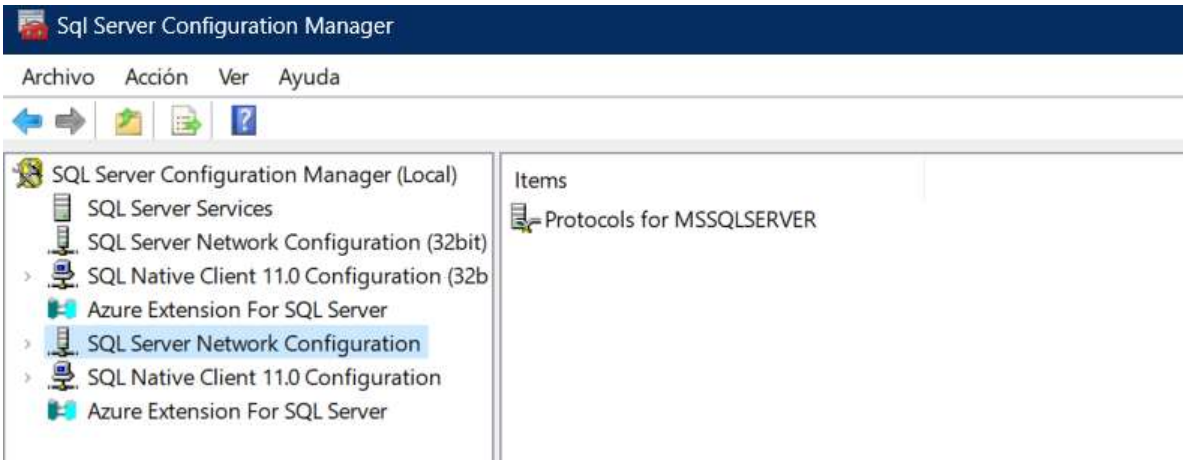
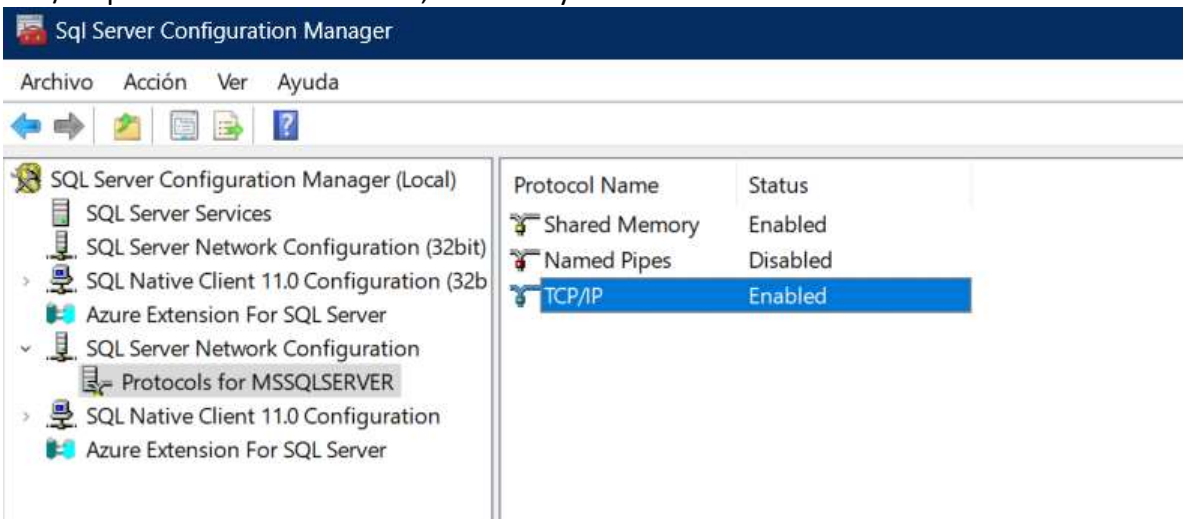
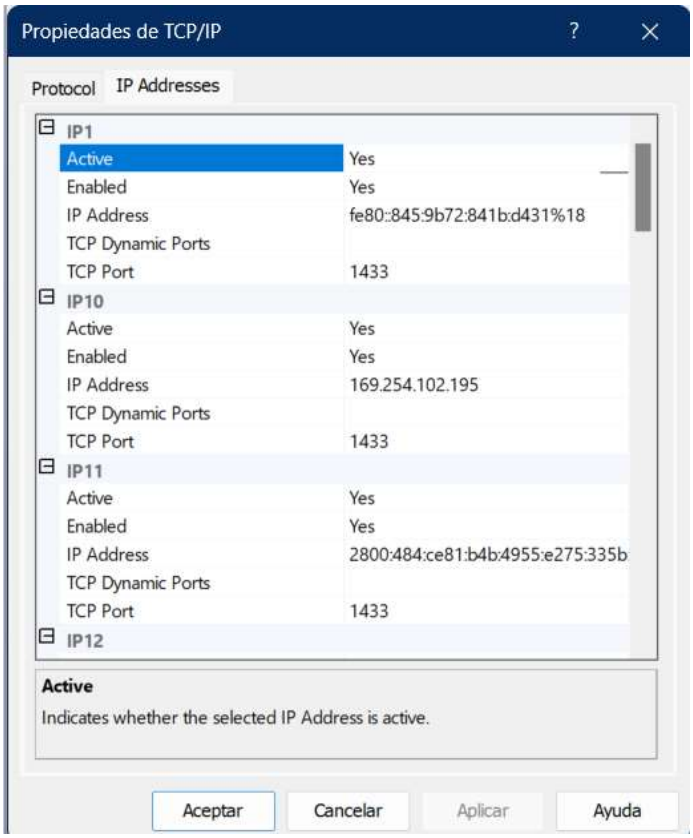
Para la configuración de nuestra conexión al SQL server iremos a Manager de SQL server y realizaremos unas configuraciones:



Tendremos que buscar el índice de SQL Server Network Configuration, en el apartado del SQL desplegamos el Protocols for MSSQLSERVER, y buscamos en el menú de despliegue el TCP/IP que deberá estar Disable, nosotros ya lo habilitamos



Se configura el puerto de escucha de nuestros server SQL, en este caso será el 1433:



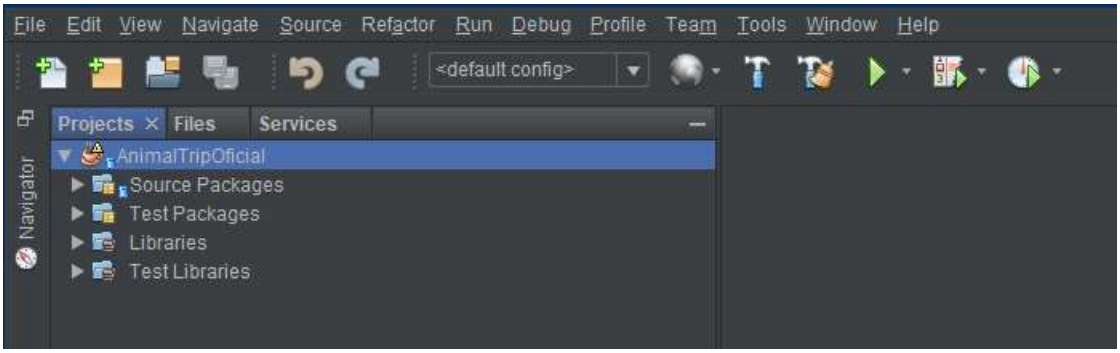
Desplazaremos todas las opciones donde se encuentren en Active No y seleccionamos Yes.

De igual manera debemos asegurarnos de que nuestros servicios estén en ejecución, para este paso nos desplazamos en el menú hasta la opción de: SQL Server Services.

