SYSTEM TASKFLOW

MANUAL TÉCNICO

Manual Técnico – TASKFLOW

1. Información General

Nombre del sistema: TASKFLOW

Tipo de sistema: Sistema de atención de tickets

Lenguaje de programación: Java

Versión JDK: Java SE 22

IDE: Apache NetBeans IDE 22

Framework de interfaz gráfica: JavaFX con vistas en FXML

Base de datos: PostgreSQL

2. Arquitectura del Sistema

2.1. Estructura General

El sistema está dividido en las siguientes capas:

- **Vista (FXML):** Diseñada con Scene Builder o manualmente. Define la interfaz gráfica del usuario.
- **Controlador:** Maneja eventos de la UI, validaciones y control del flujo.
- **Modelo:** Contiene las clases de dominio que representan los datos y se comunican con la base de datos.
- **Persistencia:** Utiliza JDBC para la conexión y operaciones con PostgreSQL.
- **Utilitarios:** Funciones de utilidad para operaciones comunes (fechas, alertas, validaciones, etc).

3. Requisitos del Sistema

3.1. Software

- Java Development Kit (JDK) 22
- Apache NetBeans IDE 22
- PostgreSQL 14 o superior
- Scene Builder (opcional para editar FXML)
- Librerías JDBC de PostgreSQL

3.2. Hardware

- Mínimo 4 GB de RAM
- Procesador Intel i3 o superior
- 500 MB de espacio disponible

4. Configuración del Entorno

4.1. Instalación de JDK y configuración de variables de entorno

1. **Descargar JDK 22**

   Desde el sitio oficial de Oracle:

   https://www.oracle.com/java/technologies/javase-downloads.html

2. **Instalar JDK**

   Ejecuta el instalador y sigue los pasos por defecto.

4.2. Instalación de Apache NetBeans IDE y Scene Builder

1. **Descargar Apache NetBeans 22**

   https://netbeans.apache.org/download/nb22/

2. **Instalar NetBeans**

   Ejecuta el instalador y asegúrate de que detecta el JDK instalado.

3. **Descargar Gluon Scene Builder**

   https://gluonhq.com/products/scene-builder/

4. **Asociar Scene Builder con NetBeans**

   - Abre NetBeans
   - Ir a *Tools > Options > Java > JavaFX*
   - En *Scene Builder Home*, selecciona la ruta donde instalaste Scene Builder.

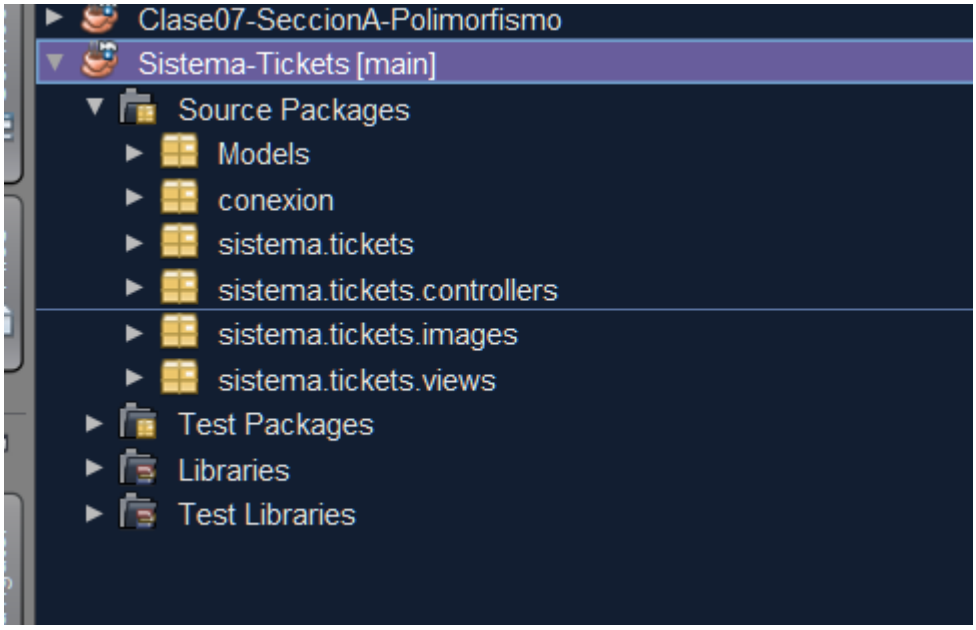
Estructura del Proyecto

El proyecto está organizado en una estructura de carpetas que facilita el mantenimiento y la comprensión del código. A continuación, se describen las principales carpetas del sistema:

- **Modelos**: Contiene las clases que representan las entidades del sistema. Estas clases se utilizan para mapear los datos que se almacenan y se manipulan en la aplicación.
- **Conexión (Base de Datos)**: Incluye los archivos necesarios para establecer la conexión con la base de datos. Aquí se configuran los parámetros de conexión y, en algunos casos, se definen métodos para interactuar directamente con la base de datos.
- **Controladores**: Aquí se encuentran las clases que manejan la lógica de negocio de la aplicación. Los controladores reciben las solicitudes del usuario, procesan la información y devuelven las respuestas adecuadas, ya sea en forma de vistas o de datos.
- **Views:** Contiene las vistas de la aplicación, es decir, las interfaces gráficas que interactúan con el usuario. Estas vistas están desarrolladas utilizando JavaFX y diseñadas con FXML, lo que permite separar la lógica de presentación del código de control. Cada vista suele tener un archivo .fxml que define la estructura visual (layouts, botones, campos, etc.), y su respectivo controlador en Java que gestiona los eventos y la lógica de interacción.

