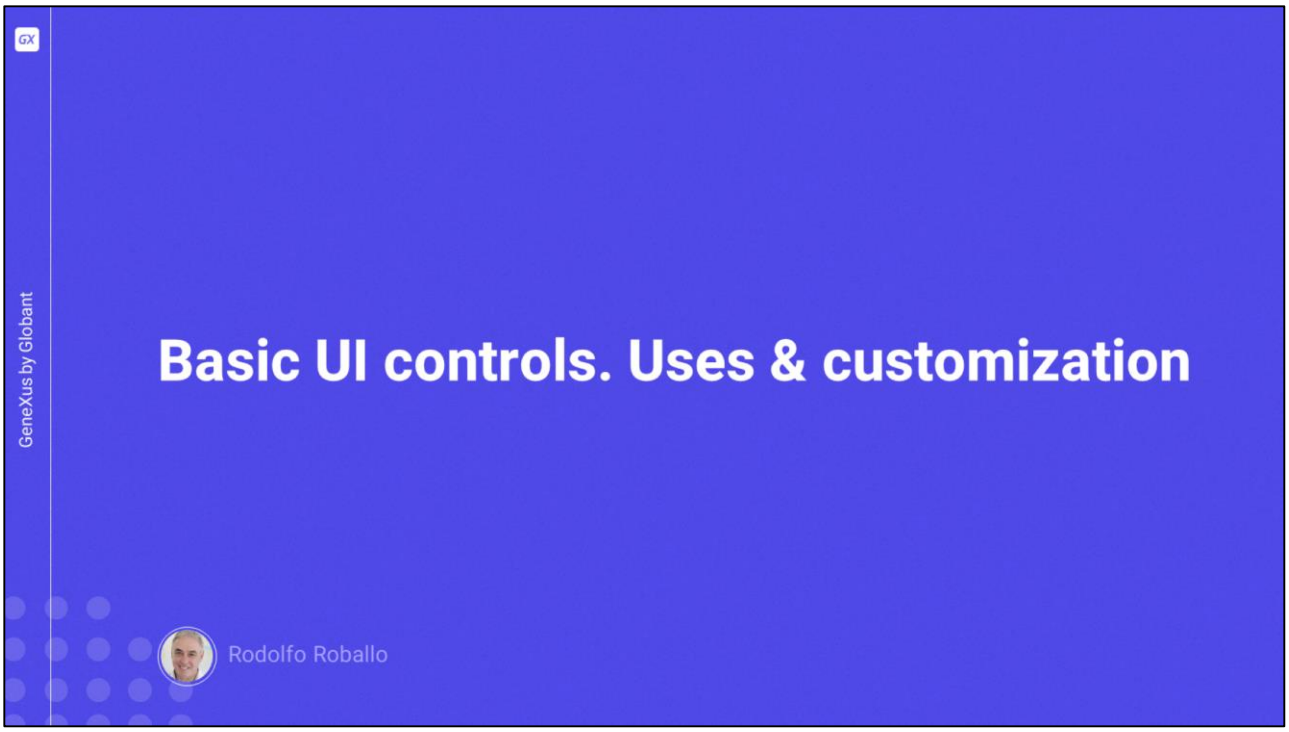
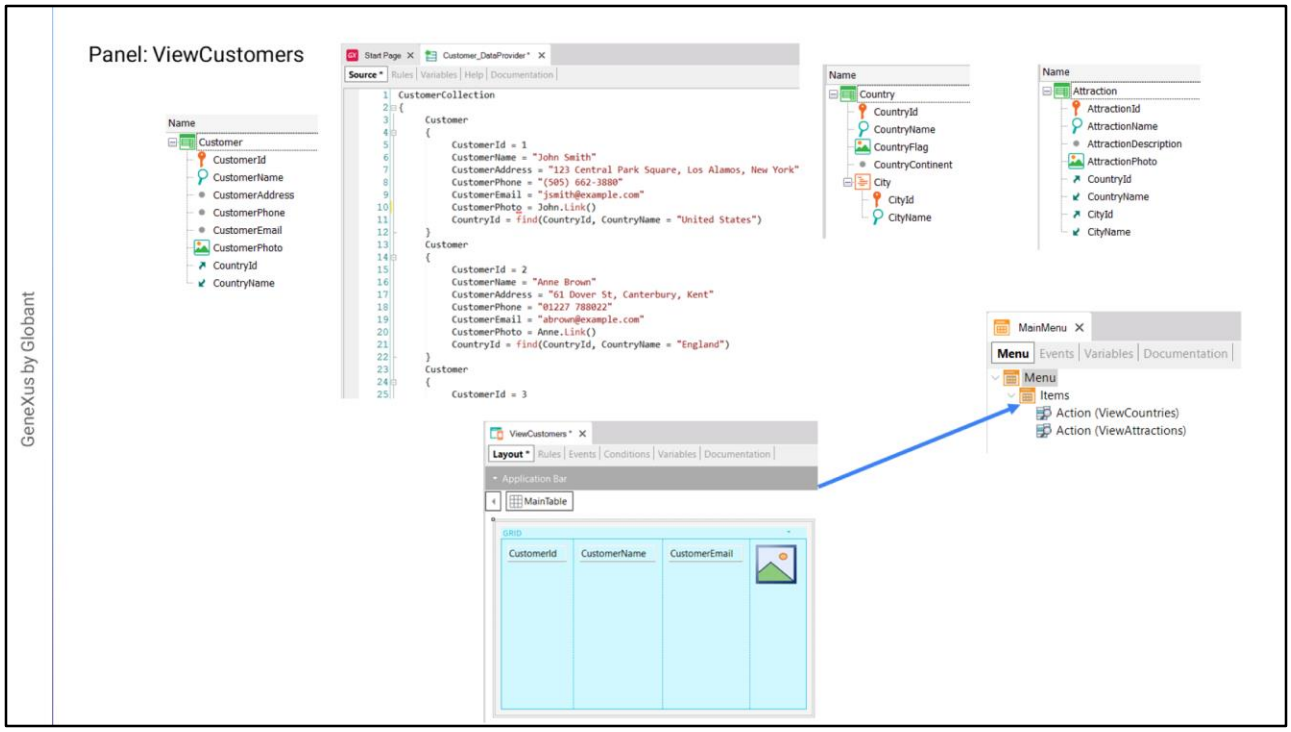


In any software application, the correct use of the available screen controls is essential in order to provide the best user experience.

This video will show the most frequently used controls, as well as how we can organize them in the panel layout and customize their appearance.

