**Content**

[**Java GUI** 2](#_Toc177846796)

[**Different ways to program GUIs in Java** 2](#_Toc177846797)

[**AWT (Abstract Window Toolkit)** 2](#_Toc177846798)

[**Swing** 2](#_Toc177846799)

[**SWT (Standard Widget Toolkit)** 2](#_Toc177846800)

[**JavaFX** 2](#_Toc177846801)

[**Comparison of their characteristics** 2](#_Toc177846802)

[**Scene Builder** 3](#_Toc177846803)

[**JFoenix** 3](#_Toc177846804)

[**MVC in JavaFx** 3](#_Toc177846805)

[**Model** 4](#_Toc177846806)

[**View** 4](#_Toc177846807)

[**Controller** 4](#_Toc177846808)

[**Material we will use:** 4](#_Toc177846809)

[**Download Intellij IDEA** 4](#_Toc177846810)

[**Descargar JDK 17.0.1** 7](#_Toc177846811)

[**Descargar J FoeniX for Java 9** 8](#_Toc177846812)

[**Create new project in Intellij IDEA** 8](#_Toc177846813)

[**Scene builder contains:** 12](#_Toc177846814)

[**Explicación archivo module-info.java** 13](#_Toc177846815)

[**Exercise 1: Hello World** 14](#_Toc177846816)

[**Explanation of the controller** 19](#_Toc177846817)

[**@FXML:** 19](#_Toc177846818)

[**Exercise 2: Login** 27](#_Toc177846819)

[**Apply css** 38](#_Toc177846820)

[**Advantages of using CSS vs. styling elements from the Scene Builder's own options** 40](#_Toc177846821)

[**Advantages of using CSS:** 40](#_Toc177846822)

[**Disadvantages of using Scene Builder for layouts:** 40](#_Toc177846823)

[**Notes** 41](#_Toc177846824)

# **Java GUI**

A GUI in Java is a visual interface that allows users to interact with an application graphically, rather than using the command line. Instead of typing text commands, users can use buttons, menus, text fields, and other visual elements to interact with the application.

## **Different ways to program GUIs in Java**

### **AWT (Abstract Window Toolkit)**

AWT is an older library that is also part of the Java standard library. It provides basic components for building graphical interfaces, such as buttons and menus, but its functionality is limited compared to Swing and JavaFX.

### **Swing**

Swing is a graphics library that has been part of the Java standard library since JDK 1.2. It provides a number of graphical components, such as buttons, panels, tables and dialog boxes, that you can use to build graphical interfaces.

### **SWT (Standard Widget Toolkit)**

SWT is a toolkit for building graphical interfaces in Java using native operating system components, providing a more consistent look and feel with system applications.

### **JavaFX**

JavaFX is a platform for creating graphical user interface (GUI) applications in Java, introduced in Java 8 to replace Swing as the main GUI library. It provides powerful tools for developing modern, stylized visual interfaces, from desktop to mobile and web applications, using CSS for styling and FXML (an XML language) for interface design.

**Comparison of their characteristics:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Aspect** | **AWT** | **SWT** | **Swing** | **JavaFX** |
| Launch | Early years of Java | Developed by Eclipse Foundation | Introduced in Java 1.2 (1998) | Successor to Swing, launched in 2011 |
| Components | Basic, native to the operating system | Native operating system widgets | More advanced, non-native | Advanced and customizable components |
| Styles | Limited to OS options | More options, but still limited | Customization with Look & Feel | Full CSS support |
| Advanced Graphics | No | No | No | Yes (2D and 3D) |
| Performance | Medium | High | Medium | High |
| Portability | High | Medium | High | High |
| Ease of Use | Basic and simple, but less flexible | More advanced, but requires OS knowledge | Más flexible que AWT, pero más complejo que JavaFX | Modern and easy to use with tools such as Scene Builder |

**We will work with JavaFx**

# **Scene Builder**

It is a graphical tool that simplifies the design of interfaces in JavaFX.

It allows you to design user interfaces by dragging and dropping visual components, without the need to write code.

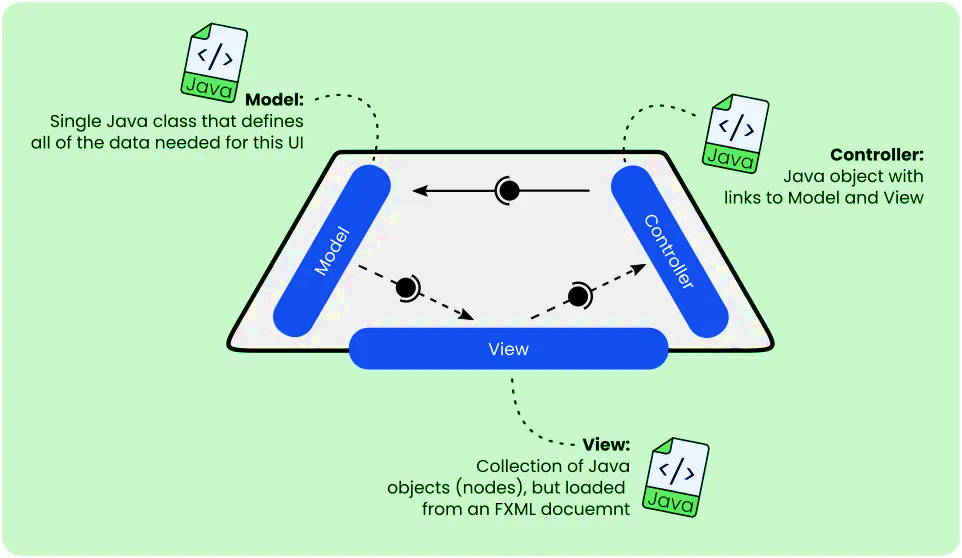
**Features:**

* **Visual Interface:** Provides an intuitive design environment.
* **Real Time Preview:** Allows you to see how the interface will look like as you design it.
* **Automatic Code Generation:** Generates the FXML code that describes the graphical interface.

# **JFoenix**

It is a component library for JavaFX that provides UI controls with a modern style inspired by Google's Material Design. It is designed to help developers create desktop applications with a more attractive and responsive look and feel. JFoenix extends the standard JavaFX controls, adding new styles and functionality, such as buttons, text boxes, progress bars, lists and dialogs, that follow the visual principles of Material Design.

# **MVC in JavaFx**

****

## **Model**

The model represents the application data and business logic. It should not have any direct dependencies on the view or controller, thus allowing the business logic to be kept separate from the user interface.

**Features:**

* **Properties:** Usually JavaFX properties such as StringProperty, IntegerProperty, etc., are used to enable observation and data binding.
* **Validation:** You can include logic to validate data within the model.
* **Persistence:** The model can include methods to persist data in a database or file.

## **View**

The view is responsible for displaying data to the user and providing user interface controls. In JavaFX, you can build the view using JavaFX components in code or in FXML files.

**Features:**

* **Data Binding:** You can bind model properties directly to view controls.
* **Styling:** You can apply CSS to style the view.

## **Controller**

The controller handles the user interface logic and acts as an intermediary between the model and the view. It responds to view events and updates the model as needed.

**Features:**

* **Event Handling:** Configures event handlers for view controls.
* **View Update:** Can update the view when the model changes, and vice versa.
* **Model Interaction:** The controller can call methods on the model to obtain or modify data.

# **Material we will use:**

JDK 17.0.1

JavaFX SceneBuilder 22.0.0

IntelliJ IDEA 2023.3.8

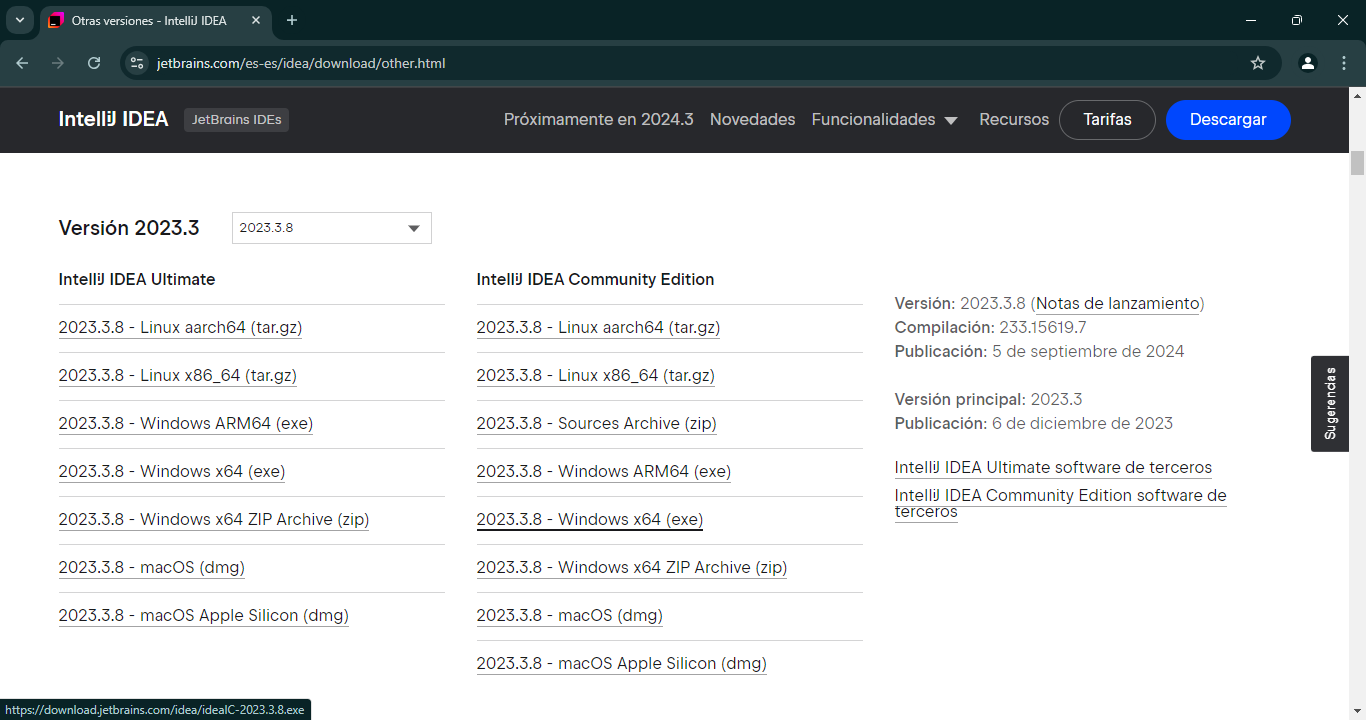
JFoenix 9.0.0

# **Download Intellij IDEA**

Search for Intellij IDEA download or enter the following link

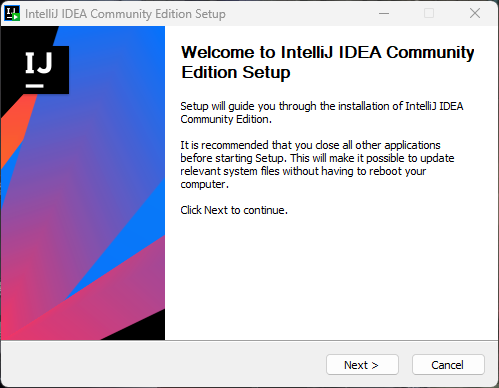
<https://www.jetbrains.com/es-es/idea/download/other.html>

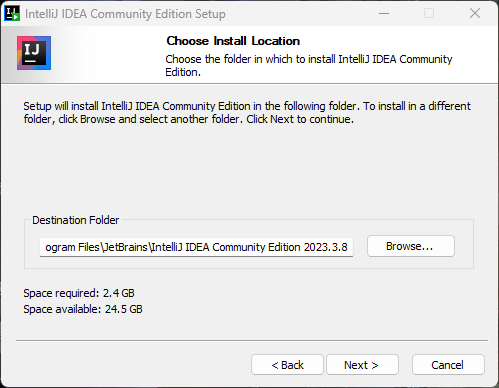
Search IntelliJ IDEA Community Edition to download the free version, download .exe (Windows)

