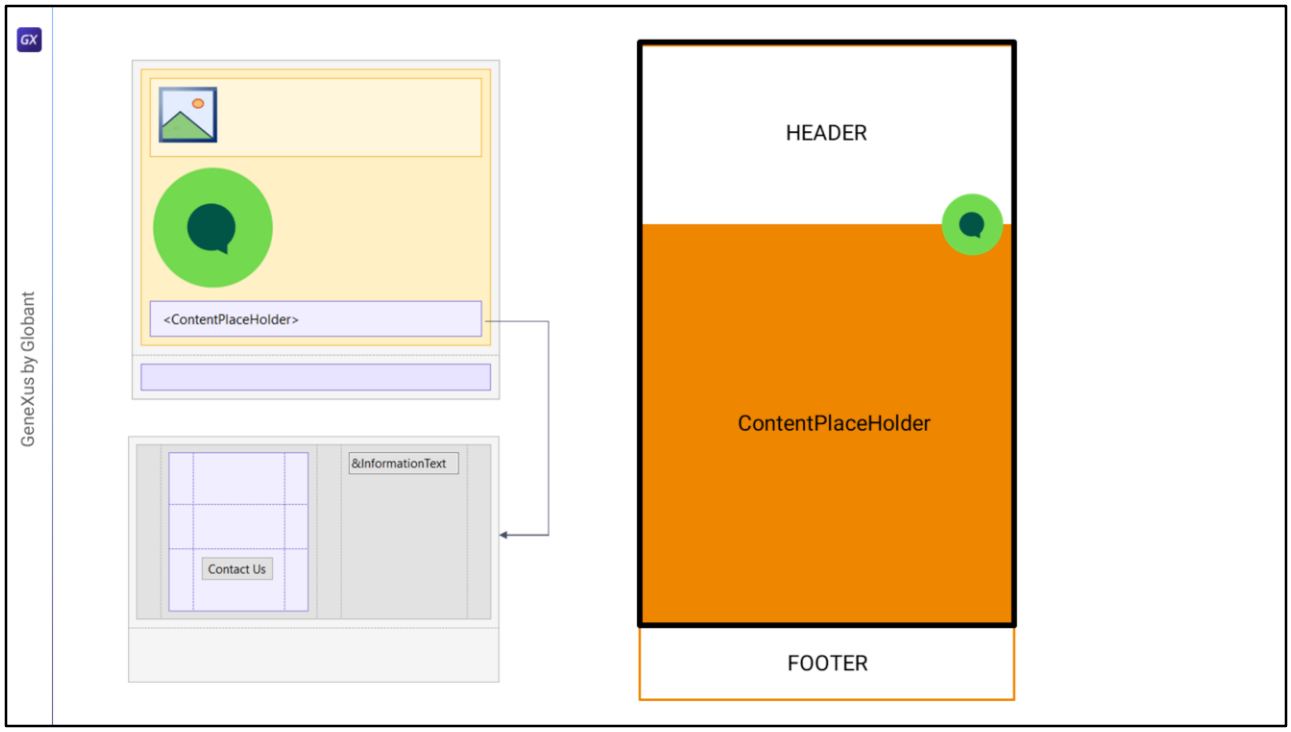
GeneXus by Globant

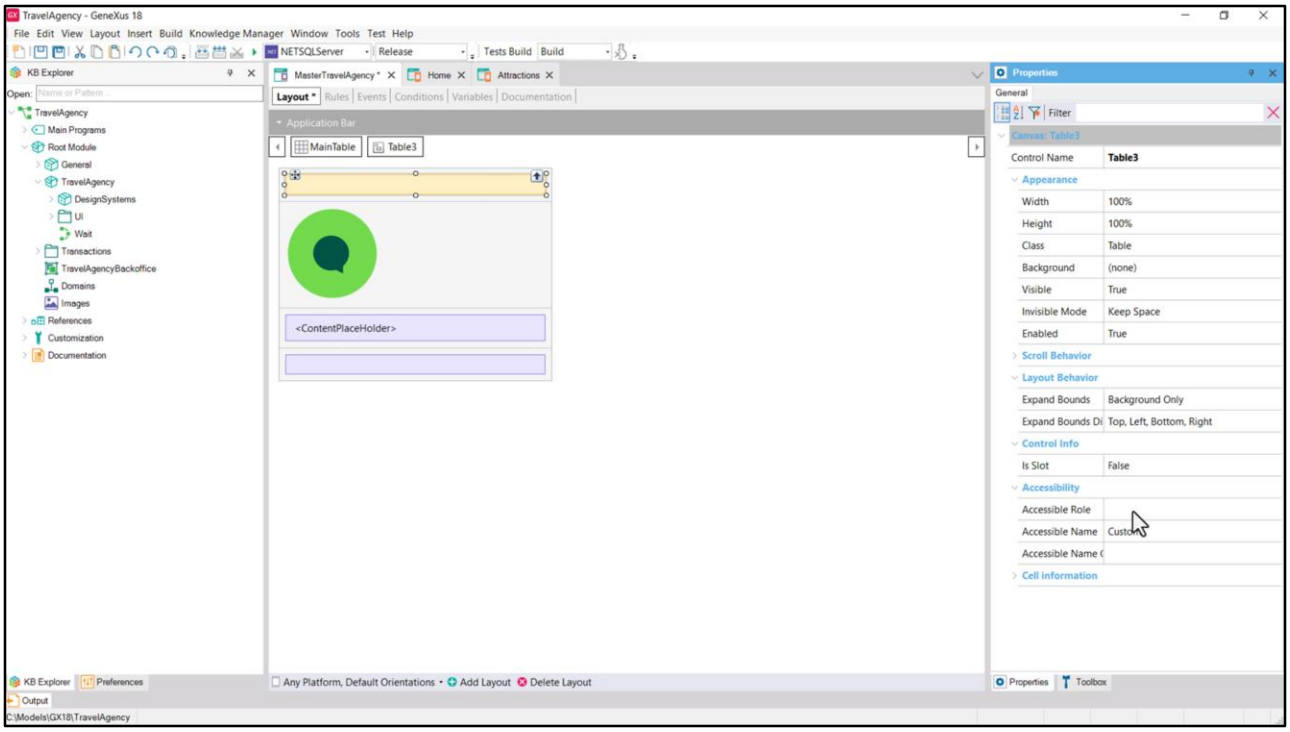
Header in GeneXus: background image and chatbot button

Cecilia Fernández

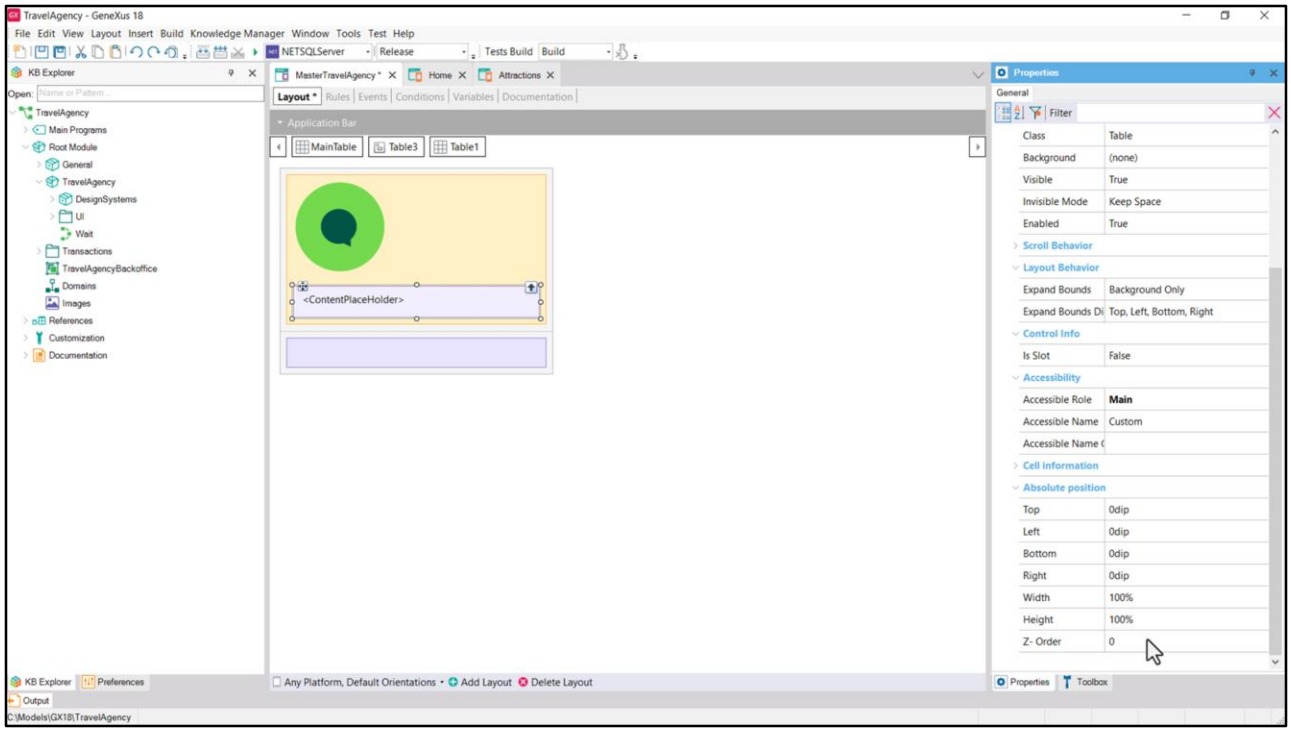


In the previous video, we analyzed at a conceptual level two possible implementations for the Master Panel.

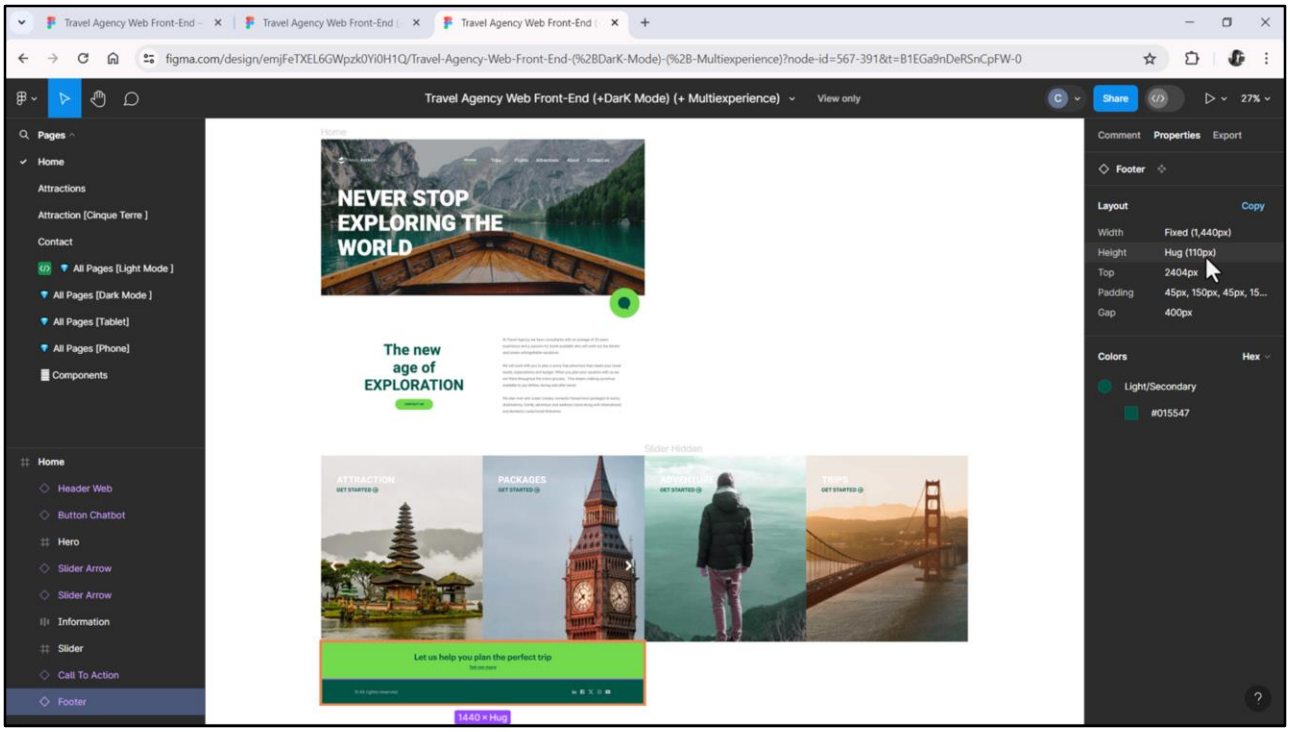
Let's implement the second solution, which seemed the most conceptually appropriate.



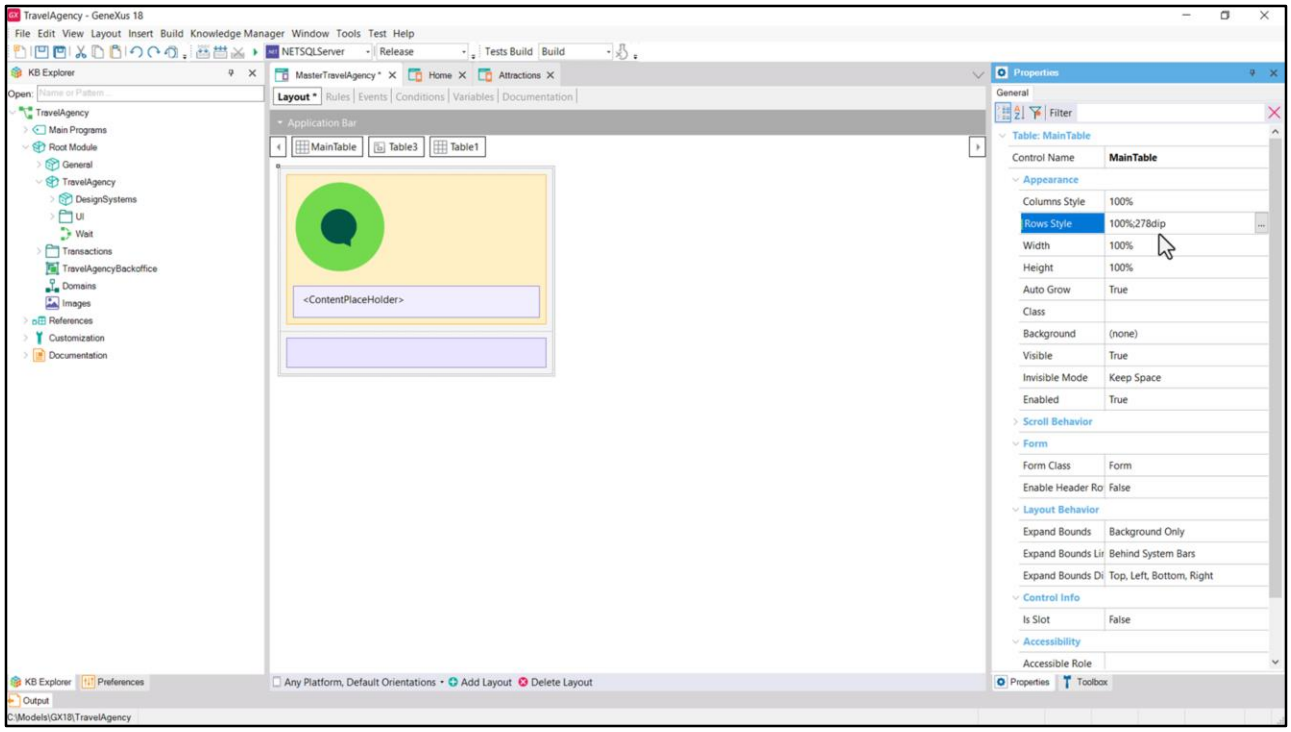
Here is the Main table. We will have to insert a canvas control in the first row; let's see its properties. To this one we are not going to set a Role because it will have to group the Header and the ContentPlaceHolder.



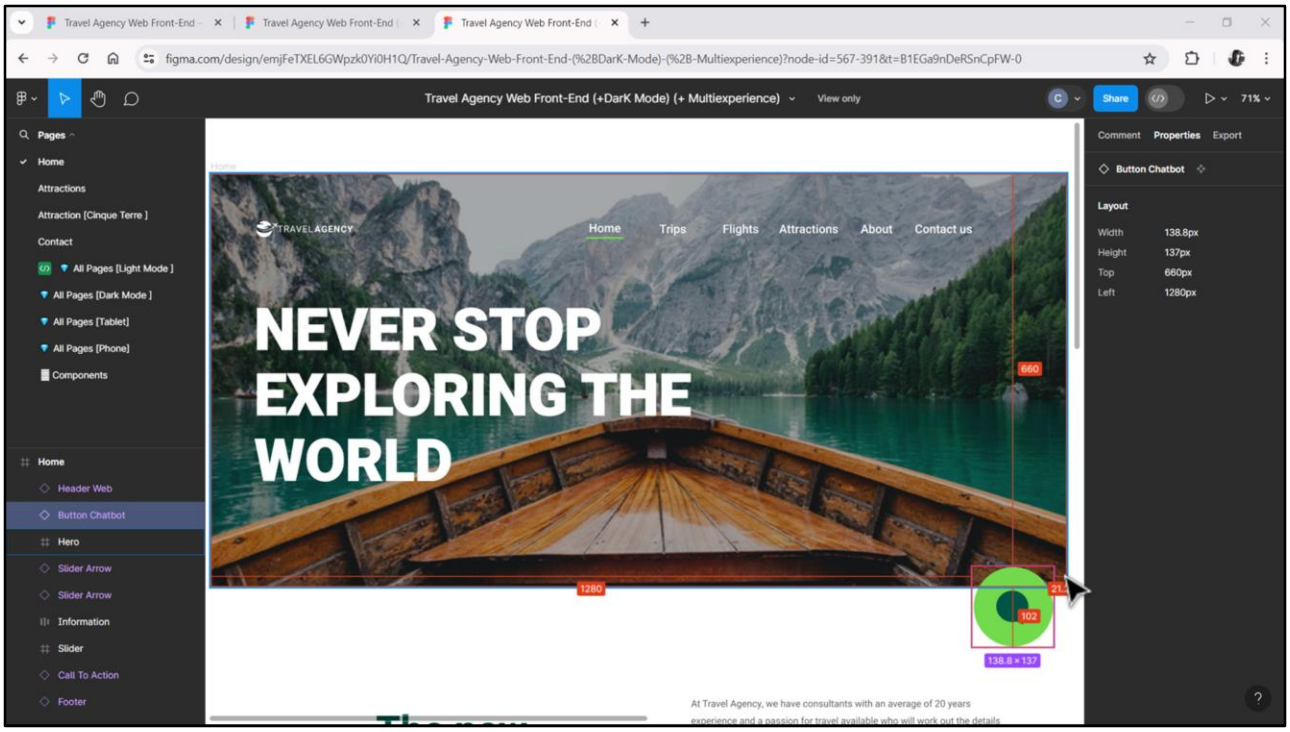
Let's drag the chatbot button into it, to see how the Absolute Position properties are now displayed...
And to the table with the contentplaceholder, and we see that the Absolute Position properties also appear...



Now the Main table has two rows. That of the Footer must have a height of 168 + 110, that is, 278 dips.

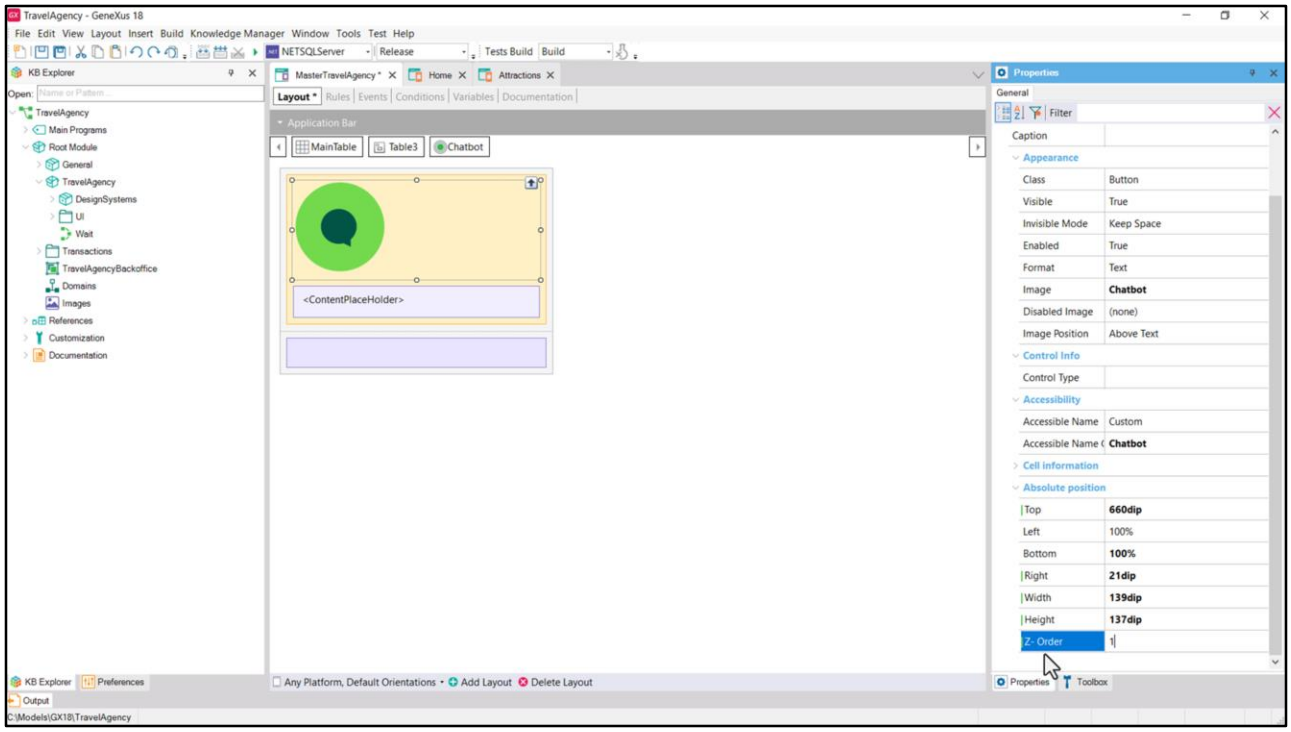


In the Main table we set Rows Style to: 100% for the first row, and 278 dips for the second one.