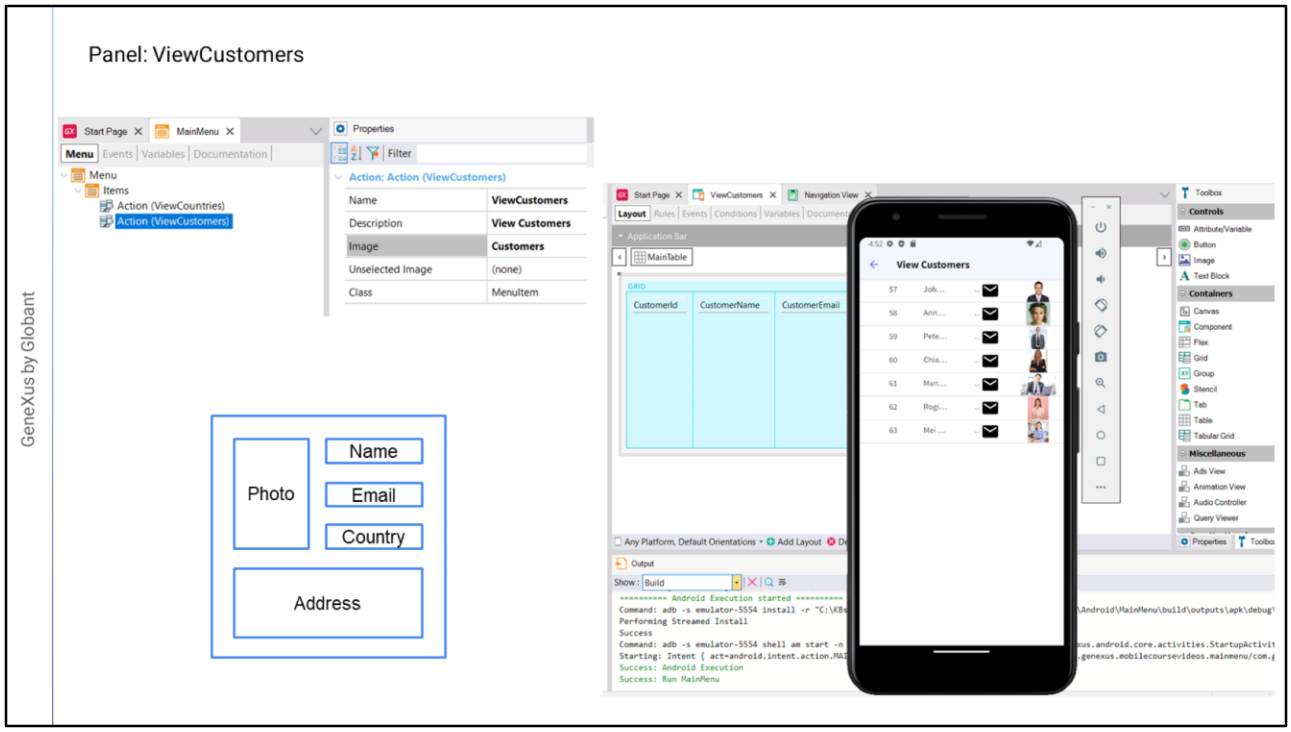


We are going to create a panel to view the travel agency's customers data.

First of all, we import into our KB the Customers.xpz file that we found in the Materials section of the course page.

The Customer transaction was imported with attributes for the main data such as the identifier, name, address, phone, email, and photo. We can also see that the data provider Customer_DataProvider was imported, including customer data and other objects.

To view customer data, we created the ViewCustomers panel, dragged a grid control with the attributes CustomerId, CustomerName, CustomerEmail, CustomerPhoto, and CountryName... and saved.



Now we open the MainMenu object and drag the ViewCustomers panel to the Items node. In the Image property of the created action, we assign the Customers image.

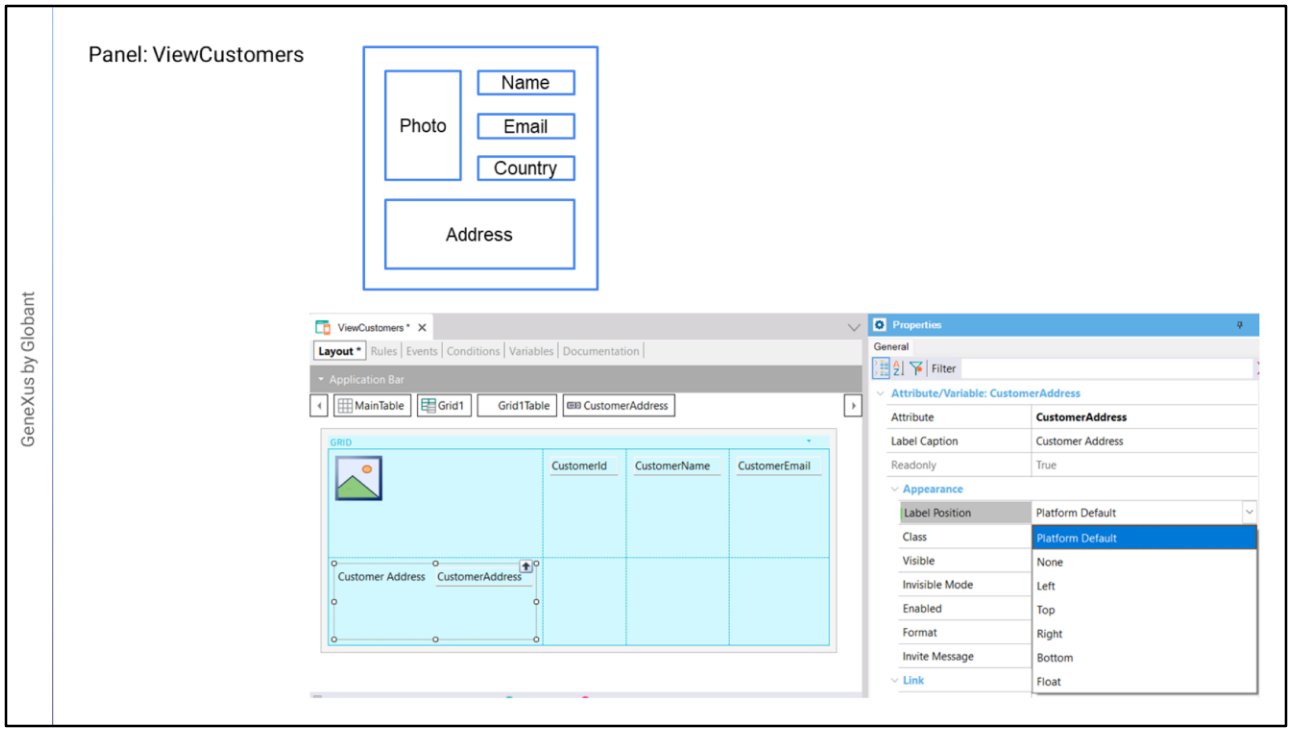
Note: in order to see the Menu as it is shown here, create the Panel ViewAttractions, drag it to the Items node of the MainMenu object and assign in the Image property the image named 'Attractions"'. We will work on this Panel in a future video.

We recommend to always run the application with the emulator previously opened, to optimize prototyping times. In this way, when GeneXus tries to instantiate an emulator to run the mobile application, it will find this emulator running and use it. In this way, we will save the instantiation time of the virtual device and the execution will be much faster. Now we press F5.

The list of clients appears without any layout, and there are even some data that we don't get to see due to lack of space.

We will try to get to the following card layout, which seems to be the most appropriate. Each row of the customers grid will be displayed with this layout, with one customer per row.

Note that the ID is not included in this layout and that we want to show the customer's address which was not included in the original grid. Let's add the address by right-clicking on the grid and choosing “Add attribute or variable” and Customer Address.