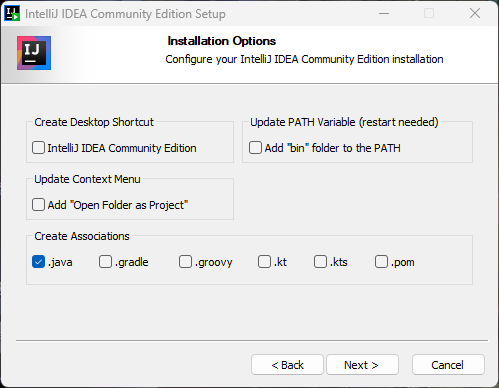


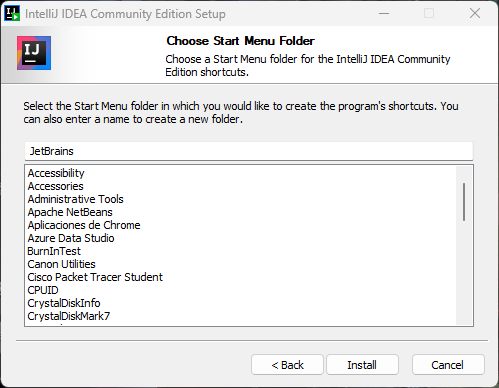


The wirzar opens



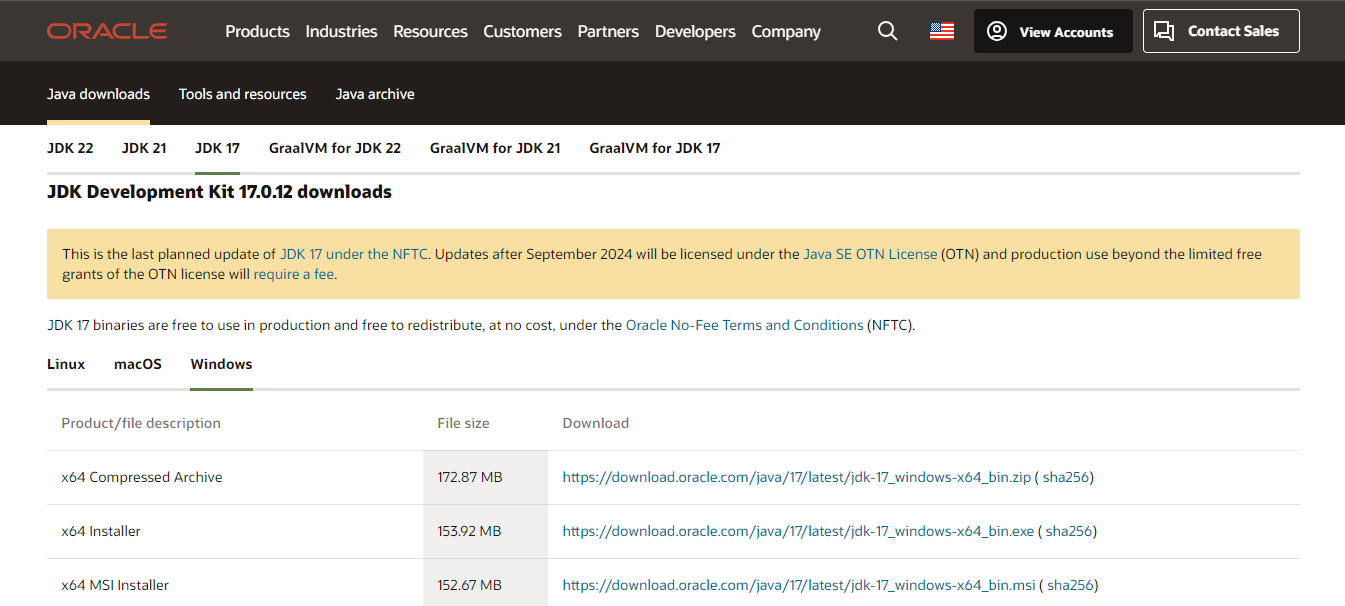






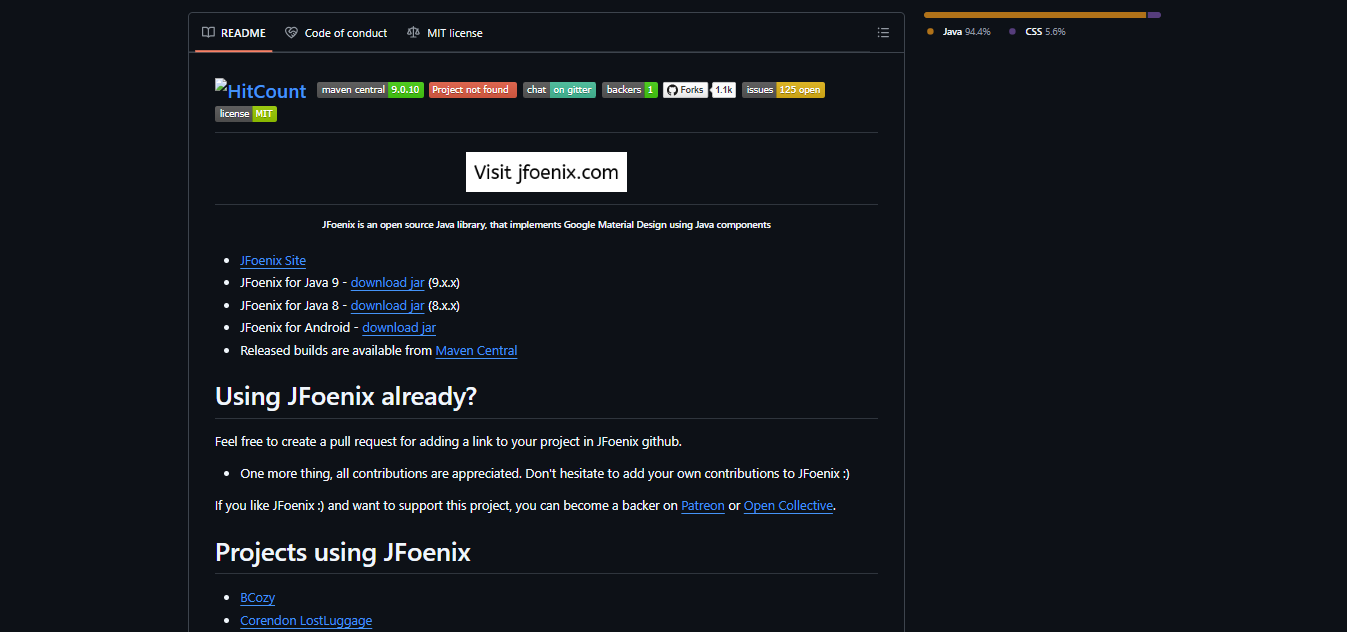
# **Descargar JDK 17.0.1**

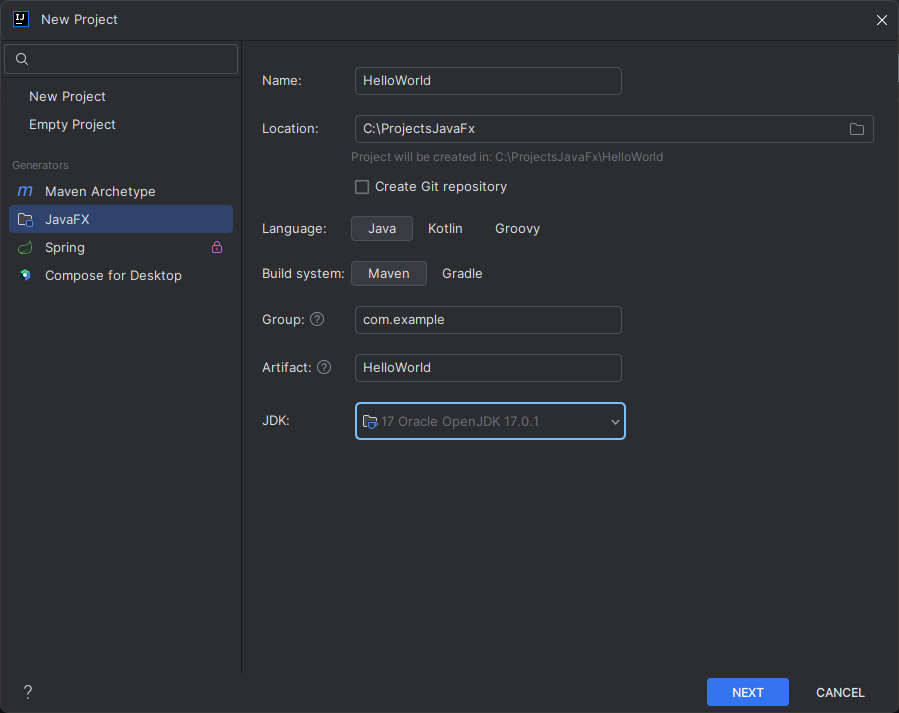
<https://www.oracle.com/java/technologies/downloads/?er=221886>



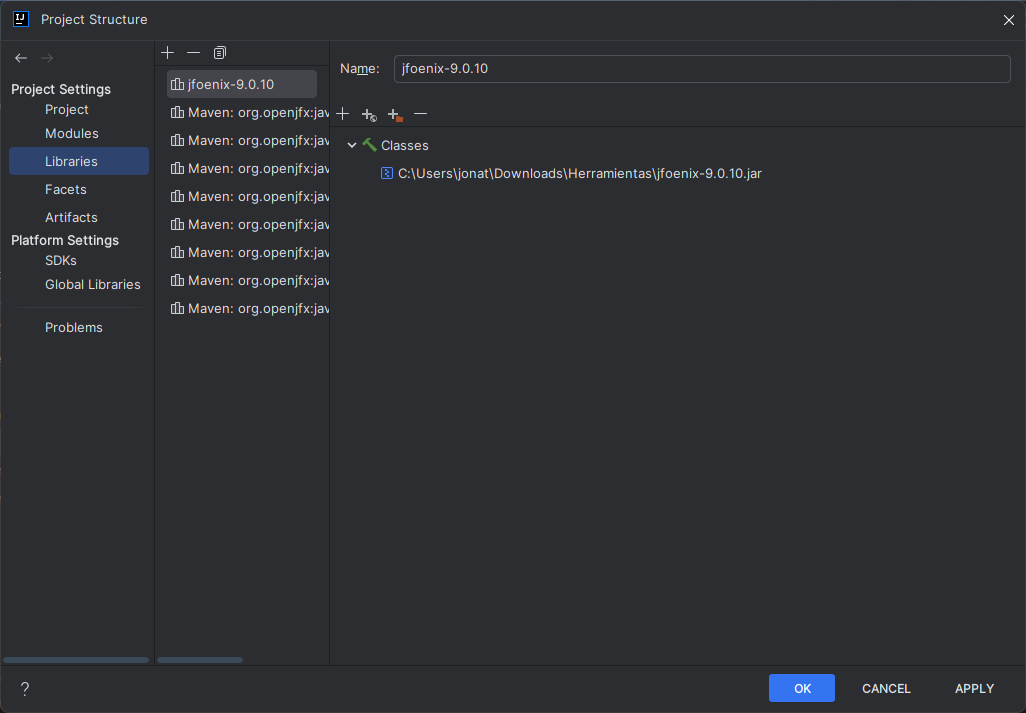
# **Descargar J FoeniX for Java 9**

<https://github.com/sshahine/JFoenix?tab=readme-ov-file>

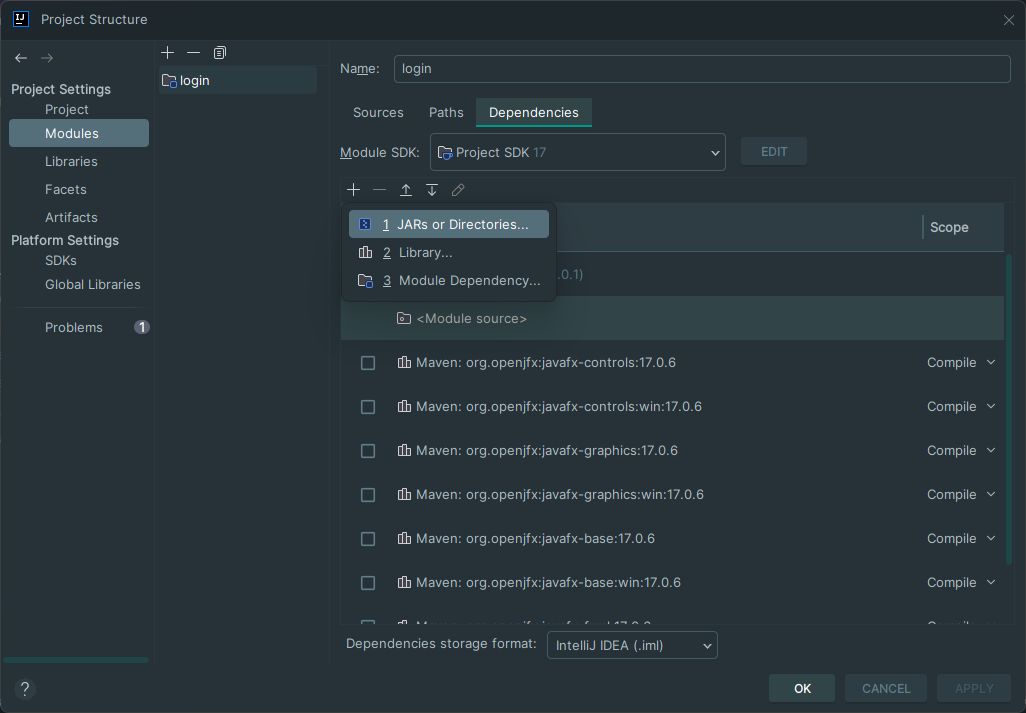


**Create new project in Intellij IDEA**

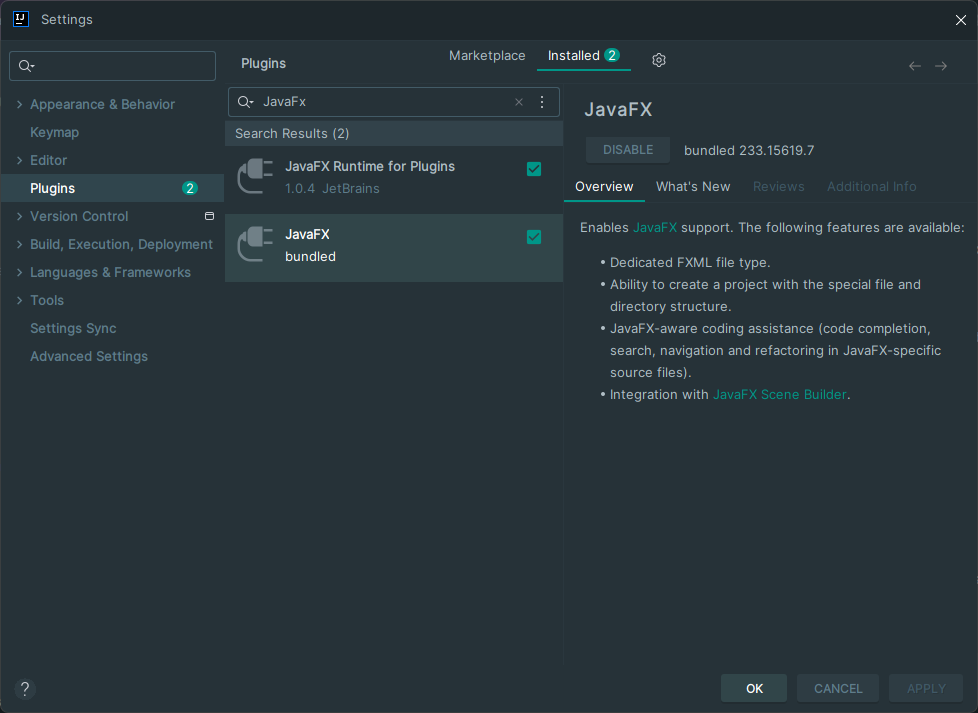
Configure environment to work with scene builderGo to File > Project Structure > Libraries > + > Java > Find the JFoenix .jar file and add it.



Go to File > Project Structure > Modules > + > Directory JARs > Browse for the JFoenix .jar file and add it.



Go to File > Configuration > Plugin > look under installed plugins to see if the Java Fx plugin is installed and enabled



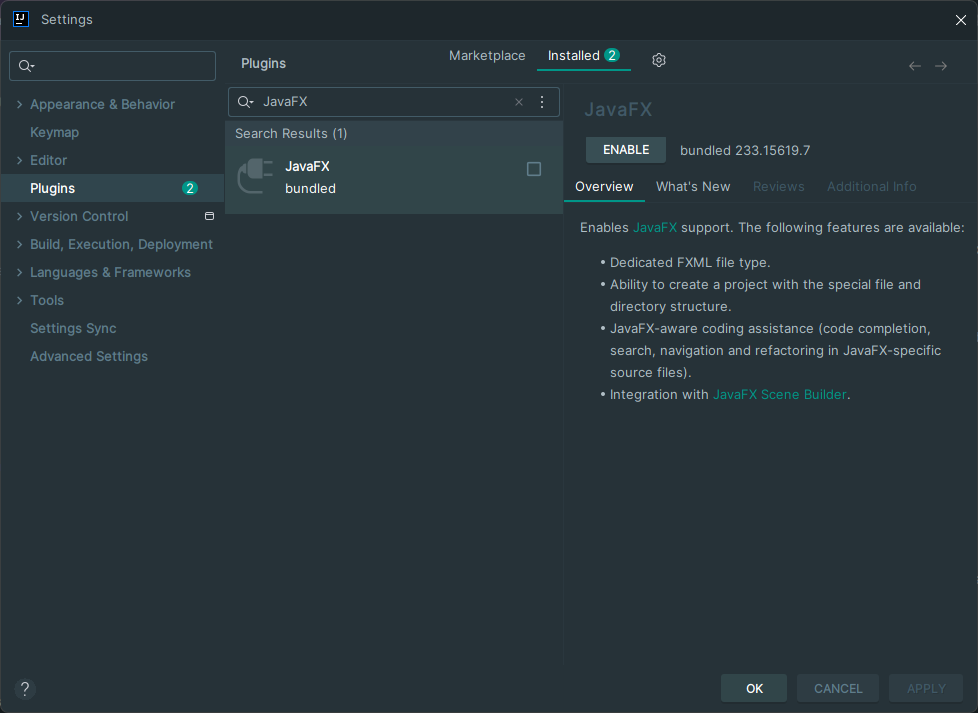
1

3

4

2

If not enable: Go to File > Configuration > Plugin > installed > Search for JavaFx and enable it



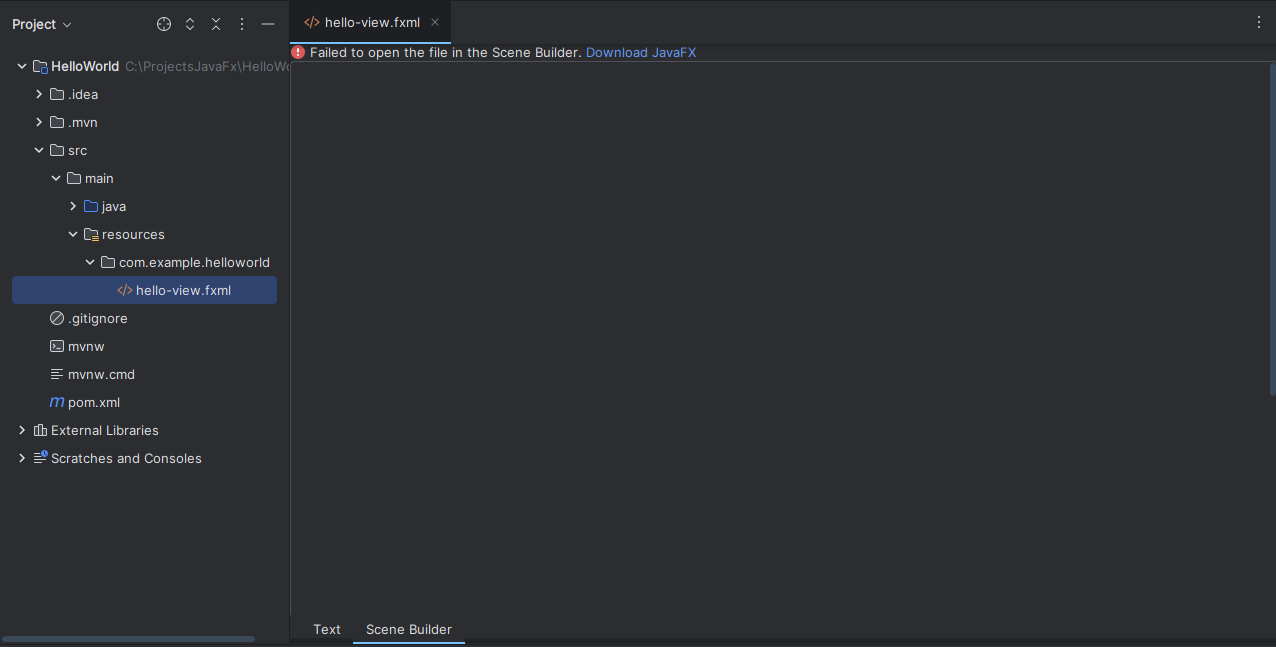
4

3

2

1

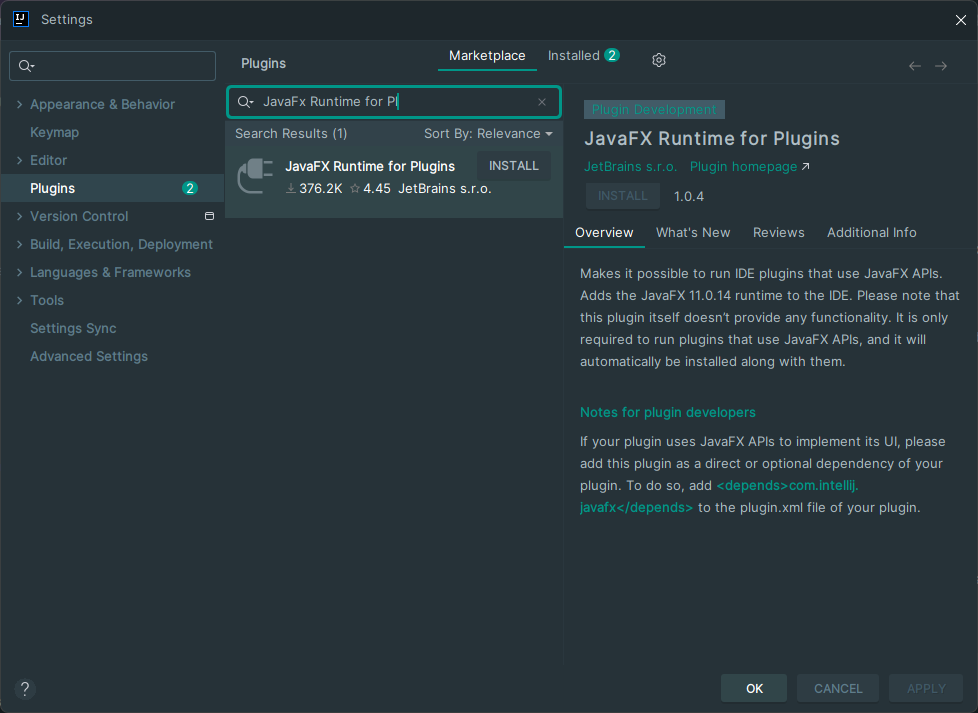
Open .fxml file go to src/main/resources/hello-view.fxml, at the bottom you will see scene builder when you change it you will get this error



2

1

Go to File > Settings > Plugin > download JavaFX Runtime for Plugins -> restart IDE



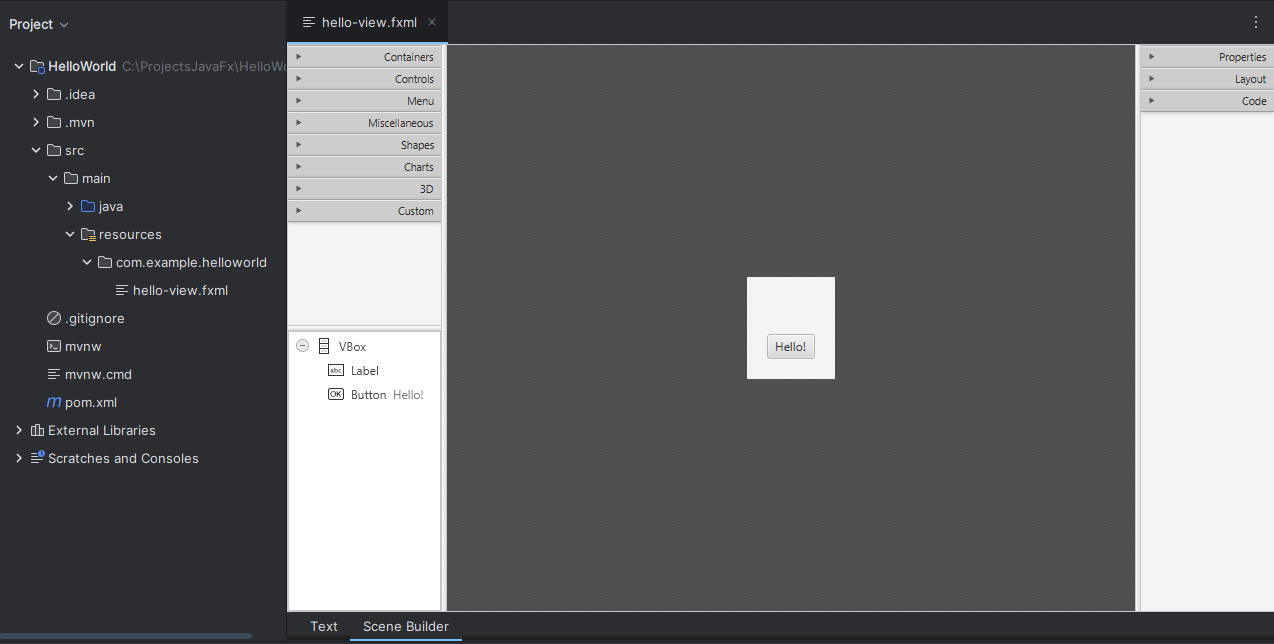
4

3

2

1

Then the error will say download scene builder kit for JavaFx, install it and you are done.



