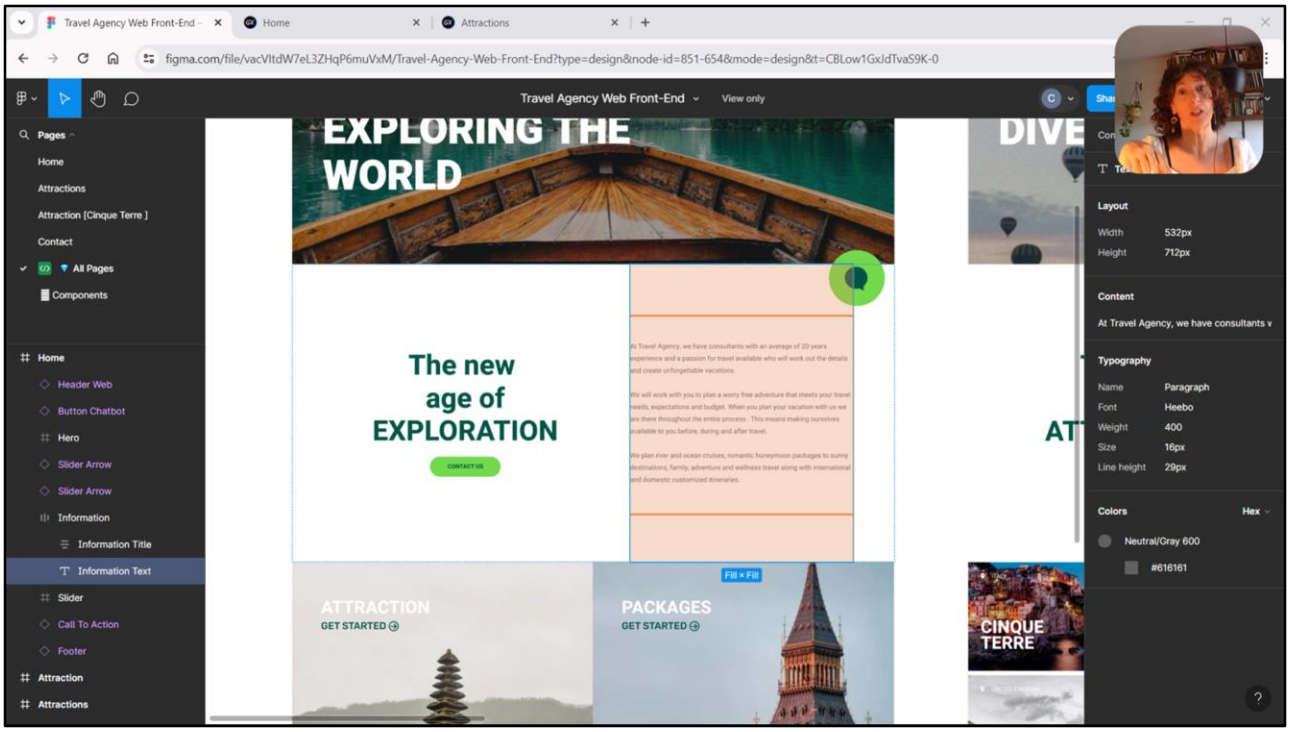
GeneXus by Globant

First Layout in GeneXus. Style (cont.)

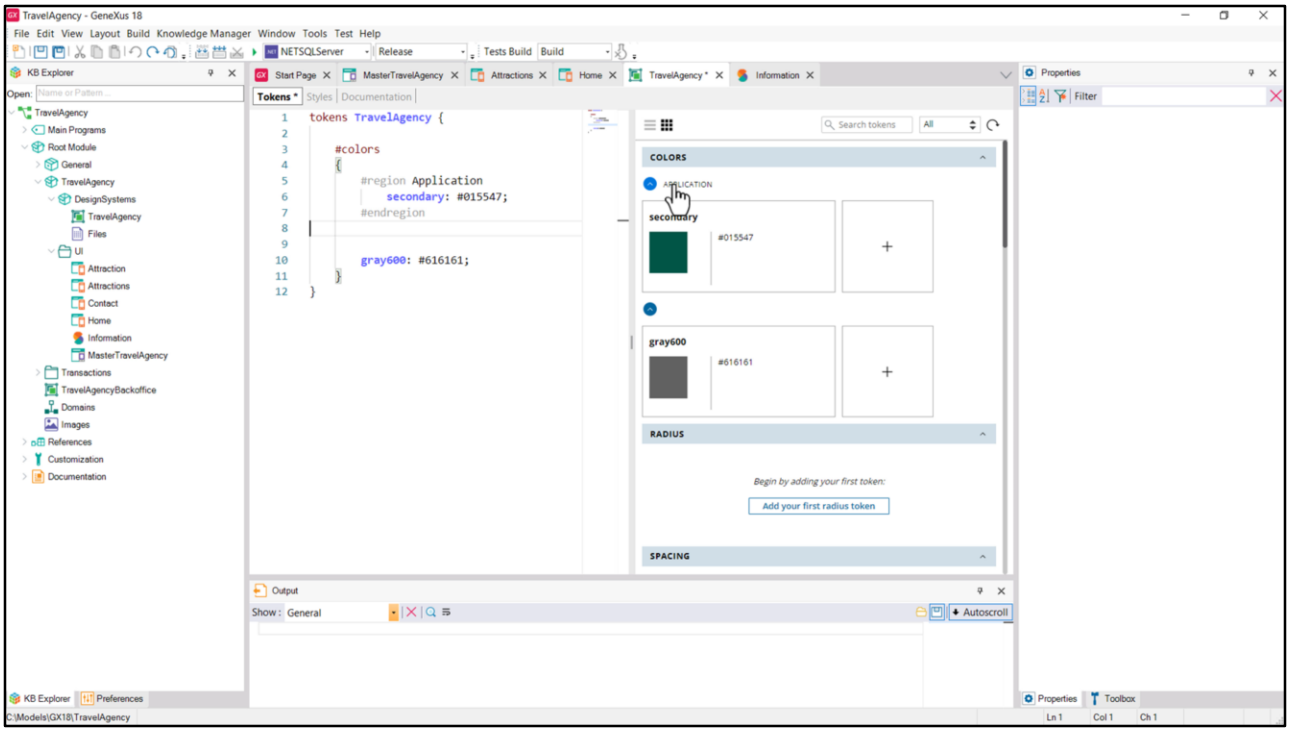
Spacing, logical properties, composite classes

Cecilia Fernandez



In this video, we will give a style to the paragraph of our stencil and we will see some specific features that did not appear in the previous video, in the case of the textblock. For example, we will see how to set a top and bottom margin to the element. Also, we will see the differences between physical properties –that is to say, those that have to do with the up, down, left, right coordinates– compared to the logical properties –that is to say, those that are established with respect to the beginning and end of the element, among other things.

We will leave the analysis of the button's style for the next video.



First, to give a style to the paragraph we will start by doing the same as we did for the text on the left.

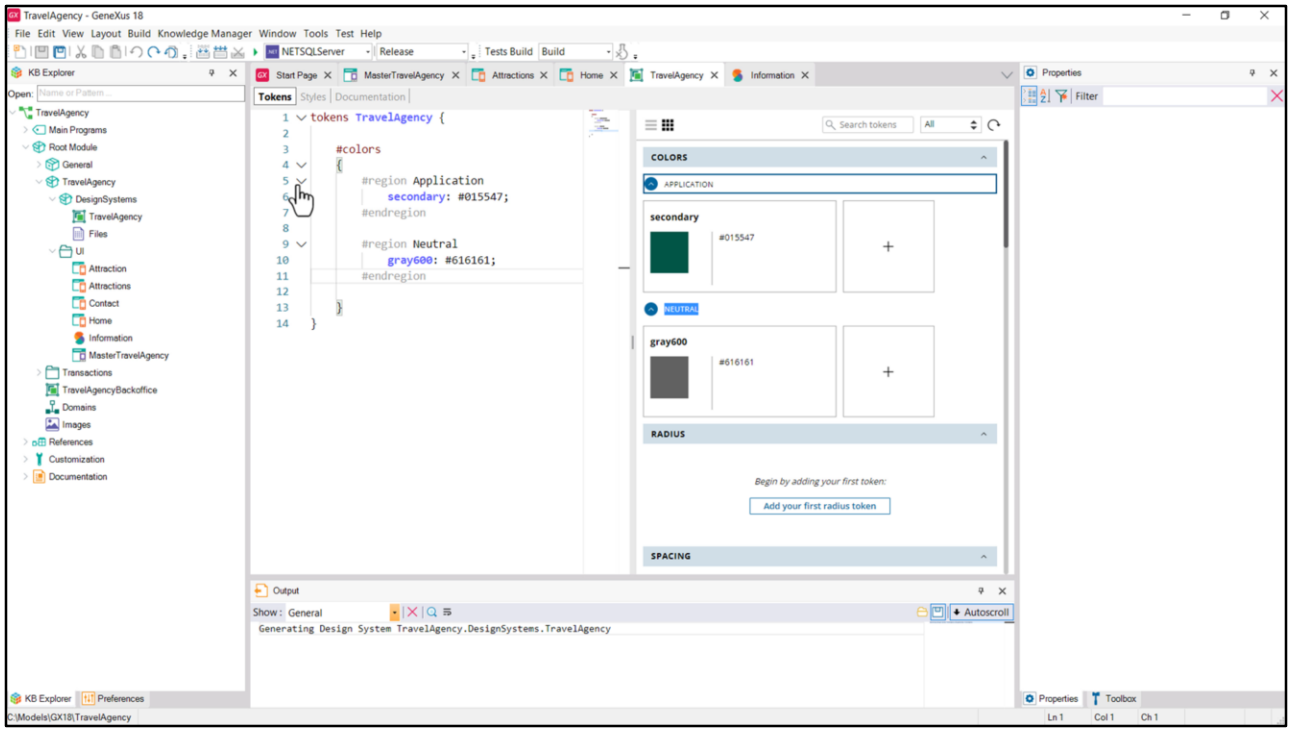
Among its typographic properties, we see the Paragraph style. And the color is this type of gray. I'm going to copy... and if I paste... we see that this property appears... what I'm going to copy is the value...

...because I'm going to start by creating that token, gray600, in the tokens tab of my DSO. And there I copy it. We see that it's showing up in the editor on the right... I could modify it from here, if I wanted to, instead of doing it from here.

And I'm going to order the information, defining here what are known as regions, which are blocks in the DSO that only serve the purpose of internally organizing the information. They have no other function.

Then we can define a region... with a numeral... there we give a name to the region. We will call it Application, for example, the region that will contain the color tokens of the application. Here is the Secondary one, but later we will also have the Primary one, and so on. Note that here I left an unnecessary plus sign ... there it is.

