Ingeniería de Sistemas

ESTRUCTURA **DE DATOS**

PROYECTO FINAL

Integrantes

Sarah Montes

Valentina Ojeda

Manuel Julián Pacheco

Contenido

NOTAS ADICIONALES

Dado el caso no se cuenten con las herramientas de desarrollo para Java en versión 8, se necesitará crear una cuenta en Oracle para acceder al archivo para realizar la descarga.

Cabe resaltar que se requiere un computador portátil o un ordenador funcional, pues los programas a utilizar poseen una interfaz compleja y sirven múltiples funciones, haciendo que el desarrollo del proyecto se complique por otros dispositivos.

Se recomienda extremadamente además tener previos conocimientos de seudocódigo y lenguaje de programación java, pues la plataforma a utilizar depende vitalmente del uso de este lenguaje y posesión de los conocimientos, pues se harán uso de múltiples términos que podrán resultar complejos al lector si no cuenta con estos.

REQUISITOS

Para una mejor implementación de esta guía, se recomienda tener instalada la versión más reciente de *Apache NetBeans*. También es esencial contar con las herramientas de desarrollo para Java (JDK) en su versión 8 para garantizar un funcionamiento estable y óptimo de los componentes vitales, *JavaFX*, que es una plataforma requerida para la realización de la creación y desarrollo del proyecto. Por medio de las previamente mencionadas, se crea un proyecto compuesto por ventanas que contienen varios elementos capaces de ejercer múltiples. Por esto, también se estará utilizando la plataforma para creación de ventanas llamada *Scene Builder*. A continuación, se hallan las herramientas y sus vínculos a sus páginas, que contiene las descargas e información adicional.

Página de JavaFX

Página de Apache NetBeans

Página de JDK 8

Página de Scene Builder

INTRODUCCIÓN

Por medio de la creación de la guía paso a paso y sus componentes adicionales, se busca facilitar o apoyar el proceso de creación de un proyecto consistente de un pseudo – catalogo, conformado con botones, ventanas de dialogo emergentes, tablas, y entre otros elementos; que permiten realizar múltiples tareas que se asemejan a las de un catalogo de compras virtual auténtico, tales como guardar la información de un producto dentro de un carro de compras, para que posteriormente se almacenen los datos dentro de una tabla que contendrá la información. Además de esto, también se espera ayudar al lector fomentar conocimientos con respecto a Java, sus herramientas relacionadas y la creación de ventanas. Tener en cuenta que debido a que el plugin de Java Fx es del año 2010, no es estable comparada a otras plataformas y no se recomienda usar.

"La paciencia es amarga, pero dulces son sus frutos"
- Jean Jacques Rousseau

PACKAGE: IMAGES

Para este package, simplemente se cargaron las imágenes que se utilizaran para propósitos estéticos e informar visualmente al observador.

Paso a Paso:

1. **Preparación de las imágenes: Antes de integrarlas en el proyecto, se seleccionan las imágenes que se usarán. Hay que de que estén en un formato compatible (como PNG, JPEG o GIF) y tengan una resolución adecuada para evitar distorsiones.**
2. **Creación de un directorio para las imágenes: En el proyecto NetBeans, se crea un directorio específico para almacenar las imágenes. Se hace clic derecho en el proyecto, se selecciona "Nuevo" y luego "Carpeta". Se nombra de manera descriptiva, por ejemplo, en el proyecto, como es una tienda de instrumentos, se pone directamente el nombre correspondiente del instrumento a la imagen.**
3. **Importación de las imágenes: Una vez creada la carpeta, se copian las imágenes que se deseen integrar al proyecto y se pegan dentro de esta carpeta utilizando el explorador de archivos del sistema operativo.**
4. **Agregar imágenes al código fuente: En el código Java del proyecto, se puede acceder a estas imágenes utilizando su ruta relativa.**
5. **Verificación y ejecución: Después de agregar la imagen al código, es esencial ejecutar el proyecto para confirmar que las imágenes se cargan adecuadamente y se muestran en la interfaz de usuario según lo previsto.**

