Criterio C: Desarrollo

El código mostrado a continuación pertenece a la clase “ConexionSQL” que permite el programa conectarse a una base de datos en MySQL, esto me permitirá almacenar y añadir datos a la plataforma sin la necesidad de que este se encuentre conectado. Le removí la capacidad de que en un JOptionPane le avise al usuario que su conexión es exitosa porque resultaria molesto al usuario al ejecutarse el programa.

```
public class conexionsql {
    Connection conectar = null;

    public Connection conexion(){
        try{
            Class.forName("com.mysql.cj.jdbc.Driver");
            conectar = (Connection)DriverManager.getConnection("jdbc:mysql://localhost/loginai","root","daniel2020");
            //JOptionPane.showMessageDialog(null,"Conexion Exitosa!!");

        }catch(Exception e){
            JOptionPane.showMessageDialog(null, "Conexion Fallida :( " +e.getMessage());
        }


      return conectar;
    }

}
```

Este método “acceso” perteneciente a la clase “login”, permite comprobar si las credenciales introducidas por el usuario son correctas y coinciden con los valores en la base de datos. De ser correctas, el método rediccionara al usuario a la clase “listas”, de lo contrario el usuario a través del JFrameForm del método, tiene la opción de crear una cuenta presionando el JButton “botonreg” que rediccionara al usuario a la clase “regirstroform”. Toda información es comprobada a través de una sentencia “SQL”.

```
public void accesso(){
    int resultado =0;
        String pass = String.valueOf(txtpass.getPassword());
        String usuario = String.valueOf(txtUsuario.getText());
    String SQL = "SELECT * FROM inicio WHERE usuario='" + usuario + "' AND pass='" + pass + "'";
        try{
        Statement st =con.createStatement();
        ResultSet rs=st.executeQuery(SQL);
        if(rs.next()){

            resultado=1;
            JOptionPane.showMessageDialog(null,"Bienvenido!");
            menu form = new menu();
            form.setVisible(true);
            this.dispose();
        }else{
            JOptionPane.showMessageDialog(null,"Error!");
        }

        }catch(Exception e){
            JOptionPane.showMessageDialog(null, "Conexion Fallida :("+e.getMessage()) ;
        }
}
    private void botonregActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
      new registroform().setVisible(true);
      //this.dispose();
        //Activar eso si quieres cerrarlo al registrarse...no me parece sabio hacerlo
    }
```

Este metodo perteneciente a la clase “registroform” permite al usuario introducir nuevas credenciales a la base de datos mediante una sentencia “SQL”. Si los datos son correctos el sistema le avisara al usuario a traves de un JOptionPane que el proceso fue exitoso.

```
  public void añadirusuario(){
      String SQL = "insert into inicio (usuario,pass) values(?,?)";
      String pass = String.valueOf(txtpass.getPassword());
      try{
          PreparedStatement pst=con.prepareStatement(SQL);
          pst.setString(1,txtusuario.getText());
          pst.setString(2,pass);
          pst.executeUpdate();
          JOptionPane.showMessageDialog(null,"Registro Exitoso!!!");
          this.dispose();


      }catch(Exception e){
          JOptionPane.showMessageDialog(null, "Conexion Fallida :("+e.getMessage()) ;

      }

  }
```

Este metodo se encuentra en la clase “listas”, su proposito es mostrar el contenido de la tabla alumnos en una JTable. La sentencia “SQL” permite que se llamen los valores pertenecientes a la base de datos, para que asi el metodo sea capaz de recorrer los resultados de la consulta, agregando cada fila correspondiente a la tabla visual con los datos obtenidos por medio de un bucle while y un ResultSet “rs”. Además, realiza una configuración especial para una columna particular, posiblemente asociando un desplegable para opciones y un renderizador para información adicional. Cualquier error durante este proceso se notifica al usuario mediante un cuadro de diálogo.

```
private void datosTabla() {
    DefaultTableModel modelotabla = (DefaultTableModel) tablaalumnos.getModel();
   modelotabla.setRowCount(0);
    PreparedStatement pst = null;
    ResultSet rs = null;

    try {

        // Preparar la consulta SQL
        String consulta = "SELECT numero, codigo, apellidos, nombre, grado, registro_presencia, estado FROM alumnos";
        pst = con.prepareStatement(consulta);

        // Ejecutar la consulta y llenar el modelo de tabla con los resultados
        rs = pst.executeQuery();

        while (rs.next()) {
            int numero = rs.getInt("numero");
            String codigo = rs.getString("codigo");
            String apellidos = rs.getString("apellidos");
            String nombre = rs.getString("nombre");
            String grado = rs.getString("grado");
            String registro = rs.getString("registro_presencia");
            String estado = rs.getString("estado");


            modelotabla.addRow(new Object[]{numero, codigo, apellidos, nombre, grado, registro,estado});
        }
         TableColumn estadoColumn = tablaalumnos.getColumnModel().getColumn(6);
            setBox(estadoColumn);
    } catch (Exception e) {
        JOptionPane.showMessageDialog(null, "Error: " + e.getMessage());
    }
}
```