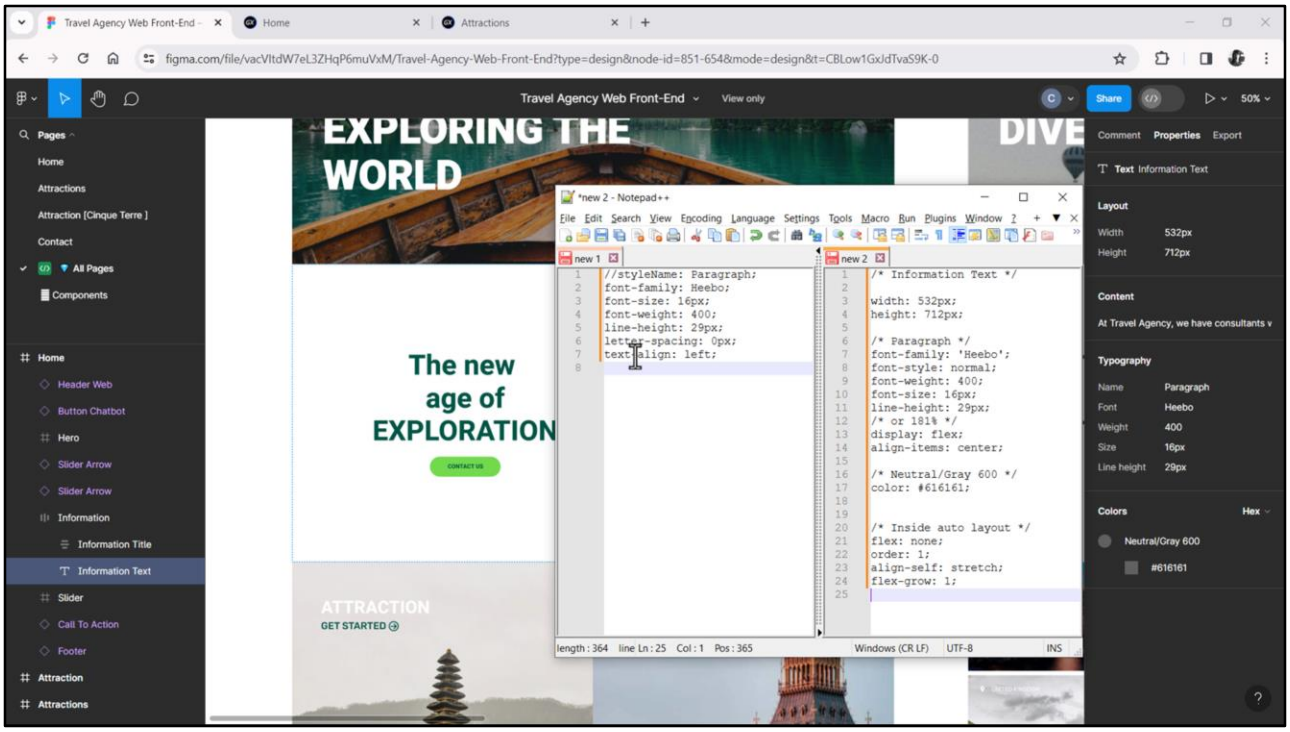
We can see that by adding this region, it is already appearing in the color editor on the right, in the graph; this region is appearing separated from this other one that we haven't named yet because it is the one that was left out.



We could leave it like this, without placing it inside any region, or I could also define a region, which I will call Neutral for neutral colors. And in there we are going to place this gray, because obviously there will be other gray colors.

Well, and there we see that we can collapse... when this grows it will make more sense. But we'll leave it this way and then we'll organize the Tokens tab in this way. Let's save.

Here also... we see that it is possible to collapse and expand the region. And the same will be true for the styles tab. We will see it later.



Now let's look for the CSS properties of the paragraph... if we copy them from here directly... they seem to be exactly the CSS properties of Chechu's style.

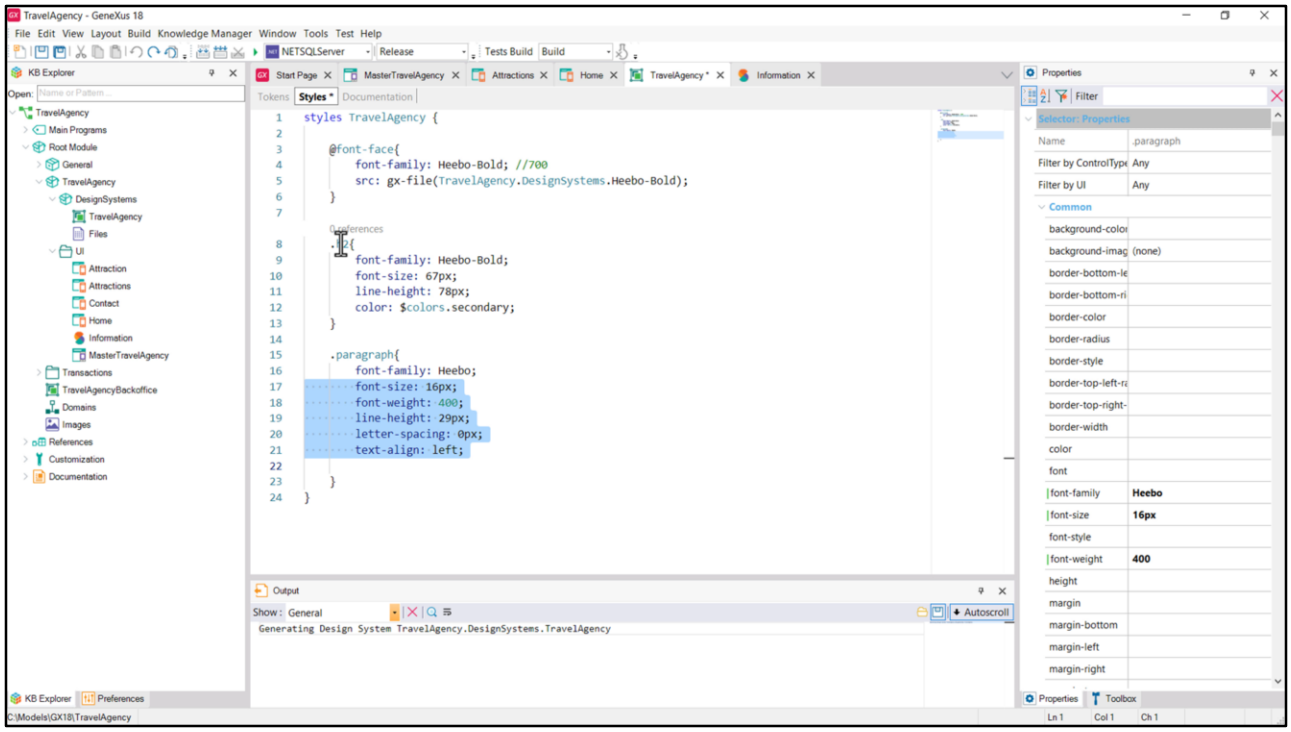
Let's see the difference with explicitly requesting the properties as CSS code...

Some are appearing here, such as letter-spacing or text-align, and are not appearing among the CSS properties listed.

Presumably this is because they are default: if they are not written they will take these values.

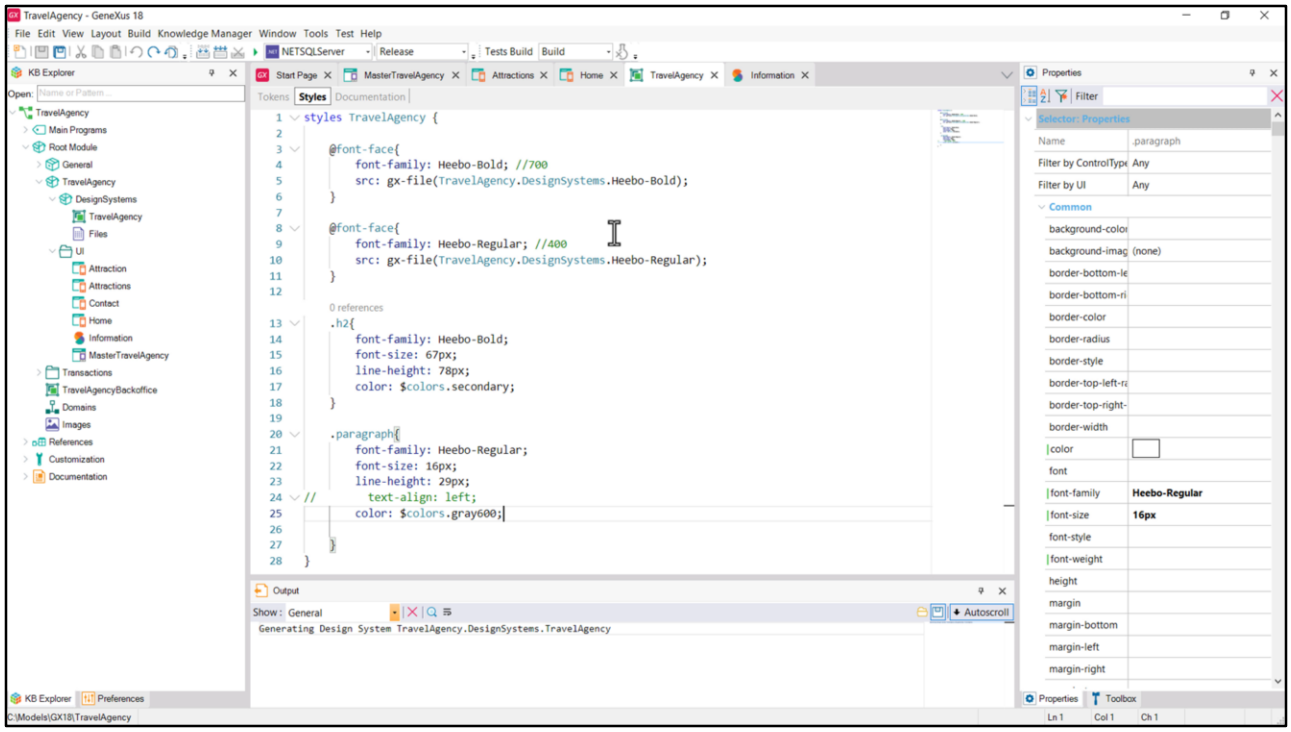
On the other hand, here is the normal font-style which is not here, but which is obviously not here because it is considered within the font family and the weight.

Let's copy these from here, then, and go to our DSO....



And in the styles tab let's select the class we will use, which we will call the same as Chechu's style: "paragraph". Let's copy the properties.

Lowercase is used to comply with the CSS BEM styles convention, which among other things indicates that all classes should be in lowercase. Keep in mind that everything we write in a DSO (as it happens with HTML in conjunction with CSS) is case sensitive; therefore, a paragraph class that starts with lowercase and a Paragraph class that starts with uppercase will not be the same. We have to be careful with that.



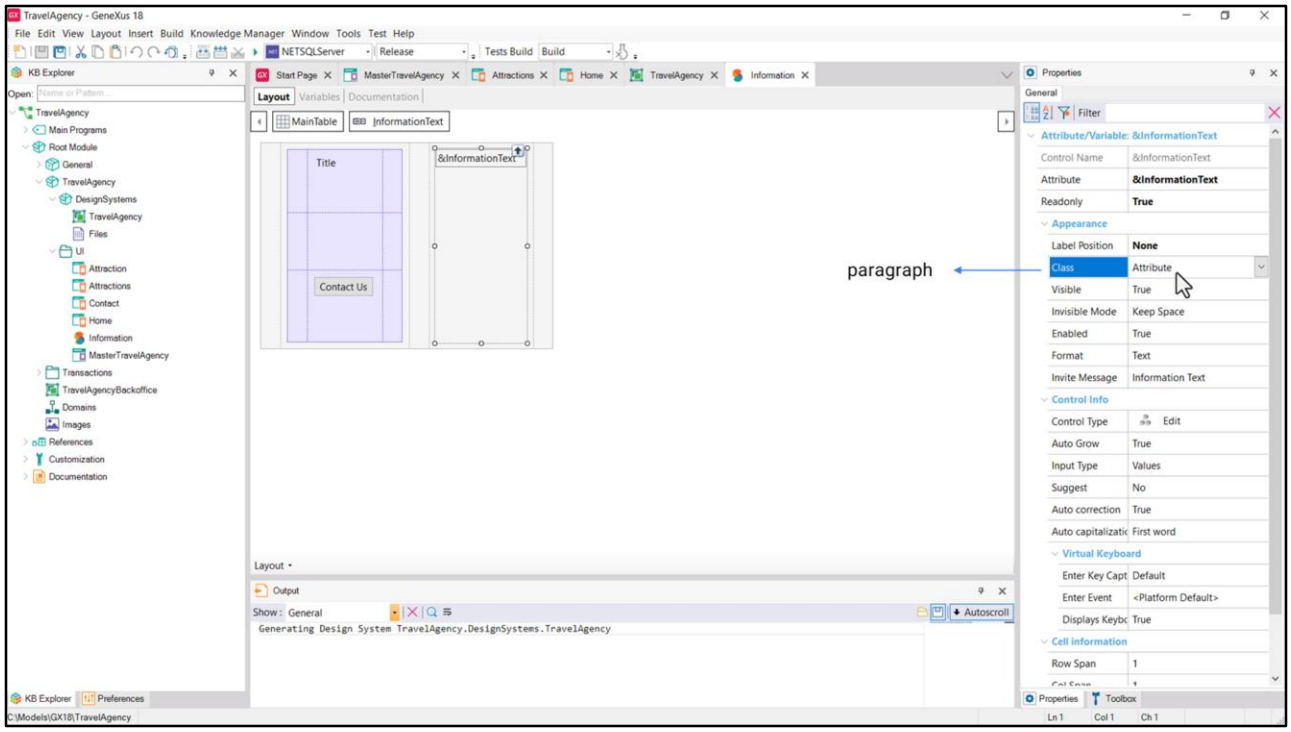
We have to integrate the font in our case, as we did for the case of the h2 class, to our DSO. Therefore, we will use the same font-face rule.
Font-family... we will call it Heebo... and which of the Heebos has 400 weight? It was the regular one, so we'll call it this way... let's write as a comment the 400 weight.

