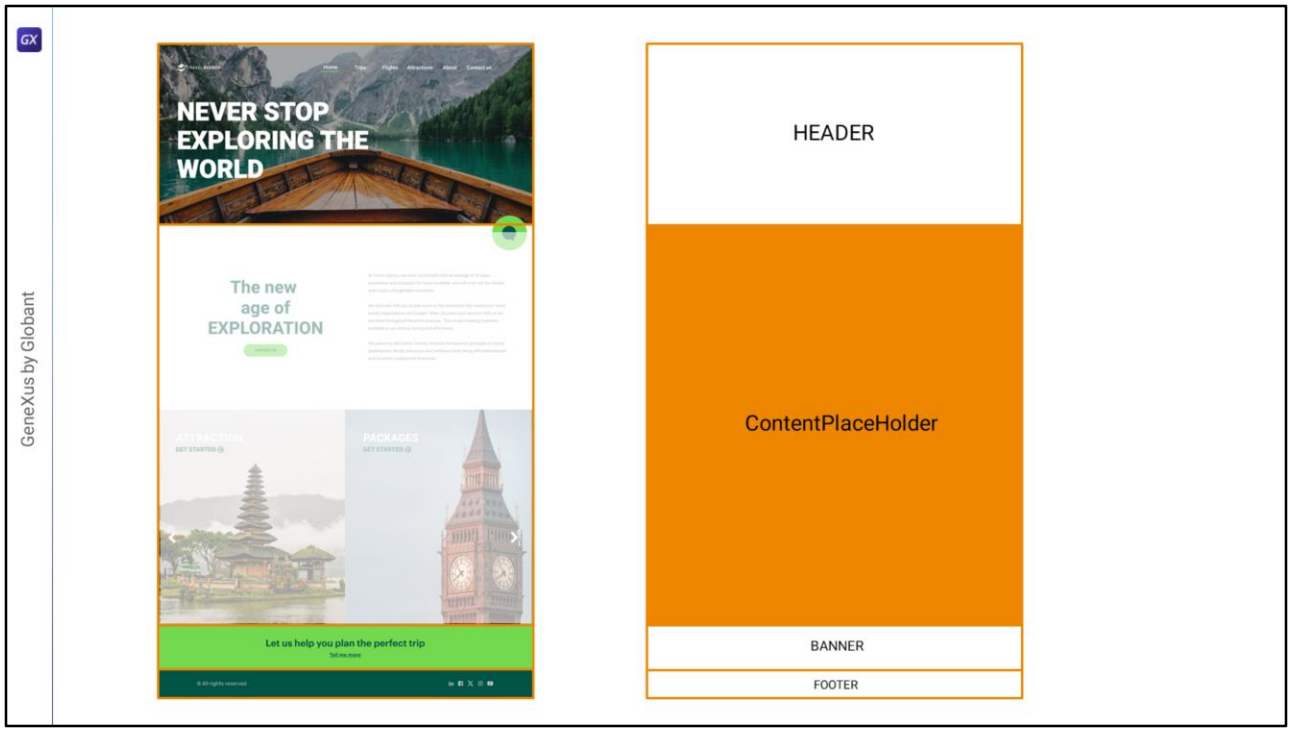
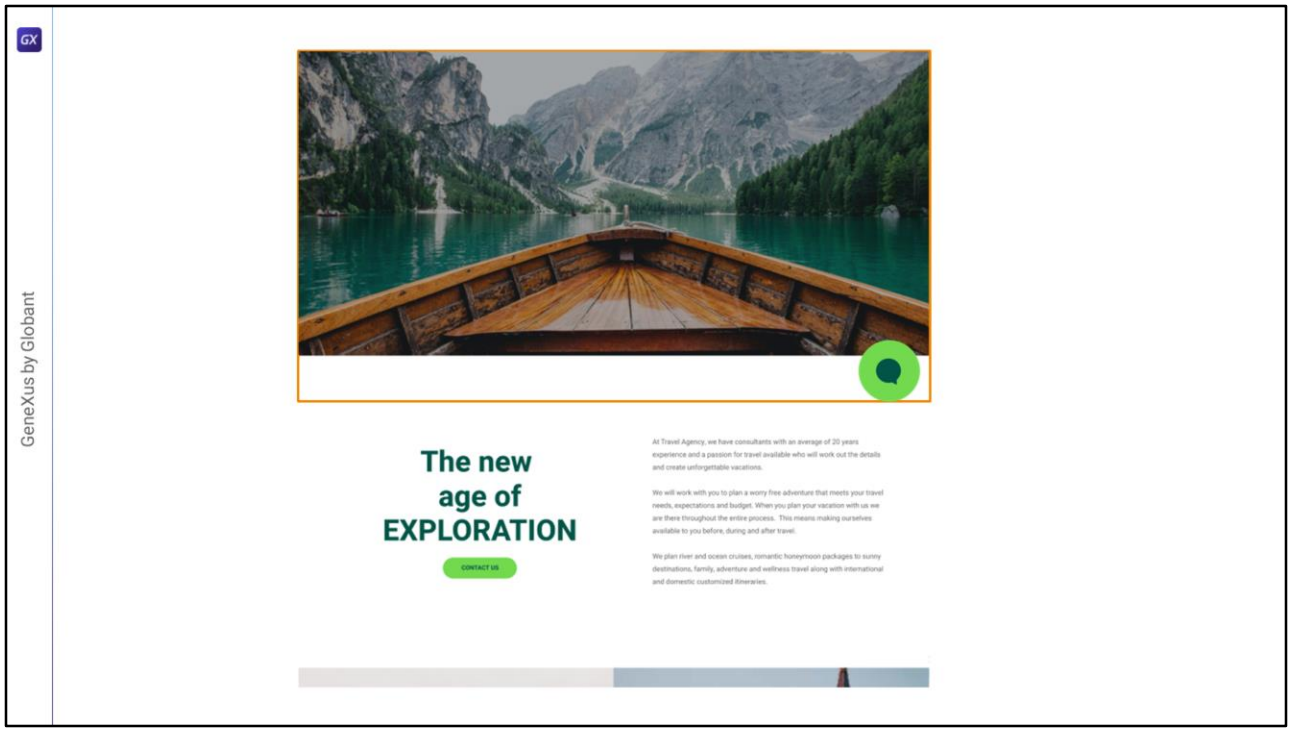
GeneXus by Globant

Canvas Control for Header

Cecilia Fernández



Our objective now is to implement the Master Panel Header.

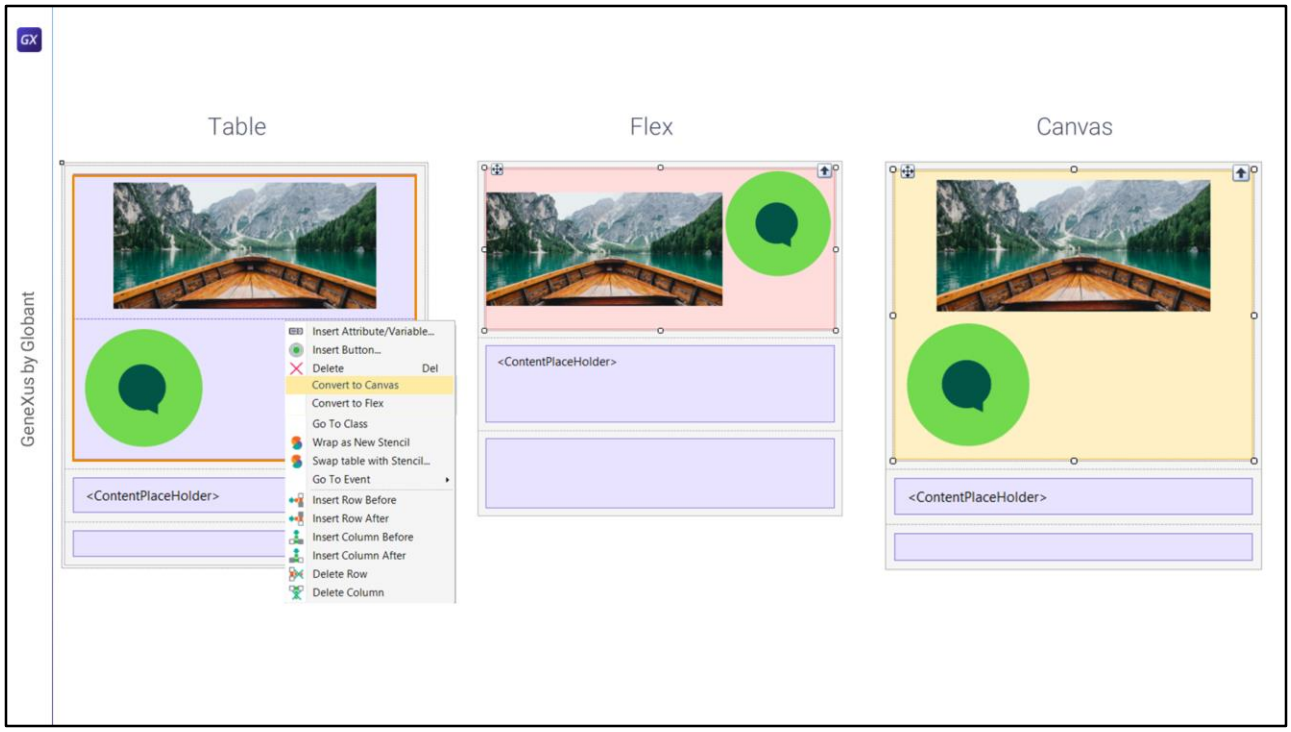


Layouts, by default, contain a single layer, in which each control occupies its own space, without overlapping the others. Each. control has its own space reserved.