A continuación, se presentan como quedan las imágenes cargadas dentro del proyecto:

Figura 1: Archivos del Proyecto:

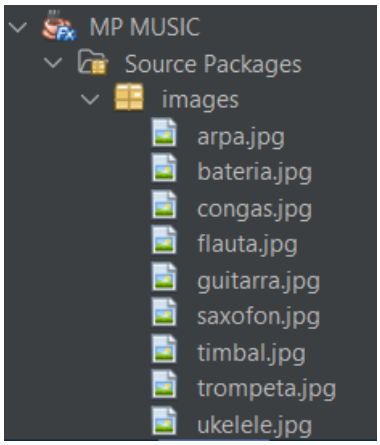
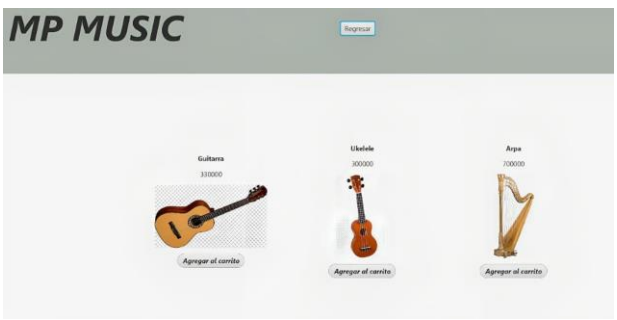
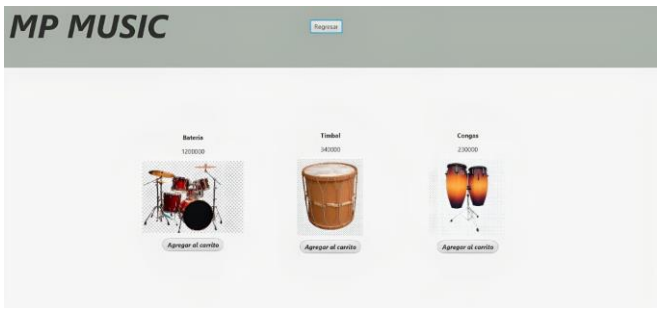


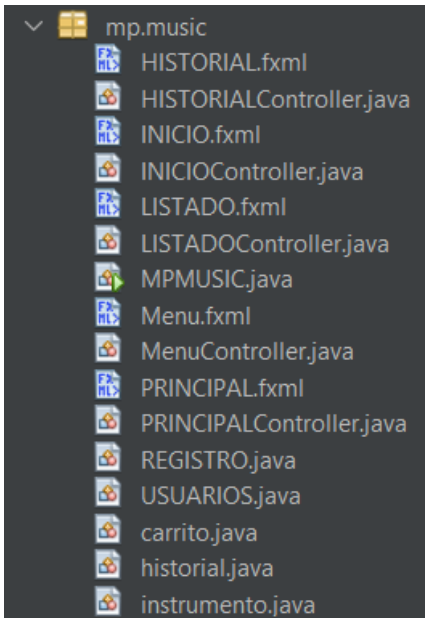
Figura 2 y 3: Visualización dentro de la ventana:





PACKAGE: MP. MUSIC

El paquete o package contiene las todas clases con sus respectivas clases, archivos FXML y clase principal, donde se halla los elementos esenciales para el funcionamiento de la ventana. A continuación, se realiza en orden la muestra de las clases:



Para facilitar y evitar la extensión de la guía, se realizará un paso a paso general para todos los archivos *FXML*, que son ventanas para ser visualizadas.

Las únicas excepciones al orden son la clase principal *MPMUSIC* y los archivos *FXML*, puesto a que son ejecutables y requieren a las otras clases para ser ejecutados.