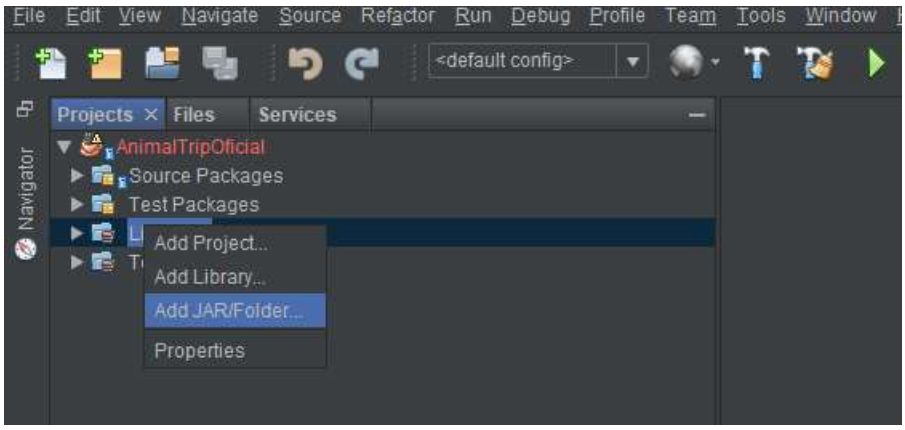
Verificamos que nuestros Services se encuentren ya activos:

Para poder realizar nuestro Proyecto del CRUD a nuestra base de datos de AnimalTrip tendremos que descargar la versión de NetBeans 8.2 la cual no nos presentara problemas de permisos para la conexión. Al igual que su conector desde la página de Microsoft en este caso utilizaremos el “mssql-jdbc-8.2.2.jre8” para la versión de nuestro Server 2022.

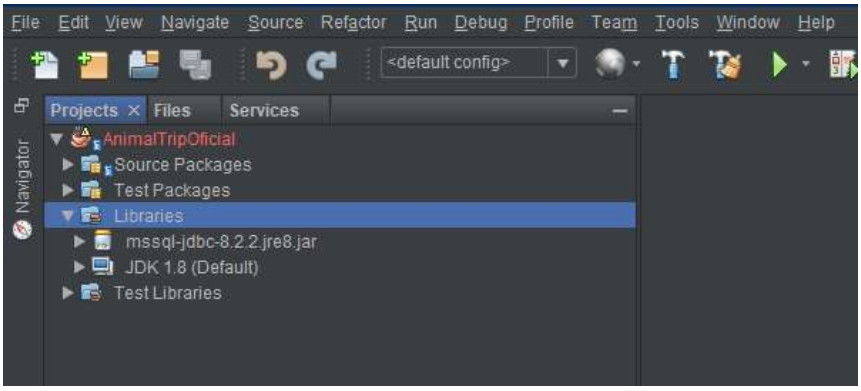
Teniendo ya instalado nuestro entorno de desarrollo, procedemos a crear un nuevo proyecto, en nuestro caso se llamará “AnimalTripOficial”:



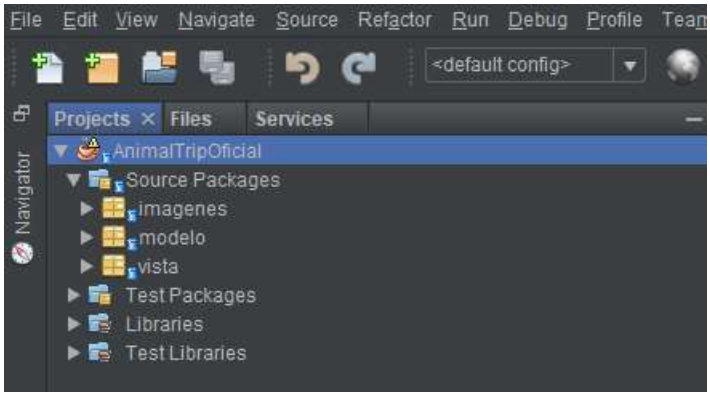
Primero debemos agregar a nuestro proyecto el conector en el apartado de librerías, damos click derecho sobre Libraries y add .jar



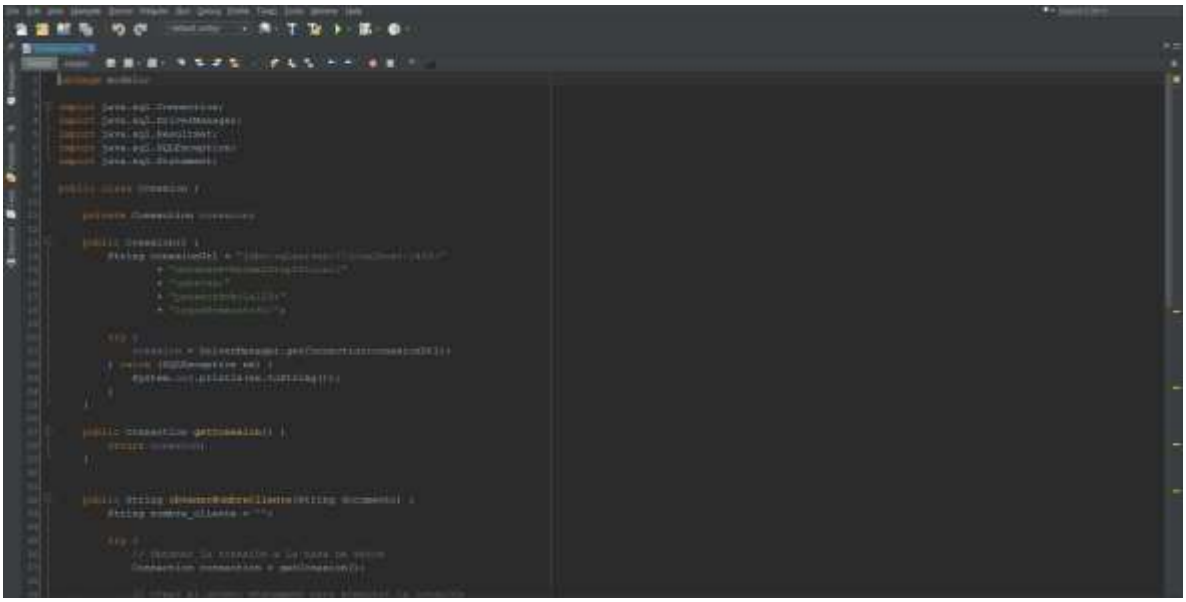
Seleccionamos nuestro archivo que descargamos y lo añadimos a nuestro proyecto:

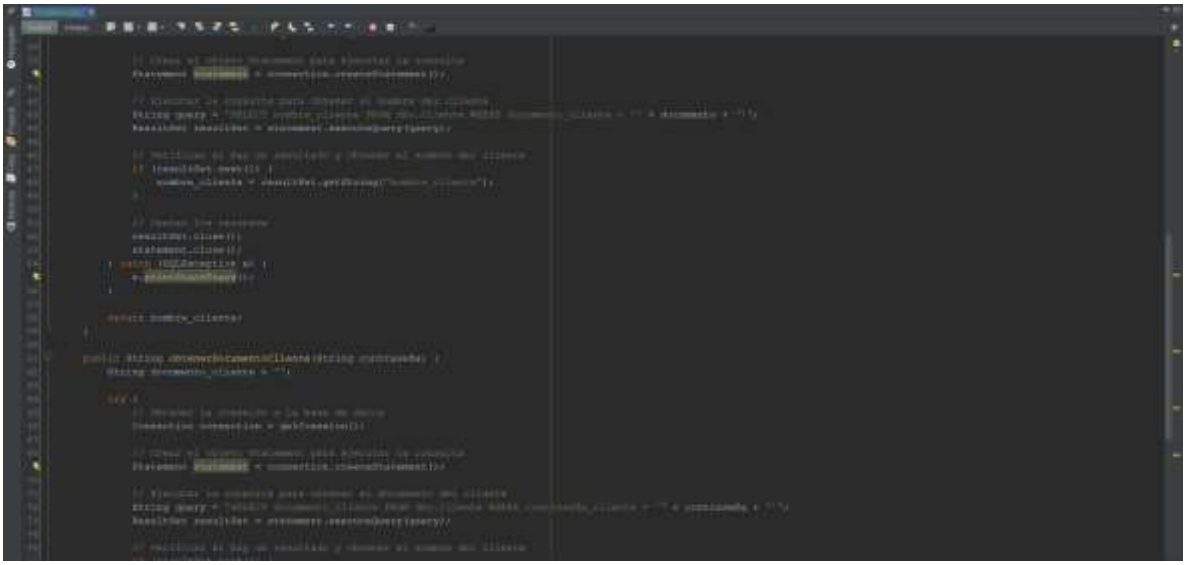


En nuestro proyecto vamos a desarrollar un Sistema de registro de empleados, mascotas, generación de reservas junto a su cancelación y generación de recibos. Junto a los logins de cada cliente y de los empleados.
Para tal motivo vamos a utilizar el modelo MVC como patrón de desarrollo de nuestro CRUD, empezamos por crear un Package llamado Modelo, otro llamado Vista y uno llamado Imágenes.
Este último serán las imágenes que incorporaremos a las vistas de los JFrame que se implementarán para cada cado del sistema.