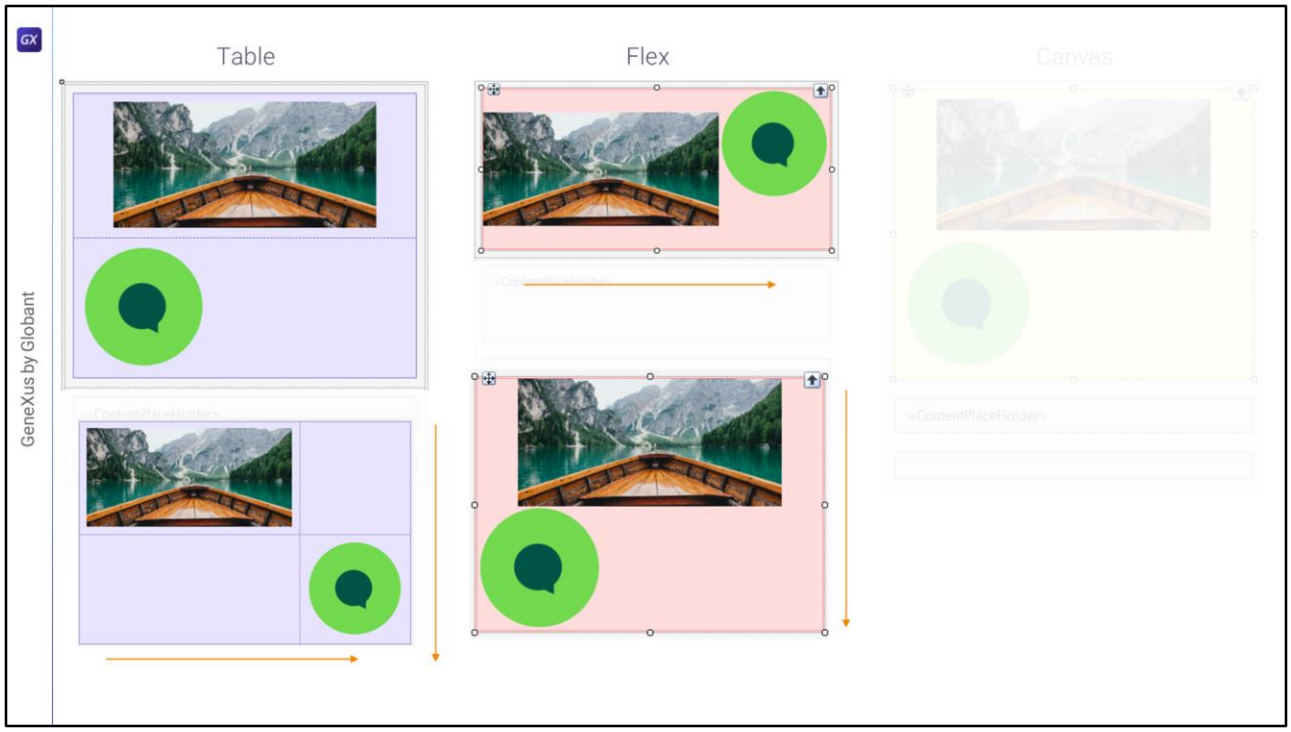
So, for example, if we start by modeling in the Header only the background image and the chatbot button, to overlay them we will have to place each one in a different layer.

When there is a table in row 1 of the Main table (to give it the “Header” role for accessibility, as we saw in the previous video), if we drag into it an image control, for example, for the background image and a button control for the chatbot (with this image), they will necessarily be either in two consecutive rows or in two consecutive columns; that is, each control will occupy a separate space (because these rows or columns do not overlap).

Therefore, a control can only start where the other ends, either vertically, horizontally, or horizontally and vertically (in the case of a 2x2 table where the controls are placed diagonally).

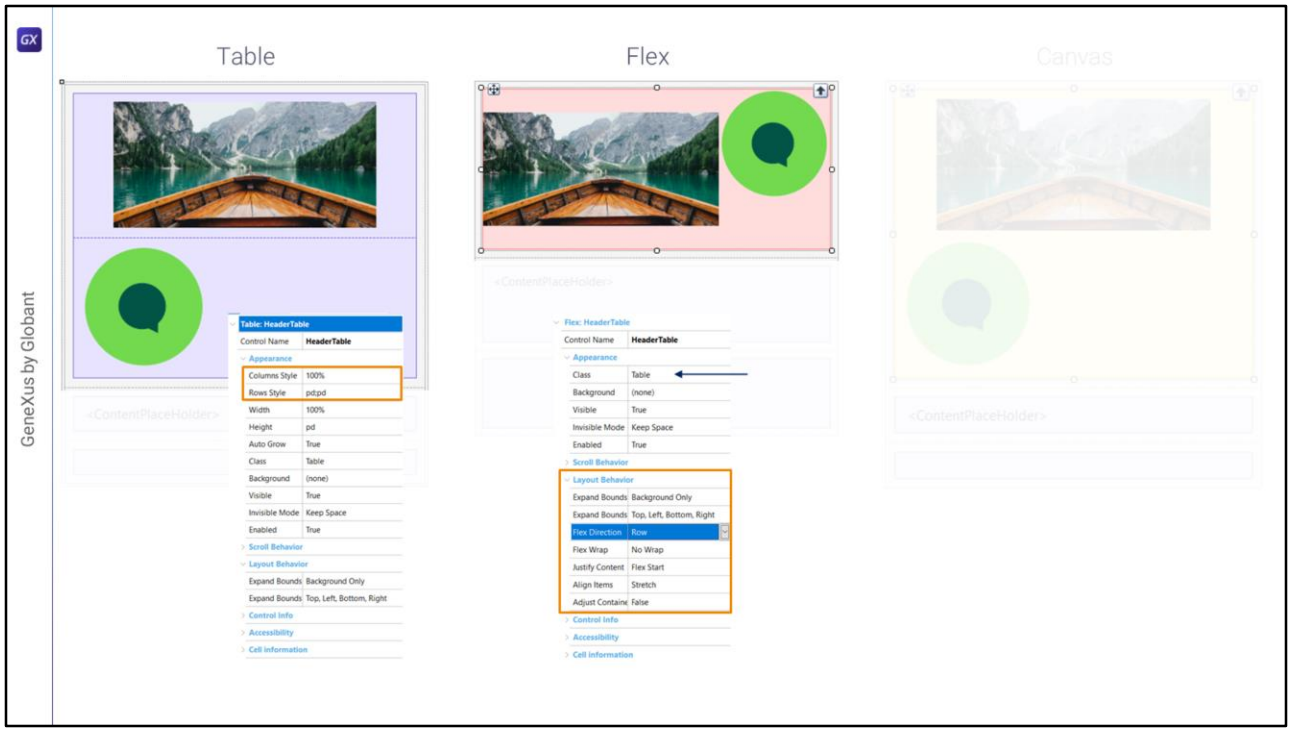


We could turn the table into a Flex container... or into a Canvas container.

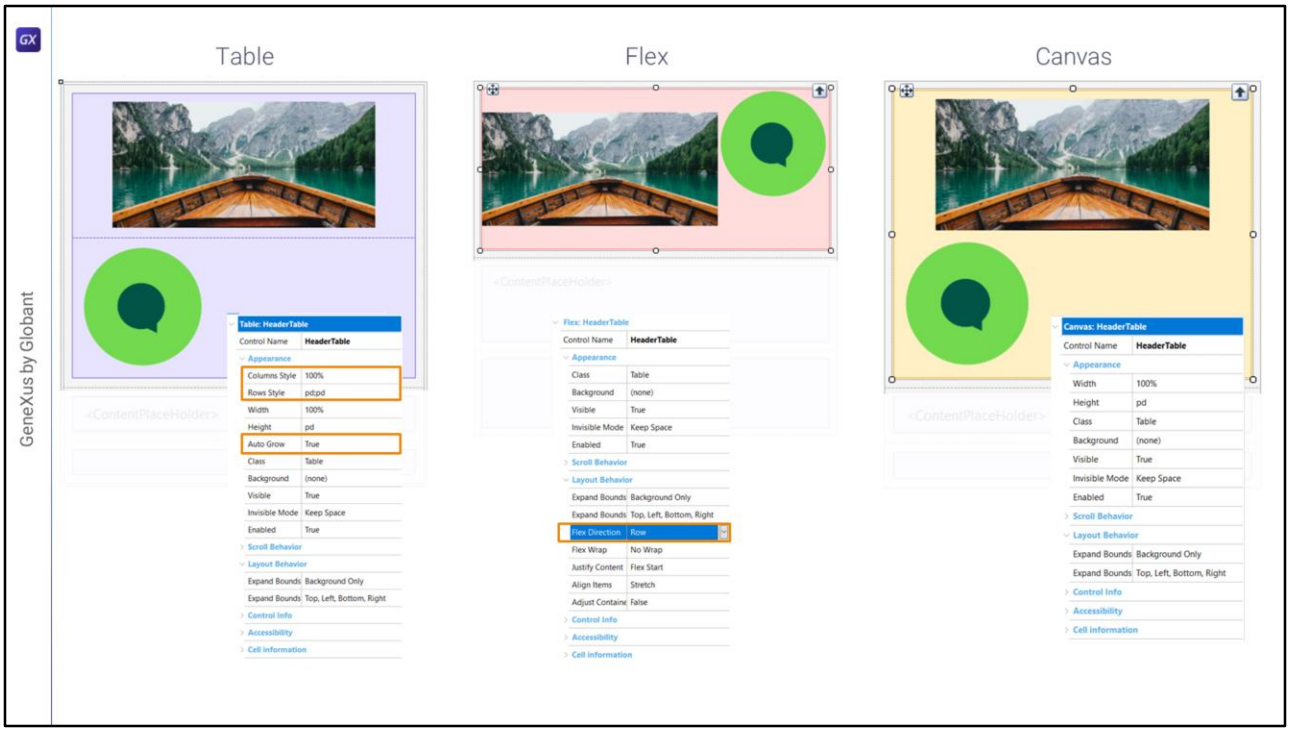


If we turn it into a Flex container... the table, which is a two-way control (it can have many rows and many columns) becomes a kind of flexible one-way table:

- A row (Row direction) with n consecutive controls, either from left to right or from right to left,
- Or a column (Column direction) with n consecutive controls.