# **Scene builder contains:**

**Left:**

**Containers:** Containers used to organize other components, such as VBox, HBox, GridPane, etc.

**Controls:** User interface controls such as buttons, labels, text fields, etc.

**Menu:** Menu-related components such as MenuBar and MenuItem.

**Miscellaneous:** Contains miscellaneous components that do not fall into other categories.

**Shapes:** Geometric figures such as rectangles, circles, lines, among others, to add shapes to the interface.

**Charts:** Charts such as bar charts or pie charts.

**3D:** Components to add 3D objects.

**Custom**: JFoeniX components.

At the bottom the components that are in use.

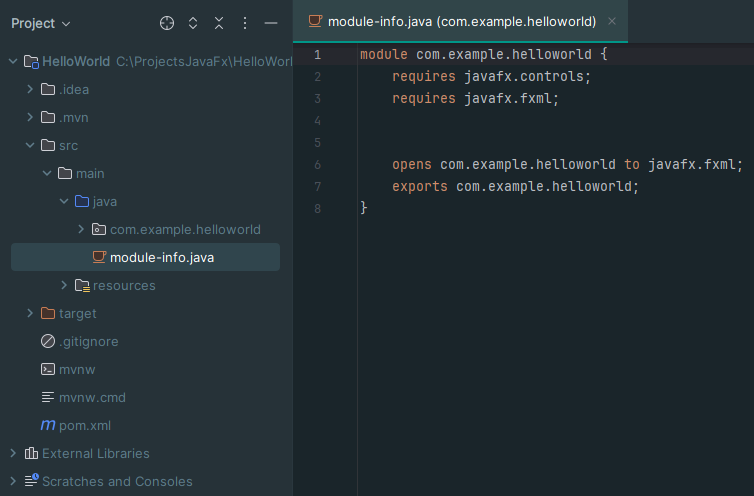
**Derecha**

**Properties: gives direct access to the essential settings of each component, allowing you to quickly customize its behavior and appearance without the need to write additional code.**

**Layout: Allows you to modify the layout and positioning of a component, such as margins, alignment and size.**

**Code: Here you can link the component with the Java code, configuring the fx:id, event handlers such as onAction and other code-related properties.**

# **Explicación archivo module-info.java**

****

module-info.java in JavaFX defines the dependencies and visibility of the project:

**requires:** Specifies the modules you need (such as javafx.controls for the UI and javafx.fxml for FXML files).

**opens:** Enables reflection access to the com.example.helloworld package (so that JavaFX can load controllers).

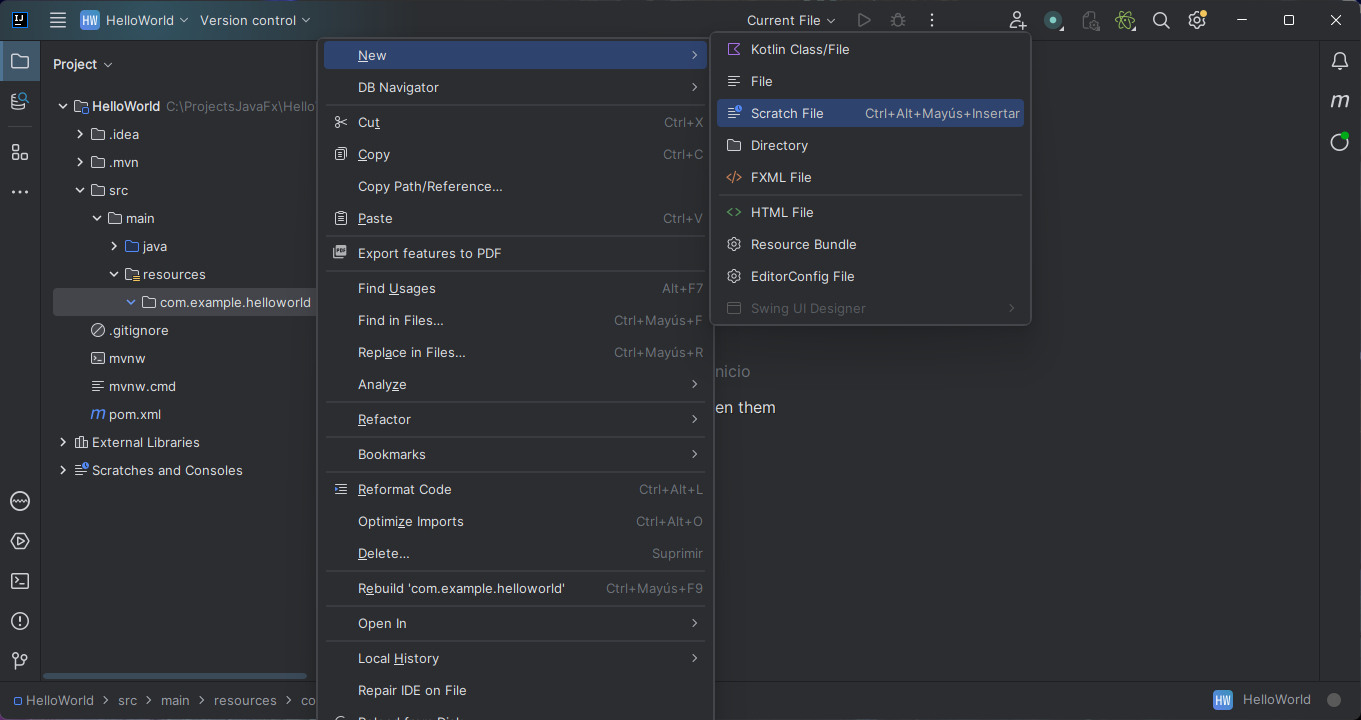
**exports:** Exposes the com.example.helloworld package so that other modules can use it.

# **Exercise 1: Hello World**

First Hello World in JavaFx

In the previously created exercise we will delete javaresource hello-view.fxml that was created automatically and we will start to do it from 0, to learn

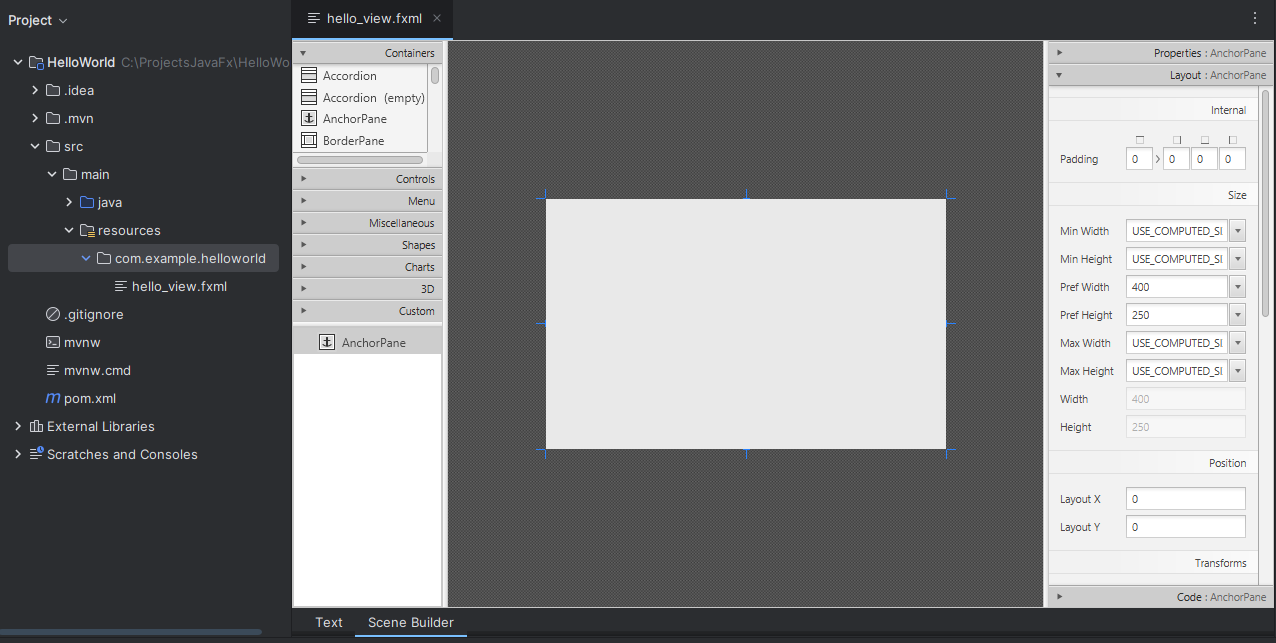
Create a new FXML file and name it hello\_view



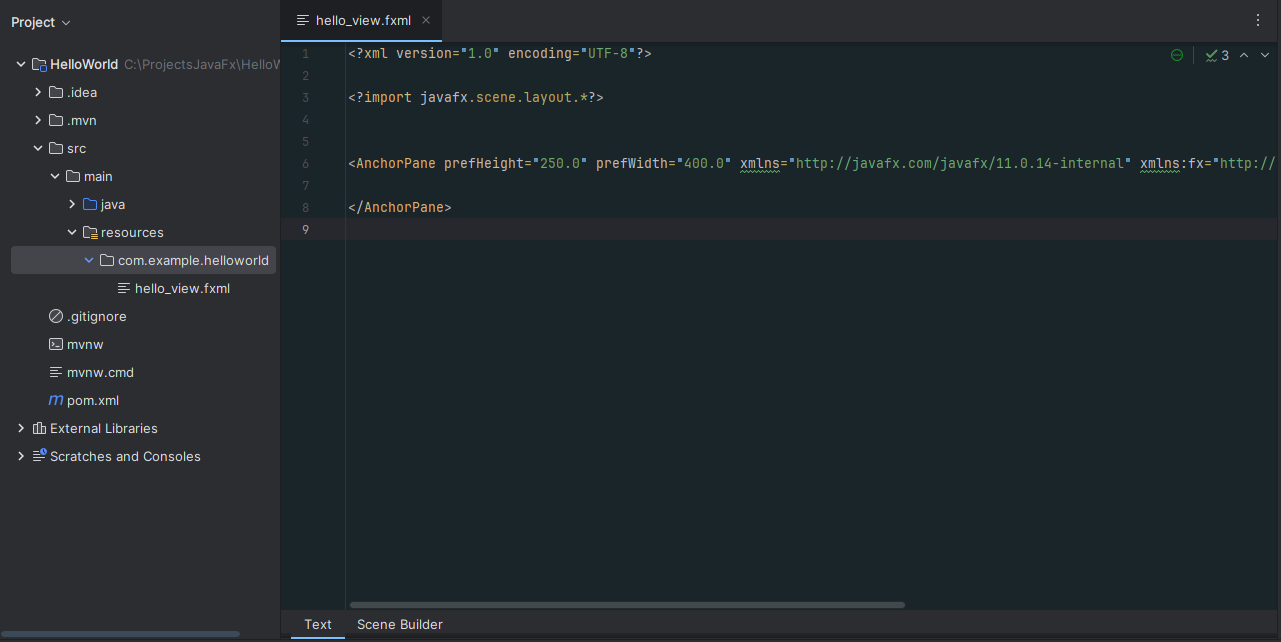
Remove from main “our package” HelloController

Open with scene builder the FXML file that we have just created.

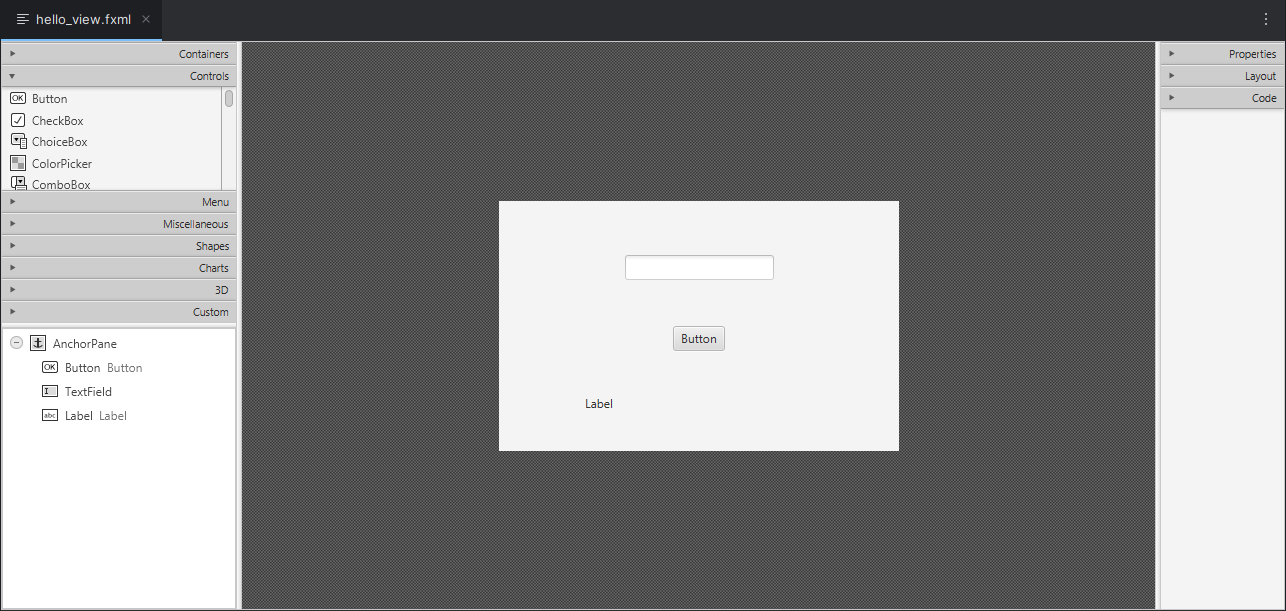
From the configurations of the right, we deploy Layout and we can configure the size of our AnchorPane, we will leave it 400 x 250



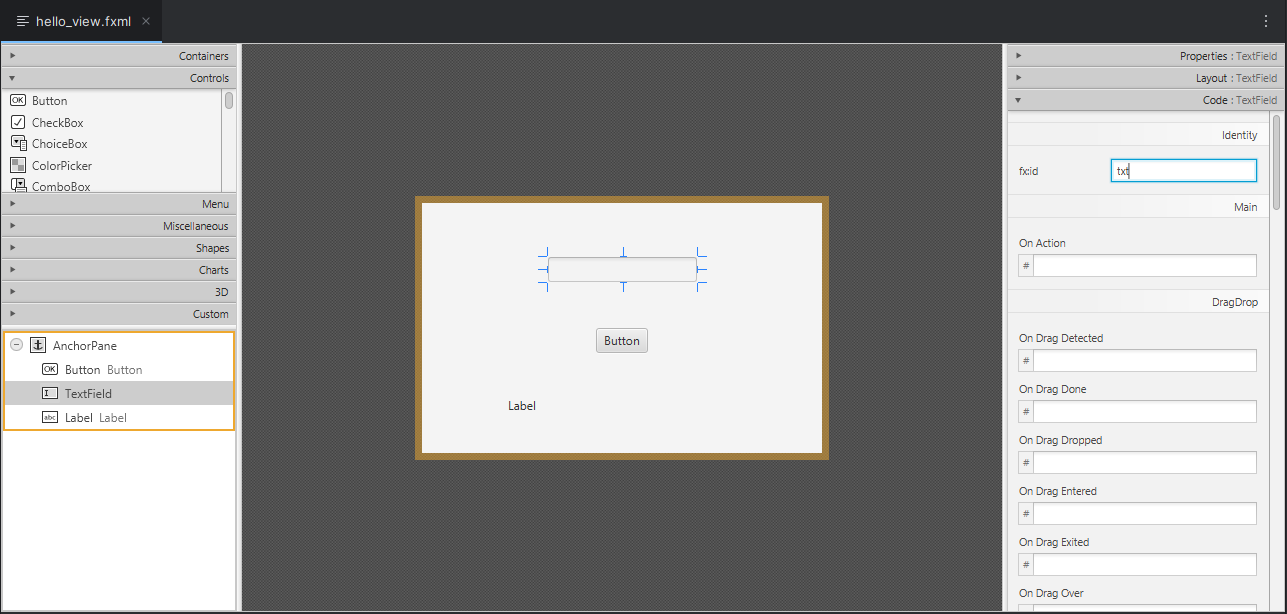
When saving in scene builder it is automatically updated in the intellij fxml.