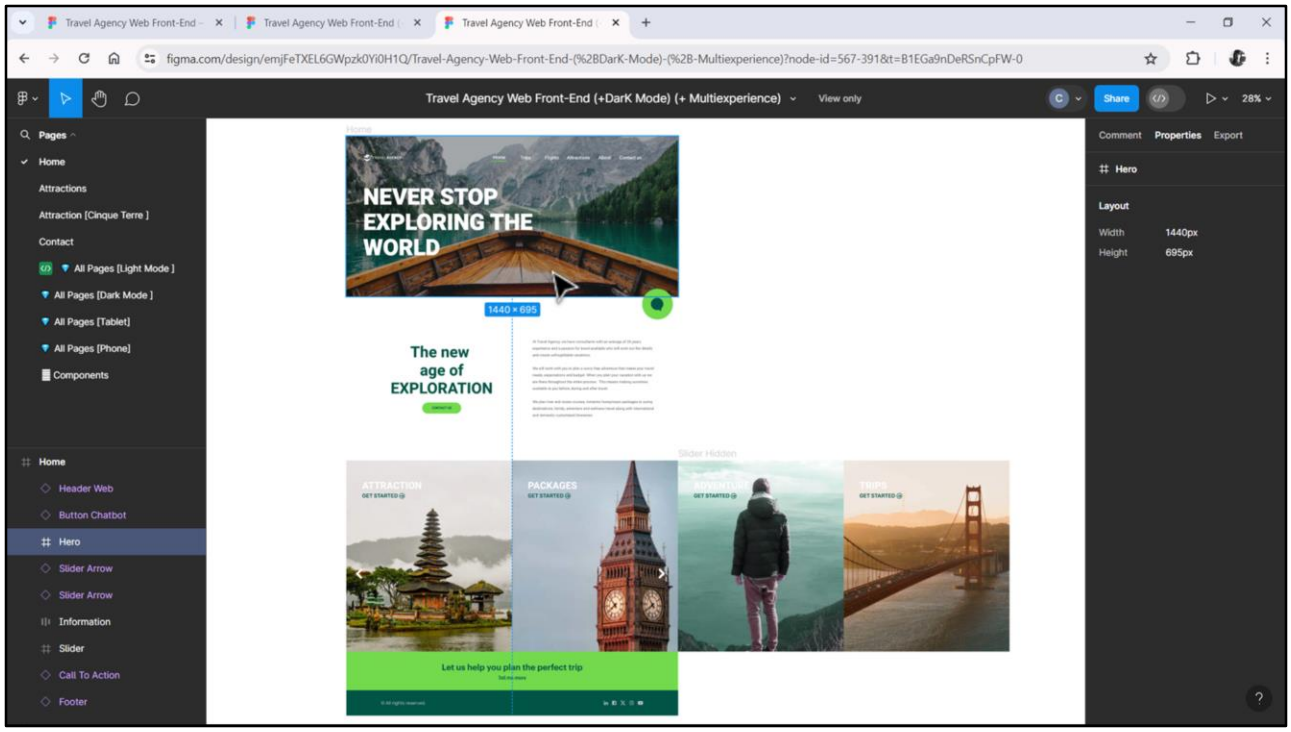
Let's give absolute positions to the two controls we have at the moment.

The chatbot button's width was 139, and it was 21 from the right...
And its height was 137 and it was 660 from the top. So…

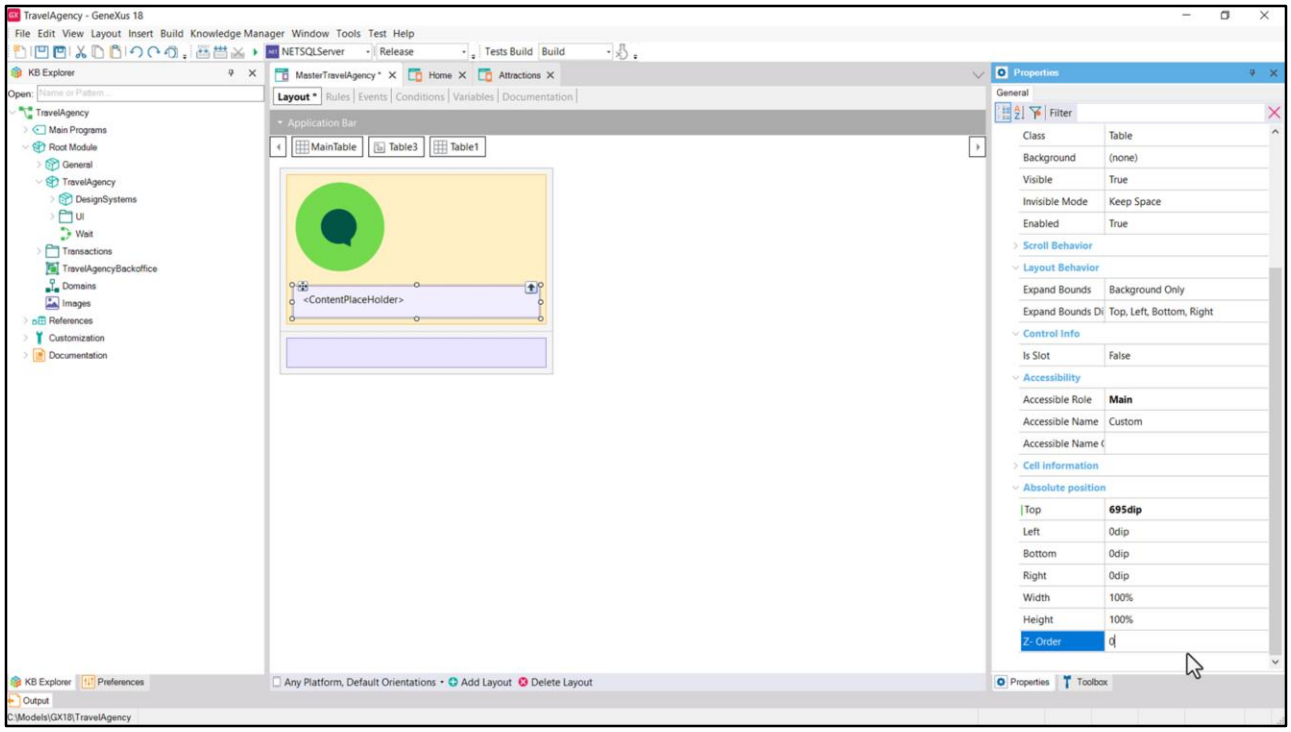


Note that if we set Right to 21 and Width to 139, Left is automatically 100%.
And likewise if we set Top to 660 and Height to 137, Bottom is 100%.

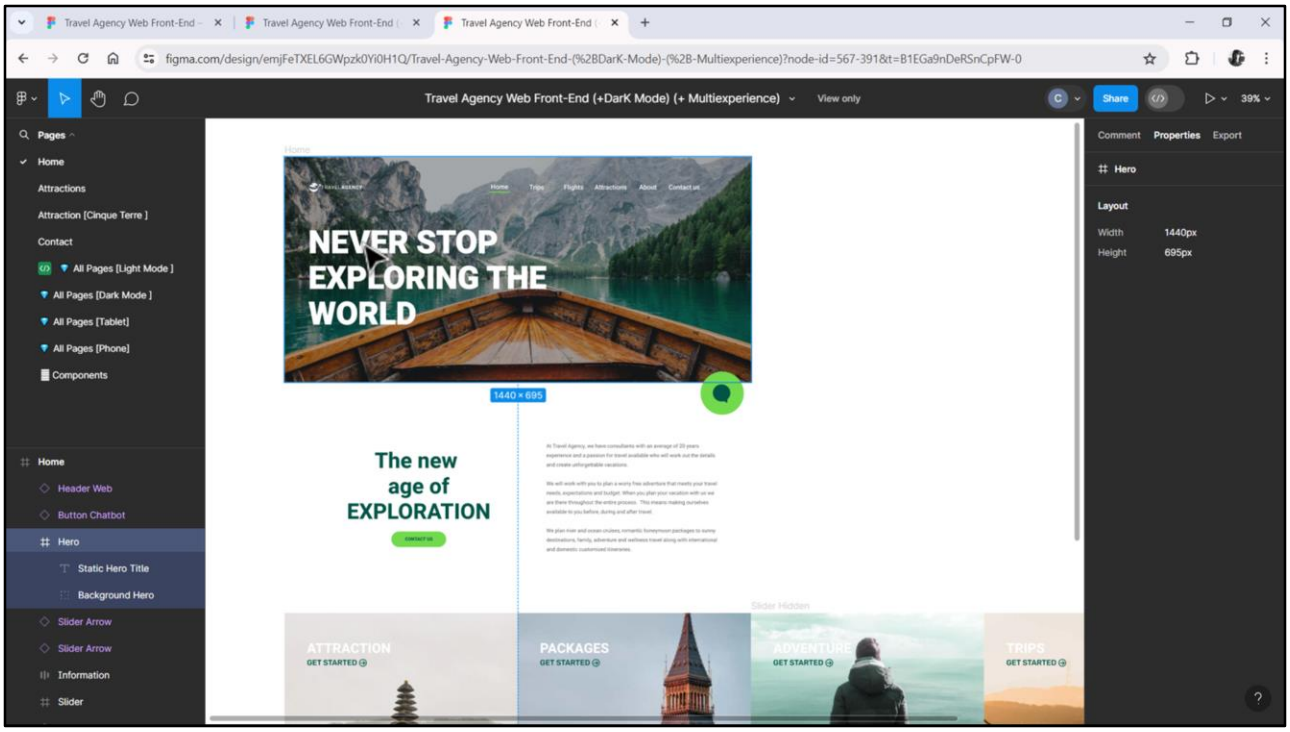
Since this button has to be in the top layer, we will enter 1 for the Z-order, because the contentplaceholder and the Header (which we haven't inserted yet) can be in the same layer 0, the bottom one.



Now we must specify the absolute positioning of the table with the contentplaceholder. We know that it must start here, that is to say, at 695 dips from the Top of the canvas... and that it must be next to the rest of the edges of the canvas.

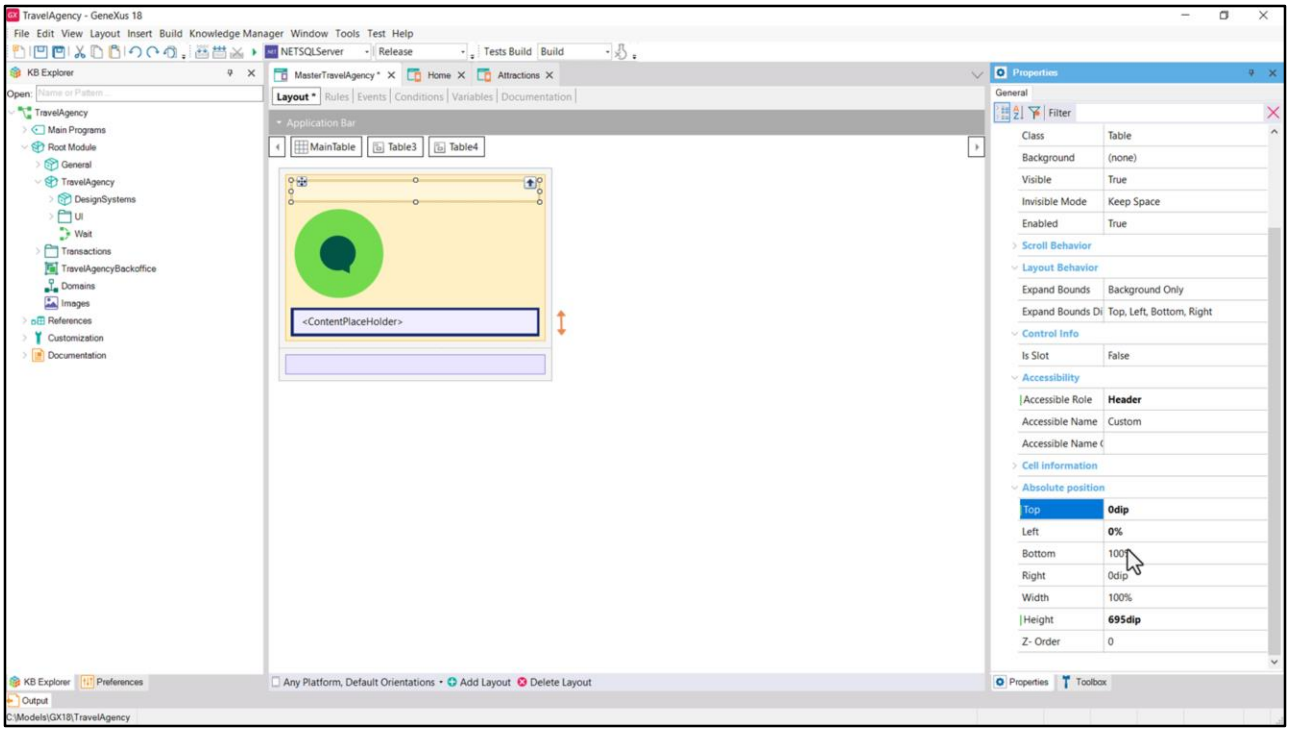


That is: Top 695, and the rest as is: 0 left, 100% width and 0 right. 100% height and 0 Bottom. And Z-order also 0.



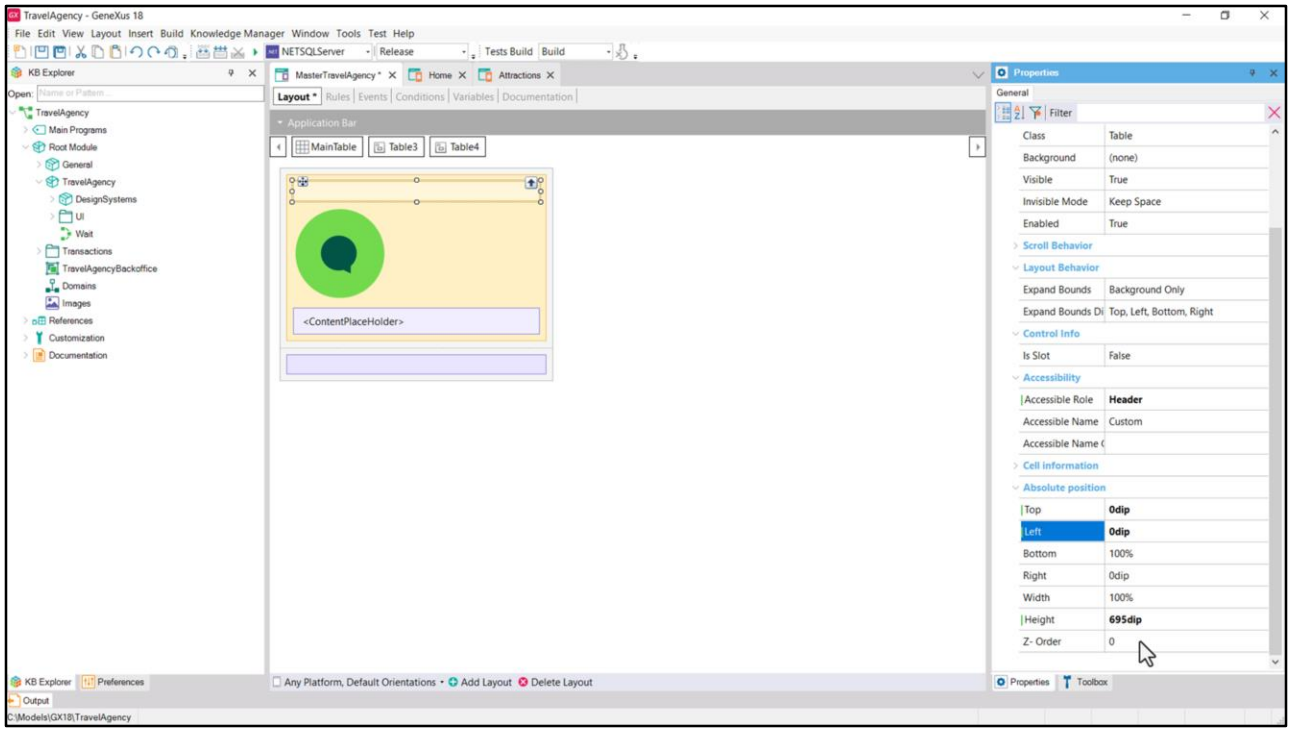
Now we need to implement the Header itself, which will contain the background image, the menu, the logo, and this text.

We will need another canvas to overlay all this. It can't be the same one that contains the button and the table with the contentplaceholder because we need to be able to assign it Header Role for accessibility. So we need a separate container.

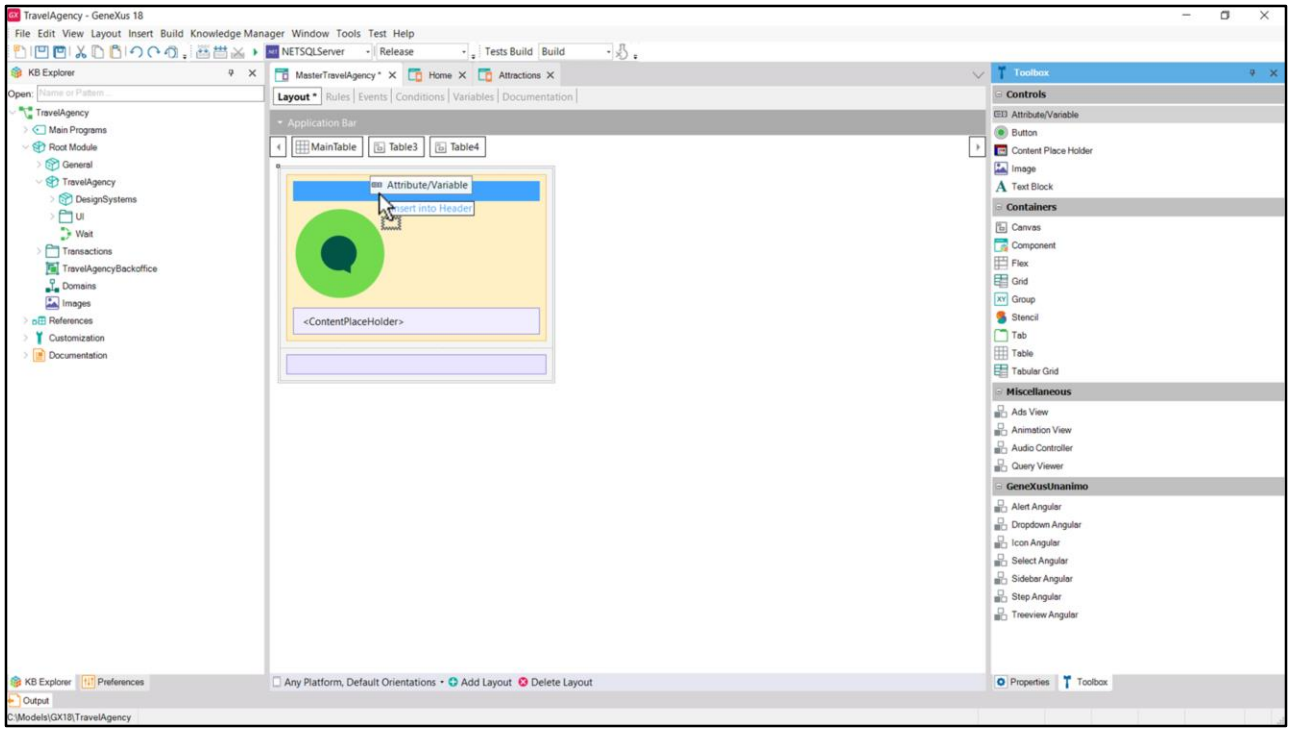


We insert, then, another Canvas, which we will call Header and in Accessible Role we will set the value Header.