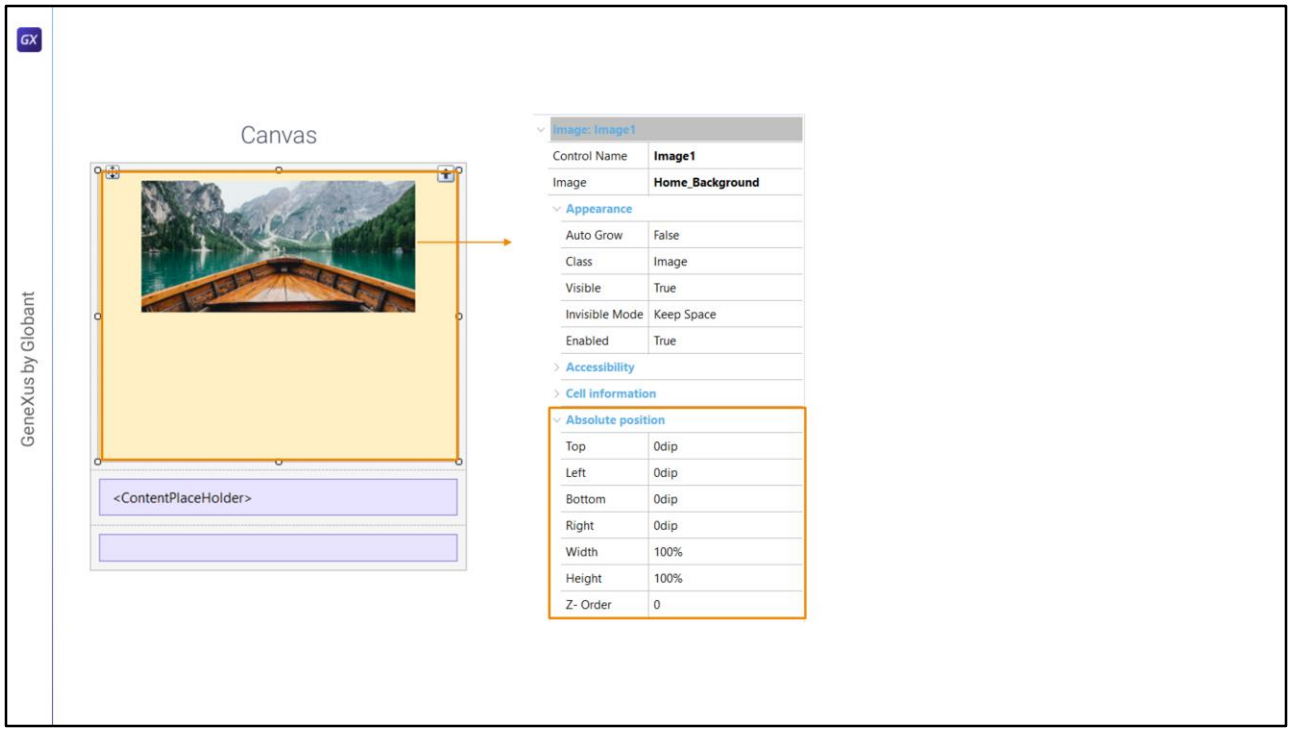
In this way, we see that the Columns Style and Rows Style properties of the Table control disappear, and instead, these ones are displayed. In particular, we see how the Flex Direction property determines whether the container's controls are placed along a row or a column. The controls included there will be placed one after the other, and we will have to set the spacing between them through properties, such as the gap property, at the level of the class that we associate with it.



But to overlay the internal controls of the container we need the other type of container, the Canvas.

If we compare the properties to the table control, we see that it has 3 fewer properties: Columns Style and Rows Style disappear and there is no replacement (note that in Flex there was Flex Direction). Here there is none because the concept of row and column disappears completely. It will be a container where everything that is placed inside it will be in an **absolute** position relative to the **edges of the container**, and it is independent of the existence of other controls that are also there. The concept of rows and columns loses meaning.

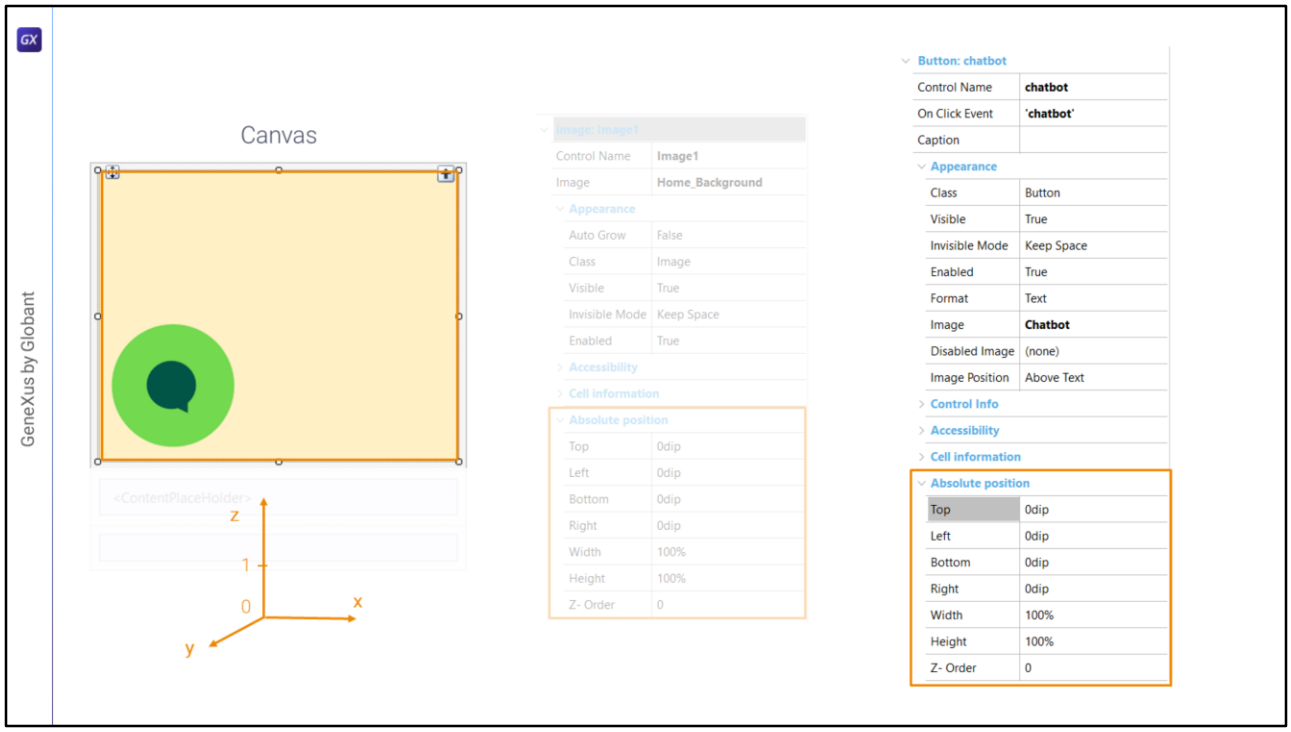
The Auto Grow property also disappears, which for the Canvas will always be True.



What will happen, then, is that all the controls inside the Canvas will add properties for absolute positioning. Each one will be in an absolute position relative to the edges of the Canvas.

And absolute positioning means that it doesn't depend in any way on the other controls that are there (inside or outside, of course); only the Canvas matters as a reference.

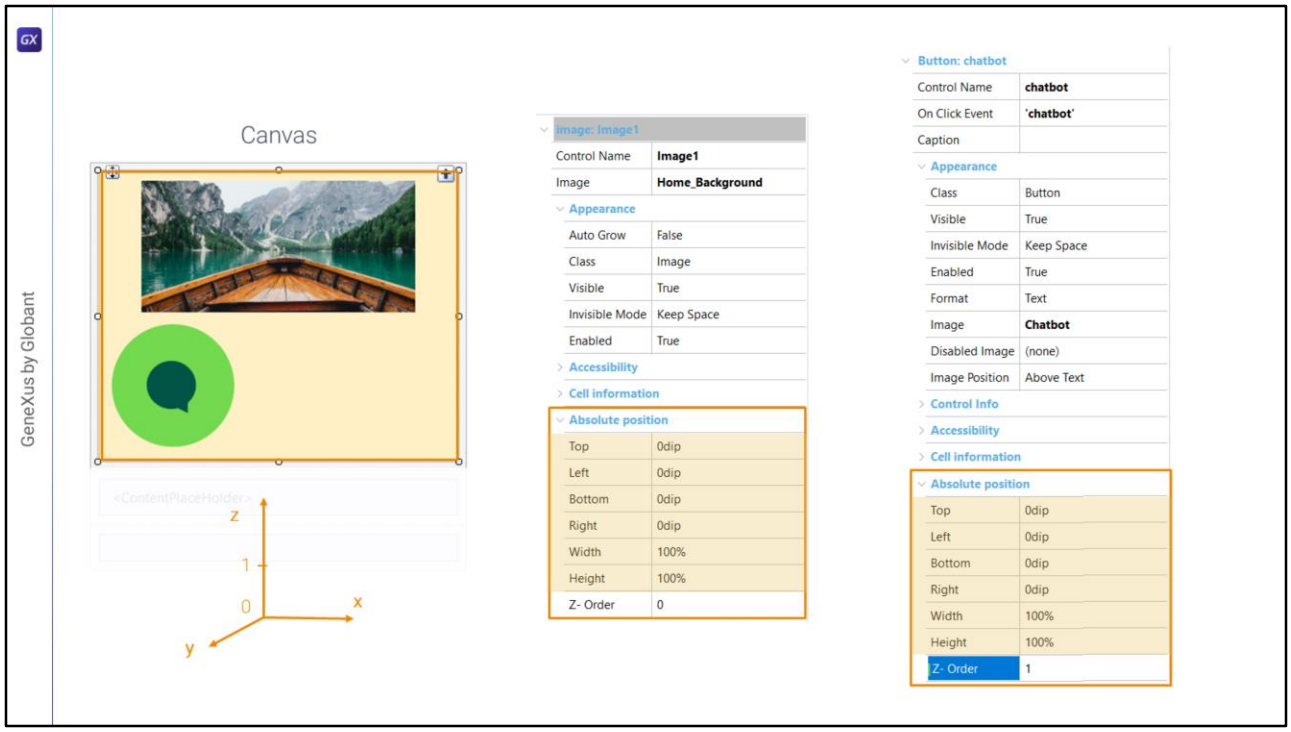
That's why when we're positioned over the image control, this **Absolute position** group is displayed...



