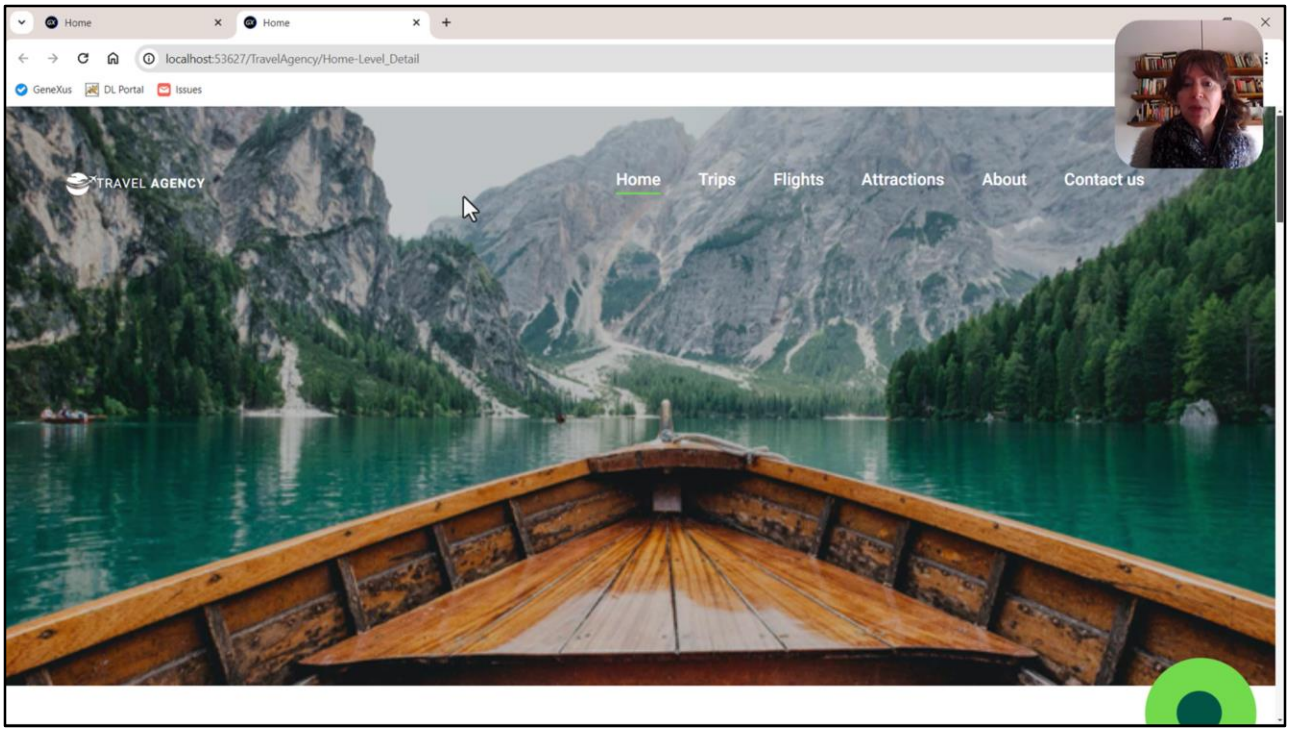
GeneXus by Globant

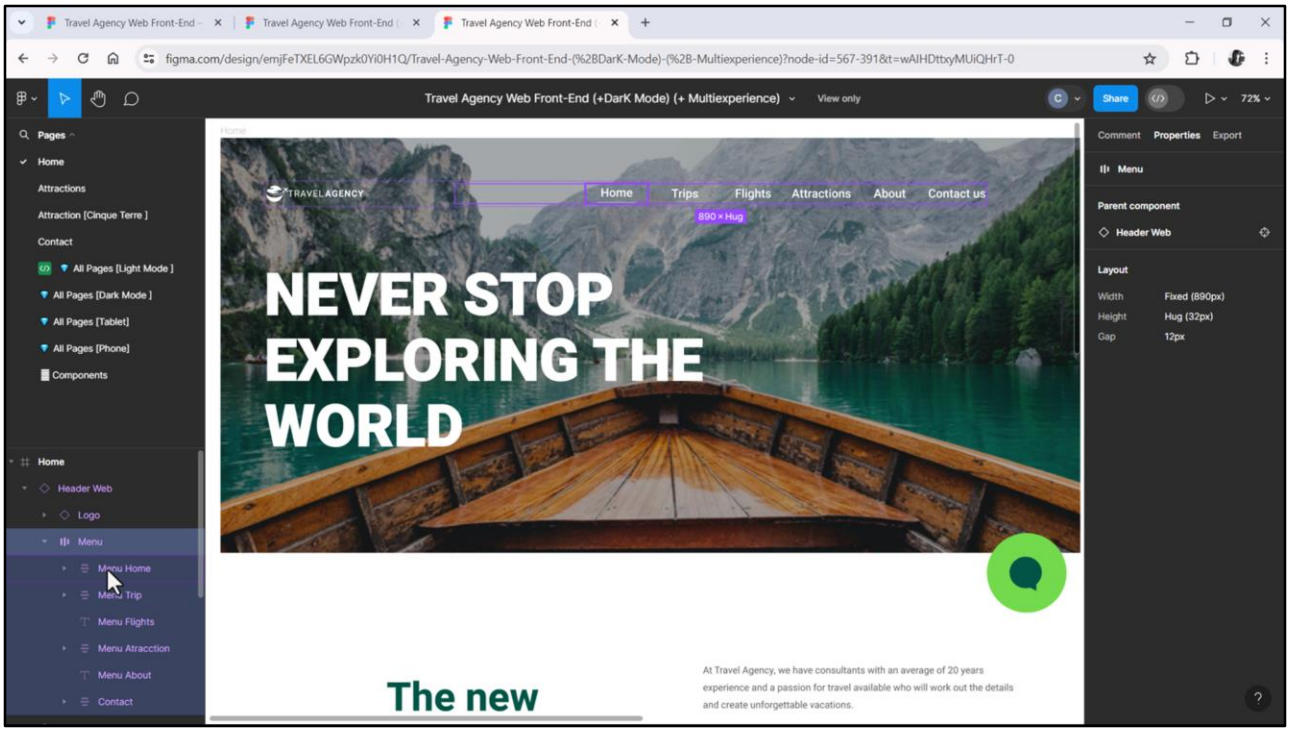
Header in GeneXus: logo, menu and title

Cecilia Fernández



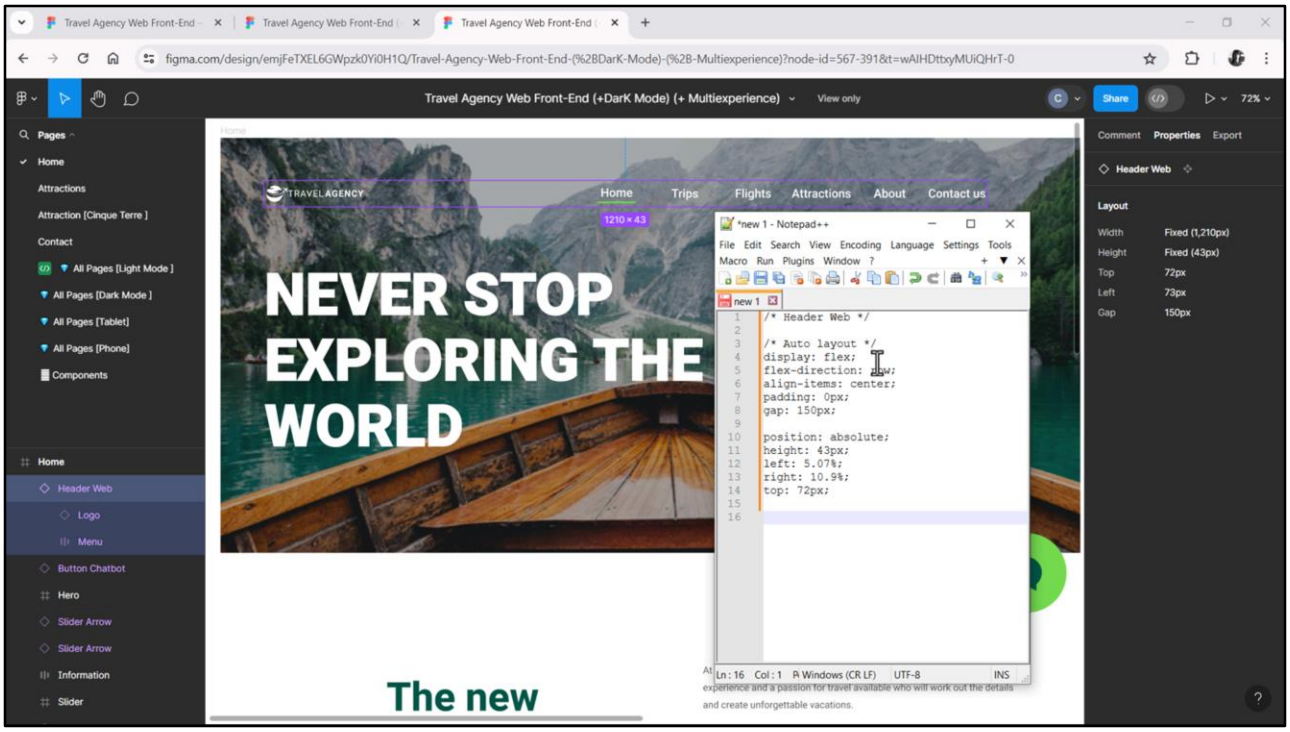
In the previous video we had obtained this, that is to say, we had managed to implement the Header image. What we are going to do now, in this video, is to implement the logo and the menu. And we are going to leave for another video the implementation of the change of screen when navigating, that is to say, that when I click on this Attractions option, for example, not only the page of attractions will load here, which will be in the contentplaceholder, but also everything will change as necessary, including the Hero image and the text that is overlaid.