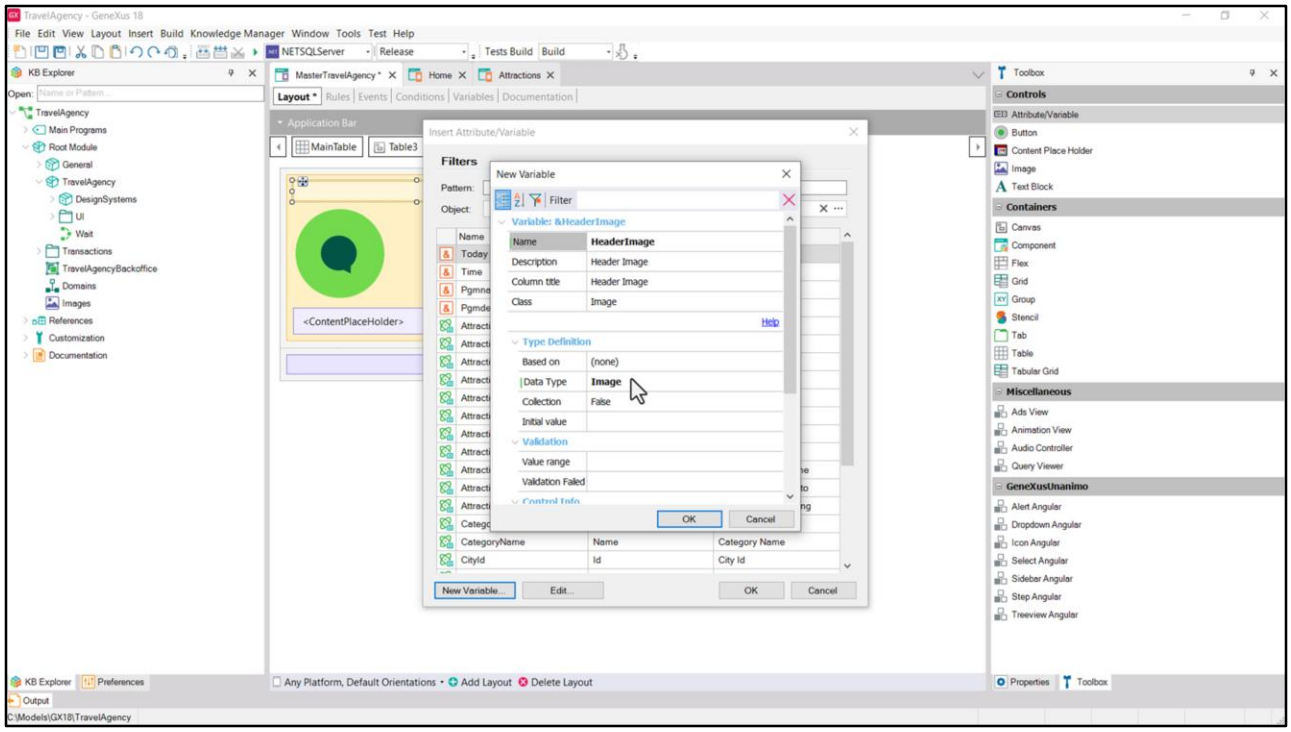
And what will its absolute positioning be in relation to the outer canvas?
First of all, its height will be 695 dips, which was the height of the background image. We want it to be next to the Top, so 0 dips from the top, which leaves the remaining 100% as Bottom (which will correspond to the height of the contentplaceholder).



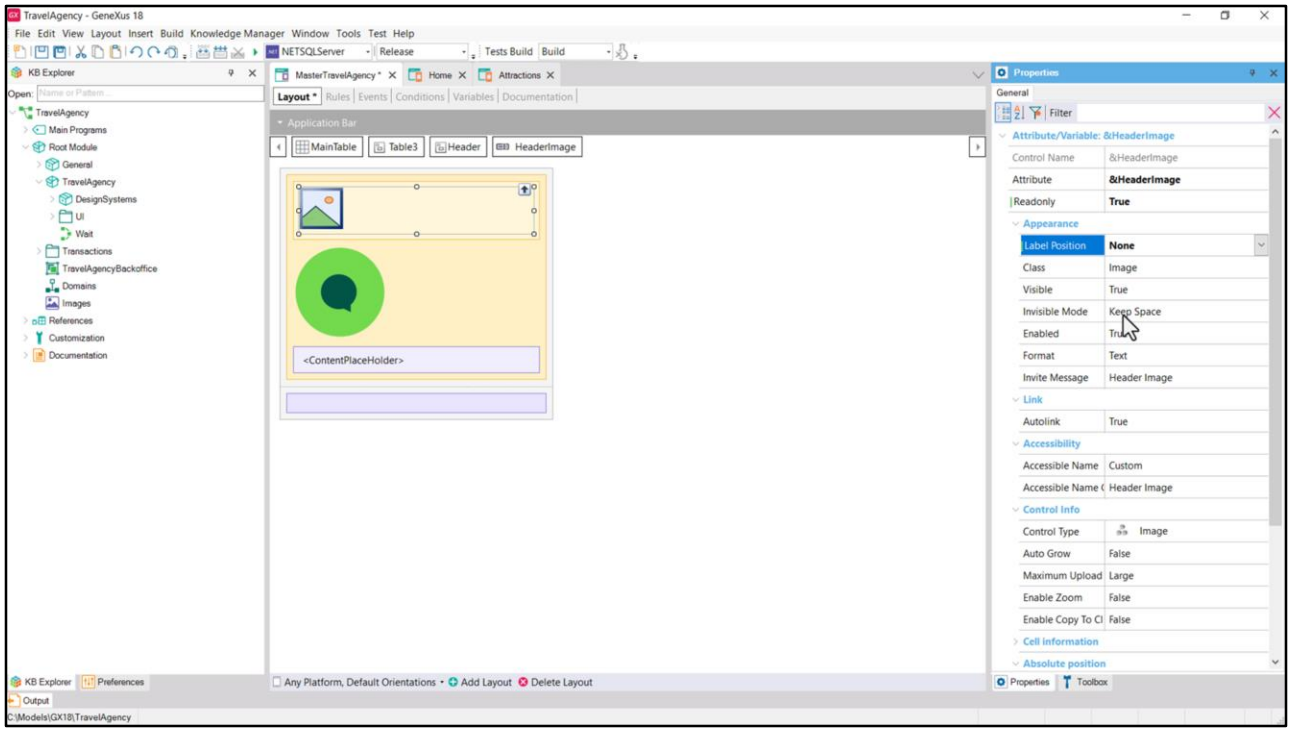
And left and right will also be at 0 dips, that is, next to it, since its width will be 100%. The Z-order property, as we have already analyzed, will be set to 0.



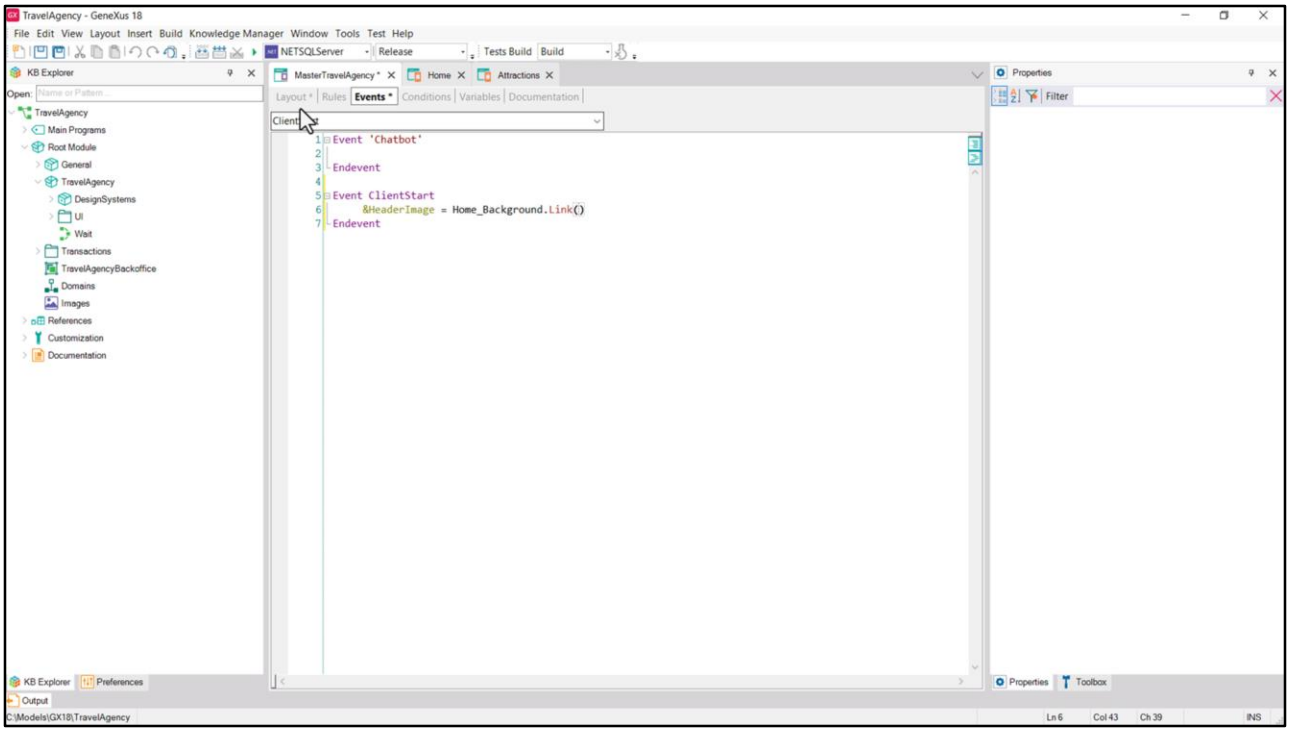
Now I have to implement the background image, the one we usually call Hero. I have two options: either I use an Image control or I use a variable control containing the image. I will use the second alternative, since this image will vary depending on the panel that is being loaded in the Contentplaceholder in each case.



I'm going to call it HeaderImage and it's going to be of Image data type.

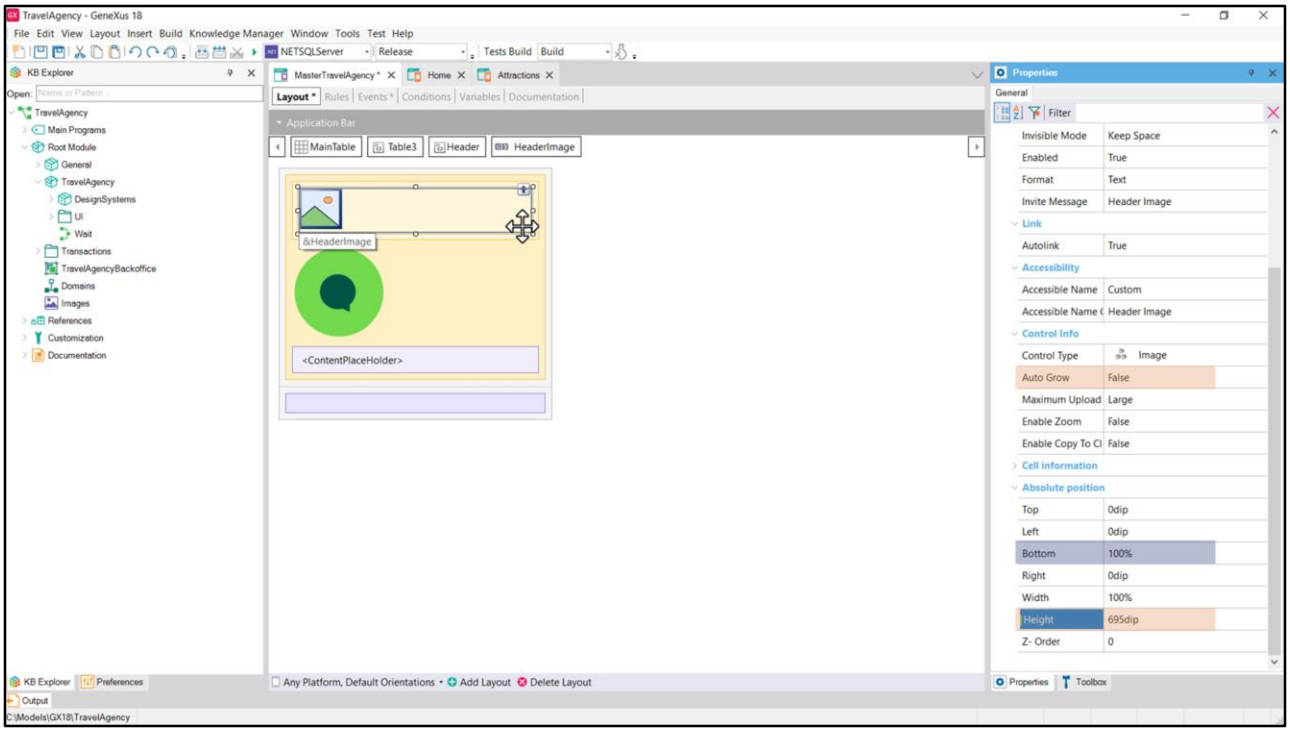


I will want it to be readonly and its label to be hidden.



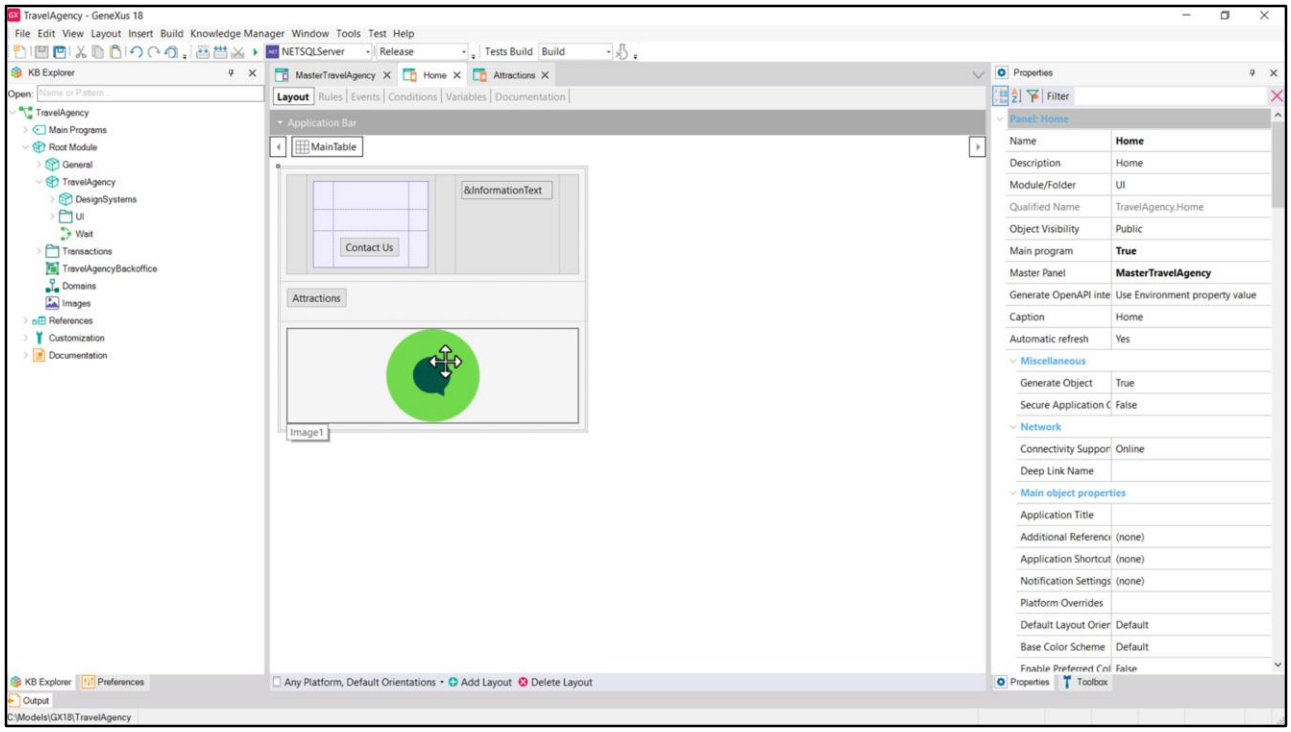
I will load it provisionally in the ClientStart event, starting (ctrl-o) from the image object that we had already inserted in the KB in the preparation stage. And to which we had called in this way... I use the Link method.

Later we will have to vary this assignment depending on what is being loaded, but we will see that later. Now we are going to leave it fixed.

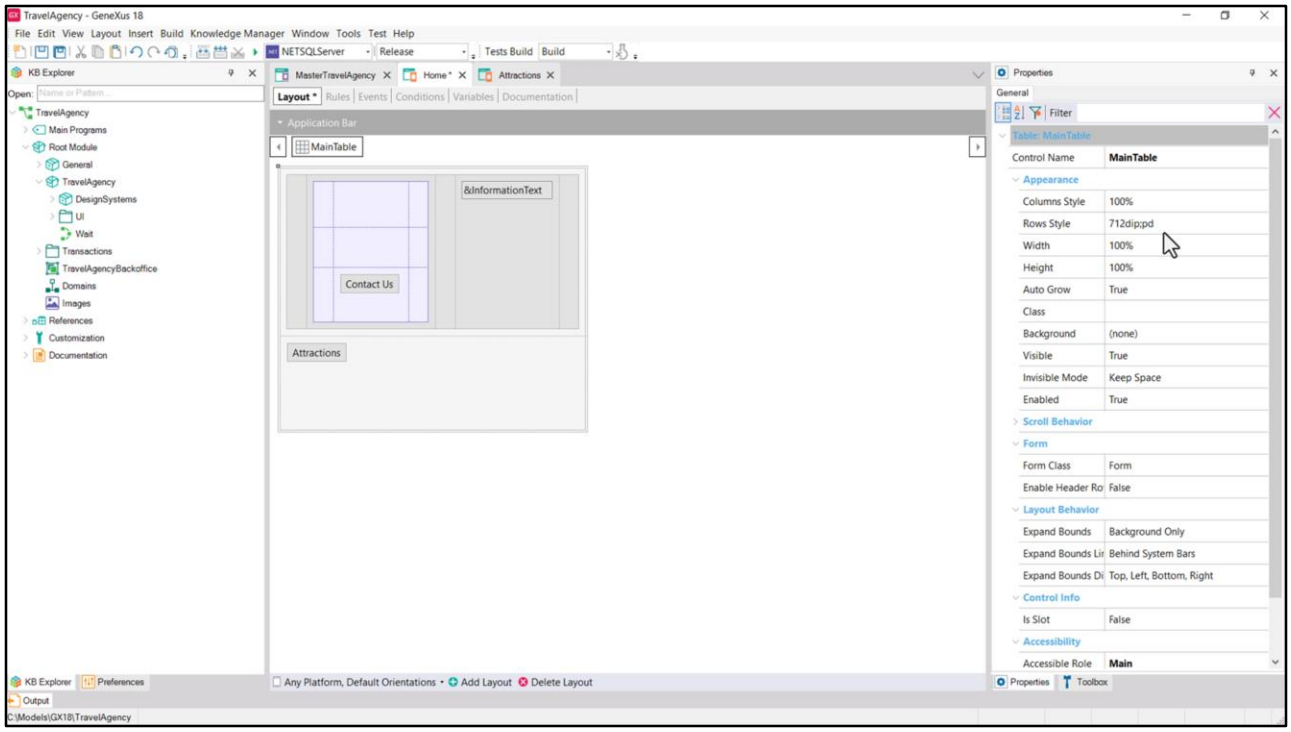


Next, we will indicate the absolute positioning of the variable relative to the Canvas Header. We will say, then, that its height will be 695 dips. And we will leave Auto Grow set to False. We leave the other properties as they are, because we want the image to be next to the top and side edges. The Canvas, as we said in the previous video, internally has Auto Grow set to true, so any control that overflows will make it expand downwards. But in this case we only have the image, set to 695 dips, so this bottom of 100%, when calculated on page load, will be 0 dips. That is, the image should reach the edge of the canvas on all four sides.