And the same happens for the button control.

The last property is the easiest to understand: it determines the layer in which the control will be placed.

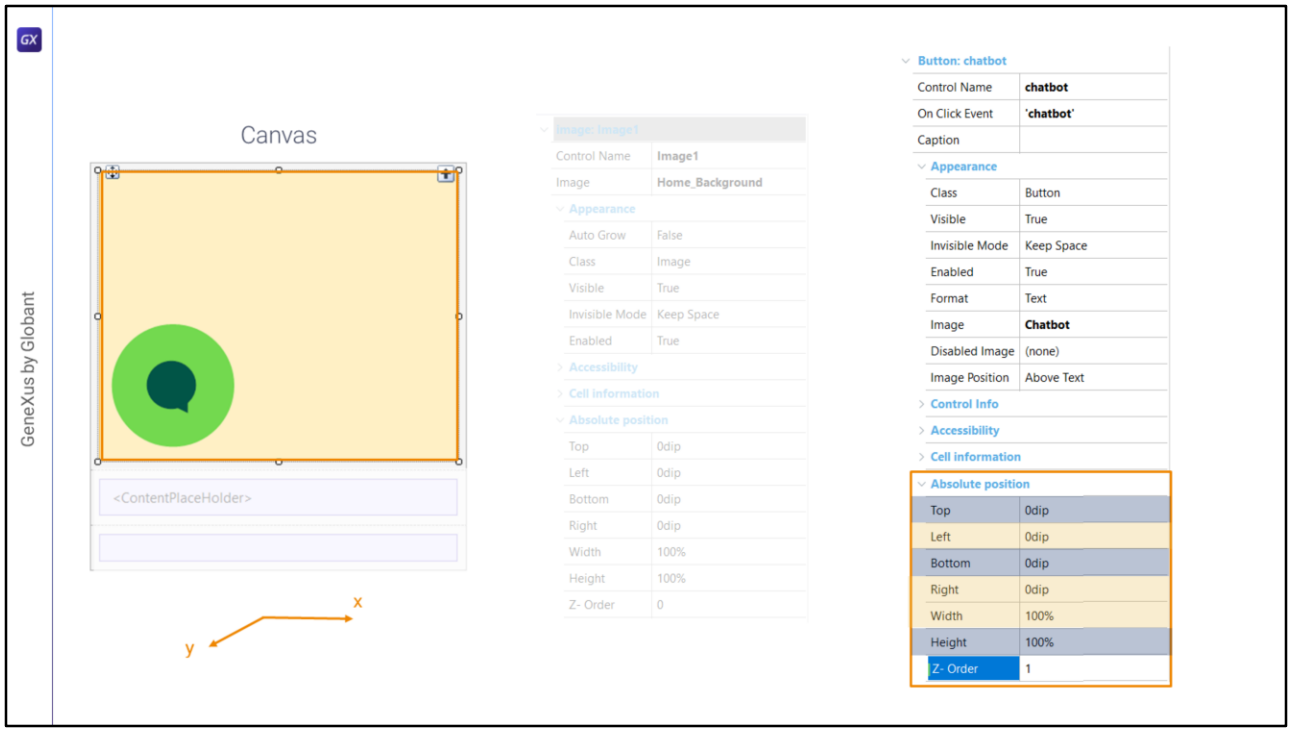
To achieve the overlay, the other dimension –z– is displayed; it is precisely the one that introduces layers. The deepest layer will be the one with value 0, and from there on we can number the layers.



In our case we would leave the image with layer 0 and place the button in layer 1 so that it is on top.

Now let's take care of the rest of the properties that are related according to two axes: x and y.

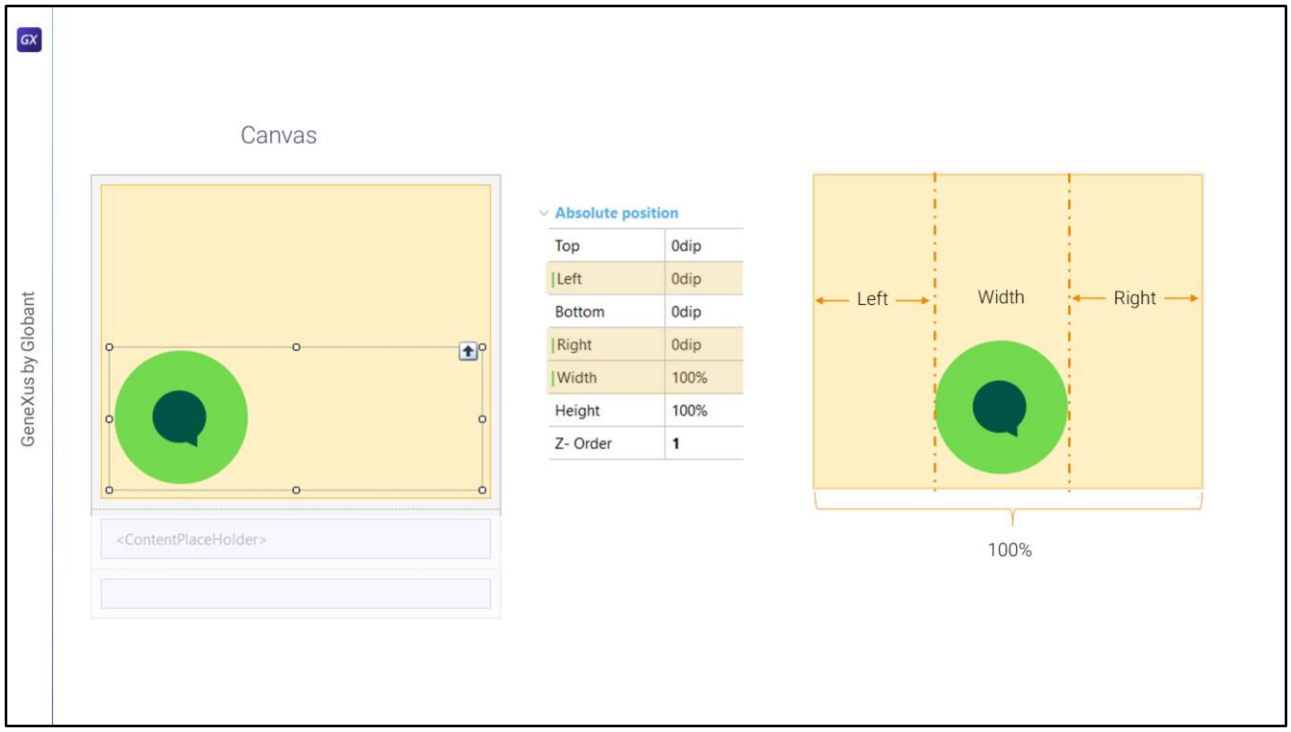
As we said, as far as the positioning of each control within the Canvas is concerned, the sibling controls do not matter at all. Only the control itself and the boundaries of the Canvas matter.



So let's analyze the Chatbot button.

These properties are relative to the positioning of the button regarding the x-axis. And these regarding the y-axis.

The ones for each axis are related to each other, so when we set 2 of the properties, the 3rd one will be automatically determined as well, as a percentage.

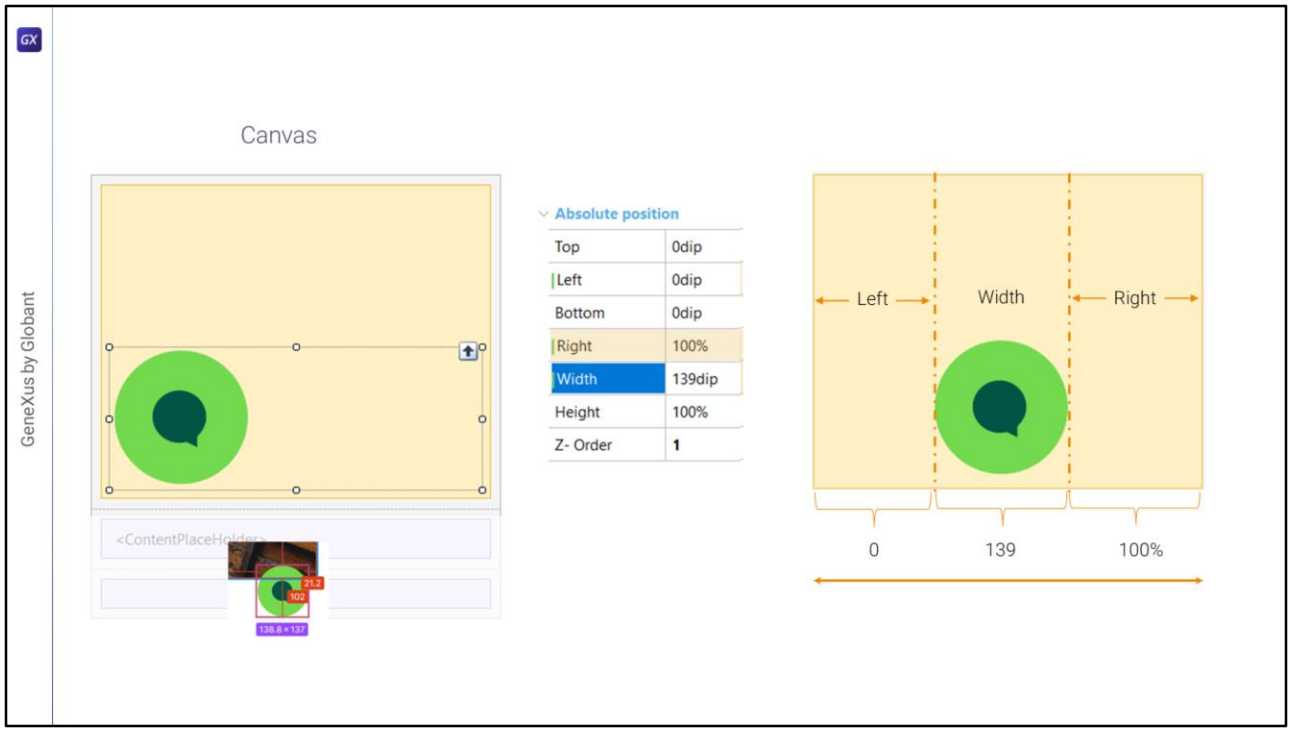


Let's see it with those of the x-axis.