En el Package Modelo creamos la clase Conexión (); que será la encargada de realizar el enlace de nuestro IDE con el motor BD Server. En esta clase de define la programación del Socket de comunicación, definiendo el usuario de conexión, la contraseña y la BD a la cual queremos acceder.
Además de la información del puerto por el cual se estará comunicando nuestro motor BD Server.





En el packages Vista se crea un JFrame llamado MainMenu.java en el cual se implementará el registro de los clientes de la página de viajes de mascotas, también los botones de inicio de sesión tanto de clientes como de empleados. Estos se verán reflejados en la BD AnimalTripOficial de Server, además de ello se realizará una validación de que el cliente y el empleado exista en la BD.

En el modelo también agregamos una nueva Clase llamada ClientMenu.java para crear el menú del cliente cuando ya se haya logueado en el sistema, validando su información y brindándole las acciones que puede realizar como cliente, como lo son: “Hacer reserva”, “Cancelar reserva”, “Generar recibo”, “Consultar mascotas”, “Registrar mascota” y “Cerrar sesión”.

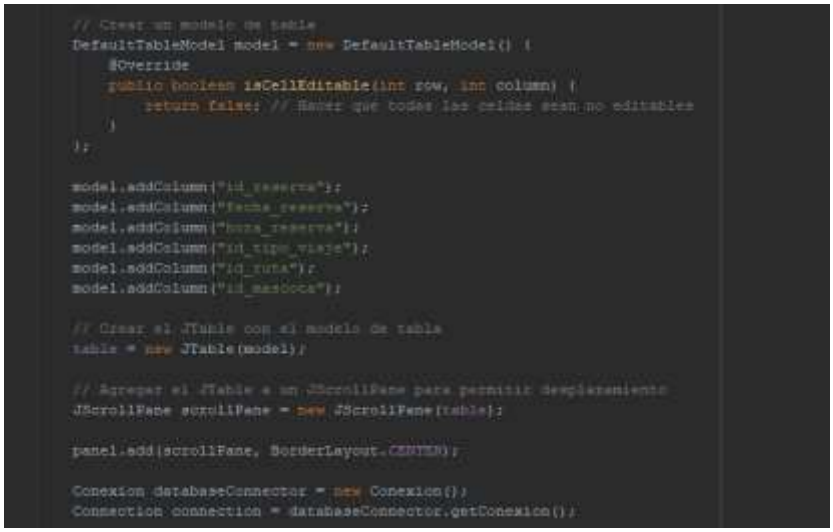
Aquí llamamos el nombre y documento del cliente que se ha logueado, con eso en el JFrame aparece tanto el nombre como el ID o documento del cliente que se logueo.
Se crearon los botones para realizar las acciones que mencionamos anteriormente.

```
lblCliente = new JLabel("Hola, " + nombre_cliente);
lblCliente1 = new JLabel("ID: " + documento_cliente);

lblCliente.setBounds(10, 10, 200, 30);
lblCliente.setFont(lblCliente.getFont().deriveFont(Font.BOLD | Font.ITALIC));
panel.add(lblCliente);

lblCliente1.setBounds(10, 30, 300, 30);
lblCliente1.setFont(lblCliente1.getFont().deriveFont(Font.BOLD | Font.ITALIC));
panel.add(lblCliente1);

// Botones para realizar acciones
btnHacerReserva = new JButton("Hacer Reserva");
btnHacerReserva.setBounds(100, 110, 150, 25);
btnCancelarReserva = new JButton("Cancelar Reserva");
btnCancelarReserva.setBounds(100, 150, 150, 25);
btnGenerarRecibo = new JButton("Generar Recibo");
btnGenerarRecibo.setBounds(100, 190, 150, 25);
btnConsultarMascotas = new JButton("Consultar Mascotas");
btnConsultarMascotas.setBounds(100, 230, 150, 25);
btnRegistrarMascota = new JButton("Registrar Mascota");
btnRegistrarMascota.setBounds(100, 270, 150, 25);
btnCerrarSesion = new JButton("Cerrar Sesión");
btnCerrarSesion.setBounds(100, 310, 150, 25);
```

Aquí se generan las conexiones entre clases para que el funcionamiento de los botones, se conecta el botón de reserva con la clase CreateReservation.java, de igual manera con los otros botones, se van conectando con las clases correspondientes que en un momento se mencionarán.

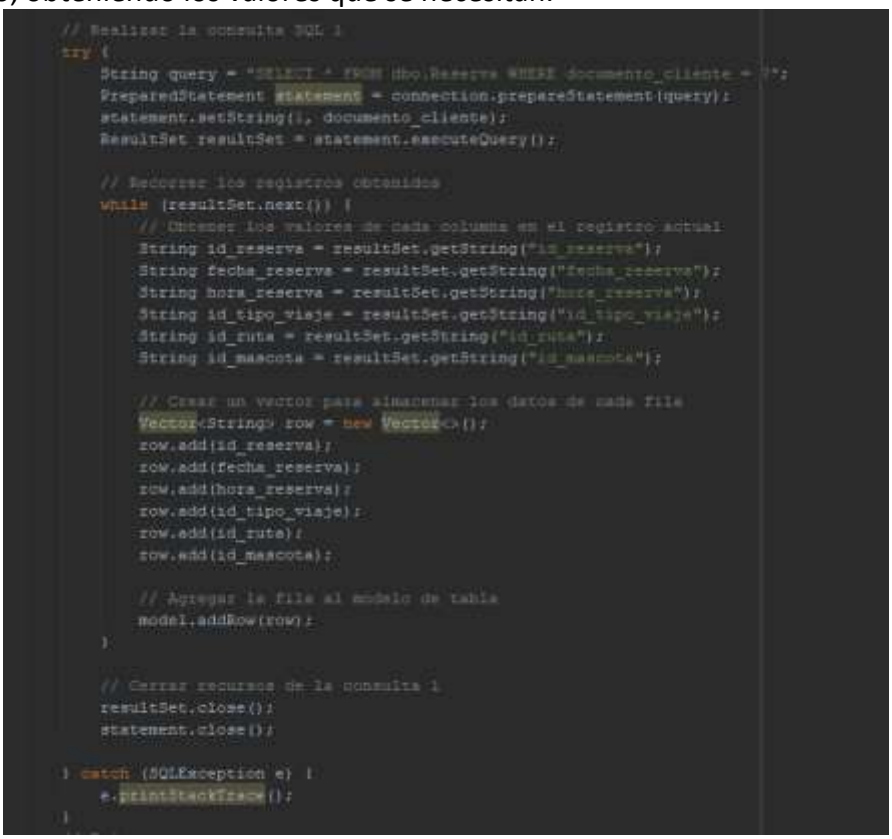
```
btnHacerReserva.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        // Cerrar el JFrame actual
        dispose();
        CreateReservation reservacion = new CreateReservation(nombre_cliente, documento_cliente);
        reservacion.setVisible(true);
    }
});

btnCancelarReserva.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        // Cerrar el JFrame actual
        dispose();
        DeleteReservation cancelarReservacion = new DeleteReservation(nombre_cliente, documento_cliente
        cancelarReservacion.setVisible(true);
    }
});

btnGenerarRecibo.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        // Cerrar el JFrame actual
        dispose();
        GenerarRecibo recibo = new GenerarRecibo(nombre_cliente, documento_cliente);
        recibo.setVisible(true);
    }
});

btnConsultarMascotas.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        // Cerrar el JFrame actual
        dispose();
        ReadMascotas consultarMascotas = new ReadMascotas(nombre_cliente, documento_cliente);
        consultarMascotas.setVisible(true);
    }
});

btnRegistrarMascota.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        // Cerrar el JFrame actual
        dispose();
        NewPet registro_mascota = new NewPet(nombre_cliente, documento_cliente);
        registro_mascota.setVisible(true);

    }
```

En la clase CreateReservation.java, se crea un modelo de una tabla para manejar la creación de reservaciones.