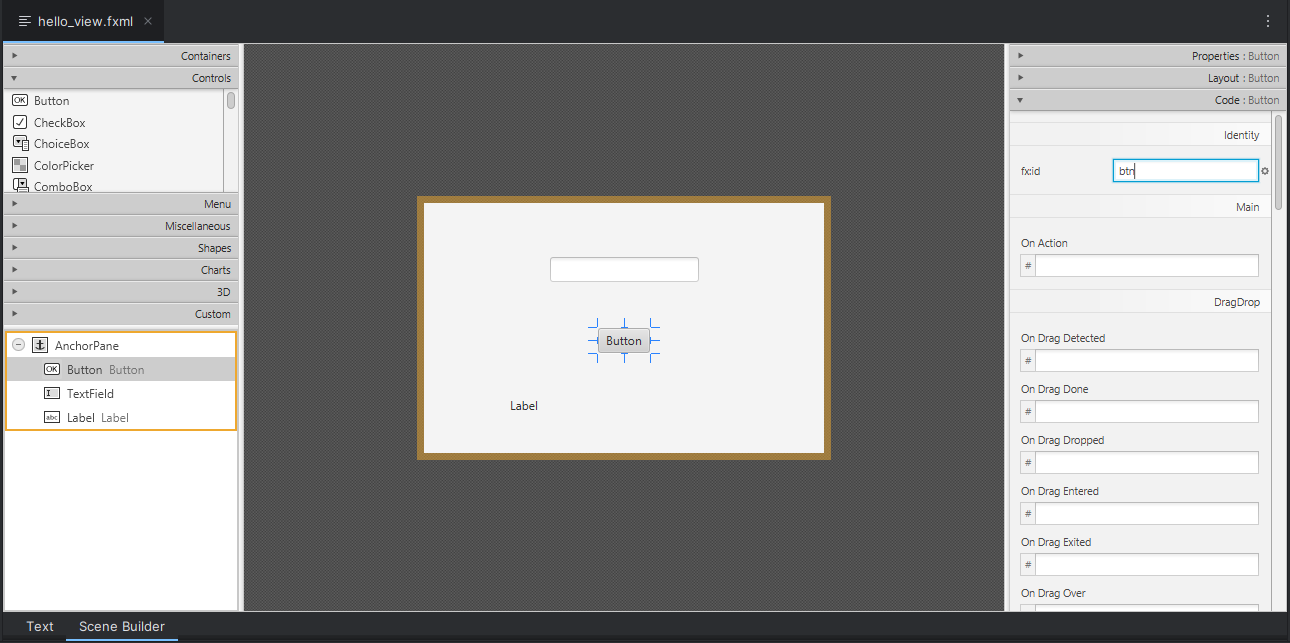
We will add 3 components, 1 button, 1 label and 1 txtField that are in controls



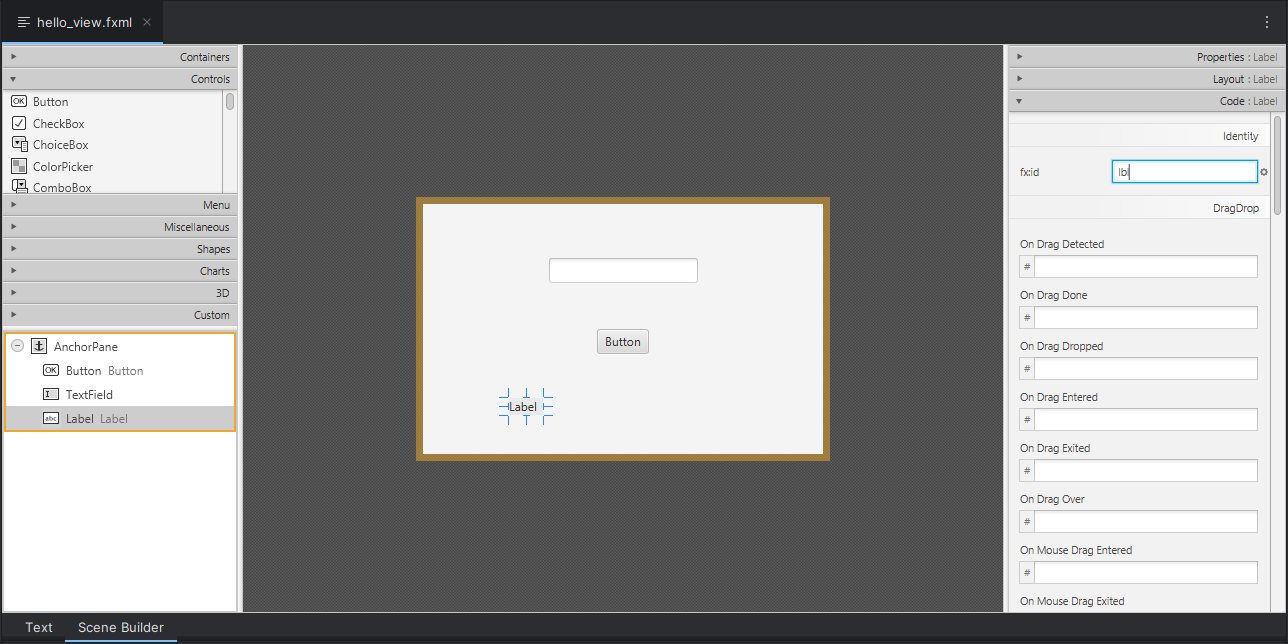
To each component we put an id, click on the component and then in the configurations on the right, in code we add the id

TxTfield

Button

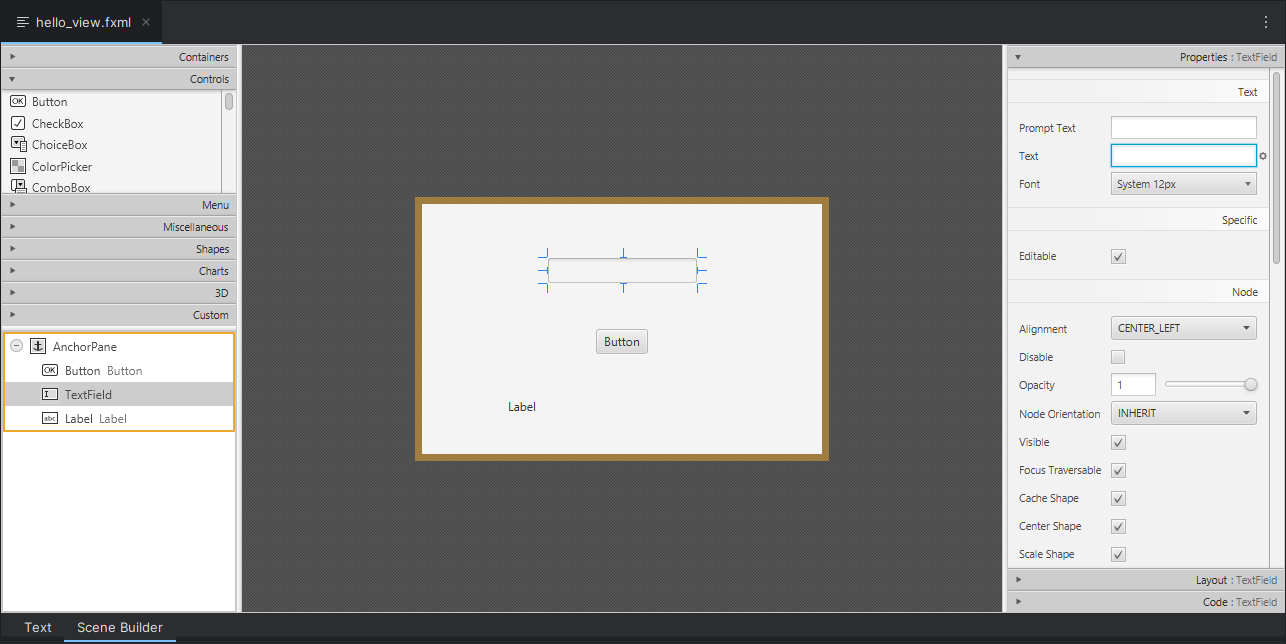


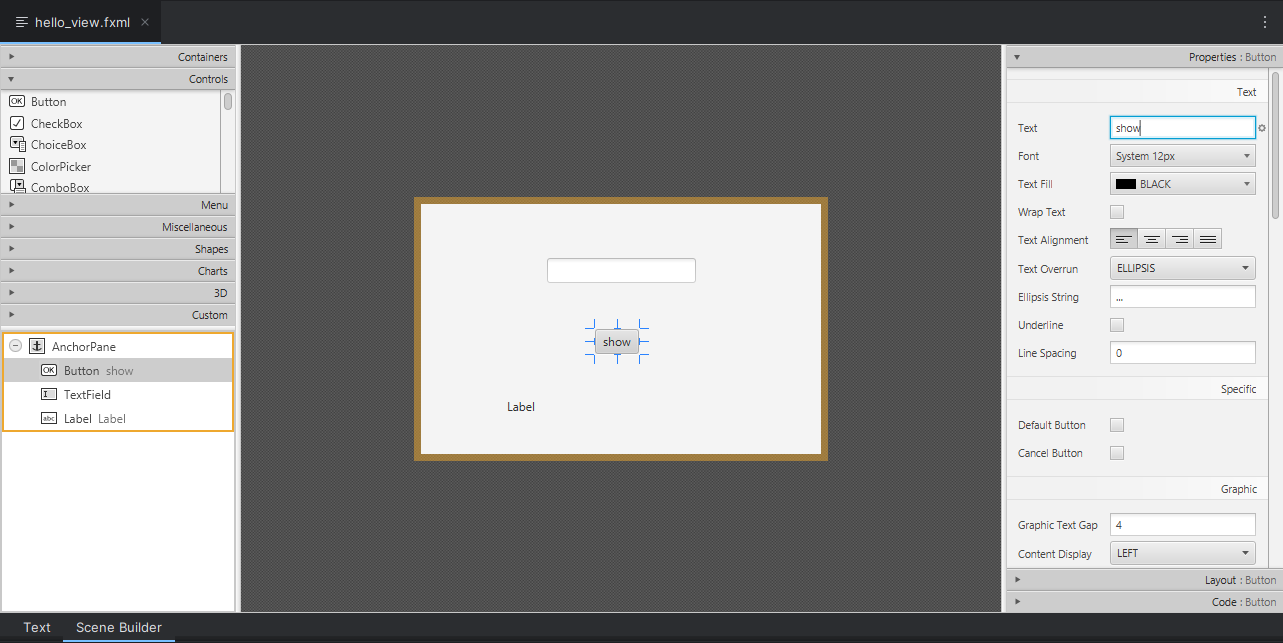
Label

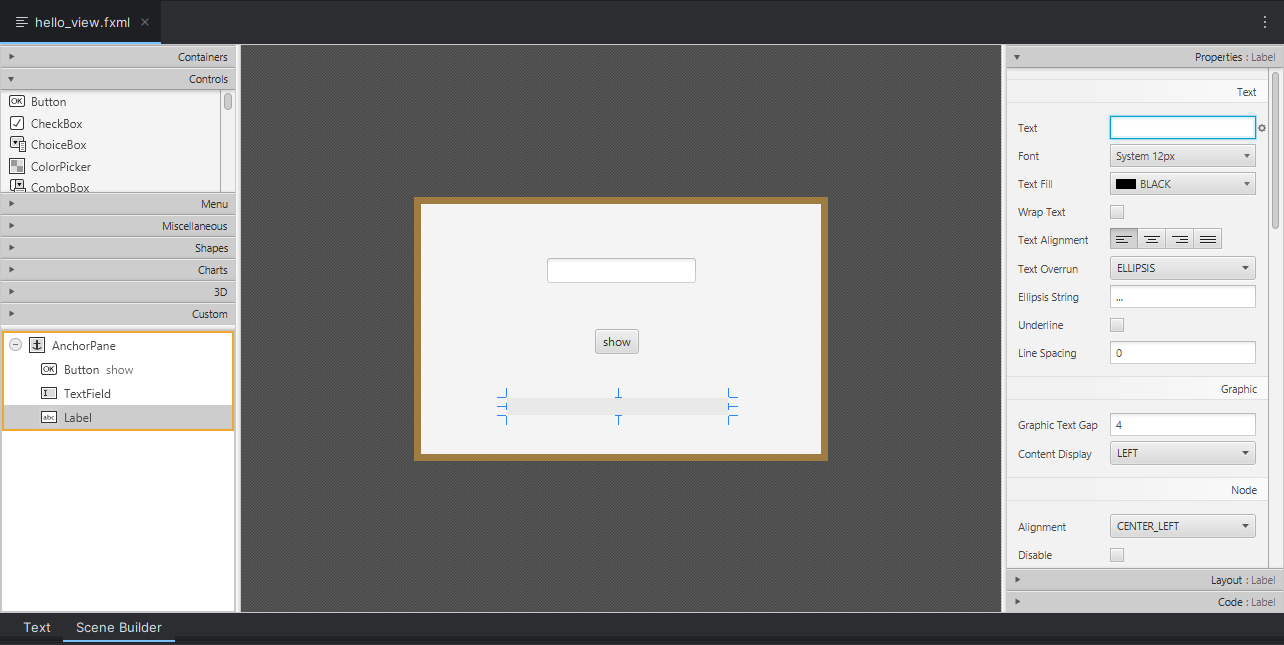


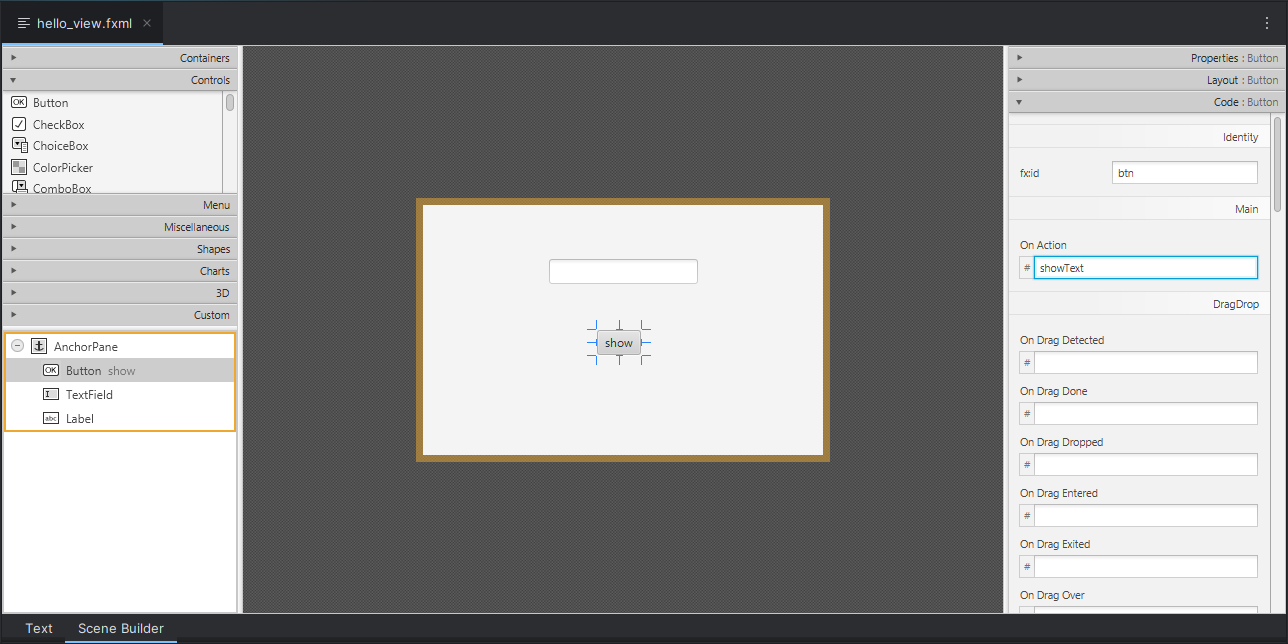
In properties we configure what we want to show initially

The TxtField is left empty.



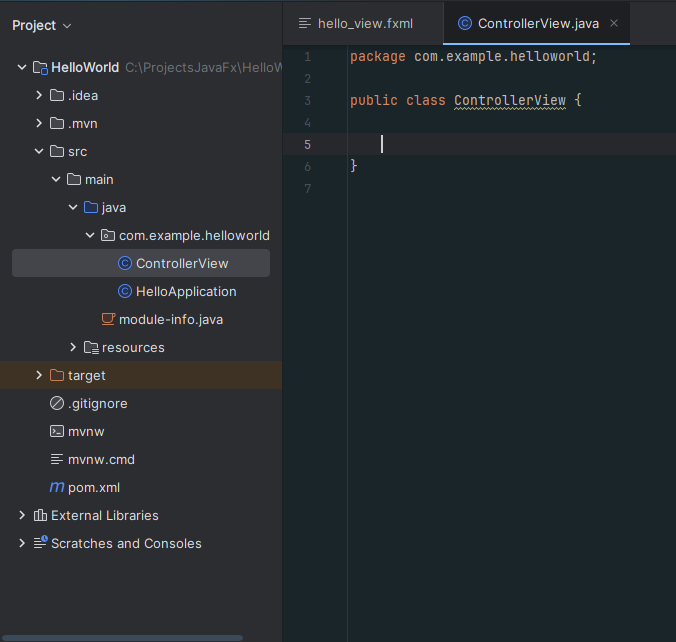
To the button we add show  


And the label is left empty

We add action to the button in the settings on the right in code > On Action, to tell it what function to call when clicked.  