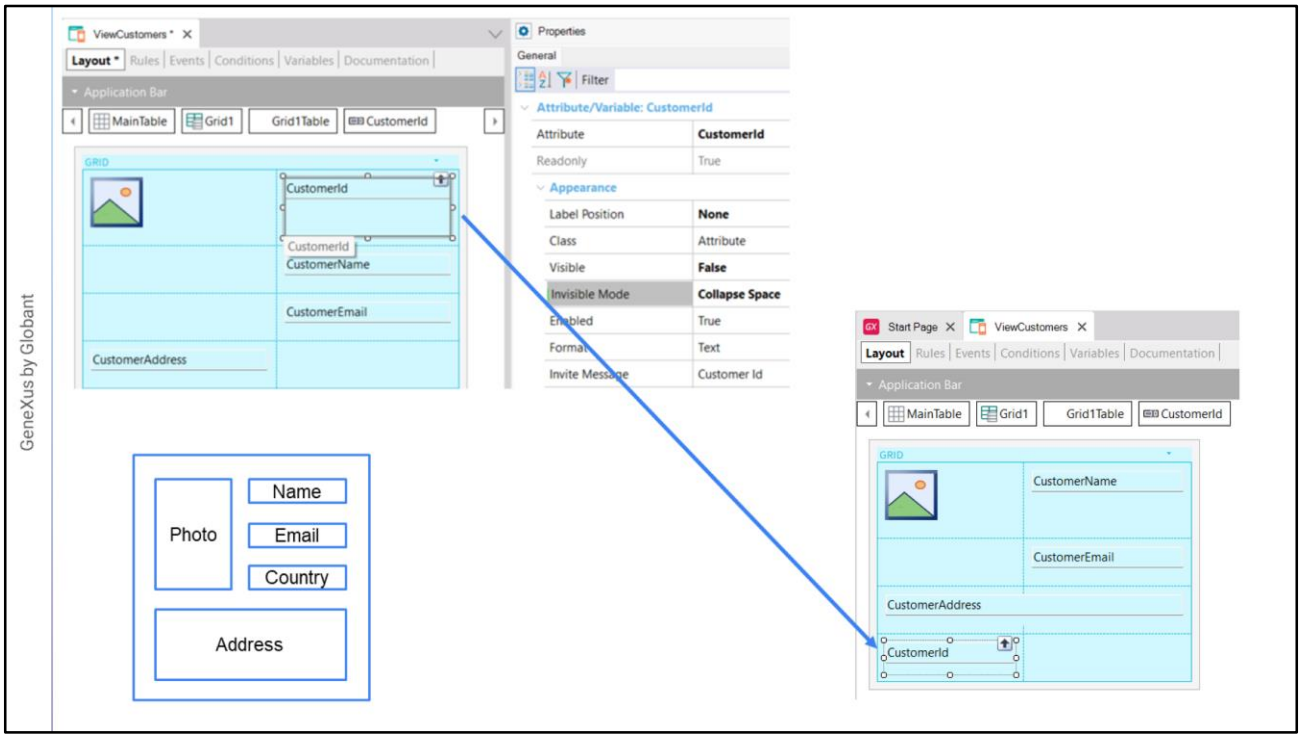
Let's start by moving the photo attribute to the left side of the grid and dragging the address field below the photo.

Note that CustomerAddress has a label on the left. If we go to the attribute properties, below the Appearance section, we see that the Label Position property has the Platform Default value. For Android it is the Top value, above the content, and for iOS it is to the left.

If we open the combo we see a series of values that change the default place where the label associated with the attribute will be displayed.

The value None means that the label will not be displayed. The values Left, Top, Right, and Bottom determine that the label will be displayed to the left, top, right, or bottom of the content field.

With the Float value, the label will be displayed on the position of the content attribute or variable, and when the user starts typing in the field, the label will move up the field as if floating. This only works for editable fields and data of numeric or character type; for all other types, the value Top is adopted.



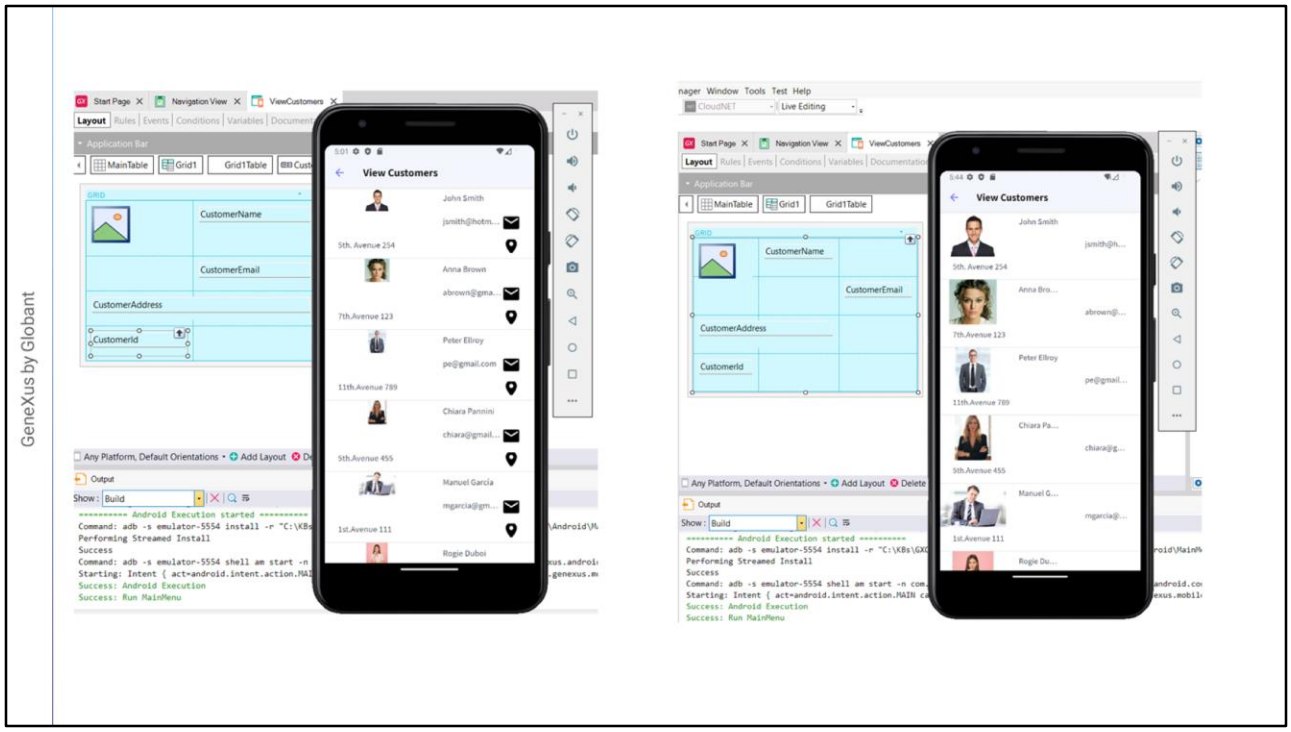
For CustomerAddress, we will choose the value None, so the label disappears.

We move the name, email, and country attributes below CustomerId. Since we don't want to show the CustomerId, we can remove it or set its Visible property to False. If we need to obtain other data from a selected customer, we must send by parameter the value of its identifier, so we should hide the attribute by setting the Visible property to False instead of deleting it.

Below the Visible property is the Invisible Mode property, which is set to Keep Space, the default value. This means that even if we hide the attribute as we did, it will still keep the space it takes when it is visible.

If we open the combo, we see that we also have the Collapse Space value, which means that the attribute will be hidden and the space will be collapsed, allowing the other controls to take the available space. However, this option only works to collapse row data (horizontal collapse), but not columns (vertical collapse). Therefore, since CustomerId is in the same row as CustomerPhoto, we must remove CustomerId from there, and if we move it to the last row, it will be collapsed.