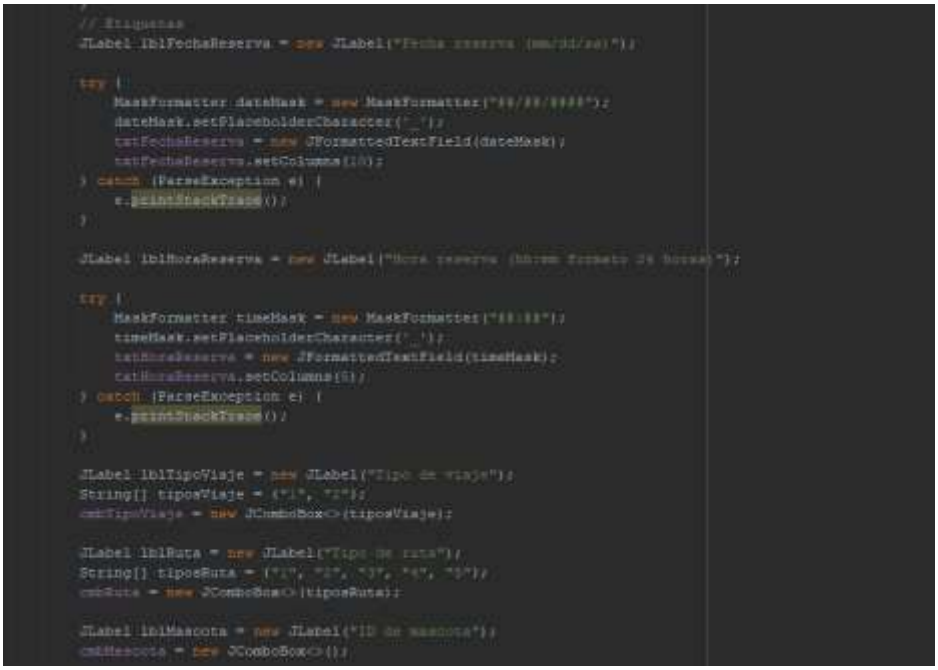
```
// Crear un modelo de tabla
DefaultTableModel model = new DefaultTableModel() {
    @Override
    public boolean isCellEditable(int row, int column) {
        return false; // Hacer que todas las celdas sean no editables
    }
};

model.addColumn("id_reserva");
model.addColumn("fecha_reserva");
model.addColumn("hora_reserva");
model.addColumn("id_tipo_viaje");
model.addColumn("id_ruta");
model.addColumn("id_mascota");

// Crear el JTable con el modelo de tabla
table = new JTable(model);

// Agregar el JTable a un JScrollPane para permitir desplazamiento
JScrollPane scrollPane = new JScrollPane(table);

panel.add(scrollPane, BorderLayout.CENTER);

Conexion databaseConnector = new Conexion();
Connection connection = databaseConnector.getConexion();
```

Aquí se crean cada una de las consultas de SQL que necesitamos, en este caso es la consulta de mostrar los registros de la tabla de reserva donde aparezca el documento de cliente relacionado, obteniendo los valores que se necesitan.



```
// Realizar la consulta SQL 1
try {
    String query = "SELECT * FROM dbo.Reserva WHERE documento_cliente = ?";
    PreparedStatement statement = connection.prepareStatement(query);
    statement.setString(1, documento_cliente);
    ResultSet resultSet = statement.executeQuery();

    // Recorrer los registros obtenidos
    while (resultSet.next()) {
        // Obtener los valores de cada columna en el registro actual
        String id_reserva = resultSet.getString("id_reserva");
        String fecha_reserva = resultSet.getString("fecha_reserva");
        String hora_reserva = resultSet.getString("hora_reserva");
        String id_tipo_viaje = resultSet.getString("id_tipo_viaje");
        String id_ruta = resultSet.getString("id_ruta");
        String id_mascota = resultSet.getString("id_mascota");

        // Crear un vector para almacenar los datos de cada fila
        Vector<String> row = new Vector<>();
        row.add(id_reserva);
        row.add(fecha_reserva);
        row.add(hora_reserva);
        row.add(id_tipo_viaje);
        row.add(id_ruta);
        row.add(id_mascota);

        // Agregar la fila al modelo de tabla
        model.addRow(row);
    }

    // Cerrar recursos de la consulta 1
    resultSet.close();
    statement.close();

} catch (SQLException e) {
    e.printStackTrace();
}
```

Luego, se crean e inicializan los combobox con cada uno de los valores que necesitamos insertar en la base de datos:

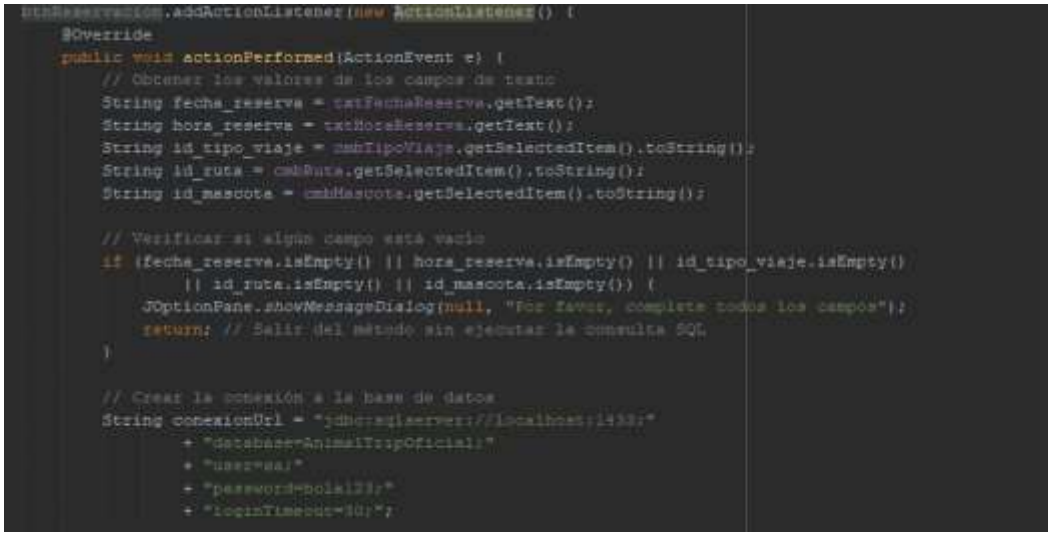


En el segundo query, seleccionamos el id_mascota de la tabla Mascota y se insertarán en el comboBox para que el cliente solo pueda insertar valores que si existan y estén relacionados con él.

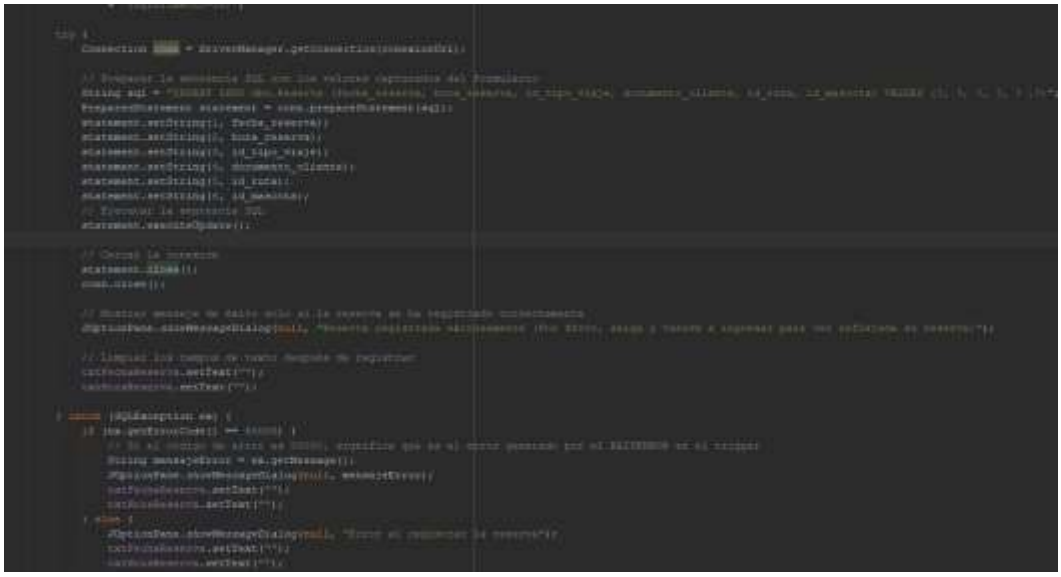
```
// Realizar la consulta SQL
        try {
            String query = "SELECT id_mascota FROM dbo.Mascota WHERE documento_cliente = ?";
            PreparedStatement statement = connection.prepareStatement(query);
            statement.setString(1, documento_cliente);
            ResultSet resultSet = statement.executeQuery();

            // Recorrer los registros obtenidos
            while (resultSet.next()) {
                // Obtener el valor de id_mascota en el registro actual
                String id_mascota = resultSet.getString("id_mascota");

                // Agregar el valor al JComboBox
                cmbMascota.addItem(id_mascota);
            }

            // Cerrar recursos de la consulta
            resultSet.close();
            statement.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }

// Agregar el JComboBox y el JLabel al contenedor adecuado (por ejemplo, un JPanel)
        panel.add(cmbMascota);
        panel.add(lblMascota);
```

```
// Agregar un ActionListener al JComboBox
        cmbMascota.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                // Obtener la selección actual del JComboBox
                String selectedMascota = (String) cmbMascota.getSelectedItem();

                // Actualizar el texto del JLabel con la selección actual
                lblMascota.setText("ID de mascota: " + selectedMascota);
            }
        });
```