- **Images**: Carpeta destinada a almacenar imágenes utilizadas por la aplicación, tales como logotipos, íconos o imágenes cargadas por el usuario.

Cada carpeta cumple una función específica y se mantiene organizada para asegurar la escalabilidad y el orden del proyecto a lo largo de su desarrollo.



TIPOS DE CLASES IMPLEMENTADAS

Ejemplo de Clase en la Carpeta "Modelos"

En esta sección se presenta un ejemplo de una clase ubicada dentro de la carpeta Modelos. Estas clases representan entidades del sistema y están diseñadas para reflejar la estructura de las tablas en la base de datos. Cada atributo de la clase corresponde a una columna de la tabla, y permiten trabajar con los datos de forma orientada a objetos dentro del programa.

Estas clases suelen incluir:

- Atributos privados que almacenan la información.
- Constructores para inicializar los objetos.
- Métodos "getter" y "setter" para acceder y modificar los valores de cada atributo.

A continuación, se muestra un ejemplo de este tipo de clase:

```
14
15  public class Ticket implements Serializable {
16
17      private static final long serialVersionUID = 1L;
18      private int id;
19      private String titulo;
20      private String descripcion;
21      private Timestamp fechaCreacion;
22      private Integer id_tecnico_asignado;
23      private Integer id_usuario;
24      private Integer id_departamento;
25      private Integer id_prioridad;
26      private Integer id_estado;
27
28      private String tecnico;
29      private String estado;
30      private String prioridad;
31      private String departamento;
32
33      public Ticket() {
34      }
35
```

Como se puede observar, esta clase facilita la gestión de datos en el sistema, permitiendo que otras partes del programa (como los controladores) puedan acceder a la información de manera clara, estructurada y segura.

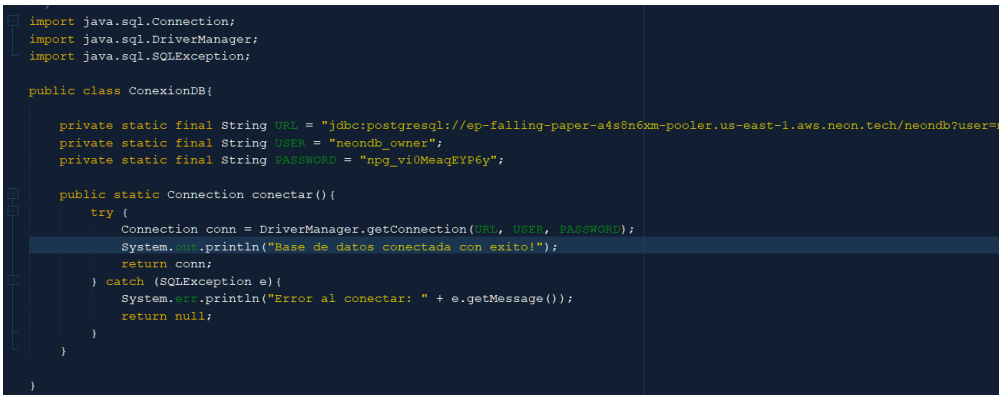
Ejemplo de Clase en la Carpeta "Conexión"

La carpeta Conexión contiene clases encargadas de gestionar la comunicación entre la aplicación y la base de datos. A continuación, se muestra un ejemplo típico: la clase ConexionDB.

Esta clase se encarga de establecer la conexión con una base de datos PostgreSQL utilizando JDBC. Sus componentes principales son:

- **URL de conexión**: especifica la dirección del servidor de base de datos, el nombre de la base (neondb), el usuario, la contraseña y el modo de seguridad (SSL).
- **Método conectar()**: es un método estático que intenta establecer una conexión con la base de datos. Si la conexión es exitosa, devuelve un objeto Connection. En caso de error, muestra un mensaje por consola y devuelve null.

Este tipo de clase es fundamental para centralizar y reutilizar el código de conexión en toda la aplicación. Gracias a ella, cualquier clase que necesite acceder a la base de datos puede hacerlo fácilmente llamando a ConexionDB.conectar().

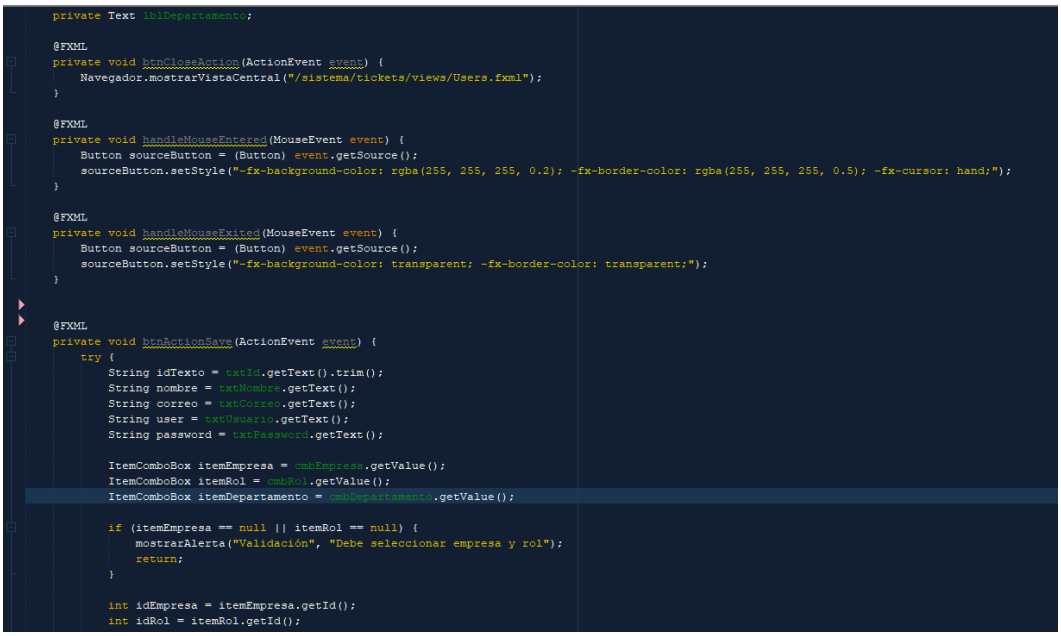


```java
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class ConexionDB{

    private static final String URL = "jdbc:postgresql://ep-falling-paper-a4s8n6xm-pooler.us-east-1.aws.neon.tech/neondb?user=
    private static final String USER = "neondb_owner";
    private static final String PASSWORD = "npg_vi0MeaqEYP6y";

    public static Connection conectar(){
        try {
            Connection conn = DriverManager.getConnection(URL, USER, PASSWORD);
            System.out.println("Base de datos conectada con exito!");
            return conn;
        } catch (SQLException e){
            System.err.println("Error al conectar: " + e.getMessage());
            return null;
        }
    }

}
```

Ejemplo de Clase en la Carpeta "Controladores"

Las clases dentro de la carpeta Controladores son responsables de manejar la lógica de negocio y la interacción entre la interfaz gráfica (las vistas en FXML) y los modelos de datos. Estas clases reciben las acciones del usuario (como clics de botones o selección de opciones), procesan la información, y ejecutan las acciones necesarias, como guardar datos, mostrar mensajes o cambiar de vista.

En aplicaciones JavaFX, estos controladores están asociados a archivos .fxml y permiten separar la lógica visual de la lógica funcional.

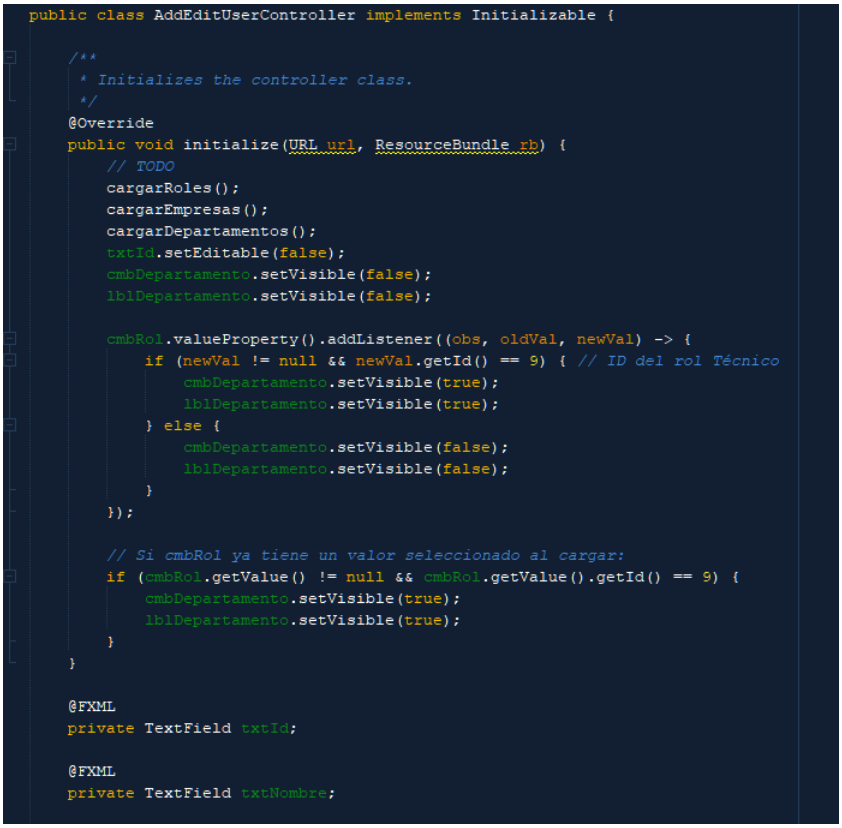


```java
    private Text lblDepartamento;

    @FXML
    private void btnCloseAction(ActionEvent event) {
        Navegador.mostrarVistaCentral("/sistema/tickets/views/Users.fxml");
    }

    @FXML
    private void handleMouseEntered(MouseEvent event) {
        Button sourceButton = (Button) event.getSource();
        sourceButton.setStyle("-fx-background-color: rgba(255, 255, 255, 0.2); -fx-border-color: rgba(255, 255, 255, 0.5); -fx-cursor: hand;");
    }

    @FXML
    private void handleMouseExited(MouseEvent event) {
        Button sourceButton = (Button) event.getSource();
        sourceButton.setStyle("-fx-background-color: transparent; -fx-border-color: transparent;");
    }


    @FXML
    private void btnActionSave(ActionEvent event) {
        try {
            String idTexto = txtId.getText().trim();
            String nombre = txtNombre.getText();
            String correo = txtCorreo.getText();
            String user = txtUsuario.getText();
            String password = txtPassword.getText();

            ItemComboBox itemEmpresa = cmbEmpresa.getValue();
            ItemComboBox itemRol = cmbRol.getValue();
            ItemComboBox itemDepartamento = cmbDepartamento.getValue();

            if (itemEmpresa == null || itemRol == null) {
                mostrarAlerta("Validación", "Debe seleccionar empresa y rol");
                return;
            }

            int idEmpresa = itemEmpresa.getId();
            int idRol = itemRol.getId();
```

A continuación, se muestra un ejemplo de clase controlador llamada AddEditUserController:

Esta clase gestiona el formulario para agregar o editar usuarios en el sistema. Algunas de sus responsabilidades principales son:

- **Inicialización**: En el método initialize, se cargan las listas de roles, empresas y departamentos, y se configura el comportamiento de la interfaz según el rol seleccionado.
- **Validación de campos**: Se asegura de que todos los datos ingresados por el usuario cumplan ciertos requisitos antes de guardar (longitud del nombre, formato de correo, seguridad de la contraseña, unicidad de usuario/correo, etc.).



```
public class AddEditUserController implements Initializable {

    /**
     * Initializes the controller class.
     */
    @Override
    public void initialize(URL url, ResourceBundle rb) {
        // TODO
        cargarRoles();
        cargarEmpresas();
        cargarDepartamentos();
        txtId.setEditable(false);
        cmbDepartamento.setVisible(false);
        lblDepartamento.setVisible(false);

        cmbRol.valueProperty().addListener((obs, oldVal, newVal) -> {
            if (newVal != null && newVal.getId() == 9) { // ID del rol Técnico
                cmbDepartamento.setVisible(true);
                lblDepartamento.setVisible(true);
            } else {
                cmbDepartamento.setVisible(false);
                lblDepartamento.setVisible(false);
            }
        });

        // Si cmbRol ya tiene un valor seleccionado al cargar:
        if (cmbRol.getValue() != null && cmbRol.getValue().getId() == 9) {
            cmbDepartamento.setVisible(true);
            lblDepartamento.setVisible(true);
        }
    }

    @FXML
    private TextField txtId;

    @FXML
    private TextField txtNombre;
```

- **Gestión del formulario**: Se muestra u oculta el campo de "departamento" según si el rol seleccionado es "Técnico".
- **Persistencia**: Usa una clase PersonaBuilder para crear el tipo adecuado de objeto (como Tecnico o Usuario), y luego guarda o actualiza el usuario en la base de datos.
- **Interacción visual**: También maneja eventos del mouse para cambiar el estilo de los botones y controla la navegación al cerrar el formulario.

Este controlador es un claro ejemplo de cómo se puede mantener una lógica ordenada y clara dentro de una interfaz gráfica, favoreciendo la reutilización del código y la experiencia del usuario.

Carpeta "Images"

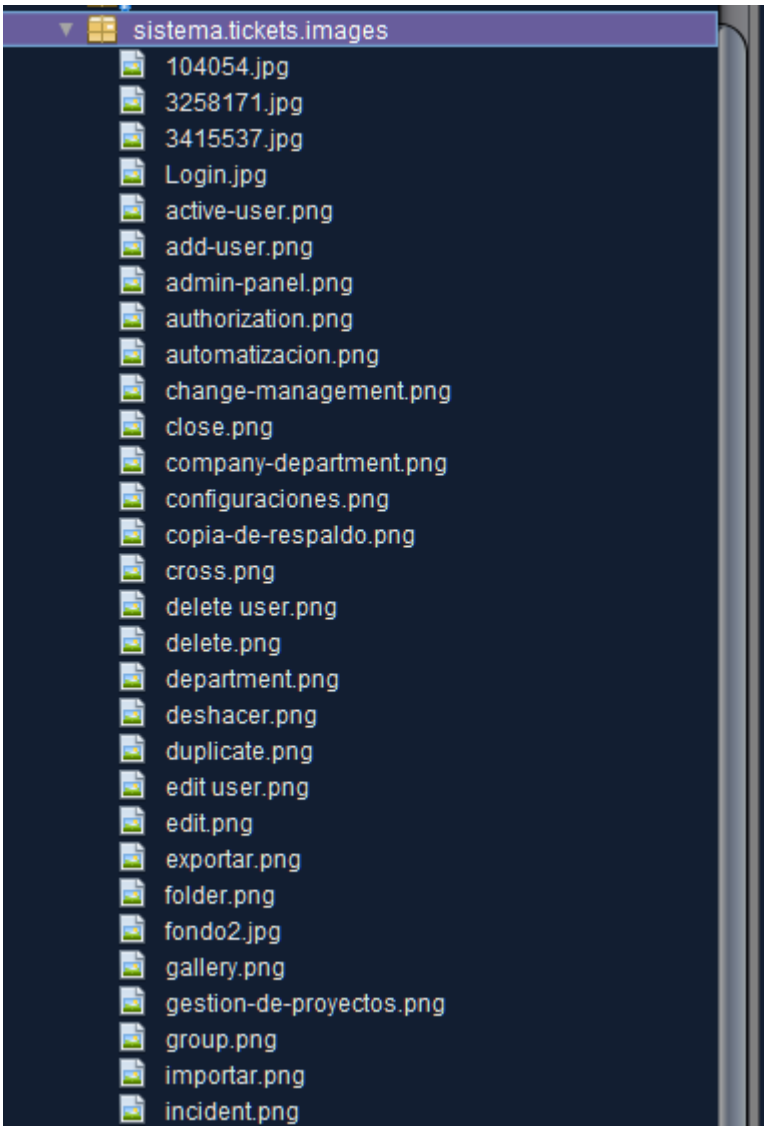
La carpeta Images está destinada al almacenamiento de recursos gráficos utilizados en la aplicación, principalmente **iconos e imágenes decorativas** que se emplean en las vistas diseñadas con JavaFX.

Su propósito es centralizar todos los elementos visuales, lo que permite:

- **Mantener el proyecto organizado**, evitando que las imágenes estén dispersas por otras carpetas.
- **Facilitar la reutilización de recursos**, ya que desde esta carpeta se pueden acceder a los mismos íconos en distintas partes de la interfaz.
- **Mejorar la presentación visual** de la aplicación, utilizando imágenes que acompañan botones, menús o encabezados, generando una experiencia más intuitiva para el usuario.

Por ejemplo, allí se pueden almacenar:

- Íconos para botones como “Agregar”, “Eliminar”, “Editar”, “Guardar”, etc.
- Logotipos de la empresa o aplicación.
- Imágenes de fondo o ilustraciones específicas.



Carpeta "Views"

La carpeta Views contiene todas las **interfaces gráficas de usuario** (GUI) de la aplicación. En un proyecto desarrollado con JavaFX, estas interfaces están diseñadas utilizando archivos **FXML**, que permiten definir la estructura visual de las ventanas, formularios, botones, campos de texto y demás elementos gráficos de forma declarativa.

¿Qué tipo de archivos contiene?

1. **Archivos .fxml**

   Son archivos XML que describen la disposición de los componentes gráficos (layouts, botones, tablas, etiquetas, etc.). Cada archivo .fxml representa una pantalla o una ventana específica dentro del sistema, como formularios de registro, tablas de visualización, menús, entre otros.

2. **Controladores asociados (Controller.java)**

   Aunque no están dentro de la carpeta Views, cada vista .fxml tiene una clase controladora en la carpeta Controladores que define la lógica de interacción: qué ocurre cuando el usuario hace clic en un botón, cambia una opción, etc.

3. **Estilos (opcional)**

   En algunos casos también pueden asociarse archivos .css para definir estilos personalizados que se aplican sobre los componentes visuales, mejorando la apariencia de la aplicación.