CLASE: HISTORIAL.JAVA

Métodos:

- **initialize: Inicializa el controlador de la clase, estableciendo las columnas de la tabla y configurando el contenido.**
- **actionEvent: Maneja los eventos de acción, en este caso, específicamente el botón de regresar.**
- **LoadStage: Carga una nueva ventana o escena y muestra la misma. Además, maneja el evento de cierre de la ventana.**
- **handle: Maneja el evento de cierre de la ventana.**

Atributos:

- **list: Instancia de la clase REGISTRO.**
- **table: TableView donde se muestran los datos.**
- **nom, desc, precio: TableColumn para la tabla, definidos para los nombres, descripciones y precios respectivamente.**
- **historia: Lista observable de elementos de tipo instrumento.**
- **btnRegresar: Botón para regresar al menú principal.**

Paso a paso:

Inicialización de Variables y Métodos:

1. **Se inicializan** las variables y métodos requeridos para el controlador, incluyendo listas, tablas y botones.
2. **Configuración de TableView:** Se configura la TableView llamada table para mostrar información sobre los instrumentos.
3. **Se definen las columnas** (nom, desc, precio) que representan el nombre, la descripción y el precio de los instrumentos.
4. **Manejo de Eventos:** Se implementa un método llamado actionEvent para manejar eventos de clic en el botón btnRegresar.
   Al hacer clic en este botón, se carga la ventana PRINCIPAL.fxml.
5. **Inicialización del Controlador**: Se ejecuta el método initialize al inicio, configurando la TableView y sus columnas para mostrar los datos del historial de instrumentos.
6. **Carga de Nueva Ventana**: El método LoadStage maneja la carga de una nueva ventana.
7. Oculta la ventana actual y muestra una nueva ventana cargada desde el archivo FXML PRINCIPAL.fxml.
8. **Cierre de la Aplicación**: Se implementa un evento que se activa cuando se cierra la ventana. Al cerrarse, la aplicación se cierra completamente usando Platform.exit().

CLASE: MENUCONTROLLER.JAVA

Atributos:

- **list: Objeto de tipo REGISTRO.**
- **pila: Objeto de tipo carrito.**
- **nom1, nom2, nom3, lima1, lima2, lima3, pre1, pre2, pre3, desc1, desc2, desc3: Variables de tipo String.**
- **lbl1, lbl2, lbl3, lbprecio1, lbprecio2, lbprecio3: Objetos de tipo Label.**
- **ima1, ima2, ima3: Objetos de tipo ImageView.**
- **btnRegresar, btninfo1, btninfo2, btninfo3: Objetos de tipo Button.**

Atributos:

- **initialize:** Método que inicializa la clase, configurando las etiquetas, imágenes y botones con los datos iniciales.
- **actionEvent:** Método que maneja los eventos de los botones (btnRegresar, btninfo1, btninfo2, btninfo3), realizando acciones específicas como cargar otra ventana FXML o agregar un instrumento a una lista.

Paso a paso:

Inicialización de Variables y Métodos:

1. **Declaración de atributos:** Se definen atributos para varios elementos de la interfaz gráfica, como etiquetas (Label), imágenes (ImageView) y botones (Button), así como objetos de otras clases (REGISTRO y carrito).
2. **Método actionEvent:** Este método maneja eventos de botones. Cuando se hace clic en un botón (btnRegresar, btninfo1, btninfo2, btninfo3), se ejecuta una acción específica.

- **btnRegresar:** Al hacer clic, carga la ventana PRINCIPAL.fxml.
- **btninfo1, btninfo2, btninfo3:** Al hacer clic, agrega un instrumento a la lista instrumentos y la pila pila.

3. **Método initialize: Este método inicializa la interfaz gráfica con los datos iniciales.**

- Asigna texto y contenido de imagen a los elementos visuales (Label e ImageView).

4. **Método LoadStage: Este método carga una nueva ventana FXML y muestra su contenido.**

- Se oculta la ventana actual.
- Se carga el nuevo FXML.
- Se crea una nueva ventana (Stage) y se muestra.