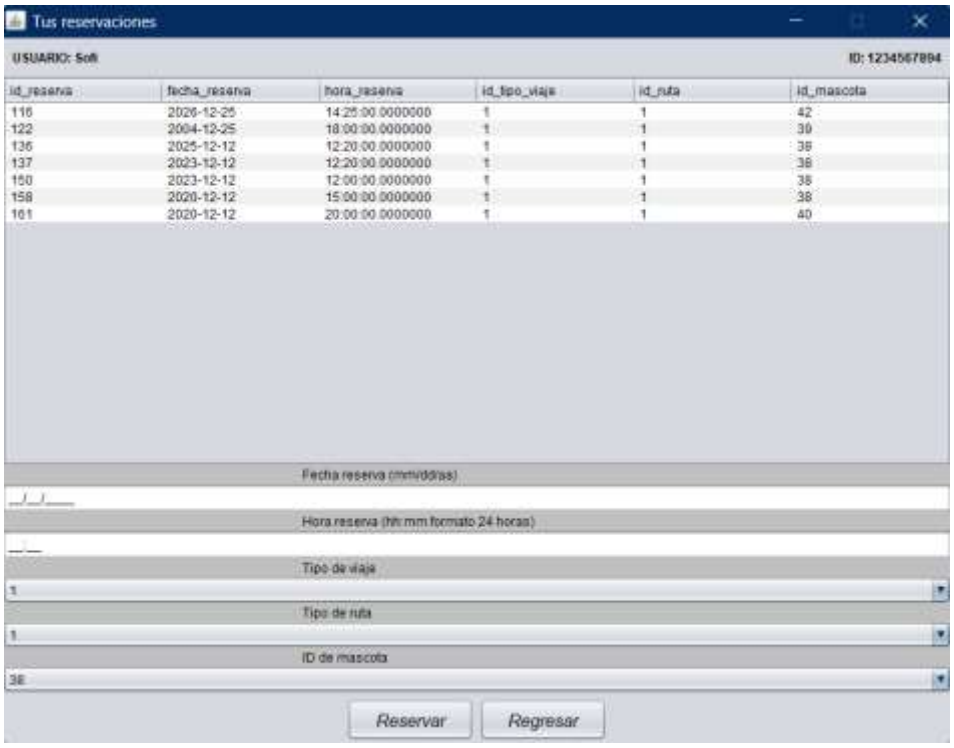
En el siguiente botón de reservación, inicializo los String de cada columna de la tabla y lo convierto a texto, se verifica si algún campo está vacío antes de enviar el formulario y se establece la conexión con la base de datos.



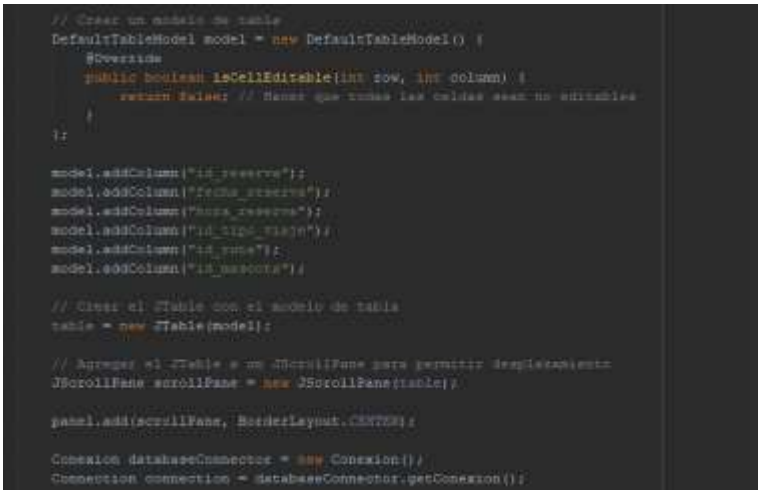
En el siguiente bloque de código se prepara la inserción de los datos que el cliente ingresó a la tabla Reserva, añadiéndole a esto, se implementa el mensaje de error de un trigger que hicimos donde no permite la inserción de una reserva si la mascota tiene menos de 2 años.



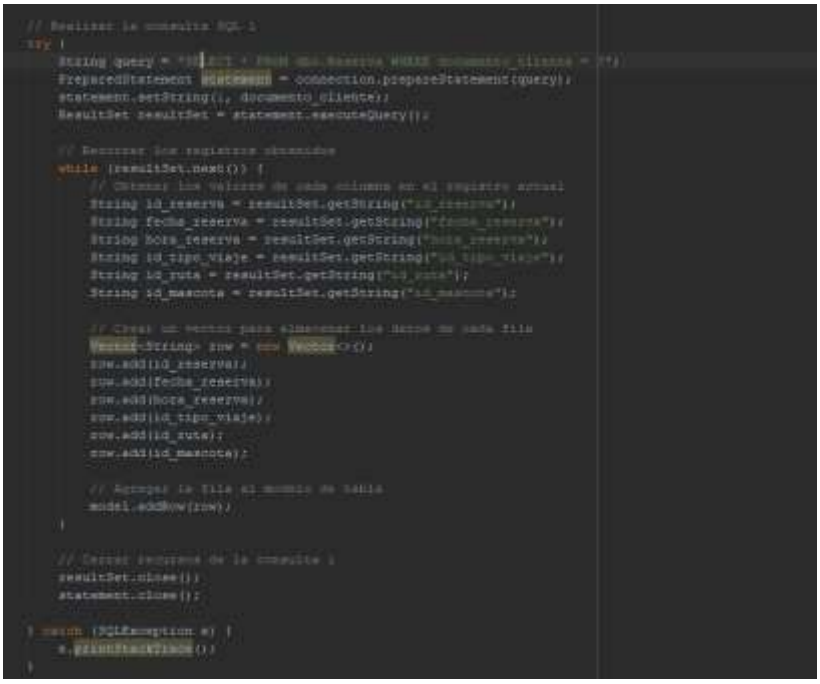
Finalmente, la clase en ejecución se ve así:



En la siguiente clase DeleteReservation.java se implementa toda la lógica necesaria para poder poner en funcionamiento el cancelamiento de una reservación.
En principio, se crea de nuevo el modelo de la tabla que aparecerá al momento de ejecutar la clase, recordemos que estas clases necesitan el nombre y documento del cliente logueado por lo cual los pasaremos como parámetros al inicio.