And again, where is it located? In the file that we had embedded in the KB in the previous video. So, since I did that, I can call it here like this.

I am going to delete this property, because it is already included in this other one... font-size 16, line-height...
And we don't need these two as we were saying, because the space between the letters, if nothing is specified, is already 0 pixels. So I'm going to delete it.
And for now I will leave this one commented.

We only have to add the color property, the color that will take the text that has this class, which will be the color token gray600.

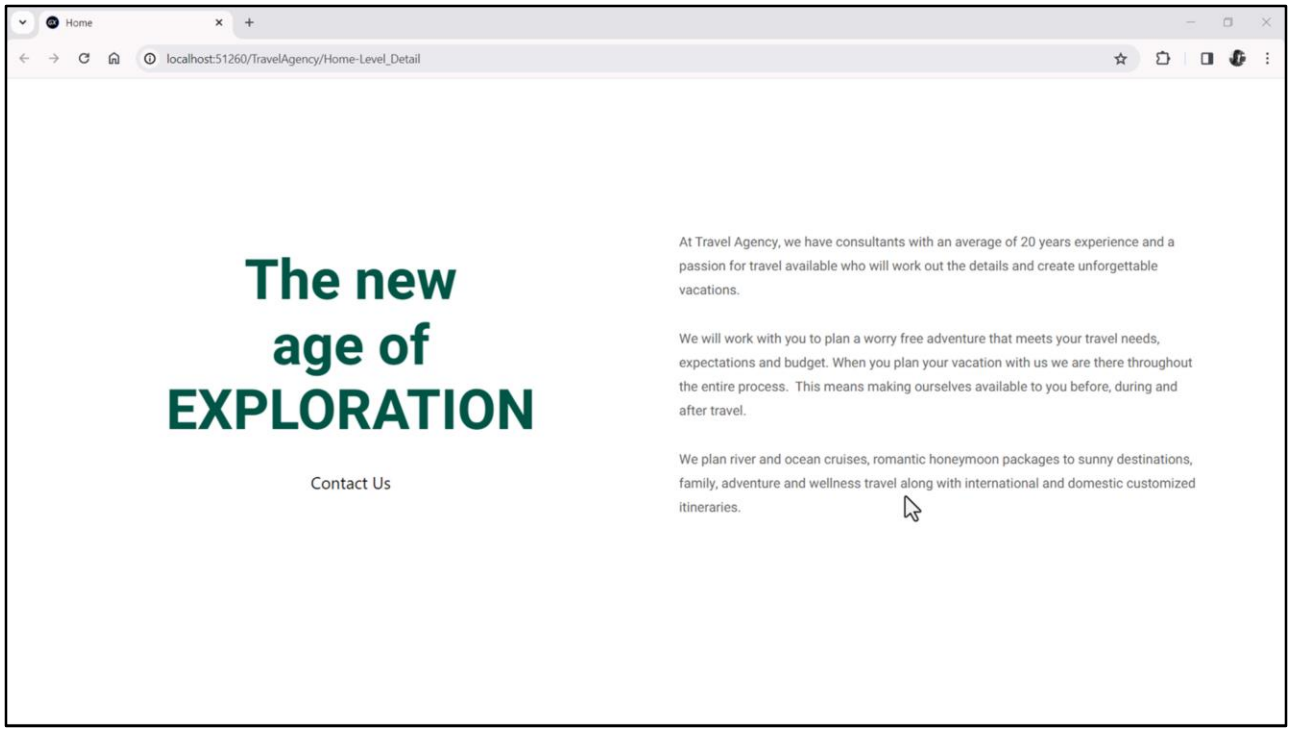
Okay, we save.



Finally, in this case we have to associate the class we have just created with the variable control. It will not be this Attribute class, which was the default one. Since it is not selected in our DSO, it's as if it were empty, that is to say, without associating a value with the properties, so it will take the default values in the browser. We were saying, then, to change this class for the paragraph.

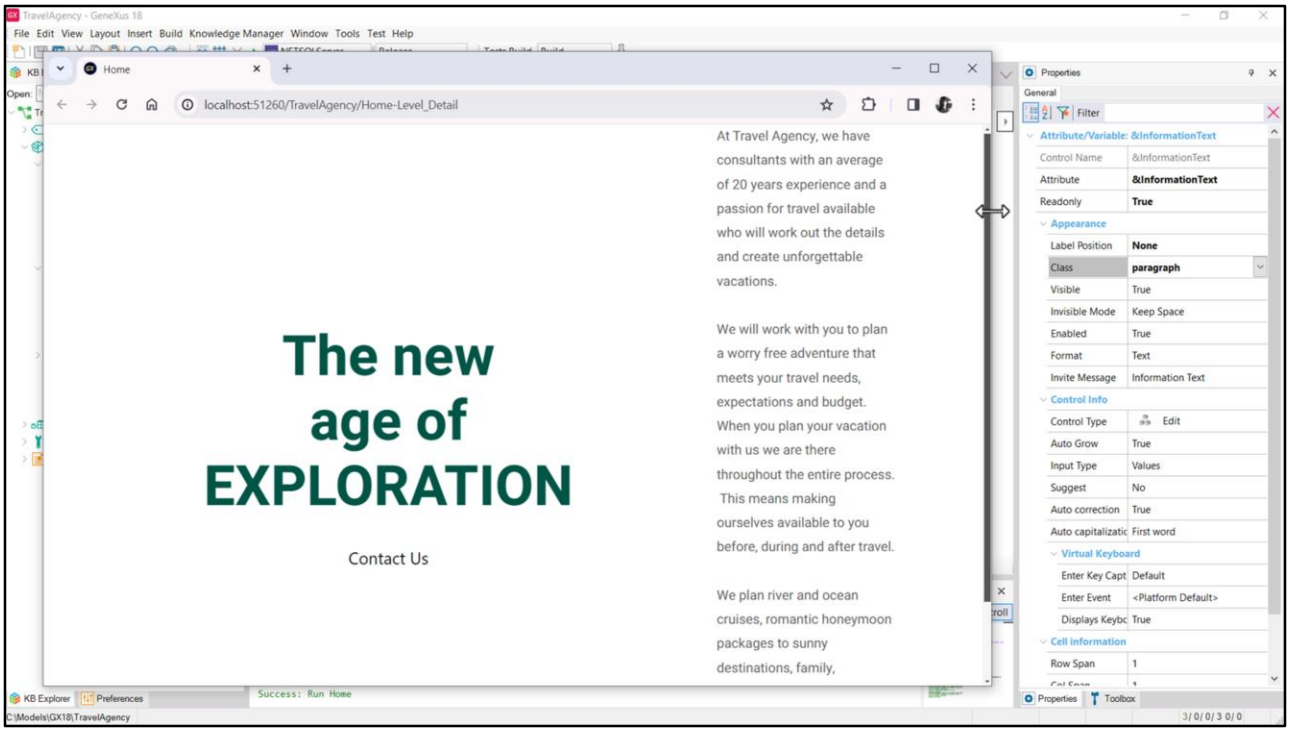
Having done this, we save... as we saw before... it is automatically changed for the variable instantiated in the Home panel, and the same will be for the variable instantiated in that of Attractions.

Okay, so let's run it to test...

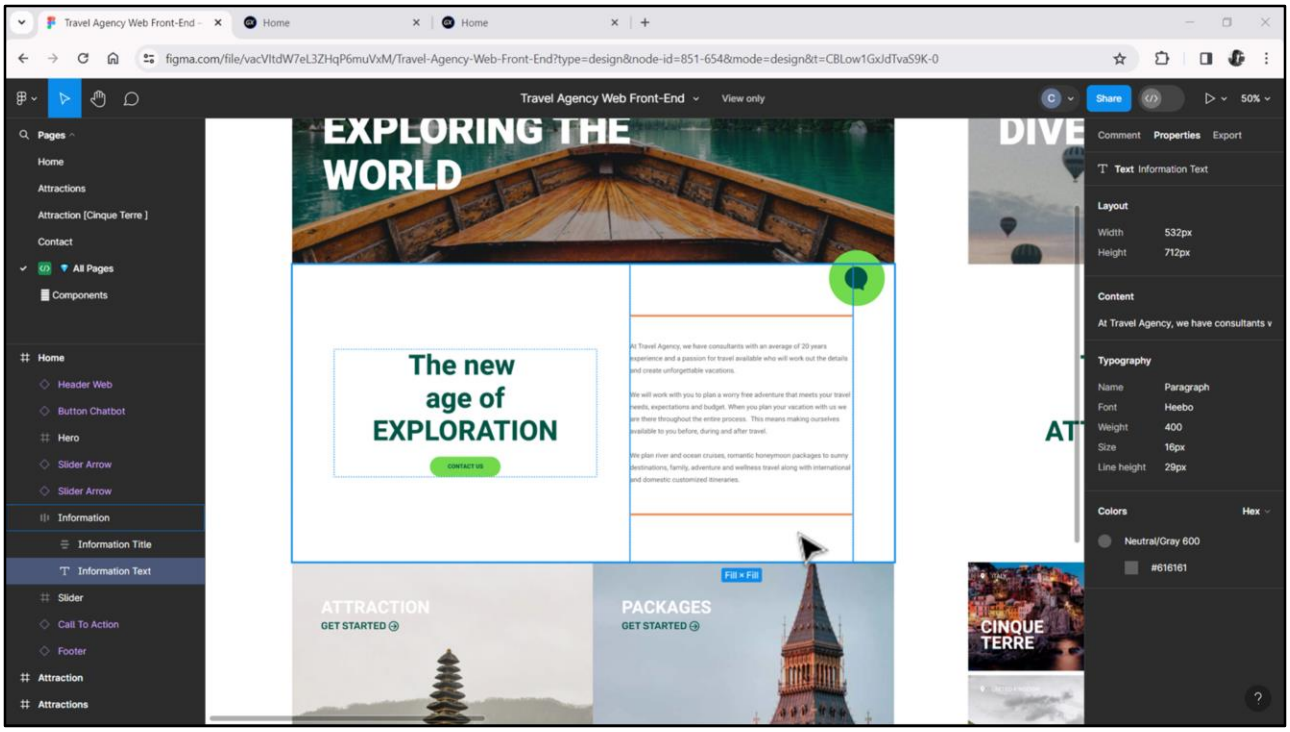


... perfect.

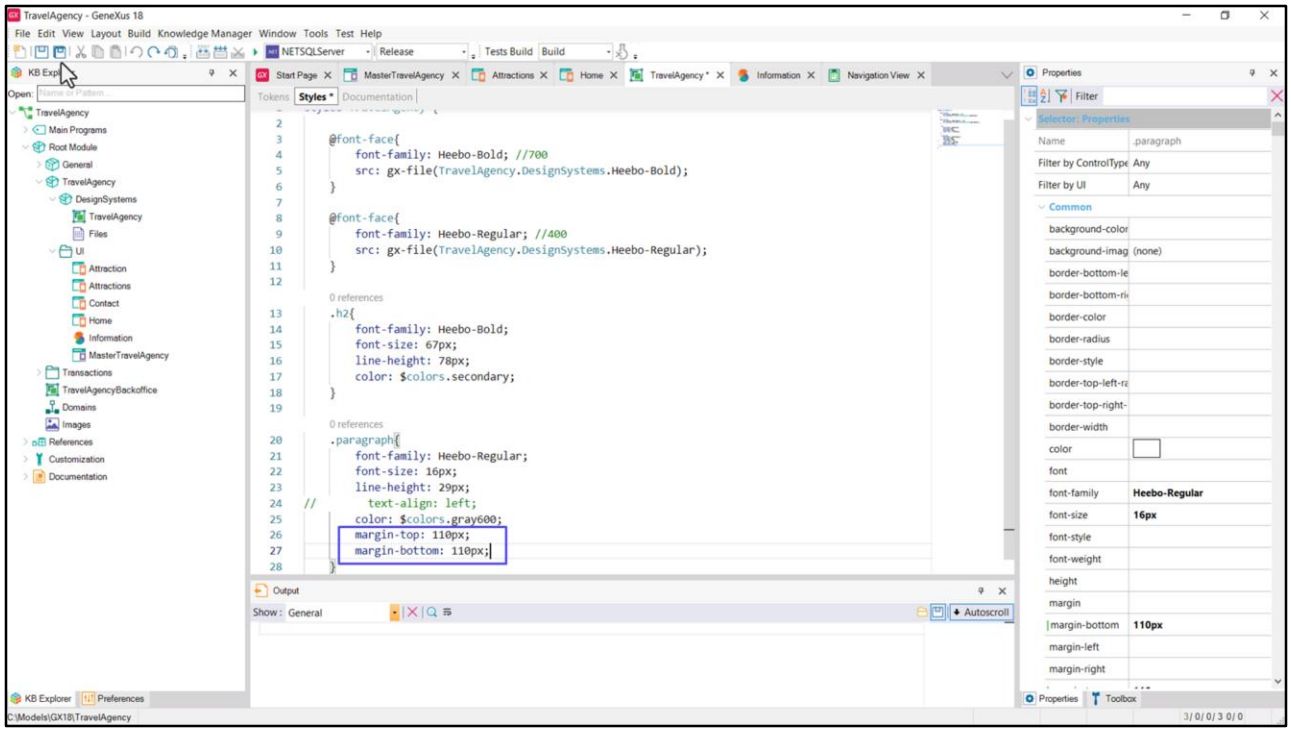
Clearly the text-align left property was not necessary, because we left a comment and it is still aligning to the left.