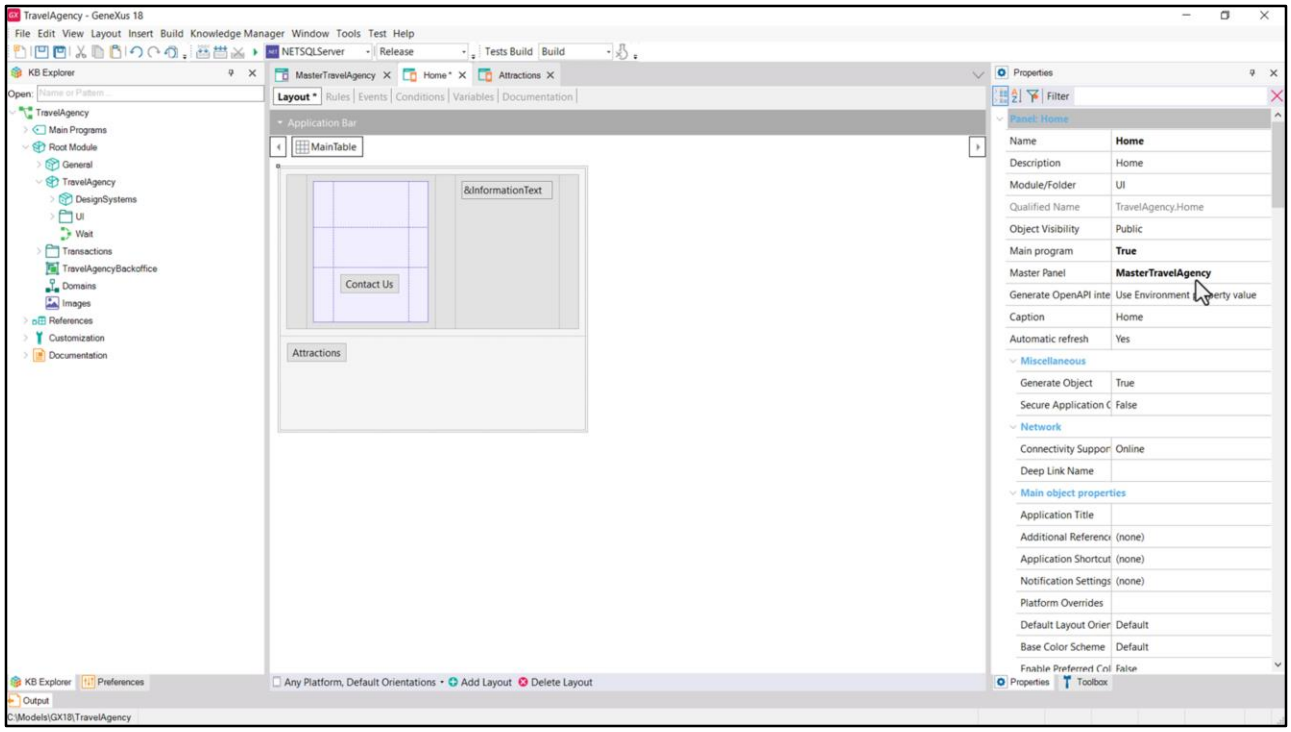
Let's save the Master Panel.



In the Home Panel we had placed this image to show something I don't remember. We remove it, and leave the button to call Attractions for now.

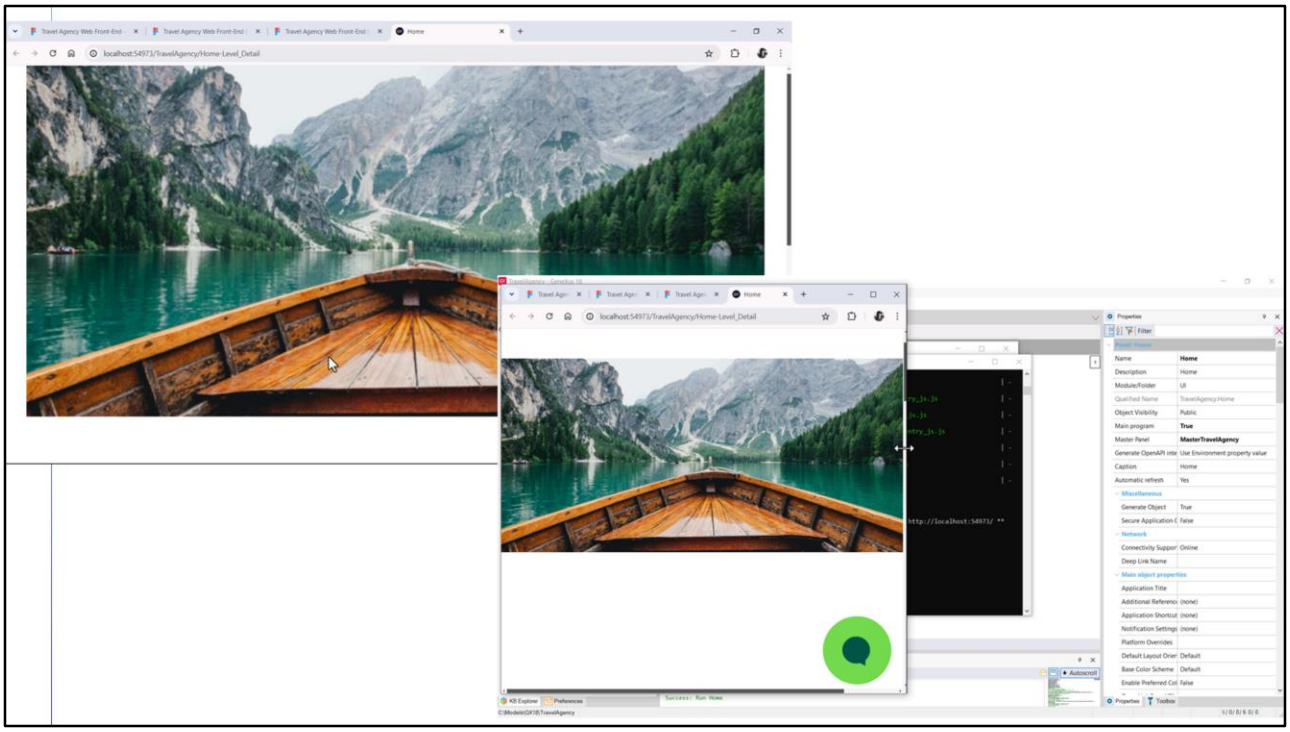


And Rows Style is left as we had it before. With this solution we don't need to make any changes here.

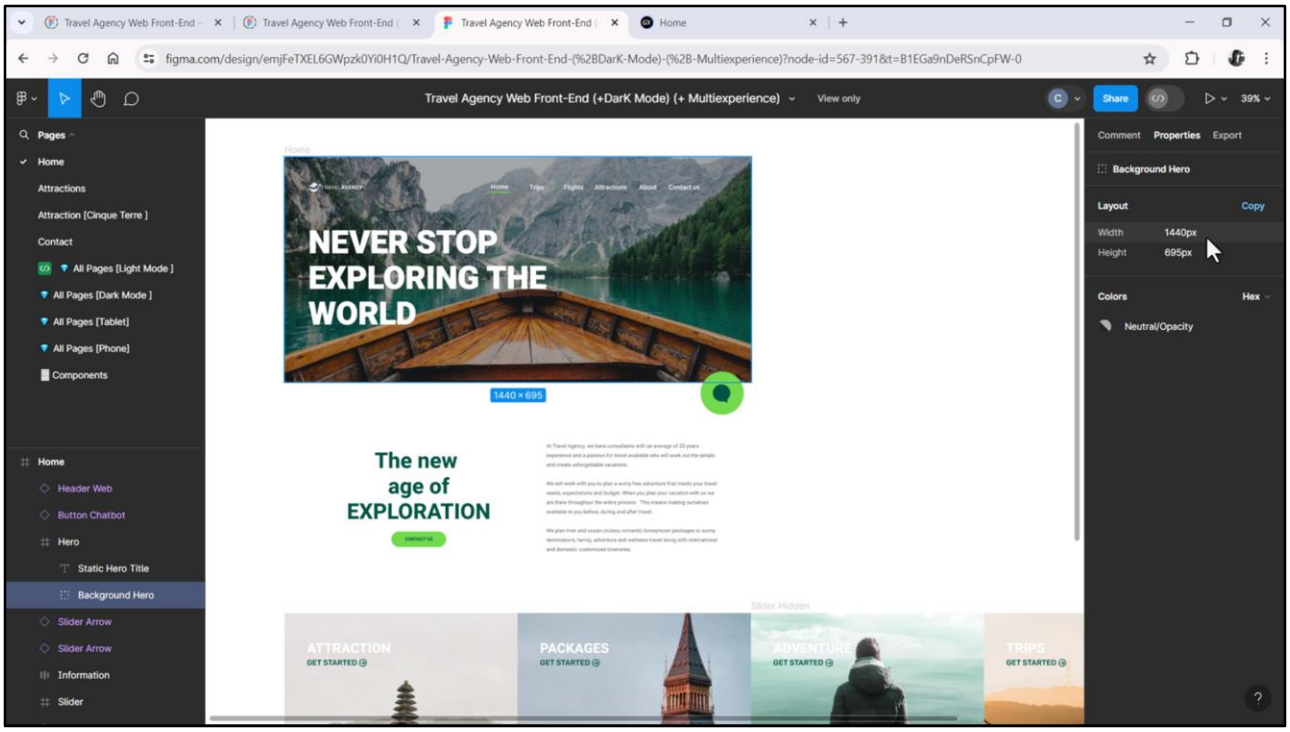


We see that the Master Panel of Home is the one we have been working with.

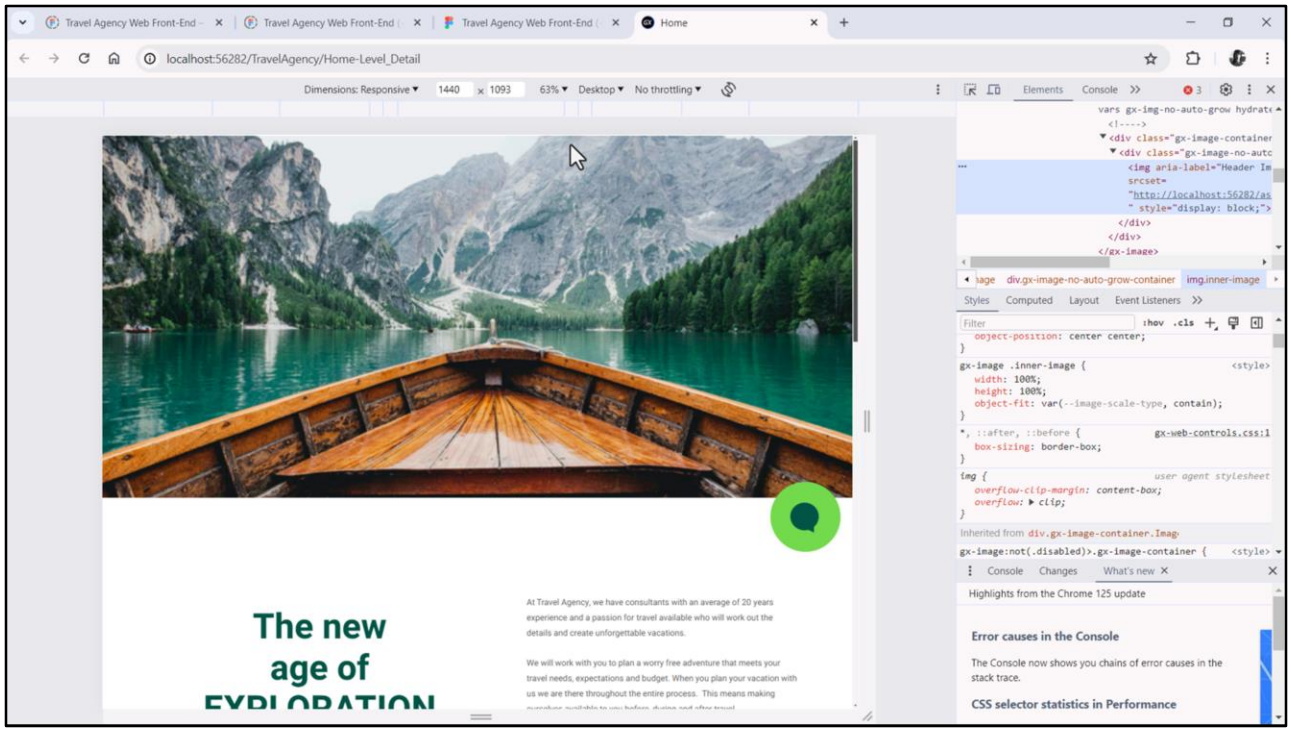
Let's save and run this Home.



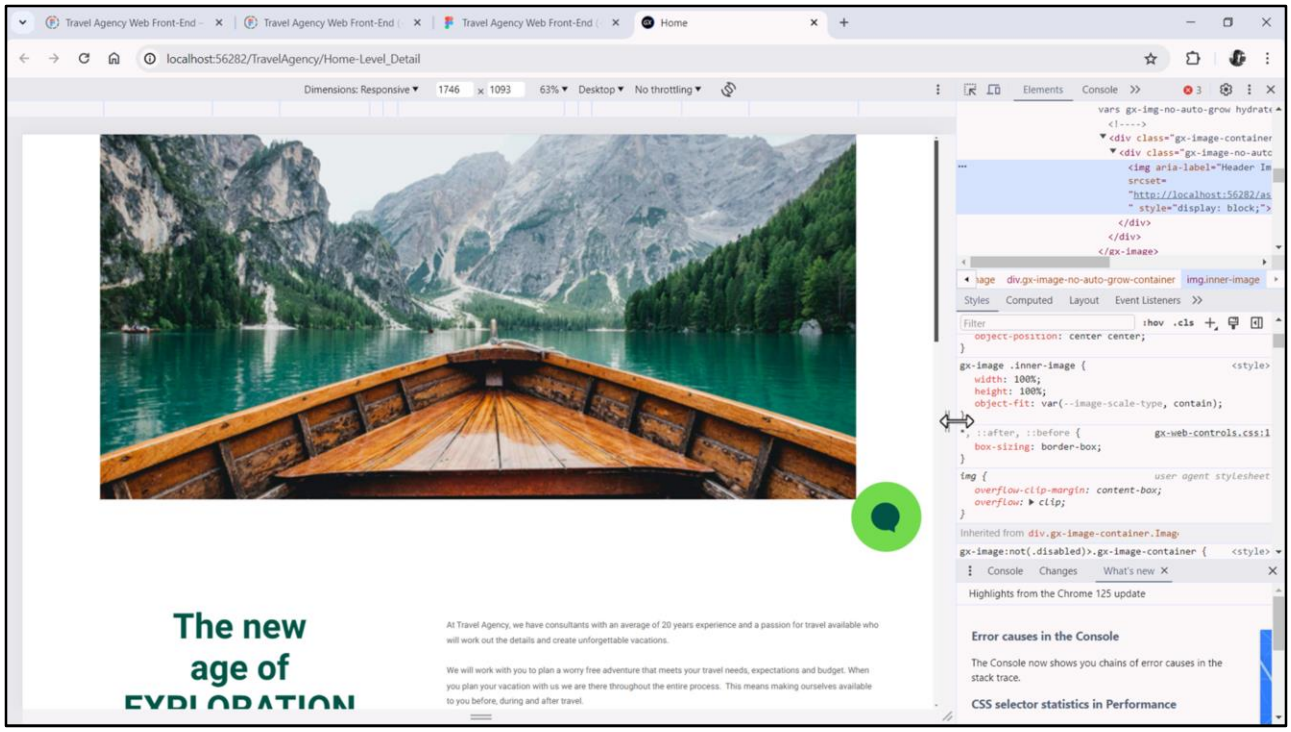
Here we see the Home panel, but something is wrong with the image. It is not reaching the edges.
And if, for example, we reduce the width of the browser, we see that the image shrinks to be always fully displayed. Meanwhile. the chatbot button is always where we said, at a distance of 660 dips from the top position. But what about the image? Let's see what it is doing.



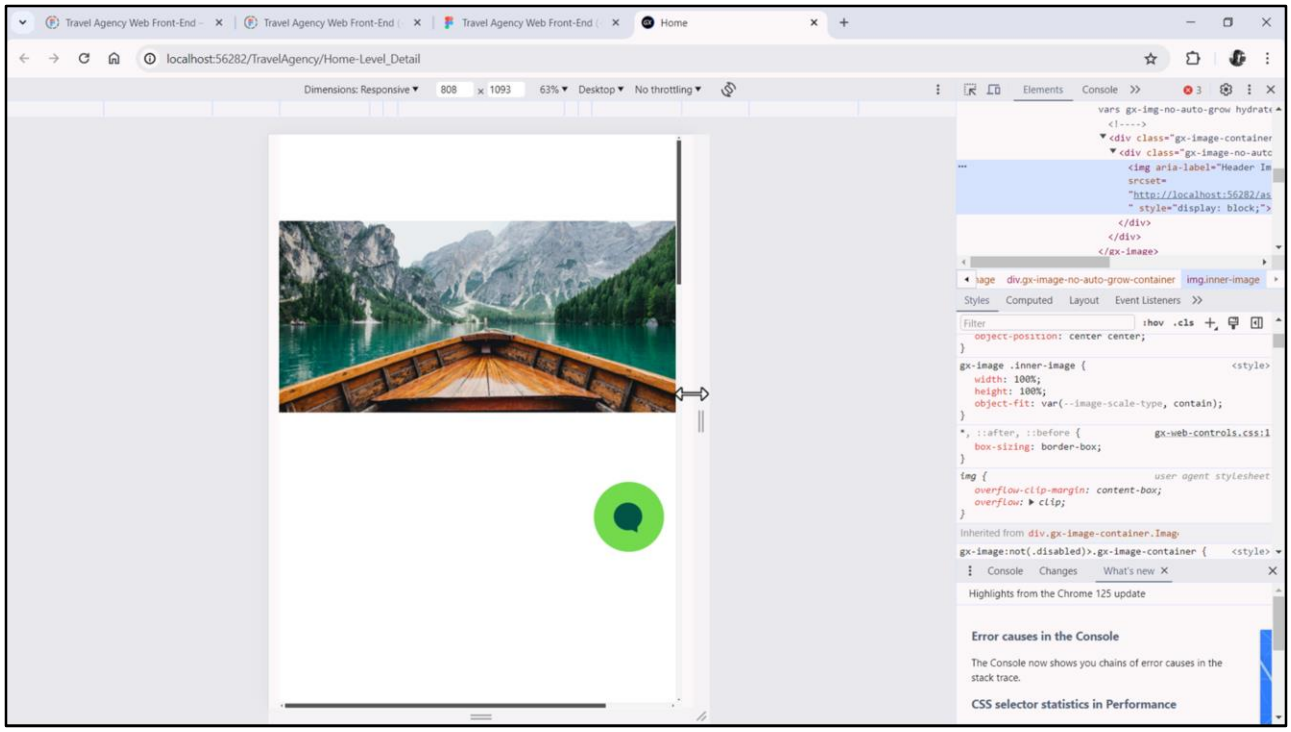
The image we inserted in the KB is 1440 pixels wide and not 695 pixels high, because we had exported it from Figma, remember? With its 3 densities.



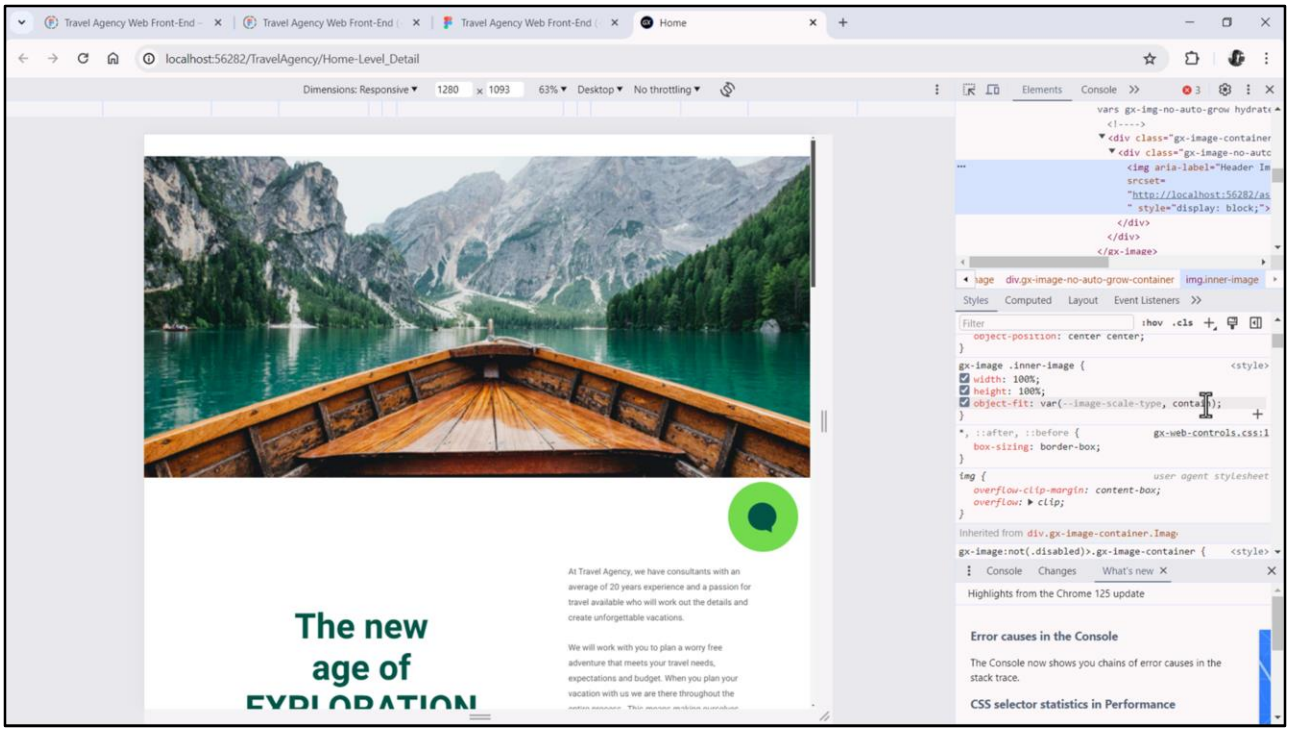
If we inspect it in Chrome, we can see that when the screen width is 1440, only there it looks the way we want it to.



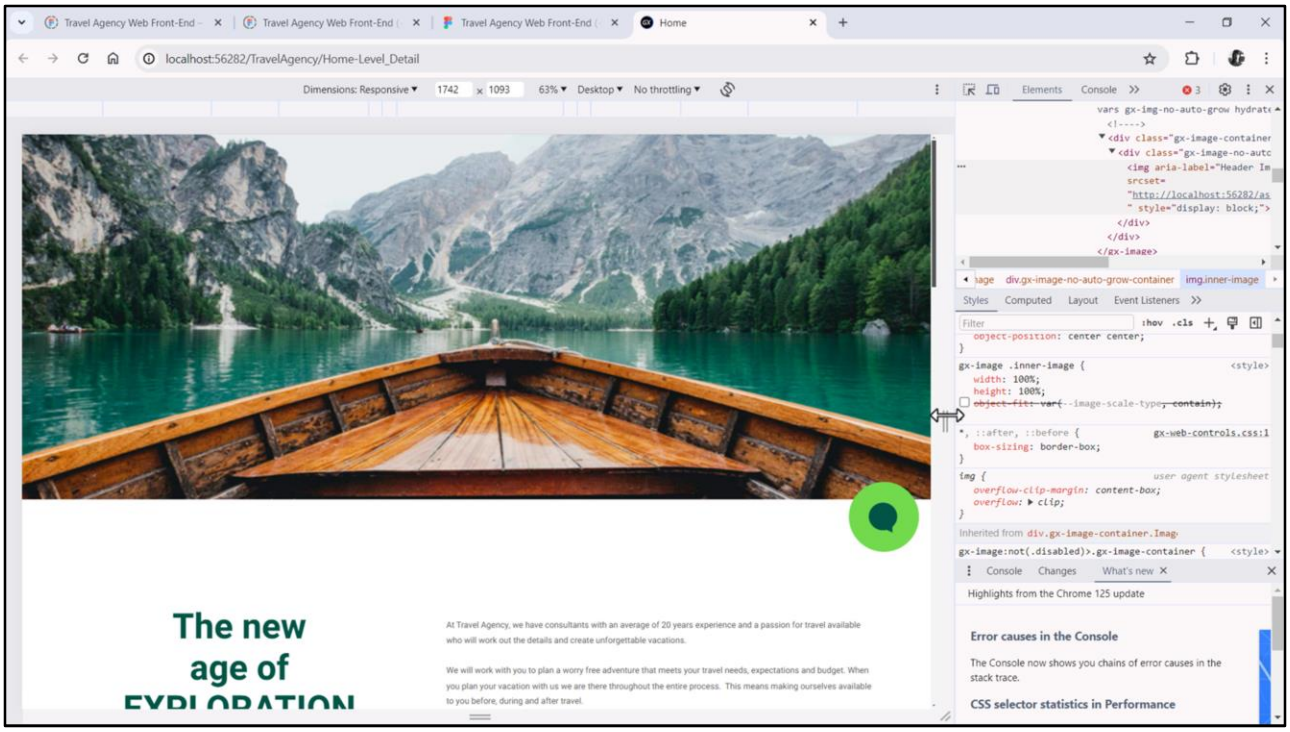
On the other hand, if we widen it, it remains fixed, at its exact size.



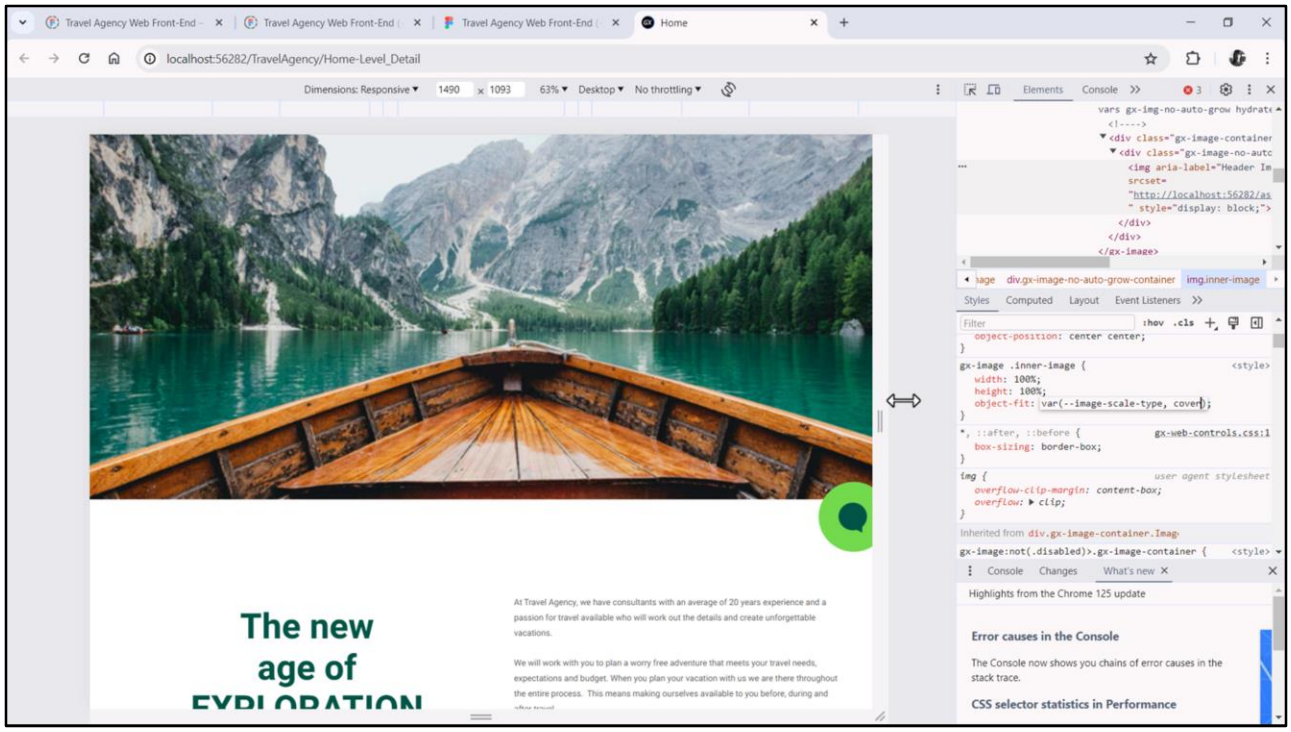
And if we reduce it, on the other hand, it starts to decrease proportionally so as to fit completely into the space available.



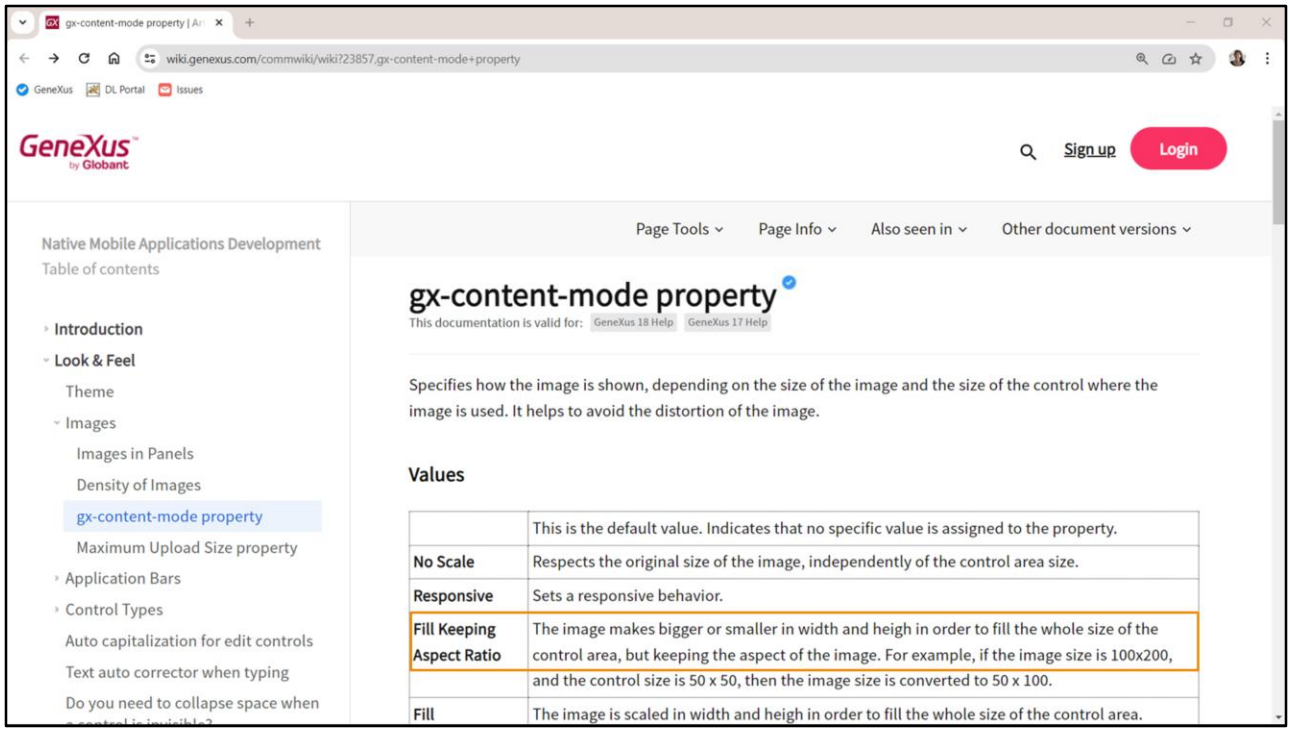
What is happening has to do with this CSS property, object-fit, which is assuming this contain value.



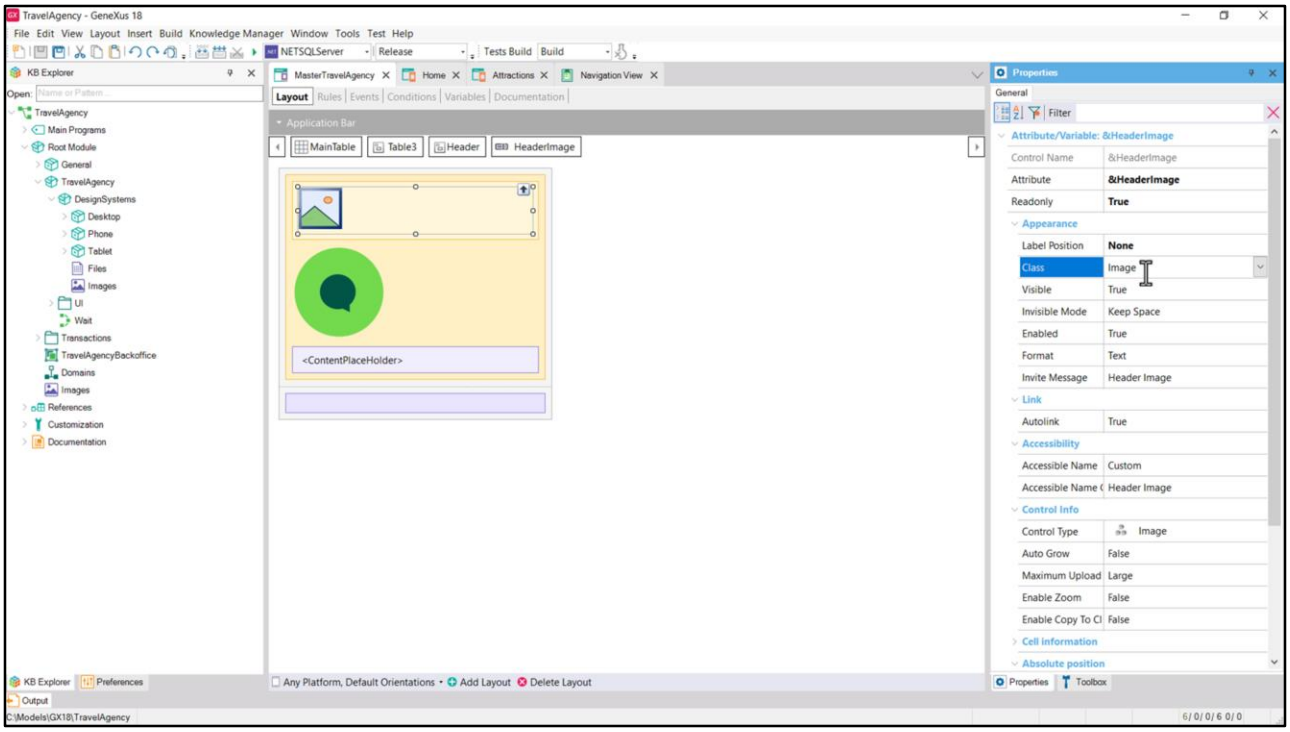
Let's see what happens if we remove it... it stretches or compresses so as to take up the entire container. This is not what we want.



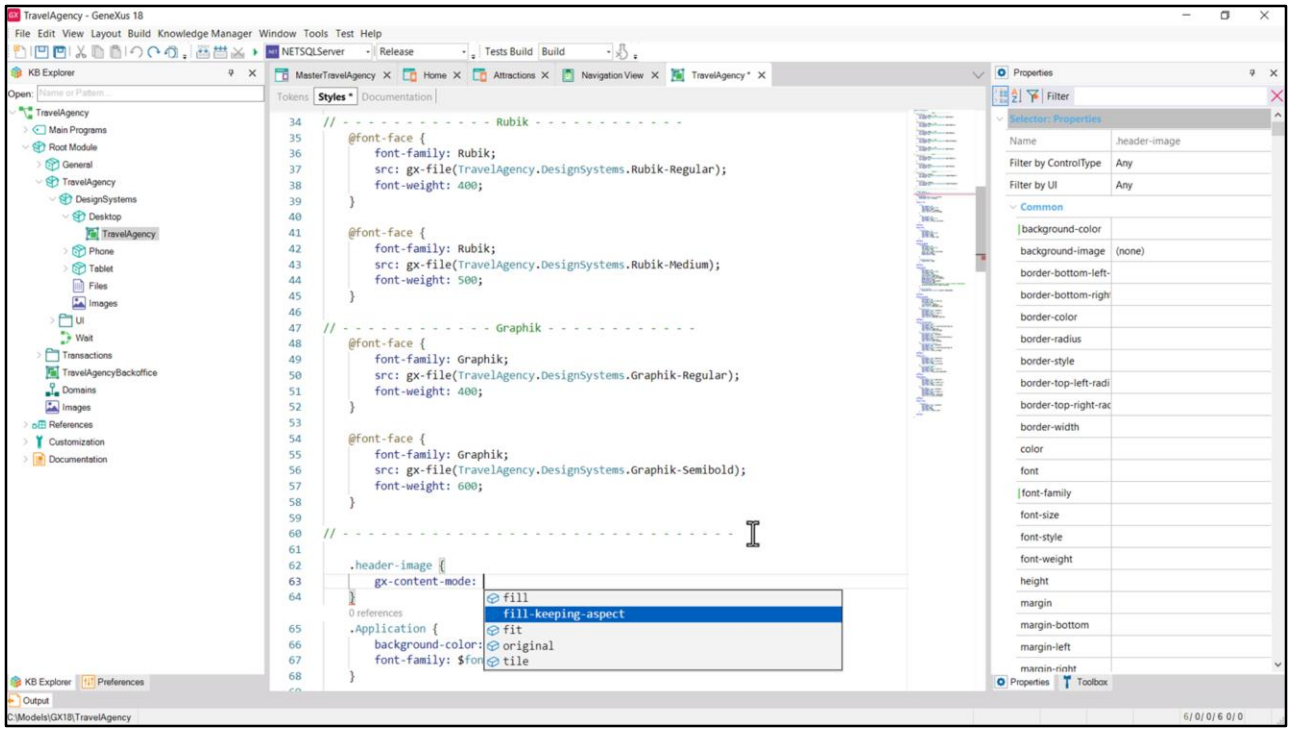
We will want the cover behavior... That is to say, that it stretches or compresses to occupy the whole container, but proportionally, so it will be necessary to cut part of the image, evidently.



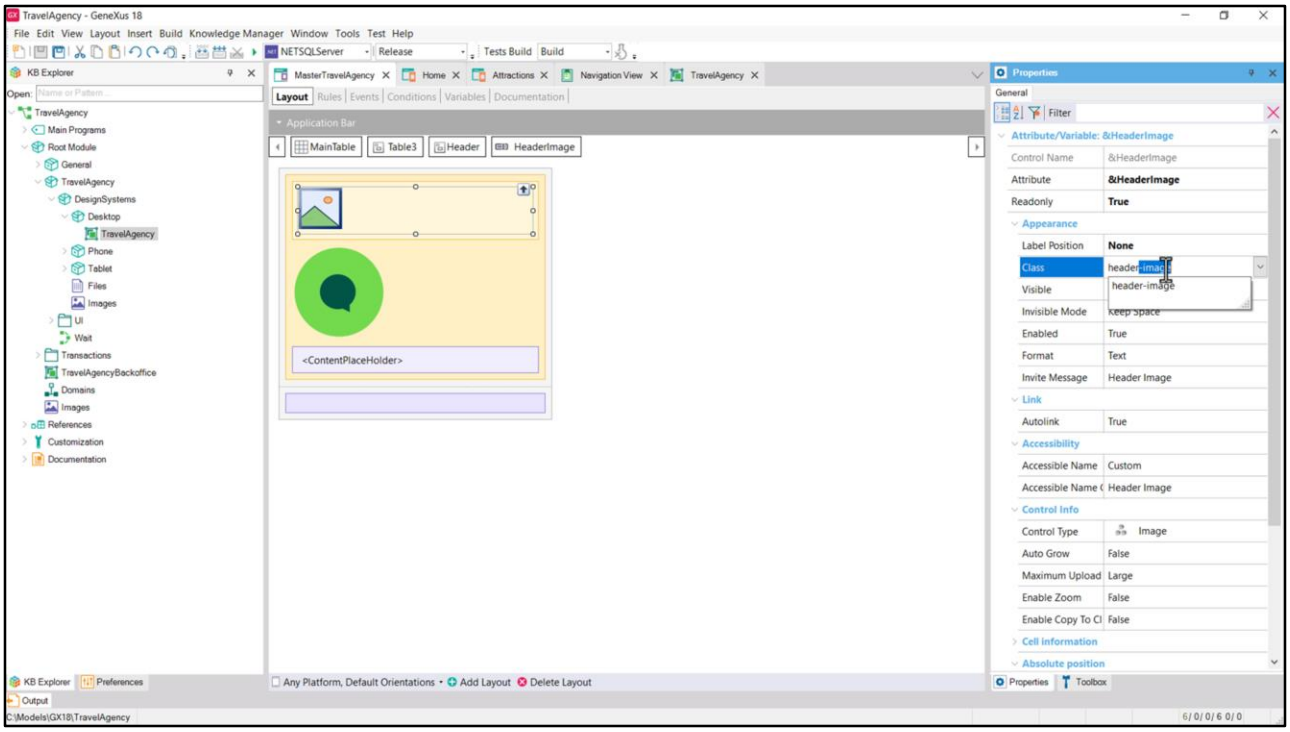
This CSS property in GeneXus has another name, so that it is also valid for native applications. It is **gx-content-mode**, which is explained here in the GeneXus wiki, and we see that it applies to Android, Apple and Angular. We will want this behavior: that it fills the space while keeping the proportions.