5. **Objetos FXMLLoader, Node, Scene, Stage, y WindowEvent:** Se utilizan para manejar y cargar distintas escenas y ventanas FXML.

CLASE: PRINCIPALCONTROLLER.JAVA

Atributos:

- **btnViento, btnPercusion, btnCuerda, btnClose, btnCarrito, btnHistorial: Objetos de tipo Button que representan botones en la interfaz gráfica.**
- **lbl1, lbl2, lbl3, lbprecio1, lbprecio2, lbprecio3: Objetos de tipo Label que representan etiquetas en la interfaz gráfica.**
- **ima1, ima2, ima3: Objetos de tipo ImageView que representan imágenes en la interfaz gráfica.**

Métodos:

- **actionEvent(ActionEvent e):** Maneja eventos de acción en botones.
- **initialize(URL url, ResourceBundle rb):** Método de inicialización de la interfaz gráfica.
- **LoadStage(String url, Event event):** Carga una nueva ventana FXML y muestra su contenido

Paso a paso:

1. **Paquetes Importados:**

- Importaciones de bibliotecas y clases necesarias para el funcionamiento del programa.

2. **Declaración de Clase:**

- PRINCIPALController es una clase que controla la lógica de la interfaz gráfica.
- Implementa la interfaz Initializable de JavaFX, lo que implica la implementación del método initialize.

3. **Declaración de atributos:**

- Para manejar botones, etiquetas, imágenes y otros elementos de la interfaz.
- Inicialización de algunas variables con valores específicos.

4. **Métodos:**

- actionEvent(ActionEvent e): Maneja los eventos generados por los botones de la interfaz.
- initialize(URL url, ResourceBundle rb): Inicializa la interfaz gráfica y establece algunos valores iniciales.
- LoadStage(String url, Event event): Carga una nueva ventana FXML y muestra su contenido.

5. **Implementación de Acciones:**

- Se establecen acciones para cada botón, definiendo lo que sucede al hacer clic en ellos.
- Dependiendo del botón presionado (btnViento, btnPercusion, btnCuerda, btnClose, btnCarrito, btnHistorial), se configuran valores y se carga una nueva ventana FXML.

6. **Inicialización:**

- Se inicializan objetos y estructuras de datos (Stack) en el método initialize.

7. **Gestión de Escenarios:**

- La función LoadStage maneja la apertura y cierre de ventanas FXML.

CLASE: INICIOCONTROLLER.JAVA

Atributos:

- **btnLogin: Botón para el inicio de sesión.**
- **btnSignup: Botón para el registro de usuarios.**
- **txtIUser: Campo de texto para ingresar el nombre de usuario en el inicio de sesión.**
- **txtRUser: Campo de texto para ingresar el nombre de usuario en el registro.**
- **txtIPass: Campo de texto para ingresar la contraseña en el inicio de sesión.**
- **txtRPass: Campo de texto para ingresar la contraseña en el registro.**
- **UsuarioActual:** String - Variable estática para almacenar el usuario actual.

Métodos:

- **initialize(URL url, ResourceBundle rb):** void - Método de inicialización del controlador. Se ejecuta al cargar el FXML y se encarga de configurar la interfaz gráfica.
- **LoadStage(String url, Event event):** void - Método para cargar y mostrar una nueva ventana FXML en función de la URL proporcionada.
- **actionEvent(ActionEvent e):** void - Método para manejar los eventos generados por los botones btnSignup y btnLogin. Realiza la lógica correspondiente al registro e inicio de sesión de usuarios.
- **LoadStage(String url, Event event):** void - Método para cargar y mostrar una nueva ventana FXML en función de la URL proporcionada. Se encarga de cerrar la ventana actual y mostrar una nueva.
- **Logger.getLogger(INICIOController.class.getName()).log(Level.SEVERE,null, ex**): void - Método para registrar excepciones ocurridas durante la ejecución en el logger.
- **Platform.exit(): void -** Método para cerrar la aplicación JavaFX.
- **listaUser.llenarlista(user, pass):** void **-** Método para llenar una lista de usuarios con un nombre de usuario y contraseña proporcionados.
- **listaUser.login(user, pass):** USUARIOS - Método para realizar el proceso de inicio de sesión con un nombre de usuario y contraseña proporcionados.
- **JOptionPane.showMessageDialog(null, "Datos de acceso incorrectos, ingrese nuevamente"):** void - Método para mostrar un mensaje de alerta en caso de que los datos de inicio de sesión sean incorrectos.

