Documentación del Código Fuente - Proyecto Java (MVC)

1. Datos Generales del Proyecto

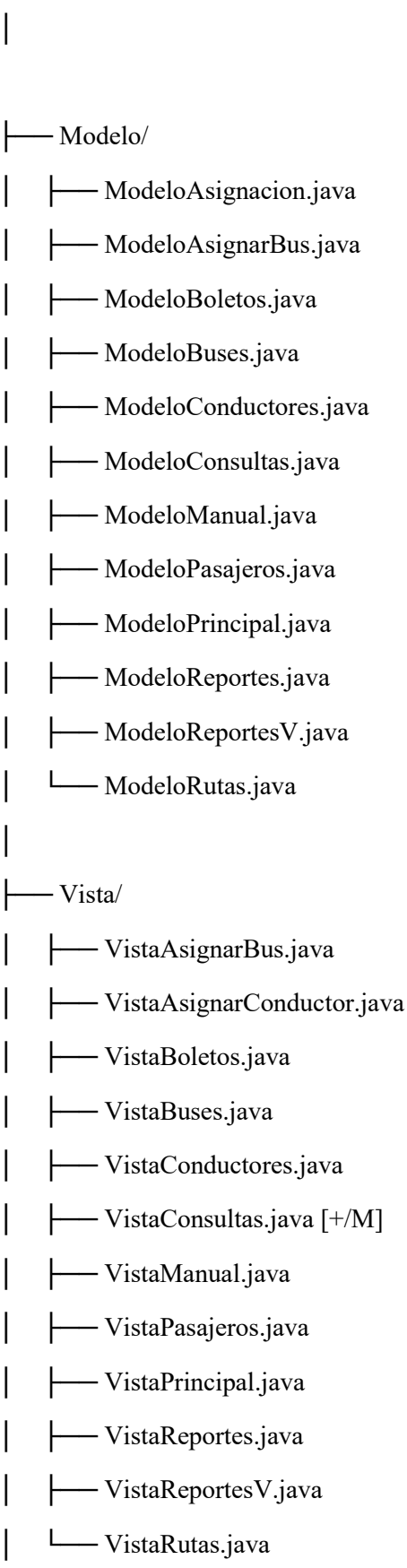
Nombre del Proyecto: Sistema para gestión de Transporte
Lenguaje: Java
Entorno de desarrollo: NetBeans
Patrón usado: MVC (Modelo - Vista - Controlador)
Fecha: Mayo 2025
Autores:

Diego José Cowo Balcarcel	1890 - 24 - 20453
Dulce Mariana Orellana Hernández	1890 - 24 – 26987
Joseph Manuel Alessandro Ruiz	1890 - 24 - 22310

2. Estructura de Carpetas del Proyecto

/src

```
├── Controlador/
│   ├── ControladorAsignacion.java
│   ├── ControladorAsignarBus.java
│   ├── ControladorBoletos.java
│   ├── ControladorBuses.java
│   ├── ControladorConductores.java
│   ├── ControladorConsultas.java
│   ├── ControladorManual.java
│   ├── ControladorPasajeros.java
│   ├── ControladorPrincipal.java
│   ├── ControladorReportes.java
│   ├── ControladorReportesV.java
│   └── ControladorRutas.java
```


Modelo/
ModeloAsignacion.java
ModeloAsignarBus.java
ModeloBoletos.java
ModeloBuses.java
ModeloConductores.java
ModeloConsultas.java
ModeloManual.java
ModeloPasajeros.java
ModeloPrincipal.java
ModeloReportes.java
ModeloReportesV.java
ModeloRutas.java
Vista/
VistaAsignarBus.java
VistaAsignarConductor.java
VistaBoletos.java
VistaBuses.java
VistaConductores.java
VistaConsultas.java [+/M]
VistaManual.java
VistaPasajeros.java
VistaPrincipal.java
VistaReportes.java
VistaReportesV.java
VistaRutas.java

└── Multimedia/
    ├── bill_2325648.png
    ├── bus (1).png
    ├── business-report_8480222.png
    ├── curriculum_1927501.png
    ├── distance_4721181.png
    ├── instruction-book_16939373.png
    ├── interview_15441435.png
    ├── speech-bubble_5726828.png
    ├── tax_12487479.png
    ├── taxi-driver_1597373.png
    ├── ticket_888629.png
    ├── time-management_8123642.png
    └── woman_9434861.png

3. Explicación del Patrón MVC Usado

El proyecto fue desarrollado siguiendo el patrón Modelo-Vista-Controlador (MVC) para separar la lógica de negocio, la presentación y el control de eventos.