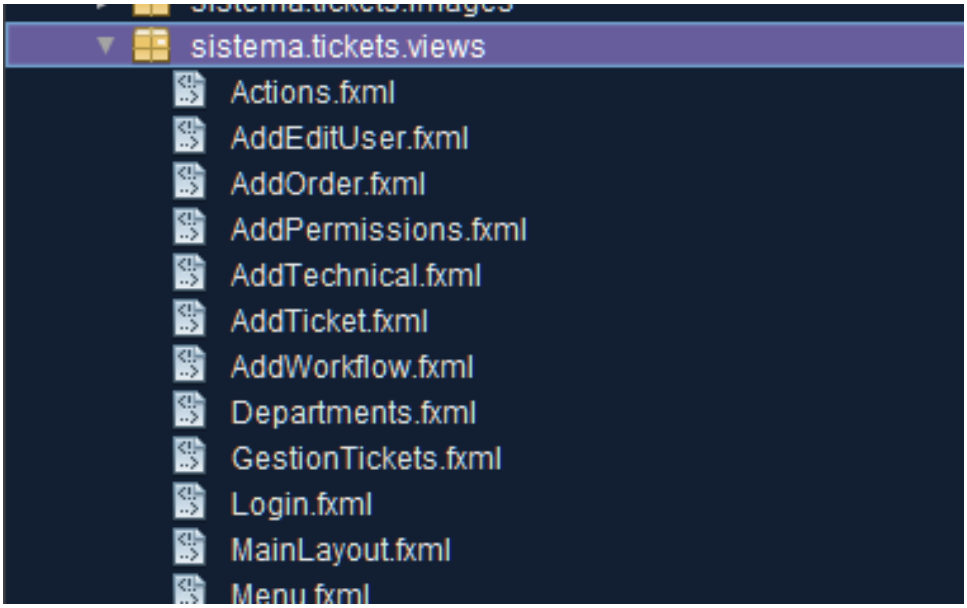
¿Qué hacen estos archivos?

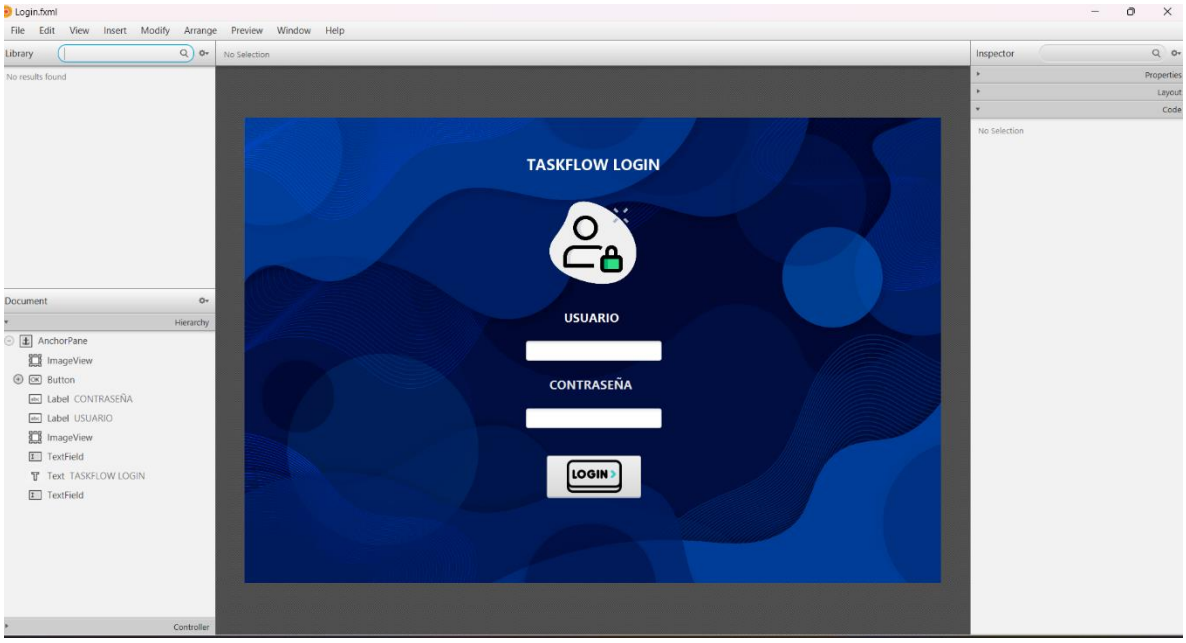
Los archivos de la carpeta Views:

- Definen **cómo se ve y se organiza la aplicación** ante el usuario.
- Permiten **separar la presentación de la lógica del programa**, facilitando el mantenimiento y la escalabilidad del sistema.
- Sirven como punto de entrada para que los controladores conecten los elementos visuales con las acciones programadas.

Por ejemplo:

- Un archivo llamado Users.fxml podría representar una vista donde se muestran los usuarios del sistema y se permite editarlos o eliminarlos.
- Otro llamado Login.fxml podría contener los campos para que un usuario se identifique antes de ingresar.




Login.fxml
File Edit View Insert Modify Arrange Preview Window Help
Library
No results found
No Selection
Inspector
Properties
Layout
Code
No Selection
Document
Hierarchy
AnchorPane
ImageView
Button
Label CONTRASEÑA
Label USUARIO
ImageView
TextField
Text TASKFLOW LOGIN
TextField
Controller
TASKFLOW LOGIN
USUARIO
CONTRASEÑA
LOGIN