Paso a paso:

1. **Crear el archivo FXML:**

- Se debe crear un archivo FXML llamado INICIO.fxml utilizando un entorno de desarrollo integrado (IDE).
- Diseñar la interfaz gráfica con elementos como botones (btnLogin y btnSignup) y campos de texto (txtIUser, txtRUser, txtIPass, txtRPass). Asignar identificadores utilizando fx:id.

2. **Crear la clase INICIOController:**

- Se debe crear una nueva clase llamada INICIOController en el proyecto.
- Implementar la interfaz Initializable en INICIOController para utilizar el método initialize.
- Anotar los elementos de la interfaz con @FXML que se deseen controlar desde el código (btnLogin, btnSignup, txtIUser, txtRUser, txtIPass, txtRPass).
- Definir los métodos actionEvent y LoadStage para manejar eventos de botones y cargar nuevas escenas, respectivamente.
- Escribir el código lógico dentro del método actionEvent para manejar las acciones del usuario, como iniciar sesión (btnLogin) y registrarse (btnSignup).

CONTINUACIÓN: INICIOCONTROLLER.JAVA

3. **Configurar el entorno y las dependencias:**

- Asegurarse de que el entorno de desarrollo esté configurado para trabajar con Java y JavaFX.
- Configurar las dependencias necesarias para JavaFX en el proyecto.

4. **Conectar el controlador al archivo FXML:**

- Vincular el controlador INICIOController al archivo FXML INICIO.fxml.

5. **Probar y ejecutar:**

- Crear una nueva instancia de la aplicación JavaFX.
- Utilizar INICIOController como controlador de la esc

CLASE: LISTADOCONTROLLER.JAVA

Atributos:

- **list**: Instancia de la clase REGISTRO.
- **lista:** Instancia de la clase historial.
- **pila**: Instancia de la clase carrito.
- **tabla**: TableView que muestra una lista de instrumentos.
- **nom**: TableColumn para el nombre del instrumento.
- **desc**: TableColumn para la descripción del instrumento.
- **precio**: TableColumn para el precio del instrumento.
- **instrumentos**: ObservableList que contiene la lista de instrumentos.
- **btnRegresar**: Botón para regresar a la pantalla principal.
- **btnComprar**: Botón para comprar un instrumento.

Métodos:

- **initialize(URL url, ResourceBundle rb):** Método de inicialización de la clase. Configura TableView y TableColumn.
- **actionEvent(ActionEvent e):** Método para manejar eventos de botones (btnRegresar y btnComprar). Al hacer clic en btnRegresar, carga la escena principal desde un archivo FXML. Al hacer clic en btnComprar, realiza las operaciones necesarias para añadir un instrumento al historial, al carrito y lo elimina de la lista de instrumentos.
- **LoadStage(String url, Event event):** Método privado que carga una nueva escena (ventana) según la URL proporcionada. Se ejecuta al presionar un botón para cambiar a otra pantalla.

CONTINUACIÓN: LISTADOCONTROLLER.JAVA

Paso a paso:

1. **Declaración de paquete e importaciones:**
   Se importan clases y paquetes necesarios para el funcionamiento del programa, como clases relacionadas con JavaFX y manejo de excepciones.
2. **Declaración de la clase LISTADOController:**
   Es una clase que implementa la interfaz Initializable, contiene atributos vinculados a la interfaz gráfica: TableView, TableColumn, ObservableList, y botones.
3. **Método actionEvent(ActionEvent e):**
   Maneja los eventos de los botones (btnRegresar y btnComprar).
   Al hacer clic en btnRegresar, carga la escena principal desde un archivo FXML. Al hacer clic en btnComprar, realiza operaciones para agregar el instrumento seleccionado al historial, al carrito y lo elimina de la lista de instrumentos.
4. **Método initialize(URL url, ResourceBundle rb):**
   Se ejecuta al inicializar la clase.
   Configura las columnas de la tabla (TableView) para mostrar los atributos de los instrumentos (nom, desc, precio) mediante PropertyValueFactory.
   Asigna la lista observable de instrumentos (instrumentos) a la tabla para su visualización.
5. **Método LoadStage(String url, Event event):**
   Método privado que carga una nueva escena (ventana) según la URL proporcionada.
   Se ejecuta al presionar un botón para cambiar a otra pantalla.
   Oculta la ventana actual y muestra una nueva ventana cargada desde un archivo FXML.

OTRAS CLASES:

Lista de Clases Adicionales:

1. **Registro**
2. **Usuarios**
3. **Carrito**
4. **Historial**
5. **Instrumento**

Lista de Atributos y Métodos:

Clase REGISTRO: Administra el registro de usuarios

Atributos:

- **cap (tipo: USUARIOS)**

Métodos:

- **public USUARIOS existenombre(String user)**
- **public USUARIOS crearnodo(String user, String pass)**
- **public void llenarlista(String user, String pass)**
- **public USUARIOS login(String user, String pass)**

Clase USUARIOS: Representa a un usuario.

Atributos:

- **user (tipo: String)**
- **pass (tipo: String)**
- **SIG (tipo: USUARIOS)**

Métodos:

- **public USUARIOS(String user, String pass) (Constructor)**

Clase carrito: Maneja un carrito de compras

Atributos:

- **pila (tipo: Stack<instrumento>)**

Métodos:

- **public instrumento buscarnombre(String nom)**
- **public void push(String nom, String desc, String precio, String comprador)**
- **public void comprar(String nom)**

Clase historial: Gestiona el historial de compras.

Atributos:

- **lista (tipo: instrumento)**

Métodos:

- **public historial()**
- **public void crearNodo(String nom, String desc, String precio)**

Clase instrumento: Modela un instrumento musical.

Atributos:

- **nom (tipo: String)**
- **desc (tipo: String)**
- **precio (tipo: String)**
- **comprador (tipo: String)**
- **sig (tipo: instrumento)**
- **ant (tipo: instrumento)**

Métodos:

- **public instrumento()**
- **public instrumento(String nom, String desc, String precio, String comprador)**
- **public String getComprador()**
- **public void setComprador(String comprador)**
- **public String getNom()**
- **public void setNom(String nom)**
- **public String getDesc()**
- **public void setDesc(String desc)**
- **public String getPrecio()**
- **public void setPrecio(String precio)**

CLASE PRINCIPAL: MPMUSIC.JAVA

Propósito:

- Es la clase principal de la aplicación.

Métodos:

- **start(Stage stage):** Este método se llama al iniciar la aplicación y establece la escena inicial cargando el archivo FXML INICIO.fxml.
- **main(String[] args): Este método es el punto de entrada principal de la aplicación Java y lanza la aplicación JavaFX.**

Explicación Detallada:

- start(Stage stage): En este método se carga la interfaz de usuario (INICIO.fxml) utilizando FXMLLoader. Este archivo FXML define la interfaz gráfica inicial de la aplicación. Se crea una escena con la interfaz cargada y se establece en la etapa (ventana) principal de la aplicación.
- main(String[] args): Es el método principal de la aplicación. Llama al método launch(args) que inicia la aplicación JavaFX, inicializando y mostrando la interfaz gráfica en la ventana principal.
- Este código es esencial para iniciar la aplicación JavaFX. Establece la estructura principal para la creación de la ventana y la carga de la interfaz de usuario definida en el archivo FXML INICIO.fxml.

Paso a paso:

1. **Creación del archivo MPMUSIC.java:**

- Se crea un nuevo archivo Java llamado MPMUSIC.java.

2. **Estructura del código:**

- Se copia el código proporcionado en MPMUSIC.java.

3. **Importaciones necesarias:**

- Se asegura de tener las importaciones adecuadas al inicio del archivo MPMUSIC.java, las cuales incluyen:
- import javafx.application.Application;
- import javafx.fxml.FXMLLoader;
- import javafx.scene.Parent;
- import javafx.scene.Scene;
- import javafx.stage.Stage;

4. **Creación de la clase MPMUSIC:**

- Define la clase MPMUSIC, la cual extiende Application.
- Implementa el método start(Stage stage) dentro de la clase MPMUSIC.
- Dentro del método start(), se carga el archivo FXML INICIO.fxml usando FXMLLoader, y se establece como el contenido de la escena.
- Se establece la escena en el Stage y se muestra la ventana principal mediante stage.show().

5. **Método main():**

- Se asegura de tener el método main(String[] args) que simplemente llama al método launch(args) para iniciar la aplicación JavaFX.

6. **Archivo FXML INICIO.fxml:**

- Se debe contar con un archivo FXML llamado INICIO.fxml, el cual define la interfaz gráfica inicial de la aplicación.

7. **Ejecución de la aplicación:**

- Se ejecuta la aplicación al iniciar la clase MPMUSIC.

CREACION DE VENTANAS

- Como esta es una aplicación distinta, no se estará describiendo los atributos y métodos, pues estos están previamente descritos en las otras clases.
- Se estará haciendo uso del software llamado “Scene Builder”, que se halla al principio de la guía su respectiva página.

Paso a paso:

1. Abre Scene Builder: Inicia Scene Builder, ya descargado en la computadora.

2. Selecciona un Diseño Base: Se elige un diseño base para la ventana.

3. Arrastra y Suelta Elementos: Desde la barra de herramientas de la izquierda, arrastra y suelta elementos como botones, etiquetas o campos de texto en el área de diseño. Se colocan estos elementos en la posición deseada.

4. Ajusta Propiedades: Hacer clic en los elementos agregados para seleccionarlos y ajustar sus propiedades.

5. Guarda el Diseño: Ir a "Archivo" y selecciona "Guardar" para guardar el diseño en un archivo FXML en tu sistema.

6. Generar el Controlador (Opcional): Si se desea agregar interacciones a los elementos (por ejemplo, para manejar eventos de botones), puedes generar un controlador. Para hacerlo, ir a "Ver" > "Controladores" y especificar el controlador que se desea usar. Luego, Scene Builder generará un archivo de controlador FXML relacionado.

7. Cierra Scene Builder: Una vez que se haya diseñado la ventana, cierra Scene Builder.

8. Usa Scene Builder en la Aplicación: En la aplicación, se usa un FXMLLoader para cargar el archivo FXML creado con Scene Builder. Luego, muestra la interfaz de usuario en una ventana, como se describe en el paso a paso anterior.

CONCLUSIONES

- El manejo de la creación de ventanas en el desarrollo de software es fundamental para la construcción de interfaces gráficas intuitivas y eficientes. A través de la implementación de códigos como los utilizados para cargar diferentes vistas, se logra una navegación coherente entre las secciones de una aplicación. Esta gestión permite una experiencia de usuario más amigable, facilitando la interacción y mejorando la usabilidad. Además, comprender el proceso de creación de ventanas es esencial para el diseño modular y la construcción eficaz de sistemas, lo que posibilita una estructura organizada y escalable en el desarrollo de aplicaciones más complejas. En resumen, el manejo adecuado de la creación de ventanas no solo mejora la estética visual, sino que también impacta en la funcionalidad y la eficiencia general de una aplicación.