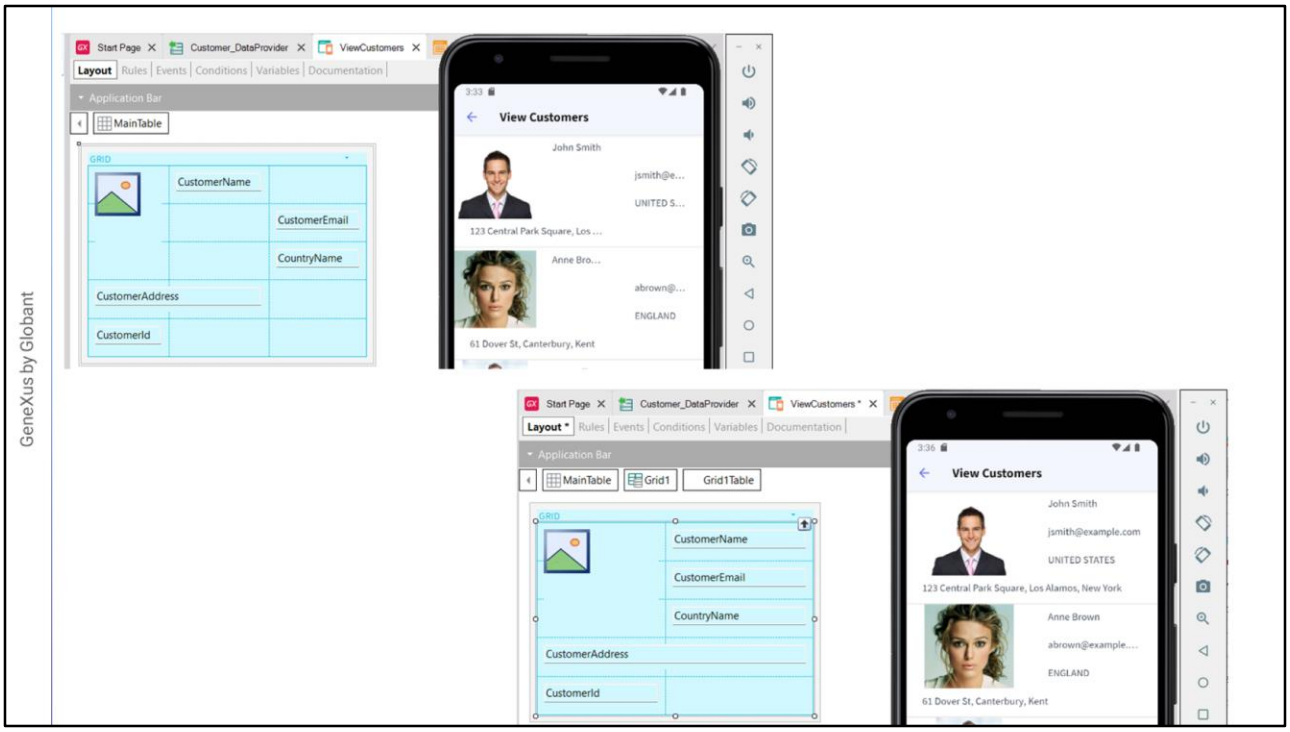
We are going to rearrange the controls a bit to achieve the design we wanted. Since we need the address to take the space of the two columns, we go to the Cell Information section and in the Col Span property we enter the value 2.
And press F5...



We tap on the View Customers icon and see that the list of customers is loaded.
Although it may be similar to what we need, it doesn't look very good. We do notice that CustomerId is not displayed and that the place it would take up is collapsed.
Let's try to improve the size of the photo a little bit and have it take 3 rows, setting the Row Span property to 3.

Before we continue with other changes, let's use the Live Editing functionality provided by the GeneXus IDE.

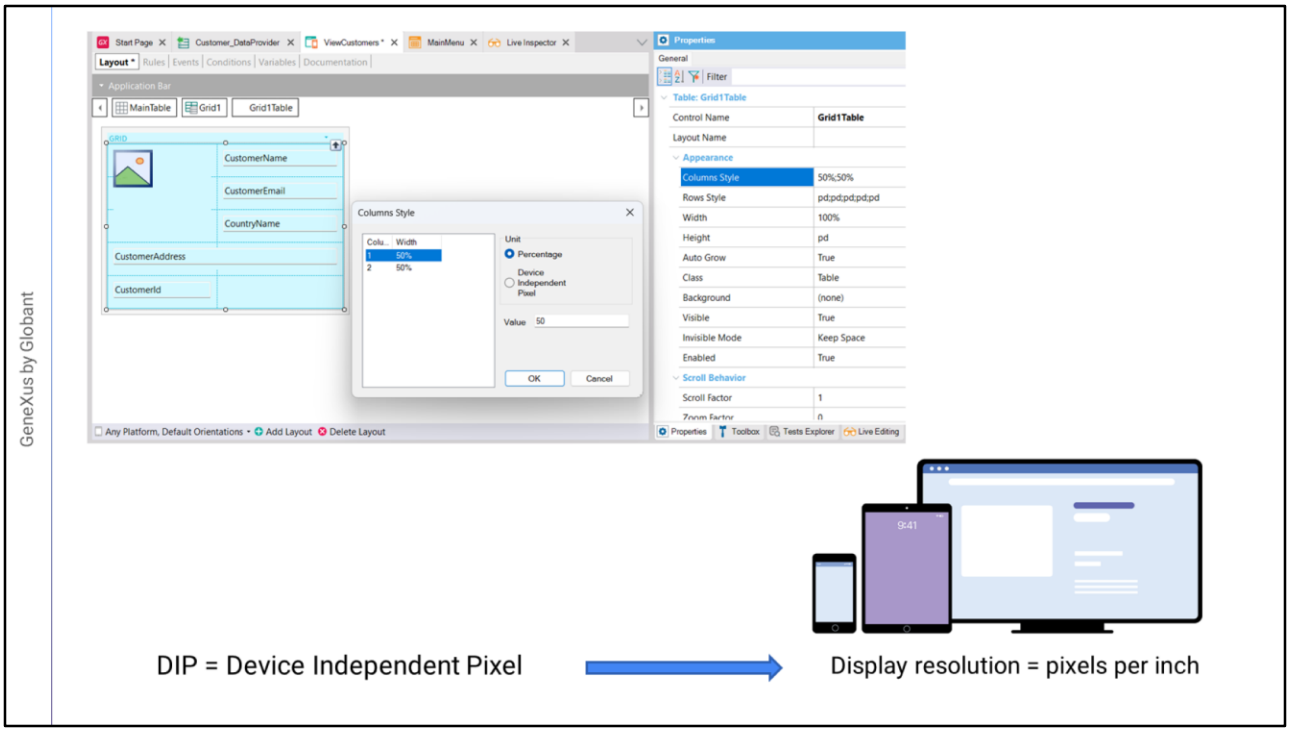
To see the changes with Live Editing, in this combo box we change the Release value to Live Editing and run the application again.
We check that Live Editing is working by looking at the Live Editing tab of the Properties window, where the IDE is connected to the emulator we had running. We also see that the Live Inspector window has opened, where we can identify the screen controls and their properties.



Returning to our design, we see the client's name, email, and country in different columns. We place them in the same column. Thanks to Live Editing, the change was immediately reflected in the emulator without having to save or rerun.

This allows us to make changes to the design, such as changing the position or alignment, assigning another class, or changing the properties of a class and thus improving the appearance of the screen and seeing the changes instantly.

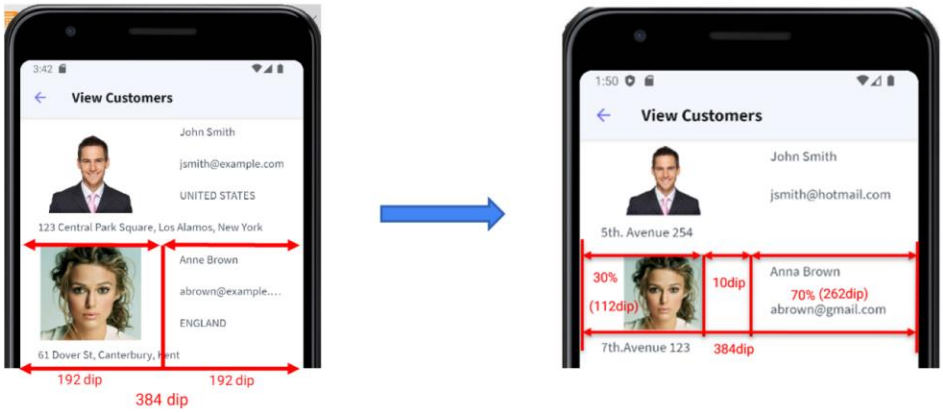
We are now going to delete the column that is empty, so we right-click and choose Delete Column. Once again, we see that the change is already visible in the emulator.



To adjust the size of the columns and rows, we select the grid table, called Grid1Table. Below the Appearance section, we find the Column Style, Row Style, Width, Height, and AutoGrow properties that allow us to change the size and behavior of the controls.

In ColumnStyle, it says 50%;50% which means that the table has two columns and that each one is taking up 50% of the available width. We open this property and see the two columns identified and that each one has the Percentage unit selected with a value of 50.

The other unit of measurement is DIPs: Device Independent Pixels. The Device Independent Pixel corresponds to an abstraction of a pixel that then an application converts into physical pixels, which allows scaling to different screen sizes. The dip value has a different number of pixels for each platform.

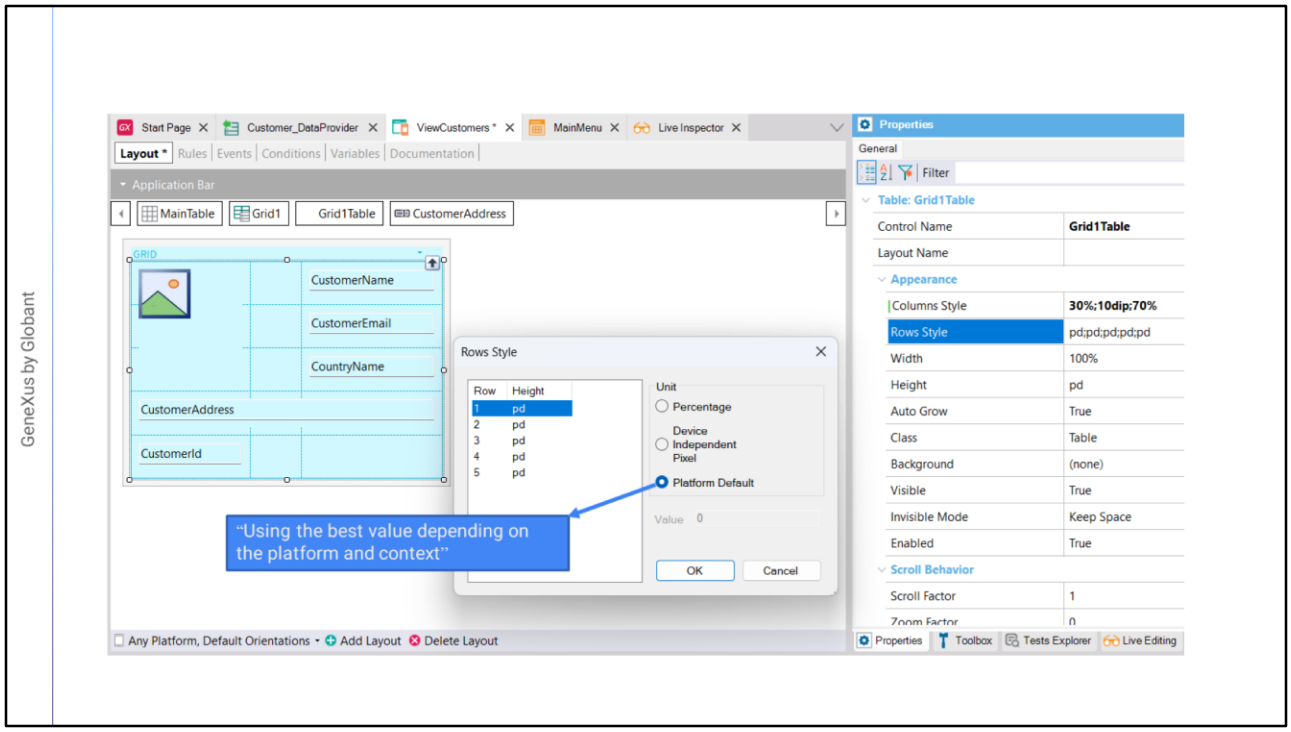


The total width is fixed and is 384 dip, so with this design each column takes up 50% or 192 dip.

But if there is a column defined in dip, the percentage values are relative to the value that results from subtracting the fixed values (in dips) from the total width. For example, if we want to separate the photo column from the text column, we can add a column as a separator.

If we had three columns, the first with 30%, the second with a fixed value of 10 dip and the third with 70%, the values of the first and third columns would be obtained by applying these percentages to the value resulting from subtracting the sum of the fixed values (here only one, 10 dip of the separator column) from the table width.

For this reason, the space available for the first and third columns is: 384 – 10 = 374 dip; therefore, the first column will have 30% of 374 which is 112 dip, and the third one the remaining 70%, which is 262 dip.

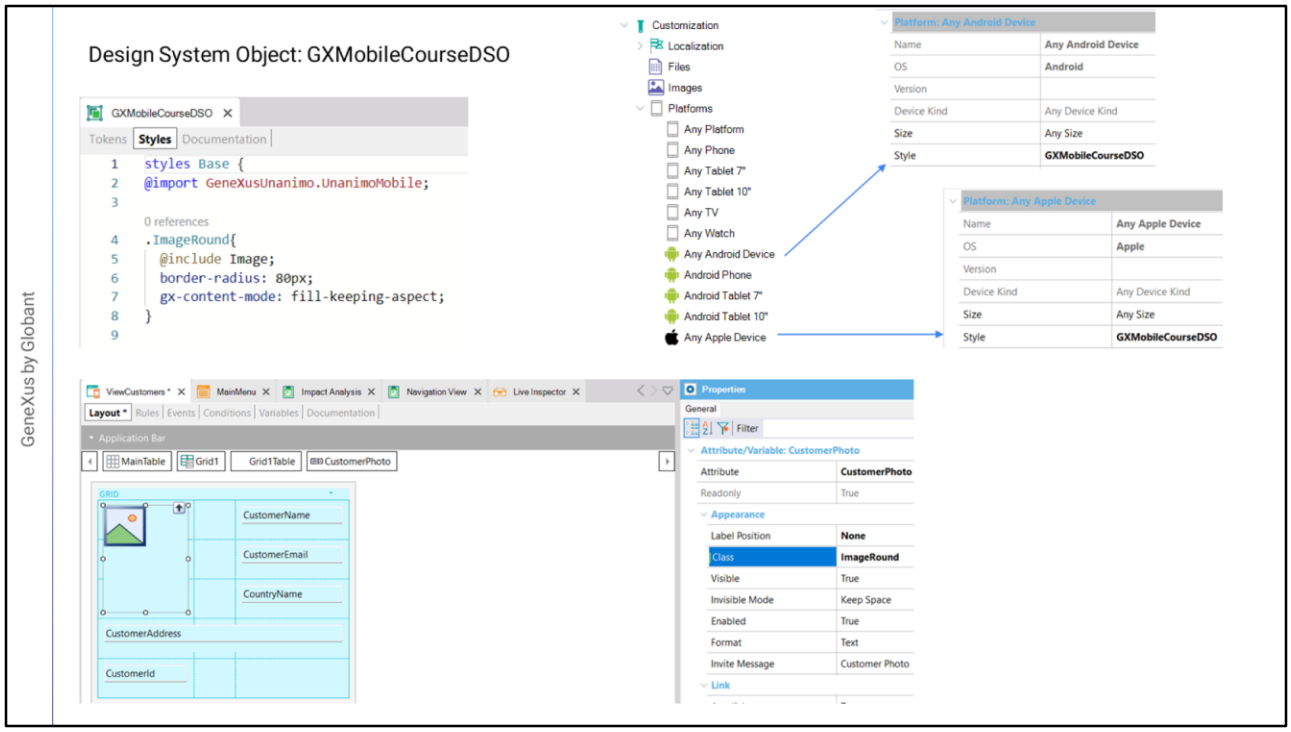


Let's select the photo, right-click on it, and choose Insert column after. We will specify the Columns Style values as we had seen, entering 30% for the first column, 10 dip for the second one and 70% for the third, and see the result. Because we are using Live Editing, it has changed and now there is a space between the name and the image.

If we now look at the Row Style property, there are 5 rows with the value pd: Platform Default.

This value varies according to the platform and, for the same platform, it also depends on the content of the cell and whether the field has a label or not; if it does, if the label will be displayed at the top or at the left. The pd value corresponds to: "Using the best value depending on the platform and context."

For example, for Android with Label Position = Top, it corresponds to 64 dip, and in iOS to 53 dip.



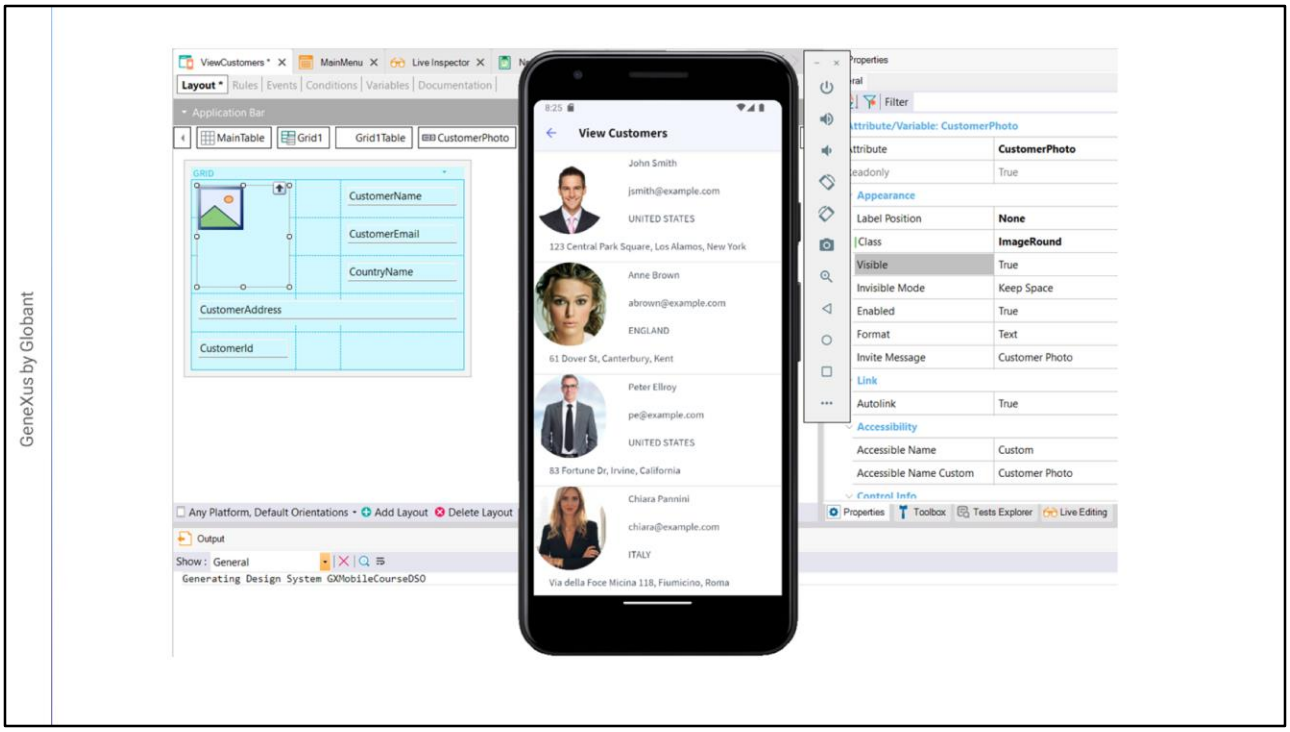
We check the execution and the rows look good, so we leave these values.

Now we are going to change the appearance of the photo so that it is round. To do so, we are going to assign to the photo control a class with properties that enable this.

We open the design system object GXMobileCourseDSO that was imported with the xpz, and in its Style tab we create a class named ImageRound. We assign to the border-radius property a value of 80 pixels that will make the image look round, and to the gx-content-mode property the value fill-keeping-aspect so that the image will automatically adjust to the available size without changing its aspect ratio; that is, the proportion between width and height.

For this Design System Object to be taken into account in mobile objects, we must assign it to the platforms in which we are going to generate. So, we open the Customization node and select AnyAndroidDevice; in its Style property, we assign the design system GXMobileCourseDSO.
We do the same for the AnyAppleDevice platform.

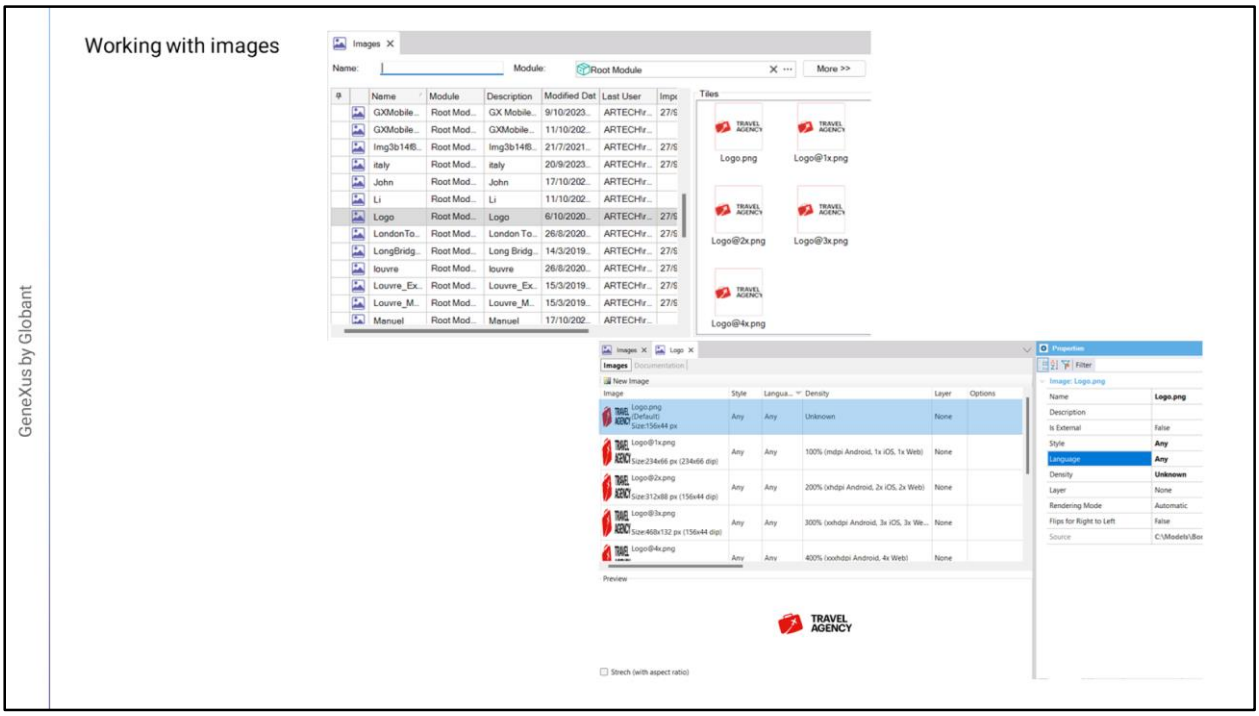
Now we select the photo and in its Class property we write ImageRound.



Note that as soon as we assign the class to the CustomerPhoto attribute, we can see the changes thanks to Live Editing. We could even change the values in the class and we would also see the change reflected automatically without even saving or compiling the object or running the application. This facilitates incremental development.

In this example, we are looking at the customer's photo, which is an image stored in the database. However, we often need to use a fixed image, such as a logo or a background image.

In this case, we must take into account that when adding the image to the KB, we must actually add several images with different resolutions, since the devices running the application may have more or less screen density and resolution.



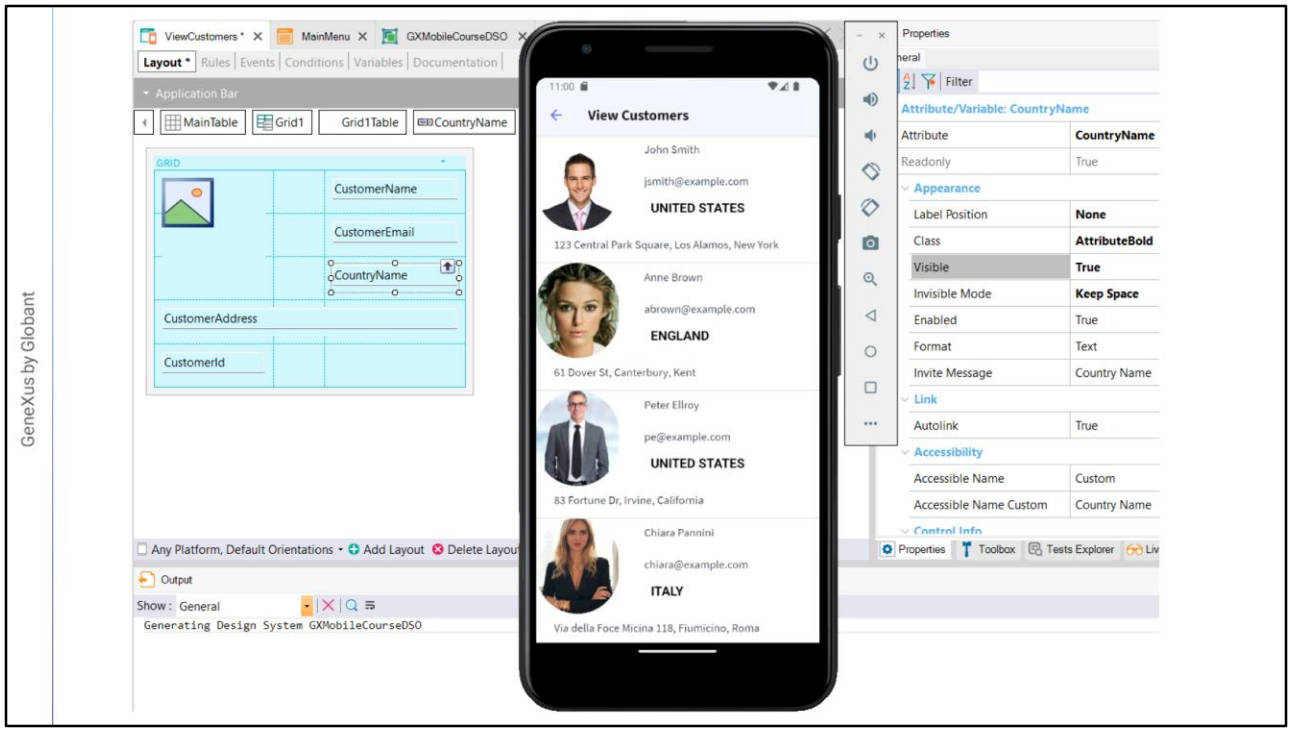
We select the image named Logo and see that there are actually several associated images. If we open the image, we can see that each image has a different resolution; when running the application, the appropriate one will be automatically selected depending on the screen resolution of the device.

We can also see that for each image we can specify which style it will use; that is to say, the associated Design System. This allows us to upload different images associated with different Design System Objects. In our example, the value Any means that the same image will be used for all the styles defined in the application.

We can also associate a specific language to it; for example, if the image includes a text in its graphic and this text varies by language, so that we can upload several associated images, one for each language defined in our application.

We can upload images with different density. For example, the value 100% corresponds to medium density (approximately 160 dpi) which is taken as a baseline for Android, as well as for Web and iOS; in the latter case, the value corresponds to screens that are not Retina Display. Remember that dip (Device Independent Pixels) are then converted to actual pixels according to the resolution of each device.

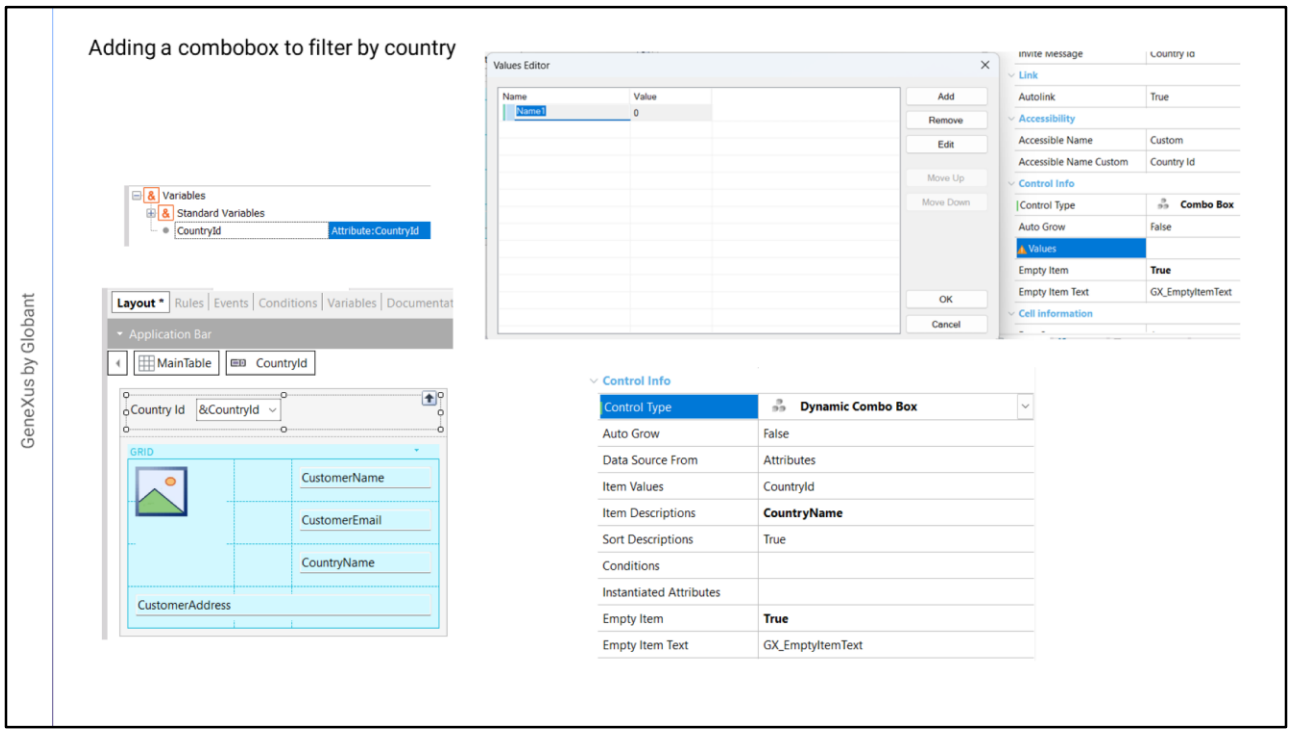
The value 200% is for Extra-high-density (about 320 dpi) corresponding to 2x Retina images for iOS and 2x for Android. A value of 300% corresponds to Extra-extra-high-density with about 480 dpi, which is for Retina 3x for iOS and 3x for Android, and 400% corresponds to devices with 640 dpi. As devices with higher resolution values become available, this value will increase.



We return to developing our application.

Now we are going to show the name of the country in bold, so we create an AttributeBold class that inherits from the Attribute class, with the following properties: font-weight in bold and the color property set to the value: $colors.AttributeBolded that we had defined as a token. Now we assign the class to the name of the country.

Also, we adjust all the data of the second column with Horizontal Alignment set to Left and Vertical Alignment set to Middle. The direction is also aligned horizontally to the left and vertically to top.

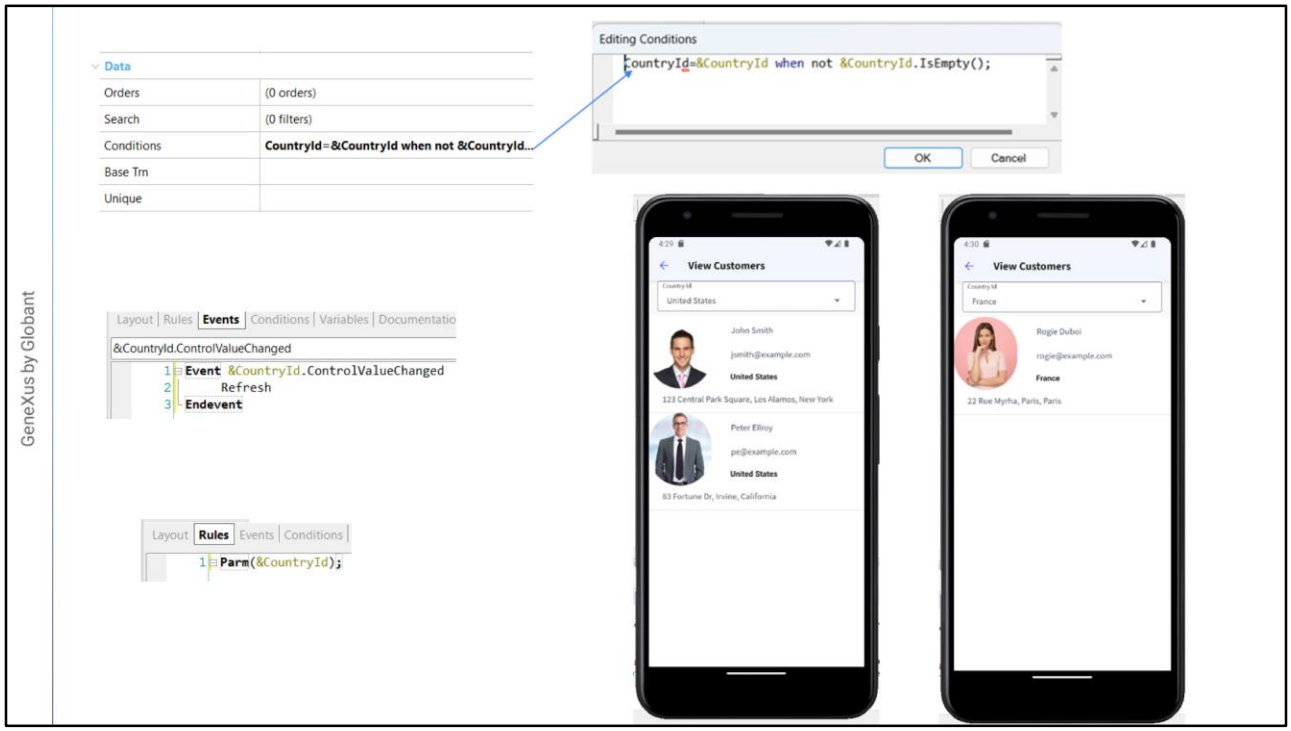


OK. To complete this screen's functionality, we will add a combo box that allows us to select the country and filter the customers by the selected country. To do so, first we create the &CountryId variable which, due to its name, is automatically based on the CountryId attribute, and add it to the panel form at the top of the grid.

If we go to the properties of &CountryId, we see that in ControlType it says Edit, so by default this variable will be seen as an edit field. To change its appearance and behavior, we open the combo of this property and see that we could assign it the ComboBox type, but in this case we should enter the values manually in the Value property to fill the combo.

We already have the countries' data in the database, so instead of this type of combo we choose a Dynamic Combo Box and see that the Items Values property is automatically assigned the CountryId attribute because this is the name of the variable, which will be the value returned by the combo when selecting a country.

The properties box says that we must assign the Items Descriptions property that will be the field whose values will appear in the combo when we open it, so we assign the attribute CountryName to this property. Lastly, as we want the combo box to start without any country by default, we set the EmptyItem property to True, leaving a default value for the Empty Item Text.



To make the country selected in the combo box act as a filter in the customer grid, we go to the Grid properties and in Conditions we add: CountryId=&CountryId when not &CountryId.IsEmpty();

In this way, only the customers whose country identifier matches the country identifier selected in the combo box will be displayed. If no identifier is selected, there will be no filter and all the customers will be displayed.

Now we go to the events tab and insert an event of the &CountryId variable –ControlValueChanged– and write the Refresh command. This will refresh the contents of the grid when we finish selecting the combo box and load the data from the database again, filtered by the selected country. We will see more about events later.

Finally, we go to the rules and add a Parm rule with the &CountryId variable. This is necessary in this mobile architecture because to minimize server queries we prioritize the use of cached data. Also, for the server to understand that we want to bring new data, we add a Parm rule with the variables we use in the filters, so that when their value changes, the page is updated.

Because we add filters and events that involve the server, to see the changes we have to compile the application again, so we press F5.

We open View Customers and see the combo box at the top that says (None) to indicate that we have not selected a country yet. We choose United States and see that the filter works and only customers from that country are shown. We select another country and check that everything works as expected.

In this video, we saw how to work with basic screen controls, and how to improve their appearance and distribution on the screen. In the following videos, we will continue to see other screen controls with interesting functionalities.