We finish the graphical part, now we will finish the configuration, following the MVC

We create a class called ControllerView in mainjava.example.helloworld



# **Explanation of the controller**

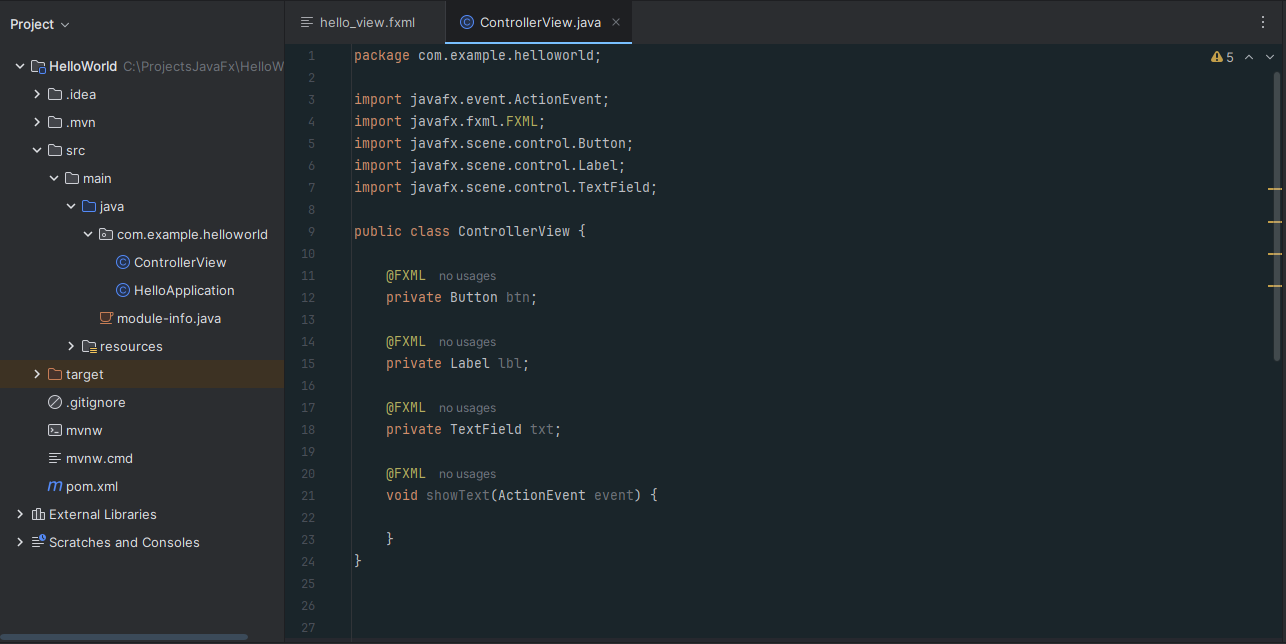
It is the bridge between the visual design described in the .fxml file and the Java application logic.

## **@FXML:**

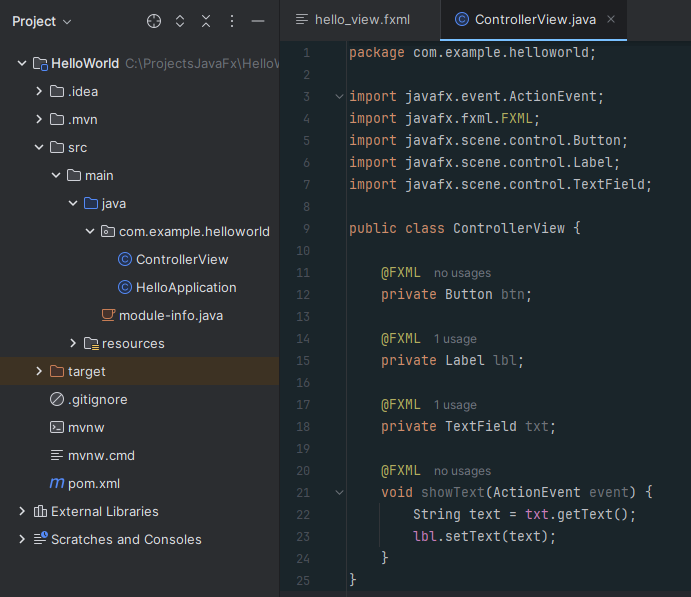
This annotation indicates that the field or method is accessible from the FXML file. The controller and the FXML file are linked through this annotation.

In attributes: When an attribute is annotated as @FXML, it means that this field refers to a GUI component that is declared in the FXML file, in methods it indicates that this method can be called directly from events defined in the FXML file, such as a button click.

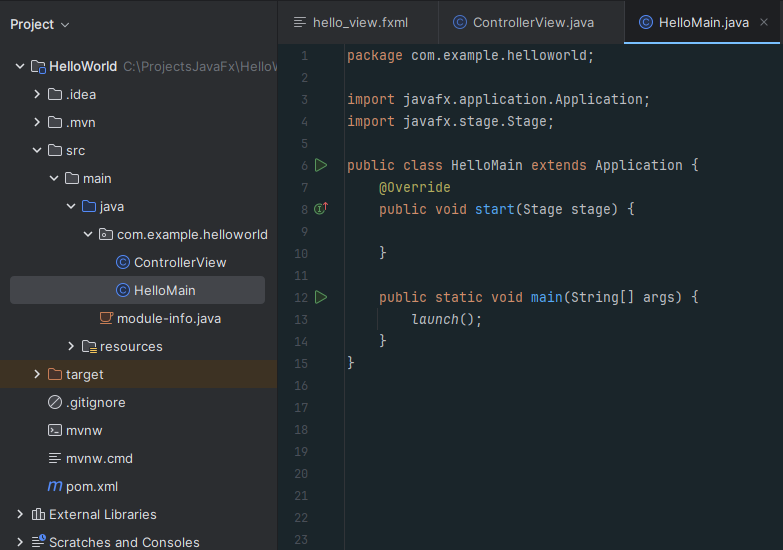
**Controller configuration**



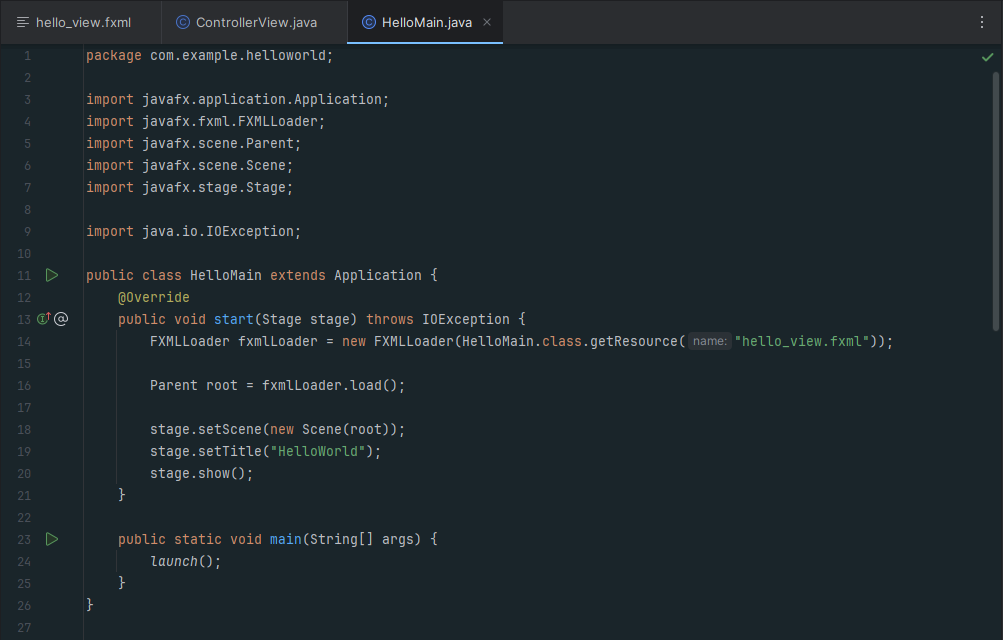
In showText we do the logic of what the controller has to do, and it is very simple, it receives what the user sends in the text, and passes it to the label.



Now we create our main class in main.example.helloworld, we extend Application meaning it is a JavaFX application, we implement the start method and add the main method

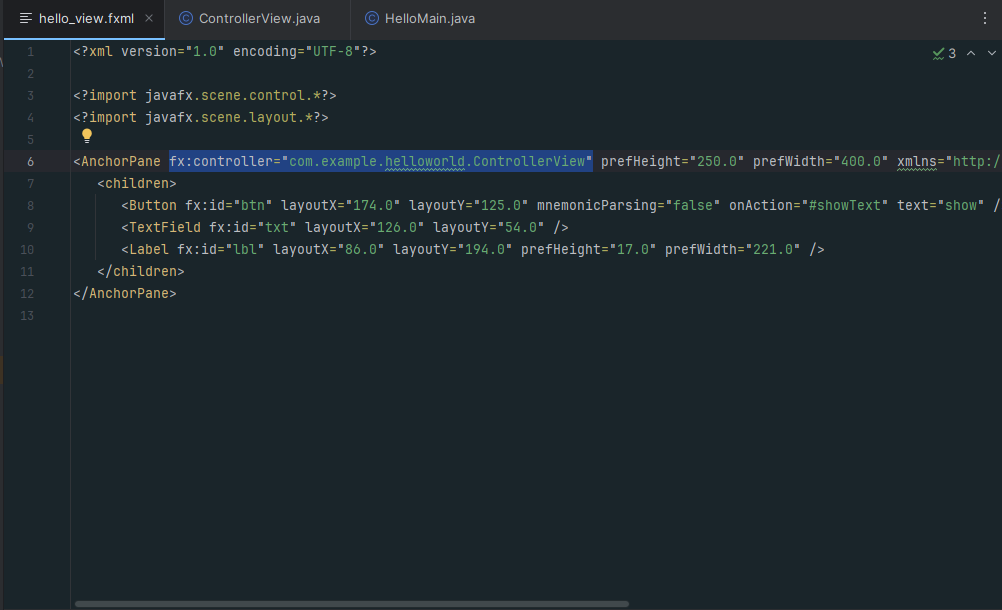


With the code

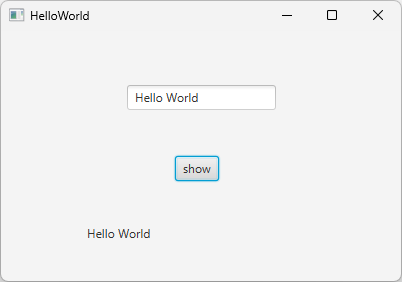


* **Start method:** This is the most important method in a JavaFX application. It is where we configure what will appear in the window.
* **Stage:** This is the main window of the application.
* **FXMLLoader:** We load a hello\_view.fxml file, which defines the graphical interface.
* **Parent root:** Here the content of the interface that is defined in the FXML file is saved.
* **Scene:** A scene is the visual content of the window.
* **stage.show():** Shows the window on the screen. Without this, the window would not appear.
* **stage.setTitle():** Add name to the window.
* **Main method:** This method starts the application. It calls launch(), which is a special JavaFX method that starts the whole life cycle of the application.

To finish linking the view with the controller we update the following code in the .fxml file



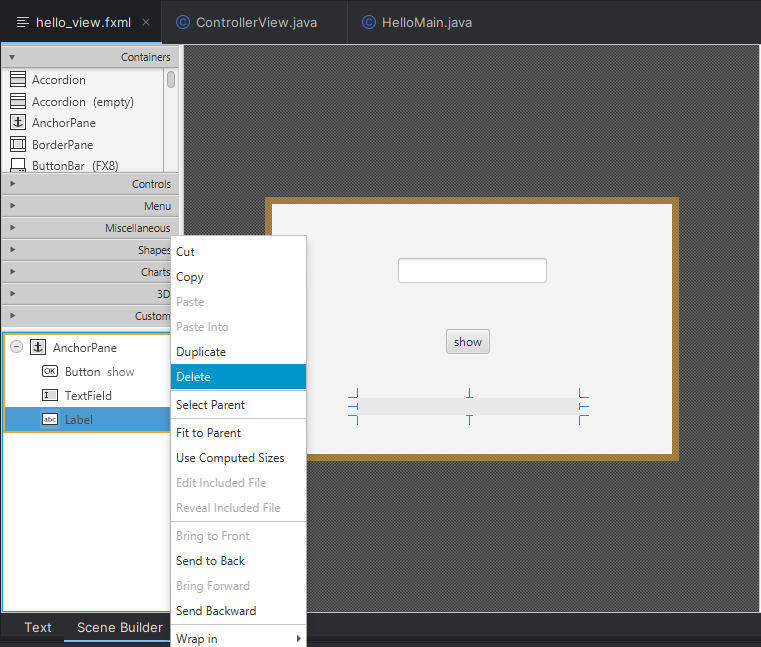
We place the name of the controller that we created