Now it's time to bring up something that we had left pending in one of the previous videos... And that was not having defined a vertical spacing for this text... remember that if we decreased the screen width it would stick to the top and bottom edges of the table?

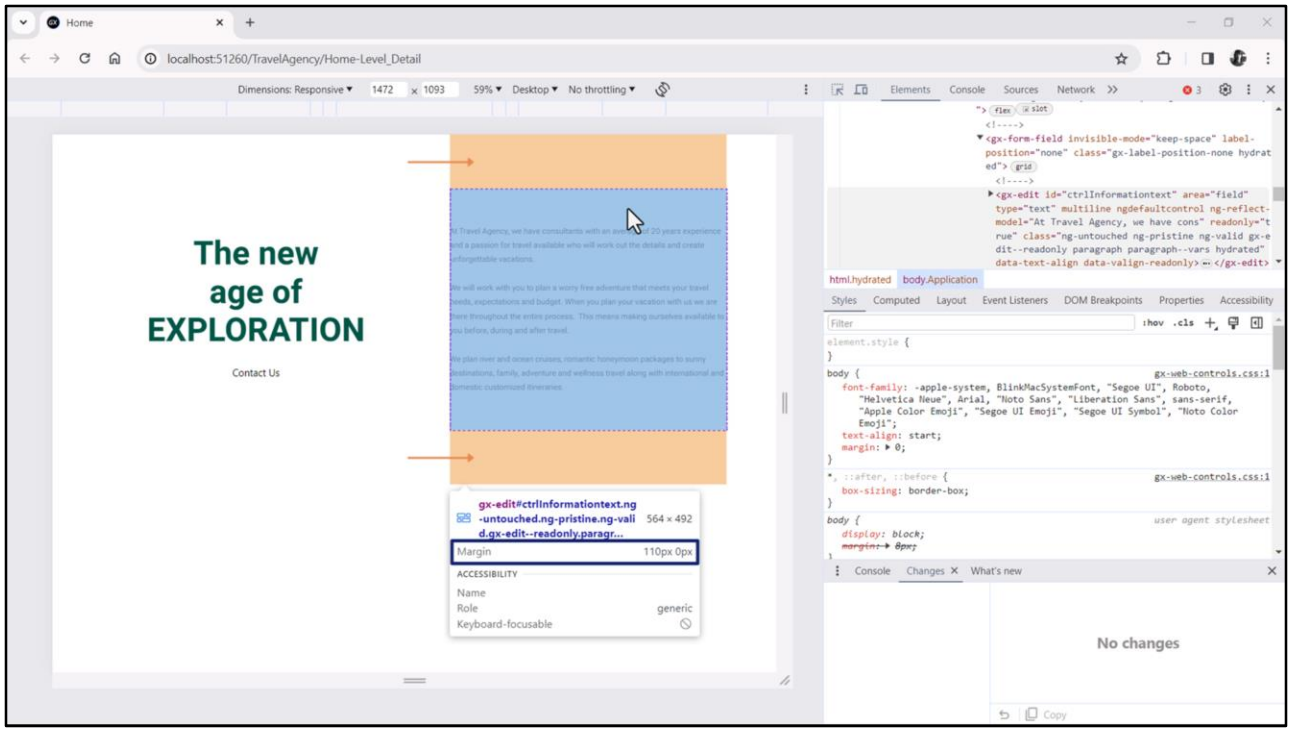


It had Fill for the height. In that video, we had seen that a good top and bottom spacing could be 110 pixels and that we could achieve that by specifying a top and bottom margin for this element, of that value. As if Chechu could have added these properties here... she cannot, clearly; the typographic style is only typographic. And she has no other way to do it either.



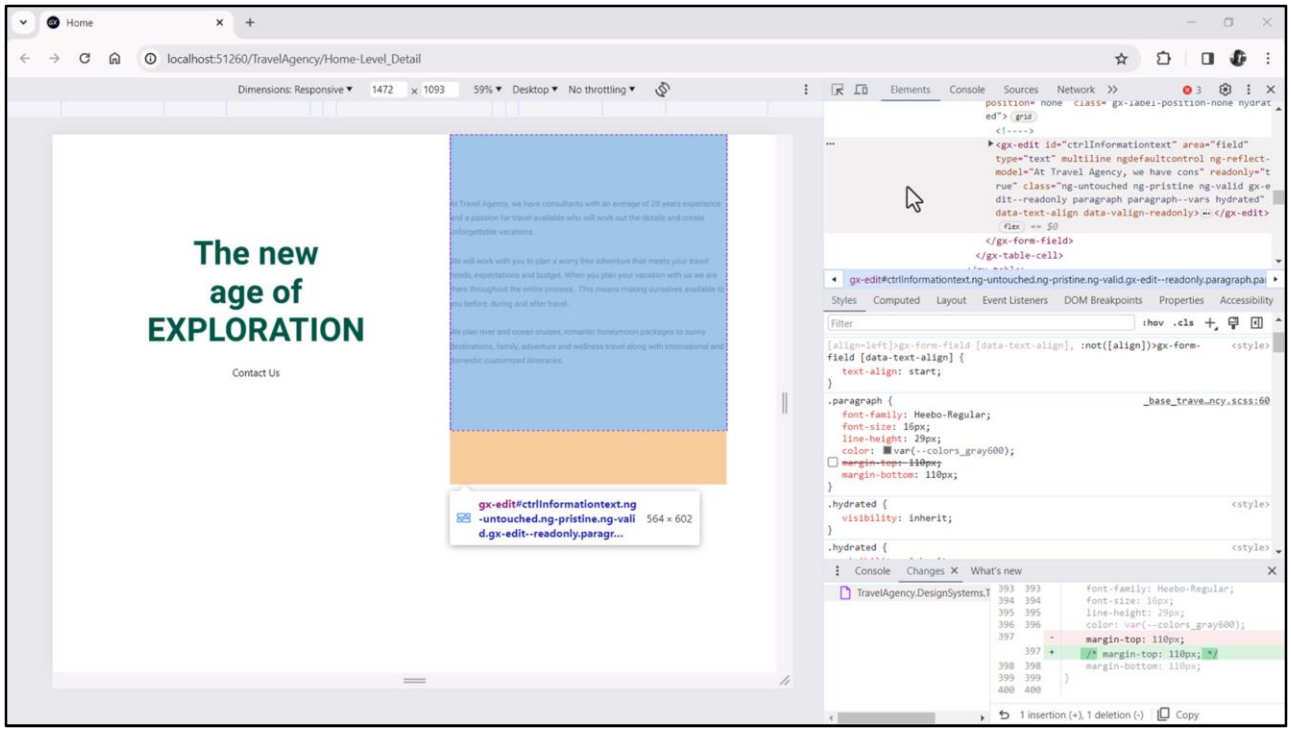
But we don't have that constraint in our class. So we could write this...

Let's try...

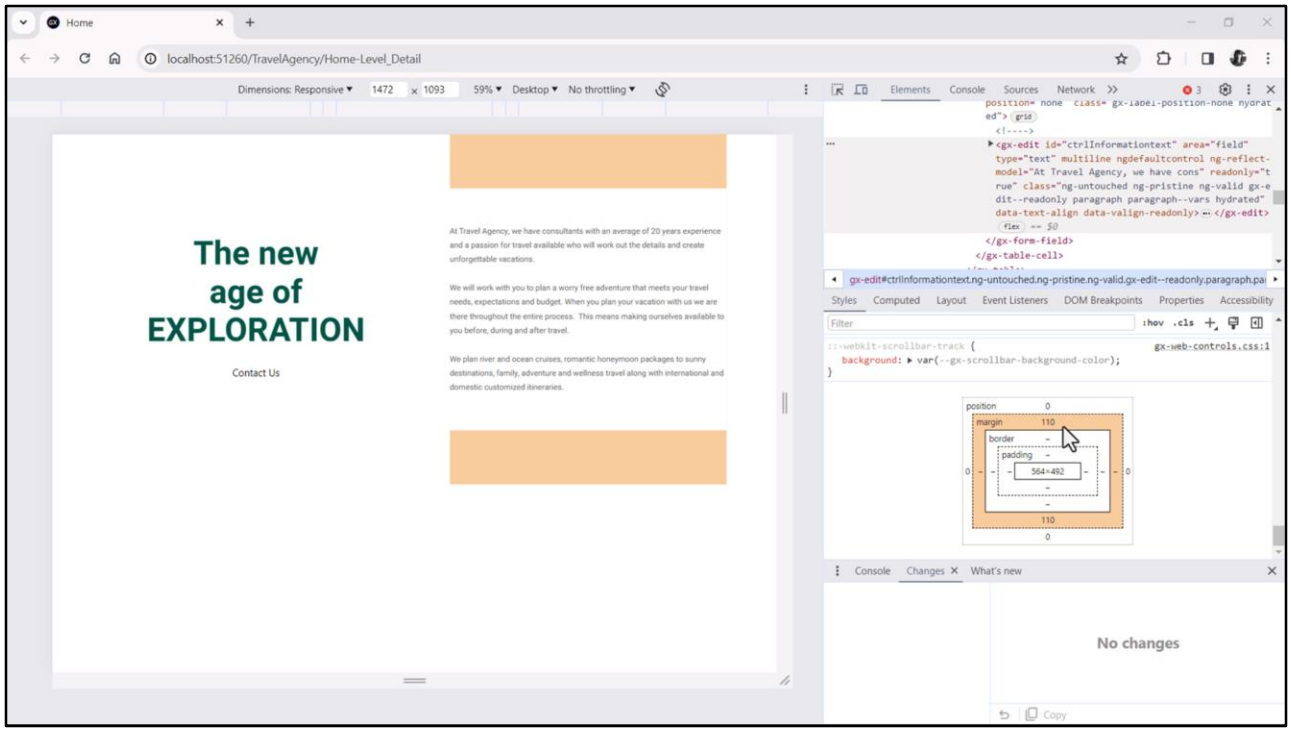


We see this margin is being kept, up and down.

Let's inspect the resulting HTML. Here we can clearly see the element and the top and bottom margin.