This is how they will be initially, indicating: 0 dips relative to the left edge of the Canvas, 0 dips relative to the right edge, and therefore that of the width of the control will be 100% of the remaining width of the Canvas, which in this case will correspond to the total width.

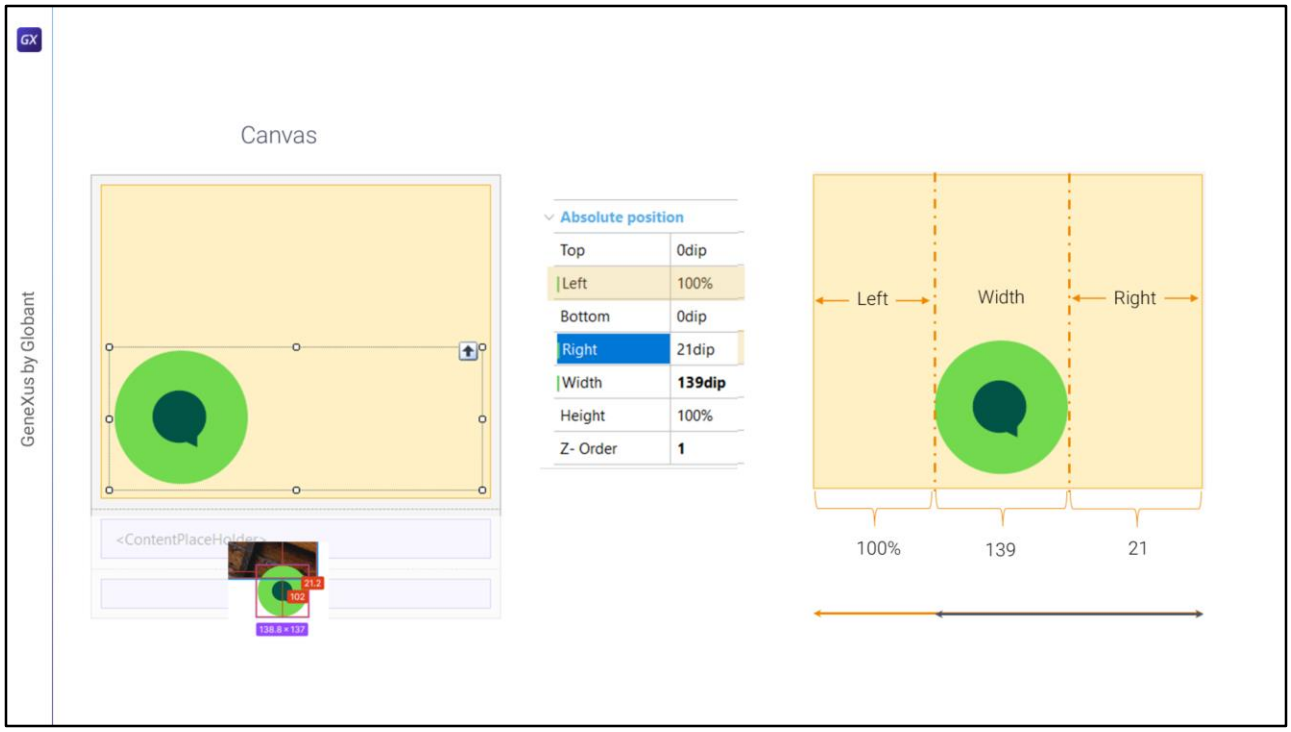
In short, it is as if two columns were placed on each side of the control, and the left one was given that of the **Left** value as width, the right one was given the **Right** value, and the middle one, where the control is placed, was given the **Width** value.

Clearly, the third one is determined by setting 2, because added together they should give 100% of the width of the canvas.



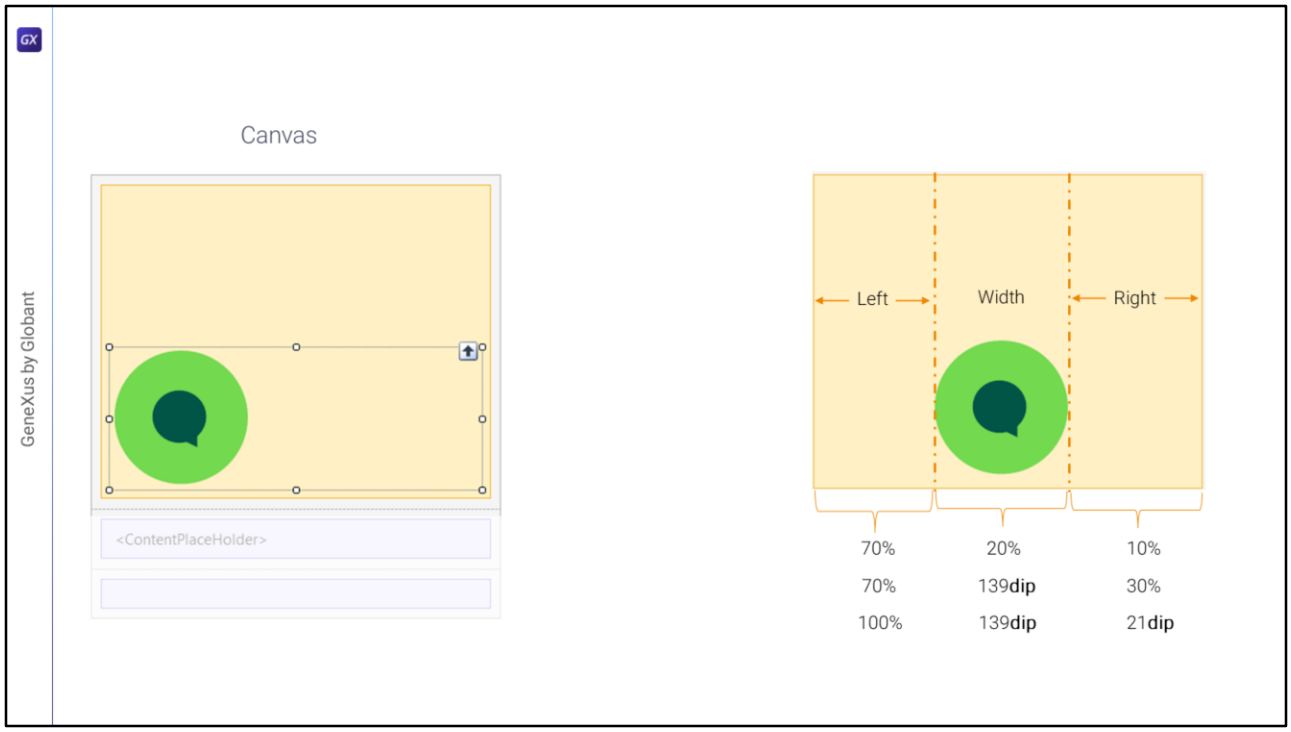
If we look in Figma for the width of the button and the distance that separates it from the right end we see this.

So, if we set the Width to be 139 dips (or pixels) when we leave the field we will see this change automatically in the Right property. What do the current values indicate? That the button control will be 0 dips from the left edge of the Canvas, that is to say, next to it; it will have a width of 139 dips, and what will distance it from the right edge of the canvas will be 100% of the remaining size, relative to the width of the canvas.