And when we run it, it would look like this  


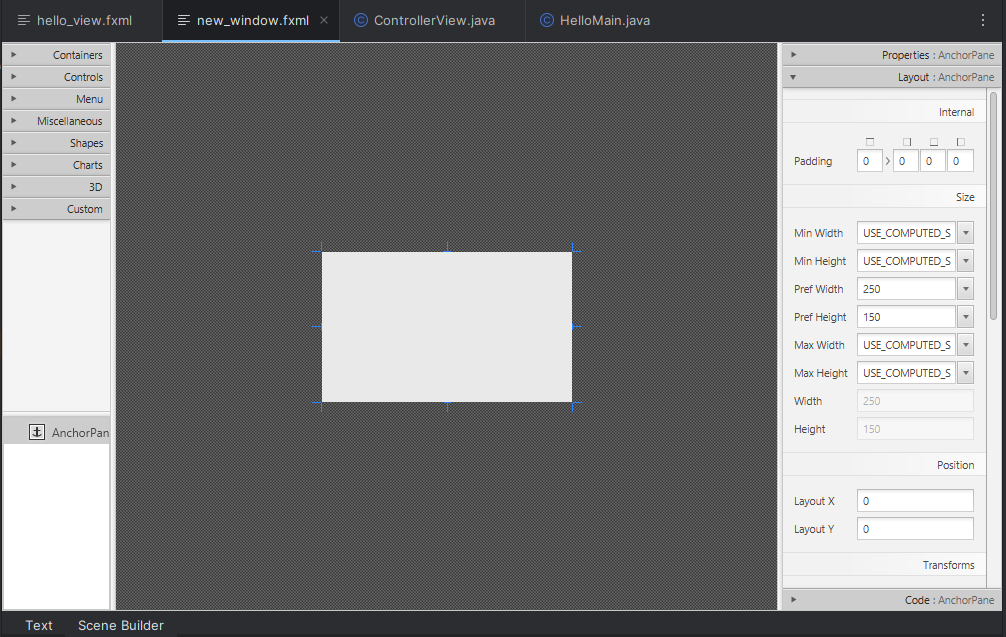
Adding more functionalities

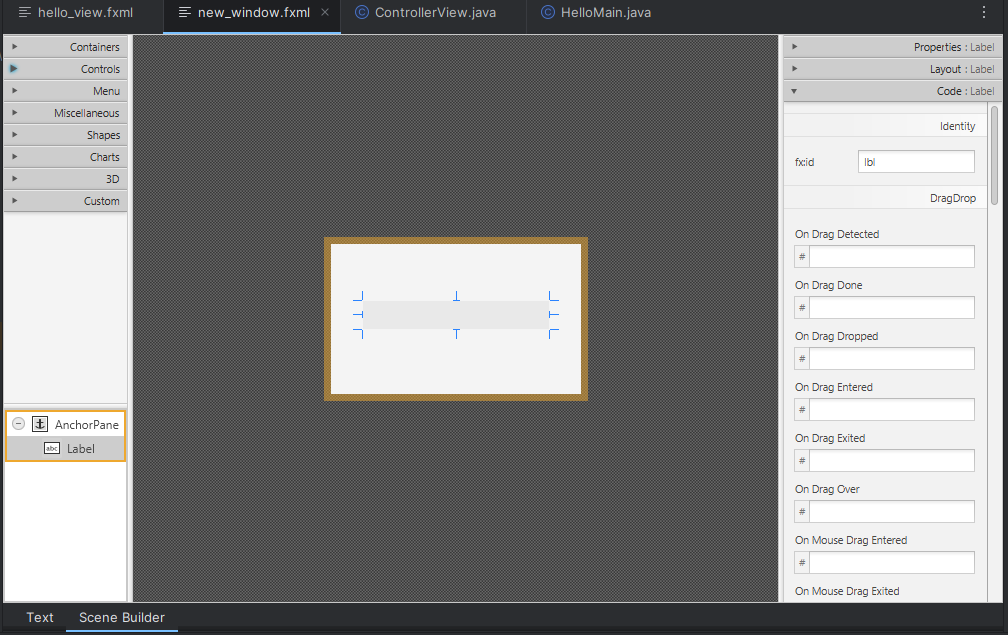
New window containing the Hello World

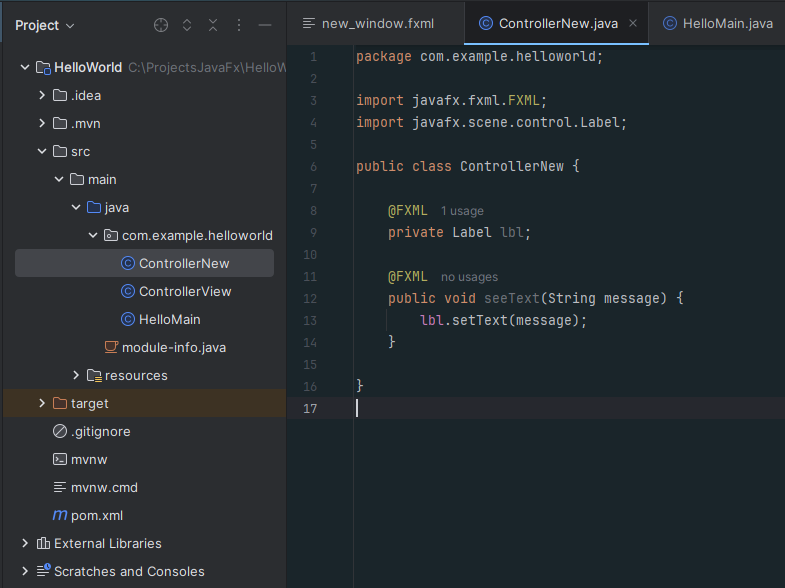
Remove the label tag

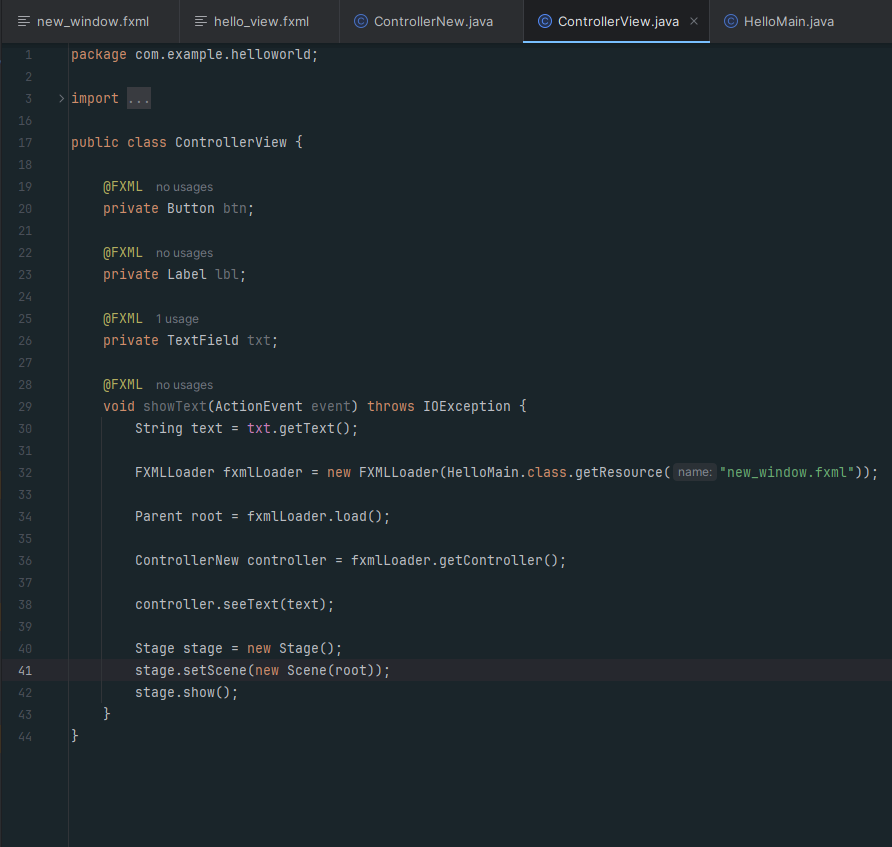


Create new .fxml file -> name it new\_widow, open it with scene builder and adjust the size

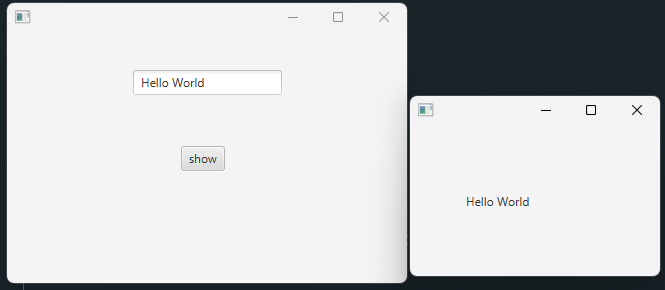


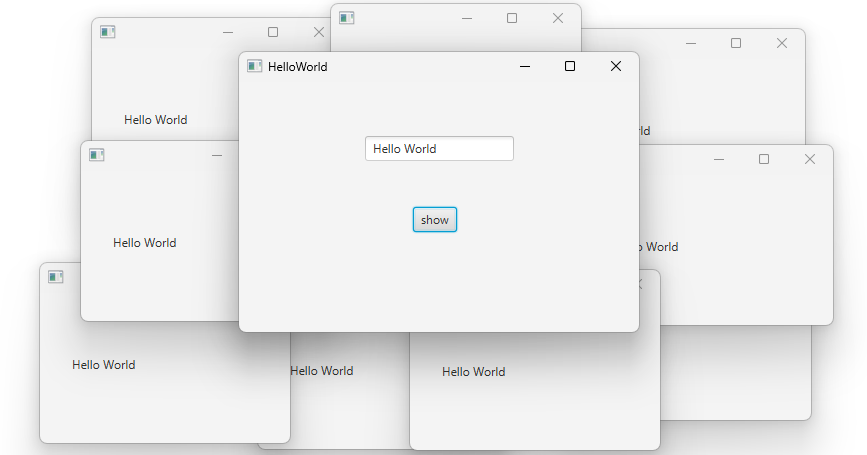
Add a new label and remove the default text and add the id  


Create a new controller class called ControllerNew, add components with its annotation, add a new function called seeText and pass the message to it. Remember to update the name in the new\_window.fxml file.  


Modify the first controller called ControllerView, delete the line that we pass the text and add the code we have in the start method, update the path to the new arxhivo.fxml, create a stage object that we import from javafx.stage.Stage; 

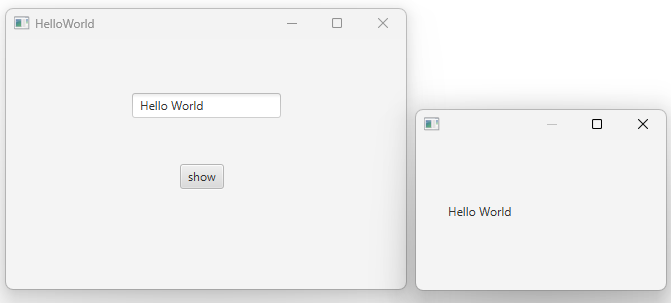
* The button defined in the FXML file, associated with an ID of 'btn'. When the user clicks on this button, the 'showText' method is triggered.
* Label that can display some text in the user interface. It is linked to an ID of 'lbl' in the FXML file.
* TextField where the user can enter text. The value of this field is obtained in the 'showText' method.
* This method is activated when the button is clicked. Its function is to open a new window. It first gets the text from the text field 'txt', then loads a new FXML file named “new\_window.fxml”, and finally, displays that new window.
* The action event that occurs when the button is clicked is passed as a parameter.
* If an error occurs when loading the FXML file, an IOException is thrown.
* The text entered by the user in the text field 'txt' is retrieved and stored in a variable.
* The new view is loaded from the FXML file “new\_window.fxml”.
* The FXML file is loaded, and the corresponding interface structure is created.
* The controller of the new window is obtained so that the text can be passed to it.
* The text from the text field 'txt' is passed to the controller of the new window.
* A new window (Stage) is created to display the new scene.
* The new scene loaded from the FXML file is set in the new window.
* Finally, the new window is displayed on the screen.

When running it, it already shows the windows passing the text  


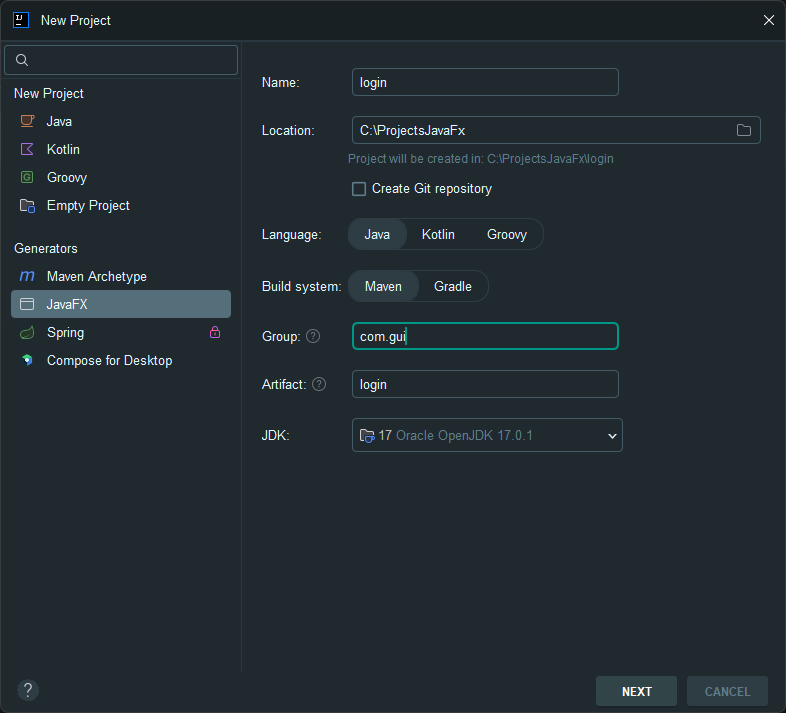
Fix that if we press the show button too many times it creates multiple Windows  


Enable modal window, so that when you create a new window, the main window is disabled.

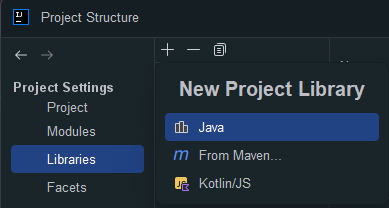
What was added serves to set the window as modal and sets the main window as its owner, blocking interaction with the main window until it is closed.

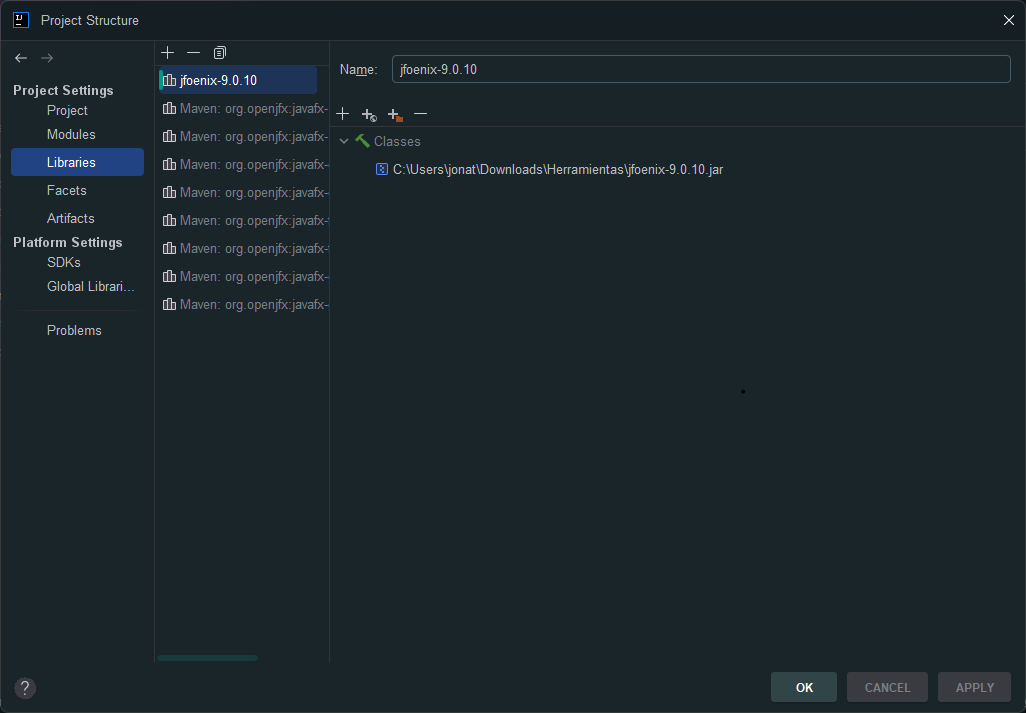
It no longer lets you interact with the main window

# **Exercise 2: Login**

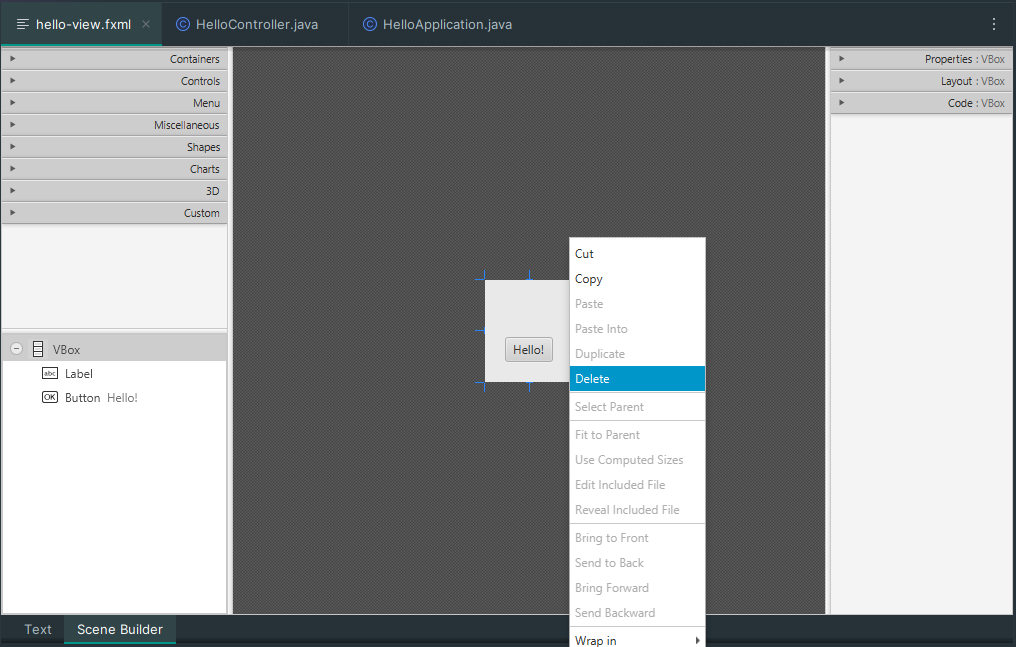
Create Project  


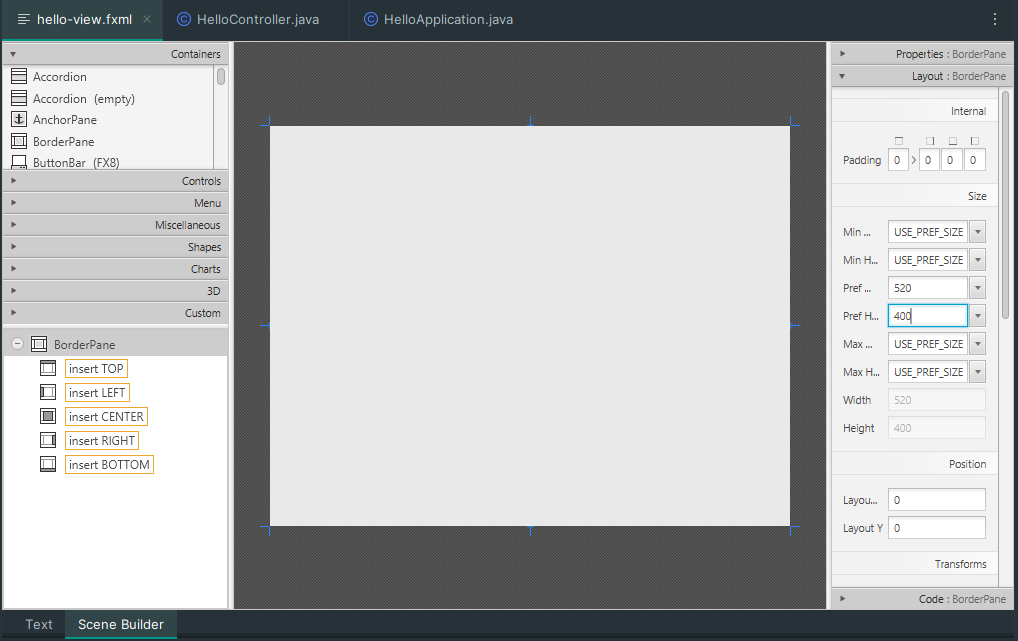
Once created, go to File > Project Structure > Libraries -> + sign and add the J Foenix .jar.