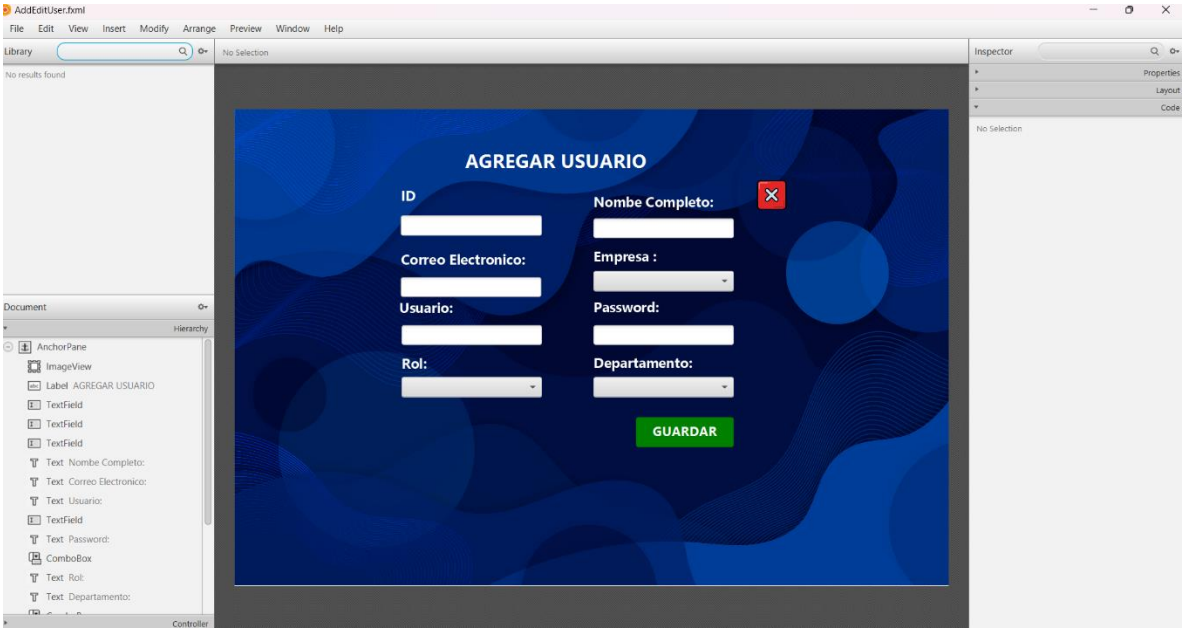
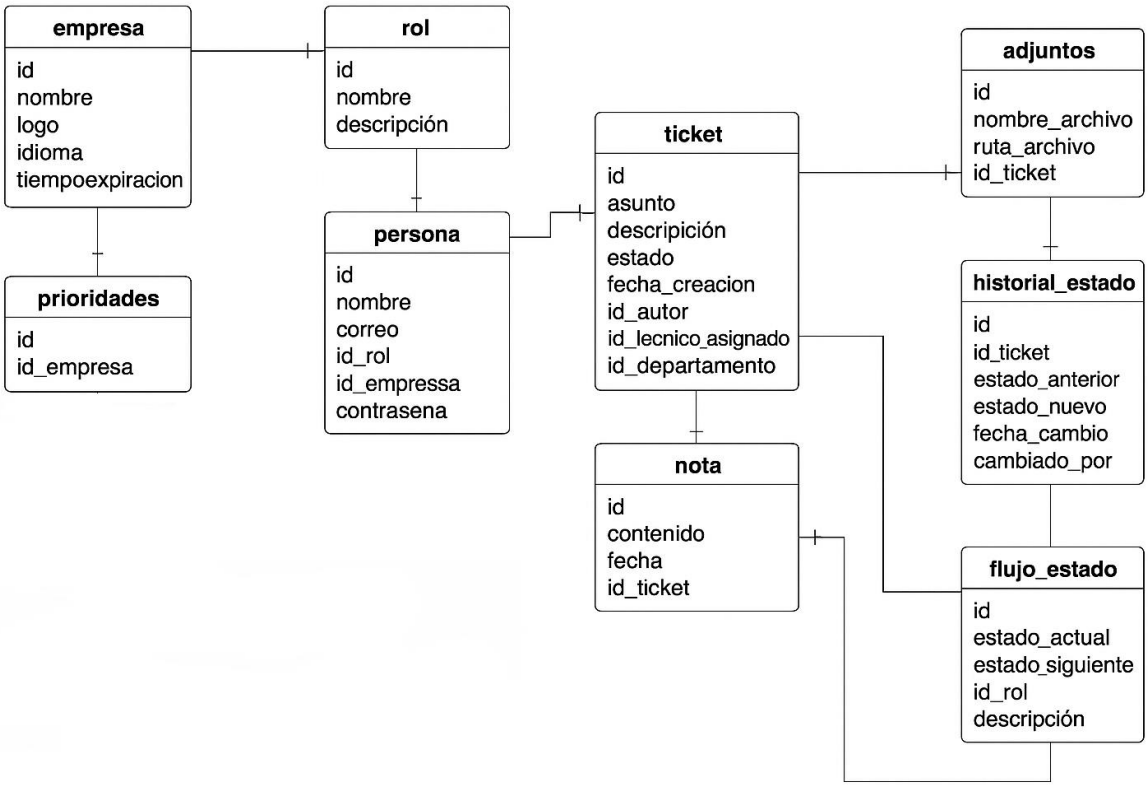
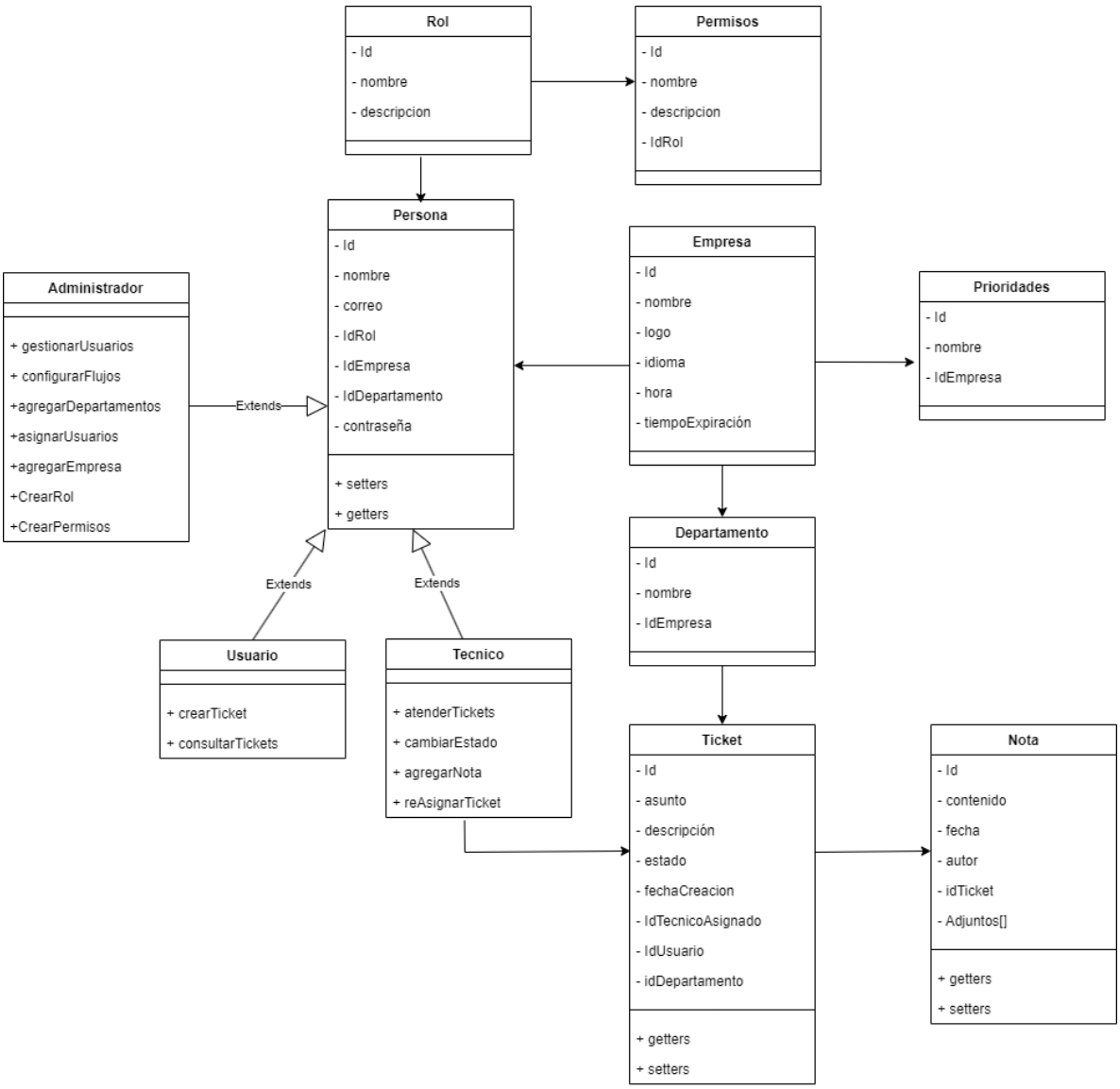

AddEditUser.fxml
File Edit View Insert Modify Arrange Preview Window Help
Library
No results found
No Selection
Inspector
Properties
Layout
Code
No Selection
Document
Hierarchy
AnchorPane
ImageView
Label AGREGAR USUARIO
TextField
TextField
TextField
Text Nombe Completo:
Text Correo Electronico:
Text Usuario:
TextField
Text Password:
ComboBox
Text Rol:
Text Departamento:
Controller
AGREGAR USUARIO
ID
Nombe Completo:
Correo Electronico:
Empresa :
Usuario:
Password:
Rol:
Departamento:
GUARDAR

DIAGRAMA ENTIDAD RELACIÓN



DIAGRAMAS UML




SISTEMA DE TICKETS
crear empresa
<<include>>
configurar parametros
configurar flujos de trabajo
Registrar Usuarios
<<include>>
asignar departamento
Crear Estados de los tickets
Registrar departamentos
Gestionar roles
<<include>>
gestionar permisos
Gestionar Tickets
<<include>>
crear ticket
<<include>>
asignar ticket
<<include>>
actualizar ticket
Consultar lista de tickets
Agregar nota al ticket
Administrador
Tecnico
Usuario