Modelo: gestiona la lógica de los datos y el acceso a archivos como buses.txt, conductores.txt, etc.

Vista: contiene las interfaces gráficas hechas con JFrame y componentes Swing.

Controlador: maneja los eventos de la vista (botones, campos) y coordina las acciones con el modelo.

Multimedia: contiene todos los archivos de multimedia utilizados en el proyecto los cuales son iconos.

4. Funciones Clave del Sistema

Registro de Buses: permite ingresar datos como placa, modelo, capacidad y guardarlos en buses.txt.

Registro de Rutas: permite registrar datos de la ruta como código de ruta, origen, destino, estado, horario y frecuencia y guardarlos en rutas.txt.

Asignación de rutas: asocia una ruta a un bus que ya tenga un conductor asignado.

Asignación de Conductores: asocia un conductor disponible con un bus y actualiza su estado.

Registro de Ventas: guarda los datos del pasajero, ruta y asiento en ventas.txt.
Reportes en Excel: genera un archivo Excel desde los datos de ventas o conductores usando Apache POI.

5. Explicación de Código

En el código se encuentran varios métodos comunes que cumplen funciones como cargar datos, guardar, verificar datos, asignar o desasignar y también actualizar.

Por ejemplo en esta parte se cargan los datos de un bus por medio de un ComboBox:

```
public void cargarBuses () {
    modelo.getVista().cmbBusesA.removeAllItems();

    try (BufferedReader reader = new BufferedReader(new FileReader(archivoBuses))) {
        String linea;
        while ((linea = reader.readLine()) != null) {
            String[] datos = linea.split("\\|");
```

```
            if (datos.length >= 4 && datos[3].trim().equalsIgnoreCase("Disponible")) {
                modelo.getVista().cmbBusesA.addItem(datos[0].trim());
            }
        }
    } catch (IOException ex) {
        JOptionPane.showMessageDialog(null, "Error al cargar buses disponibles.");
    }
}
```

Estos métodos como cargar se escriben en las partes del inicio de la ventana ya que cuando una ventana de inicializa tiene que cargar los datos automáticamente ya sea a un ComboBox o una tabla.

Lo que está a continuación es un método para cargar los datos de un conductor en la tabla que está en la ventana que se está trabajando, los datos los toma del archivo conductores.txt

```
public void cargarConductores() {
    DefaultTableModel model = (DefaultTableModel) modelo.getVista().tblConductoresA.getModel();
    model.setRowCount(0);

    try (BufferedReader reader = new BufferedReader(new FileReader(archivoConductores))) {
        String linea;
        while ((linea = reader.readLine()) != null) {
            String[] datos = linea.split("\\|");

            if (datos.length >= 10) {
                String estado = datos[7].trim();
                String fechaV = datos[9].trim();

                SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
                Date FechaL = sdf.parse(fechaV);
```

```
                if (estado.equalsIgnoreCase("Disponible") || estado.equalsIgnoreCase("Asignado")
&& FechaL.after(new Date())) {
                    model.addRow(new Object[]{
                        datos[0].trim(),
                        datos[1].trim(),
                        datos[2].trim(),
                        datos[5].trim(),
                        datos[7].trim(),
                        datos[8].trim()
                    });
                }
            }
        }
    } catch (Exception ex) {
        JOptionPane.showMessageDialog(null, "Error al cargar conductores disponibles.");
    }
}
```

Luego se encuentran los métodos de guardado o asignación, estos tienen cosas en común ya que cuando se guarda una venta hay varios archivos involucrados como los buses ya que debe restar su capacidad cada vez que se realice una venta. Luego hay otro ejemplo cuando se asigna un bus a una ruta, cambia su estado de asignado a ruta asignada, esto quiere decir que también se edita el archivo ruta, al final todos los datos están relacionados para un buen funcionamiento del sistema.

A continuación, un ejemplo del método para registrar venta:

```
private void registrarVenta() {
    String nombre = modelo.getVista().txtNombre.getText().trim();
    String dpi = modelo.getVista().txtDPI.getText().trim();
    String telefono = modelo.getVista().txtTelefono.getText().trim();
```

```
        String codigoRuta = modelo.getVista().txtCodigoRuta.getText().trim();
        String nombreRuta = modelo.getVista().cmbRuta.getSelectedItem().toString().trim();
        String destino = modelo.getVista().txtDestino.getText().trim();
        String horario = modelo.getVista().txtHorario.getText().trim();
        String precio = modelo.getVista().txtPrecio.getText().trim();
        String placaBus = modelo.getVista().txtPlaca.getText().trim();

        Date fechaViaje = modelo.getVista().dcFecha.getDate();
        if (fechaViaje == null) {
            JOptionPane.showMessageDialog(null, "Seleccione una fecha de viaje.");
            return;
        }

        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
        String fechaFormateada = sdf.format(fechaViaje);
        if (dpi.length() != 13 || !dpi.matches("\\d+")) {
            JOptionPane.showMessageDialog(null, "El DPI debe contener exactamente 13 dígitos
numéricos.");
            return;
        }
        // Validación de capacidad
        int asientos;
        try {
            asientos = Integer.parseInt(modelo.getVista().txtAsientos.getText().trim());
        } catch (NumberFormatException e) {
            JOptionPane.showMessageDialog(null, "La capacidad del bus no es válida.");
            return;
```

```
        }


        int nuevoAsiento = asientos - 1;
        if (nuevoAsiento < 0) {
            JOptionPane.showMessageDialog(null, "No hay asientos disponibles en esta ruta.");
            return;
        }


        // Registrar venta
        try (BufferedWriter writer = new BufferedWriter(new FileWriter(archivoVentas, true))) {
            writer.write(codigoRuta + " | " + nombreRuta + " | " + destino + " | " + horario + " | " +
precio + " | " + placaBus + " | " + fechaFormateada + " | " + nombre + " | " + dpi + " | " +
telefono);
            writer.newLine();
        } catch (IOException e) {
            JOptionPane.showMessageDialog(null, "Error al registrar la venta");
            return;
        }


        // Registrar pasajero si no existe
        if (!pasajeroYaRegistrado(dpi)) {
            try (BufferedWriter writer = new BufferedWriter(new FileWriter(archivoPasajeros, true)))
{
                writer.write(nombre + " | " + dpi + " | " + telefono + " | " + fechaFormateada);
                writer.newLine();
            } catch (IOException e) {
                JOptionPane.showMessageDialog(null, "Error al guardar pasajero");
            }
```

```
        }
        // Verificar si la ruta sigue "Asignada"
        if (!rutaAsignada(codigoRuta)) {
            JOptionPane.showMessageDialog(null, "Esta ruta ya no está disponible para ventas.");
            cargarRutas(); // Opcional: refresca el comboBox
            return;
        }


        // Actualizar capacidad, estados y limpiar
        actualizarAsientos(placaBus, nuevoAsiento); // Actualiza archivo buses.txt
        actualizarConductorYRuta(placaBus, codigoRuta, nuevoAsiento); // Cambia estado si
capacidad == 0
        actualizarCapacidadAsignacion(placaBus, nuevoAsiento);
        JOptionPane.showMessageDialog(null, "Venta registrada correctamente");
        limpiarCampos();
        cargarRutas();
    }
```

Como siempre es importante los llamados a los métodos en las acciones que realiza cada ventana, por ejemplo, al oprimir registrar debe hacer el llamado a el método registrarVenta(), como se muestra a continuación:

```
public void actionPerformed(ActionEvent e) {
        if (e.getSource() == modelo.getVista().cmbRuta) {
            Object seleccion = modelo.getVista().cmbRuta.getSelectedItem();
            if (seleccion != null) {
                mostrarDatosRuta(seleccion.toString());
            }
        } else if (e.getSource() == modelo.getVista().btnRegistrarVenta) {
```

```
            registrarVenta();
        }
    }
```

A lo largo del código se encuentran métodos parecidos ya que casi en todas las ventanas se realizan las mismas gestiones, a excepción de algunas como consultas y pasajeros las cuales solo son ventanas para visualizar datos y eliminarlos, a continuación se muestra un método de la ventana pasajeros:

```
public void eliminarFilaSeleccionada() {
    int fila = modelo.getVista().tblPasajeros.getSelectedRow();

    if (fila == -1) {
        JOptionPane.showMessageDialog(null, "Seleccione una fila para eliminar.");
        return;
    }

    // Obtener el DPI desde la fila seleccionada
    String dpiSeleccionado = modelo.getVista().tblPasajeros.getValueAt(fila, 1).toString();

    File temporal = new File("temporal.txt");

    try (
        BufferedReader br = new BufferedReader(new FileReader(archivoPasajeros));
        BufferedWriter bw = new BufferedWriter(new FileWriter(temporal))
    ) {
        String linea;
        boolean eliminado = false;
```

```
while ((linea = br.readLine()) != null) {
    String[] datos = linea.split("\\|");

    // Saltar líneas mal formateadas
    if (datos.length != 4) {
        bw.write(linea);
        bw.newLine();
        continue;
    }

    String dpi = datos[1].trim();

    if (!dpi.equals(dpiSeleccionado)) {
        bw.write(linea);
        bw.newLine();
    } else {
        eliminado = true;
    }
}

br.close();
bw.close();

if (archivoPasajeros.delete() && temporal.renameTo(archivoPasajeros)) {
    if (eliminado) {
        JOptionPane.showMessageDialog(null, "Pasajero eliminado correctamente.");
```

```
                cargarTabla();
                limpiarCampos();
            } else {
                JOptionPane.showMessageDialog(null, "No se encontró el pasajero a eliminar.");
            }
        } else {
            JOptionPane.showMessageDialog(null, "Error al actualizar el archivo.");
        }

    } catch (IOException e) {
        e.printStackTrace();
        JOptionPane.showMessageDialog(null, "Error al eliminar el pasajero.");
    }
}
```

- Este método elimina la fila seleccionada de la tabla de datos.

- En varias ventanas existe el método limpiar, este método hace que cada vez que guardemos o utilicemos los componentes de la ventana, limpie o borre lo que el usuario haya escrito en las áreas de texto, a continuación, se muestre un método de limpiar():

```
private void limpiarCampos() {
        modelo.getVista().txtDpi.setText("");
        modelo.getVista().txtTelefono.setText("");
        modelo.getVista().txtNombre.setText("");

        modelo.getVista().txtFechaDeViaje.setText("");
        modelo.getVista().tblPasajeros.clearSelection();
    }
```

- Cuando se elimina un dato, lo que se hace es crear un archivo temporal para guardar de nuevo los datos y se elimina el archivo original para luego renombrar el archivo temporal a el archivo original. Esta función se encuentra en las áreas donde se eliminan datos o se editan los datos

```
private void eliminarPasajero() {
    String dpi = modelo.getVista().txtDpi.getText().trim();
    if (dpi.isEmpty()) {
       JOptionPane.showMessageDialog(null, "Ingrese un DPI para eliminar.");
       return;
    }

    File temp = new File("temporal.txt");

    try (
       BufferedReader br = new BufferedReader(new FileReader(archivoPasajeros));
       BufferedWriter bw = new BufferedWriter(new FileWriter(temp))
    ) {
       String linea;
       boolean eliminado = false;

       while ((linea = br.readLine()) != null) {
          String[] datos = linea.split("\\|");
          if (!datos[0].trim().equals(dpi)) {
             bw.write(linea);
             bw.newLine();
          } else {
```

```
                eliminado = true;
            }
        }

        if (eliminado) {
            archivoPasajeros.delete();
            temp.renameTo(archivoPasajeros);
            JOptionPane.showMessageDialog(null, "Pasajero eliminado correctamente.");
            cargarTabla();
            limpiarCampos();
        } else {
            JOptionPane.showMessageDialog(null, "No se encontró el pasajero para eliminar.");
        }

    } catch (IOException ex) {
        ex.printStackTrace();
    }
}
```

- En esta parte se ve ese funcionamiento mencionado anteriormente, reemplaza el archivo original por el temporal para realizar una eliminación.
- El siguiente método muestra como se abre un archivo como el manual desde un botón:

```
private void abrirManual() {
        try {
            File 14rchive = new File(modelo.getRutaManual());
            if (14rchive.exists()) {
                if (Desktop.isDesktopSupported()) {
```

```
                Desktop.getDesktop().open(15rchive);
            } else {
                JoptionPane.showMessageDialog(modelo.getVista(),
                        "La apertura automática no es soportada en este sistema.");
            }
        } else {
            JoptionPane.showMessageDialog(modelo.getVista(),
                    "El archivo del manual no se encuentra.");
        }
    } catch (Exception ex) {
        ex.printStackTrace();
        JoptionPane.showMessageDialog(modelo.getVista(),
                "Error al abrir el manual: " + ex.getMessage());
    }
}
```

Este método permite al usuario abrir el archivo PDF del manual de usuario directamente desde la interfaz del sistema. Primero, obtiene la ruta del archivo desde el modelo. Luego, verifica si el archivo existe en la ubicación esperada. Si existe y el sistema operativo permite abrir archivos con la aplicación por defecto, el archivo se abre automáticamente. En caso de que el sistema no soporte esta función o que el archivo no se encuentre, se muestra un mensaje de advertencia al usuario. Si ocurre un error inesperado durante el proceso, también se muestra un mensaje con los detalles del error.

- Tambien esta el método para elegir la ruta de descarga del manual:

```
private void descargarManual() {
      try {
         JFileChooser fileChooser = new JFileChooser();
         fileChooser.setSelectedFile(new File("manual.pdf"));
         int opcion = fileChooser.showSaveDialog(modelo.getVista());

         if (opcion == JFileChooser.APPROVE_OPTION) {
```

```
            File destino = fileChooser.getSelectedFile();
            Path origen = Paths.get(modelo.getRutaManual());
            Files.copy(origen, destino.toPath(),
StandardCopyOption.REPLACE_EXISTING);
            JOptionPane.showMessageDialog(modelo.getVista(), "Manual guardado con
éxito.");
          }
      } catch (Exception ex) {
          ex.printStackTrace();
          JOptionPane.showMessageDialog(modelo.getVista(),
                "Error al guardar el manual: " + ex.getMessage());
      }
  }
```

- Este método permite al usuario guardar una copia local del manual en formato PDF en la ubicación de su preferencia. Al ejecutarse, se abre un cuadro de diálogo para seleccionar la carpeta de destino y el nombre del archivo. Si el usuario confirma la operación, el archivo original del manual (obtenido desde el modelo) se copia a la ubicación seleccionada, sobrescribiendo cualquier archivo existente con el mismo nombre. Al finalizar con éxito, se muestra un mensaje de confirmación. En caso de error, se notifica al usuario mostrando los detalles del problema ocurrido.

Se trabajaron varias verificaciones para hacer el código mas robusto y no haya tanto margen de error a la hora de ejecutarlo finalmente o usarlo en un ámbito real como sistema de gestión de autobuses. Para esto se utilizaron Try y Catch, esto permite verificar áreas del código y que no se cierre si existe un error.

- Este método evalúa si una hora determinada (en formato HH:mm) ya ha pasado con respecto a la hora actual del sistema. Primero, convierte la hora recibida como cadena en un objeto Date. Luego, la coloca dentro de un objeto Calendar configurando el mismo día, mes y año que la hora actual. Finalmente, compara ambas horas y retorna true si la hora proporcionada ya pasó, o false si aún no ha ocurrido. Si la cadena no se puede interpretar correctamente como una hora válida, se devuelve false como valor por defecto.

```
private boolean horaPasada(String horaT) {
      SimpleDateFormat sdf = new SimpleDateFormat("HH:mm");
      sdf.setLenient(false);
      try {
         Date horaRuta = sdf.parse(horaT);
         Calendar horaActual = Calendar.getInstance();
         Calendar horaRutaCalendar = Calendar.getInstance();
         horaRutaCalendar.setTime(horaRuta);

         horaRutaCalendar.set(Calendar.YEAR, horaActual.get(Calendar.YEAR));
         horaRutaCalendar.set(Calendar.MONTH, horaActual.get(Calendar.MONTH));
         horaRutaCalendar.set(Calendar.DAY_OF_MONTH,
horaActual.get(Calendar.DAY_OF_MONTH));

         return horaRutaCalendar.before(horaActual);
      } catch (ParseException ex) {
         return false;
      }
   }
```