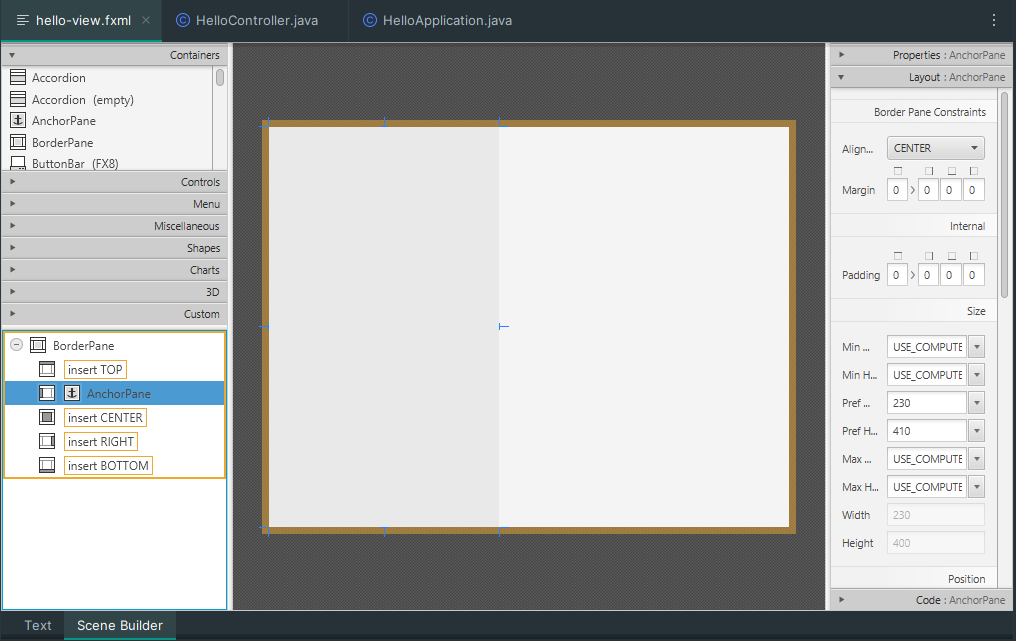


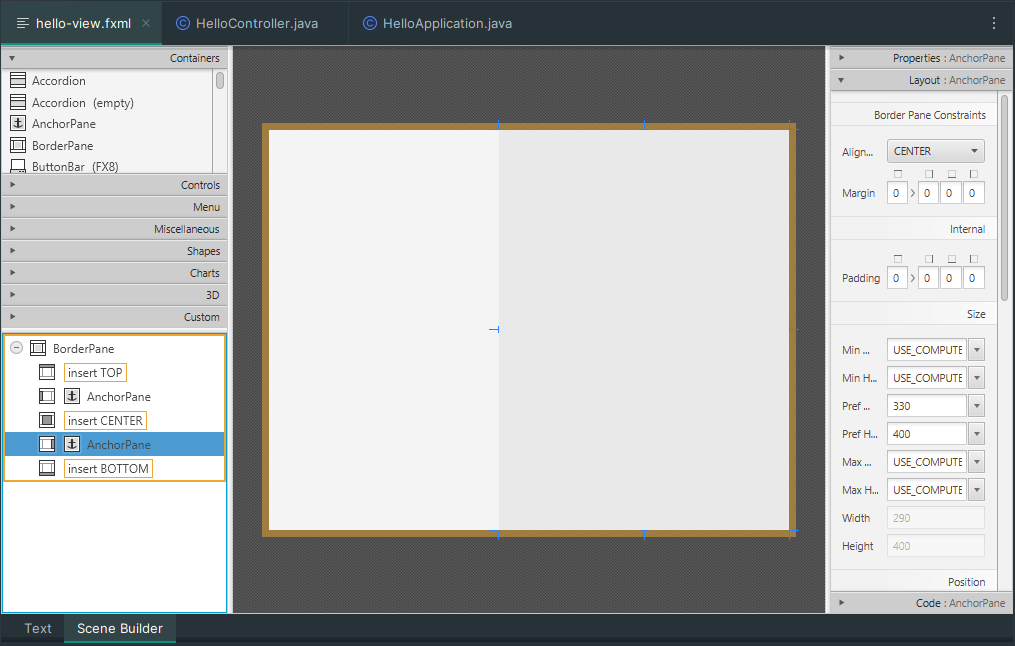


Open the hello-view.fxml file with scene builder and delete the AnchorPane it contains

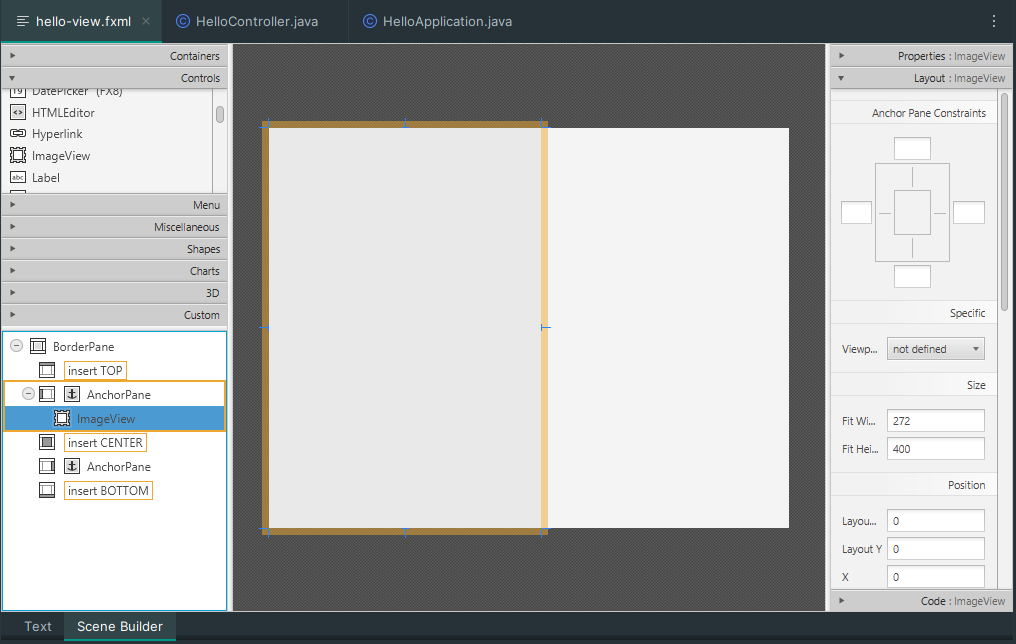


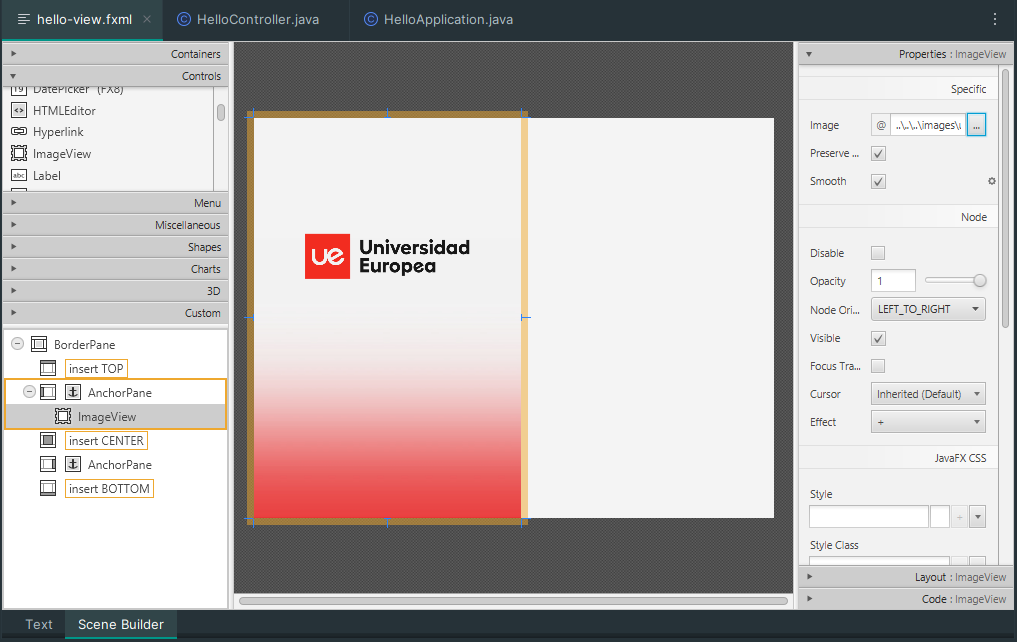
Add a BorderPane and configure it 520x400  


Add AnchorPane by dragging on the left side to where it says insert LEFT and set it to 230x410

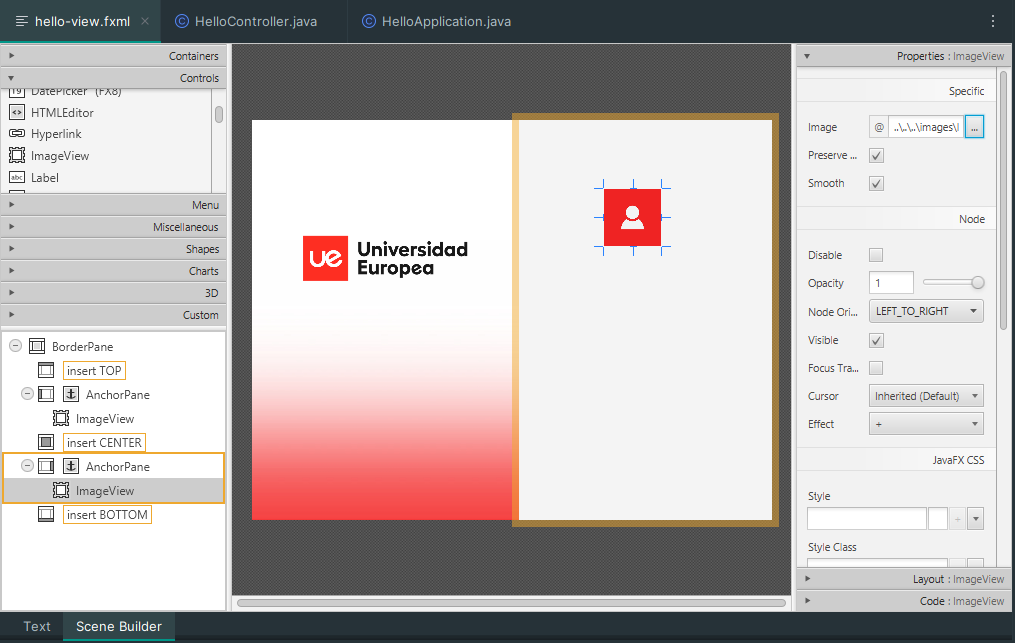
Add another AnchorPane by dragging to where it says insert RIGTH, set it to 330x400

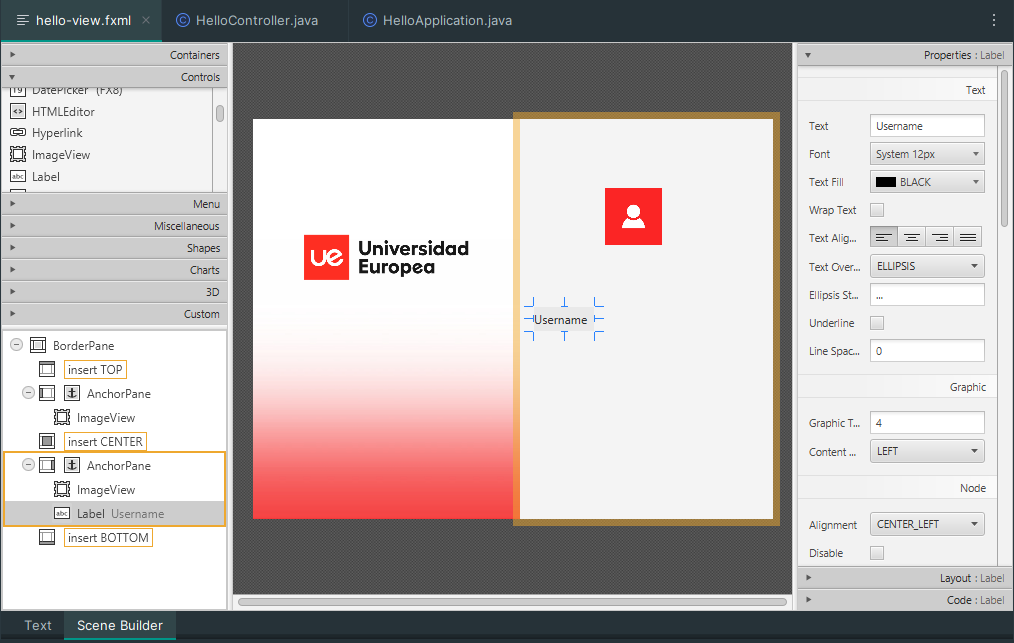
illn ImageView and set its size to 272x400



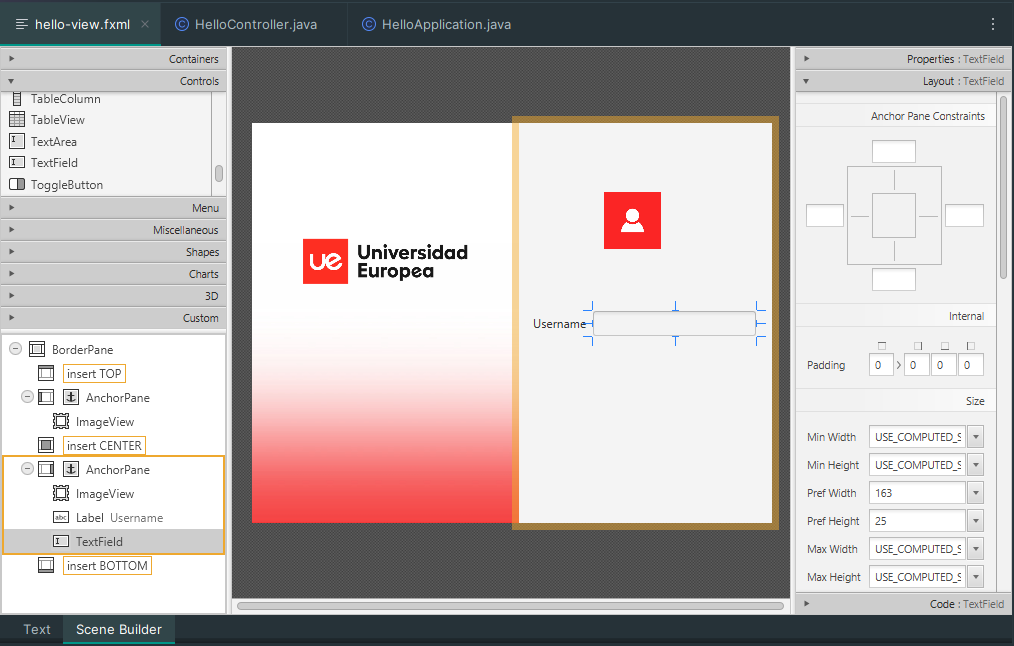
In properties > layout load image

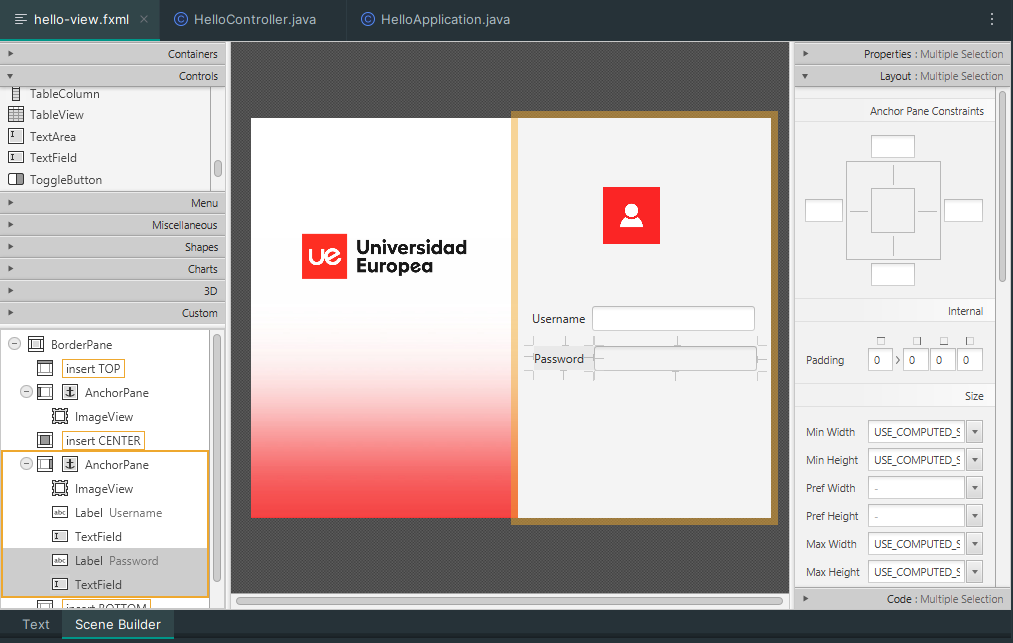
Add another ImageView with the size of 84x57 and load image from properties > layout > image



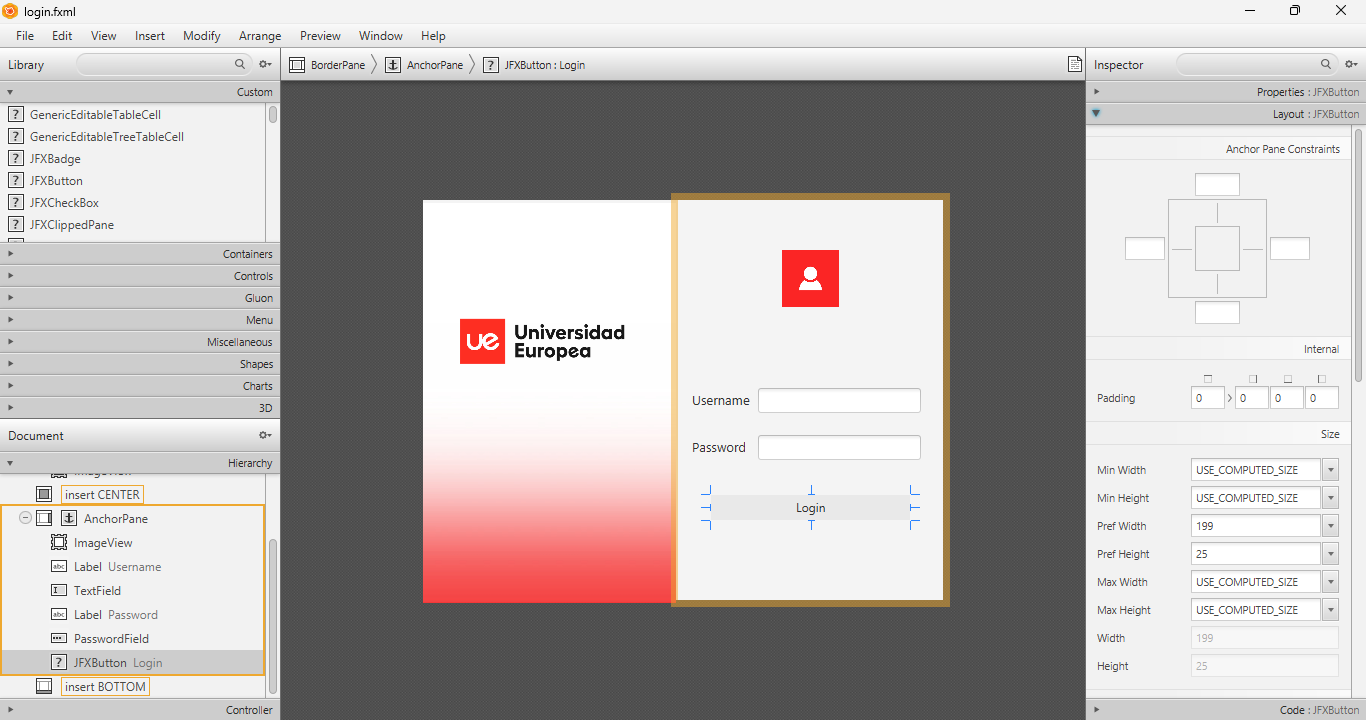
Add a new label and in properties > text = Username 

