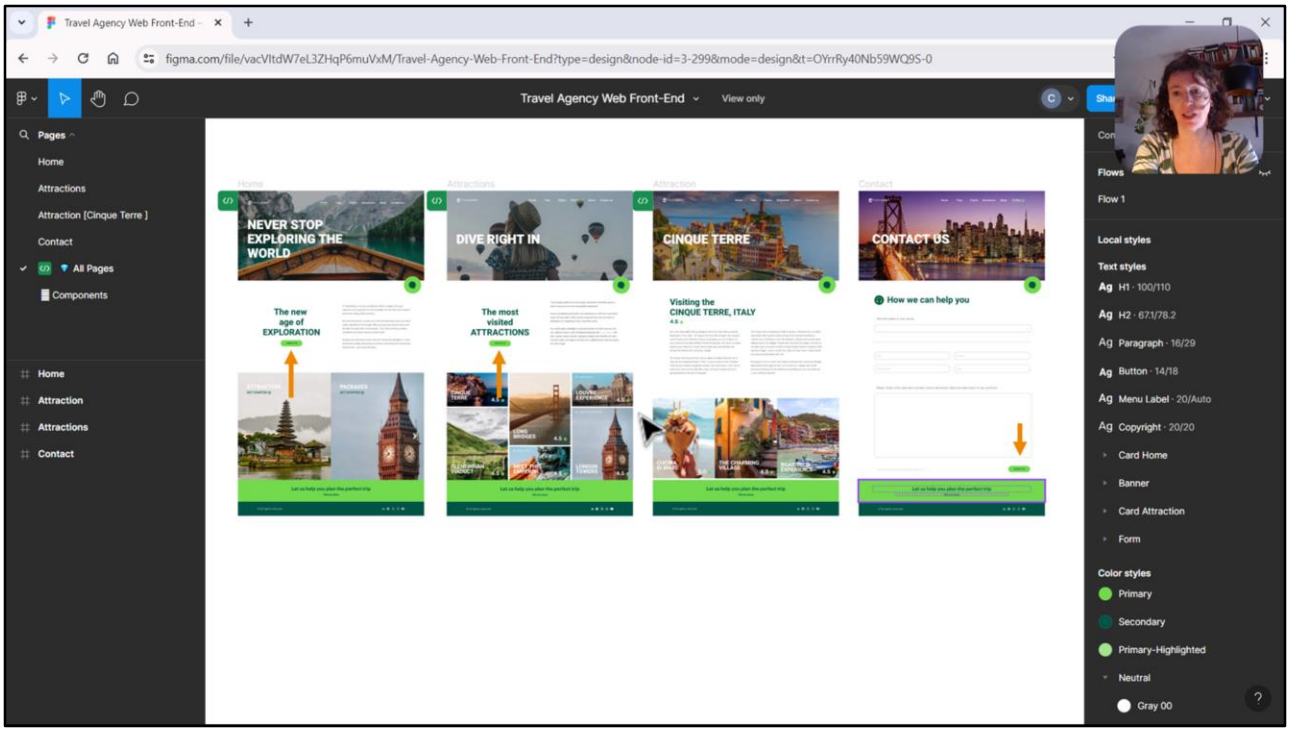
GeneXus by Globant

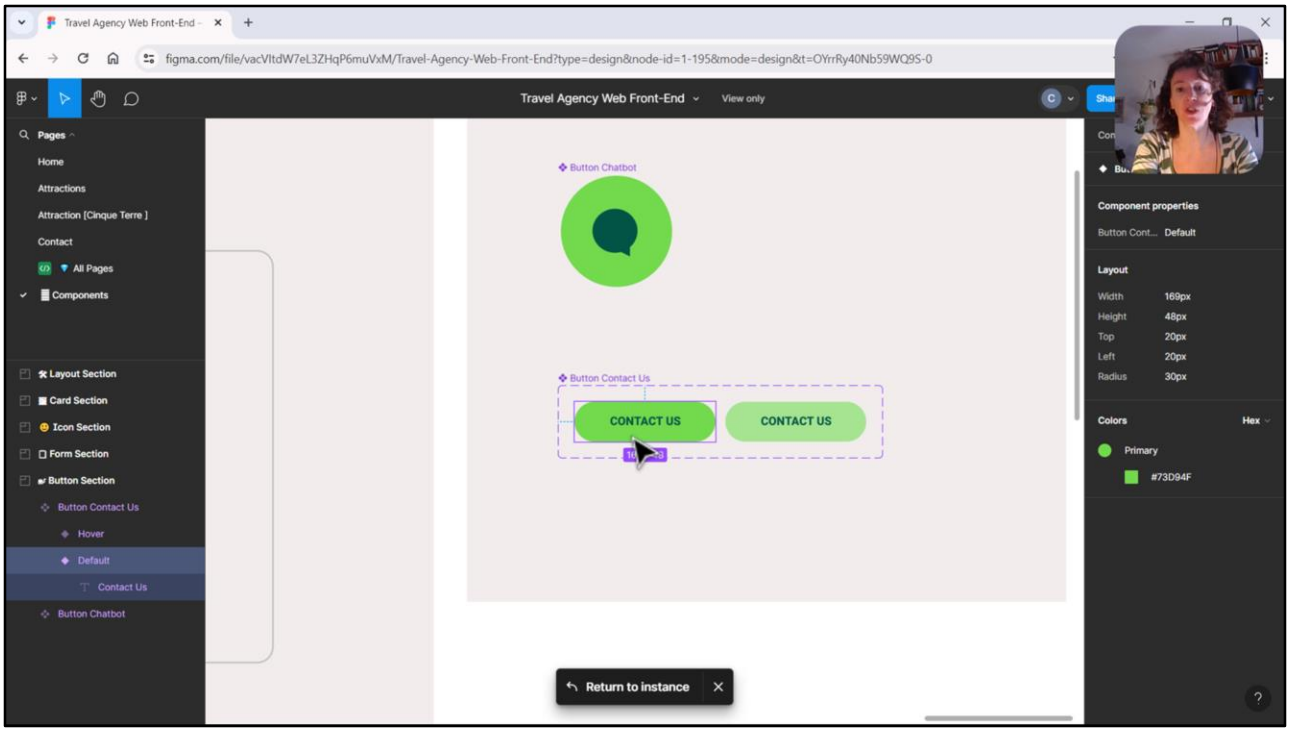
First Layout in GeneXus. Style (cont.)

Button style with behavior

Cecilia Fernández



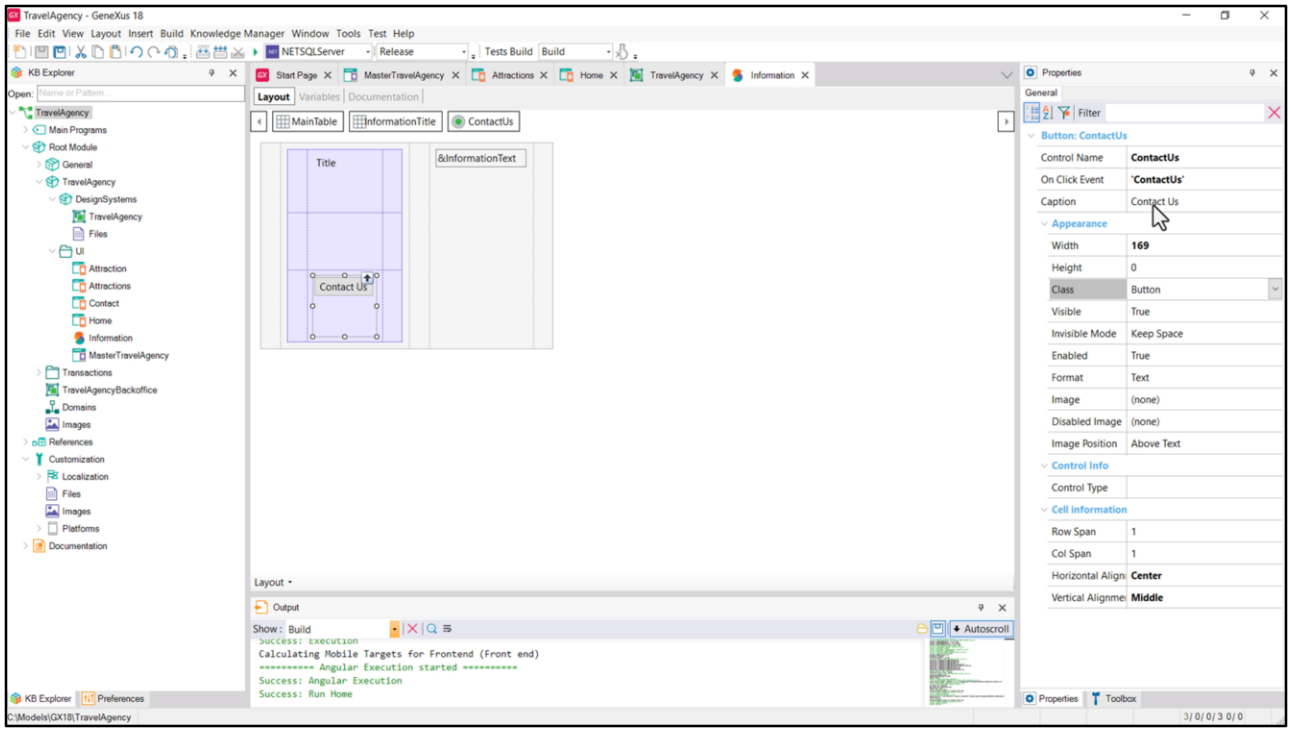
In this video, we are going to focus on the button that appears on these screens, on these three screens: Home, Attractions, and Contact.



Precisely because of these repetitions is that Chechu has decided to model this element as a component. And so here we see it in the components page.

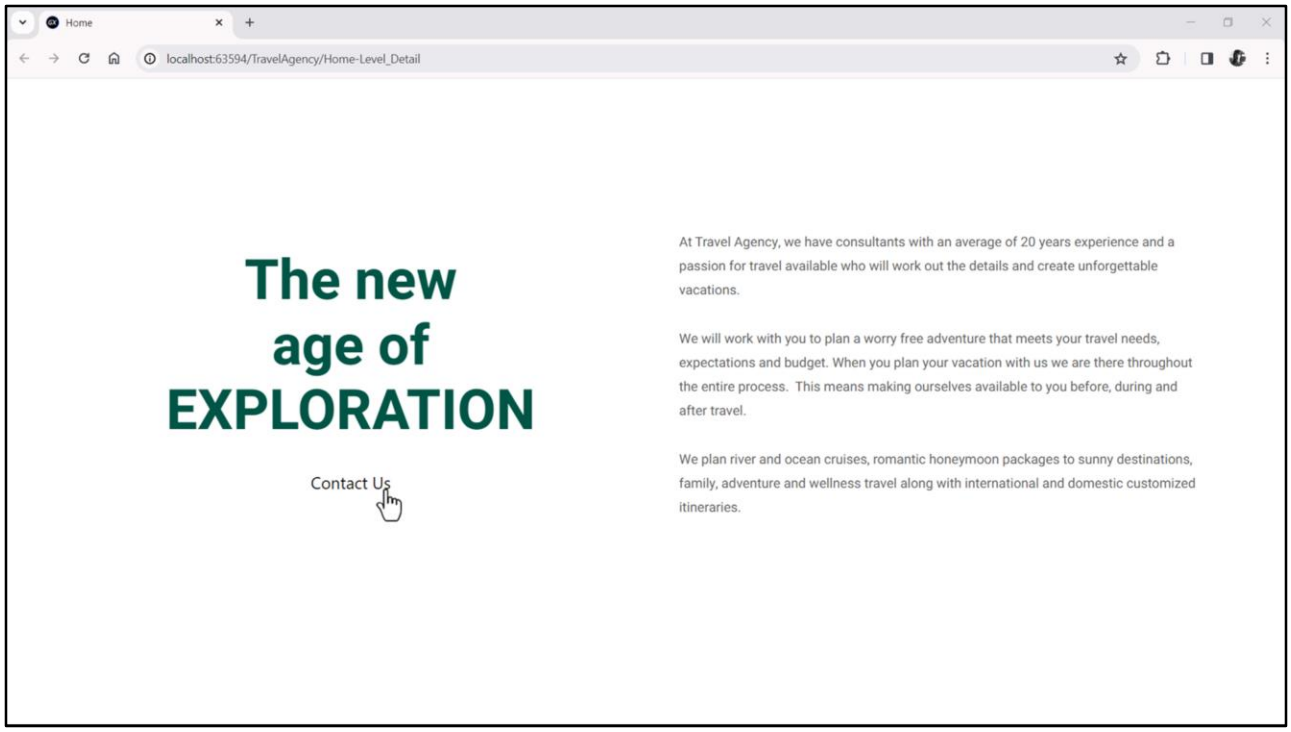
In fact here we are seeing a variant of the button, which is the default variant, that represents how the button is going to look in all our layouts. But Chechu also modeled the Hover variant, when the button is hovered over.

Let's start with the Default variant.



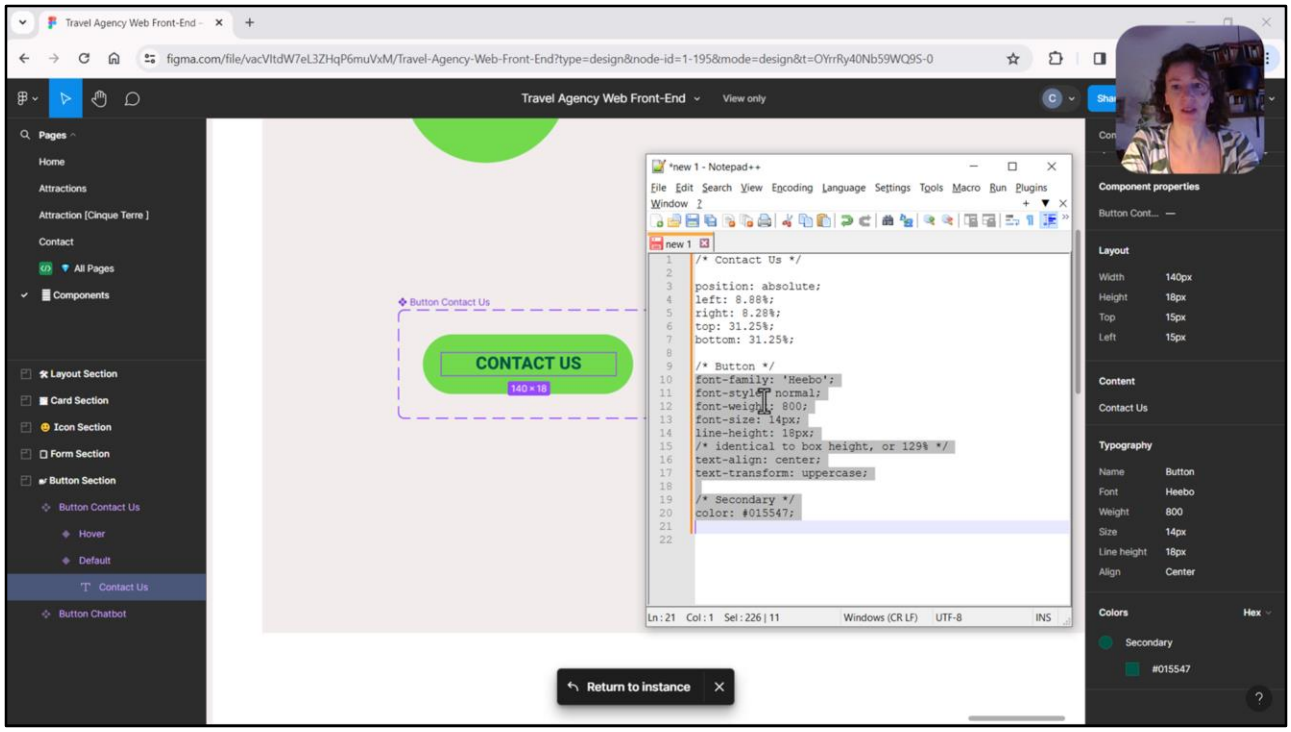
In GeneXus we have the button type control, which was the one we inserted in the Stencil, and to which we configured this Caption.

If we pay attention to the properties, we see that it has an associated class named Button... here we see the button in the Home panel.



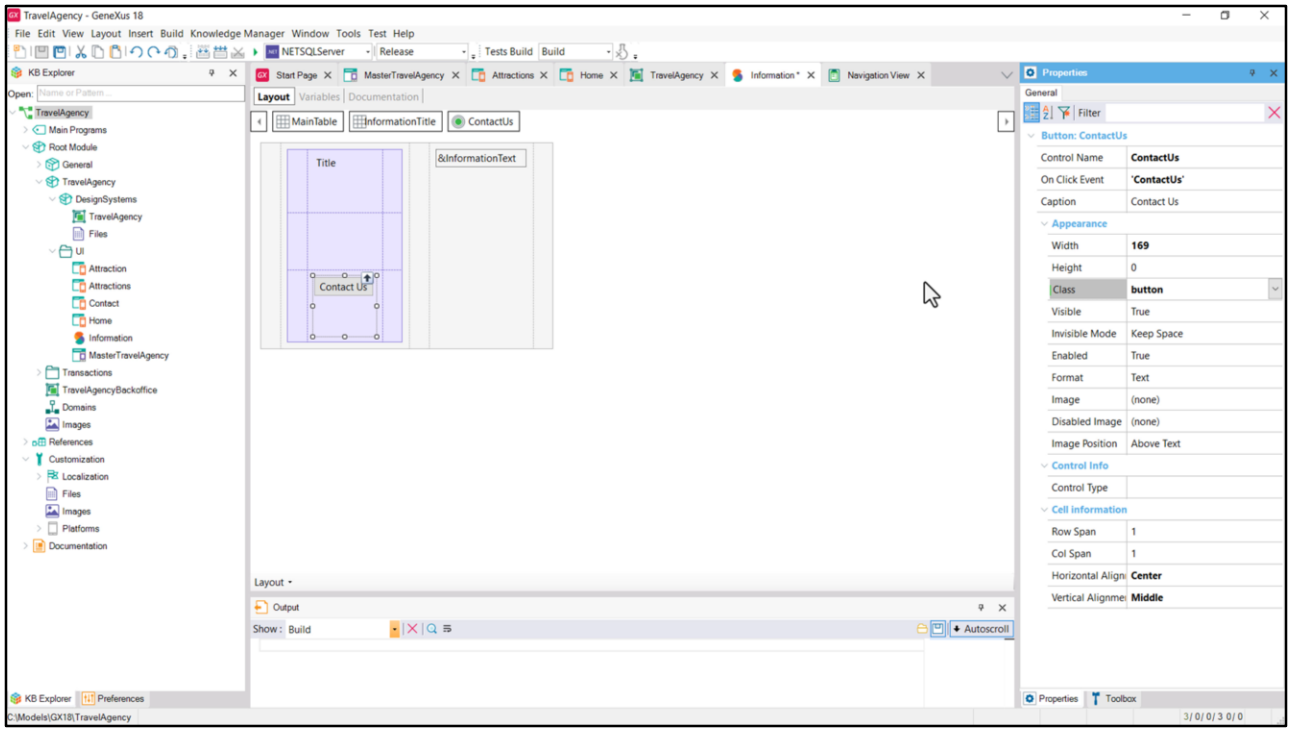
That class is not defined in our DSO, and that's why we see the button like this at runtime...

We can inspect it...

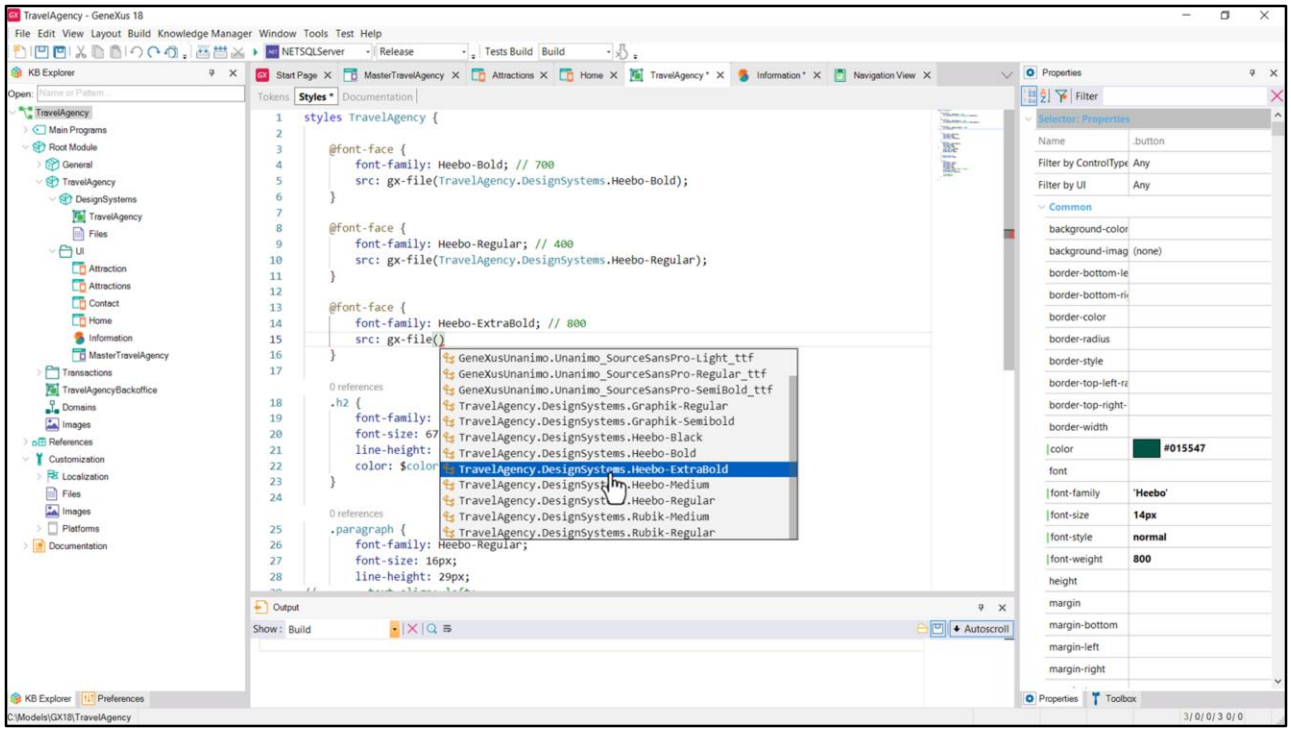


Now, in Figma we don't have a button type element. To build a button we have to do it by grouping two layers inside a container: the background layer, and an overlay layer that will contain the text, which would be the caption in the case of our GeneXus button. In fact, if we look at the properties of this text, which corresponds to the top layer, we see that Chechu gave a name to the typography, she created a typographic style named Button, with these characteristics.

What we can do is copy the CSS code as we did before, paste it in Notepad, copy the properties...

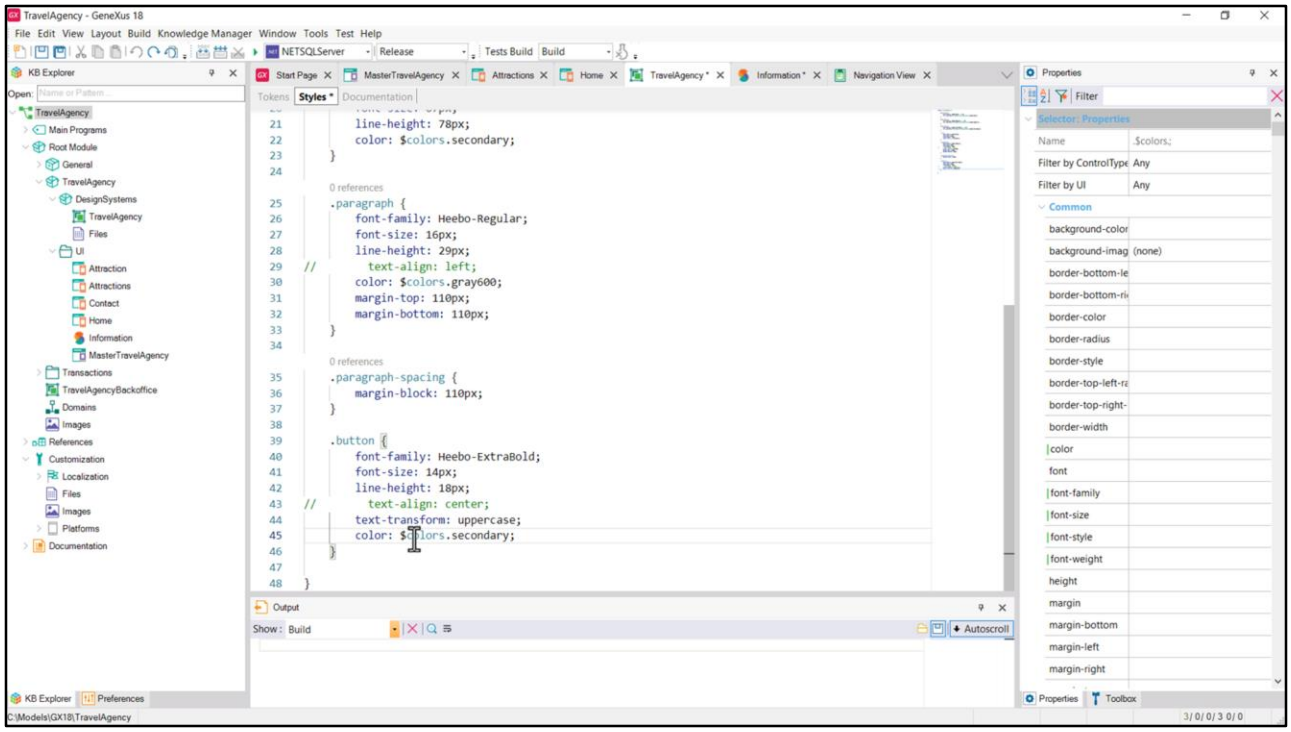


... and go to GeneXus and associate them inside our DSO to those of the Button class. But let's remember that as we are using the BEM naming convention we are going to use the class name in lowercase – since this is case sensitive the class in lowercase is not going to be the same as the class in uppercase. Well, we are going to name it in lowercase...



…and we are going to come to our DSO and type this class in lowercase.

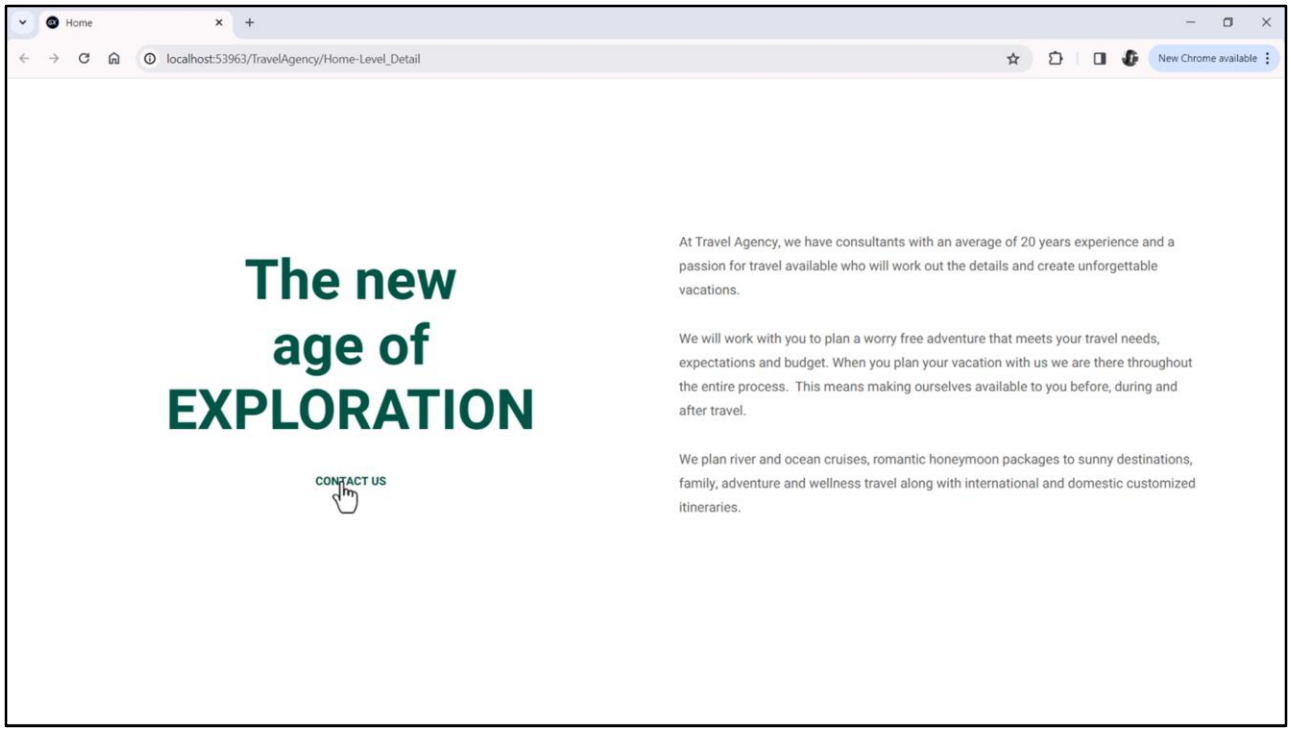
We copy the properties inside here... and we're going to do the same thing we did before for the other controls, from the previous videos; that is, we're going to introduce the font family, this Heebo that we see is 800 weight. Then we come here and specify the font face rule... I'm going to call it ExtraBold... it will correspond to this file that I inserted earlier in the KB.



And now what I do, as we know, is this. Good.

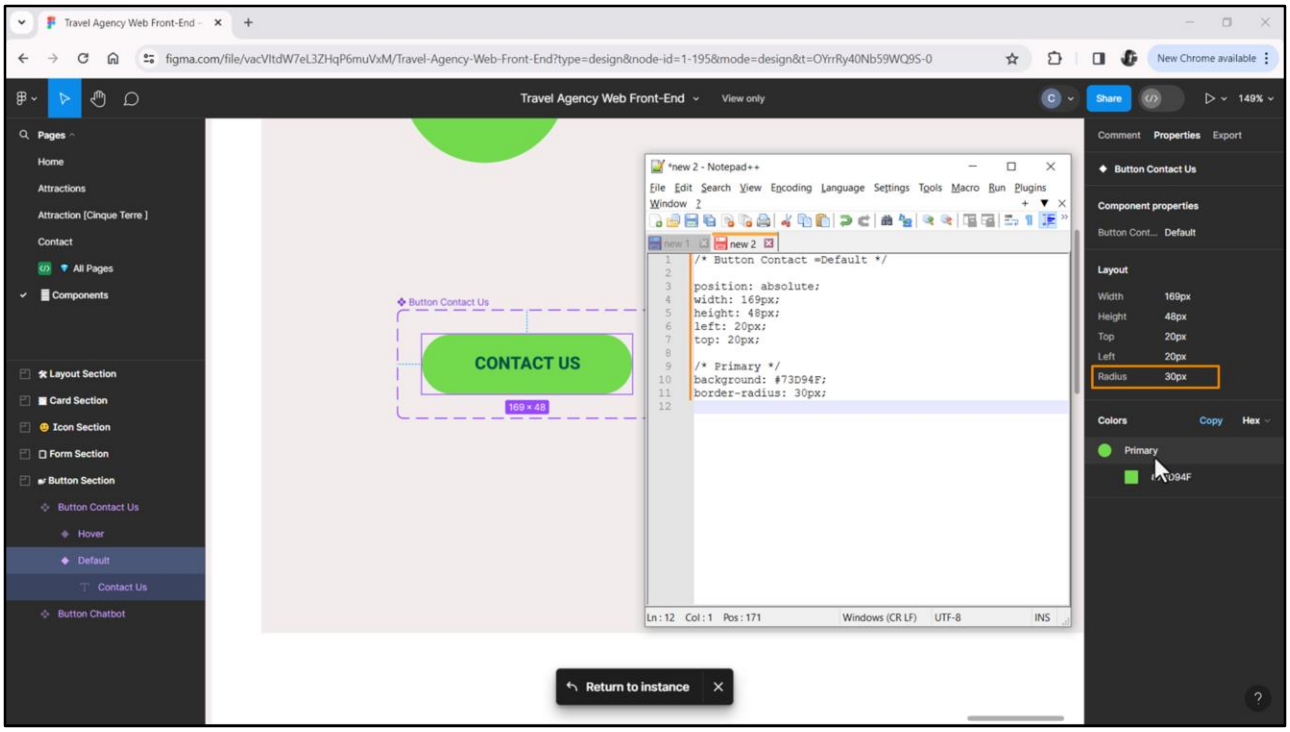
I'm not going to need text-align center, as we'll see. And here we have the text-transform CSS property that will always display the text in all uppercase, regardless of how the text is written. After that, we will not associate the absolute value with the color, but our color token, secondary... carefully... this is wrong here. OK.

Let's try this as it is.



We can see the change, can't we? That everything has been changed to uppercase, that it has the color and size we wanted, and so on.

What is missing now has to do with the background.

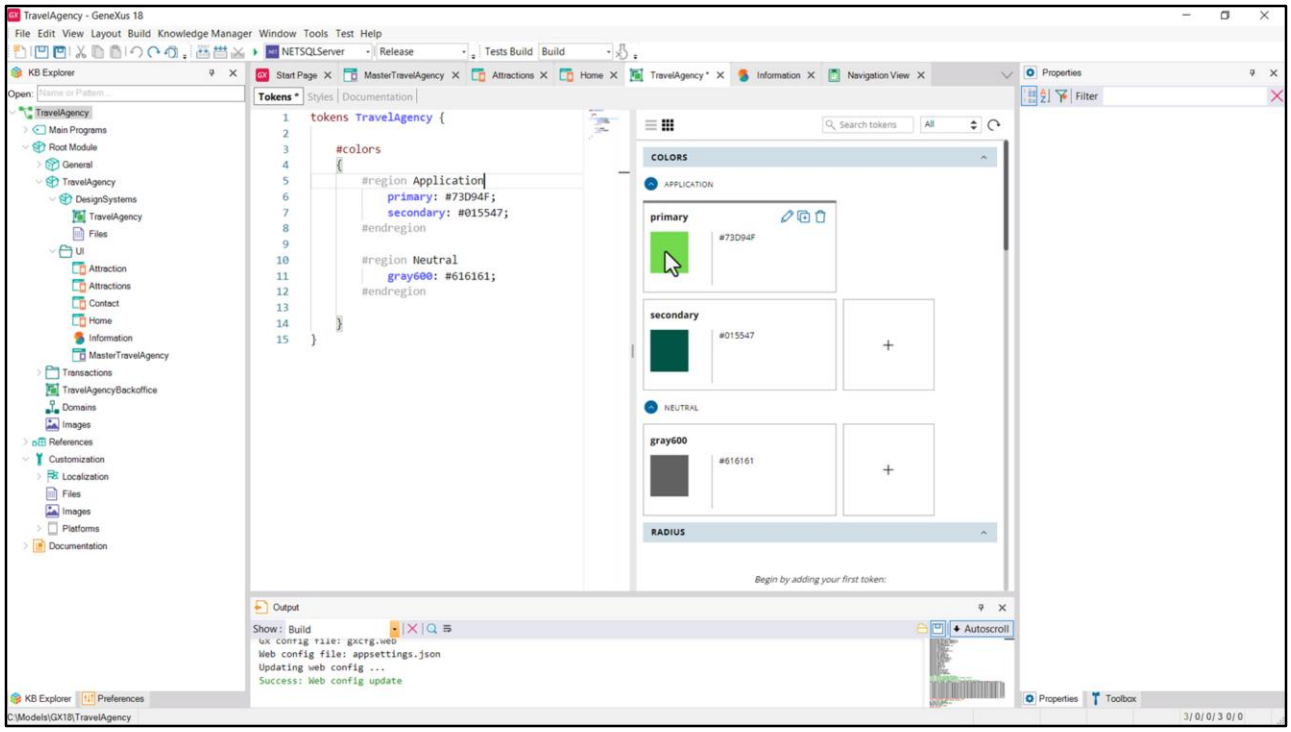


So we came to Figma and selected that background.

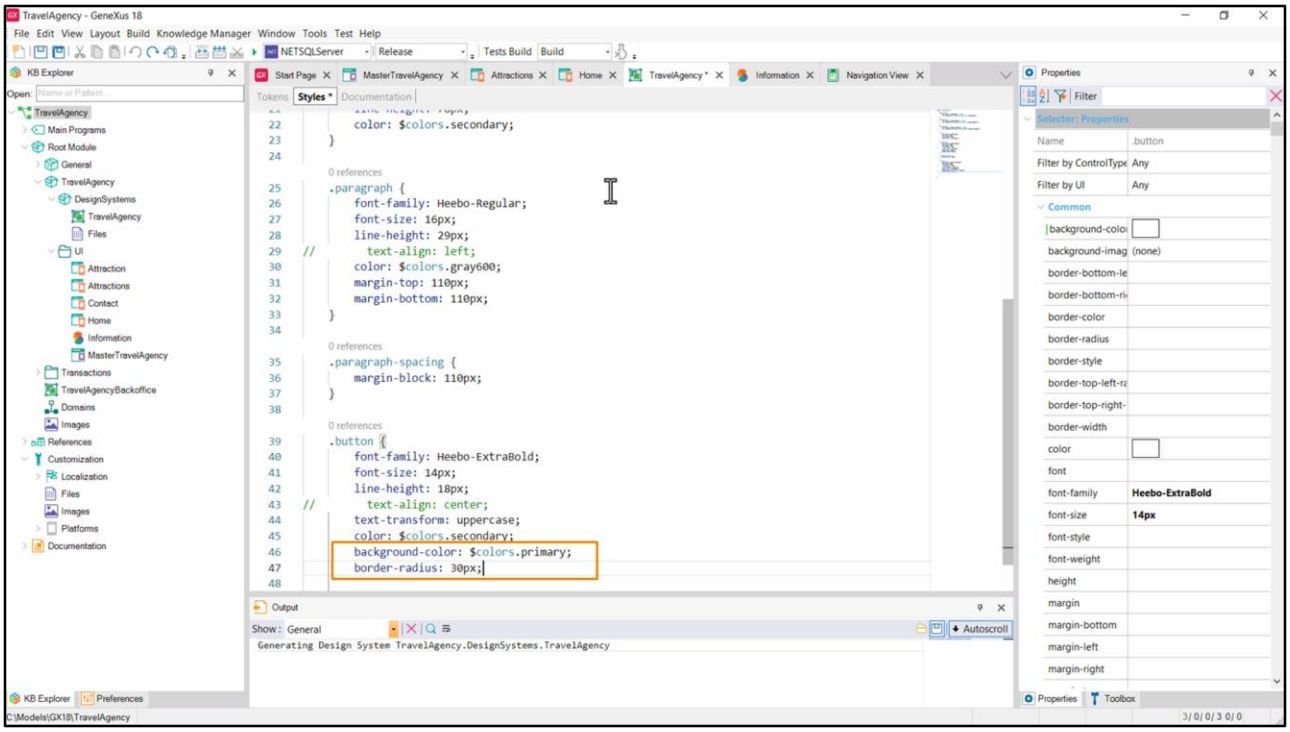
And here are the properties. In particular, let's look at this Radius, which is the one that will give precisely these rounded edges.

Again, let's choose the CSS properties, paste them here... and then we see the color property, background, which is this one here, showing this color, green, called Primary by Chechu, who gave a name to the color style. The first thing we're going to do is convert this into a primary token, with this value. Then, in addition, we need to add this CSS property to our button class.

So... let's copy this first...

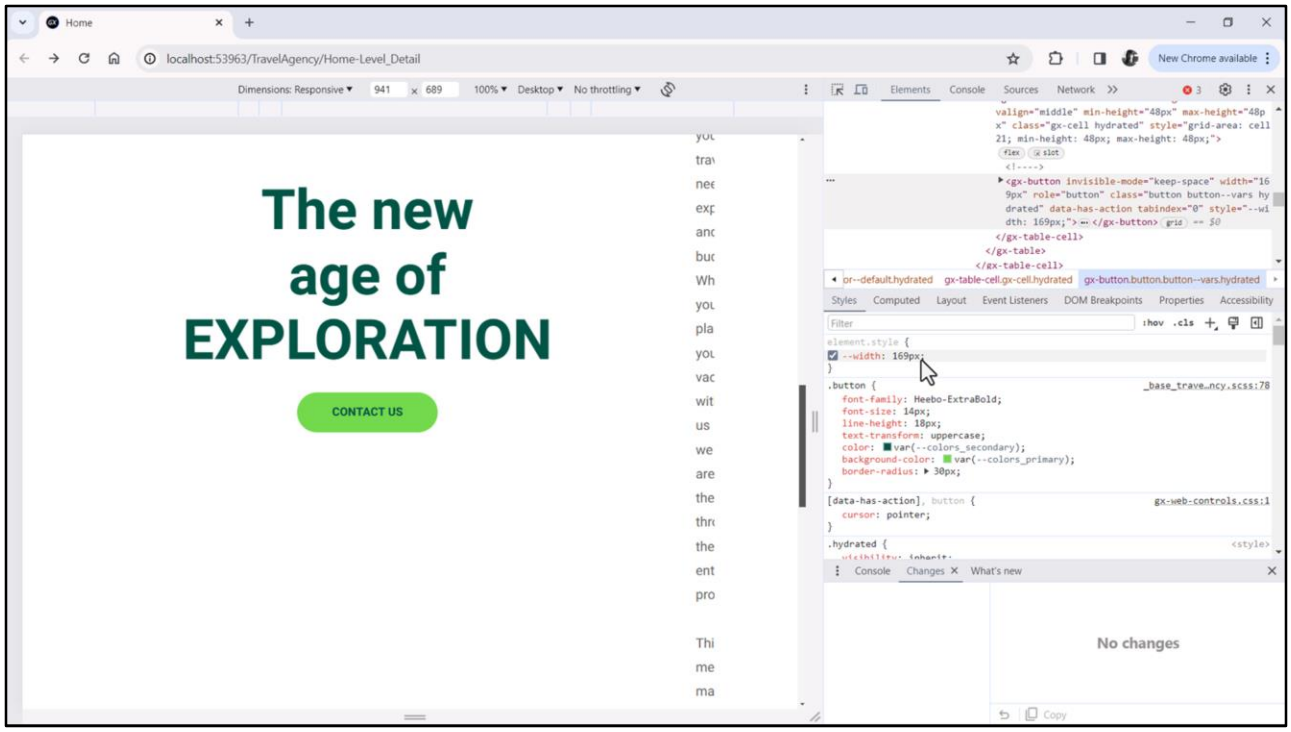


...we come to our DSO, to the tokens tab and add this token, primary.



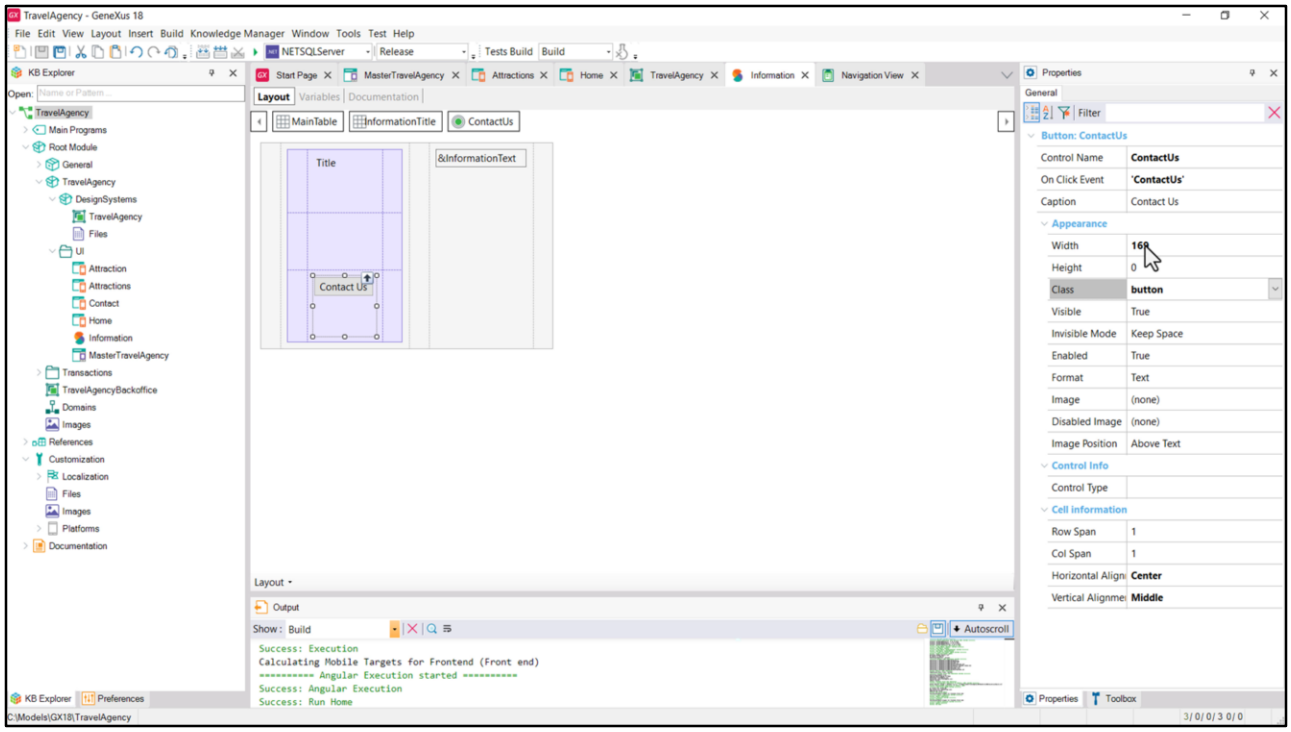
Then we go to the class, add the background-color property, and we're going to copy this other one.

Let's try this now...

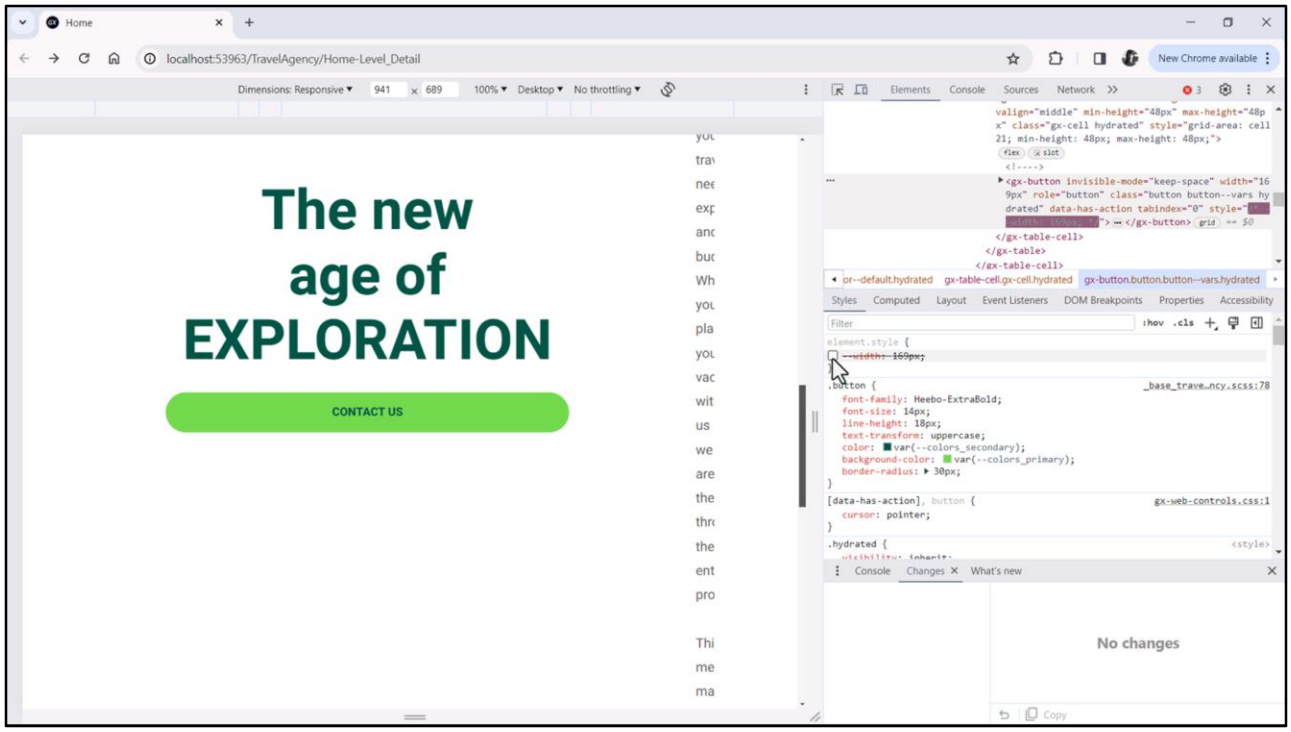


Well, this seems enough.

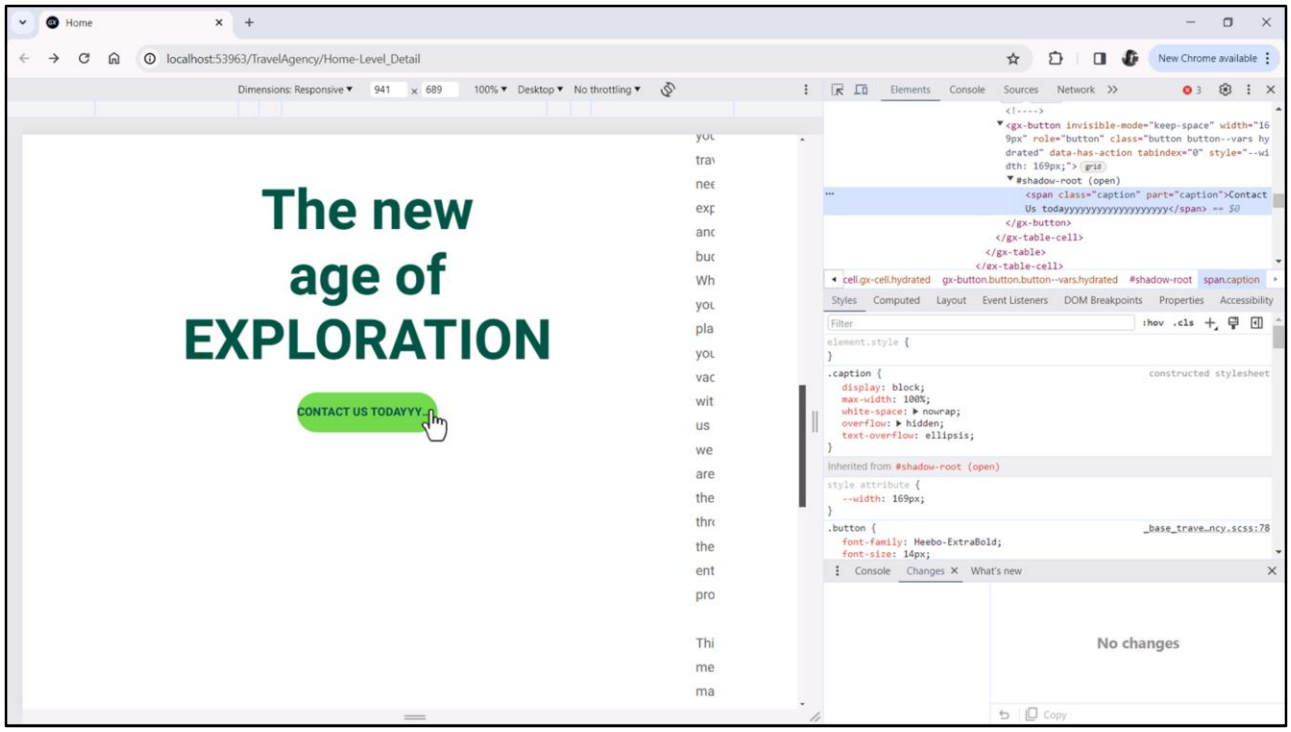
However, let's imagine what would happen if we translated the application to a language in which this text was longer. What would happen with the width of the button? If we look at it we see that it is taking as width this value, 169 pixels. Where does it come from?



From the value we had given to the Width property at the level of the button inside the stencil. We had preset the width of the button to this value. And we had said that if we left the default value that was 0, the button would expand to occupy the width of the cell in which it was placed.

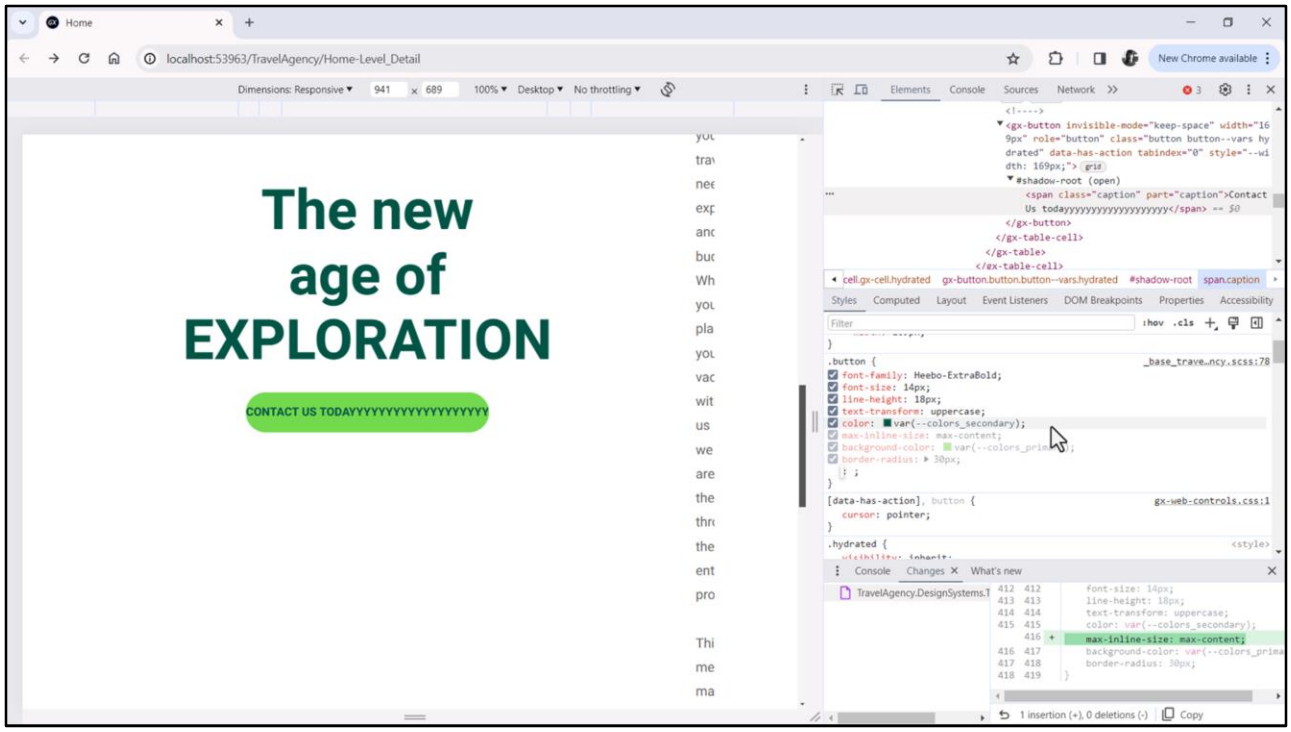


And we can see it very clearly right here if I do this.



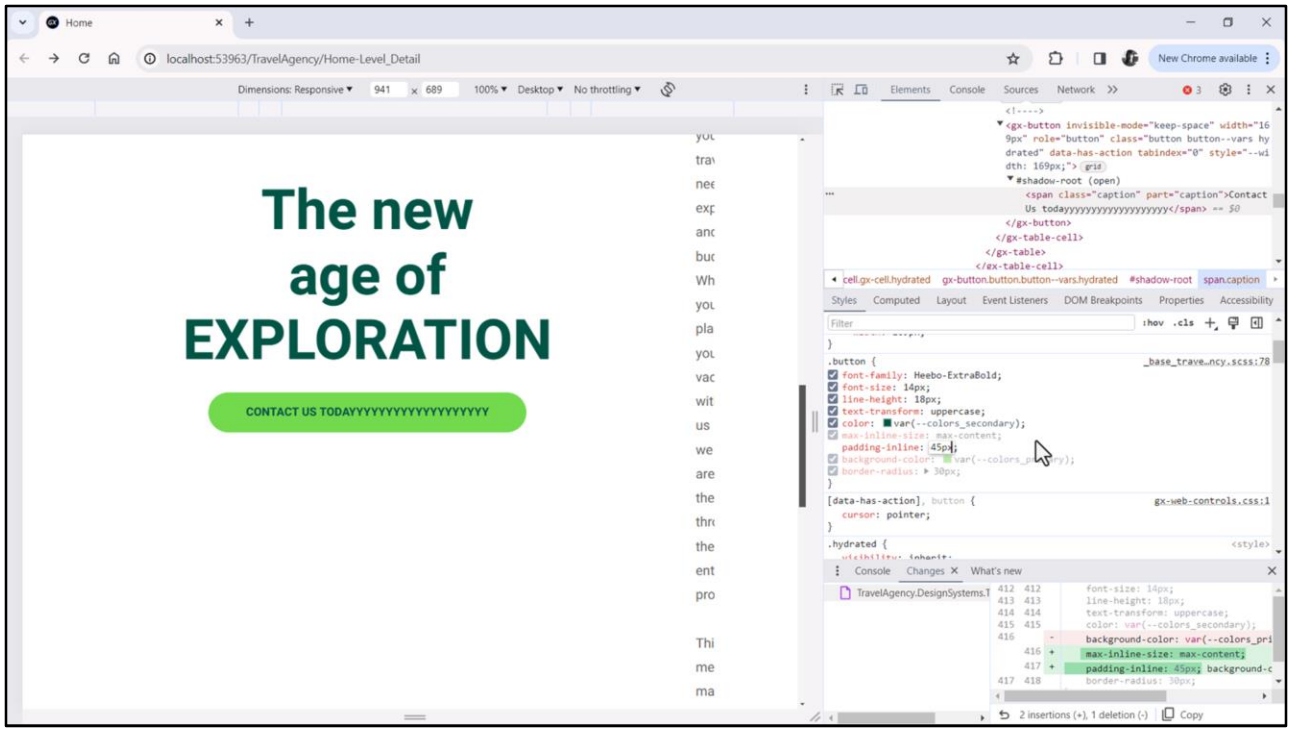
Okay, but, I was asking, what happens if I add to this Caption, if I make it wider. For example..

Well, what we are seeing is that the text is expanding to the limit of the button's width. And there it is placing an ellipsis, that is, truncating, cutting off the text... and this is not the behavior we'll want. What we will want is for the border of the button to expand too, and to leave some space on the sides. That is to say, we will not want the fixed width of 169 dips (that is very well for the “Contact us” text, with those dimensions, but not if we translate the application).



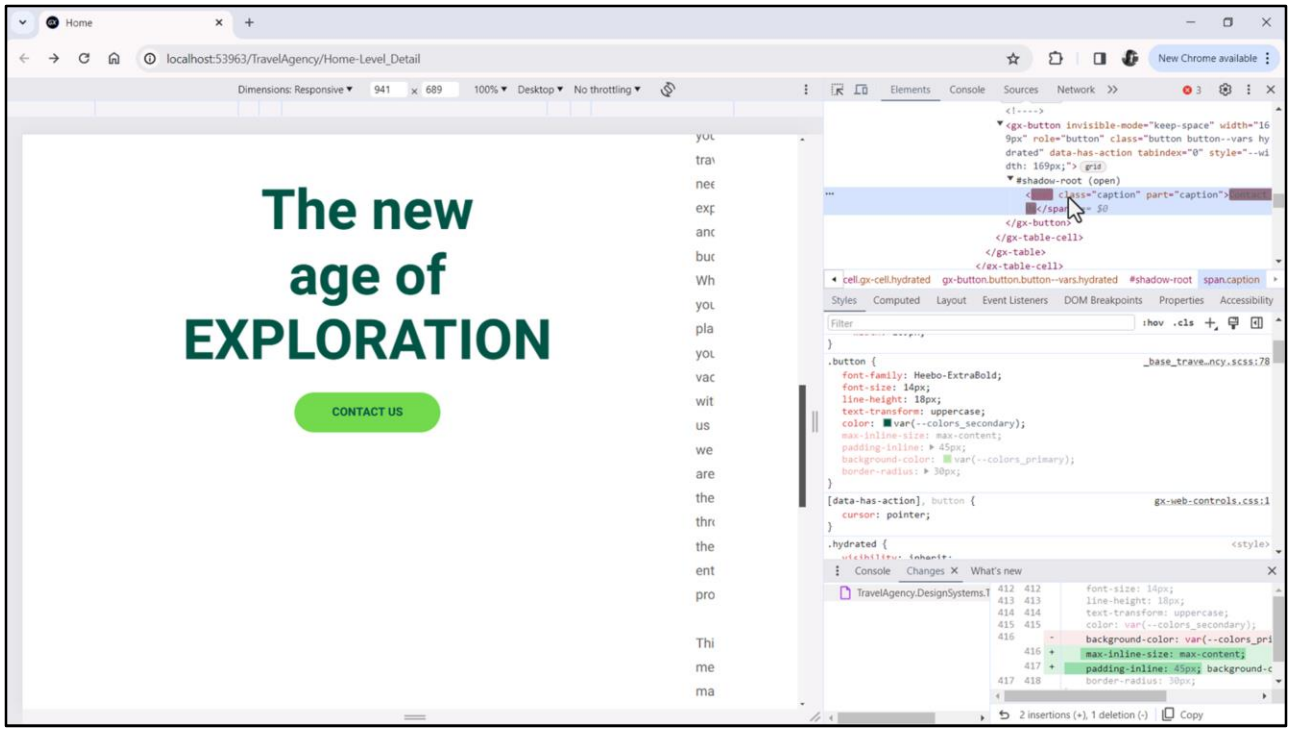
So... let's try the following. I come to the properties of the class and I am going to add a max-inline-size property, and set that property to max-content. That is to say, the maximum inline size, in the horizontal direction, corresponds to the maximum of the content.

And here we are seeing it: how the button expanded to take all the content.



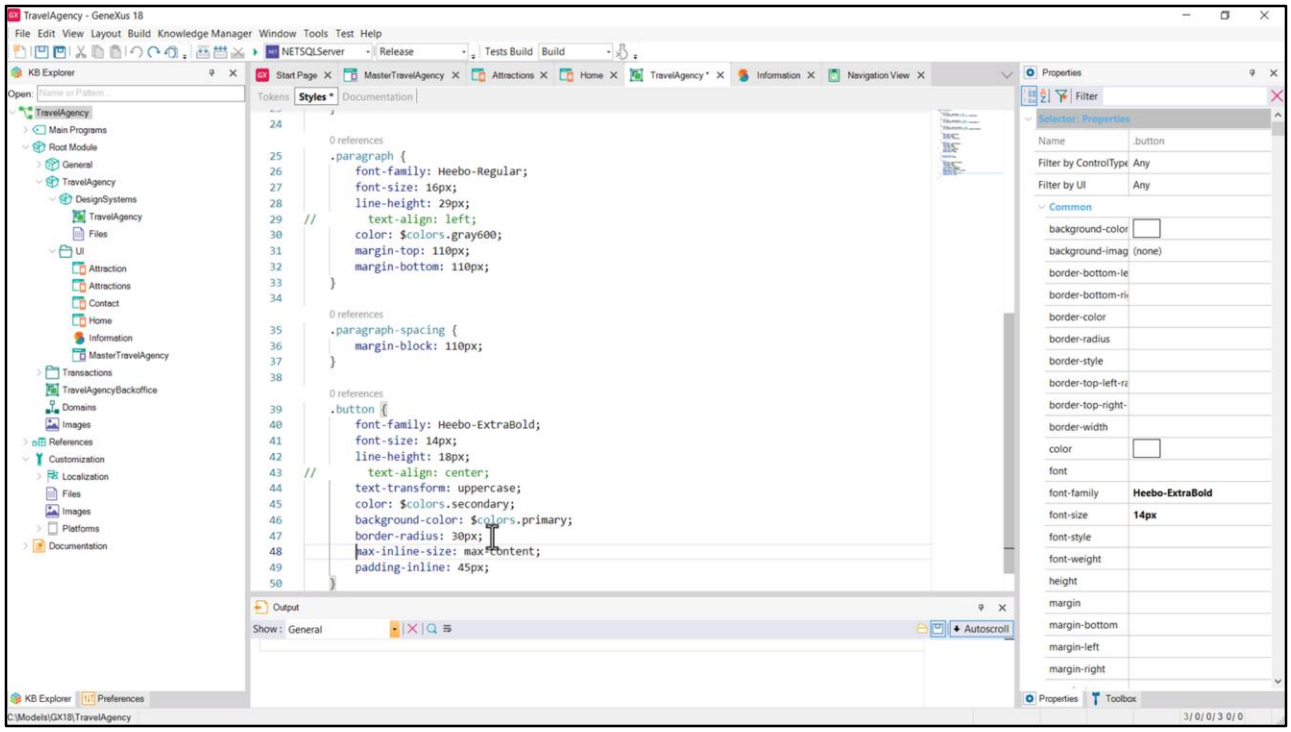
What's still missing? To leave a padding, that the content also has a padding... padding on one side and padding on the other. So, for example, let's try, padding-inline (to use, again, as we suggested before, the logical properties instead of the physical ones), and for example let's enter 45 pixels. And there we see it.

It has 45 pixels with respect to the start, 45 pixels with respect to the end, and the width of the button is what it needs to be for all that content to be wrapped.

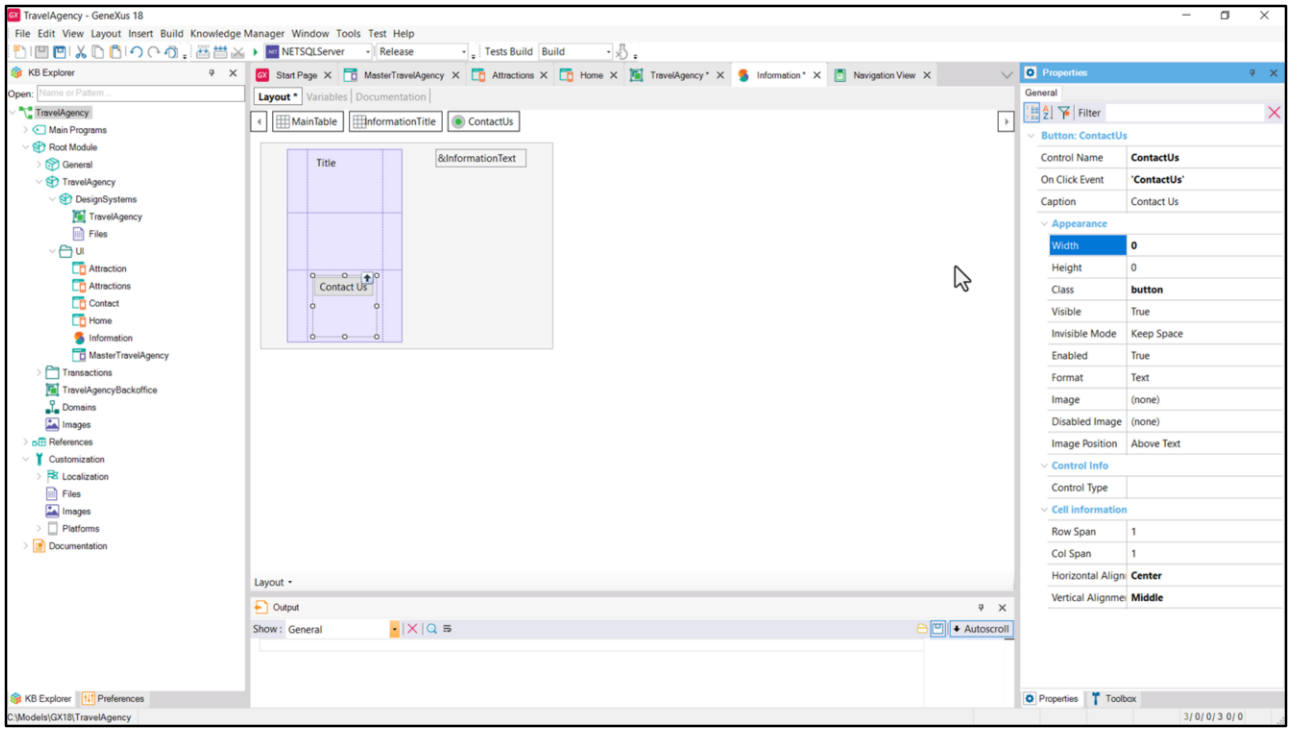


Note that if we now leave the caption as we wanted, Contact Us, the difference with what we had before is not noticeable.

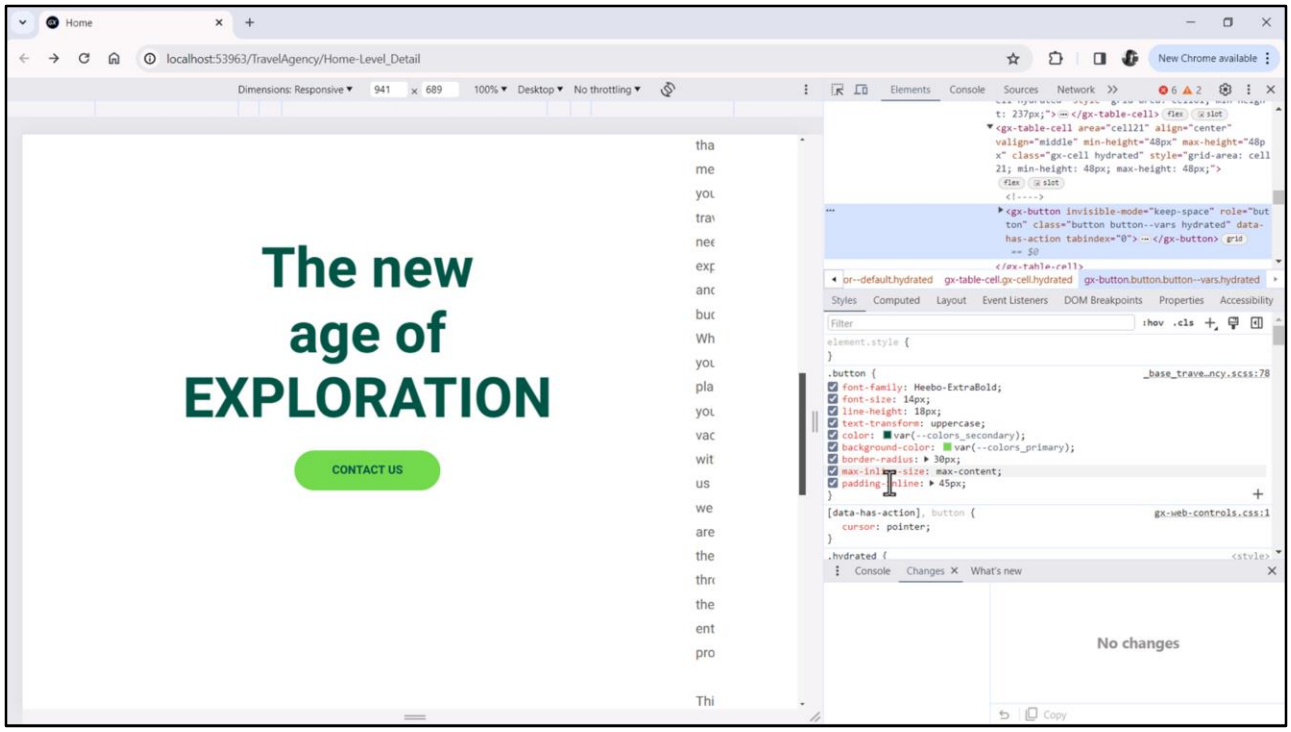
However, in this way, if we translate the application and change the text width, the button will be resized so that it always looks good.



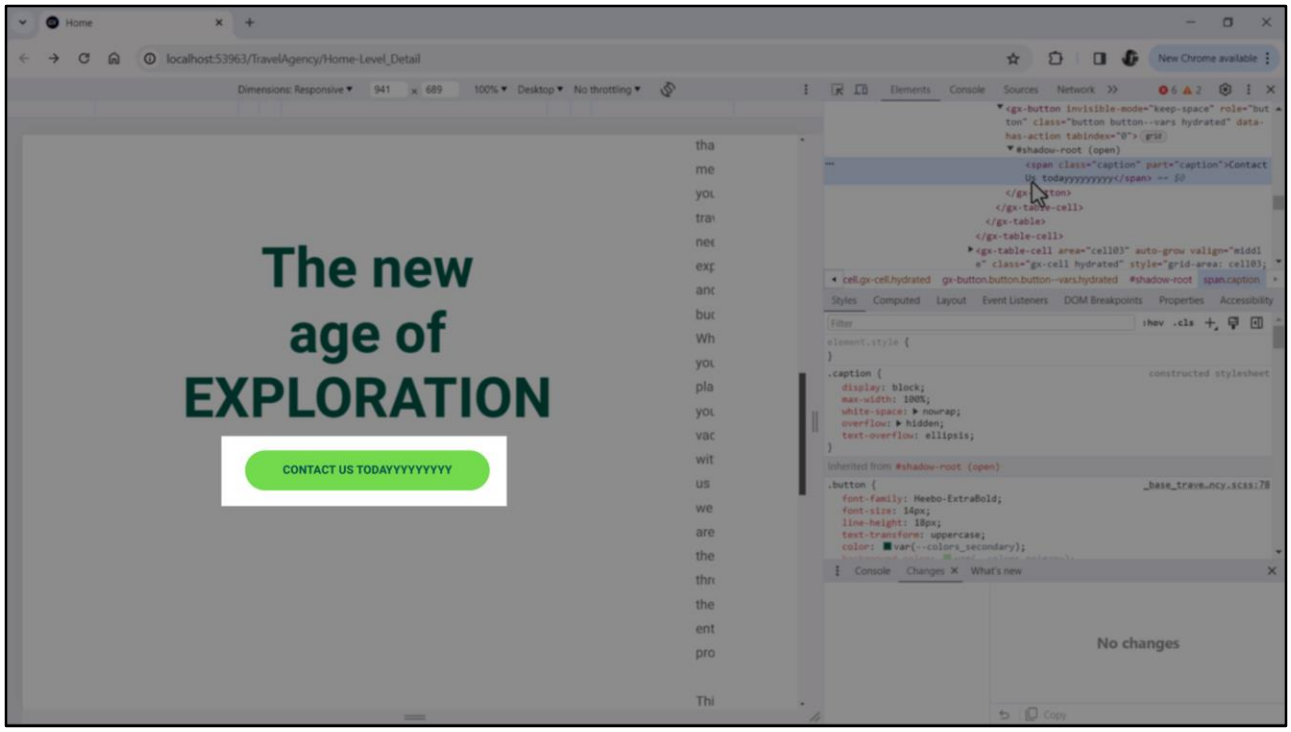
Well, then let's take these two properties to our DSO.



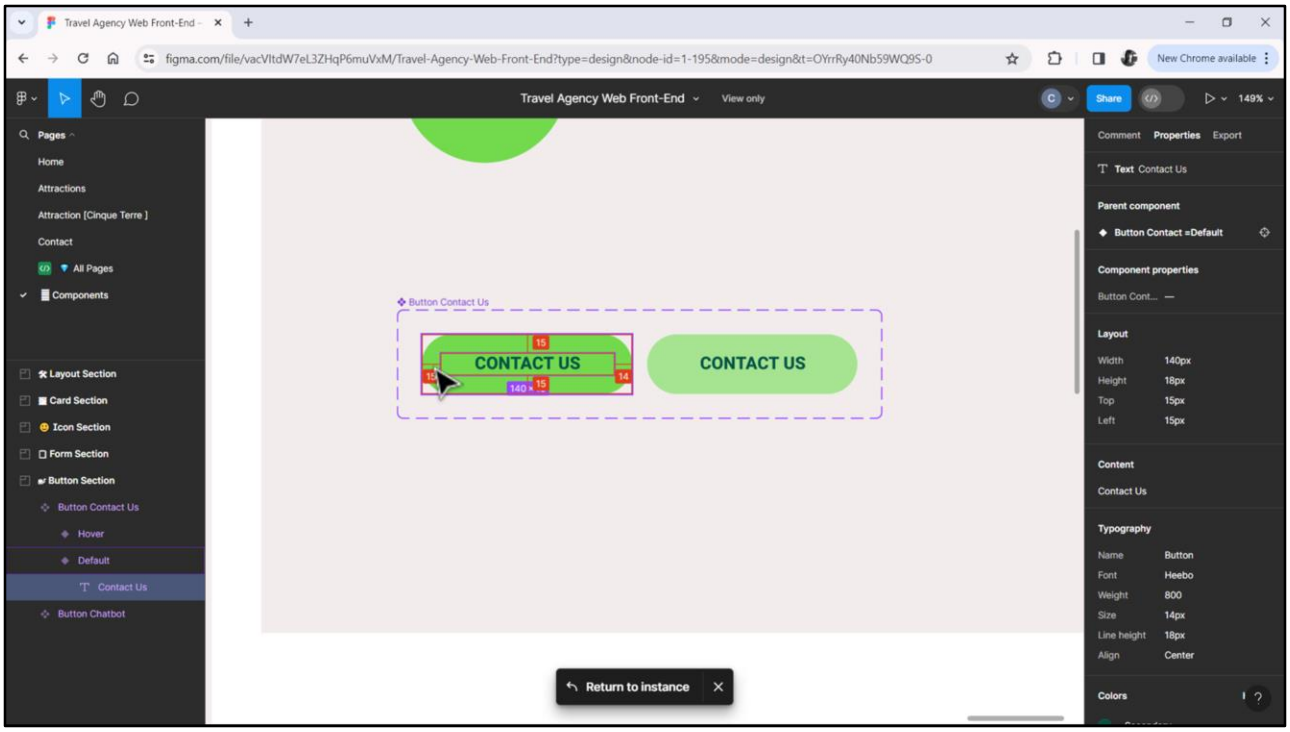
And let's remove the Width property, let's leave the default at the level of the button control in the stencil.



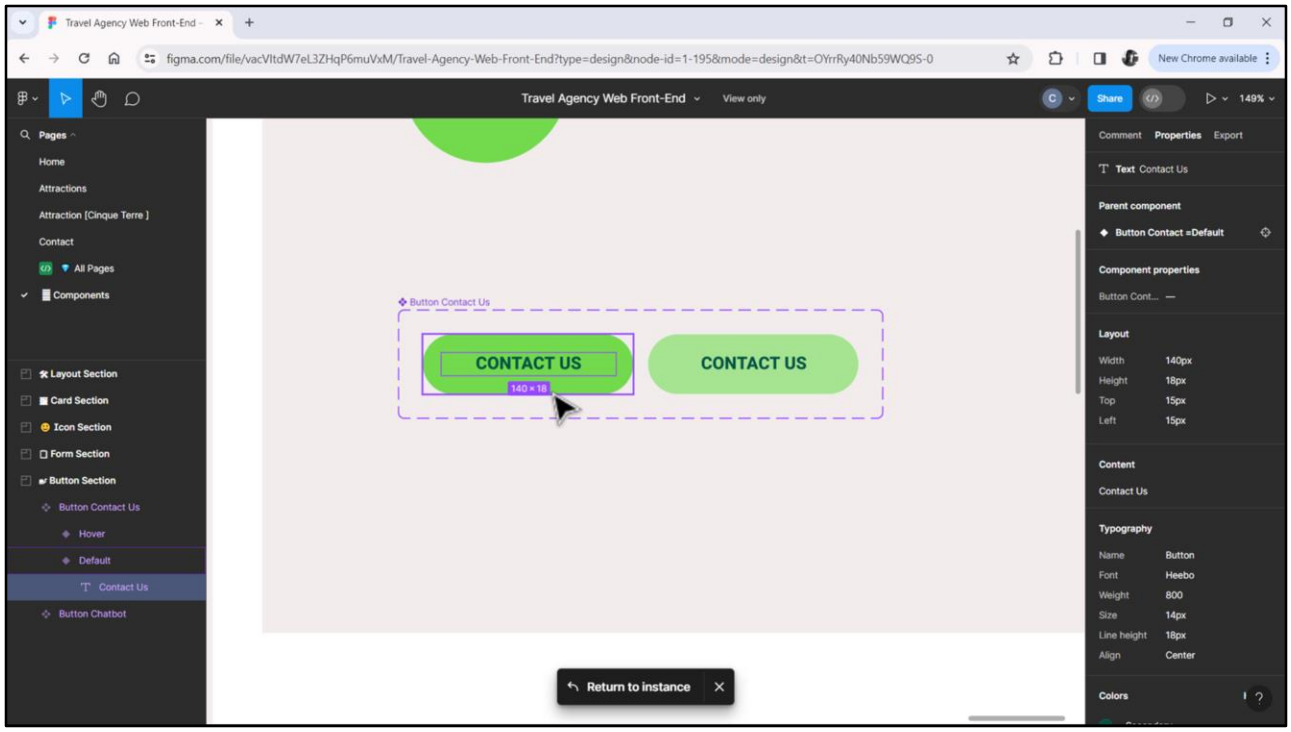
Let’s try it. As we said, it's not noticeable... but now we see the properties incorporated in the class, we don't see the width, and if we modify the caption...



...we see how it automatically resizes, as we wanted.

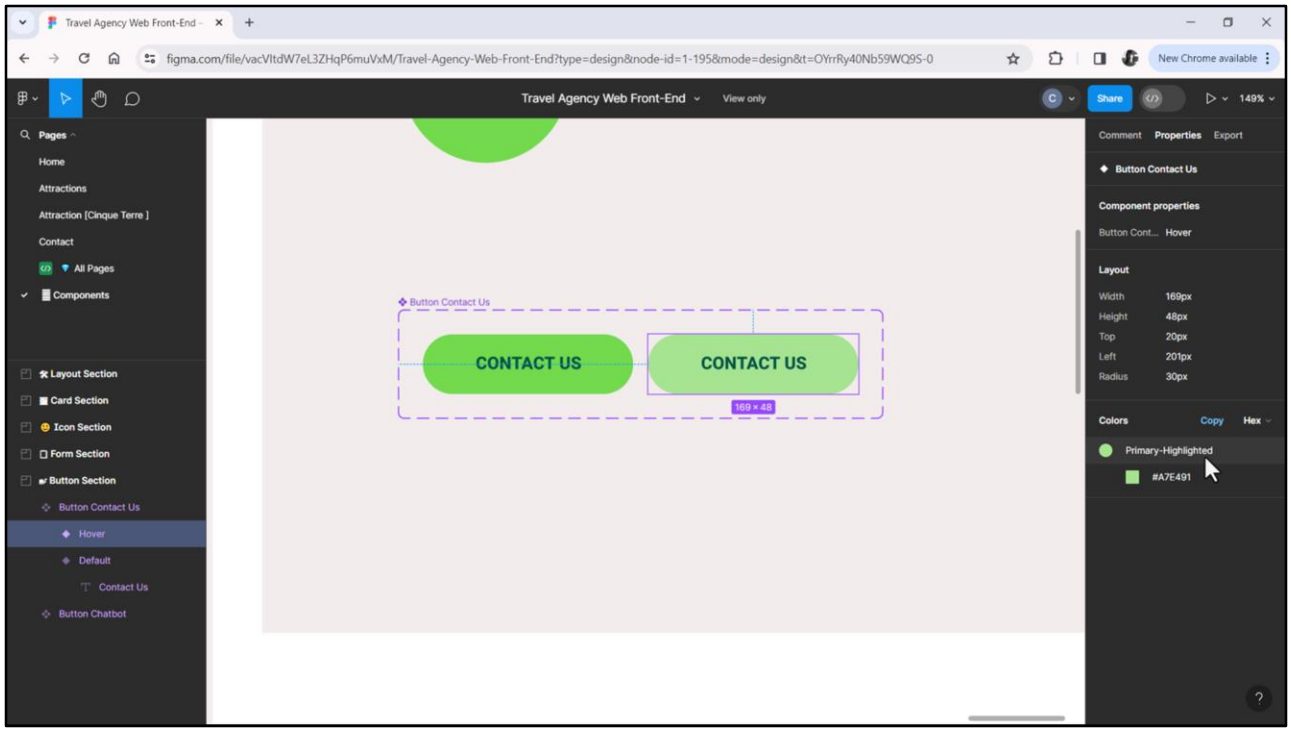


Could we have noticed any of this in the Figma file? Well, not as it was expressed. In fact, Chechu defined the text box with 140 pixels width, that is, a fixed width, leaving 15 pixels on one side and 14 pixels on the other side. We can also see 15 pixels at the top and 15 pixels at the bottom, which we have not considered. Why? Because we gave the button a height of 48 pixels, which corresponds to the sum of these 15 plus these 15 plus...



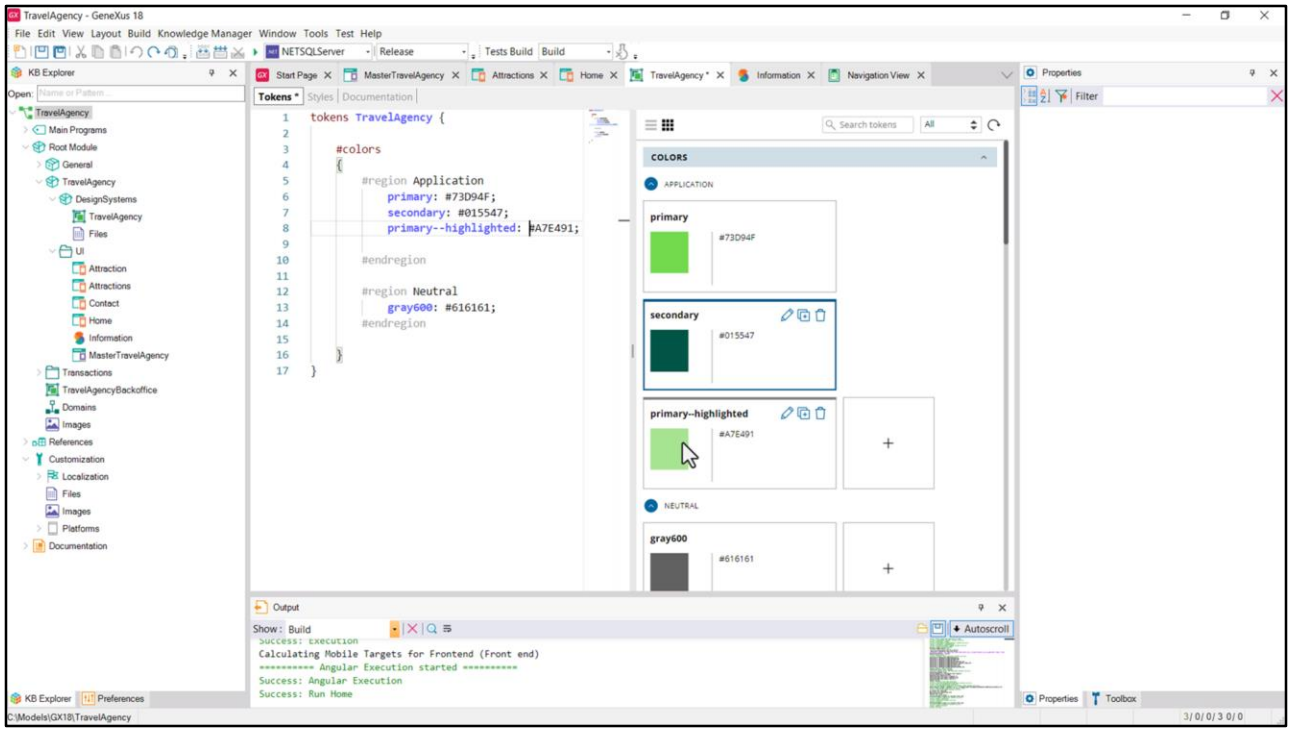
... these 18 here of the text. These are not considered because the button will not grow up or down according to the translation. What the translation will do, if anything, is to expand left and right, but not in the other direction.

In short, in this case, our implementation would have corresponded to a design in Figma in which this width was hug, that is to say, wrapped the content, and also had a padding-left and a padding-right, if I'm going to say it in physical terms, of those 45 pixels that we set.

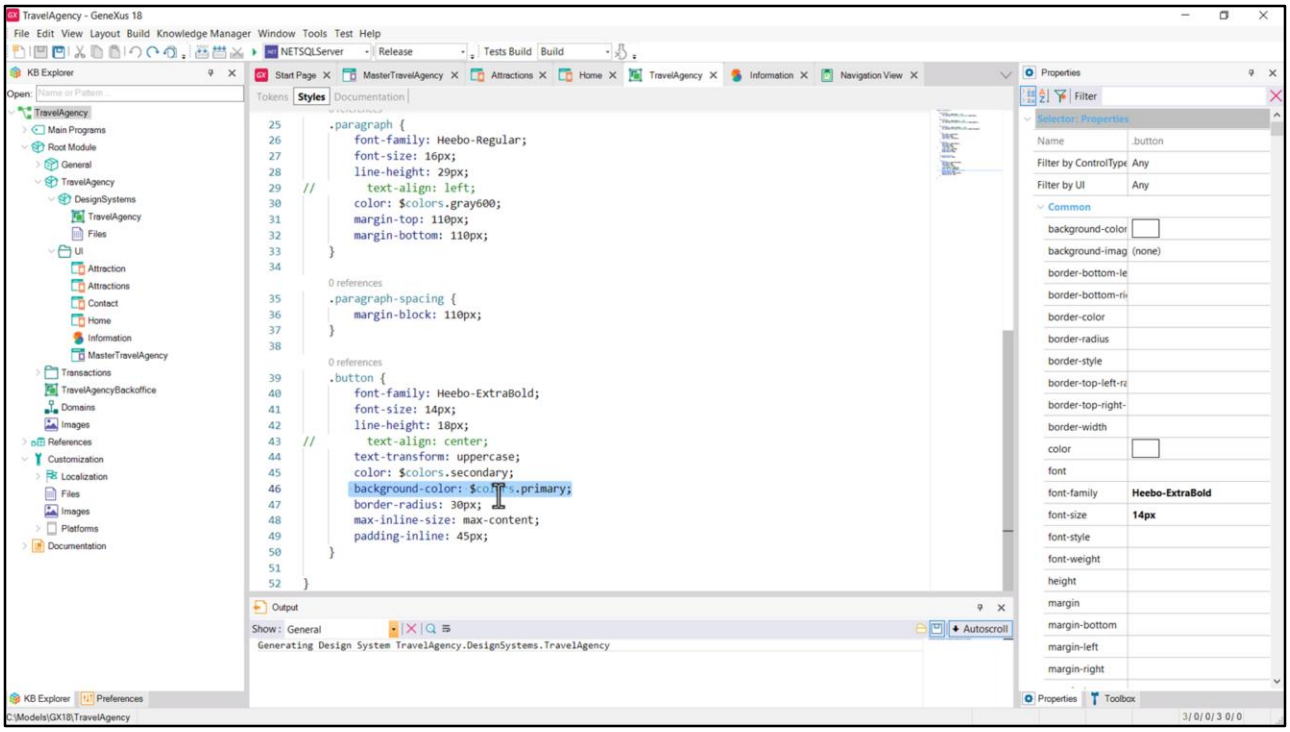


Well, now we can study how to make the button change its appearance when it is hovered over; it could also be when it is clicked on, when it is activated, so that it takes on this style, which, as far as we can see, the only thing that differs from the default style is the background-color. It takes on this color to which Chechu gave this name, Primary-Highlighted.

Then I copy it...

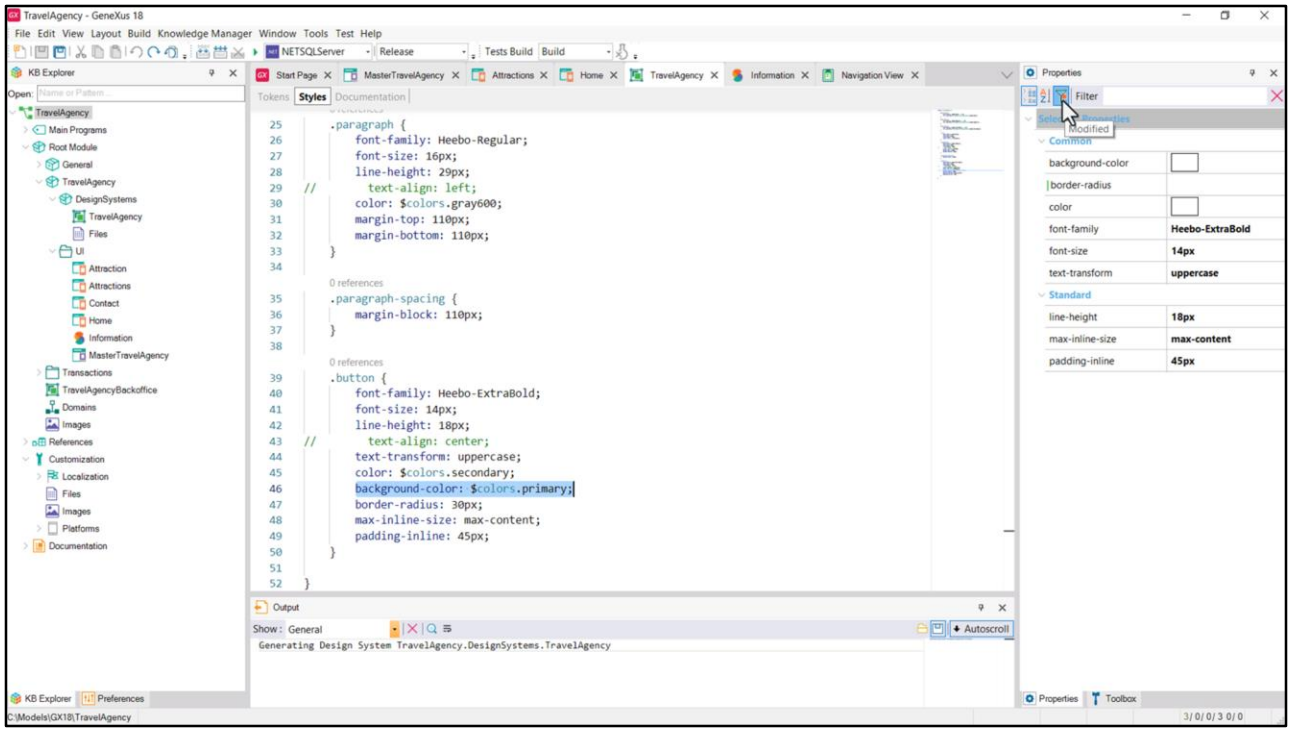


...and in the tokens section I add...



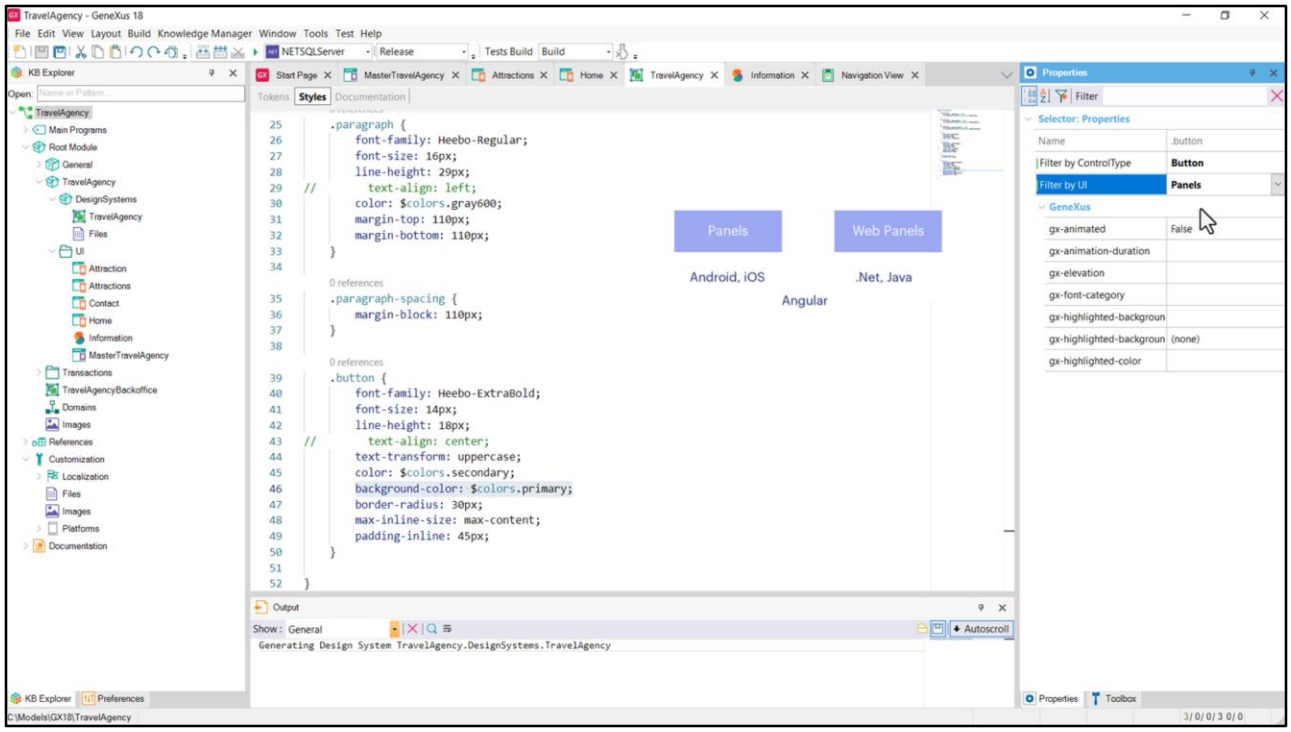
And now what I have to figure out is how to make the class associated with the button modify its background-color property to take the primary-highlighted token when the button is activated.

Note that we are talking about behavior to some extent, aren't we? When the button is activated we need to change one of the properties of the class that controls its style.



Well, it turns out that classes not only support the use of CSS properties but also GeneXus-specific properties.

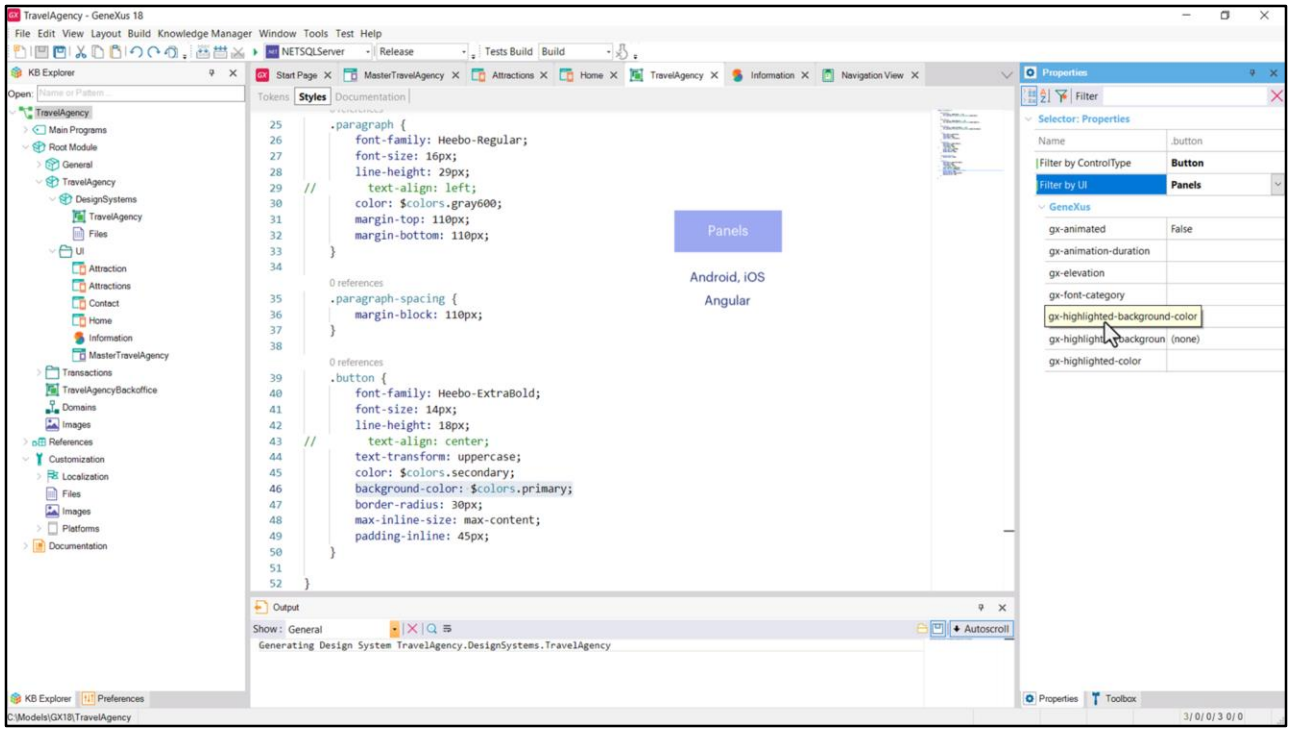
So, for example, if we come to the properties window... well, here we can filter the properties that we are using at the level of this class...



...but we also have these two filters to view not all the properties but, for example, those that correspond to controls of button type. By doing this it will show me only the valid properties for that type of control.

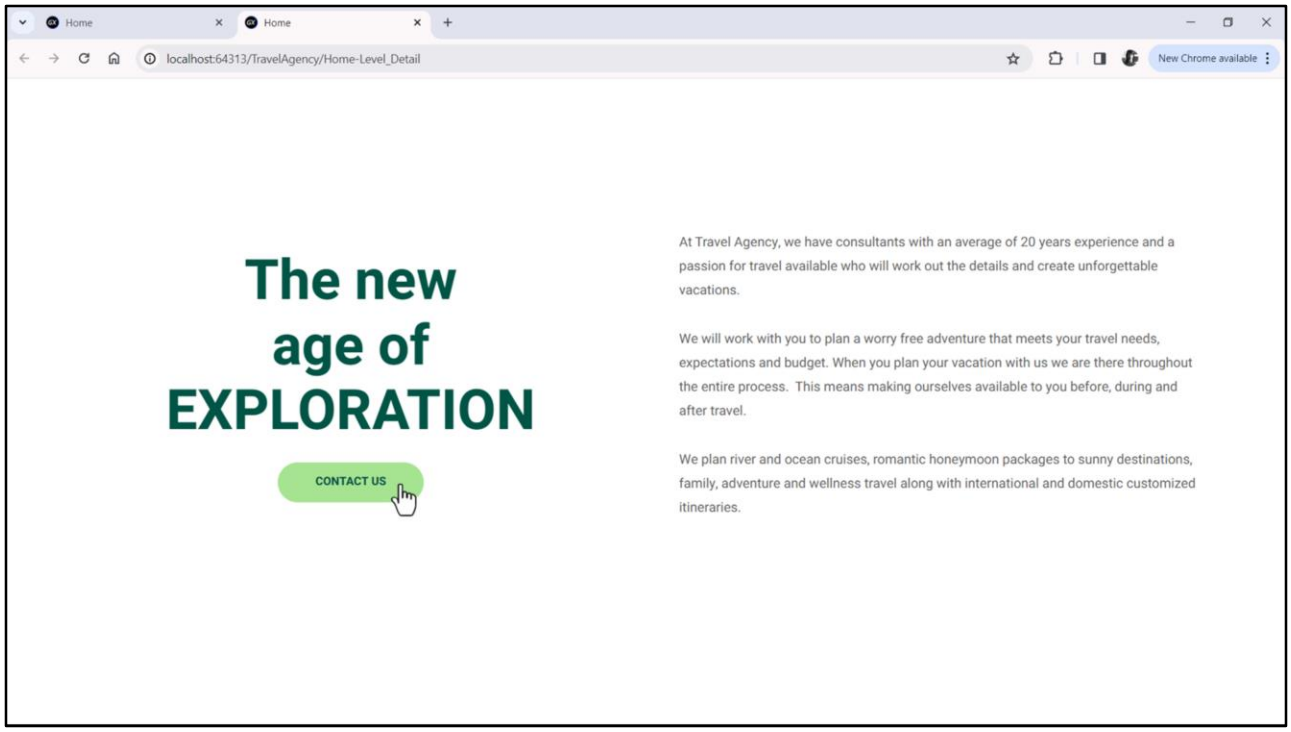
I can also filter by the type of user interface... that is to say, or all, which is what I'm filtering now... or I could choose the Panels world or the Web Panels world...

If I filter by the Panels world which is where we are now ... (remember that the Panels world emerged with smart devices, that is, with native applications for phones and tablets; then Angular joined this world... But Angular is an intermediate world, it is a hybrid between the Panels world and the Web Panels world and we are going to see that in a little while)...



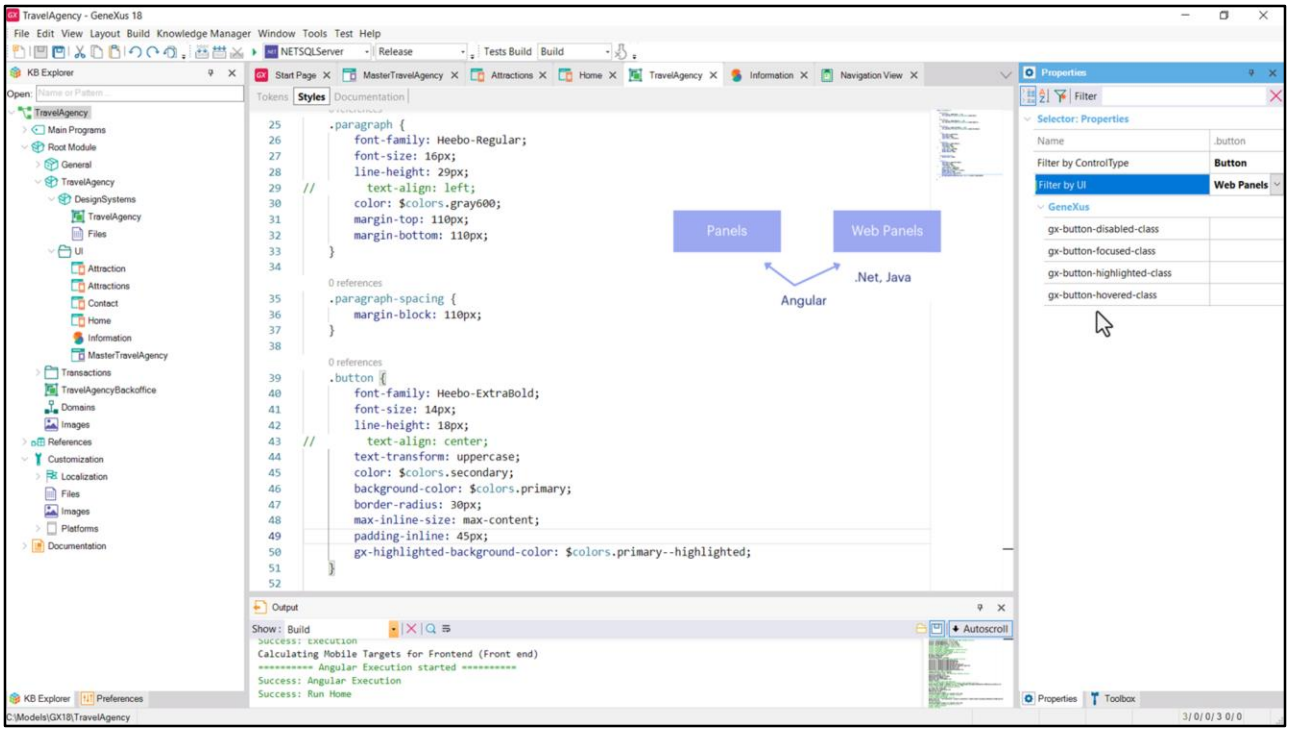
Now I'm filtering by the Panels user interface. And here we see that these GeneXus properties are appearing (gx, hyphen, and a property name)... these are the ones that I could eventually use inside the class, of a class with which I'm going to associate a control of button type in a panel... They are classes and each one of them has its own behavior.

Among the classes we see this one: gx-highlighted-background-color. This property is useful precisely for what we are looking for... I will set as color the color token primary--highlighted. Let's try it...



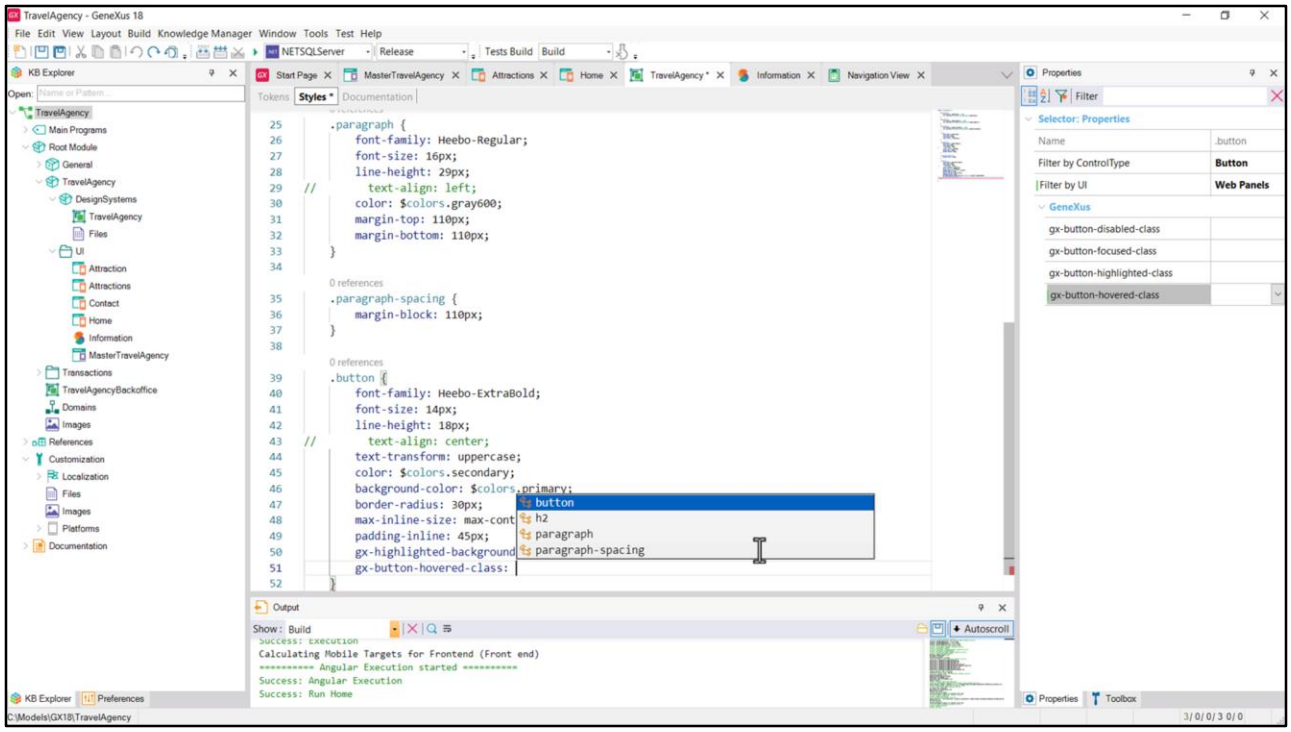
And here we see that when we click on it, the background color appears in another color. I'm clicking and holding the mouse; I haven’t released it because when I do so it's going to call the Contact panel that doesn't have anything programmed at the moment... that's why we see this.

And if we wanted this to happen on hover and not just on button activation?

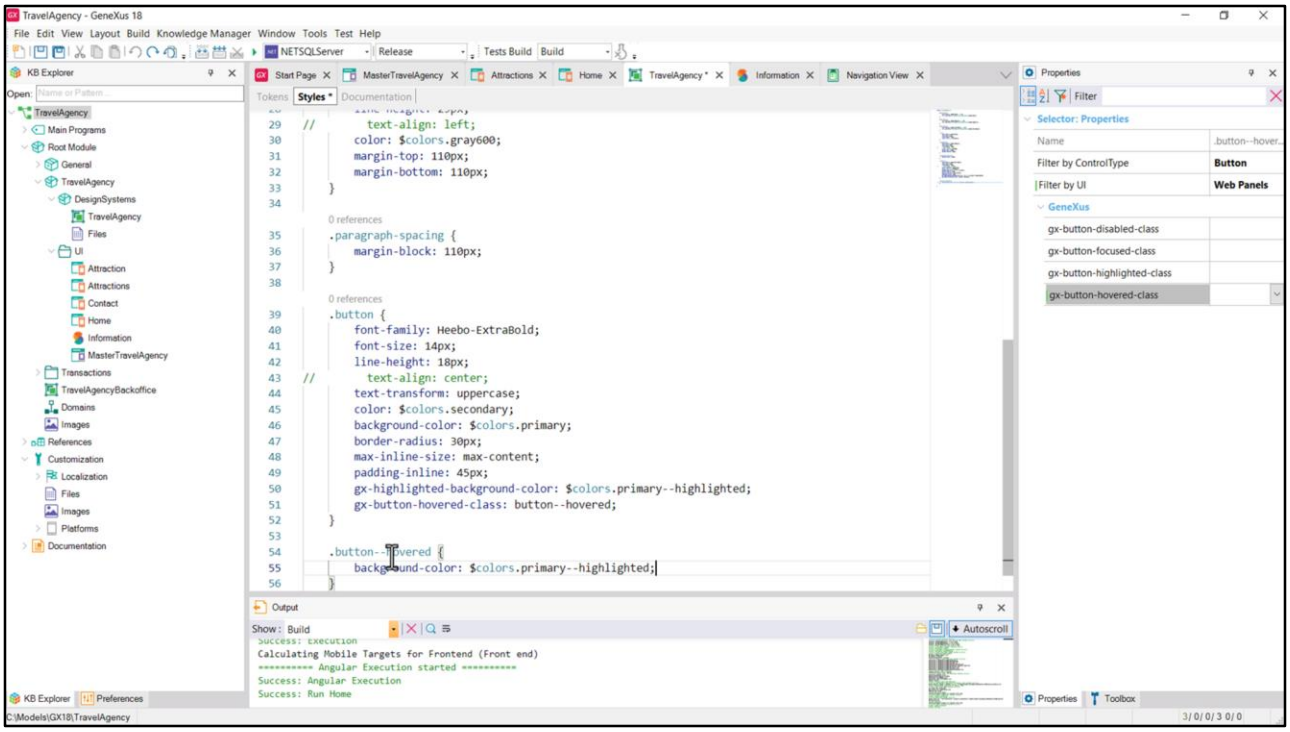


If we come to look for a property gx-hover-background-color, we won't find it. That is to say, the properties applicable to a button within a panel are these and do not include that one. It's because, I insist, these properties emerged with the panels world for native applications: Android, iOS.

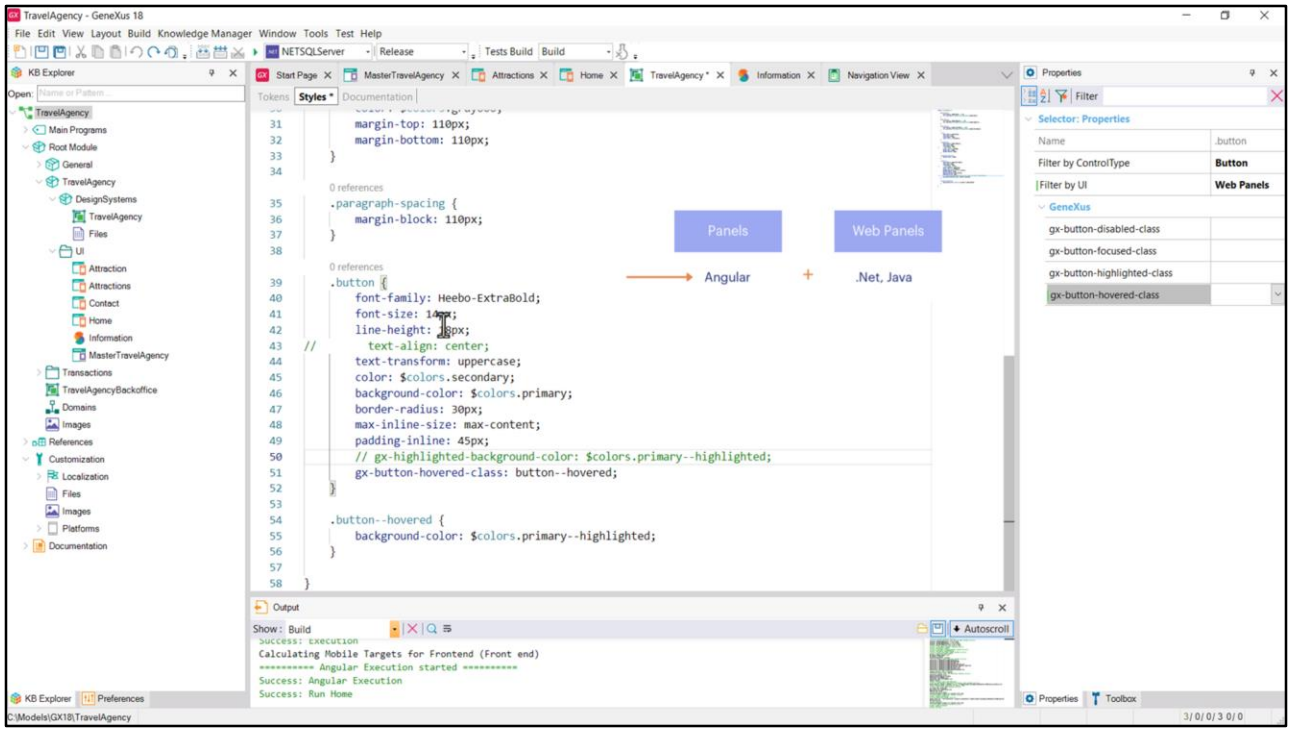
However, let's see what happens when I change the filter to that of Web Panels. These other properties associated with the buttons, which do include this hovered, appear. These properties... note that they end with class... what we are indicating if we choose this property is that...



...we are indicating which is the class that will control the style of the button that has this associated class when it is hovered over. Note that we are being offered the 4 classes that this DSO has specified so far.



We are going to create a new class called button--hovered, and in it... we are going to specify that the background-color is primary--highlighted.

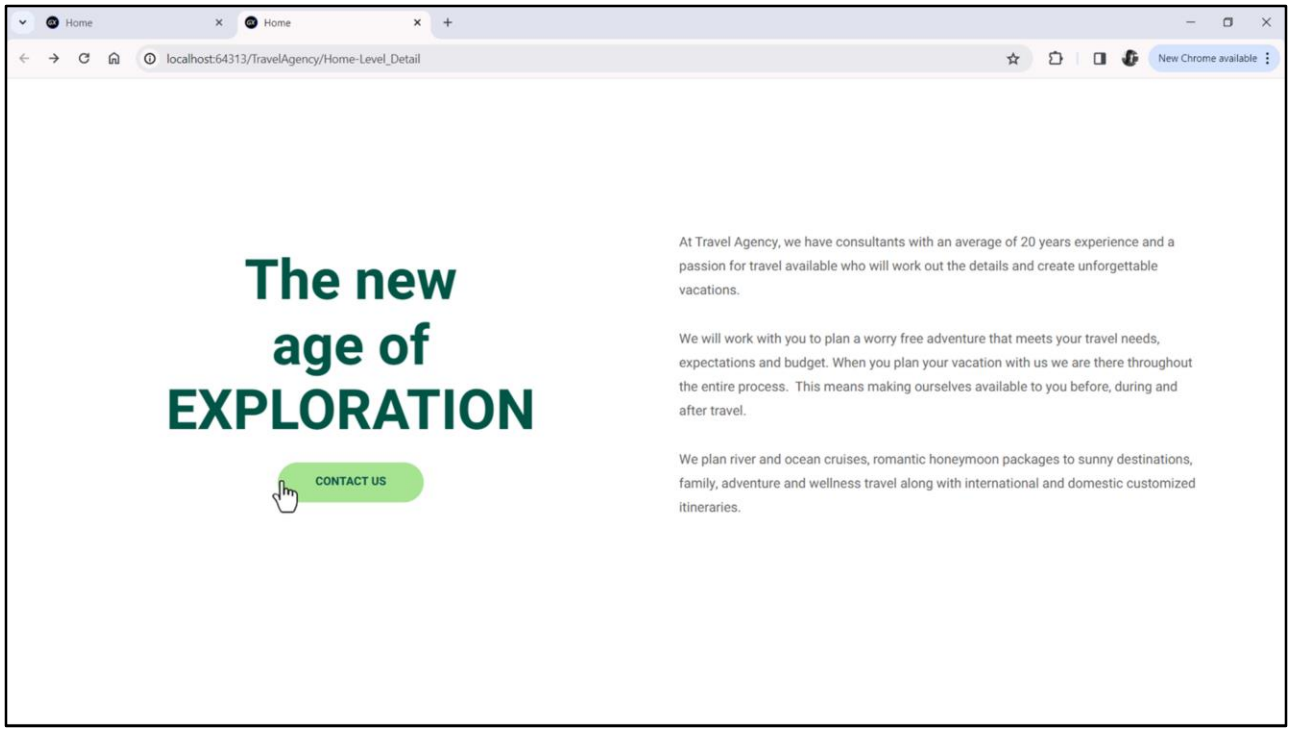


I will leave this one commented first so that the difference is clearly seen. I'm using a GeneXus property that at first we might think it will have no effect on our panel because we are in the world of Panels and not Web Panels. However, as these are properties that have to do with the web world, they will be taken into account.

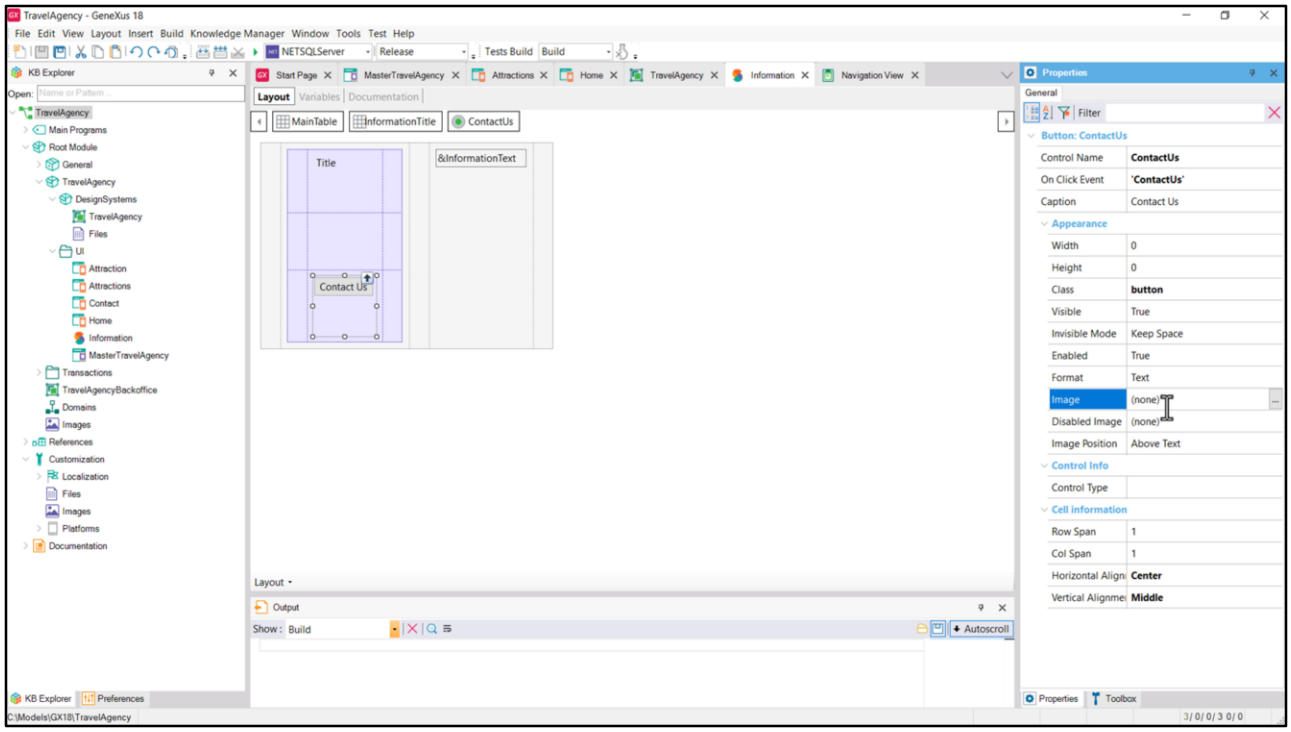
I mean, I'm going to be able to use it for a panel run for Angular, not for this panel run for Android or for iOS; for Angular I'm going to be able to use these properties as well.

So I'll summarize what I'm saying here: when the button that has this class associated with it is hovered over, these properties here corresponding to the button class will be added or overwritten by those that are in this other class, which we have selected here. And what we are indicating inside the class is to overwrite this property, the background-color. We could add new properties that do not overwrite these, but that simply add new style behaviors and well, they would be added there. In this case, as it is called the same as one that is already defined here, well, it is going to be overwritten.

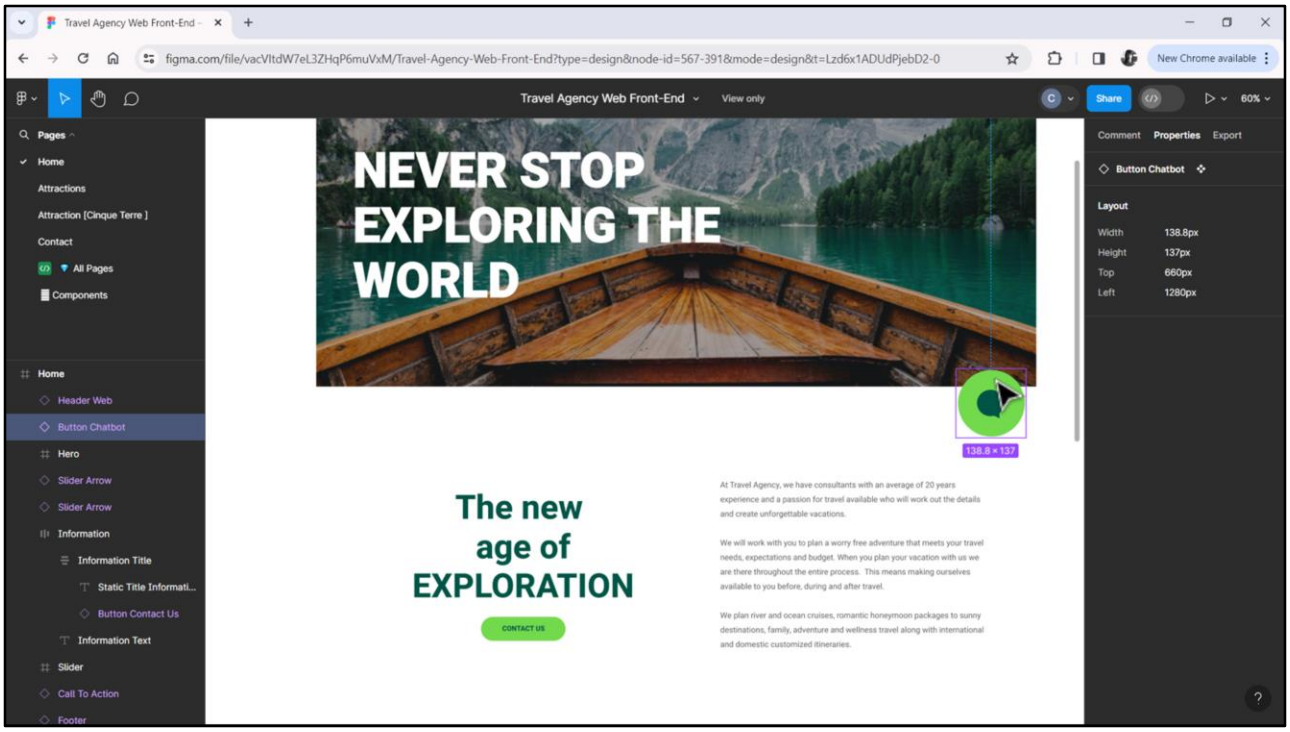
Let's give it a try.



OK, now let's see what happens when I hover the mouse over the button.

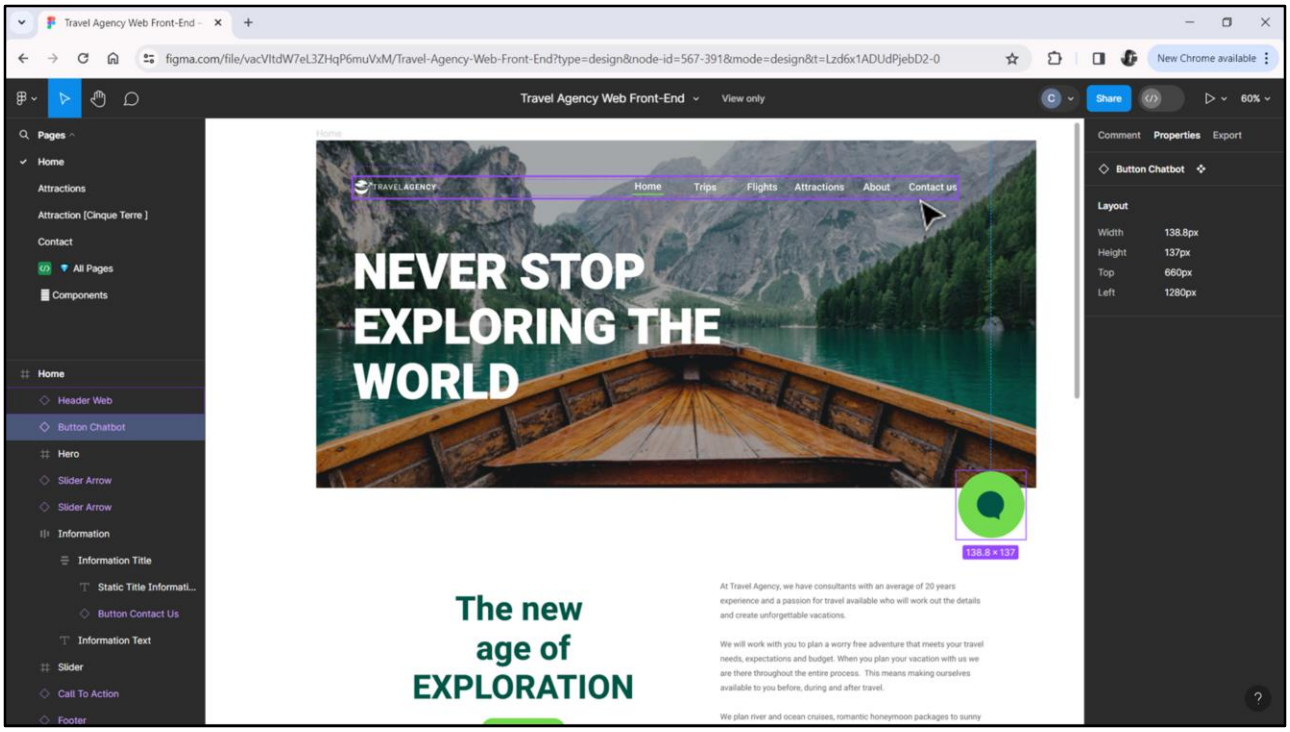


Finally, let's comment that a button can have a text, an image, or both.



A case in which we will have a button with image and without text will be the chatbot, which we will implement when we implement the Master Panel.

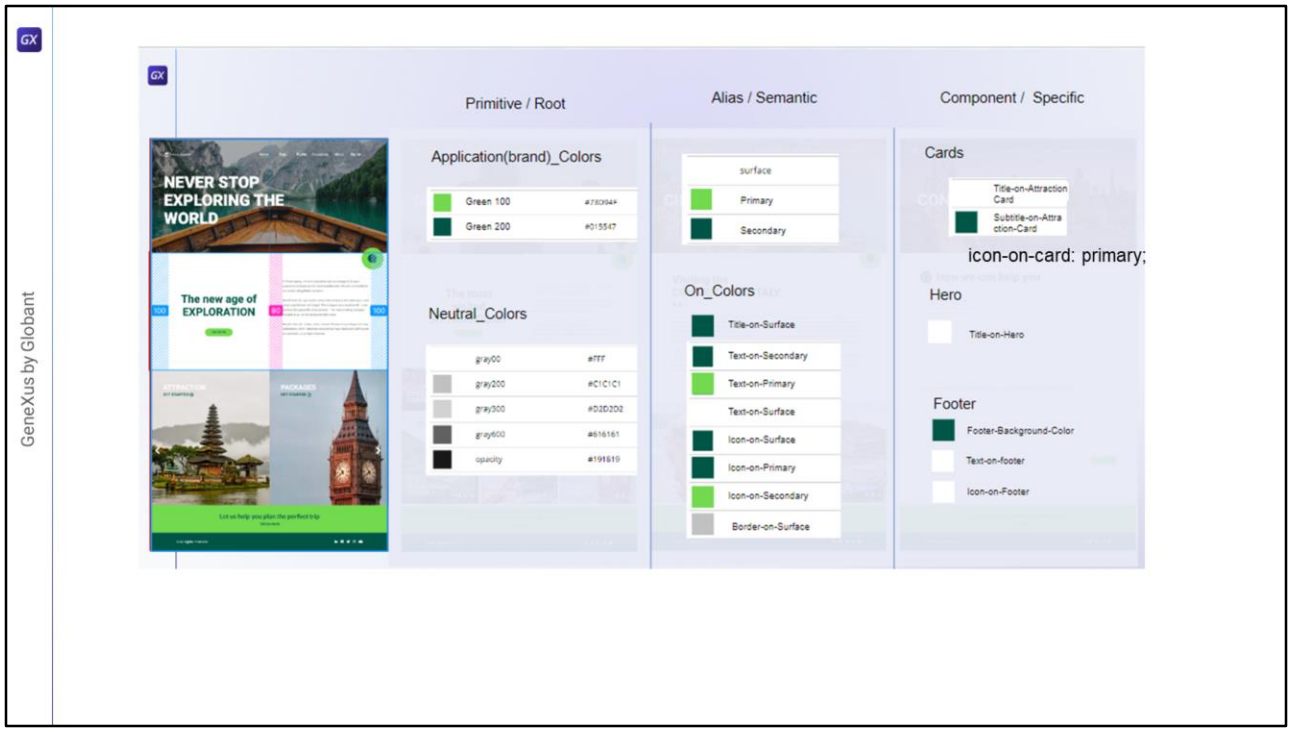
What looks like an image is actually going to be a button with an image. Why is it going to be a button? I'm telling you this in advance: because every clickable element, that is to say, that will respond to an action, must be implemented as a button.



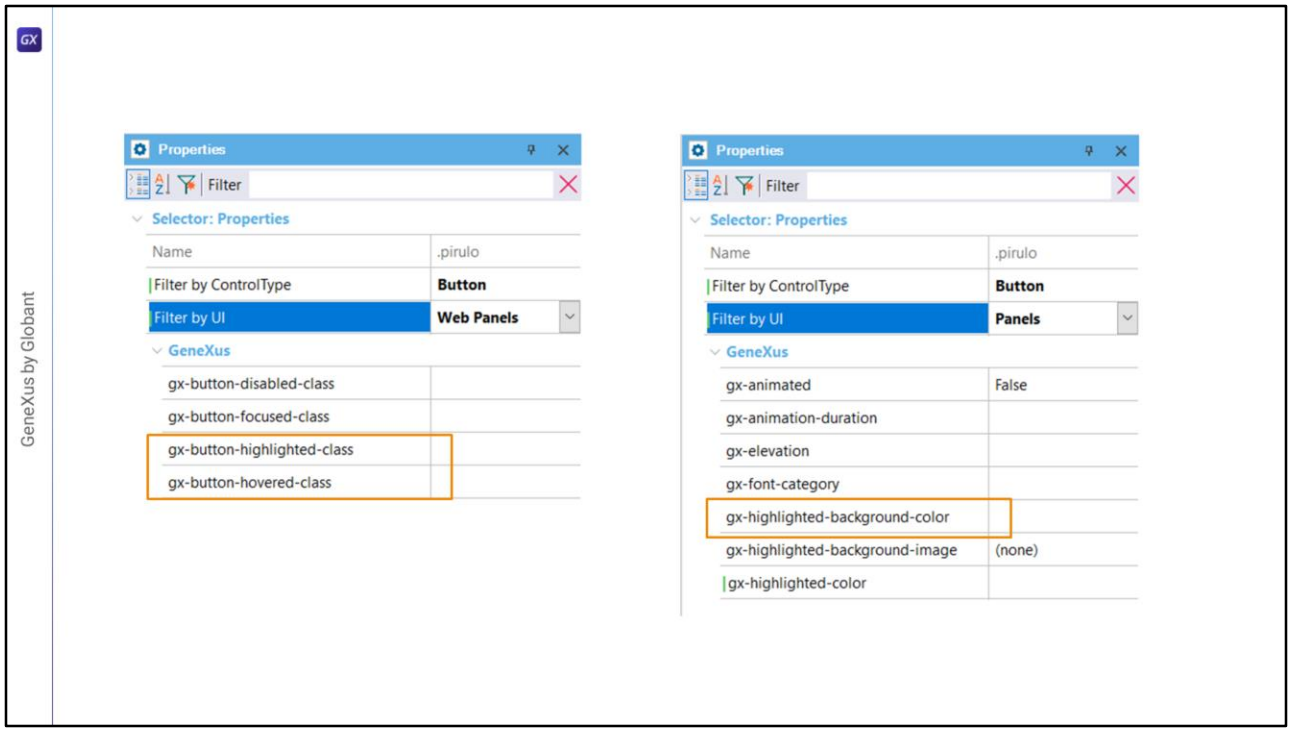
For example, menu items, which will be buttons that will have text and no background; i.e., the background will be transparent. Why? For accessibility reasons, as we will see in due time.

See you now in the next video.


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



Now let's see how to make the button look the way we want it to.



Now let's see how to make the button look the way we want it to.