- El método de buscar es uno de los mas importantes en el funcionamiento de varias ventanas, este ayuda a saber datos importantes de un dato especifico por ejemplo una ruta, busca por código y muestra todos los datos de esa ruta como el siguiente método:

```
private void buscarRuta() {
      String id = modelo.getVista().txtIdRuta.getText().trim();

      try (BufferedReader reader = new BufferedReader(new FileReader(archivo))) {
         String linea;
         while ((linea = reader.readLine()) != null) {
            String[] datos = linea.split("\\|");
            if (datos.length >= 7 && datos[0].trim().equalsIgnoreCase(id)) {
               modelo.getVista().txtOrigen.setText(datos[1].trim());
               modelo.getVista().txtDestino.setText(datos[2].trim());
               modelo.getVista().txtHorario.setText(datos[3].trim());
               modelo.getVista().txtPrecio.setText(datos[4].trim());
               modelo.getVista().cbRutas.setSelectedItem(datos[5].trim());
               modelo.getVista().cbFrecuencia.setSelectedItem(datos[6].trim());
               return;
            }
         }
         JOptionPane.showMessageDialog(null, "Ruta no encontrada");
      } catch (IOException ex) {
```

```
            JOptionPane.showMessageDialog(null, "Error al buscar la ruta");
        }
    }
```

Este método muestra los datos en los componentes específicos, siempre utilizando el try y catch para el manejo de errores.

- Por ultimo se encuentran las librerías que son parte fundamental de muchas áreas del programa y mas para la exportación de datos específicamente de un archivo de texto a un Excel, a continuación se explica cada librería:

Librerías Principales

Absolute Layout - AbsoluteLayout.java

- Permite posicionamiento absoluto de componentes en interfaces gráficas Swing
- Útil para diseños de formularios complejos con posicionamiento preciso
- Considerada obsoleta en Java moderno (se recomienda usar Layout Managers)

jcalendar-1.4.jar

- Componente para selección de fechas en interfaces gráficas
- Proporciona un calendario interactivo para selección de fechas
- Alternativa a JDatePicker para aplicaciones Swing
  Suite Apache POI (para manipulación de documentos Office)

poi-5.2.3.jar

- Biblioteca principal para trabajar con formatos Microsoft Office (.xls, .doc)
- Soporte para formatos OLE2 (versiones antiguas de Office)

poi-ooxml-5.2.3.jar

- Extensión para trabajar con formatos Office Open XML (.xlsx, .docx)
- Soporte para formatos modernos de Office (2007+)

poi-excelant-5.2.3.jar

- Integración con Apache Ant para automatización de procesos Excel
- Permite ejecutar macros y fórmulas desde Java

Otras dependencias de POI

- **poi-examples-5.2.3.jar**: Ejemplos de uso
- **poi-javadoc-5.2.3.jar**: Documentación de la API
- **poi-ooxml-full-5.2.3.jar**: Versión completa con todas las dependencias

Dependencias de Soporte

commons-io-2.15.1.jar (Apache Commons IO)

- Utilidades para operaciones de entrada/salida
- Simplifica manejo de archivos, streams y operaciones de sistema

xmlbeans-5.1.1.jar

- Herramientas para trabajar con XML
- Usado por POI para manejar documentos Office basados en XML

commons-collections4-4.4.jar

- Estructuras de datos avanzadas y utilidades para colecciones
- Extiende las capacidades del framework Collections de Java

commons-compress-1.21.jar

- Compresión/descompresión de archivos (ZIP, TAR, etc.)
- Usado por POI para manejar formatos Office que son esencialmente archivos comprimidos

Logging (registro de eventos)

log4j-api-2.17.2.jar y **log4j-core-2.17.2.jar**

- Framework de logging profesional
- Versión 2.17.2 (parcheada contra vulnerabilidades Log4Shell)
- Proporciona registro estructurado de eventos de la aplicación