Este metodo permite la personalizacion de una columna especifica del JTable. Recibe como parametro una columna de tabla y configura un JComboBox con las opciones predefinidas presentes en el codigo. Luego, se estable que este JComboBox como el editor de celdas de la columna seleccionada por el usuario, permitiendo que este pueda elegir entre las opciones proporcionadas. Ademas, configura un renderizador de celdas que muestra un mensaje emergente cuando el usuario pasa el cursor sobre la celda, indicando que es posible modificar el estado del alumno en esa columna.

```
public void setBox(TableColumn estado) {
        JComboBox<String> c = new JComboBox<>();
        c.addItem("PRESENTE");
        c.addItem("TARDE");
        c.addItem("AUSENTE");
        c.addItem("PERMISO");
        c.addItem("VIRTUAL");
        estado.setCellEditor(new DefaultCellEditor(c));
        DefaultTableCellRenderer renderer = new DefaultTableCellRenderer();
        renderer.setToolTipText("Seleccione el estado del alumno");
        estado.setCellRenderer(renderer);
    }
```

Este método hace uso de las bibliotecas zxing (Maven Repository: com.google.zxing (mvnrepository.com), 2023, Maven Repository) para generar el qr, y qrgen para facilitar la conversión de este código QR al formato jpg, y de esta forma pueda ser visible en el programa. De esta manera, el método toma el texto ingresado de la base de datos en el campo “txtcodigo”, cree un código QR correspondiente a ese texto y lo muestre como una imagen dentro de un label llamado “QRLabel”.

```
private void GenerarQR(){
    ByteArrayOutputStream out= QRCode.from(this.txtcodigo.getText()).to(ImageType.PNG).stream();
        ImageIcon imageIcon=new ImageIcon(out.toByteArray());
        this.QRlabel.setIcon(imageIcon);
}
```

Este método se activa cuando el usuario le hace clic a una fila del JTable. Cuando esto sucede, captura la fila seleccionada y extrae los datos específicos de esta fila. Luego, coloca estos datos en múltiples JTextField. Además, llama el método GenerarQR ( ) para que este sea ejecutado y pueda mostrar un código QR basado en el código del alumno que se encuentre en el JTextField “txtcodigo”.

```
    private void tablaalumnosMouseClicked(java.awt.event.MouseEvent evt) {
        // TODO add your handling code here:

        int filaselecc = tablaalumnos.rowAtPoint(evt.getPoint());
     txtnombres.setText(tablaalumnos.getValueAt(filaselecc,3).toString());
        txtapellidos.setText(tablaalumnos.getValueAt(filaselecc,2).toString());
        txtcodigo.setText(tablaalumnos.getValueAt(filaselecc,1).toString());
        txtgrado.setText(tablaalumnos.getValueAt(filaselecc,4).toString());
         txtnum.setText(tablaalumnos.getValueAt(filaselecc,0).toString());
       GenerarQR();


    }
```

Este método permite generar un archivo de Excel a partir de los datos presentes del JTable en la interfaz de usuario. Primero, abre un JFileChooser para que el usuario pueda seleccionar la ubicación y el nombre del archivo a guardar. Luego, crea un archivo Excel en formato ".xls" y utiliza la librería Apache POI para escribir los datos de la tabla en dicho archivo. Itera sobre las filas y columnas de la tabla, creando celdas en el archivo Excel y llenándolas con los valores correspondientes. Una vez completada la escritura, cierra el archivo y lo abre automáticamente en la aplicación asociada para su visualización.

```
public void exportarExcel(JTable t) throws IOException {
    JFileChooser chooser = new JFileChooser();
    FileNameExtensionFilter filter = new FileNameExtensionFilter("Archivos de excel", "xls");
    chooser.setFileFilter(filter);
    chooser.setDialogTitle("Guardar archivo");
    chooser.setAcceptAllFileFilterUsed(false);
    if (chooser.showSaveDialog(null) == JFileChooser.APPROVE_OPTION) {
        String ruta = chooser.getSelectedFile().toString().concat(".xls");
        try {
            File archivoXLS = new File(ruta);
            if (archivoXLS.exists()) {
                archivoXLS.delete();
            }
            archivoXLS.createNewFile();
            Workbook libro = new HSSFWorkbook();
            FileOutputStream archivo = new FileOutputStream(archivoXLS);
            Sheet hoja = libro.createSheet("Mi hoja de trabajo 1");
            hoja.setDisplayGridlines(false);
            for (int f = 0; f < t.getRowCount(); f++) {
                Row fila = hoja.createRow(f);
                for (int c = 0; c < t.getColumnCount(); c++) {
                    Cell celda = fila.createCell(c);
                    if (f == 0) {
                        celda.setCellValue(t.getColumnName(c));
                    }
                }
            }
```

```
            int filaInicio = 1;
            for (int f = 0; f < t.getRowCount(); f++) {
                Row fila = hoja.createRow(filaInicio);
                filaInicio++;
                for (int c = 0; c < t.getColumnCount(); c++) {
                    Cell celda = fila.createCell(c);
                    if (t.getValueAt(f, c) instanceof Double) {
                        celda.setCellValue(Double.parseDouble(t.getValueAt(f, c).toString()));
                    } else if (t.getValueAt(f, c) instanceof Float) {
                        celda.setCellValue(Float.parseFloat((String) t.getValueAt(f, c)));
                    } else {
                        celda.setCellValue(String.valueOf(t.getValueAt(f, c)));
                    }
                }
            }
            libro.write(archivo);
            archivo.close();
            Desktop.getDesktop().open(archivoXLS);
        } catch (IOException | NumberFormatException e) {
            throw e;
        }
    }
}
private void generarActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
     try {
    // Llamar al método exportarExcel y pasar la tabla que quieres exportar
    exportarExcel(tablaalumnos);
} catch (IOException ex) {
    System.out.println("Error: " + ex);
}

}
```

Numero de Palabras: 646