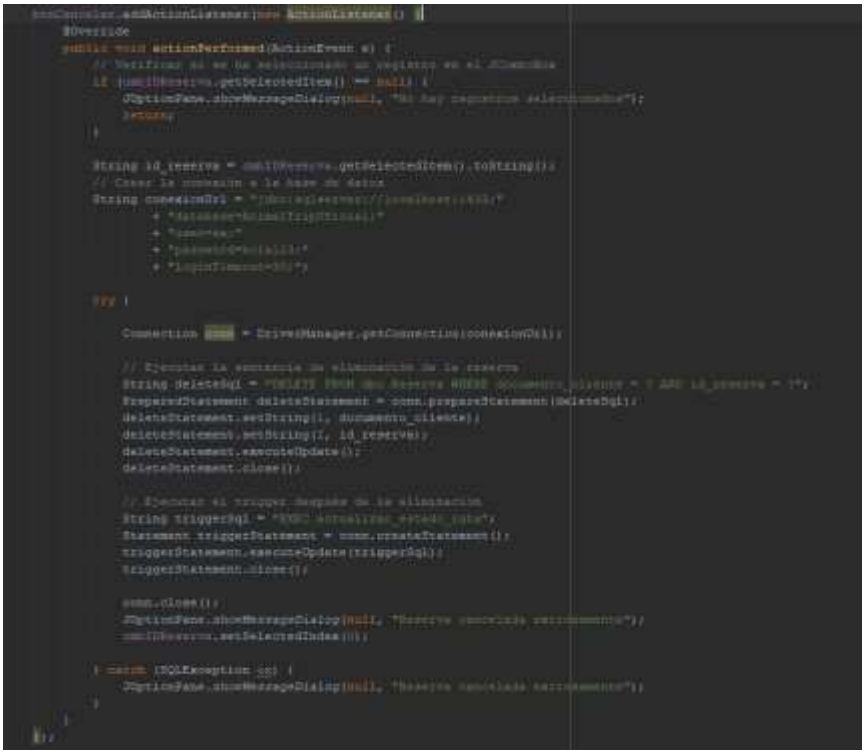
Se tiene de igual manera la consulta para poder ver todos los registros de la tabla de reserva:



También necesitamos hacer nuevamente una consulta para obtener el valor de id_reserva y agregarlo al combobox para que el usuario pueda utilizar las reservas que son de él:

```
// Realizar la consulta SQL
try {
    String query = "SELECT id_reserva FROM dbo.Reserva WHERE documento_cliente = ?";
    PreparedStatement statement = connection.prepareStatement(query);
    statement.setString(1, documento_cliente);
    ResultSet resultSet = statement.executeQuery();

    // Recorrer los registros obtenidos
    while (resultSet.next()) {
        // Obtener el valor de id_mascota en el registro actual
        String id_reserva = resultSet.getString("id_reserva");

        // Agregar el valor al JComboBox
        cmbIDReserva.addItem(id_reserva);
    }

    // Cerrar recursos de la consulta
    resultSet.close();
    statement.close();
} catch (SQLException e) {
    e.printStackTrace();
}
```

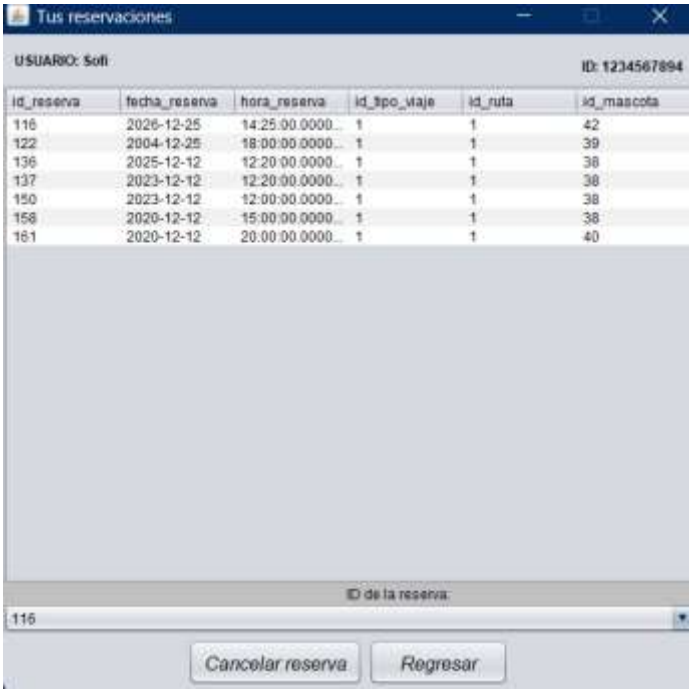
El botón de cancelar reserva tiene la siguiente lógica, se crea la conexión a la base de datos y se implementa la sentencia de la eliminación de la reserva de cliente logueado:



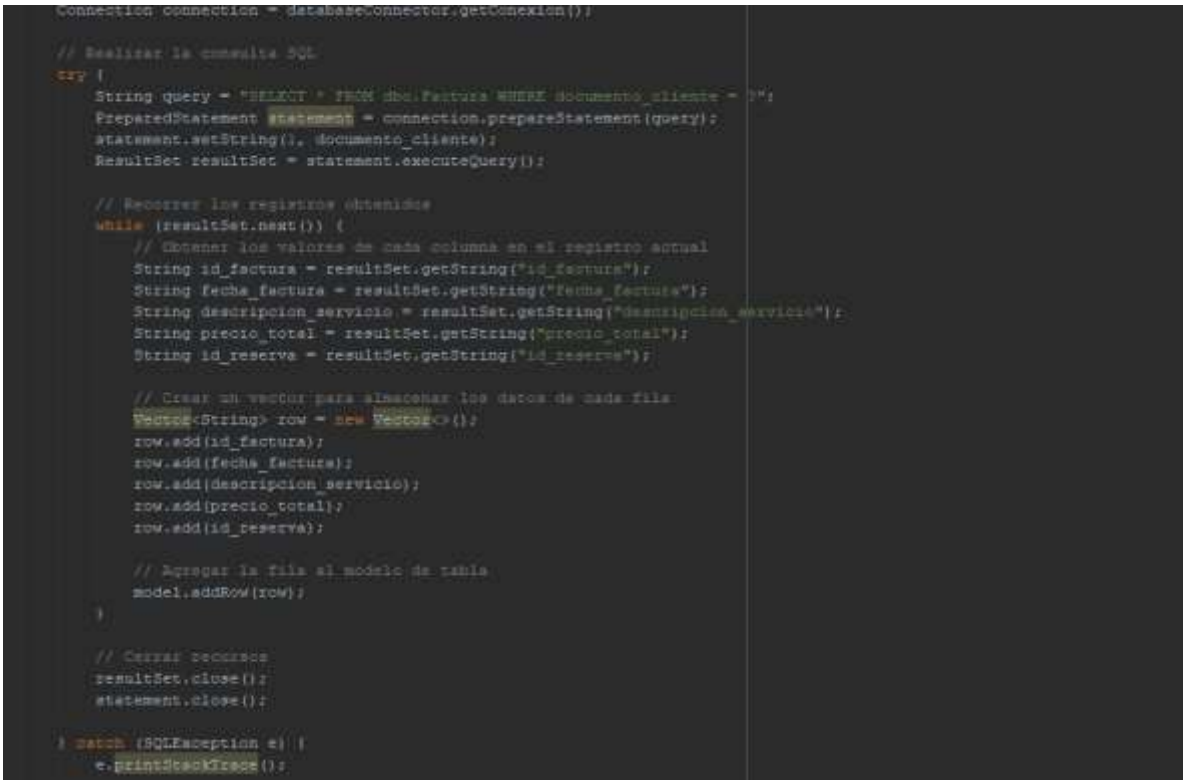
Además de esto, se implementó el trigger llamado actualizar_estado_ruta el cual su funcionamiento se basa en actualizar la columna “estado” de la tabla ruta de “Ocupado” a “Asientos disponibles”:

```
USE [AnimalTripOficial]
GO
/****** Object:  Trigger [dbo].[actualizar_estado_ruta]    Script Date: 01/06/2023 12:51:01 a. m. ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER TRIGGER [dbo].[actualizar_estado_ruta]
ON [dbo].[Reserva]
AFTER DELETE
AS
BEGIN
  UPDATE Ruta
  SET estado = 'Asientos disponibles'
  FROM DELETED d
  WHERE Ruta.id_ruta = d.id_ruta;
END
```

La clase en ejecución se ve así:



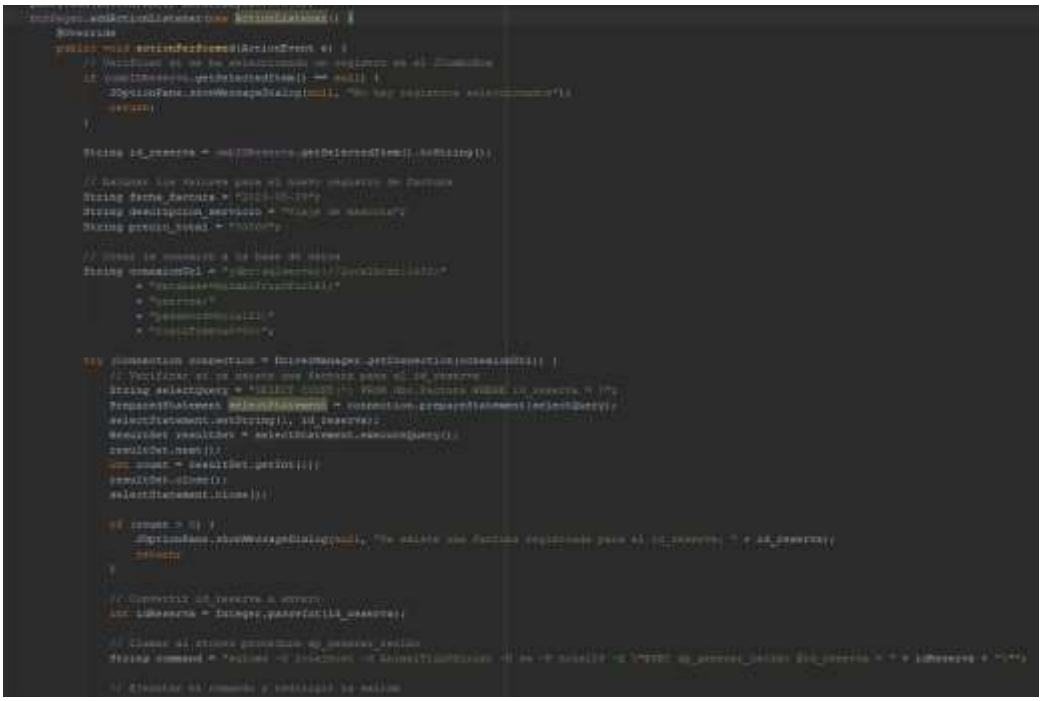
En la clase GenerarRecibo.java se volvió a crear un modelo de tabla para mostrar la información de la tabla factura.



Se pide el id de la reserva para así poder crear la factura con los datos del cliente y añadirle el id de la reserva además de ingresar los valores en el combobox para que el usuario pueda elegir a que reserva quiere generarle una factura:

```
// Realizar la consulta SQL
try {
    String query = "SELECT id_reserva FROM dbo.Reserva WHERE documento_cliente = ?";
    PreparedStatement statement = connection.prepareStatement(query);
    statement.setString(1, documento_cliente);
    ResultSet resultSet = statement.executeQuery();

    // Recorrer los registros obtenidos
    while (resultSet.next()) {
        // Obtener el valor de id_mascota en el registro actual
        String id_reserva = resultSet.getString("id_reserva");

        // Agregar el valor al JComboBox
        cmbIDReserva.addItem(id_reserva);
    }

    // Cerrar recursos de la consulta
    resultSet.close();
    statement.close();
} catch (SQLException e) {
    e.printStackTrace();
}
```

En el botón de pagar se implementa la conexión con la base de datos normalmente y empieza con las verificaciones, primero se verifica si ya existe una factura para la reserva indicada y luego se insertan los valores asignados en el registro.



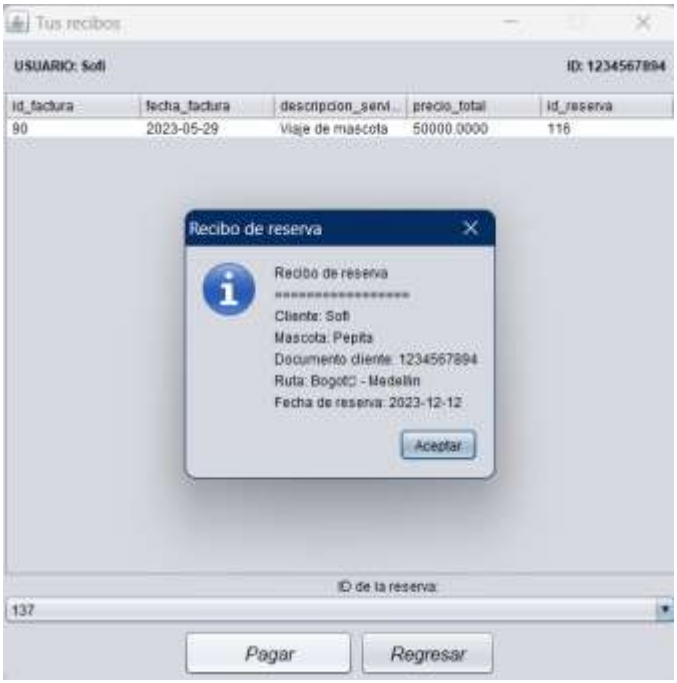
Luego, se implementa un Stored Procedure llamado sp_generar_recibo, el cual se encarga de enviar en un mensaje la factura generada con el id del cliente y demás datos relevantes.

Stored Procedure:

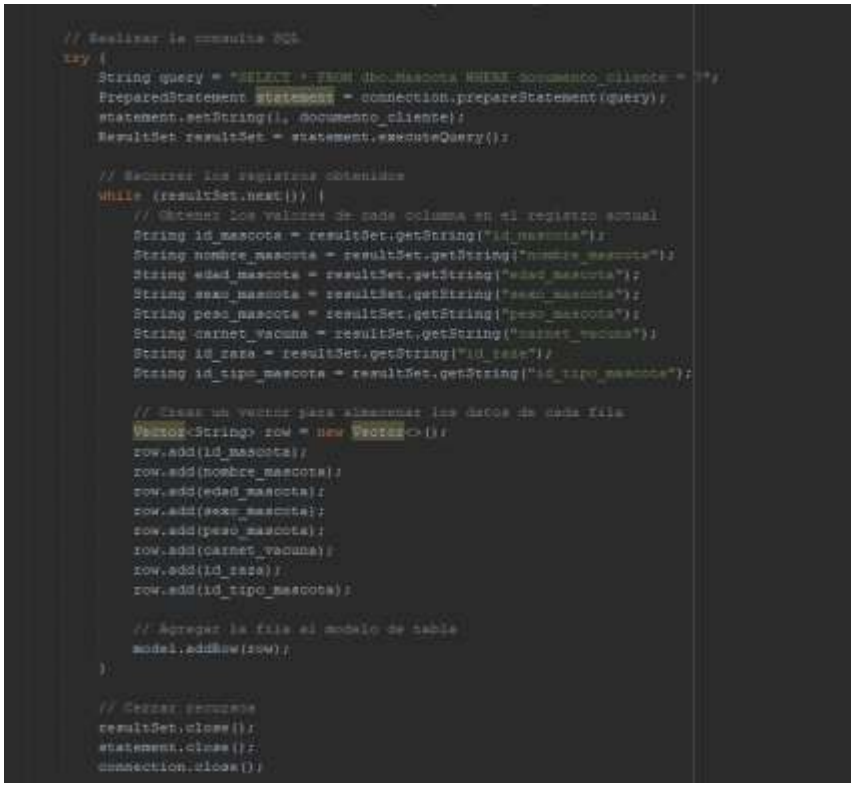
```
USE [AnimalTripOficial]
GO
/****** Object:  StoredProcedure [dbo].[sp_generar_recibo]    Script Date: 01/06/2023 01:03:48 a. m. ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROCEDURE [dbo].[sp_generar_recibo]
    @id_reserva int
AS
BEGIN
    DECLARE @nombre_cliente varchar(25)
    DECLARE @nombre_mascota varchar(25)
    DECLARE @documento_cliente int
    DECLARE @telefono_cliente varchar(10)
    DECLARE @nombre_ruta varchar(25)
    DECLARE @fecha_reserva date
    DECLARE @precio_total int -- Corregido a tipo decimal

    SELECT @nombre_cliente = c.nombre_cliente,
           @documento_cliente = c.documento_cliente,
           @telefono_cliente = c.telefono_cliente,
           @nombre_mascota = m.nombre_mascota,
           @nombre_ruta = ru.nombre_ruta,
           @fecha_reserva = r.fecha_reserva
    FROM Reserva r
    INNER JOIN Cliente c ON c.documento_cliente = r.documento_cliente
    LEFT JOIN Mascota m ON m.id_mascota = r.id_mascota
    INNER JOIN Ruta ru ON ru.id_ruta = r.id_ruta
    WHERE r.id_reserva = @id_reserva;

    PRINT 'Recibo de reserva'
    PRINT '-----------------'
    PRINT 'Cliente: ' + @nombre_cliente
    PRINT 'Mascota: ' + @nombre_mascota
    PRINT 'Documento cliente: ' + CONVERT(varchar(10), @documento_cliente)
    PRINT 'Ruta: ' + @nombre_ruta
    PRINT 'Fecha de reserva: ' + CONVERT(varchar(10), @fecha_reserva)
END
```