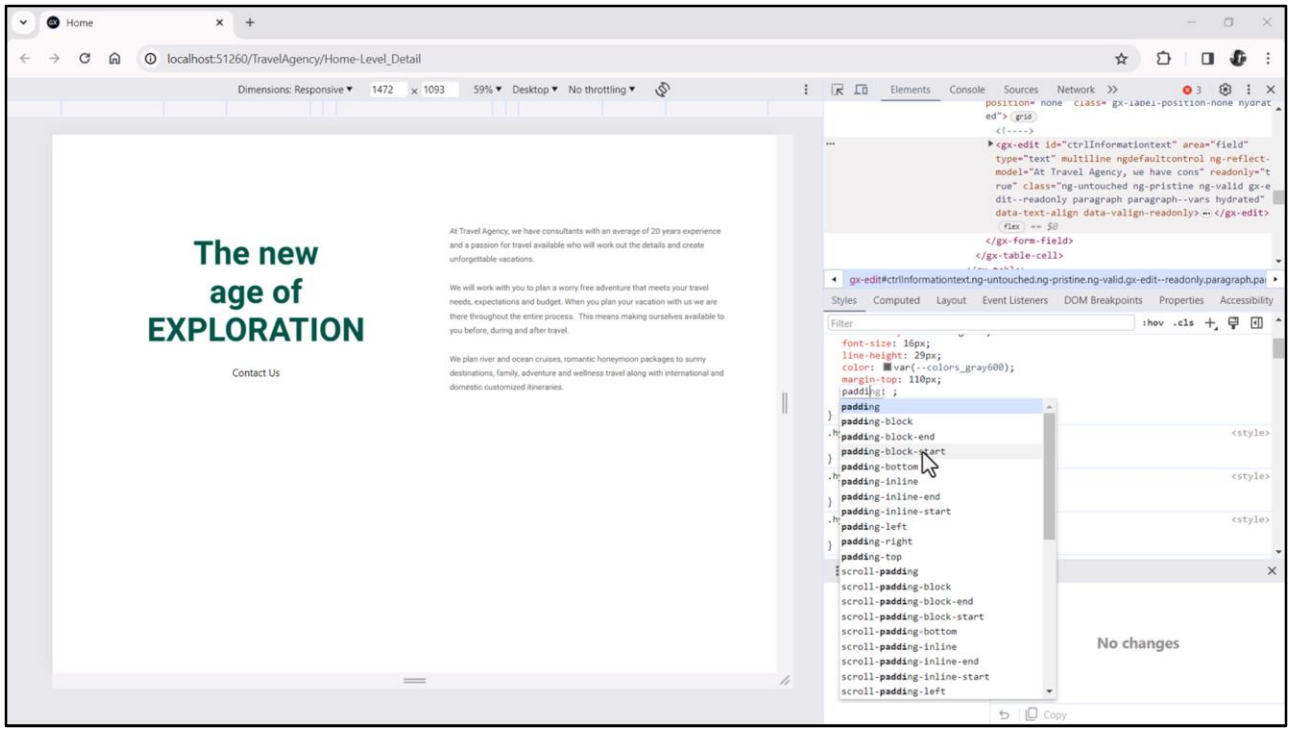
If we look here for the class that is applied, which is paragraph, we see the two properties, and for example, we could turn off margin-top and see how doing so effectively removed that top margin.



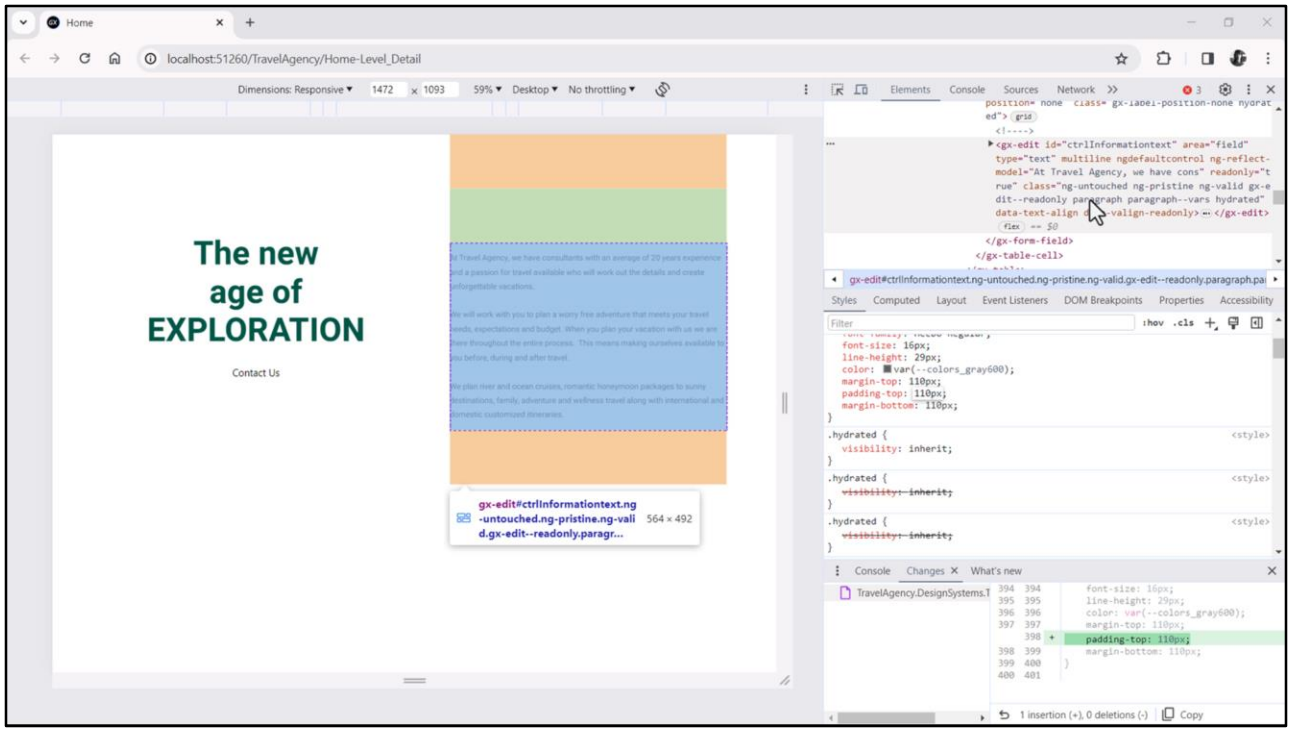
At the bottom of the page we can see precisely those 110 pixels of margin at the top and at the bottom. Note that no border or padding has been defined, and here we have the internal element.

Why did we choose to give that spacing through top and bottom margin and not do it through padding? Visually we would see the same, let's try changing the margin of 110 for a padding of 110, for the top margin and padding. Let's leave the bottom margin as it is, to see the difference.

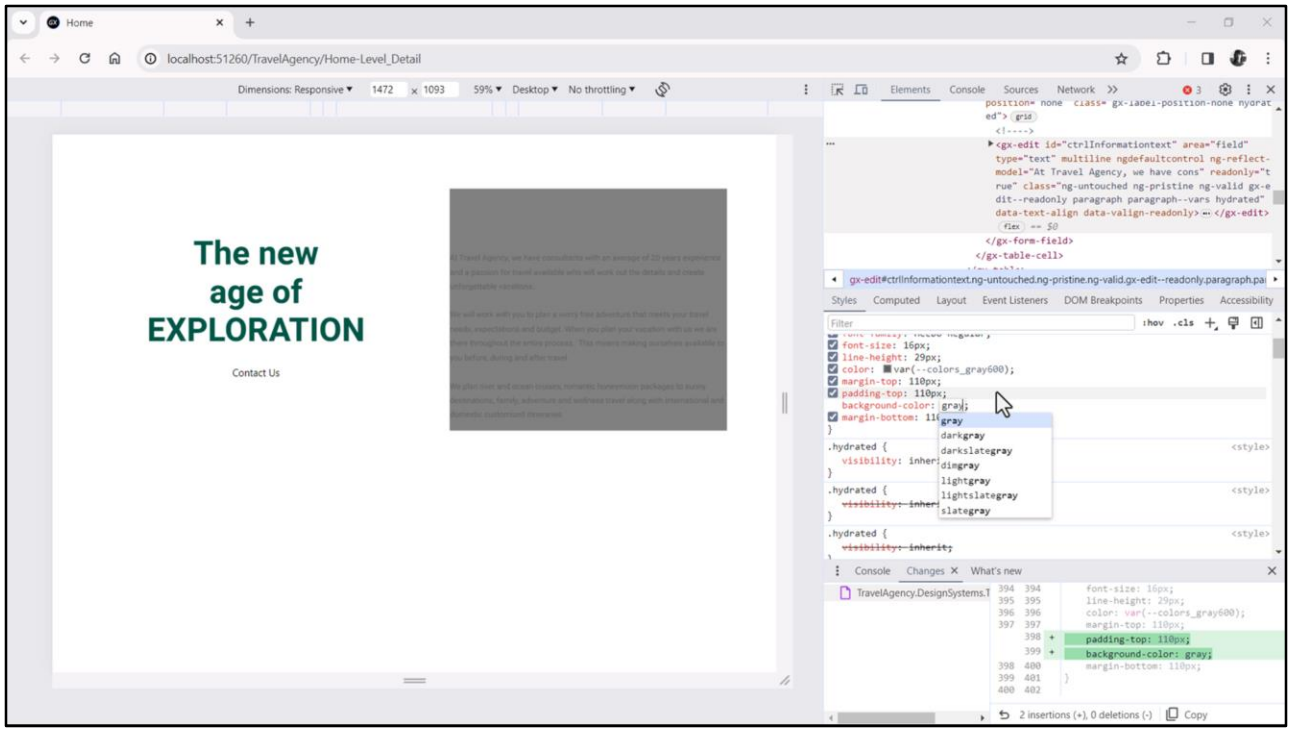
But here I am going to go ahead: the element is what is inside the border, from this limit inwards. The margin will be outside the element, and let's see the implications of that.



So first, I'm going to go back to the class, and we're going to set the padding property... we're going to choose padding-top... I'm going to mention them now because we're going to get into this soon, but do you see these block block block padding and inline padding?

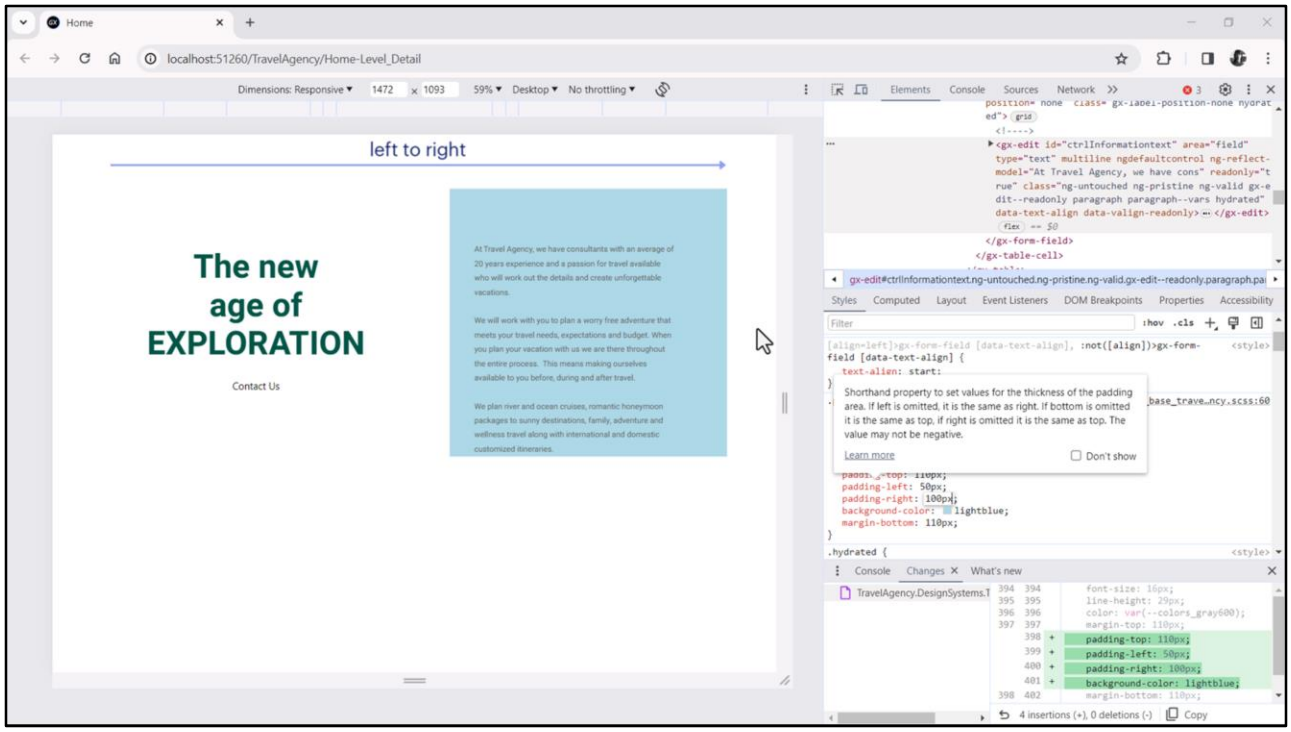


OK, padding-top 110 pixels. And we see that clearly indicated there: it's that green area.



Let's see what happens –I'm going to leave both: the top margin and the top padding– and let's see what happens if I give a background color to this element. Background color... and for example we are going to place a gray.

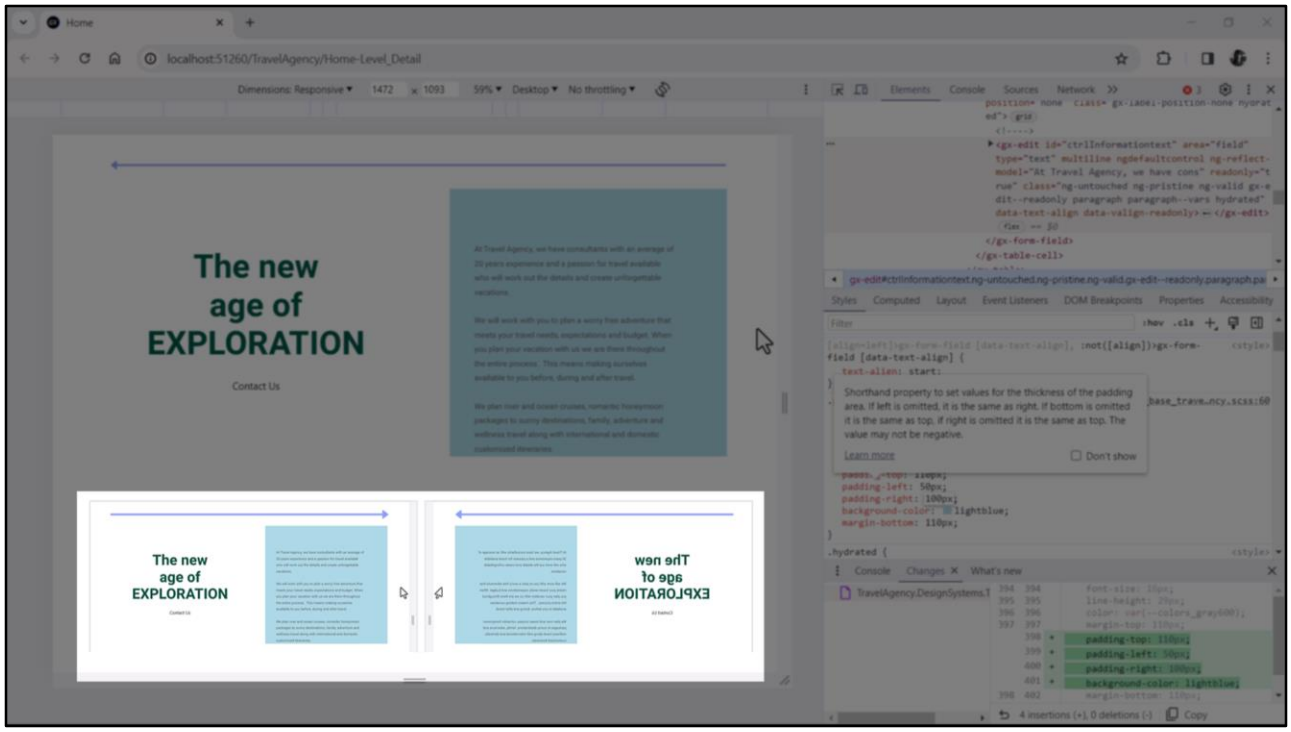
What is happening? We see that both the internal area of the control and the padding area, that green area, are being grayed out. Meanwhile, the margin area is not being applied that background color. Likewise, if we implemented an event, clickable at the level of this content, this area would also be clickable, while the margin area would not be.



Well, now we're going to get into this issue of the difference between physical properties (padding top, right, bottom, left) and logical properties, and now we're going to see what that's all about.

First, let's change this gray for a lighter one. Light... well, I'm going to leave light blue. And what I'm going to do so that the difference is clear is to write a padding... left... of 50 pixels... and a padding right of 100, twice as much. That is to say, for the left border we have a padding that is half of the padding of the right one.

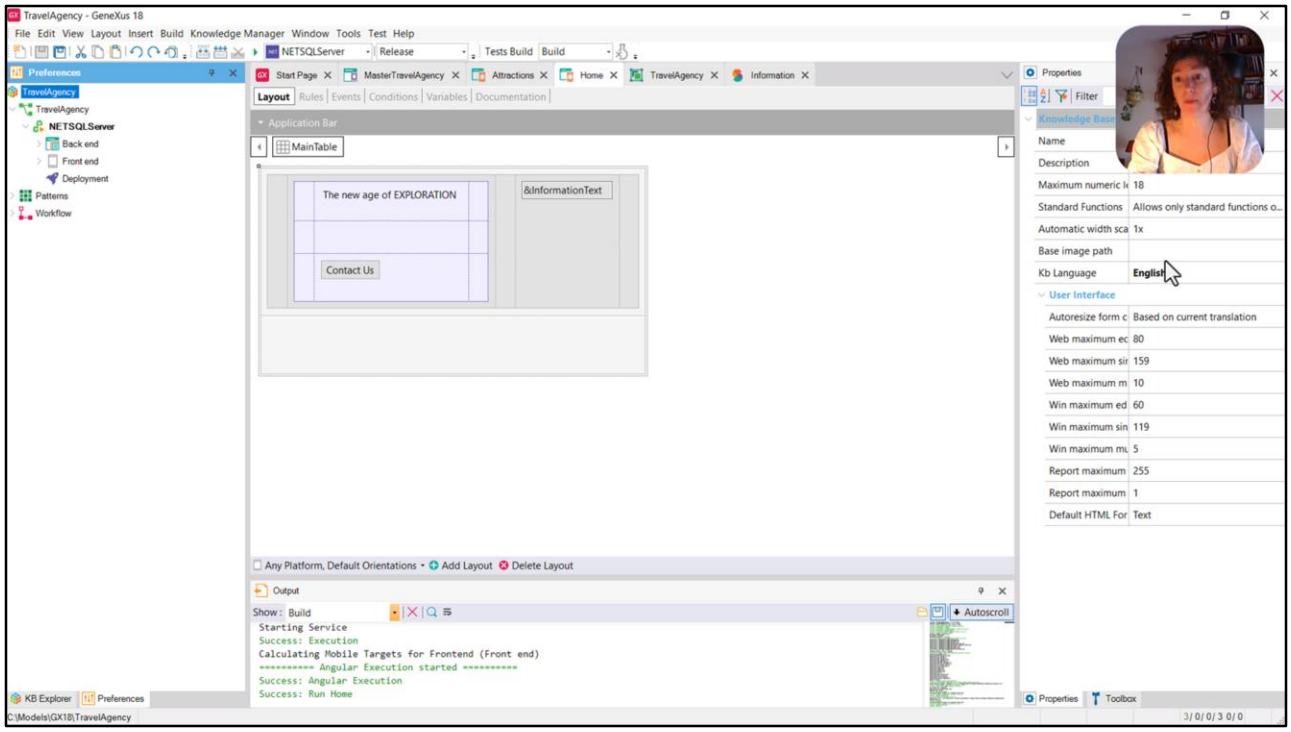
Now, the question I have... we are running this application with a text direction going from left to right...



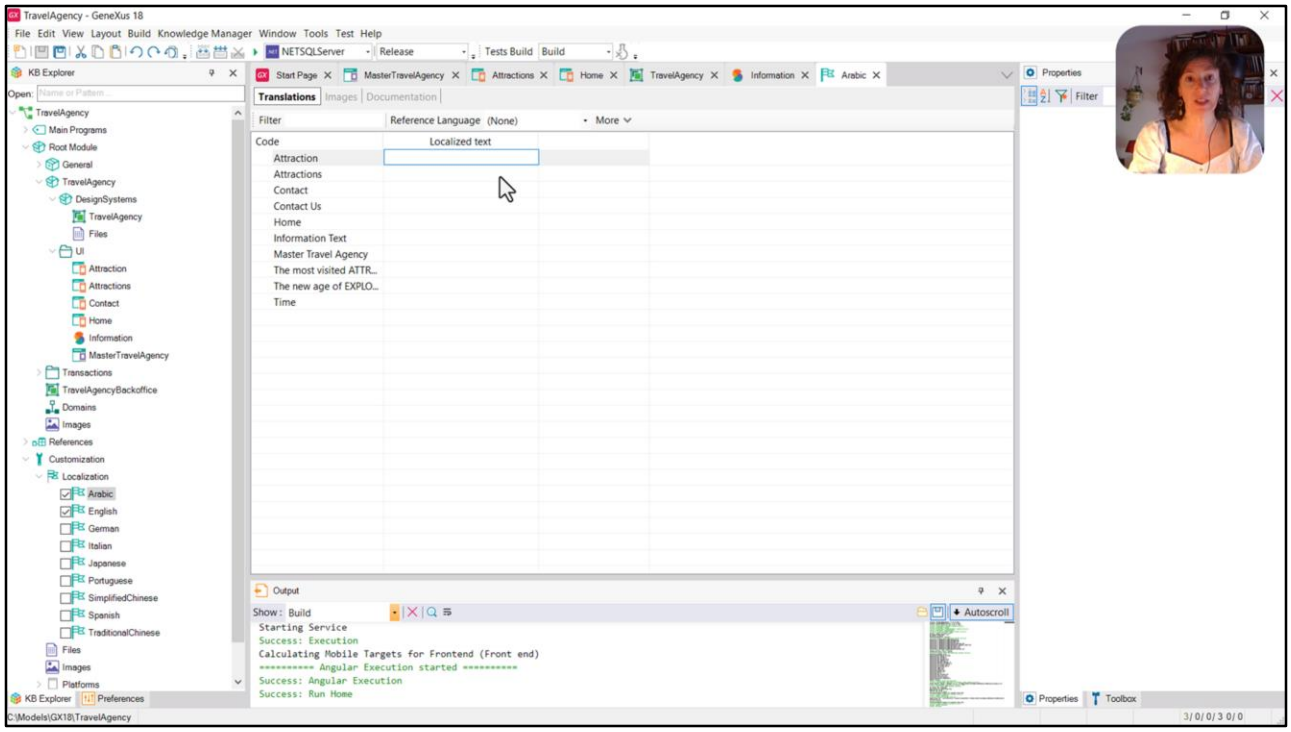
If we were running the application for Hebrew or Arabic, it would have to display all this that we are showing here as if we had a mirror that completely reversed the direction. So it would be in a right to left direction.

What would we want to happen in that case, i.e., by changing the flow direction of the screen?

We would want the table to be mirror-reversed, where the first column on the left becomes the first column on the right, the second column becomes the second column, the third column becomes the third column, the fourth column becomes the fourth column, and the fifth column becomes the fifth column, in that order. At the level of the location of the controls in our layouts, this will be done automatically.