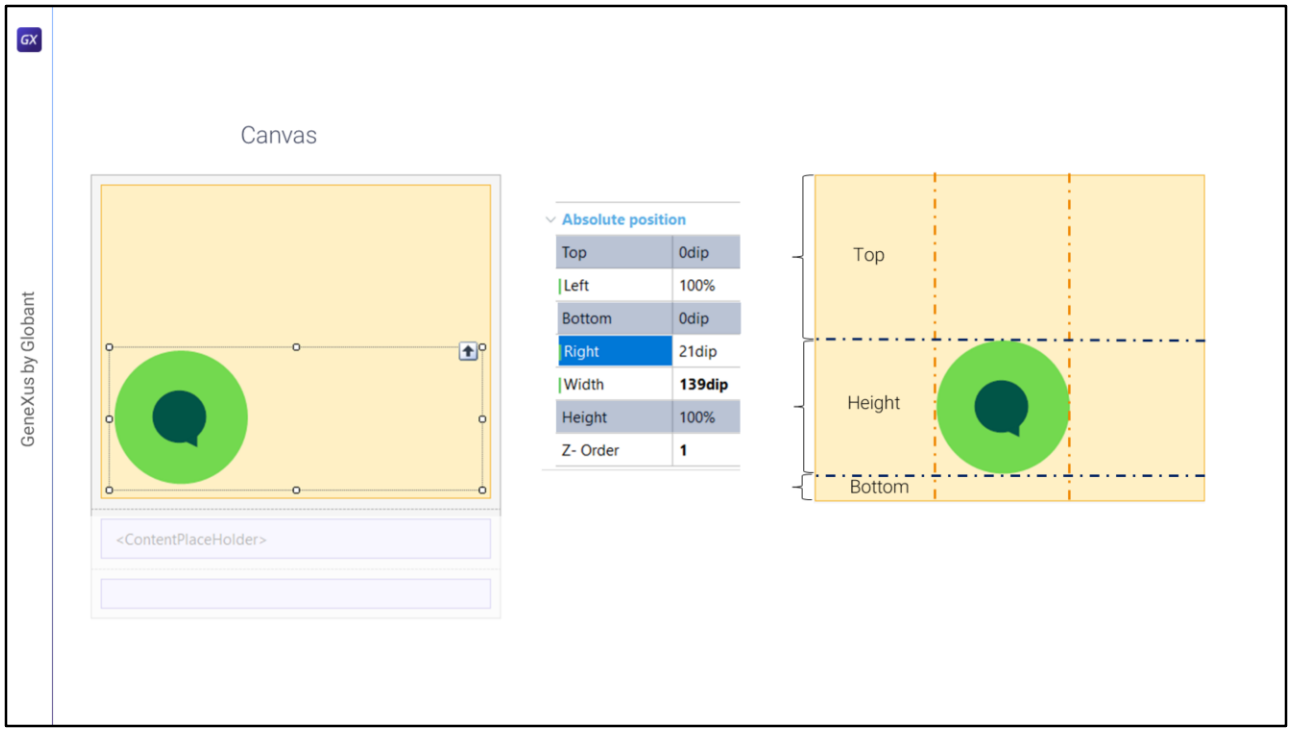


However, we want it to be 21.2 dips (rounded to 21) from the right edge of the Canvas. Therefore, by specifying that distance from the right, it automatically changes the distance relative to the left edge to 100%.

Again: it will be 100% of what results from subtracting the fixed values of 139 dips and 21 dips from the width of the Canvas.

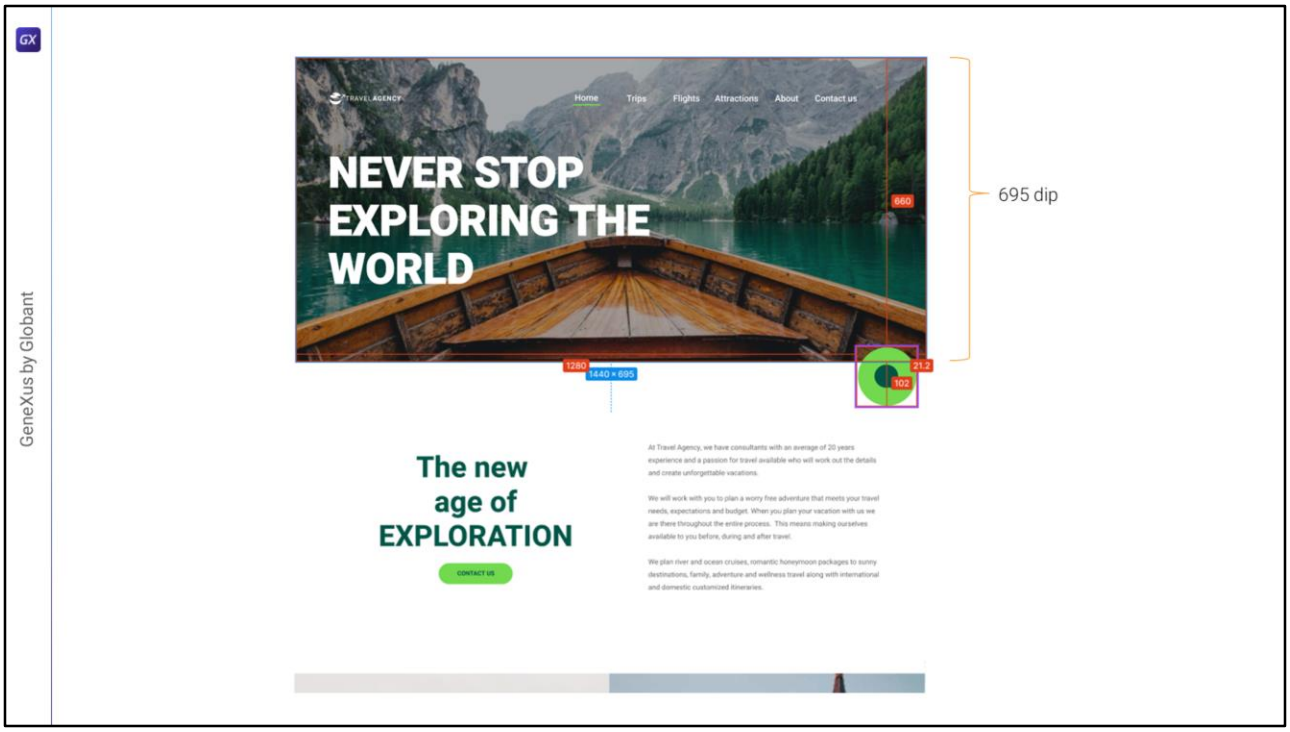


Left, **Width** and **Right** can be all 3 in percentages (and must add up to 100%), two of them can be in percentages (and must add up to 100%), or only one, in which case it will always be 100%.

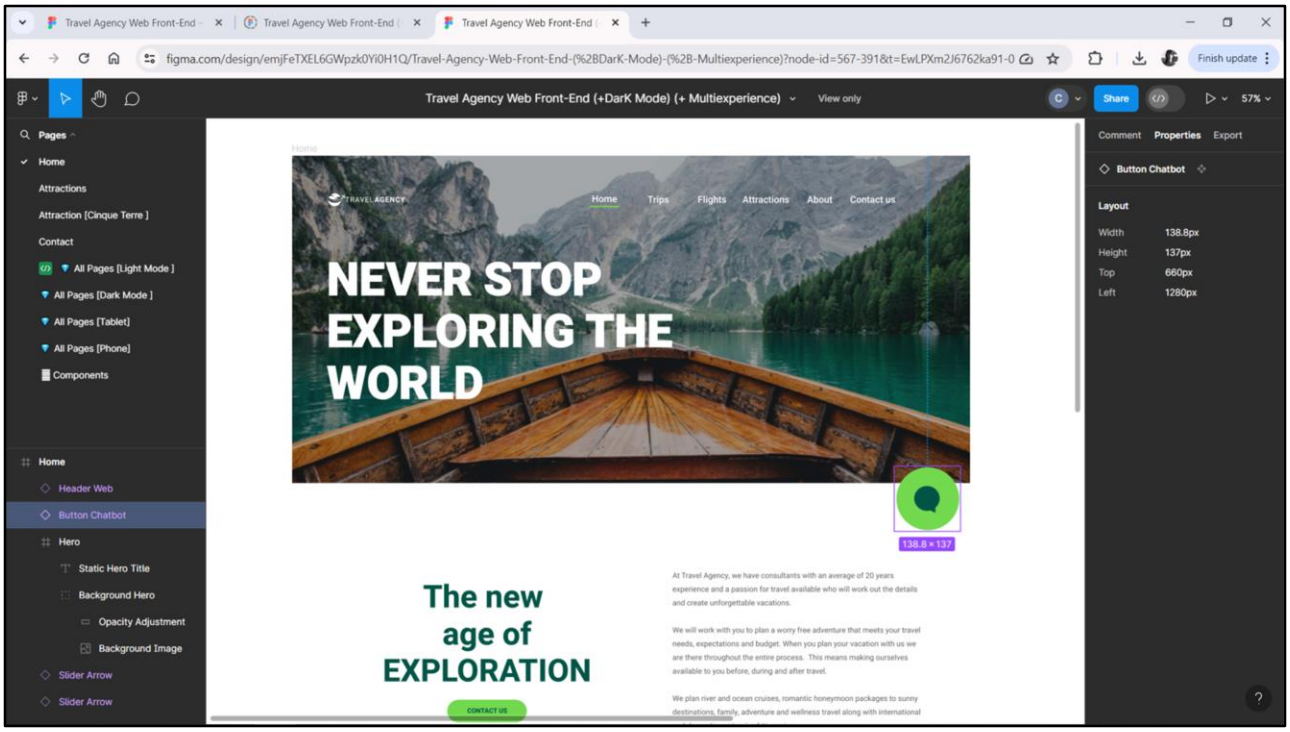


The same type of analysis applies to the other axis, the y-axis.

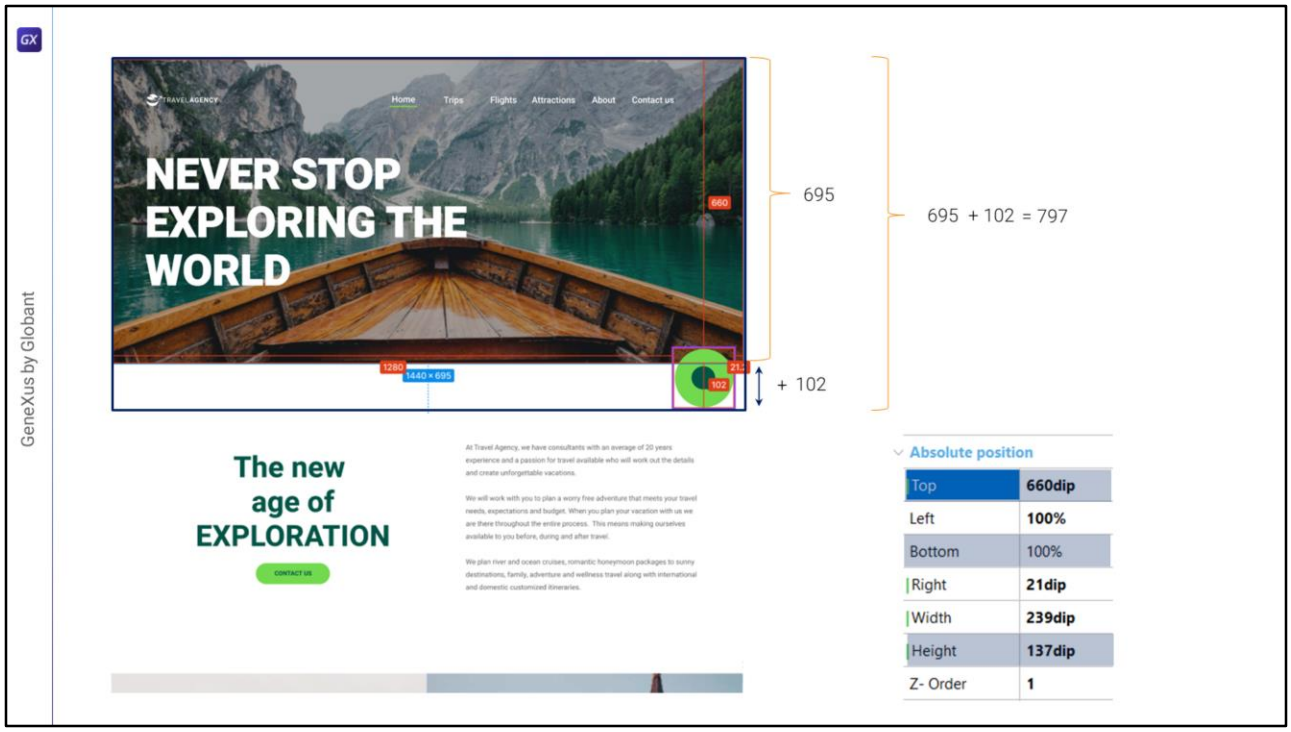
So we will have **Top** for the distance from the top edge of the canvas, **Height** for the height of the control and **Bottom** for the distance from the bottom edge.



If we look for the dimensions in Figma, we see that the image will be 695 pixels or dips high, and that the button will start vertically at 660 dips or pixels from the top edge of the canvas.



On the other hand, the height of the button will be 137 pixels or dips....

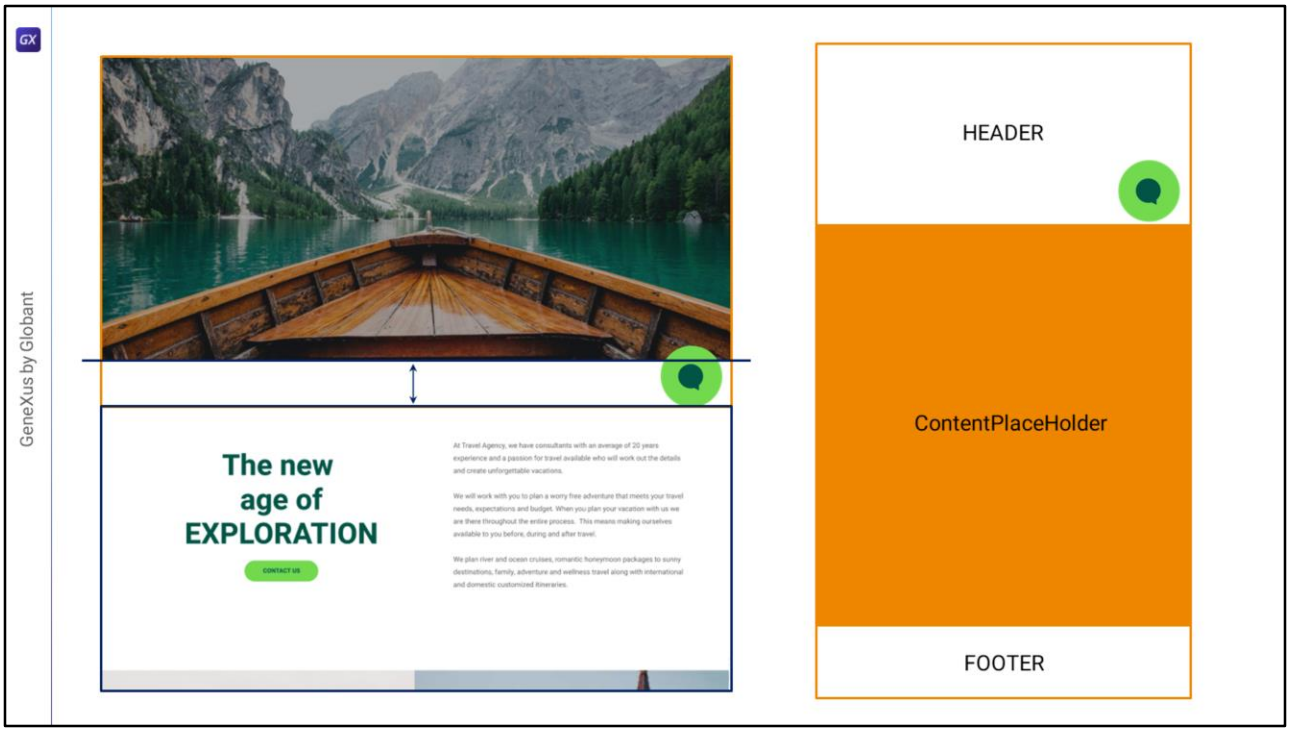


We will define the properties as follows: **Top**: 660, **Height**: 137, and as we know, by setting 2, the third one, **Bottom**, will necessarily be 100%. 100% of what? Of subtracting from the height of the Canvas, which we haven't said what it is yet, 660 corresponding to Top and 137 corresponding to Height.

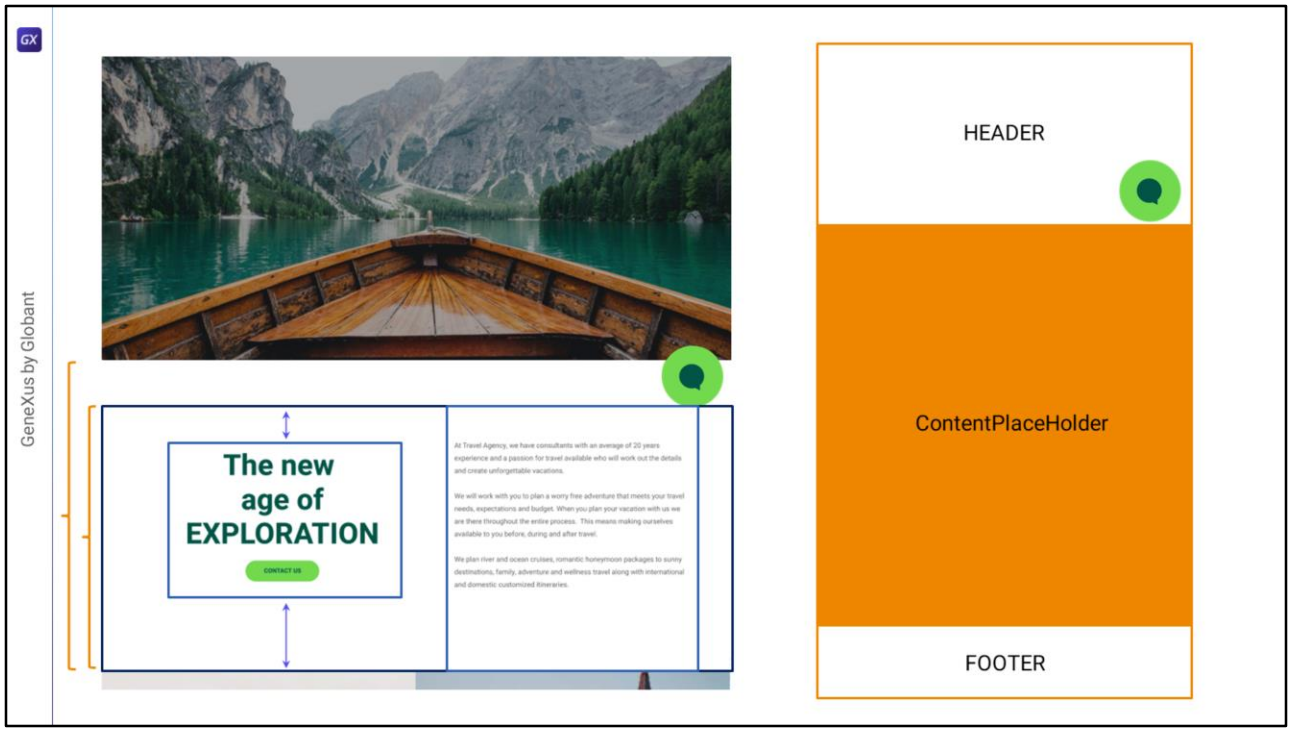
What will be the height of the Canvas? We must specify it. We could think of setting the height of the canvas to 695 dips, so that the button overflows its limit. And then that Bottom ends up being a negative number: -102 dips in this case.

However, even if we set that Height of 695 for the Canvas, **it will stretch it**, because internally every canvas has the **Auto Grow** property set to **true**; this will cause it to stretch to contain all its controls in the vertical direction. Therefore, the height of the Canvas will end up being of 695 plus these 102. This means that this Bottom of 100% will end up being of 0 dips, that is to say, the button will be placed vertically on the lower edge of the canvas.