Cada una de ellas es importante para la exportación a Excel, a continuación los métodos utilizados.

```
public void exportarExcel() {

    String[] columnas = { "Código", "Nombre", "Teléfono", "Licencia", "Tipo", "Fecha Ingreso", "Estado", "Vencimiento", "Placa Bus" };

    SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");



    // Selector de archivo

    JFileChooser fileChooser = new JFileChooser();

    fileChooser.setDialogTitle("Guardar Reporte de Conductores");
```

```
   fileChooser.setSelectedFile(new File("Reporte Conductores (" + sdf.format(new Date()) +
").xlsx"));
   int seleccion = fileChooser.showSaveDialog(null);

   if (seleccion != JFileChooser.APPROVE_OPTION) {
      return; // el usuario canceló
   }

   File archivo = fileChooser.getSelectedFile();

   try (BufferedReader reader = new BufferedReader(new FileReader("conductores.txt"));
        XSSFWorkbook workbook = new XSSFWorkbook()) {

      XSSFSheet sheet = workbook.createSheet("Conductores");
      Row header = sheet.createRow(0);
      for (int i = 0; i < columnas.length; i++) {
         header.createCell(i).setCellValue(columnas[i]);
      }

      String linea;
      int rowI = 1;

      while ((linea = reader.readLine()) != null) {
         if (linea.trim().isEmpty()) continue;

         String[] datos = linea.split("\\s*\\|\\s*", -1);
```

```
    if (datos.length >= 10) {
        String codigo = datos[0].trim();
        String nombre = datos[1].trim();
        String telefono = datos[3].trim();
        String licencia = datos[5].trim();
        String tipo = datos[6].trim();
        String vencimiento = datos[8].trim();
        String estado = datos[7].trim();
        String fechaIngreso = datos[9].trim();
        String placaBus = obtenerPlacaBusAsignado(codigo);

        Row row = sheet.createRow(rowI++);
        row.createCell(0).setCellValue(codigo);
        row.createCell(1).setCellValue(nombre);
        row.createCell(2).setCellValue(telefono);
        row.createCell(3).setCellValue(licencia);
        row.createCell(4).setCellValue(tipo);
        row.createCell(5).setCellValue(fechaIngreso);
        row.createCell(6).setCellValue(estado);
        row.createCell(7).setCellValue(vencimiento);
        row.createCell(8).setCellValue(placaBus);
    }
}

for (int i = 0; i < columnas.length; i++) {
    sheet.autoSizeColumn(i);
```

```
        }


        try (FileOutputStream out = new FileOutputStream(archivo)) {

            workbook.write(out);

            String mensaje = "Reporte generado con " + (rowI - 1) + " registros";

            modelo.getVista().lblRutaArchivo.setText(mensaje + " en: " +
archivo.getAbsolutePath());

            JOptionPane.showMessageDialog(null, mensaje);

        }


    } catch (IOException e) {

        e.printStackTrace();

        JOptionPane.showMessageDialog(null, "Error al exportar a Excel: " + e.getMessage());

    }

}
```

- Este método permite generar un reporte en formato Excel (.xlsx) con la información de los conductores registrados en el sistema. Utiliza la biblioteca Apache POI para crear y escribir el archivo Excel.

  Al ejecutarse, se abre una ventana de diálogo para que el usuario seleccione la ubicación y el nombre del archivo a guardar. Luego, lee los datos desde el archivo conductores.txt, interpreta y separa cada línea utilizando el carácter | como delimitador, y crea una hoja de cálculo con columnas como Código, Nombre, Teléfono, Licencia, Tipo de licencia, Fecha de ingreso, Estado, Fecha de vencimiento de licencia y Placa del bus asignado (obtenida mediante un método auxiliar).

  Cada registro válido es agregado como una nueva fila en el archivo. Después de escribir todos los registros, ajusta automáticamente el ancho de las columnas para que se adapten al contenido y guarda el archivo en la ubicación elegida por el usuario. Finalmente, muestra un mensaje indicando cuántos registros fueron exportados y actualiza una etiqueta en la vista con la ruta del archivo generado. Si ocurre un error, se muestra un mensaje con los detalles del problema.

6. Consideraciones Finales

- El proyecto no utiliza base de datos, sino archivos .txt.
- Todos los archivos deben estar en la misma carpeta que el .jar para su funcionamiento.
- Se recomienda tener Java instalado para ejecutar el programa.
- Se recomienda leer el manual de usuario para instalar y utilizar de manera eficiente el sistema