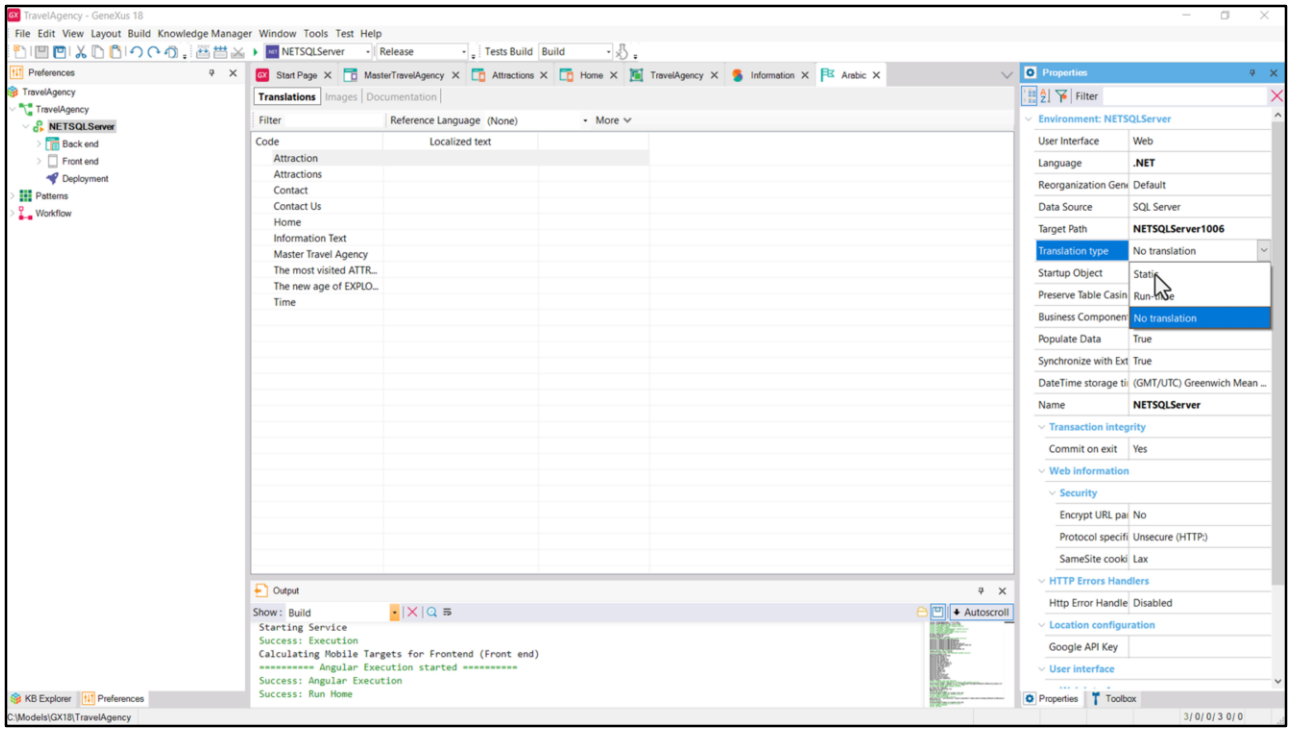


Of course I'm referring to the case where we want to translate the application. If we select the Preferences tab, we can see that at KB level we have the Kb Language property set to English.



If we come to the KB Explorer, below the Localization node, we see this English object that matches the language of the KB. And for the moment it is the only one incorporated in the KB. What is going to be done? The texts to be translated are going to be fetched from all the objects in our KB.

Then, in this way, let's suppose that we want to translate the application into Arabic. It will be enough to turn on this property, because it will incorporate that object inside the KB, it will extract... I'm going to open it... all the constants, all the texts it found in the different objects, to allow us to write the translations of those texts.



And then, we go to the Preferences window and we can, at the Environment level, set the type of translation we want to make. If it is static, where we are going to indicate the language to which we want to translate the application, of those that we have incorporated in the KB; or, on the contrary, if we want it to make it dynamically. When doing it dynamically we will be able to set at the objects level the language that we want the application to be rendered with each time. Of course, we will have to add some programming.

OK, so this is the context we are in, right? What I was saying a little while ago is that once I translate the application into Arabic, the table columns are automatically going to be mirror reversed. And so will the texts. They will appear in the language I have chosen, with the translations I have entered.

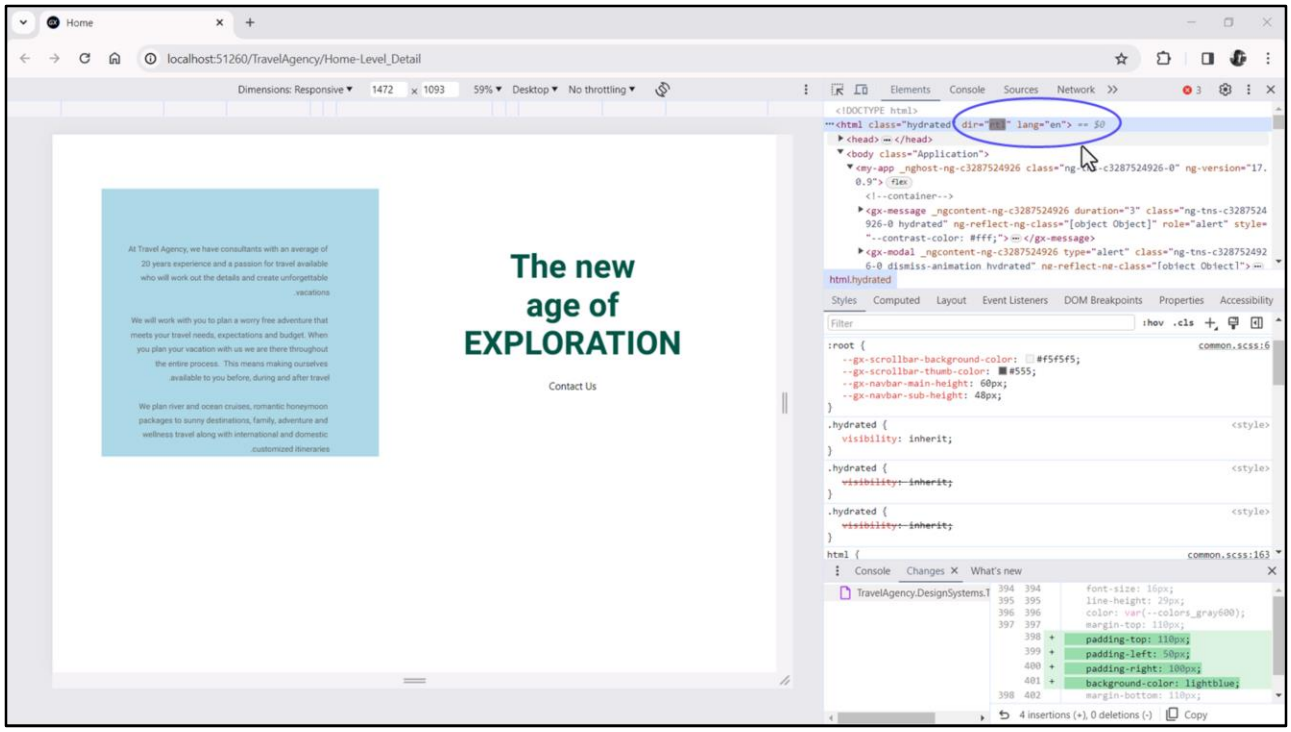
...it would look like this.

Now, let's focus on what we were interested in, the padding. When we used left and right references to provide the padding, we did not realize that in fact the correct references would be start and end, in relation to the flow direction and not left and right... because in the right to left flow we don't want the padding on the left to be 50, and on the right to be 100, but the other way around.

We need to use logical references, that is to say, the padding in the direction of the flow (inline) with respect to the beginning, to be 50; and the padding in the direction of the flow (inline), with respect to the end of the element, to be 100. That is to say, we are becoming independent of the left and right coordinates, which makes it difficult when we want to make this inversion.

But in the same way and for consistency, if we change the horizontal direction properties, we will also have to change the properties that have to do with top and bottom... Then the analogous but logical properties appear: block-start and block-end.

Therefore, inline represents the horizontal direction, the writing direction, and block the vertical direction.



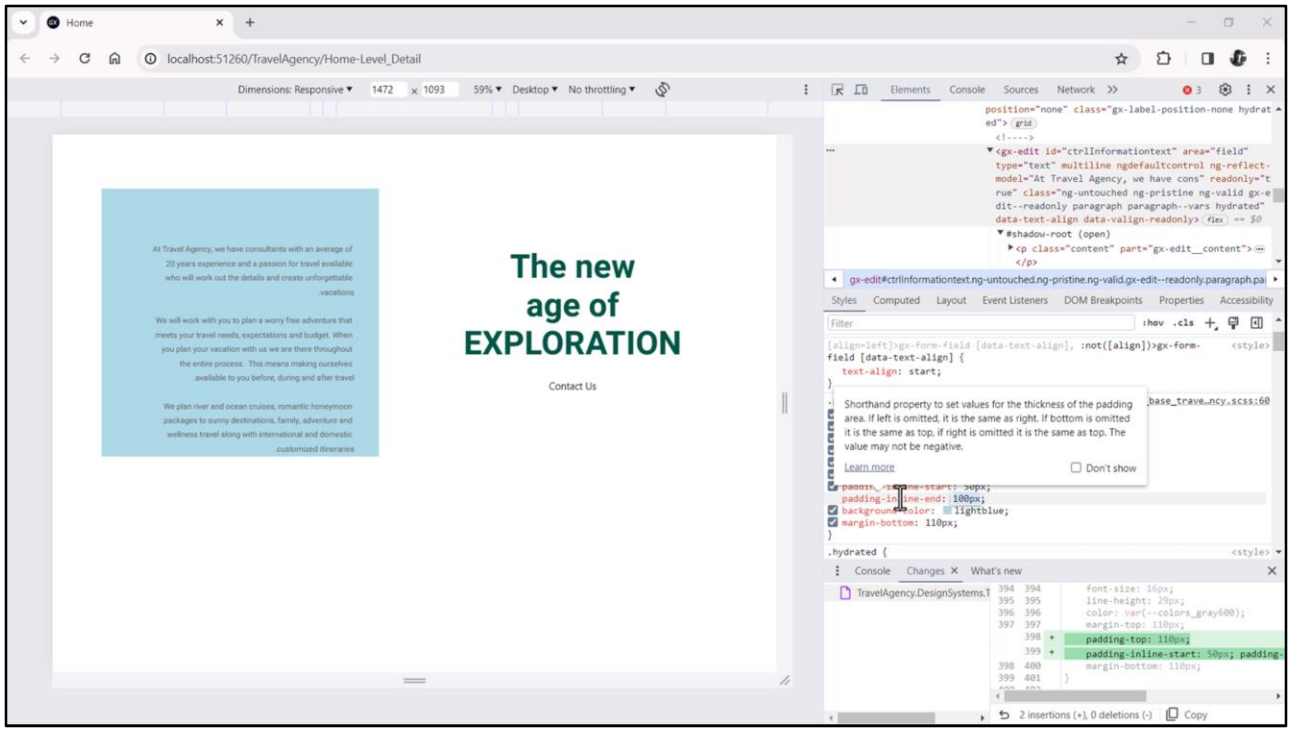
To make all this clearer, let's see it here. We have, then, padding-left and padding-right. And what I am going to do is to come to the HTML tag to change the direction from left to right to right to left, knowing that the language is English, as we were saying.

So now when I press Enter... we see that everything we wanted was indeed reversed,

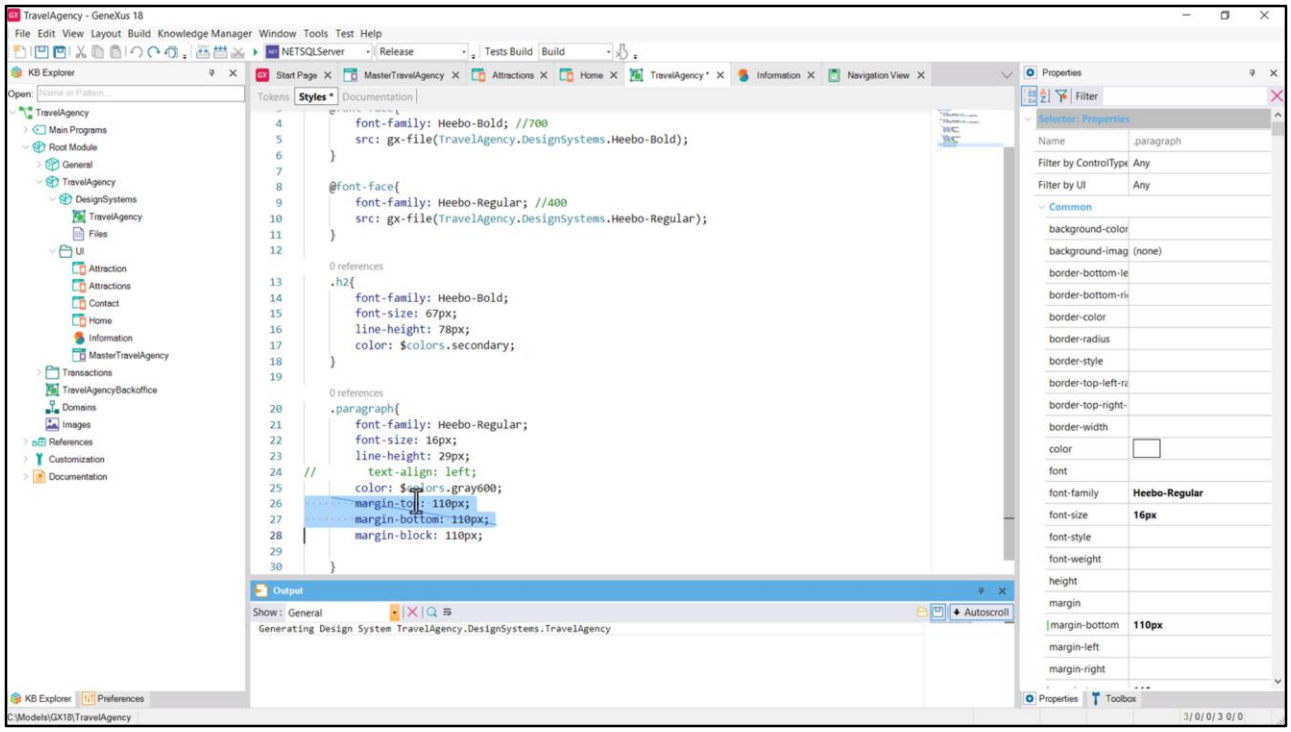
(the texts are not reversed because the language is English, but we don't care, when the language is Arabic or Hebrew they will be reversed).

So, as I said, everything we wanted will be reversed,

except for the padding... since we have used the physical properties it will keep the **left** padding at 50 and the **right** one at 100. We didn't want this, we wanted it to be reversed, therefore...

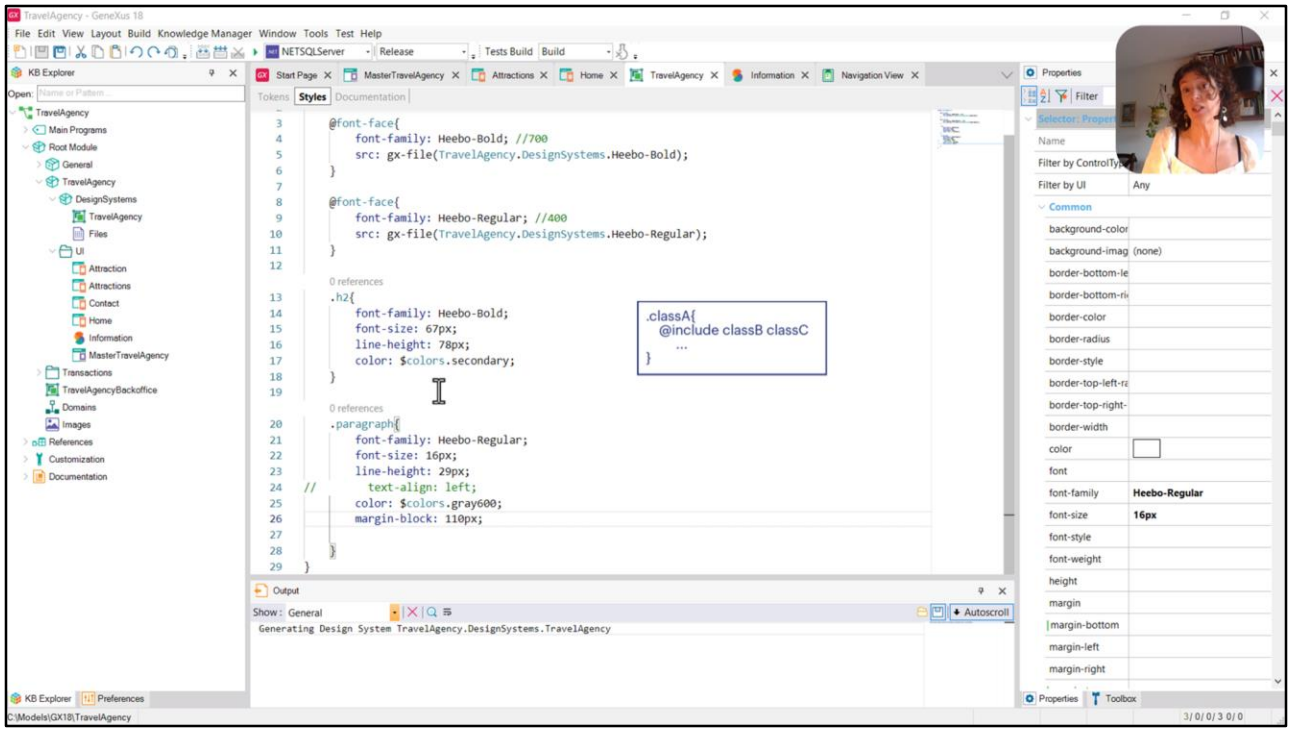


...we are going to modify the physical properties changing them for the logical ones... so instead of padding-left we are going to use padding-inline-start of 50... and for padding right we are going to use padding-inline-end.



Well, all this we saw is to strongly recommend not to use the physical properties but to use the logical ones. In the previous case we saw the example with padding but it is exactly the same with margin.

Here we are using the physical ones, so the suggestion is to use instead of these... margin-block would be in this case... start and end... Or, if I want to use the shorthand notation, just block, which in this case will suit me because they also have the same value, so I can place it only once. And remove these two.

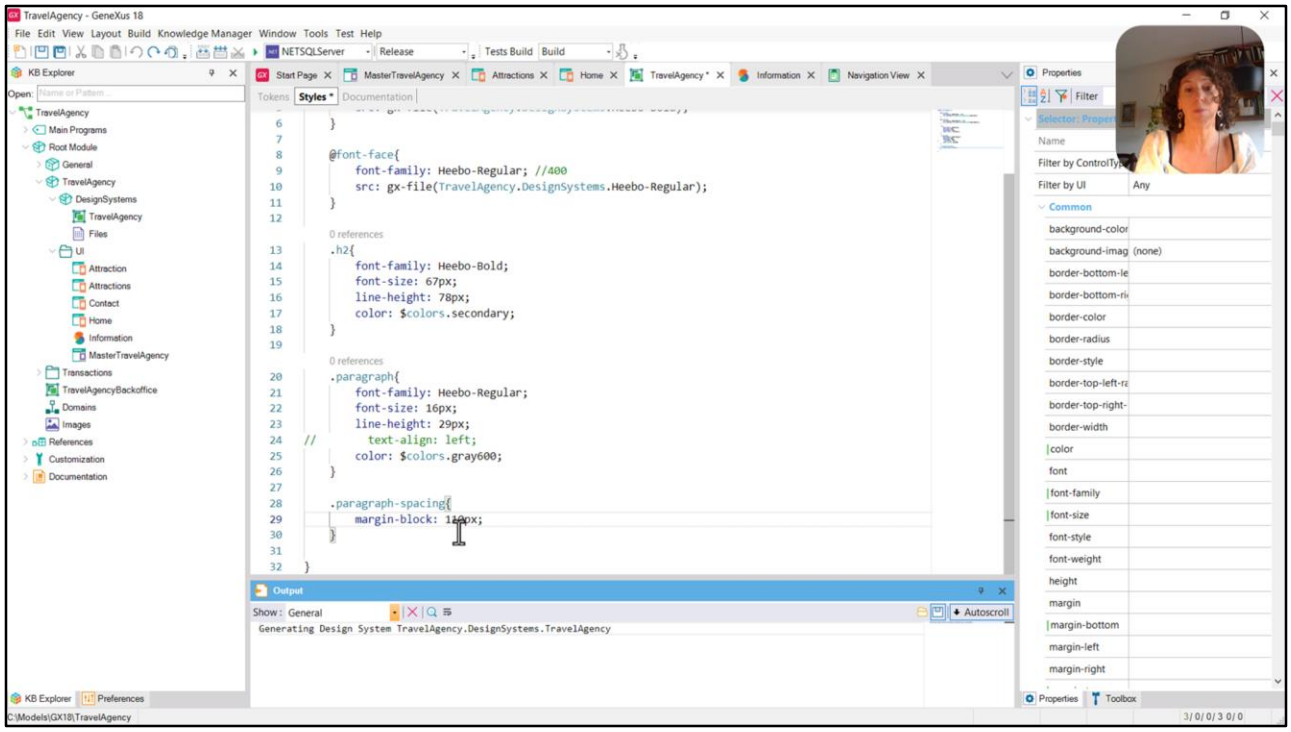


A couple of remarks before the end...

The classes are independent of the type of control to which they end up being applied. By independent I mean that I don't need to apply this class to this or that type of control at any time. Whether all the properties will be applied, or some of them will not, will depend on the type of control with which I associate this class. The ones that will be applied will be those that should be applied, and the others will be ignored.

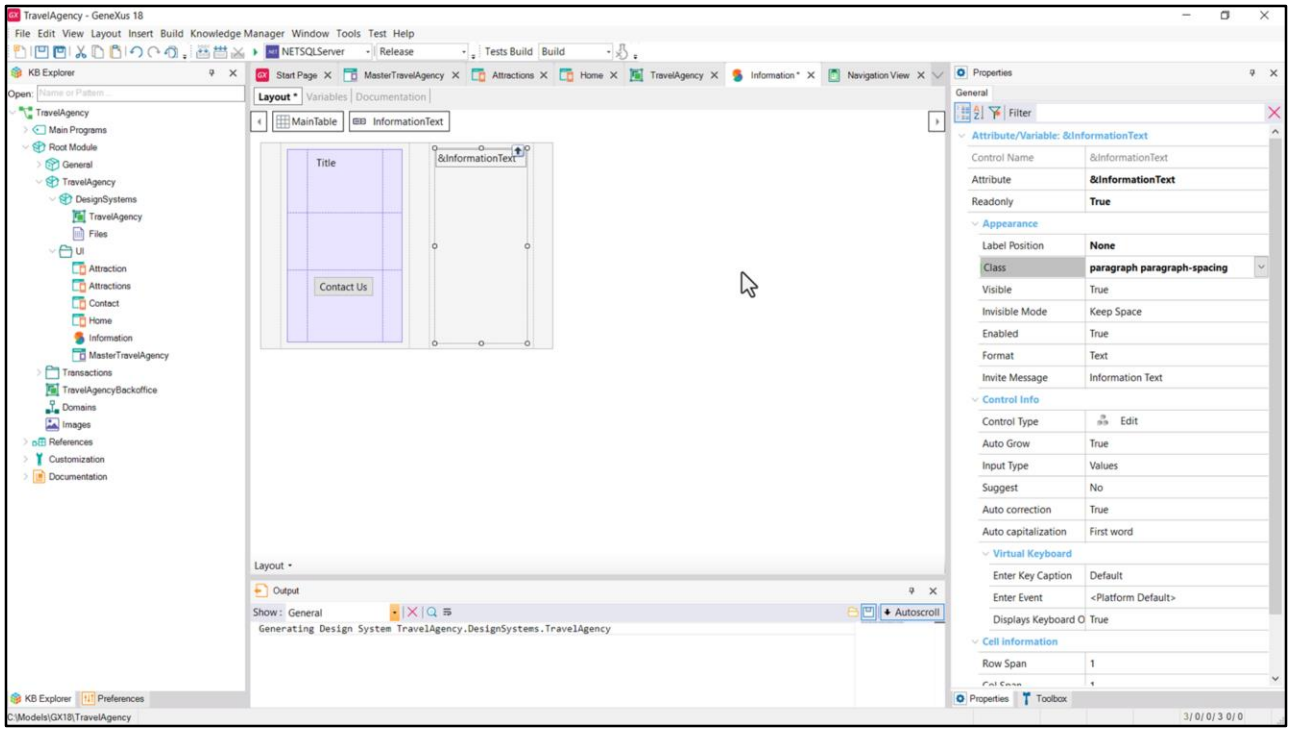
In this case, for example, we apply the h2 class to a control of textblock type and the paragraph class to a control of attribute/variable type, which of course, in this case are very similar control types. But this is not mandatory.

On the other hand, to give a style to a control, it is not necessary to place all the style characteristics within the same class. Classes can be composed, either by defining classes composed of other classes within the styles tab, and that will be done with the include rule; or by assigning more than one class to the control.



For example, let's suppose we want to separate the typographic characteristics of our paragraph from the characteristics that have to do with spacing.

Then what we can do is remove this property here, the one of the margin, and leave the paragraph class clean in terms of typography... and select another class, which for example I can call like this... and it is there, in that class, that I define the characteristics of the margin.



And then what I do (I'm going to save) is come to my stencil, and instead of the variable control having only the paragraph class, I'm going to add this other one. So from now on our control is going to have two classes.

This is the type of decision we will have to make: if we want to isolate, separate, then, some properties in different classes, to be able to compose them later.

Well, having seen all this, we are now ready to start looking at how we style the button. See you in the next video.


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