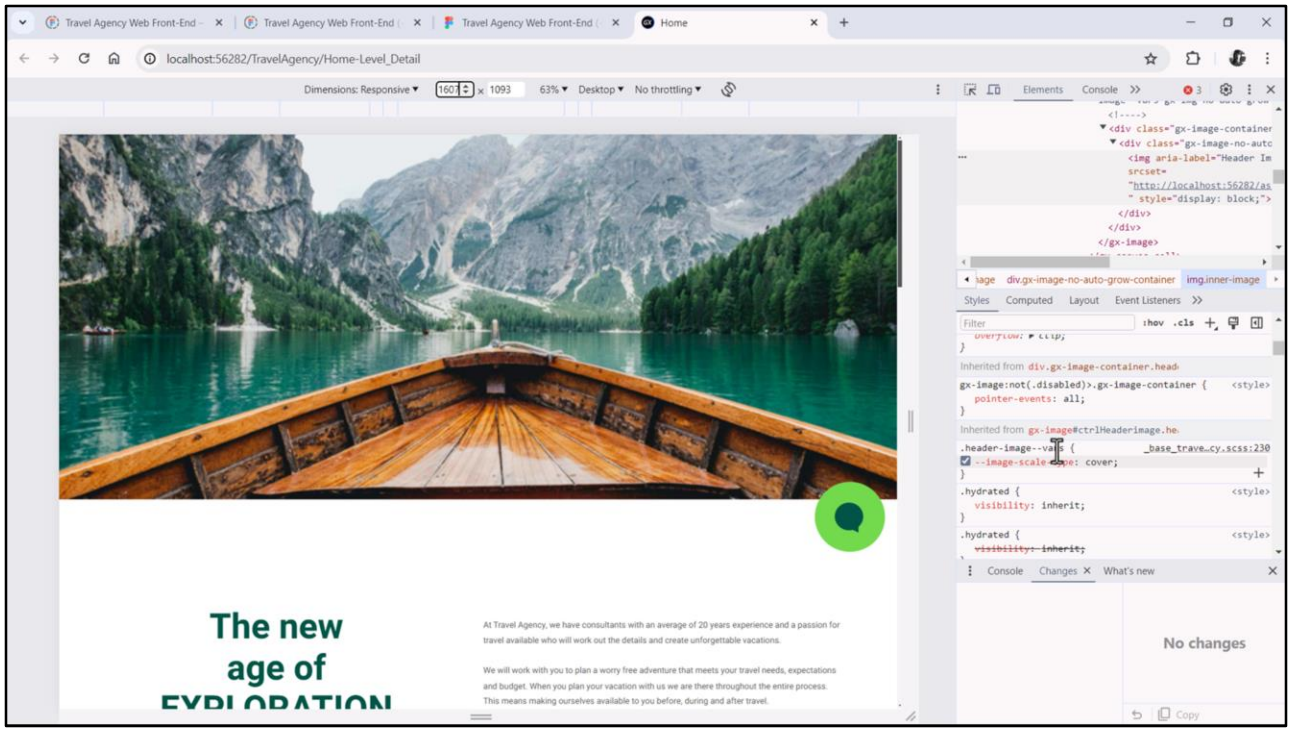
Therefore, we must define a class for the image, which contains this property.



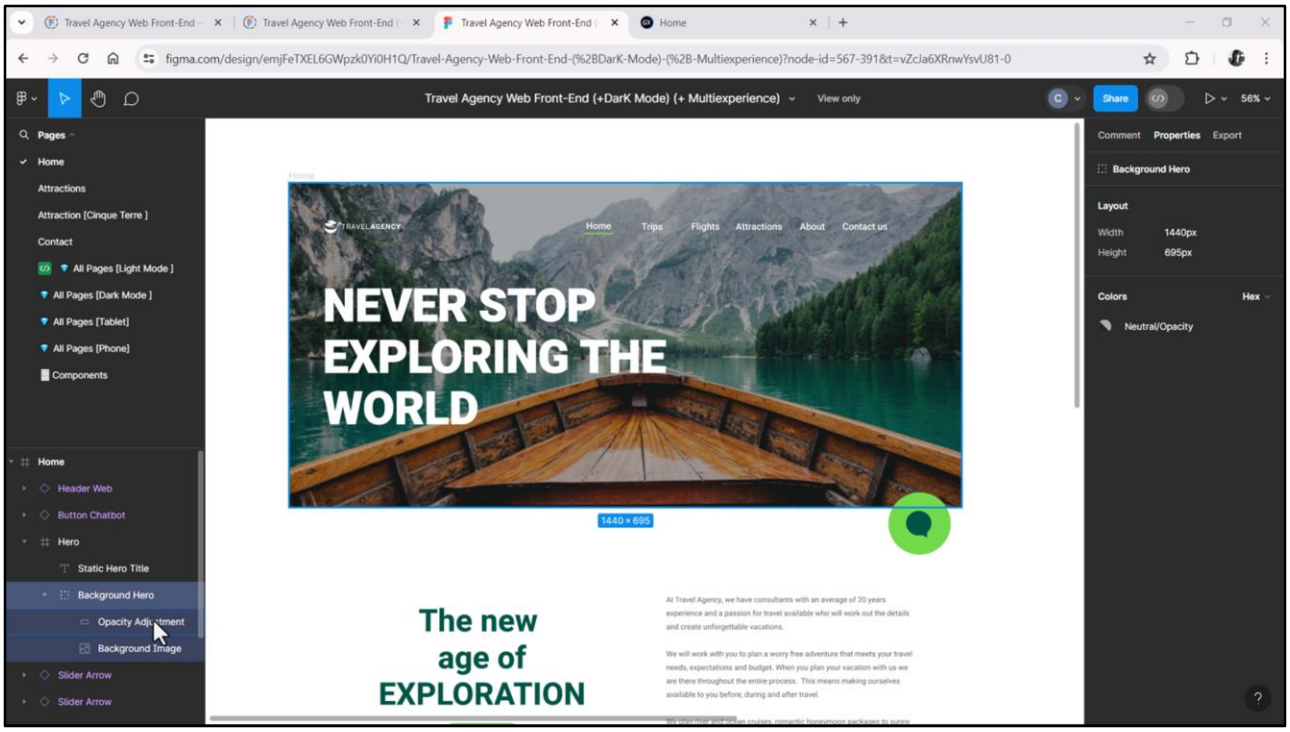
So I open the corresponding DSO, it is Desktop, and somewhere I specify the class, which I will call header-image. And there I specify the property and its value.



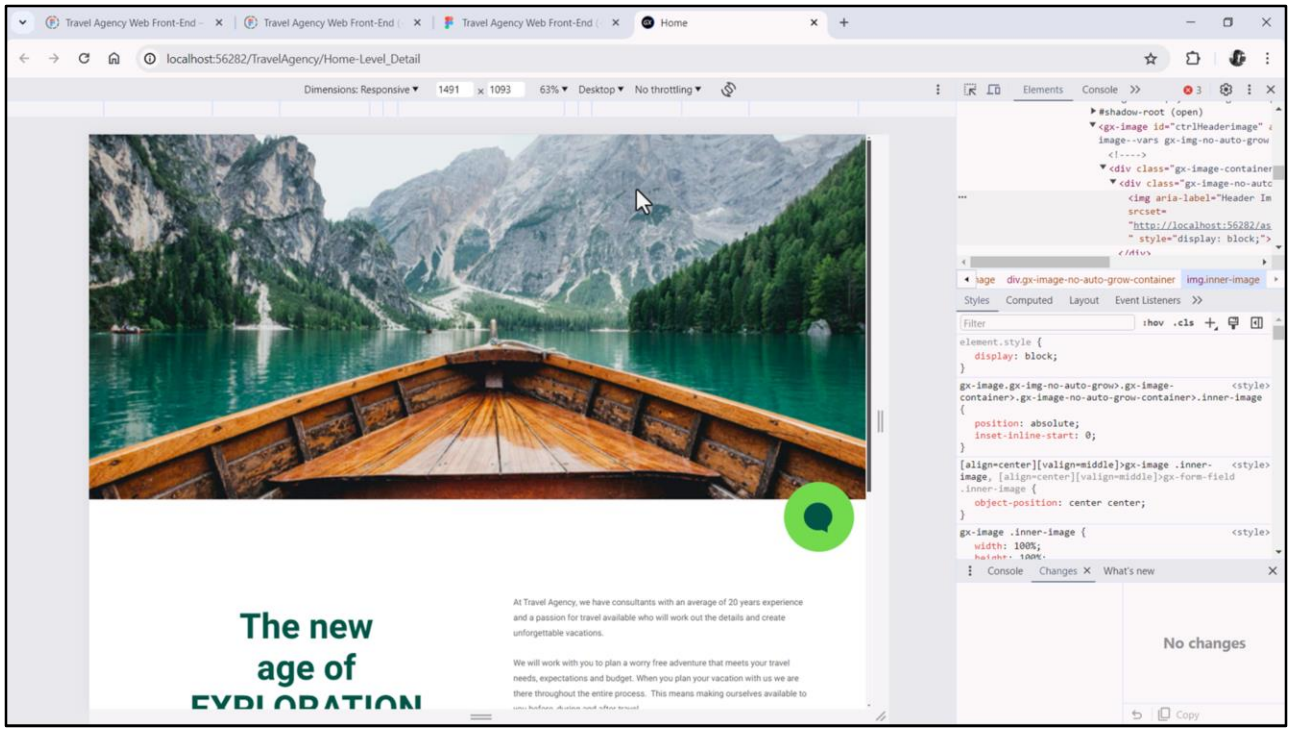
Then I associate it with the control in the Master Panel layout. Let's give it a try.



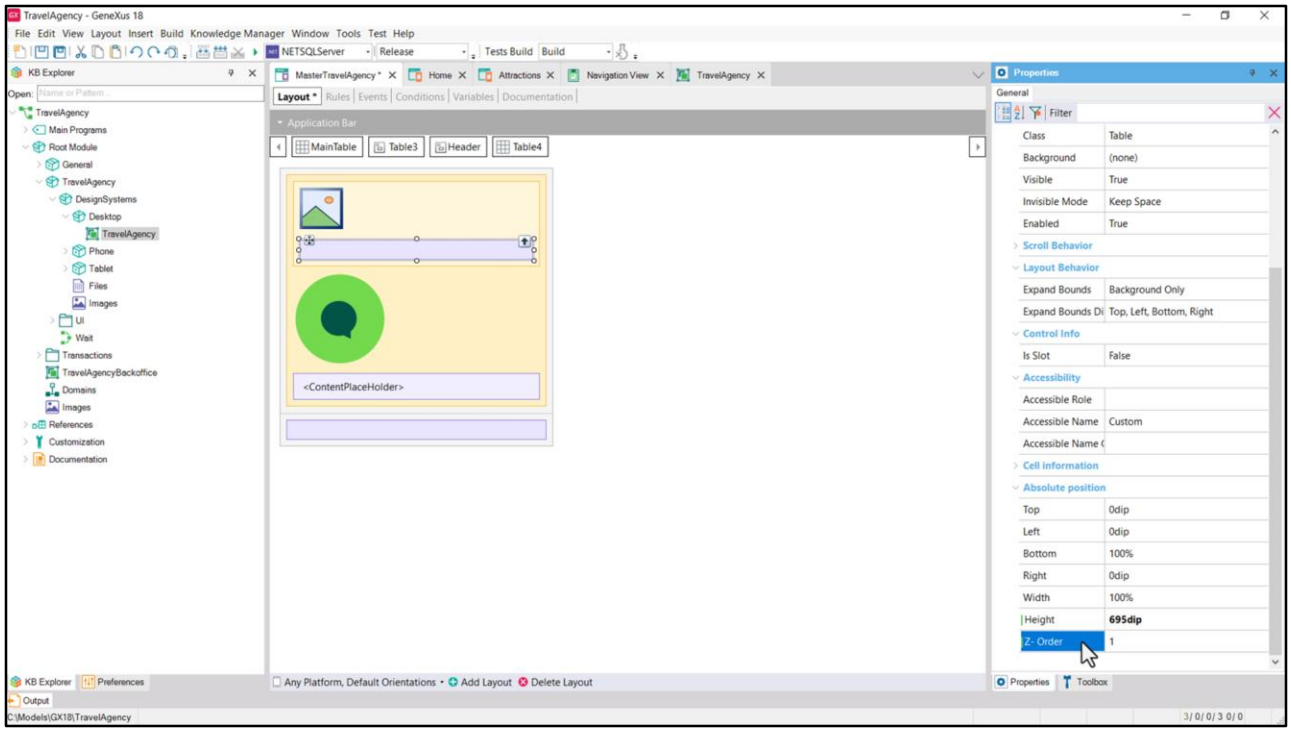
Now when enlarging or reducing, we can see the image as we want. Here we can see the class at work, converted by GeneXus to the necessary code. We don't have to understand any of this.



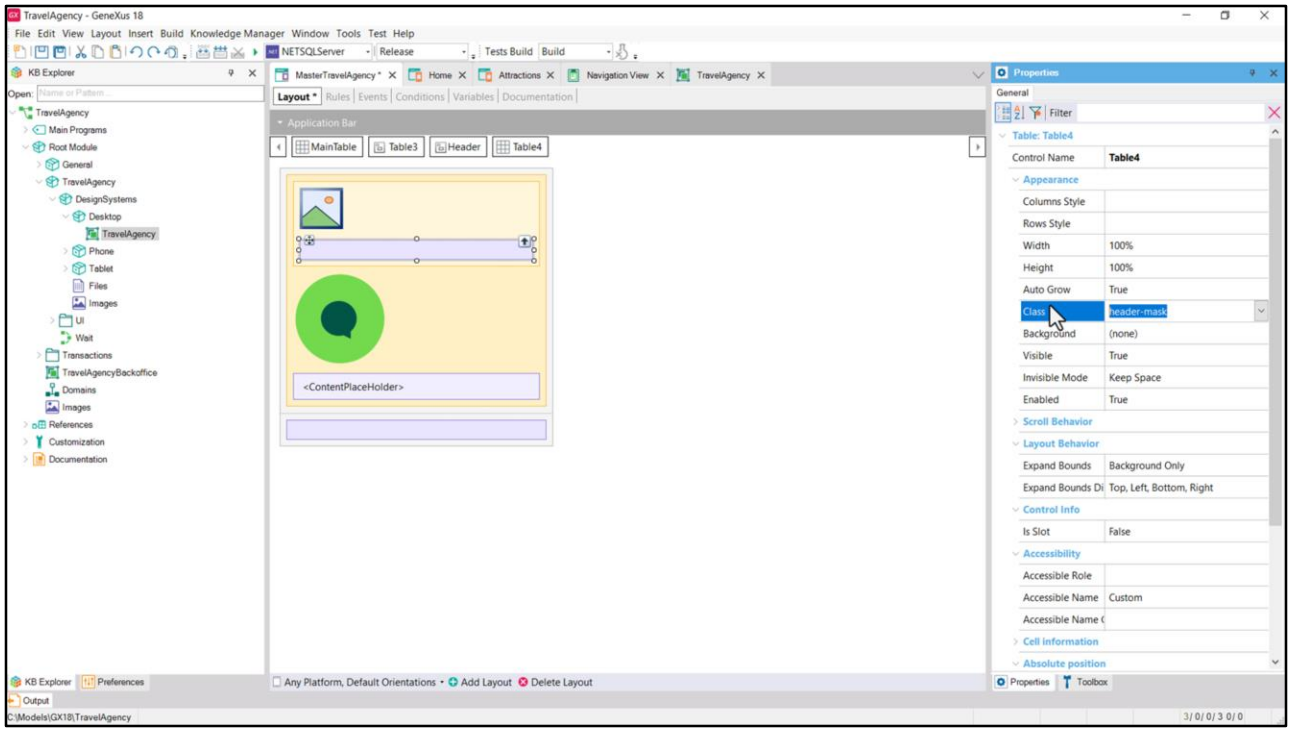
Another thing that we can already notice if we go to Figma, is that there is a mask over the image, to make it darker and achieve a better contrast of all these elements that are going to be overlaid on it.



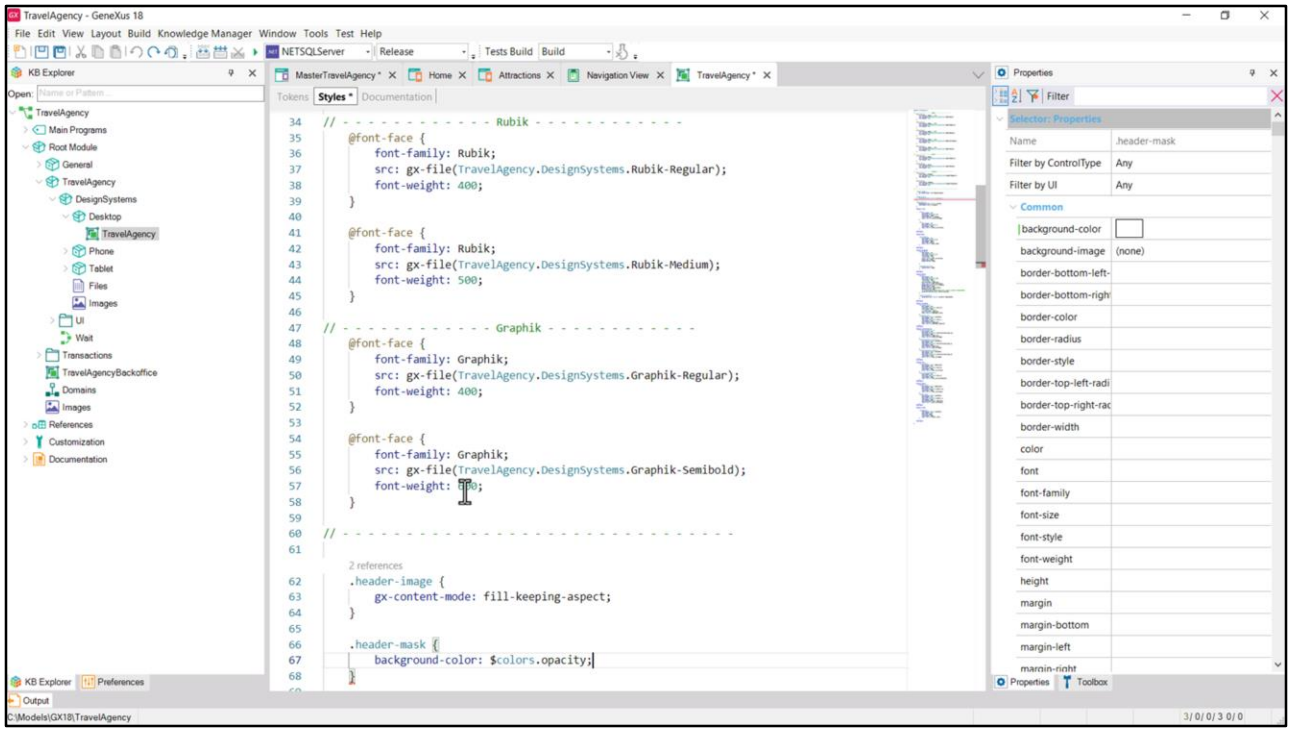
Here it is rather light. So we need that mask.



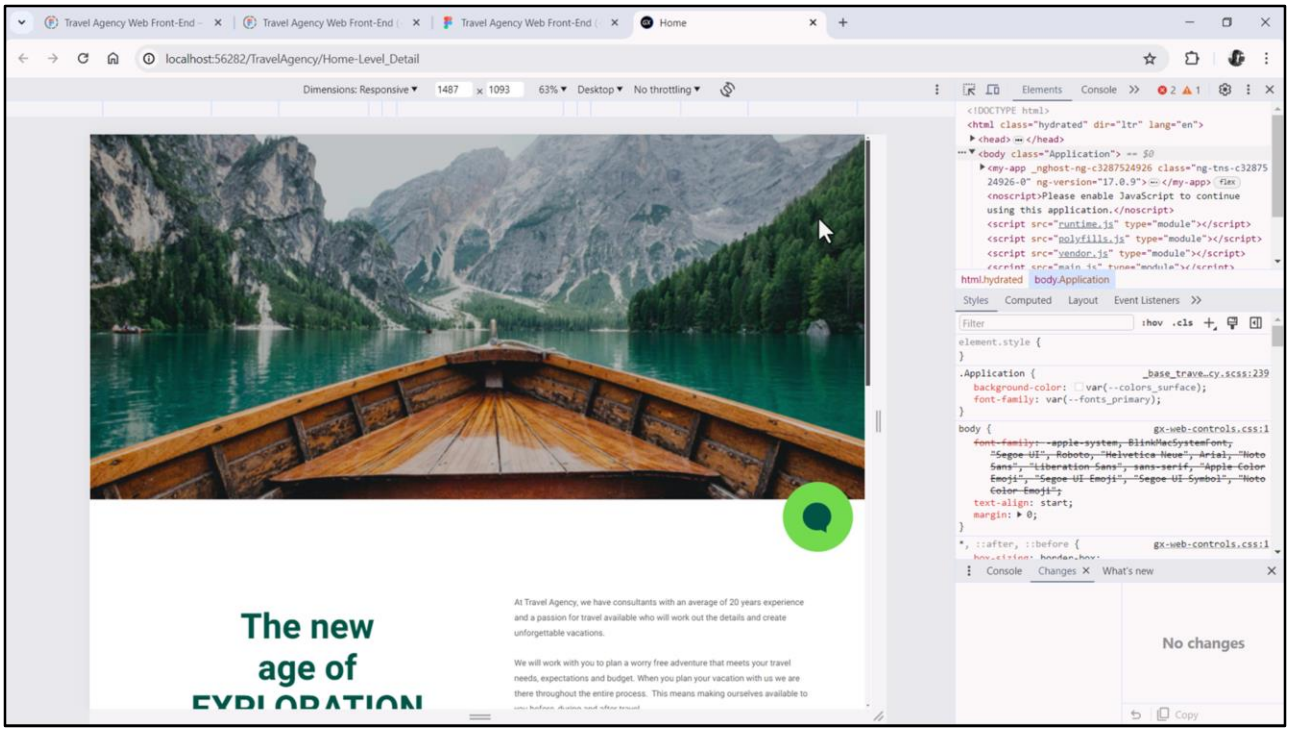
One way to implement it is through an empty table, overlapping it, with exactly the same dimensions, that is, 695 dips high and the rest reaching the edges of the canvas, that is, 0 dips top, 0 left and right and bottom 100%; when it is loaded and calculated it will also be 0, and we can give Z-order the value 1 to make sure it is on top of the image, which had 0 z position.



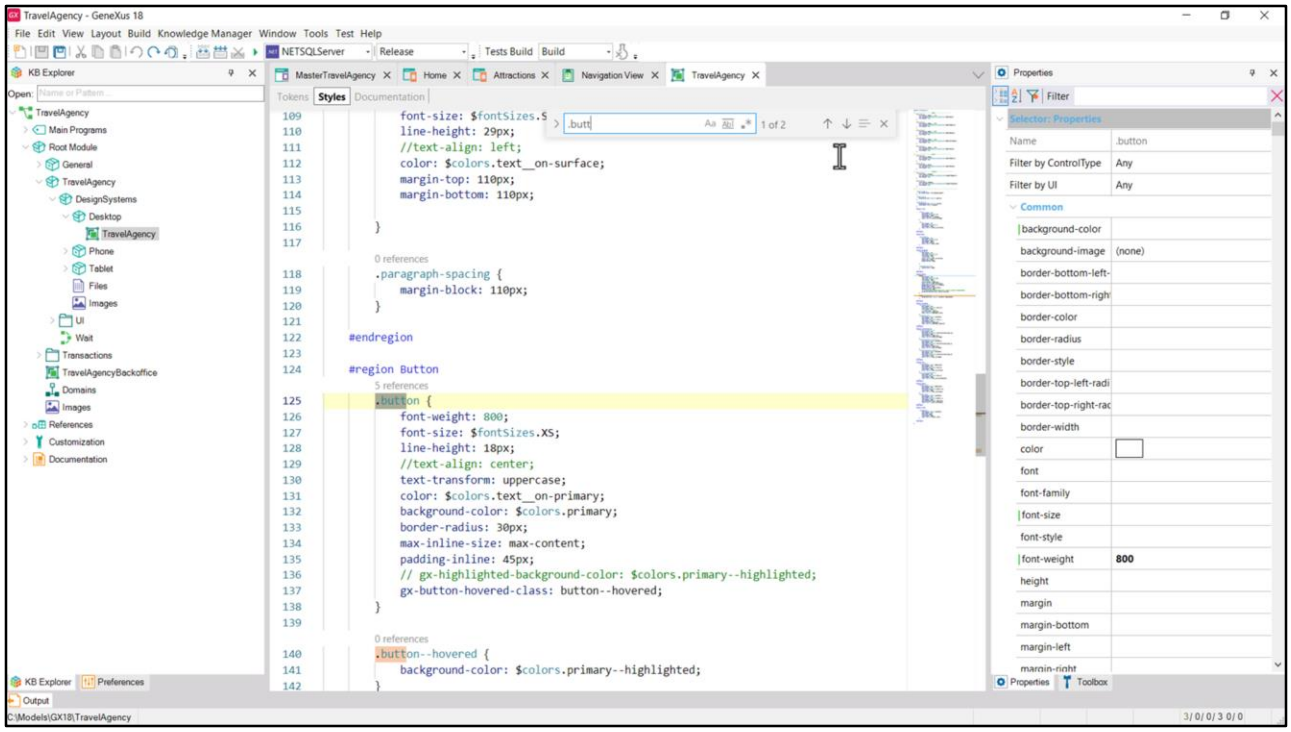
Next, we will have to specify the background color of the table through a class. So we will name it header-mask.



And we specify it in the DSO, remembering that we already had the opacity color token defined. Then to the background-color property we associate that color token.



If we now run it, we see the image with the mask.

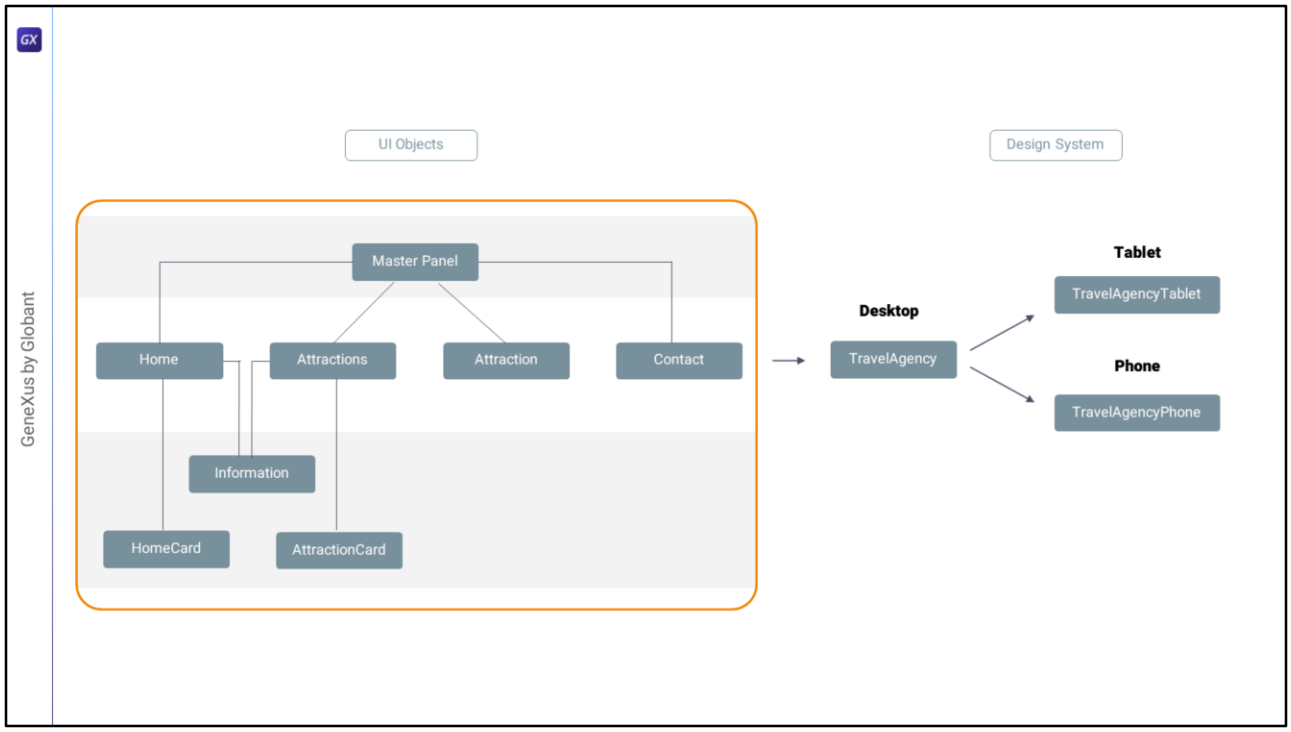


Before I go on, you may have noticed that the DSO is not getting more complex.
When I went to specify the two classes we need to create for the Header: the one for the image and the one we would use for the mask, I wrote them after the font-face rules, but I could have written them anywhere.

If we review what we had specified in this DSO, besides the rules to incorporate the fonts, they were all the classes for the typographic styles of the whole application, and also for the one for the contact button, the ones that controlled the style of the button.

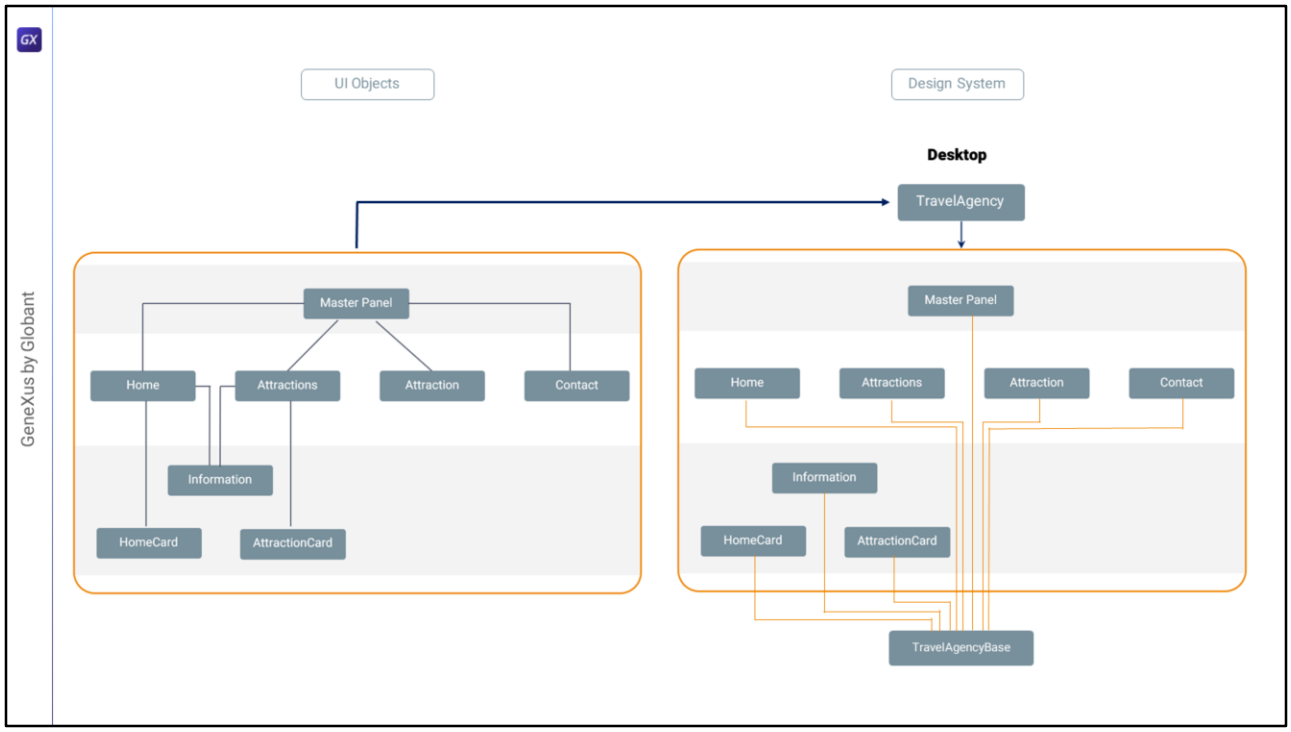
But now we have to start incorporating classes for the different controls of all the layouts. Expect this to grow quite a lot. The editor allows us to search with F control.

To make all this easier to manage and understand, one possibility that I'm going to choose (I'll discuss it a little later) will also be to componentize the DSO in some way.



Let me explain: all we had done so far was to specialize the TravelAgency DSO for what will be the Tablet size on the one hand, and the Phone size on the other.

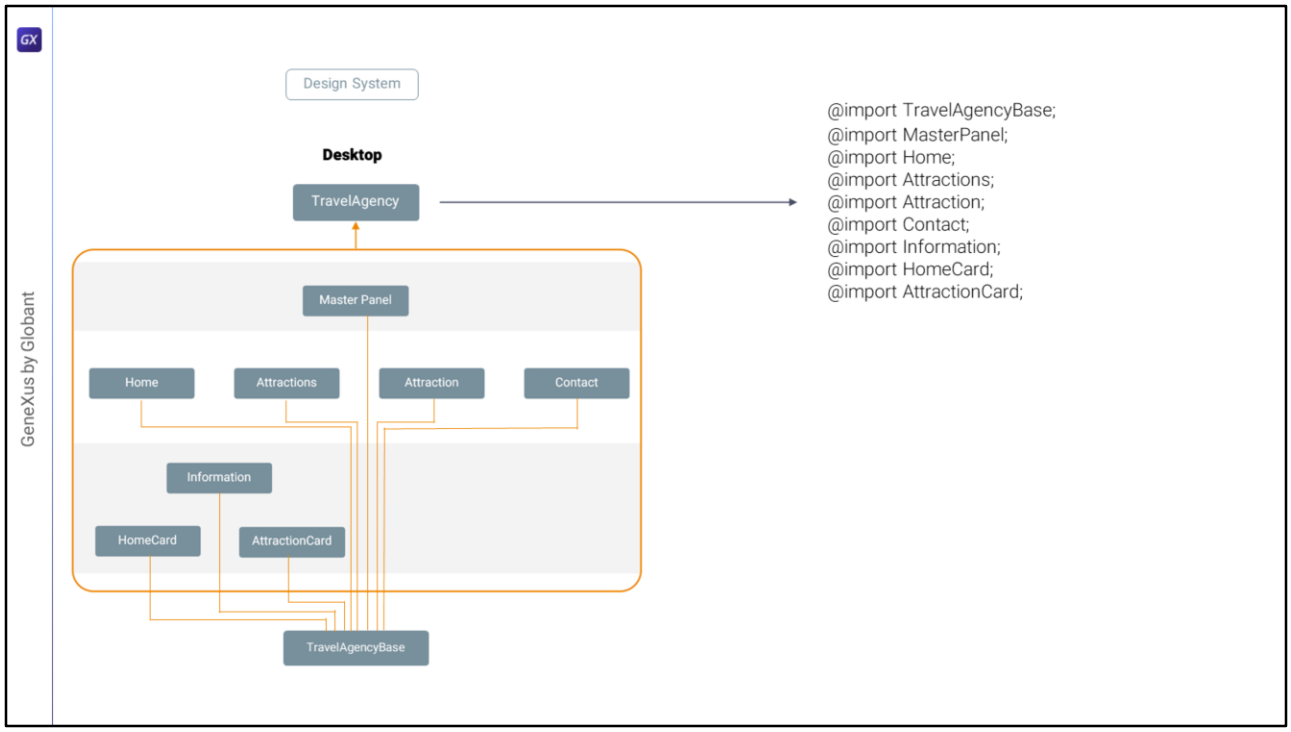
But actually nothing prevents us from having a tree of DSOs for each one and not a single object.



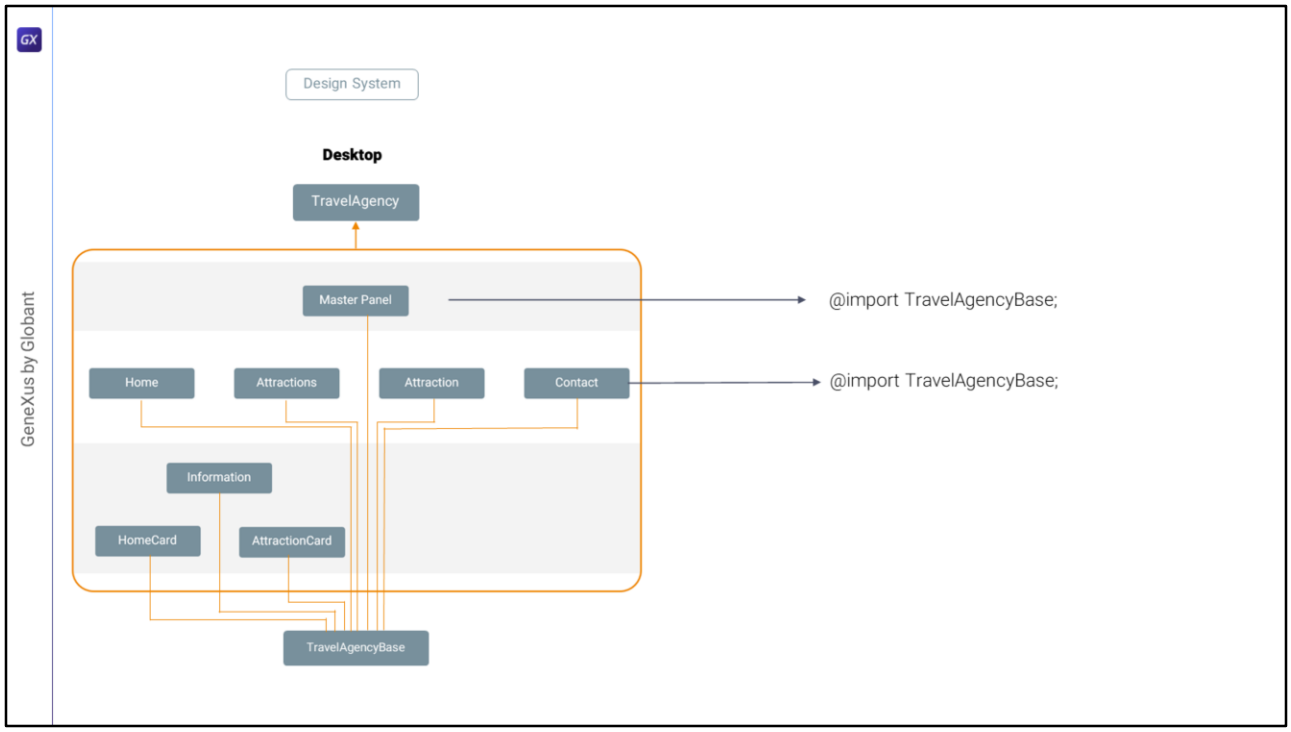
That is, let's only think about the Desktop size... Wouldn't it be more organized, for example, to have a base DSO, with all the definitions for the entire application, such as the inclusion of fonts, and all the analysis of tokens that we already did? If you want, we could even leave there all the classes for the typography... and then have as many DSOs as objects, which define, each one, the classes that will apply to the controls of its layout.

Each one importing the general definitions of the Base DSO, and specializing what they need, and defining their own classes. The criteria should be that everything that is common to several objects should go in the Base DSO, and what is absolutely specific, in the DSO of the object.

Since we cannot set a DSO for each UI object, because there is a single DSO for the whole platform, it will be enough to have the parent DSO, Travel Agency, which will only import all these others.



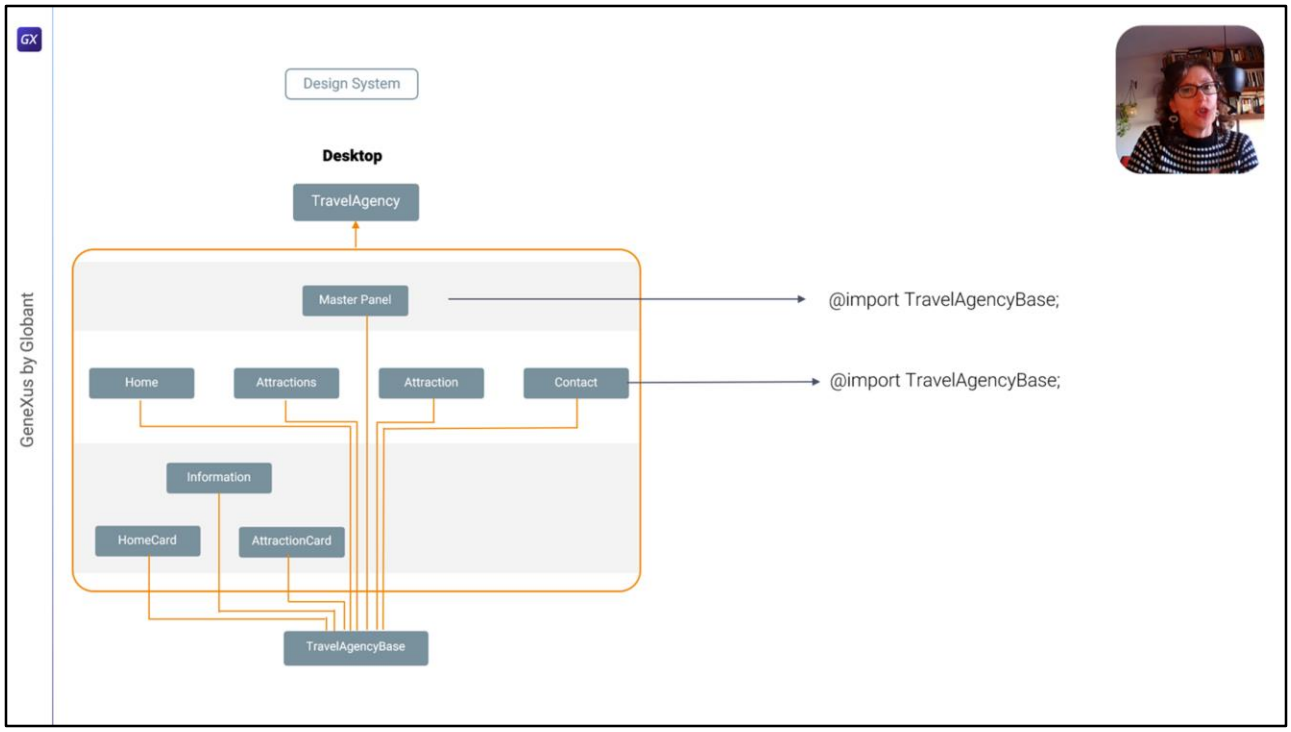
In short, the TravelAgency DSO will only have as many import rules as DSOs for each object we have defined, plus an import of the Base DSO (which in principle would not be necessary because it will be imported by the others, but conceptually it is fine).



Next, the DSO that gives a particular style to each object will probably need to import the base DSO, which has the general definitions. That of the Master Panel, Contact, and all the others.

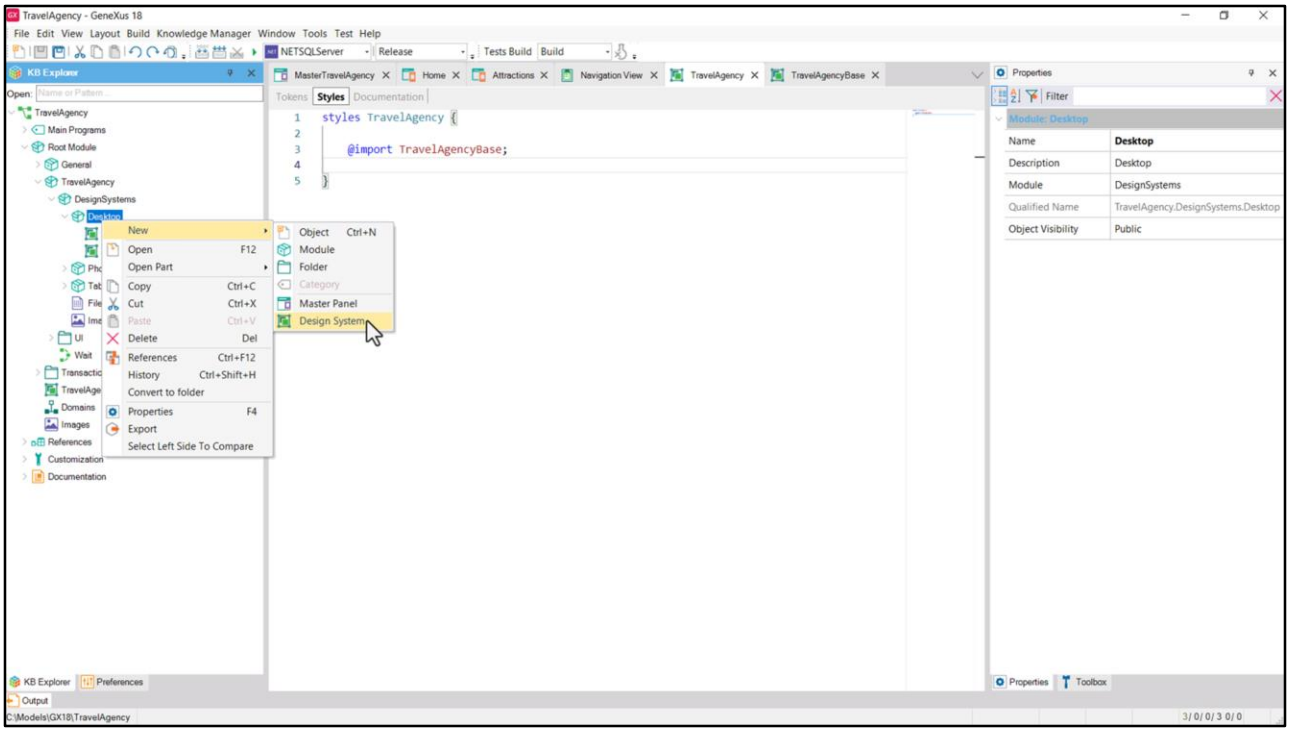
We might think that importing the Base DSO here, and also in each of the DSOs will repeat it unnecessarily and will burden the generated application, in the case of Angular, with a huge CSS. It won't. In the final CSS the Base DSO will appear only once.

We will have to see how granular we go (I'm thinking about stencils, for example). It's always about tradeoffs here.

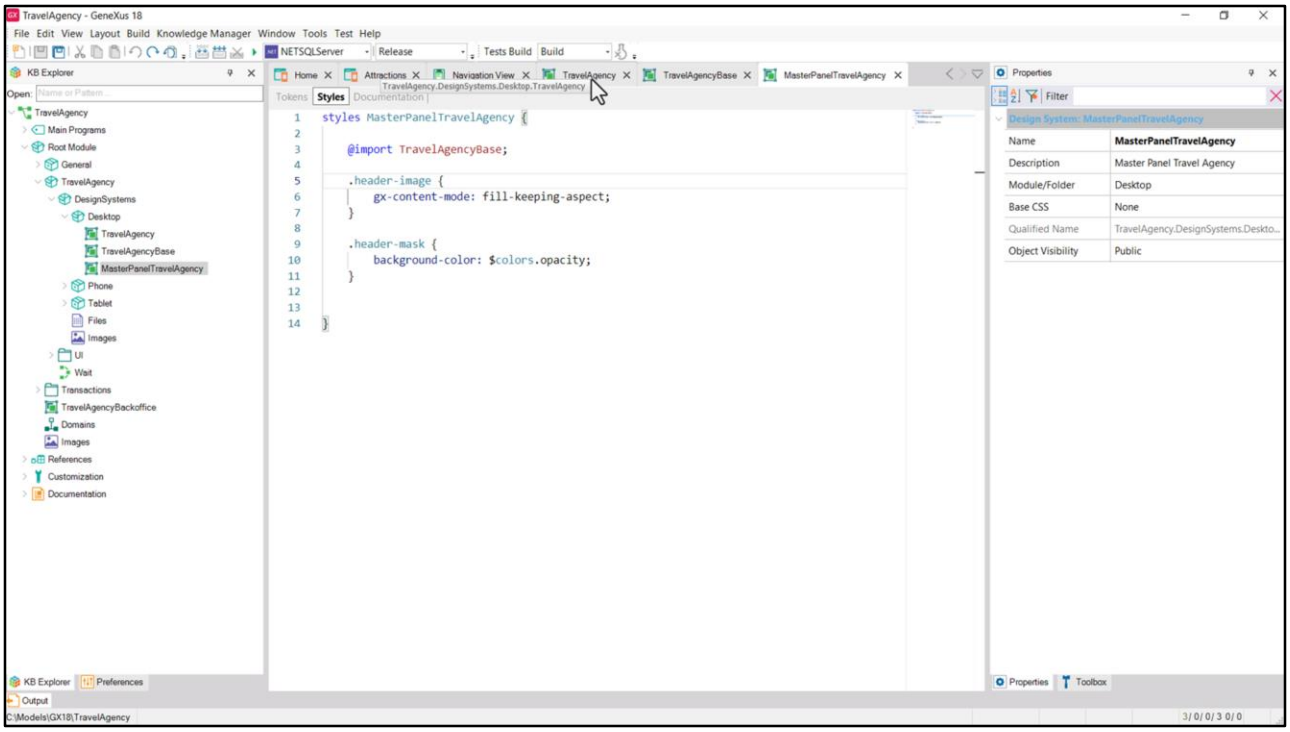


This is a possible solution to deal with complexity, but there is no consensus. In fact, I discussed it with several colleagues who regularly work on the frontend within the GeneXus team and there wasn't much agreement.

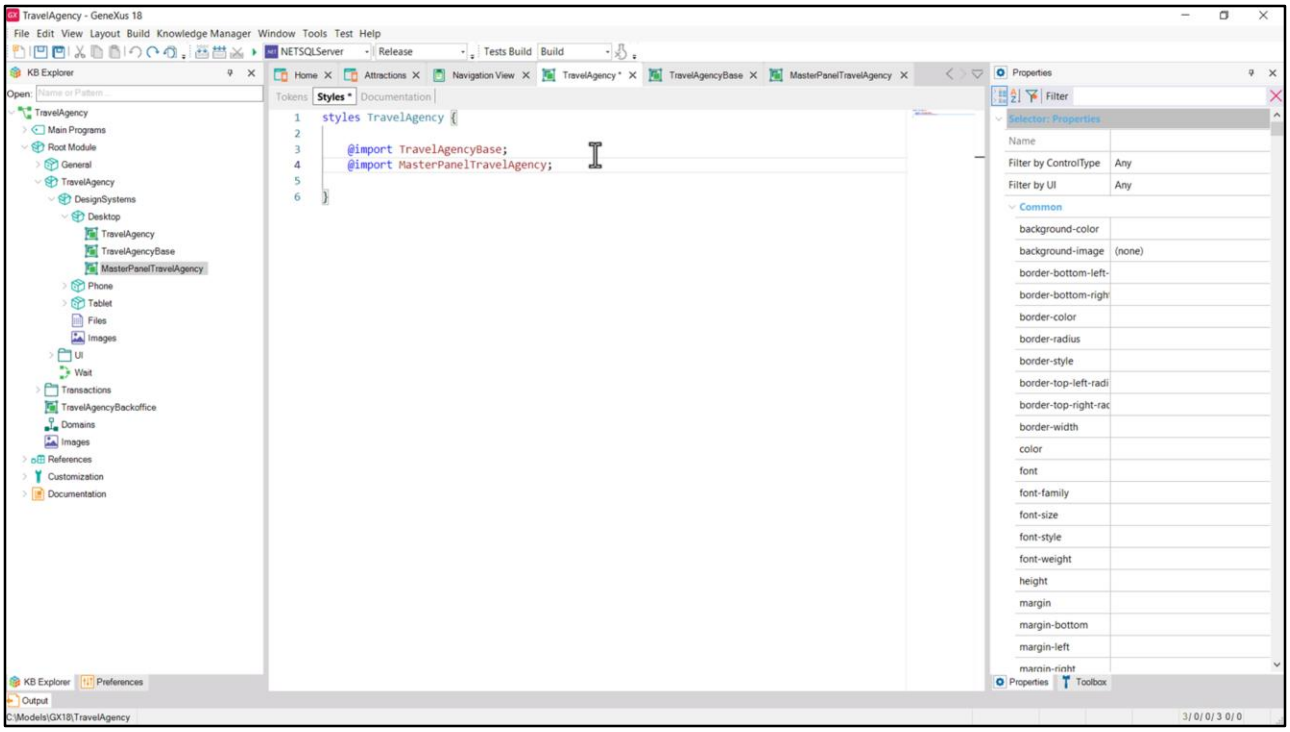
However, I leave it open for discussion; this is not the time to settle this, obviously, but I wanted to introduce it because it is the one I'm going to start using, and in any case we will be able to question it later on.



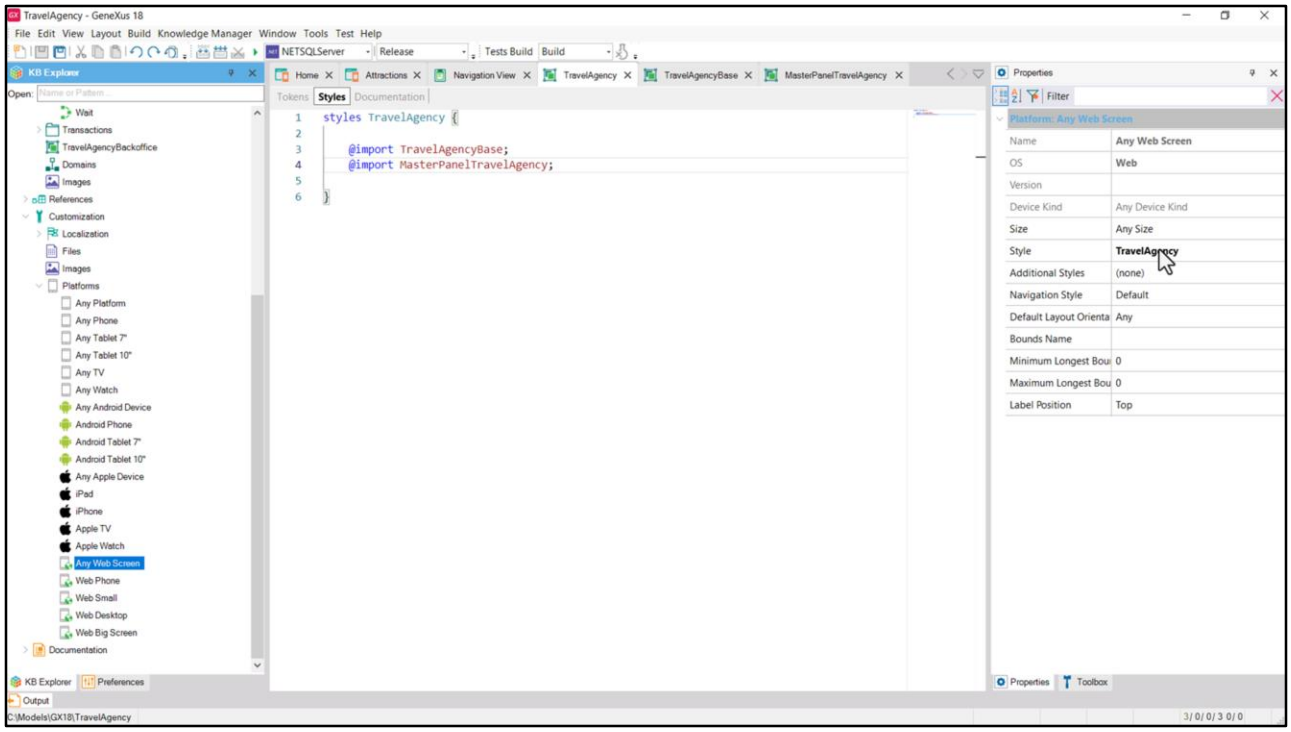
In short, I will save this object with the name TravelAgencyBase.
I will empty TravelAgency, and for the moment I will import this other one.
On the other hand, I will create another DSO...



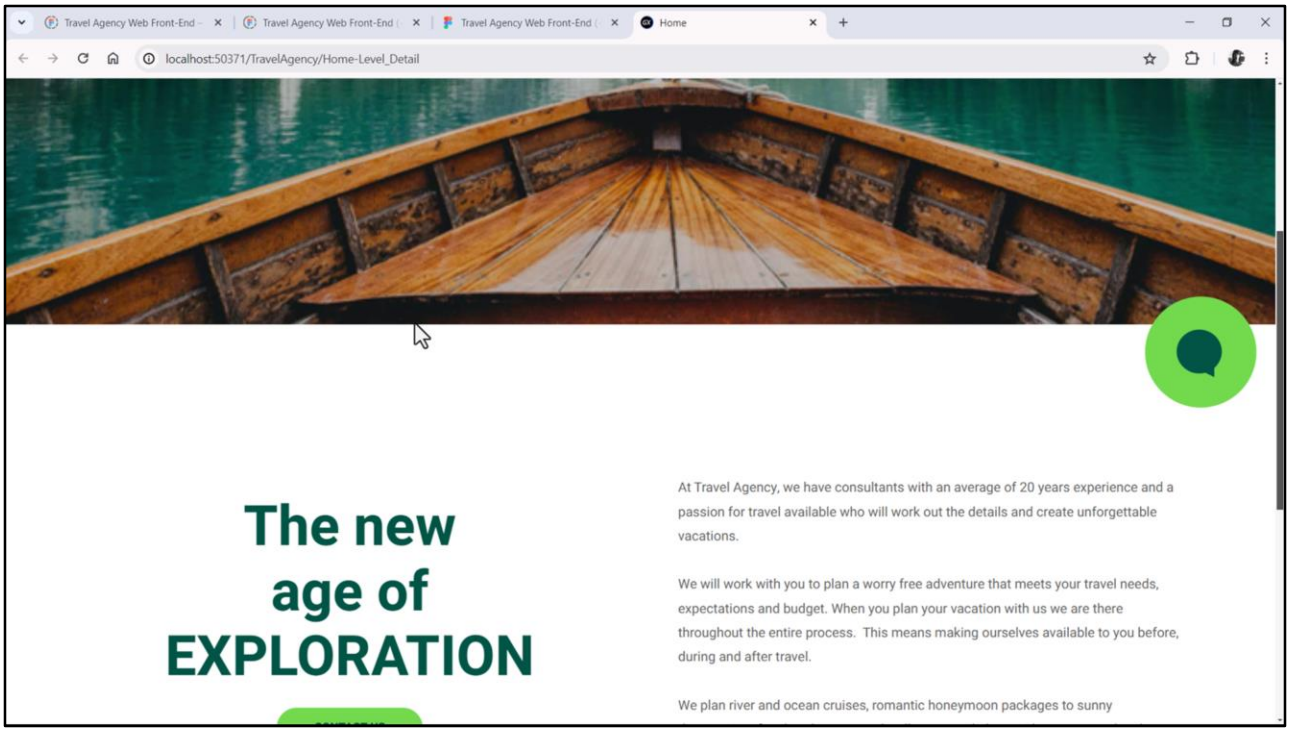
…which I will call the same as the Master Panel. In it I will import TravelAgencyBase, and will move from it the classes that we specified for the Header for the moment.



Finally, I also import this other one into the parent DSO.

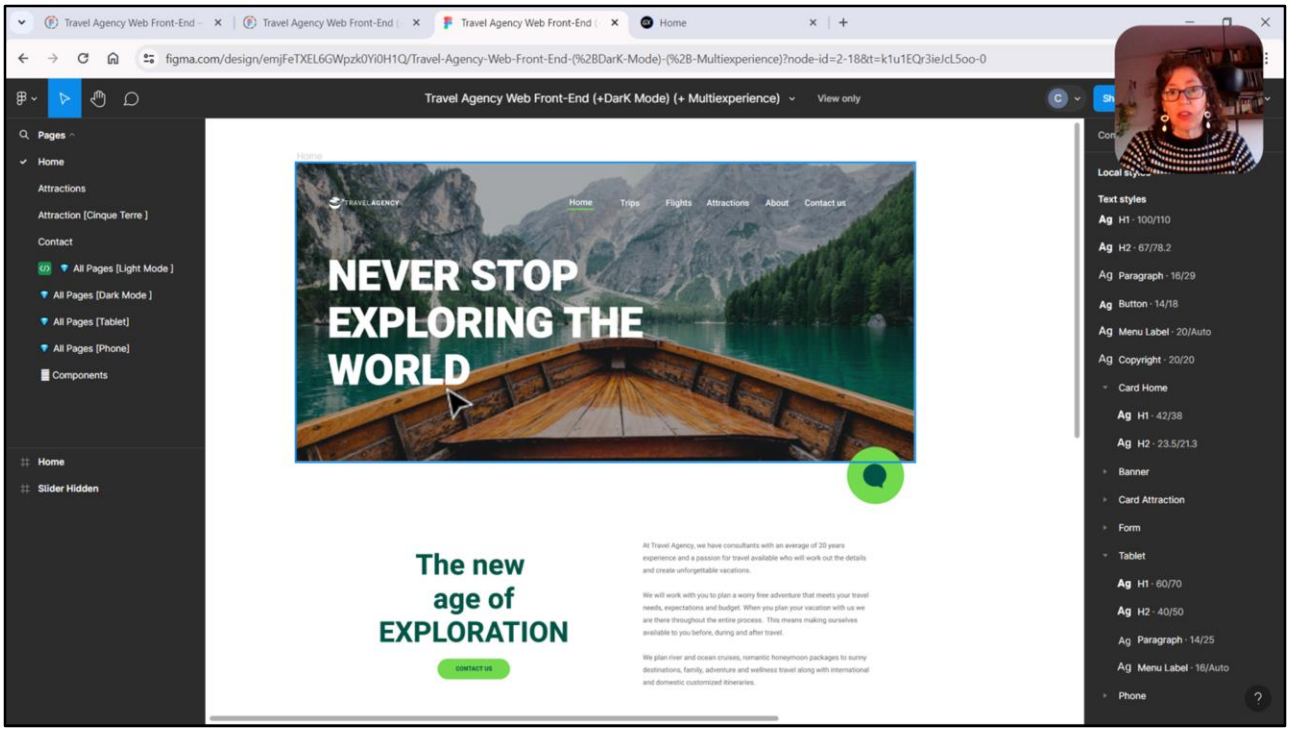


Everything should continue to work in the same way, since the platform we are running on still has this DSO, TravelAgency, which now imports these other two. So it will contain both the classes and tokens of the Base, as well as those of this other one.



Let's run it to make sure that everything looks exactly the same.

Yes, it looks exactly the same.



Since this video is already a little long, let's continue in the next one with the text we want to overlay on the image, the logo, and the menu.


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