La clase en ejecución se ve así:

Funcionamiento del Stored Procedure al momento de pagar una reserva:



Para la clase ReadMascotas.java que es la encargada de mostrar simplemente las mascotas que tiene el cliente simplemente realizamos la consulta que necesitamos, en este caso seleccionar todos los registros de la tabla mascota donde el documento del cliente sea el que está logueado y luego se obtienen los valores y se agregan los datos:

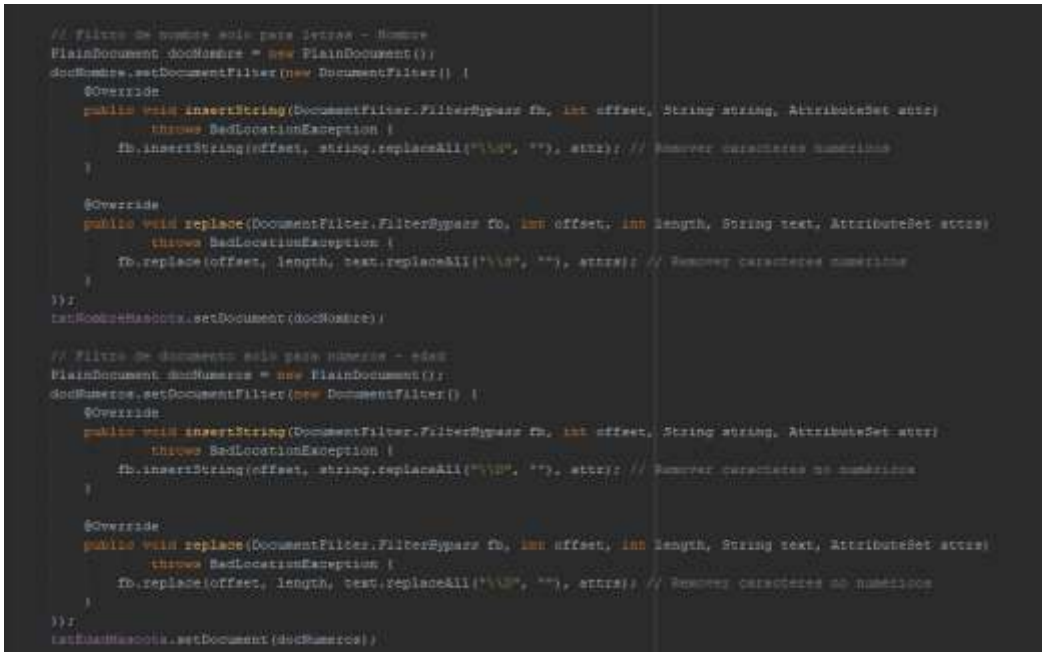


El funcionamiento de la clase ReadMascotas es el siguiente:

Y como última clase en el apartado de cliente, tenemos el registro de las mascotas (NewPet.java), el cual cada cliente tiene permitido esto, solo registrar la mascota para el usuario logueado.

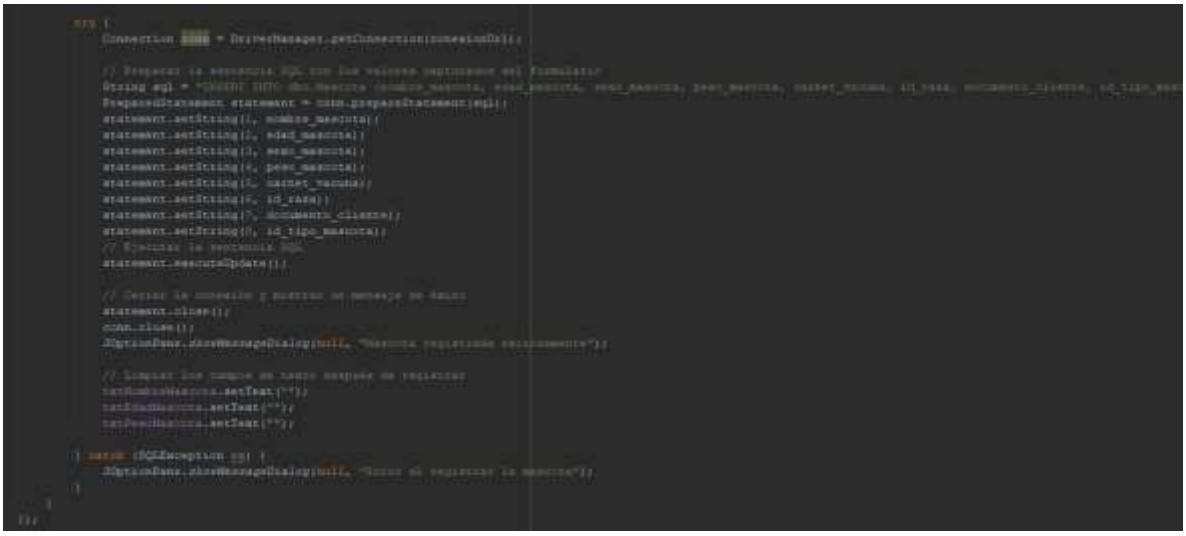
Se crearon los JLabel necesarios para ingresar los datos a los registros de la base de datos de mascota:

Además, tanto en esta clase como en las anteriores y posteriores, se generaron si era necesario filtros de caracteres para no permitir el ingreso de ciertos caracteres no permitidos, en este caso filtros para el nombre, la edad, y el peso.



En el botón de registrar se obtuvieron los valores de los campos de texto, valida que ninguno puede estar vacío al enviar el formulario, se crea la conexión a la base de datos y el query de insert para registrar os datos correctamente.

```
btnRegistrar.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        // Obtener los valores de los campos de texto
        String nombre_mascota = txtNombreMascota.getText();
        String edad_mascota = txtEdadMascota.getText();
        String sexo_mascota = cmbSexoMascota.getSelectedItem().toString();
        String peso_mascota = txtPesoMascota.getText();
        String carnet_vacuna = cmbCarnetVacuna.getSelectedItem().toString();
        String id_raza = cmbRaza.getSelectedItem().toString();
        String id_tipo_mascota = cmbTipoMascota.getSelectedItem().toString();

        // Verificar si algun campo esta vacio
        if (nombre_mascota.isEmpty() || edad_mascota.isEmpty() || sexo_mascota.isEmpty()
                || peso_mascota.isEmpty() || carnet_vacuna.isEmpty() || id_raza.isEmpty()
                || id_tipo_mascota.isEmpty()) {
            JOptionPane.showMessageDialog(null, "Por favor, complete todos los campos");
            return; // Salir del método sin ejecutar la consulta SQL
        }

        // Crear la conexion a la base de datos
        String conexionUrl = "jdbc:sqlserver://localhost:1433;"
                + "database=AnimalTripOficial;"
                + "user=sa;"
                + "password=bola123;"
                + "loginTimeout=30;";
```



```
try {
    Connection conn = DriverManager.getConnection(conexionUrl);

    // Preparar la sentencia SQL con los valores capturados del formulario
    String sql = "INSERT INTO dbo.Mascota (nombre_mascota, edad_mascota, sexo_mascota, peso_mascota, carnet_vacuna, id_raza, documento_cliente, id_tipo_mas
    PreparedStatement statement = conn.prepareStatement(sql);
    statement.setString(1, nombre_mascota);
    statement.setString(2, edad_mascota);
    statement.setString(3, sexo_mascota);
    statement.setString(4, peso_mascota);
    statement.setString(5, carnet_vacuna);
    statement.setString(6, id_raza);
    statement.setString(7, documento_cliente);
    statement.setString(8, id_tipo_mascota);
    // Ejecutar la sentencia SQL
    statement.executeUpdate();

    // Cerrar la conexión y mostrar un mensaje de éxito
    statement.close();
    conn.close();
    JOptionPane.showMessageDialog(null, "Mascota registrada exitosamente");

    // Limpiar los campos de texto después de registrar
    txtNombreMascota.setText("");
    txtEdadMascota.setText("");
    txtPesoMascota.setText("");

} catch (SQLException ex) {
    JOptionPane.showMessageDialog(null, "Error al registrar la mascota");
}
    }
});
```

Finalmente, para culminar el apartado de cliente, así se ve la clase de registro de mascotas en ejecución: