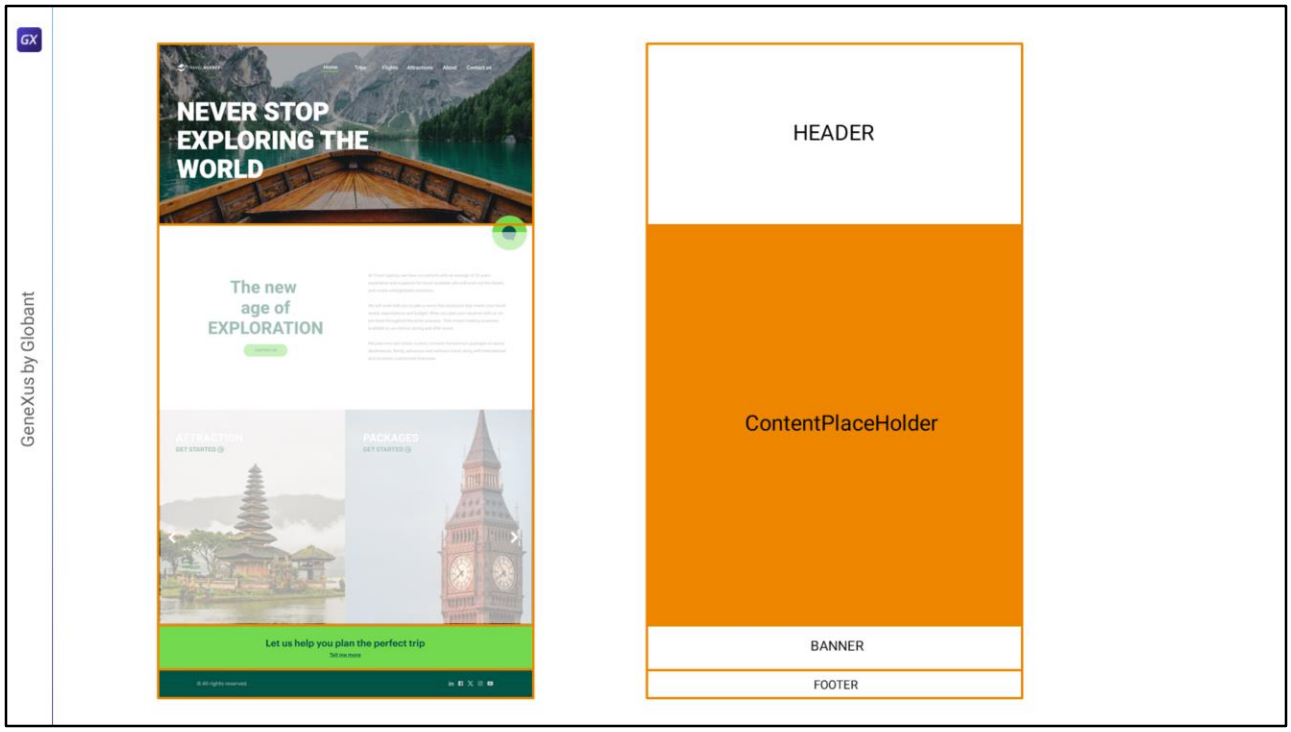
GeneXus by Globant

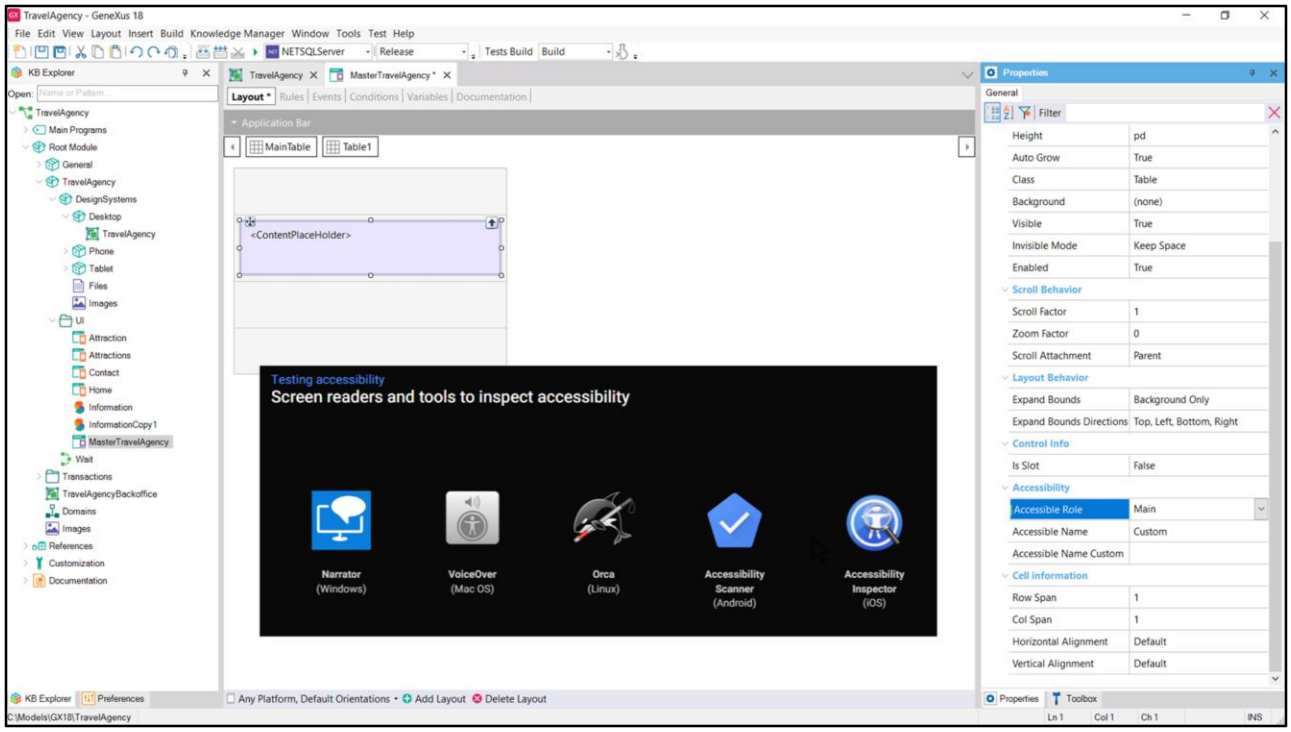
Accessibility and first steps for Master Panel

Cecilia Fernández



Here we have the first screen of the application, from which we will extract what corresponds to the Master Panel. The area in the middle will correspond to the space in which the layouts of each individual panel will be loaded: the Home, Attractions, Attraction and Contact panels.

Therefore, the structure of the Master Panel could look like this: as a table with 4 rows; in the first one, we will implement the Header; the second one will contain the ContentPlaceHolder control, which is precisely the control that causes the individual pages that have this Master Panel to be loaded there; for the third row we will have the implementation of the banner; and in the fourth one we will implement the footer.

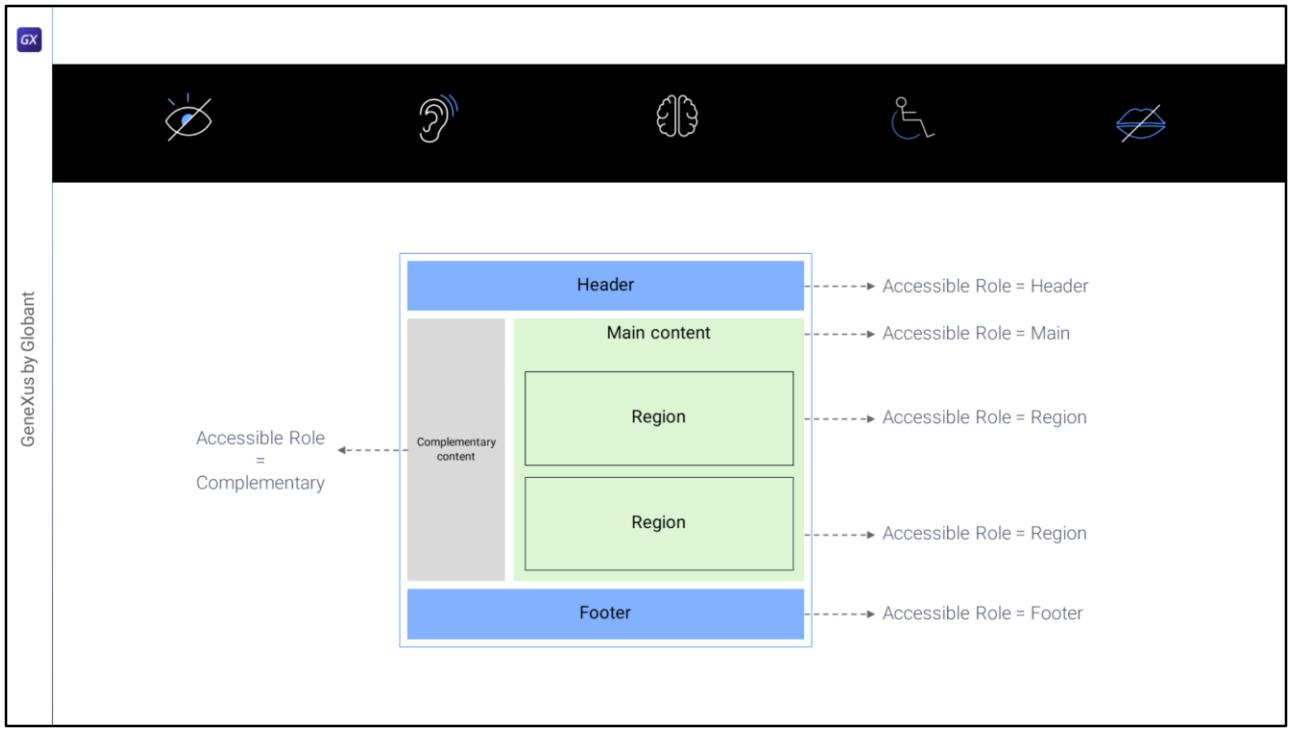


So, let's start with this structure in our MasterPanel object.

We had only the ContentPlaceHolder, but now we will place it inside a table, and both of them inside another table that will have 4 rows.

Why do we place the ContentPlacerHolder inside a table and not simply inside the cell of row 2? Because the table will allow us to make its content semantic: we will be able to say that the main content of the page will be here. What for? To make the application more accessible. For example, a visually impaired user can use one of the many screen reader programs available that will inform them via audio that this part corresponds to the main content.

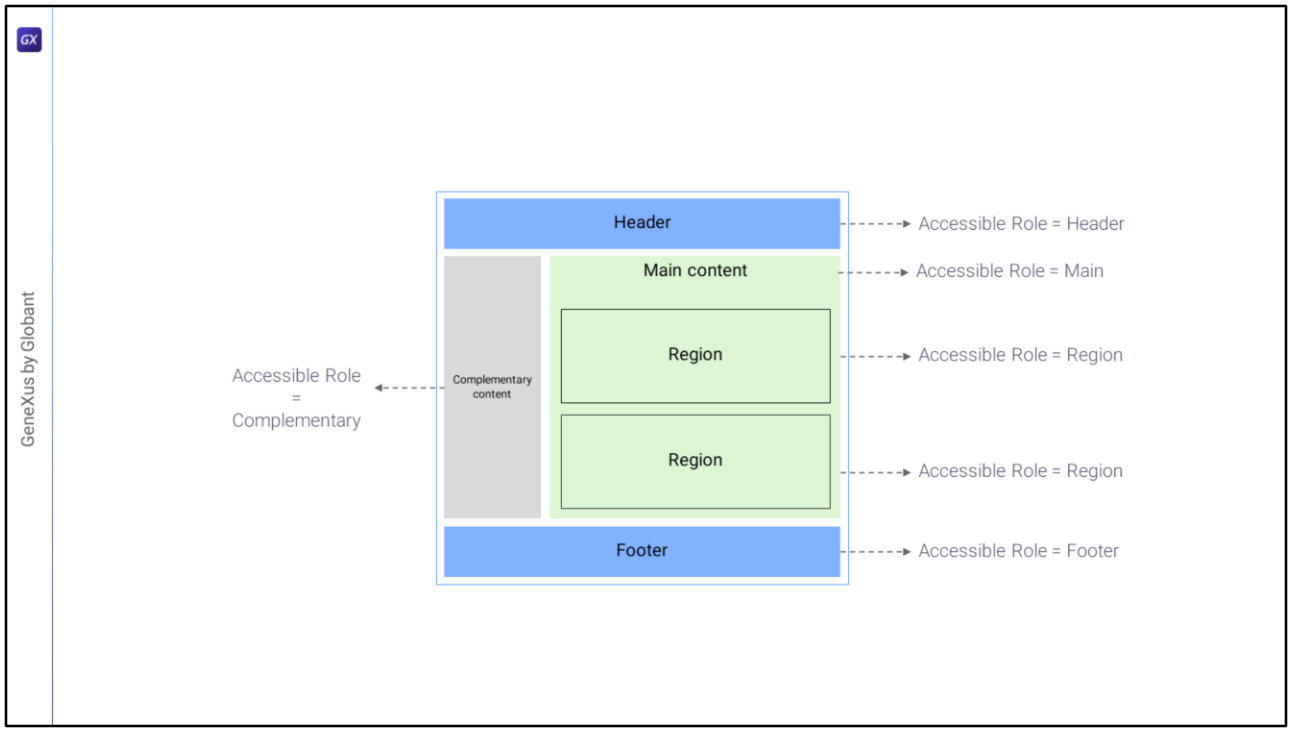
Let's look at the accessibility options shown at the table level. There are 3 of them: 2 and 3 go together.
The first one, that of the role, can have one of these values.



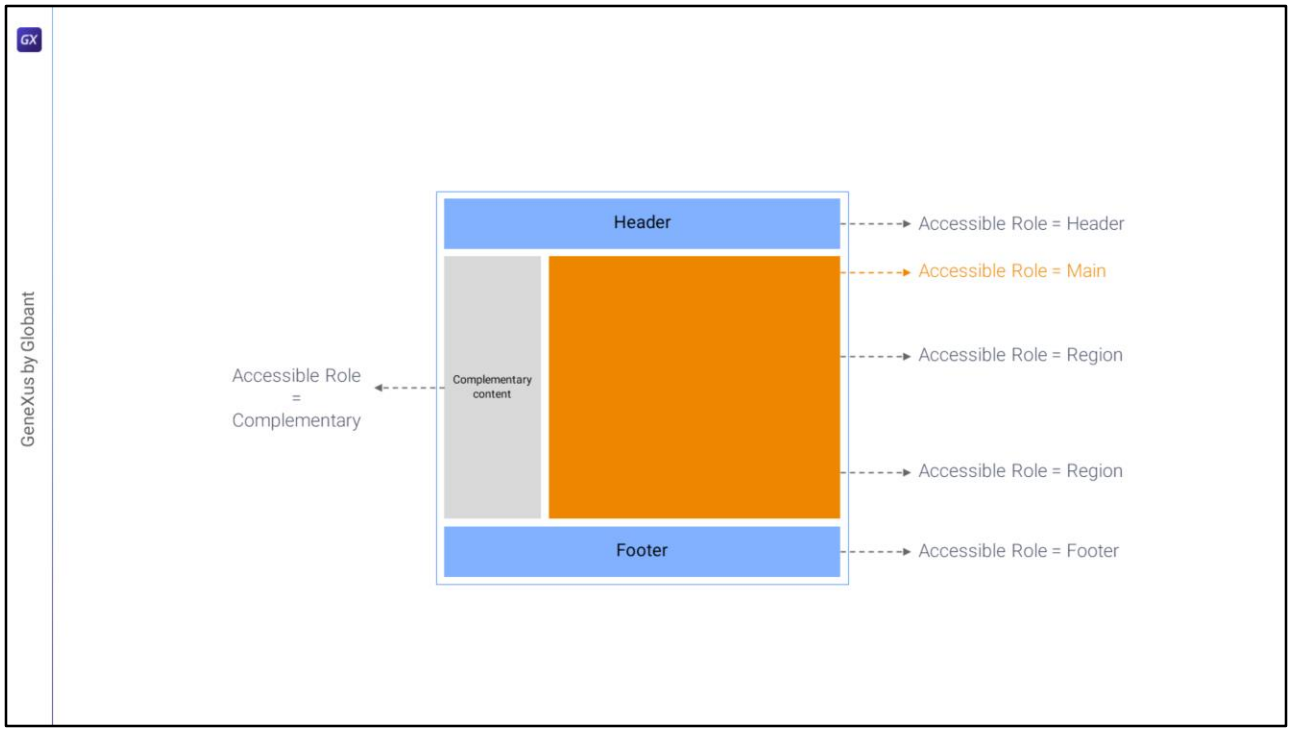
Here are the semantics of most of them.

The accessibility of our applications is becoming increasingly important and in many cases is even required by law, in many countries. It not only benefits people with physical or cognitive disabilities, but also when there are Internet speed limitations, for example, because users can know what content goes there even if they have to wait to view it. In addition, it improves search engine optimization (known as SEO), and therefore people will be able to find our web application more easily.

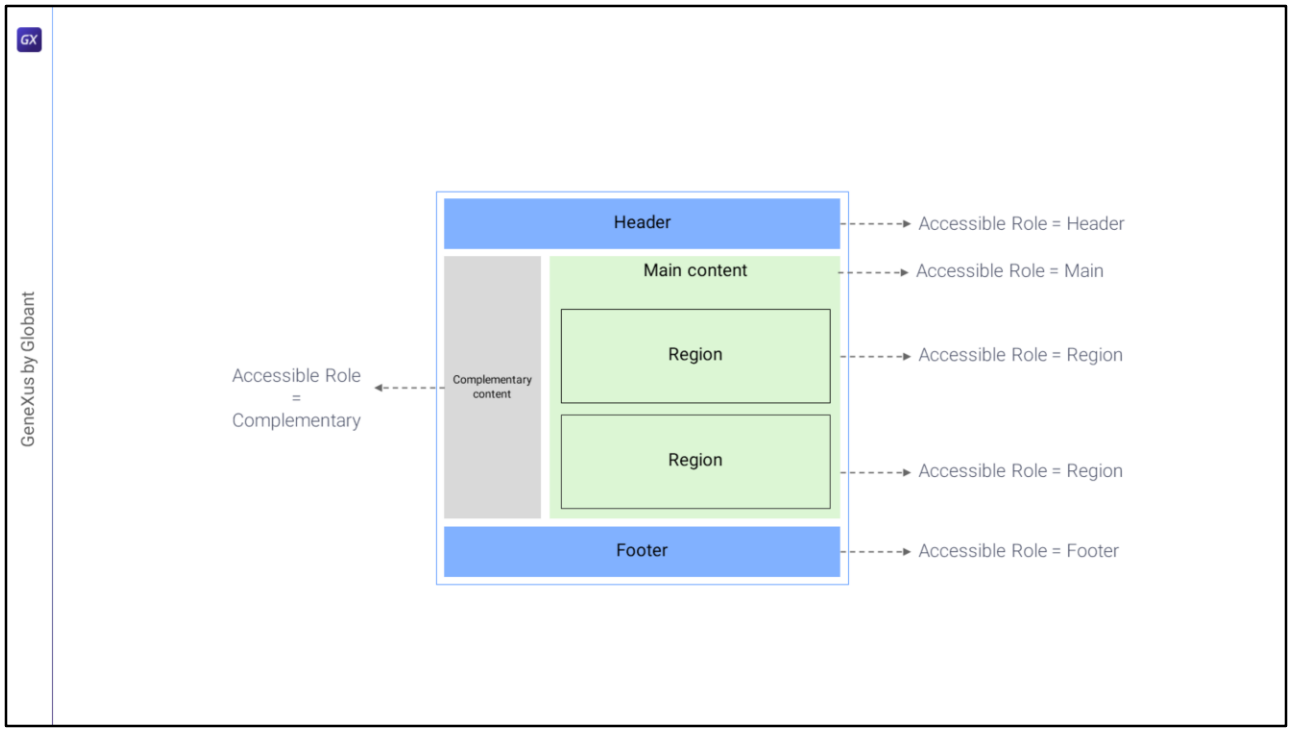
If from the start of development we strive to make the application as accessible as possible, the cost is minimal and the benefit is high. Having to redo the development to address accessibility later will have a much higher cost, so let's take care of this from the beginning!



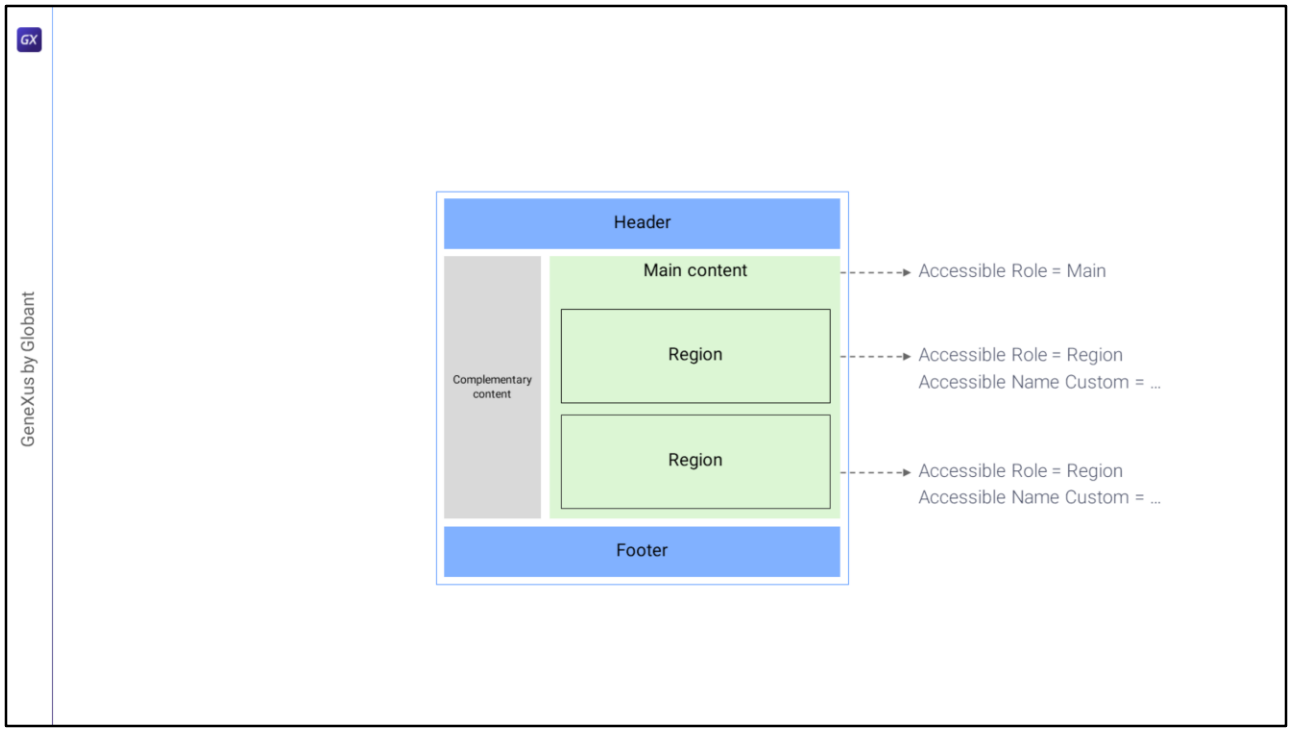
It is obvious, then, that we will have to set up the layout in a way that allows us to specify the **roles of the containers**...



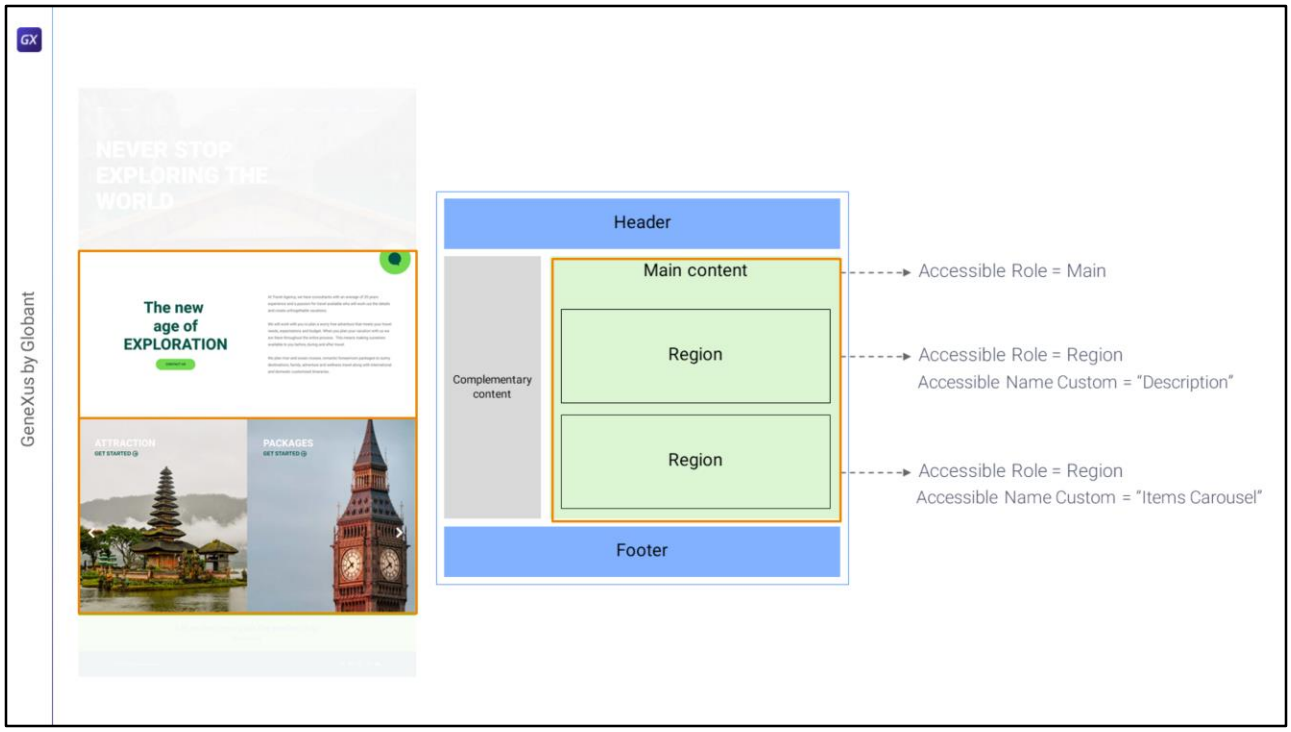
(that's why we need to place the ContentPlaceHolder inside a table, because only the table controls (all their variations, that is to say, also flex and canvas) have the Role property).



This property will only have an effect for Angular. At the moment the native application doesn't do anything with this information.

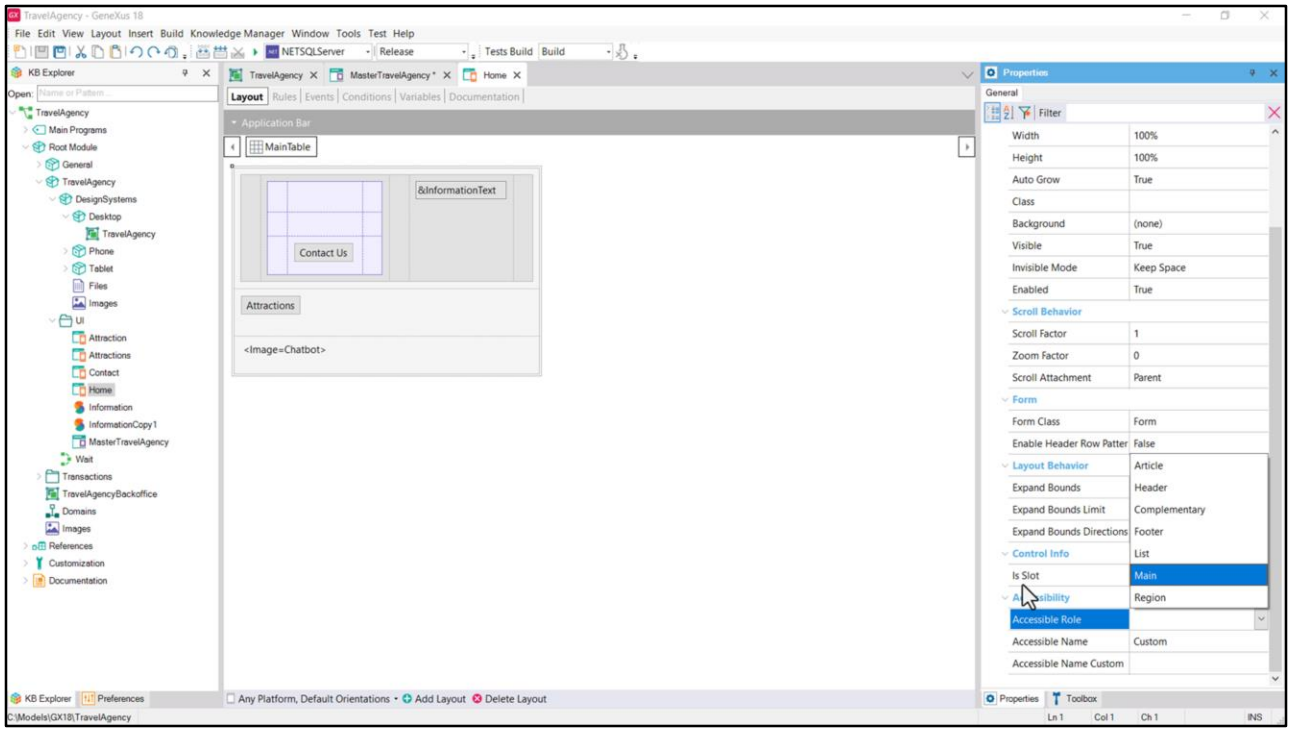


On the other hand, while there can only be one Main container per page, there can be many regions, so in order to describe them semantically we will need another property, which will be Accessible Name Custom.

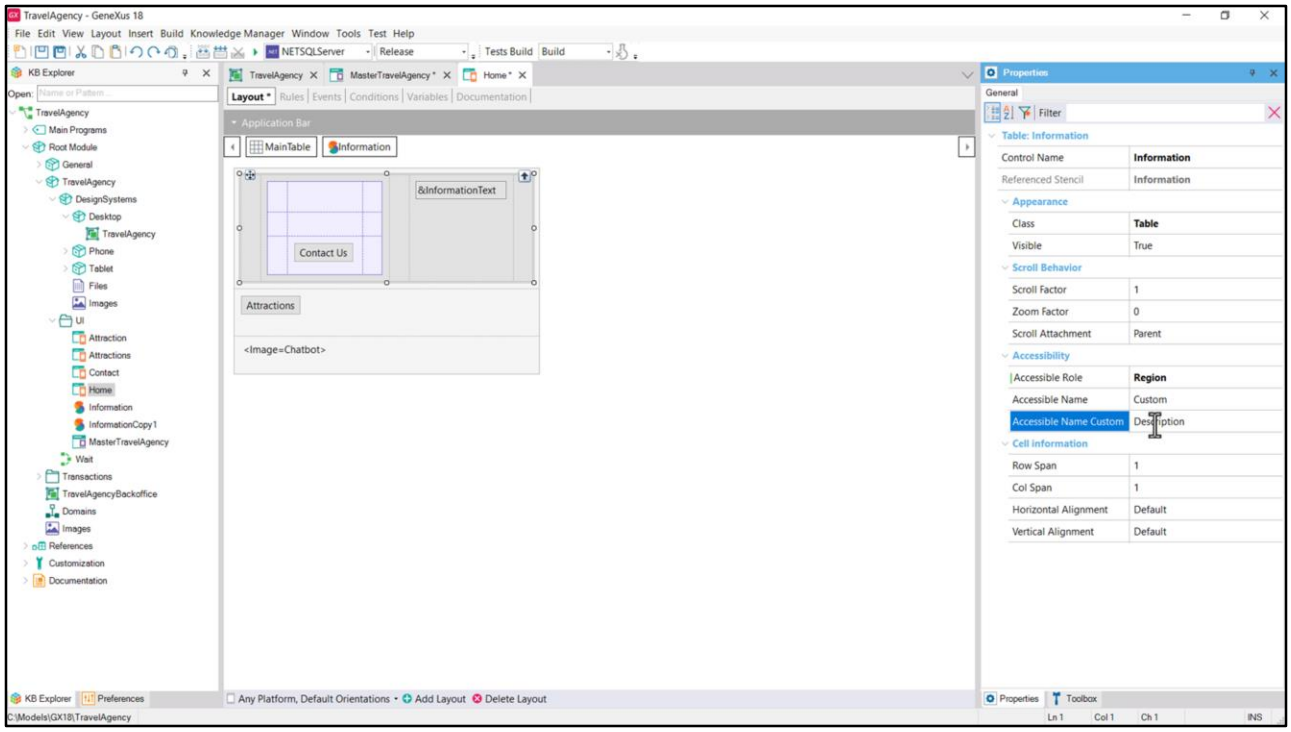


If we think of the Home panel (and not the Master Panel): we will have a Main table with two rows, and inside each one a table with Accessible Role Region and Accessible Name Custom. The descriptive name of the content of the first region could be “Description”, and that of the second one will be “Items Carousel” to indicate that there will be a carousel of items there.

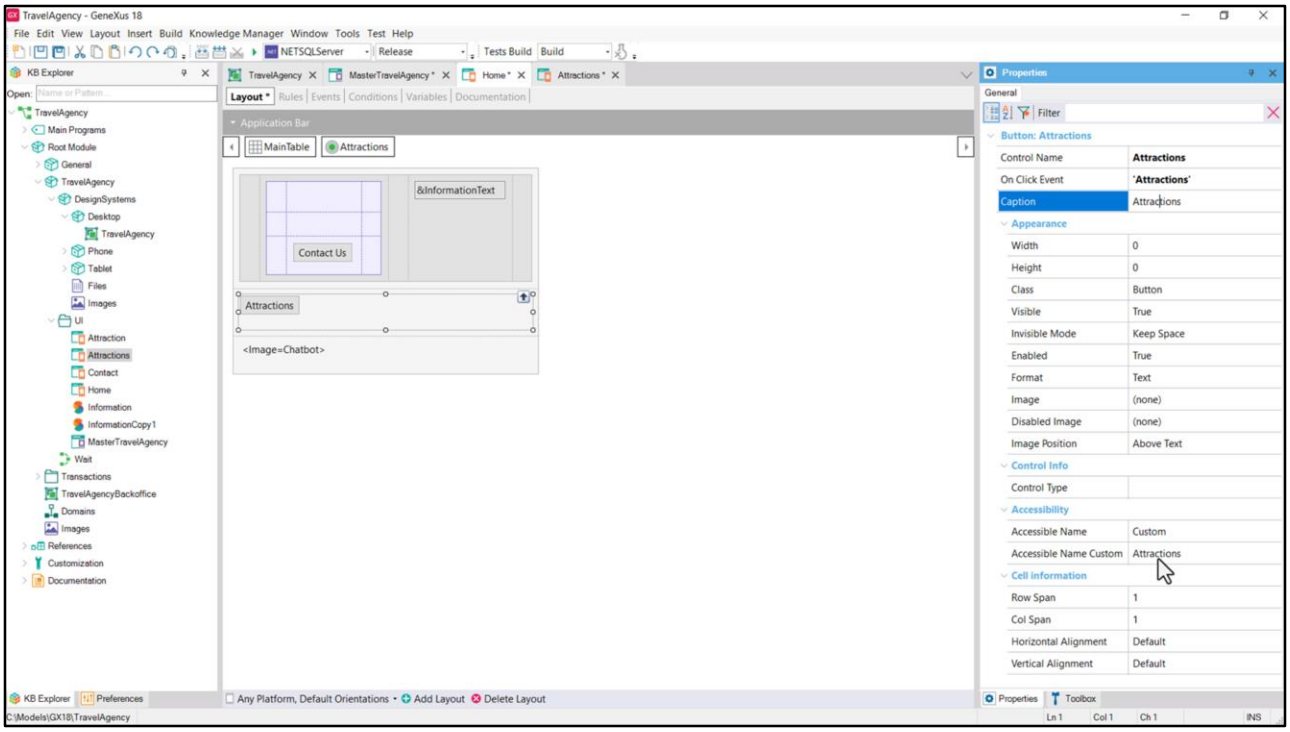
Then let's go to the Home page…



...to the main table we would set the Main one as Role...



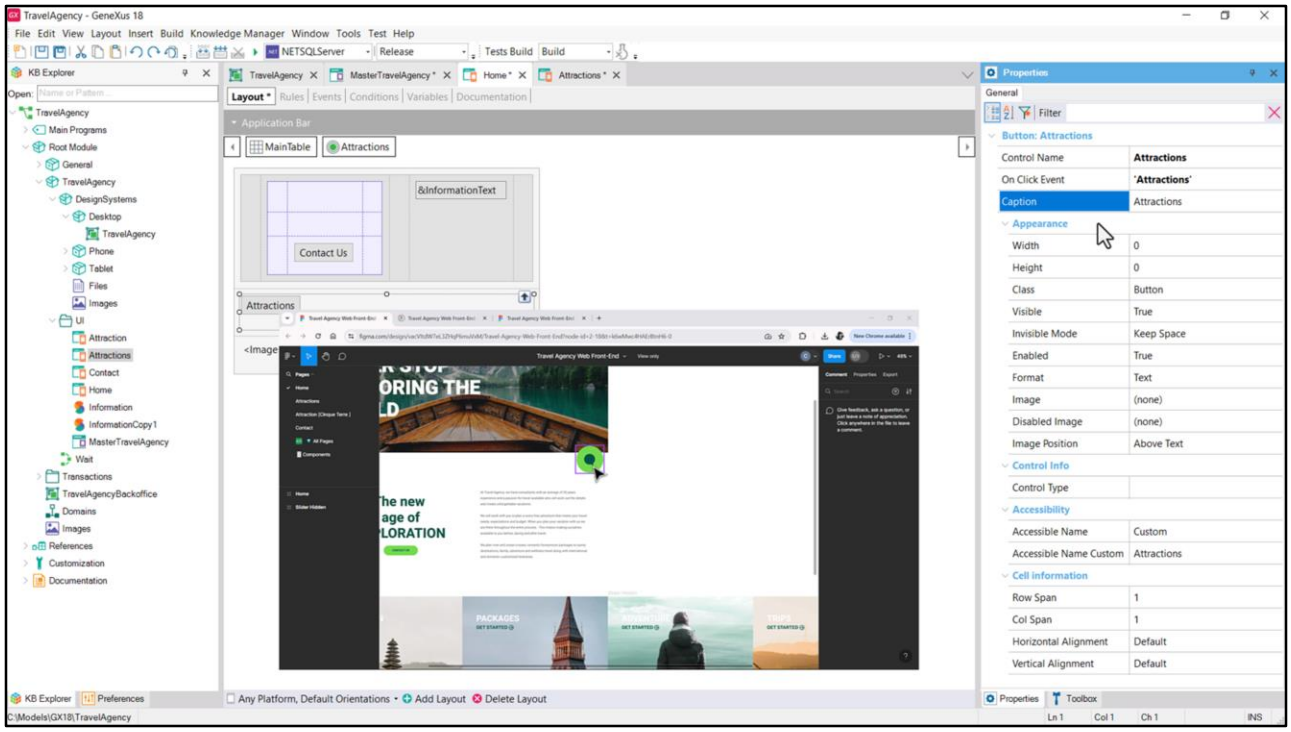
...to the table of row 2 we would set Region, to the Accessible Name we leave the default value Custom (because the other possible value is to take the description from the caption of a text block that could do in this case but that we'll see later), and in this property we would set the Description value as the description. We will do the same in the Attractions panel.



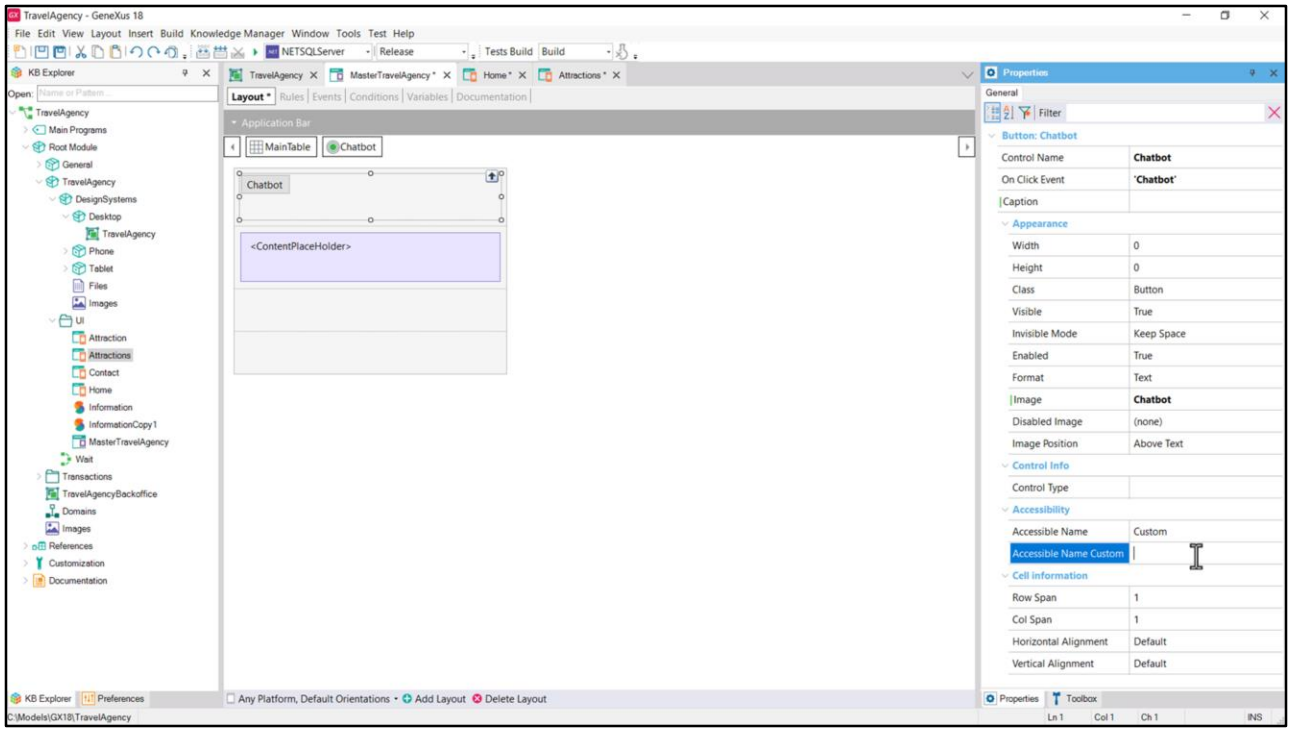
We haven't implemented the carousel for the Home yet, and we already know that this button is here temporarily; we will remove it when we implement the menu. Also, we had left this image just to show something about the images in a previous video; it won't remain either.

But let's take this opportunity to note that both controls have accessibility properties. For the button with the Attractions Caption, note that by default GeneXus placed that same value for the semantic description of the button.

The fact that it is a **button** (and not a textblock with an associated event, for example) and that its semantic description is Attractions will have a couple of important consequences in the application generated for Angular. On one hand, because it is a button it already has the universal semantics of the button: it can be handled with the keyboard, by placing the focus on it with the tab key and pressing Enter, for example. And because it is a button and has a description, a screen reader will inform the user that it is a button with the Attractions description.
Here we didn't have to do anything because the button took the Caption Accessible Name Custom.

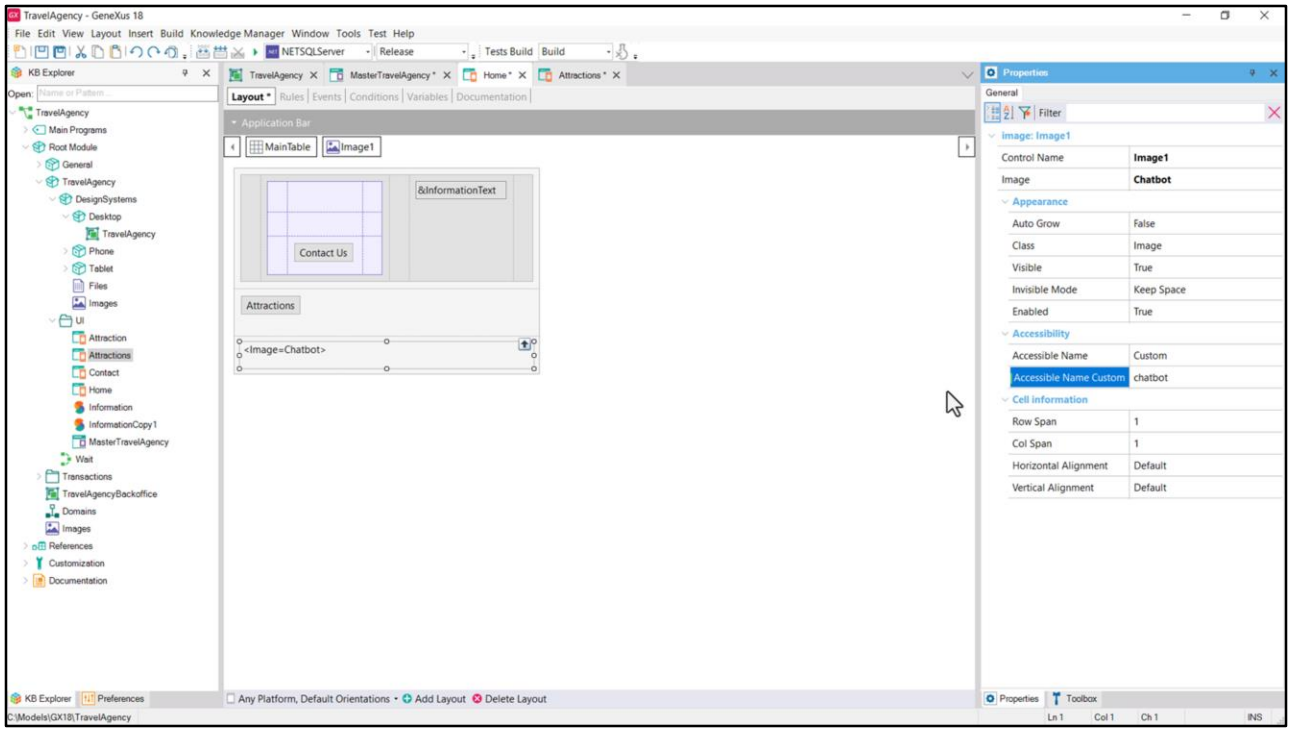


But if it didn't have a Caption, for example, because it will be shown with an image only, as it will be the case of the chatbot image...

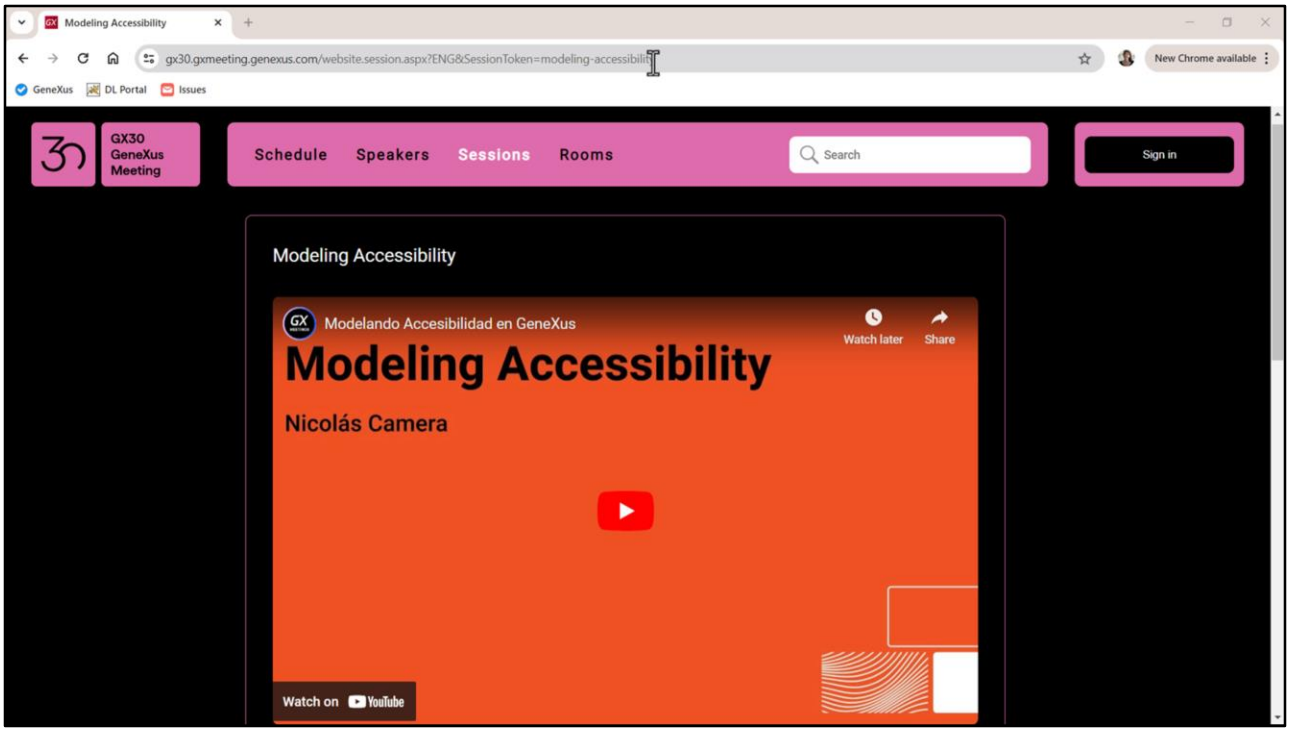


...and we can already do it in the Master Panel to show it...

I drag the button control... associate a name with the event that will be fired when it is clicked on... that name is the one associated with the Caption by default... but I don't want the user to view a text but the chatbot image... so I associate the image that was already loaded in the KB and leave the Caption empty...
But when doing it, the Accessible Name Custom property is also empty. Here is where we have to do something explicitly. I will enter “Chatbot” as its semantics.

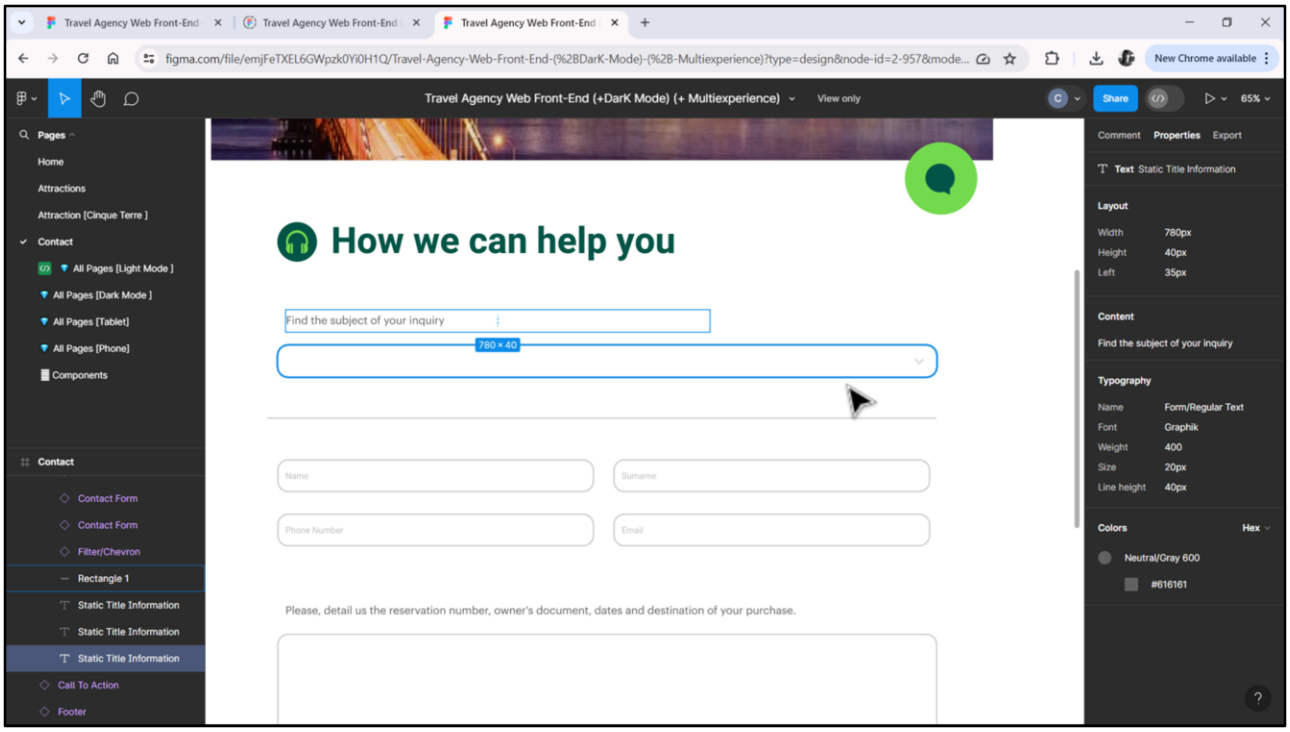


The same can be seen with the image control inserted in a layout, as an image and not as a button. By default an image control has no descriptive property, so it has no default value to give here. We will have to write it explicitly.



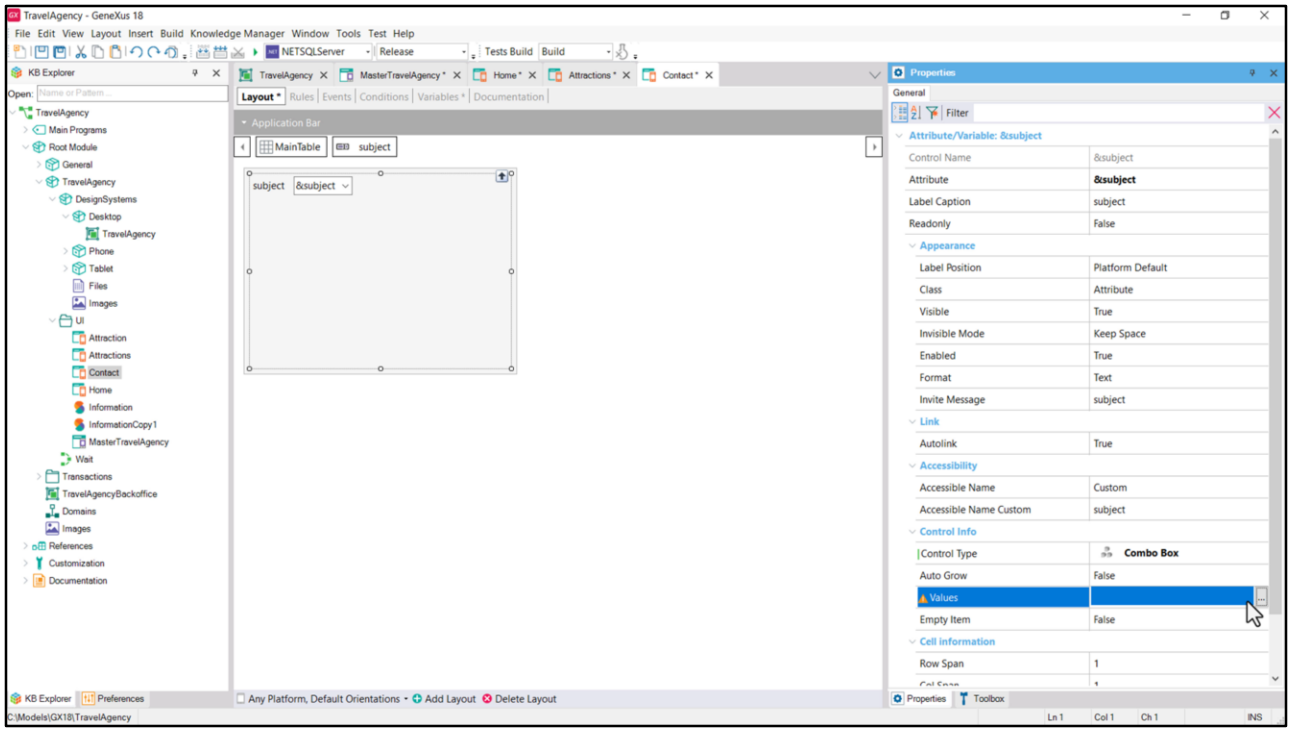
Here is a presentation from the end of 2023 where Nico Camera analyzed all the accessibility aspects to take into account and provided tips to address it from the beginning of the development with GeneXus.

There you will see that GeneXus already does a lot for us without us having to worry about anything other than choosing the appropriate controls and providing, through its properties, descriptive semantic values.

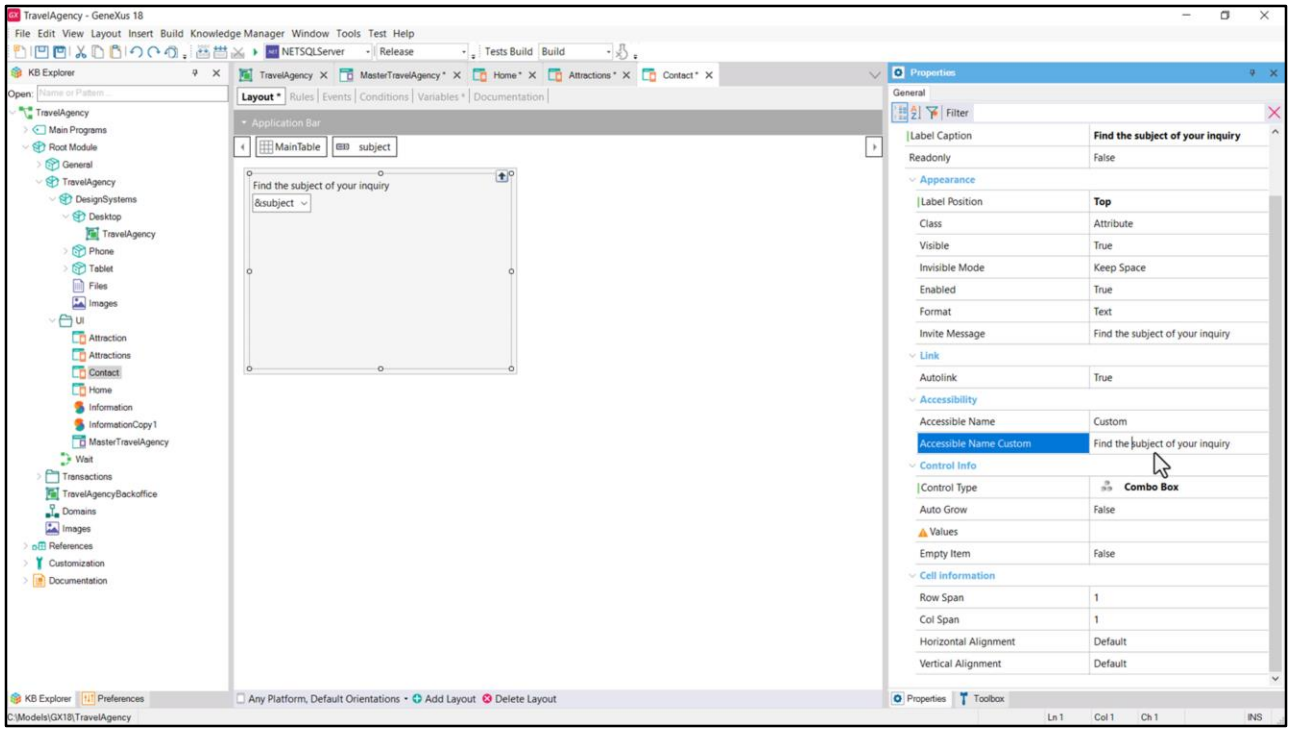


For example, our Contact form will have fields to be filled in by the user.

The first one will be a combo box, and its semantic function is given by this text... here we see that it is a combo.

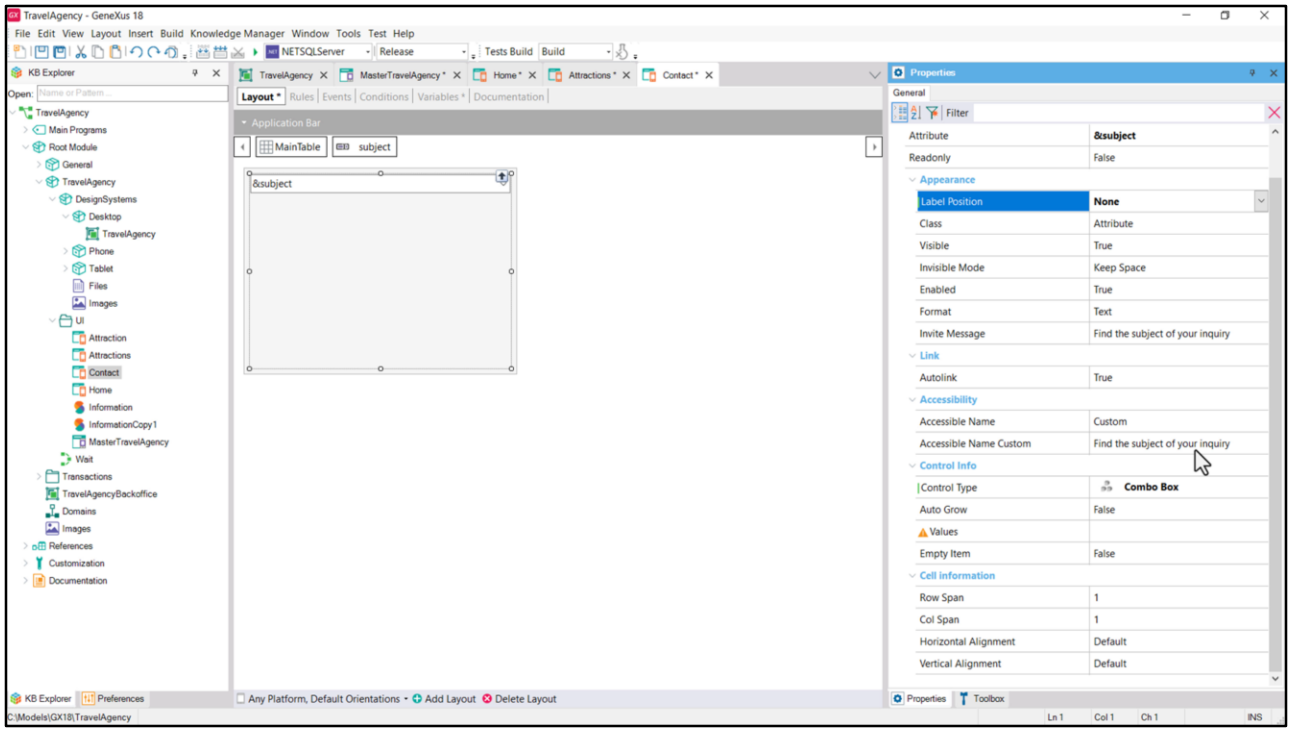


So, we will implement it with a variable, which I named in this way, and that we will insert in the layout through an attribute/variable control; we will specify the combo box as the type of control. There we will have to enter its options.

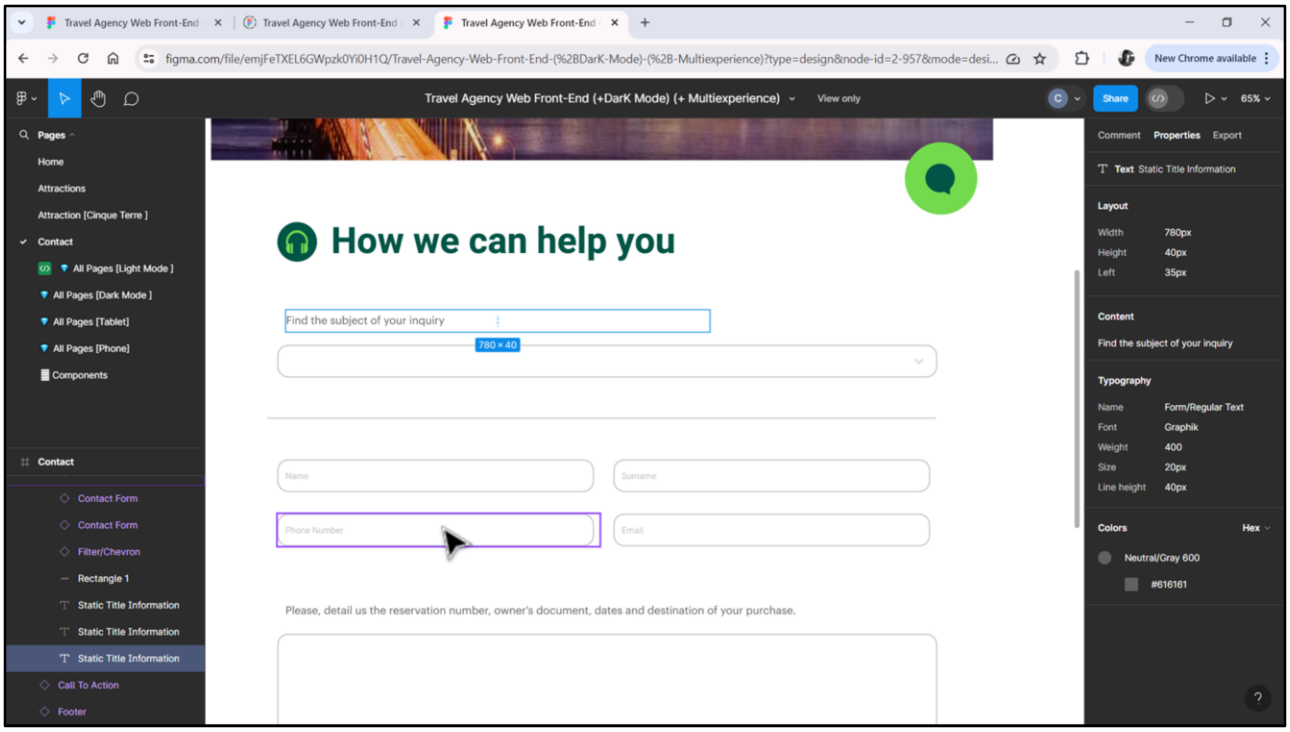


But now I'm interested in how to model the text that will appear at the top.

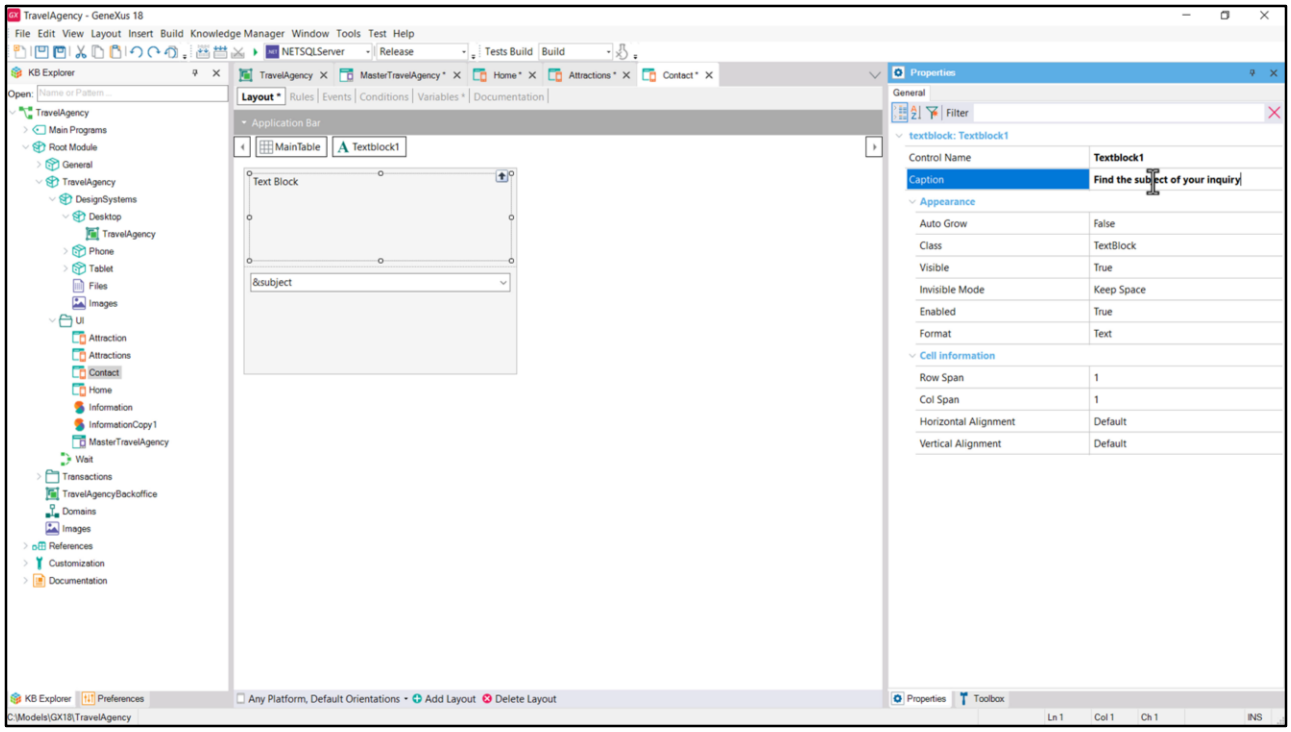
The native and most obvious way is to use the Label Caption property of the variable and then place that label on top.
The value that the Accessible Name Custom property will assume by default will be that same one...



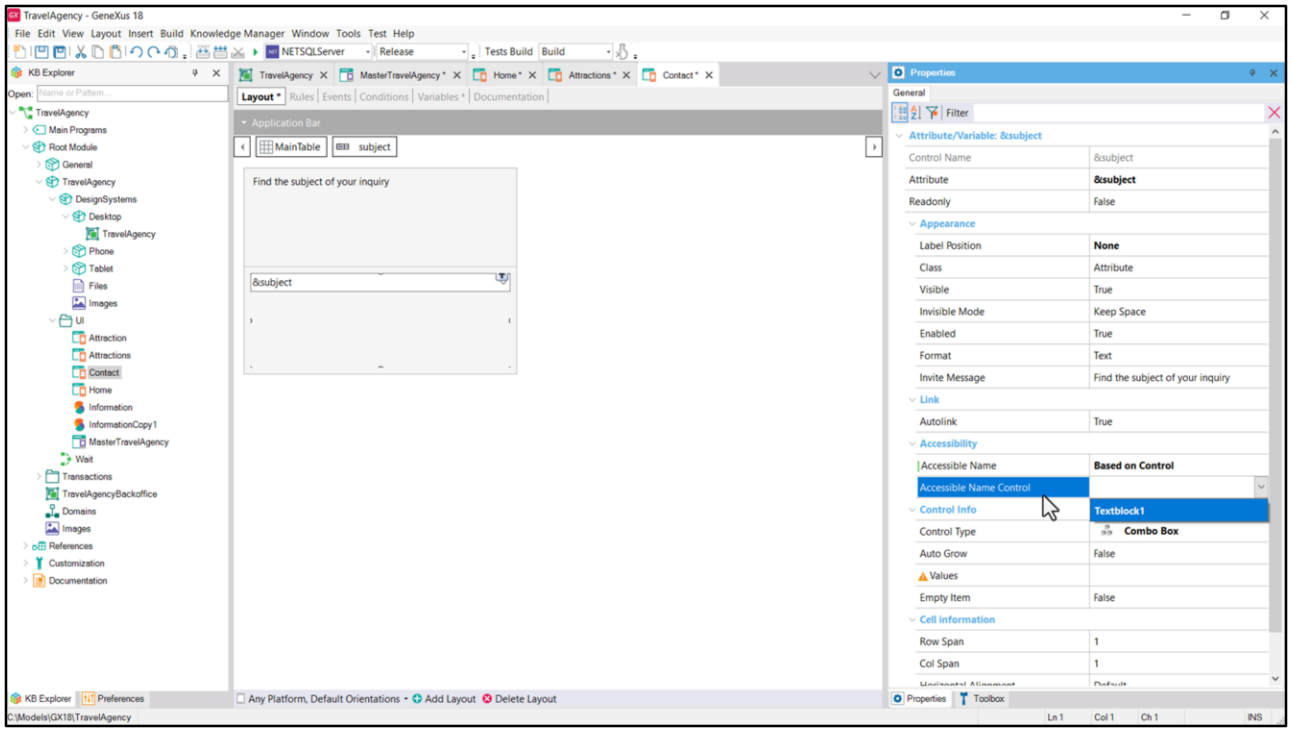
Note, on the other hand, that if we don't want the label to be visible, as we have entered this value, it will be the one assumed by default by the Accessible Name Custom property.



This will be useful for these other variables that do not have a visible descriptive text.

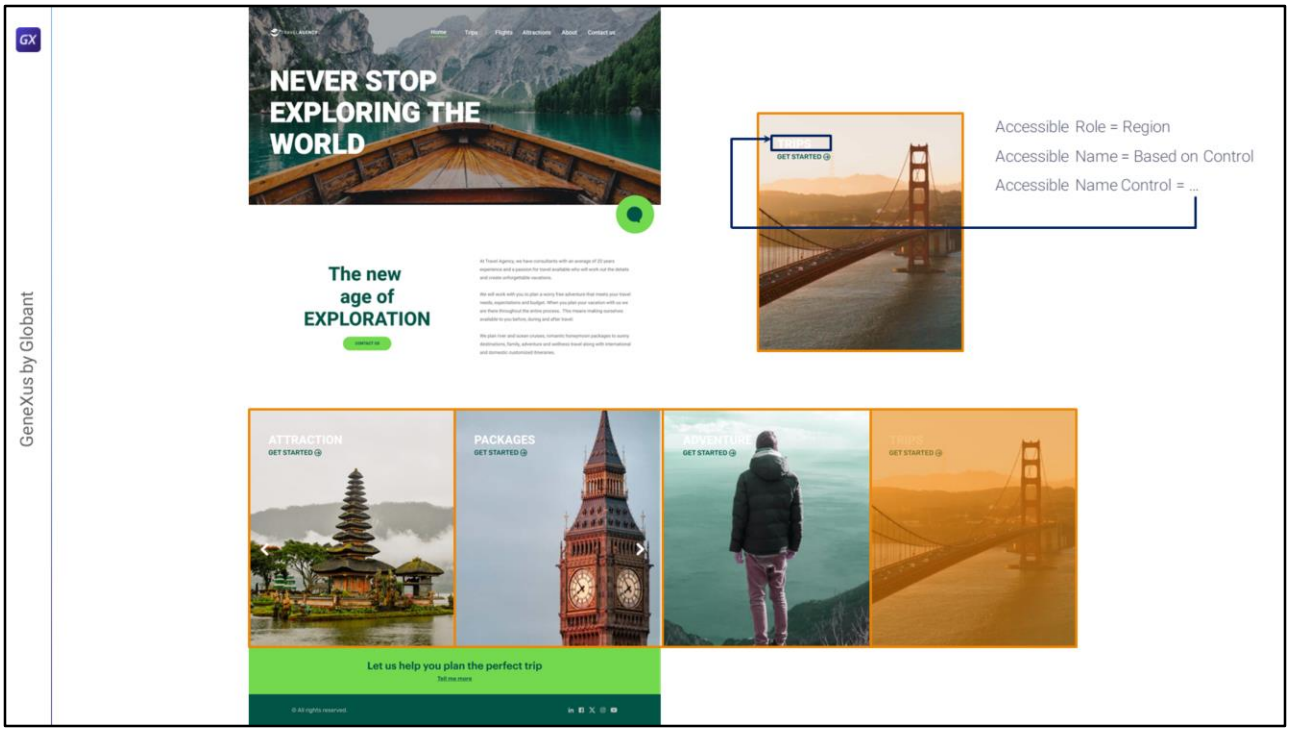


But we could also have thought of implementing the label in a separate way, that is to say, through a Text block control (an option that we do not recommend). We would then insert the control, named Textblock1, specify its caption...



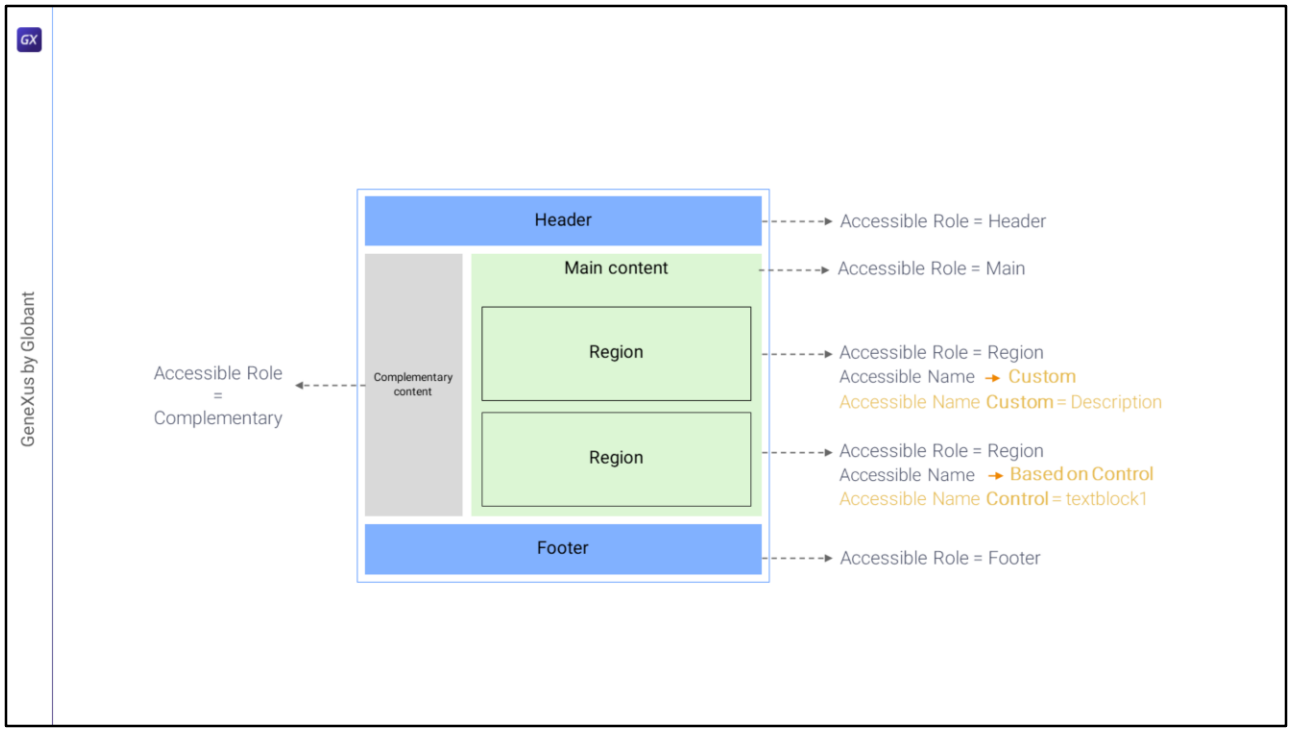
...and what we would have to do is instruct our combo box to take its name for the accessibility of this control.

But, by doing this, when the label is clicked on, the focus will not be placed on the variable as it would happen with the first solution. The reason is that in this case we are dealing with two independent controls, while in the first case the label was part of the field.



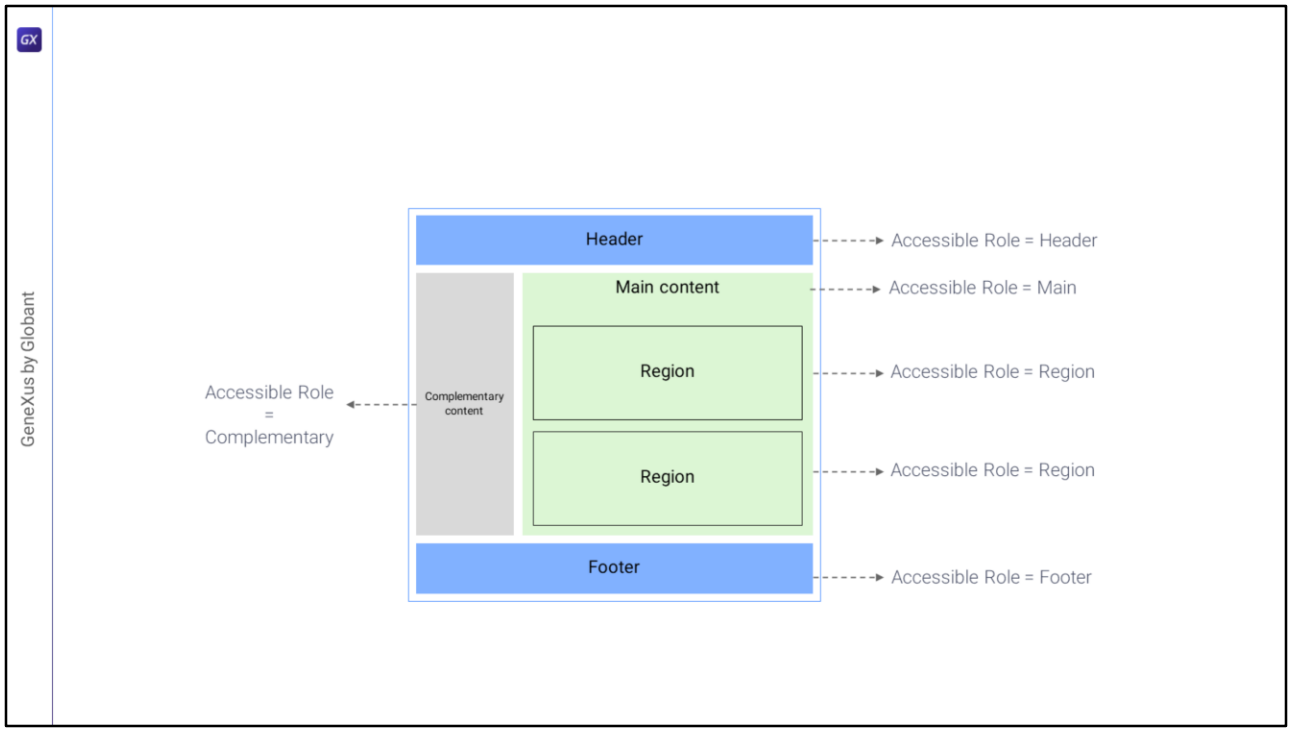
The Based on control property appears for those cases in which the control does not have a native label among its properties, but there is a text block in the layout that can describe it.

For example, the Home carousel will contain 4 cards (2 visible). We will implement it as a horizontal grid where each item will be a card. And each of these cards will be implemented by means of a canvas container (which is like a table that can overlay controls, as we will see). So to each of these canvas we will set the Accessible Role Region property, but as for the Accessible Name, note that we already have it in the layout itself. It is the one corresponding to the title of each Card, which will be implemented as a text block. Therefore, in Accessible Name we will enter Based on Control and in Accessible Name Control we will enter the name of this text block.



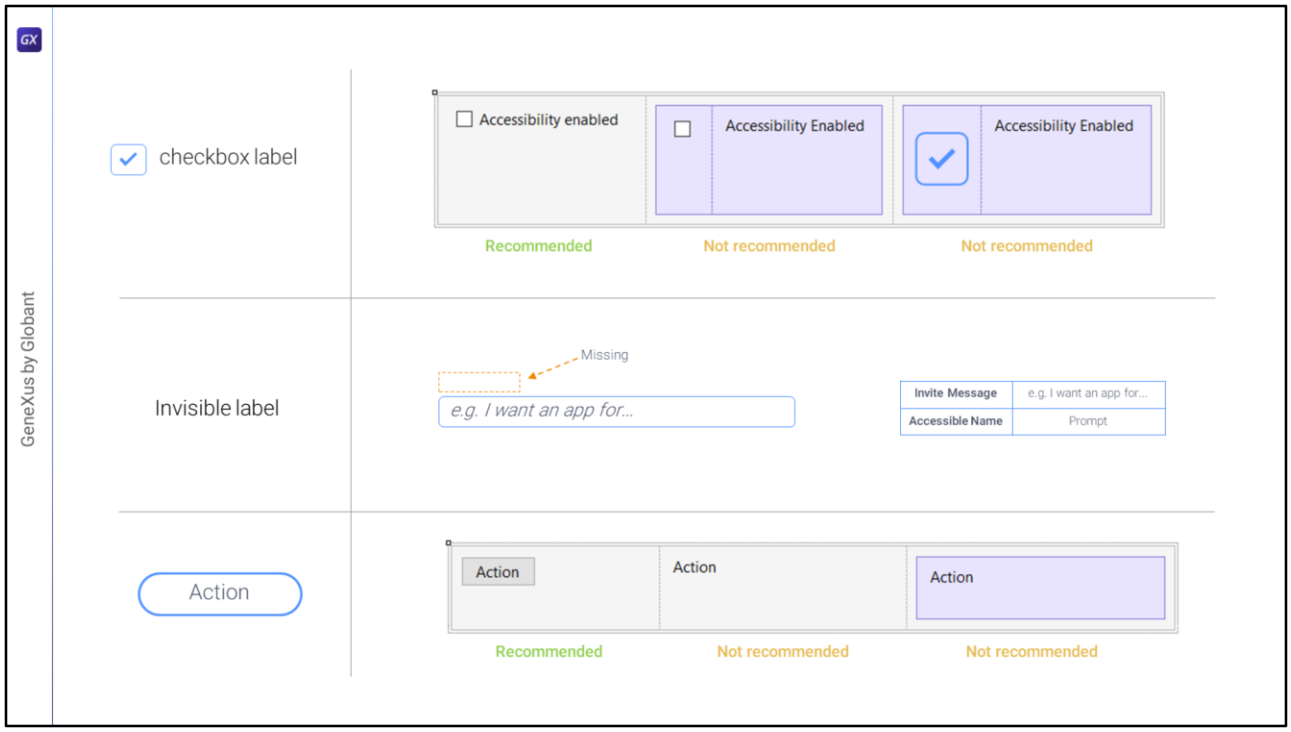
In summary, for the containers (and only for them) we have to indicate the role that each one plays on the screen.

When it is a role that can be repeated, like that of the regions, we give the semantic description to the region through the Accessible Name Custom property if we are going to provide the value right there in a customized way, writing it directly... or through the property Accessible Name Control if we want it to take its value from a text block control that is there.



For the moment, the Role property will only take effect for the Angular application and not for Android or Apple. And it is valid only for containers.

But the other two properties are also valid for other controls such as buttons, images, attributes/variables, as we saw. Also for grids. These are also valid for Android and they are currently being implemented for Apple (their implementation may have already been finished).

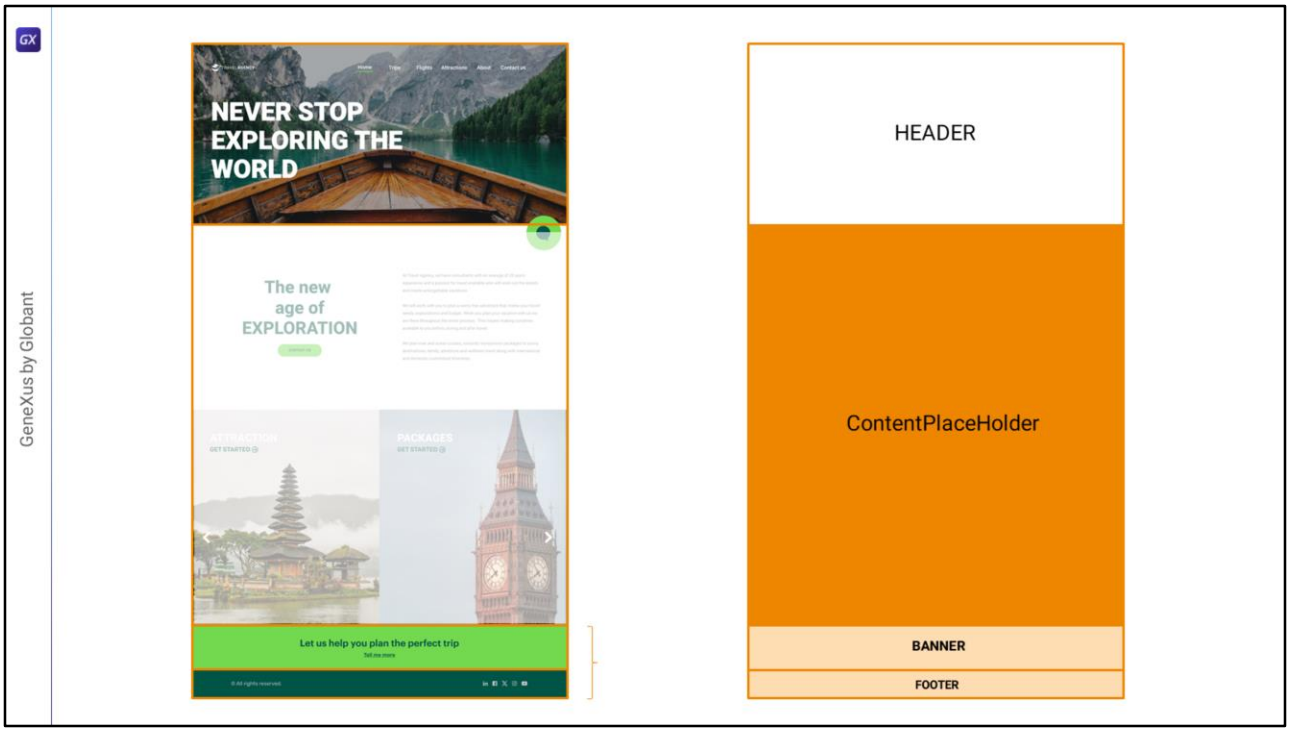


OK, we've seen these recommendations for accessibility: divide the screen into containers with specified roles and then, for the other controls:

- Use the label of the control itself and not a separate text block control. For example, the recommended way to implement a checkbox is through an attribute/variable control with checkbox control type and with the semantic indication through its own Label Caption, not with a separate text block for that, or even using an image control and a text block instead of the native control, checkbox.

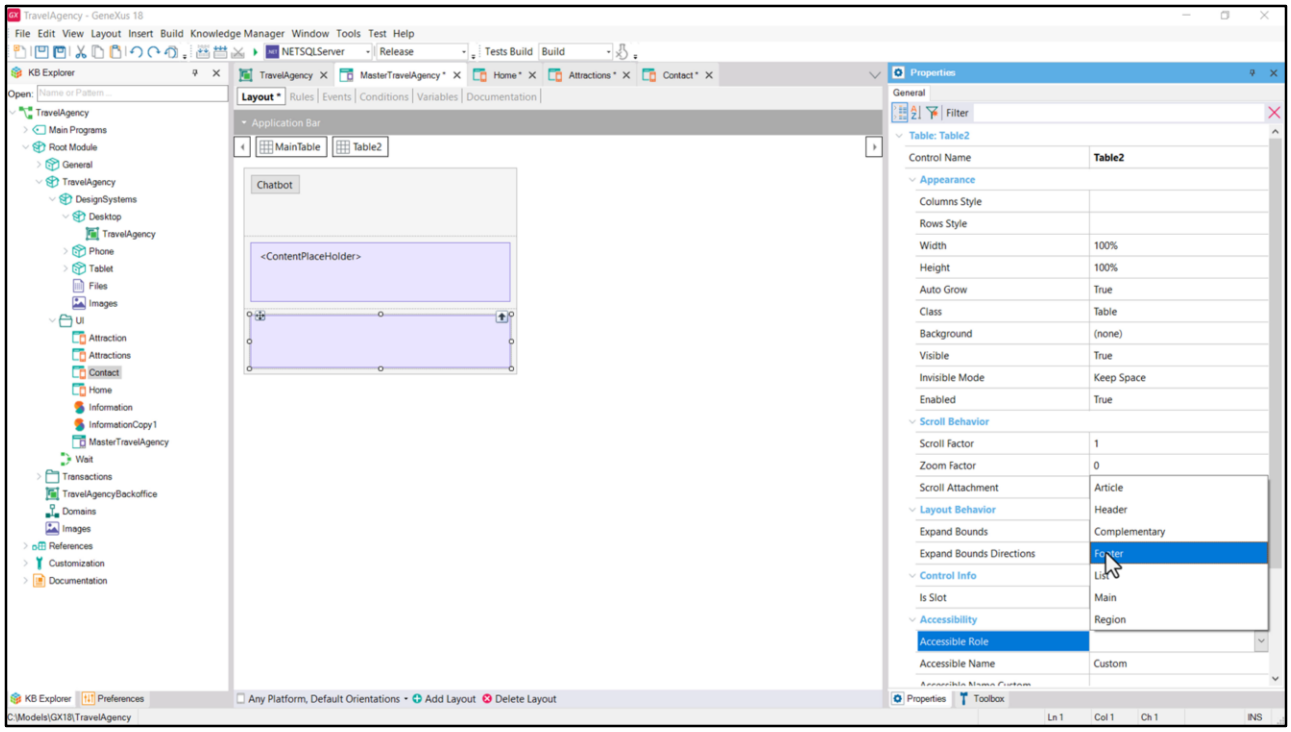
- If the control doesn't have a label, or it won't be visible, don't forget to set the Accessible Name Custom property. In this example, we have a variable that uses an internal message of the field that disappears when the user starts to write in it. It is not the descriptive text, it is a prompt to start entering a value there. That message is provided through the Invite Message property of the control, but it has no impact on accessibility. We must make sure that the field has a semantic value assigned in the Accessible Name property.

- Finally, use buttons whenever we want to implement an action and not another type of control with an associated event, such as a text block, a text block inside a table, or even (although it is not shown here) an image with an associated event. In any of these cases, the semantics of the button is lost, which is precisely the implementation of an action. For actions, we should always use buttons, which are then given the desired aesthetics, which can have no borders, no box, only text, or only an image, or both, but where it is guaranteed that it is a "button".



Well, we will put all this into practice in the following videos, as we implement the layouts.

Returning to the implementation of the Master Panel, in row one we will have to implement the Header, and in rows 3 and 4 these other parts, which Chechu called Banner and Footer in Figma. However, a Banner at the level of web semantics is understood as a Header, and actually these two sections work more like a Footer...



So we will leave a single row in which we will place a table with Footer role.

The implementation of this part is simpler, so let's get into the more complex part, which is the Header.

We will continue with it in the next video.


GeneXus by Globant
GeneXus™
by Globant
training.genexus.com