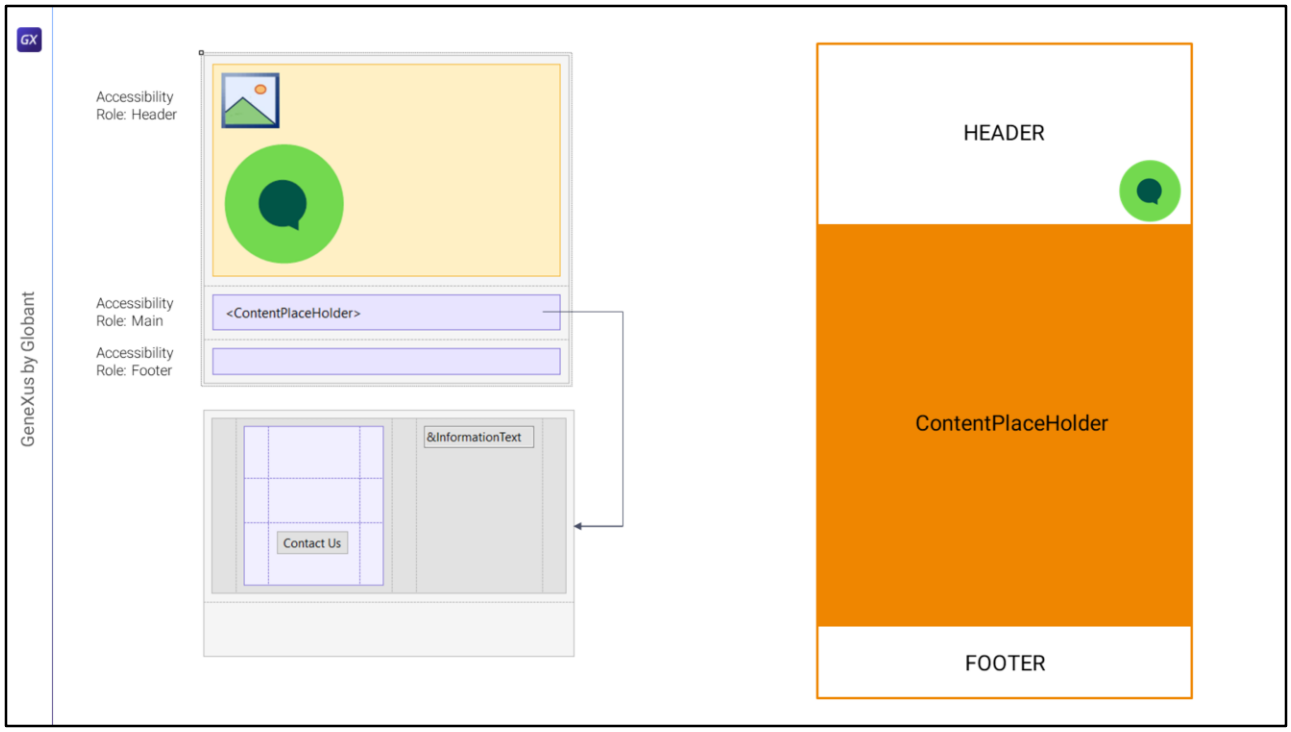
So, regardless of whether we assign it 695 dips, the height of the canvas will actually end up being the sum of this height and this height, which gives 797 dips.



So far so good. But note that this solution comes with a problem. If the Header expands to end where the button ends, then we should consider that the layout of any panel loaded in the ContentPlaceHolder will start here... and not here.

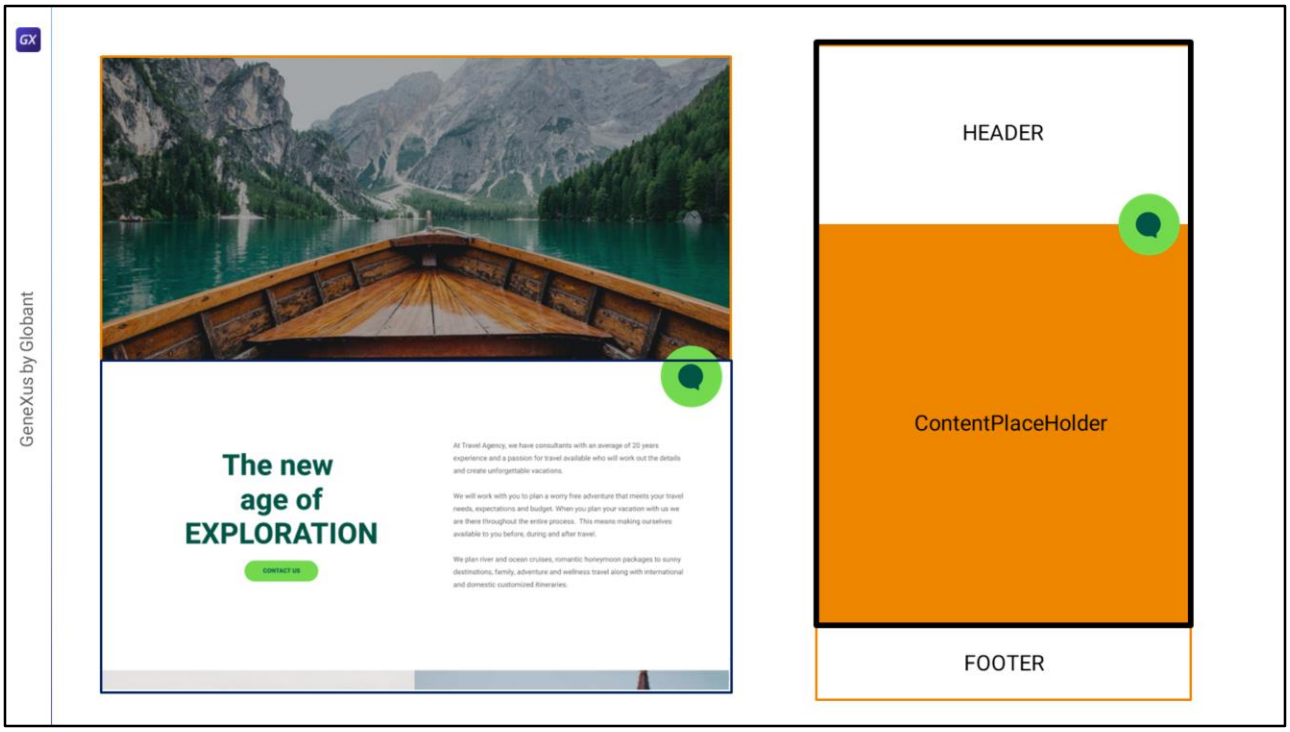


Then, if we want to keep this Master Panel structure we will have to modify the implementation of this section, changing the height of this table to this one, instead of this one...
...and the vertical alignment that we had given to this other table, since now it can no longer be in the middle...
Also, we will have to change the top and bottom margin that we had given to this textblock.

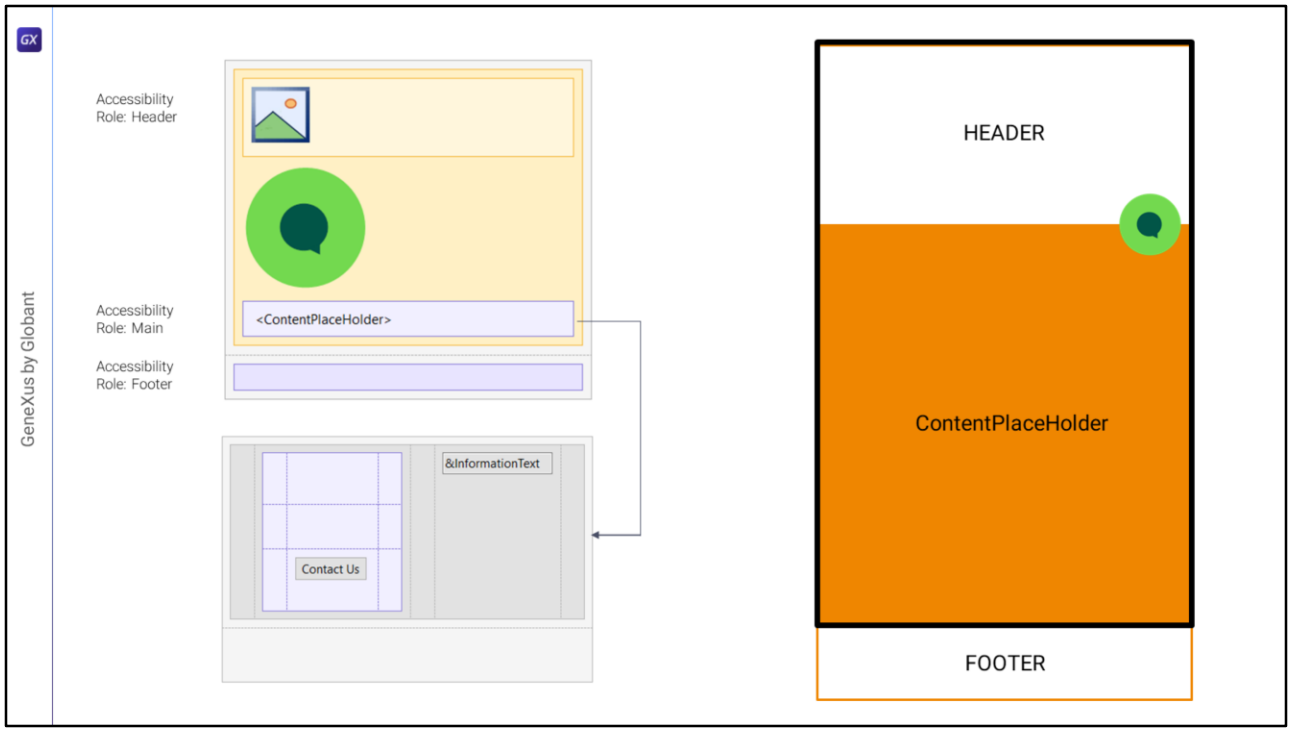


This implementation of the abstract layout of the Master Panel will look more or less like this: with a main table of 3 rows: in the first one, of Accessible Role Header, a canvas; in the second one, a table with Accessible Role Main with the content placeholder, and the third one for the Footer...

And here we see the implementation of the Home panel or the Attractions panel... where we will have to change the way in which we placed these controls vertically and the height of this table...



The other alternative, which wouldn't require making these changes, and which is more conceptually appropriate, would be to consider that since there is indeed a triple overlay –of the Header section, the button, and the ContentPlaceHolder– actually we should change the structure of the Master Panel and use a Canvas to model the triple overlay.



This is how the abstract layout of the Master Panel would look like: the main table, now with two rows instead of three: in the first one, a canvas that contains another canvas with the Header accessibility property (that's why I need it; to assign it the Role property, and because here the menu, the logo and the text will also overlap, as we will see later), the button, and the table with the ContentPlaceHolder and the Main property for the Role. The Footer goes in the second row of the Main table.

In this case, we must not modify anything in the layout of the panel that will be loaded in the ContentPlaceHolder, because it begins here.

In the following video, we will implement all this in GeneXus.


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