIO("nombreUsuario", "contraseña", idRol)

Desglose de la expresión regular:

- `USUARIO\\(`
  - Busca exactamente la palabra **"USUARIO("**.
  - Se usa `\\(` porque `(` tiene un significado especial en expresiones regulares, así que lo escapamos con `\\`.
- `"([^\"]+)"`
  - Captura el **nombre de usuario** encerrado entre comillas dobles `"..."`.
  - `[^\"]+` significa "cualquier secuencia de caracteres excepto comillas dobles".
- `,"([^\"]+)"`
  - Captura la **contraseña** de manera similar a como se capturó el nombre de usuario.
- `,(\\d+)`
  - Captura un número entero (`\\d+`), que representa el **ID del rol** del usuario.
- `\\)`
  - Indica el cierre del comando `)`.

Dentro de `insertarUsuario()`, se usa un `Matcher` para extraer los valores:

```
Matcher matcher = USUARIO_PATTERN.matcher(linea);
if (matcher.find()) {
   String nombreUsuario = matcher.group(1);  // Captura el primer grupo: nombreUsuario
   String contrasena = matcher.group(2);    // Captura el segundo grupo: contraseña
   int idRol = Integer.parseInt(matcher.group(3)); // Captura el tercer grupo: idRol
}
```

```
public class SubirArchivoDAO {


   private static final Pattern USUARIO_PATTERN =
Pattern.compile("USUARIO\\(\"([^\"]+)\",\"([^\"]+)\",(\\d+)\\)");
   private static final Pattern PIEZA_PATTERN =
Pattern.compile("PIEZA\\(\"([^\"]+)\",\\s*(\\d+\\.\\d{2})\\)");
```

```
   private static final Pattern COMPUTADORA_PATTERN =
Pattern.compile("COMPUTADORA\\(\"([^\"]+)\",\\s*(\\d+\\.\\d{2})\\)");
   private static final Pattern ENSAMBLE_PIEZAS_PATTERN =
Pattern.compile("ENSAMBLE_PIEZAS\\(\"([^\"]+)\",\\s*\"([^\"]+)\",\\s*(\\d+)\\)");
   private static final Pattern ENSAMBLAR_COMPUTADORA_PATTERN =
Pattern.compile("ENSAMBLAR_COMPUTADORA\\(\"([^\"]+)\",\\s*([a-zA-Z0-9_]+),\\s*\"(\\d{2}/\\d{
2}/\\d{4})\"\\)");


   public String procesarArchivo(InputStream fileContent) throws SQLException, IOException {
      System.out.println("ENTRO A PROCESARARCHIVO xxx");
      StringBuilder resultado = new StringBuilder();

      try (BufferedReader reader = new BufferedReader(new InputStreamReader(fileContent,
StandardCharsets.UTF_8));
           Connection conn = DBConnection.getConnection()) {

         String linea;
         while ((linea = reader.readLine()) != null) {
            linea = linea.trim();
            // Normalizar comillas tipográficas a comillas estándar
            linea = linea.replace("“", "\"").replace("”", "\"");
            System.out.println("linea: "+linea);

            if (USUARIO_PATTERN.matcher(linea).matches()) {
               resultado.append(insertarUsuario(linea, conn)).append("\n");
            } else if (PIEZA_PATTERN.matcher(linea).matches()) {
               resultado.append(insertarPieza(linea, conn)).append("\n");
            } else if (COMPUTADORA_PATTERN.matcher(linea).matches()) {
               resultado.append(insertarComputadora(linea, conn)).append("\n");
            } else if (ENSAMBLE_PIEZAS_PATTERN.matcher(linea).matches()) {
               resultado.append(insertarEnsamblajePieza(linea, conn)).append("\n");
            }else if (ENSAMBLAR_COMPUTADORA_PATTERN.matcher(linea).matches()) {
               resultado.append(insertarEnsamblajeComputadora(linea, conn)).append("\n");
            }else {
               System.out.println("NO CONICIDIO");
               resultado.append("Error en línea: " + linea + " - Comando no reconocido\n");
            }
         }
      }

      return resultado.toString();
   }
```

```
    private String insertarUsuario(String linea, Connection conn) throws SQLException {
        System.out.println("    entre a ingresar el ususario");
        Matcher matcher = USUARIO_PATTERN.matcher(linea);
        if (matcher.find()) {
            String nombreUsuario = matcher.group(1);
            String contrasena = matcher.group(2);
            int idRol = Integer.parseInt(matcher.group(3));

            if (contrasena.length() < 8) {
                return "Error: La contraseña de " + nombreUsuario + " debe tener al menos 8
caracteres";
            }

            String checkSql = "SELECT COUNT(*) FROM Usuario WHERE NombreUsuario = ?";
            try (PreparedStatement checkStmt = conn.prepareStatement(checkSql)) {
                checkStmt.setString(1, nombreUsuario);
                ResultSet rs = checkStmt.executeQuery();
                if (rs.next() && rs.getInt(1) > 0) {
                    return "Error: El usuario " + nombreUsuario + " ya existe";
                }
            }

            String sql = "INSERT INTO Usuario (NombreUsuario, Contrasena, ID_Rol) VALUES (?,
?, ?)";
            try (PreparedStatement stmt = conn.prepareStatement(sql)) {
                stmt.setString(1, nombreUsuario);
                stmt.setString(2, contrasena);
                stmt.setInt(3, idRol);
                stmt.executeUpdate();
                System.out.println("se guardo"+nombreUsuario+","+contrasena+","+idRol);
                return "Usuario " + nombreUsuario + " insertado correctamente";
            }
        }
        return " Error en el formato de USUARIO";
    }


    private String insertarPieza(String linea, Connection conn) throws SQLException {
        System.out.println("entro a insertar piezas");
    Matcher matcher = PIEZA_PATTERN.matcher(linea);
    if (matcher.find()) {
        String nombre = matcher.group(1);
        double costo = Double.parseDouble(matcher.group(2));
```

```
        // Verificar si la pieza ya existe con el mismo nombre y costo
        String checkSql = "SELECT ID, CantidadDisponible FROM Pieza WHERE Nombre = ?
AND Costo = ?";
        try (PreparedStatement checkStmt = conn.prepareStatement(checkSql)) {
            checkStmt.setString(1, nombre);
            checkStmt.setDouble(2, costo);
            ResultSet rs = checkStmt.executeQuery();
            if (rs.next()) {
                int piezaId = rs.getInt("ID");
                int cantidadActual = rs.getInt("CantidadDisponible");

                // Actualizar cantidad disponible en la pieza existente
                String updateSql = "UPDATE Pieza SET CantidadDisponible = ? WHERE ID = ?";
                try (PreparedStatement updateStmt = conn.prepareStatement(updateSql)) {
                    updateStmt.setInt(1, cantidadActual + 1);
                    updateStmt.setInt(2, piezaId);
                    updateStmt.executeUpdate();
                    return "Pieza " + nombre + " actualizada correctamente. Nueva cantidad: " +
(cantidadActual + 1);
                }
            }
        }

        // Si no existe, insertamos una nueva pieza con cantidad 1
        String sql = "INSERT INTO Pieza (Nombre, Costo, CantidadDisponible) VALUES (?, ?, 1)";
        try (PreparedStatement stmt = conn.prepareStatement(sql)) {
            stmt.setString(1, nombre);
            stmt.setDouble(2, costo);
            stmt.setInt(3, 1); // Se asegura de insertar la cantidad
            stmt.executeUpdate();
            return "Pieza " + nombre + " insertada correctamente con cantidad 1";
        }
    }
    return "Error en el formato de PIEZA";
}


    private String insertarComputadora(String linea, Connection conn) throws SQLException {
        Matcher matcher = COMPUTADORA_PATTERN.matcher(linea);
        if (matcher.find()) {
            String nombre = matcher.group(1);
            double precioVenta = Double.parseDouble(matcher.group(2));

            String checkSql = "SELECT COUNT(*) FROM Computadora WHERE Nombre = ?";
```

```
            try (PreparedStatement checkStmt = conn.prepareStatement(checkSql)) {
                checkStmt.setString(1, nombre);
                ResultSet rs = checkStmt.executeQuery();
                if (rs.next() && rs.getInt(1) > 0) {
                    return "Error: La computadora " + nombre + " ya existe";
                }
            }

            String sql = "INSERT INTO Computadora (Nombre, PrecioVenta) VALUES (?, ?)";
            try (PreparedStatement stmt = conn.prepareStatement(sql)) {
                stmt.setString(1, nombre);
                stmt.setDouble(2, precioVenta);
                stmt.executeUpdate();
                return "Computadora " + nombre + " insertada correctamente";
            }
        }
        return "Error en el formato de COMPUTADORA";
    }

    private String insertarEnsamblajePieza(String linea, Connection conn) throws SQLException
{
    Matcher matcher = ENSAMBLE_PIEZAS_PATTERN.matcher(linea);
    if (matcher.find()) {
        String nombreComputadora = matcher.group(1);
        String nombrePieza = matcher.group(2);
        int cantidad = Integer.parseInt(matcher.group(3));

        // Buscar el ID de la computadora
        String queryComputadora = "SELECT ID FROM Computadora WHERE Nombre = ?";
        int idComputadora = -1;
        try (PreparedStatement stmt = conn.prepareStatement(queryComputadora)) {
            stmt.setString(1, nombreComputadora);
            ResultSet rs = stmt.executeQuery();
            if (rs.next()) {
                idComputadora = rs.getInt("ID");
            } else {
                return " Error: La computadora '" + nombreComputadora + "' no existe";
            }
        }

        // Buscar el ID de la pieza
        String queryPieza = "SELECT ID FROM Pieza WHERE Nombre = ?";
        int idPieza = -1;
        try (PreparedStatement stmt = conn.prepareStatement(queryPieza)) {
```

```
            stmt.setString(1, nombrePieza);
            ResultSet rs = stmt.executeQuery();
            if (rs.next()) {
                idPieza = rs.getInt("ID");
            } else {
                return " Error: La pieza '" + nombrePieza + "' no existe";
            }
        }

        // Verificar si ya existe el ensamblaje
        String checkQuery = "SELECT Cantidad FROM Ensamble_Piezas WHERE
ID_Computadora = ? AND ID_Pieza = ?";
        try (PreparedStatement checkStmt = conn.prepareStatement(checkQuery)) {
            checkStmt.setInt(1, idComputadora);
            checkStmt.setInt(2, idPieza);
            ResultSet rs = checkStmt.executeQuery();
            if (rs.next()) {
                // Si ya existe, actualizar la cantidad
                int nuevaCantidad = rs.getInt("Cantidad") + cantidad;
                String updateQuery = "UPDATE Ensamble_Piezas SET Cantidad = ? WHERE
ID_Computadora = ? AND ID_Pieza = ?";
                try (PreparedStatement updateStmt = conn.prepareStatement(updateQuery)) {
                    updateStmt.setInt(1, nuevaCantidad);
                    updateStmt.setInt(2, idComputadora);
                    updateStmt.setInt(3, idPieza);
                    updateStmt.executeUpdate();
                    return " Ensamble actualizado: " + nombrePieza + " x" + nuevaCantidad + " en " +
nombreComputadora;
                }
            }
        }

        // Si no existe, insertar nuevo ensamblaje
        String insertQuery = "INSERT INTO Ensamble_Piezas (ID_Computadora, ID_Pieza,
Cantidad) VALUES (?, ?, ?)";
        try (PreparedStatement insertStmt = conn.prepareStatement(insertQuery)) {
            insertStmt.setInt(1, idComputadora);
            insertStmt.setInt(2, idPieza);
            insertStmt.setInt(3, cantidad);
            insertStmt.executeUpdate();
            return " Ensamble realizado: " + nombrePieza + " x" + cantidad + " en " +
nombreComputadora;
        }
    }
```

```
    return " Error en el formato de ENSAMBLE_PIEZAS";
}

private String insertarEnsamblajeComputadora(String linea, Connection conn) throws
SQLException {
    Matcher matcher = ENSAMBLAR_COMPUTADORA_PATTERN.matcher(linea);
    if (matcher.find()) {
        String nombreComputadora = matcher.group(1);
        String nombreUsuario = matcher.group(2);
        String fechaStr = matcher.group(3);

        // Buscar ID de la computadora y precio de venta
String queryComputadora = "SELECT ID, PrecioVenta FROM Computadora WHERE Nombre =
?";
int idComputadora = -1;
double precioVenta = 0;

try (PreparedStatement stmt = conn.prepareStatement(queryComputadora)) {
    stmt.setString(1, nombreComputadora);
    ResultSet rs = stmt.executeQuery();
    if (rs.next()) {
        idComputadora = rs.getInt("ID");
        precioVenta = rs.getDouble("PrecioVenta");  // Obtener el precio de venta correctamente
    } else {
        return " Error: La computadora '" + nombreComputadora + "' no existe";
    }
}

        // Validar existencia del usuario en la base de datos
        String queryUsuario = "SELECT NombreUsuario FROM Usuario WHERE NombreUsuario
= ?";
        try (PreparedStatement stmt = conn.prepareStatement(queryUsuario)) {
            stmt.setString(1, nombreUsuario);
            ResultSet rs = stmt.executeQuery();
            if (!rs.next()) {
                return " Error: El usuario '" + nombreUsuario + "' no existe";
            }
        }

        // Calcular costo total (sumar precio de venta de la computadora + costo de las piezas
ensambladas)
        String queryCostoPiezas = "SELECT COALESCE(SUM(P.Costo * EP.Cantidad), 0) AS
CostoTotal " +
                                  "FROM Ensamble_Piezas EP JOIN Pieza P ON EP.ID_Pieza = P.ID " +
```

```
                        "WHERE EP.ID_Computadora = ?";
        double costoPiezas = 0;
        try (PreparedStatement stmt = conn.prepareStatement(queryCostoPiezas)) {
            stmt.setInt(1, idComputadora);
            ResultSet rs = stmt.executeQuery();
            if (rs.next()) {
                costoPiezas = rs.getDouble("CostoTotal");
            }
        }

        double costoTotal = precioVenta + costoPiezas;  // Sumar precio de la computadora y
piezas ensambladas

        // Insertar ensamblaje
        String insertQuery = "INSERT INTO Computadora_Ensamblada (ID_Computadora,
NombreUsuario, FechaEnsamblaje, CostoTotal) VALUES (?, ?, ?, ?)";
        try (PreparedStatement stmt = conn.prepareStatement(insertQuery)) {
            stmt.setInt(1, idComputadora);
            stmt.setString(2, nombreUsuario);
            stmt.setDate(3, Date.valueOf(LocalDate.parse(fechaStr,
DateTimeFormatter.ofPattern("dd/MM/yyyy"))));
            stmt.setDouble(4, costoTotal);
            System.out.println("|"+idComputadora+"|"+nombreUsuario+"|"+costoTotal);
            stmt.executeUpdate();
            return " Ensamblaje de computadora '" + nombreComputadora + "' registrado con costo
total: $" + costoTotal;
        }
    }
    return " Error en el formato de ENSAMBLAR_COMPUTADORA";
}

}
```