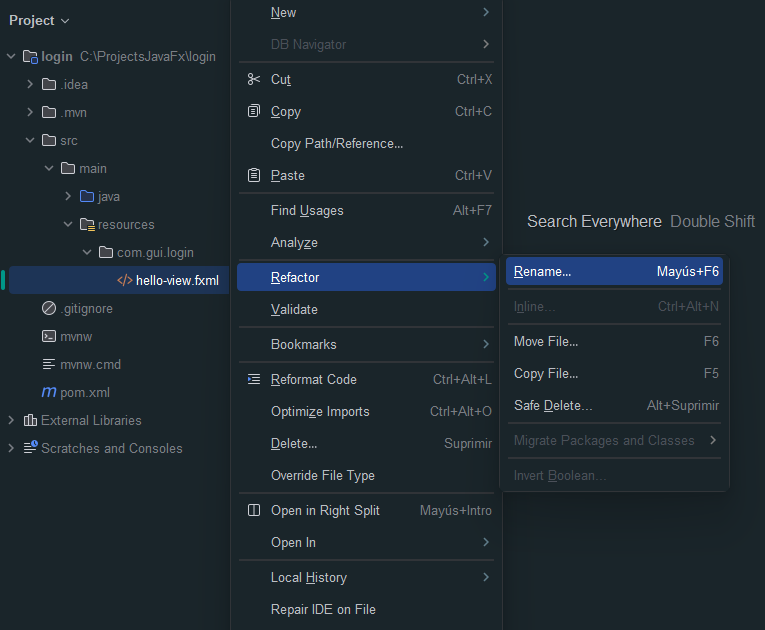
Add a textField and configure it 163x25

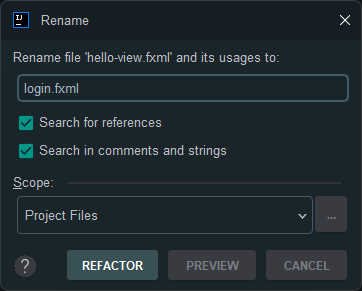


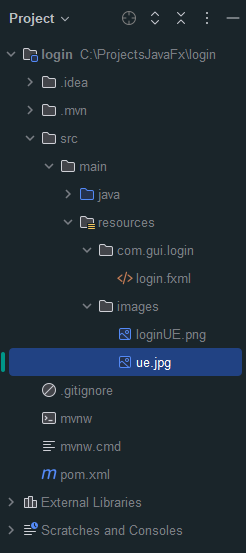
Add another label and a Password field with the same settings as above

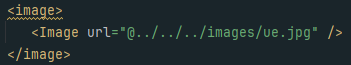
Add a JFXBUTTON, rename it in properties > text and set in Loyout 199x25

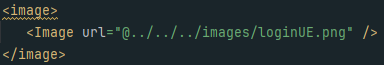


Rename the fxml file to login  


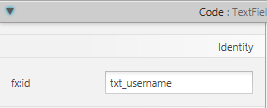


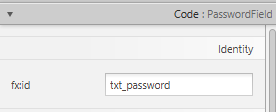
Create a new folder in resource called images and place the images in it.

Update image paths in login.fxml file  




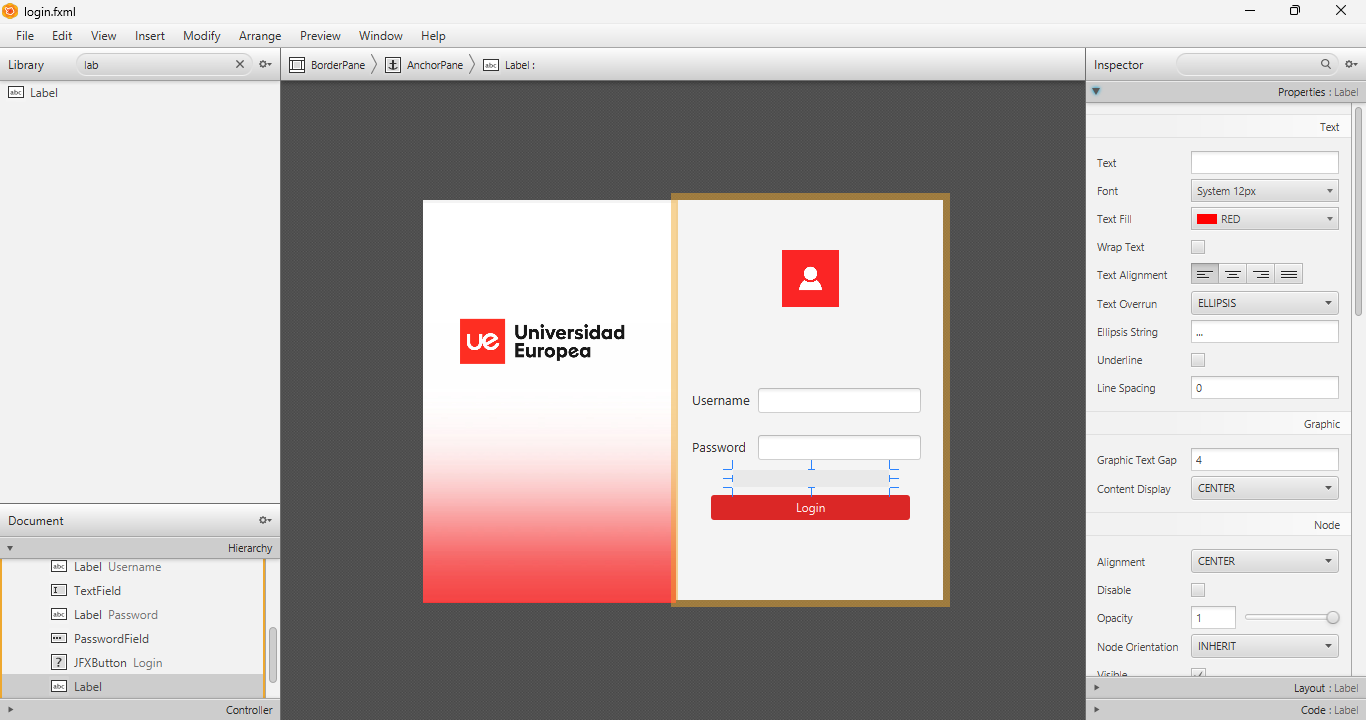
Add id to texts

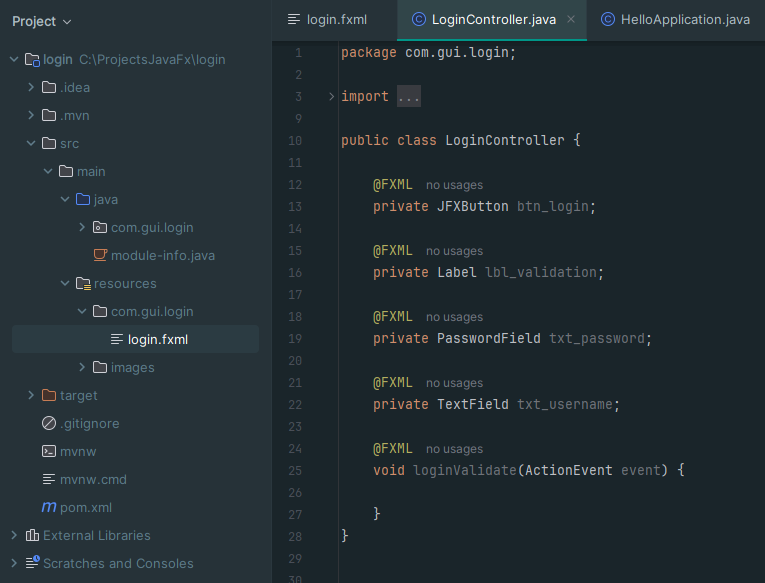


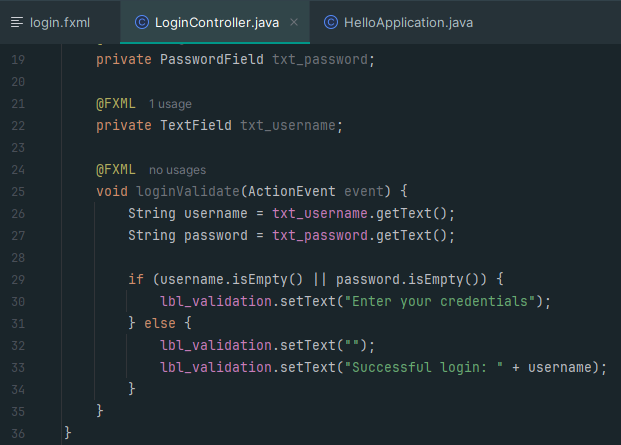


Al button add id and OnAction

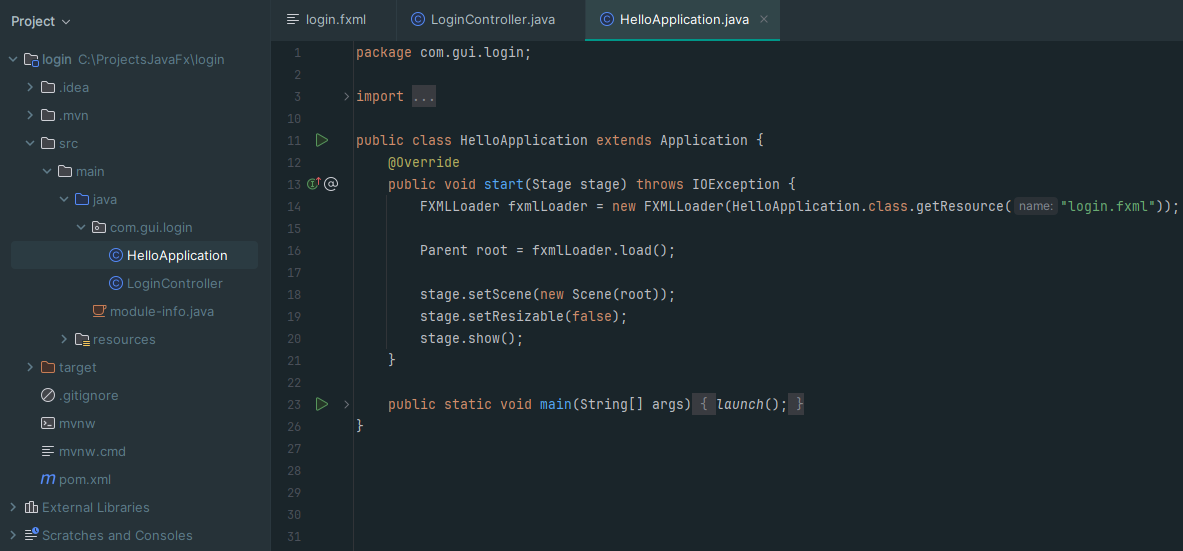


Add a label for validation that shows if the fields are empty, it ill show it in red  


Add LoginController

Adding logic in the controller to validate that no empty data is sent and assuming that you log on 

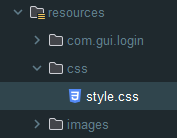
Logic so that the screen cannot be accommodated.

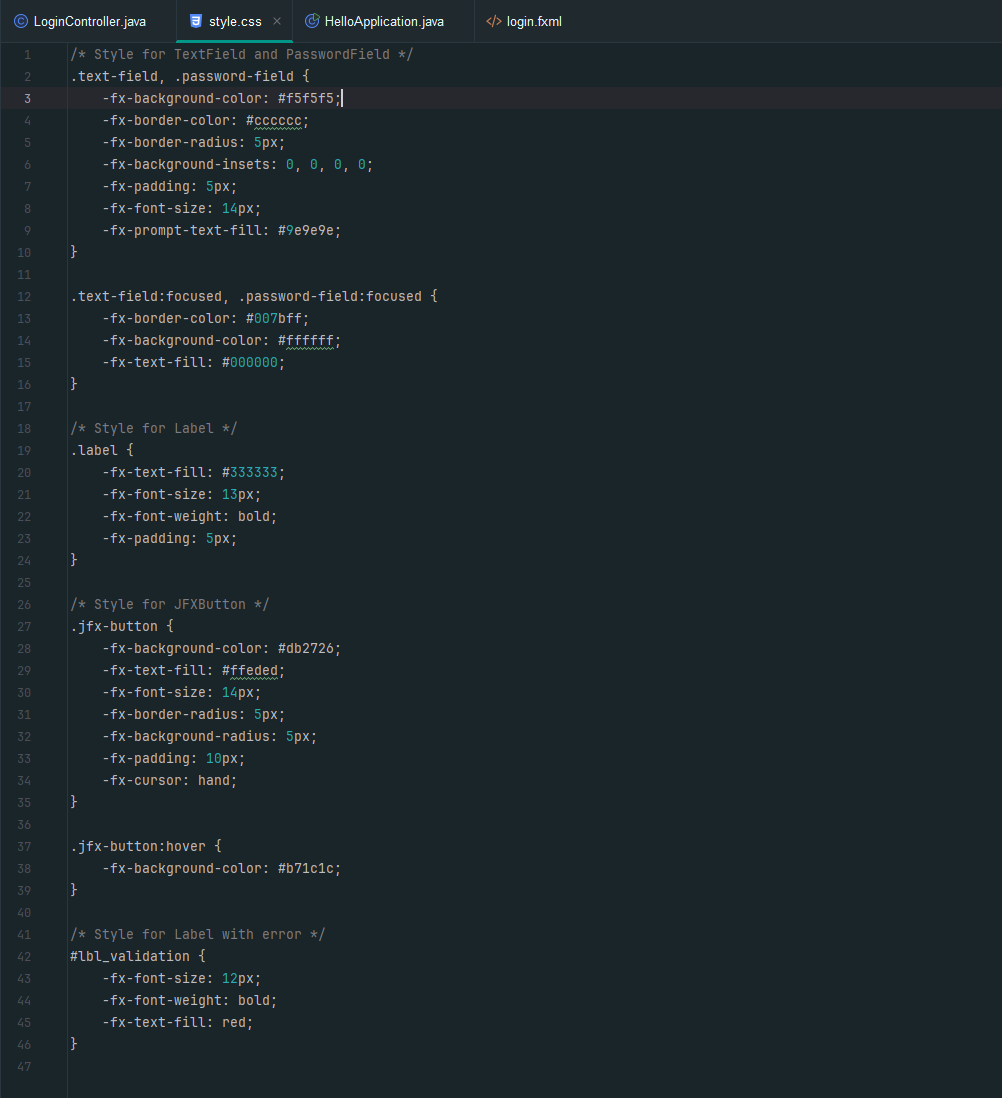


Using stage.setResizable(false); prevents the window from being resized by the user. By setting the value false, you disable the option to resize the window, which means that it ill remain at the dimensions you have defined without allowing the user to stretch or shrink it.

# **Apply css**

Create folder in resources called css, and add new file style.css



Link css file to fxml file  
css file

**TextField and PasswordField:** Set a light background color, a gray border, and a rounded border radius. Add internal padding so that the text does not touch the borders. Fields change color when in focus to highlight the blue border.

**Label:** Sets the text color to a dark gray, a font size of 13px and a bold font weight to highlight the labels.

**JFXButton:** Customize the button with a red background color, white text, and a rounded border. The button changes color to a darker red when the mouse is hovered over it.

**Label with error:** Sets a smaller font size and bold font to display error messages in red.

# **Advantages of using CSS vs. styling elements from the Scene Builder's own options**

## **Advantages of using CSS:**

* **Reusability:** Define styles once and apply them to multiple elements.
* **Flexibility:** Advanced customization and responsive designs.
* **Separation:** Separate CSS for better organization.

## **Disadvantages of using Scene Builder for layouts:**

* **Limited:** Fewer customization options than CSS.
* **Maintainability:** Can be difficult to manage for complex layouts.

**When running it  
**

# **Notes**

**IntelliJ IDEA**

If you install the latest available version of IntelliJ IDEA (2024.2.1), you may have problems when using Scene Builder, since the IDE suggests to install it, but it does not do it correctly and does not advance beyond that point. For this reason, we have chosen to work with an older version of IntelliJ IDEA (2023.3.8).

**Java**

If you are using Java 8, you are likely to encounter difficulties during installation in IntelliJ IDEA. Therefore, it has been decided to use JDK version 17.0.1.

**Installation**

Remember to save everything you download and install, such as JFoenix, JDK and other tools, in a specific folder created to manage these resources. This will make it easier for you to find the necessary paths when configuring your project, such as adding the JFoenix .jar file to the libraries and modules to use its components.