If we have time in this video, we will implement the text as well. Otherwise, we'll leave it for the next one.



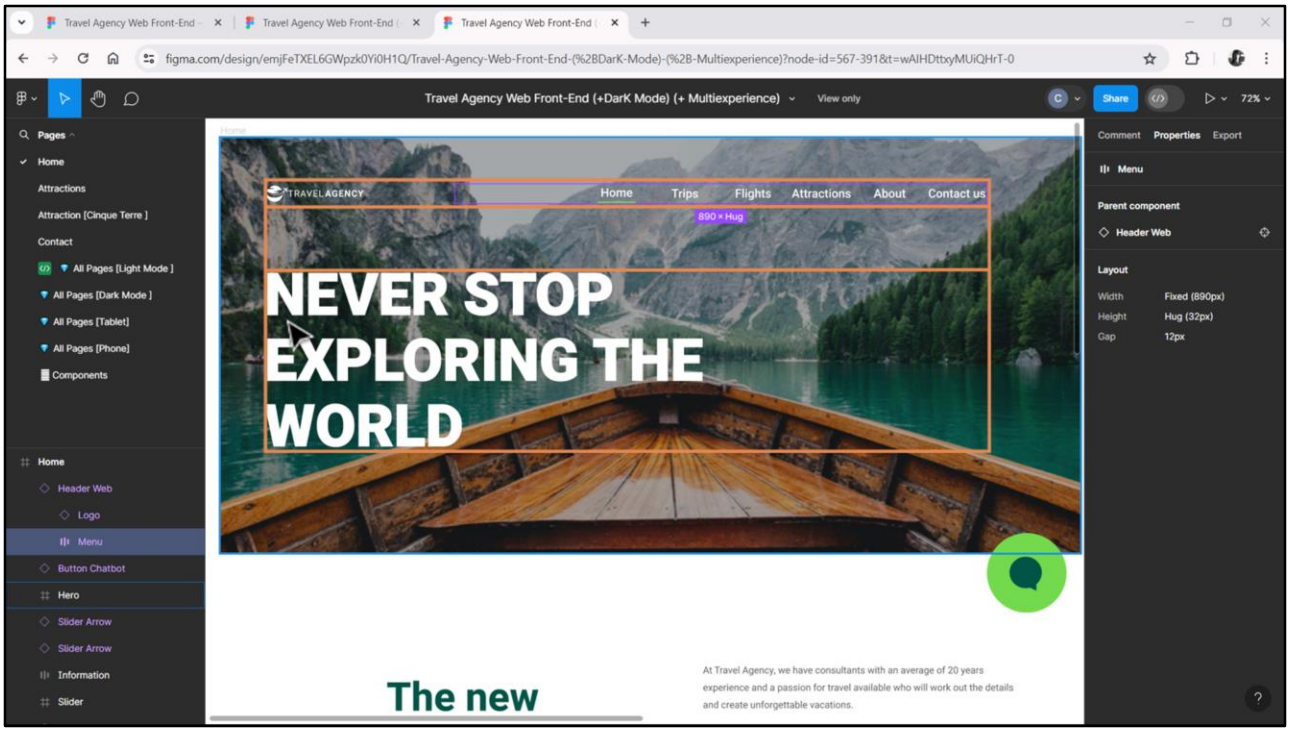
In Figma we see that they are implemented as a component that contains:

- Another component with the logo: which in turn consists of an icon and two texts.
- And a container with autolayout, that is, Flex, for the menu, a succession of items.



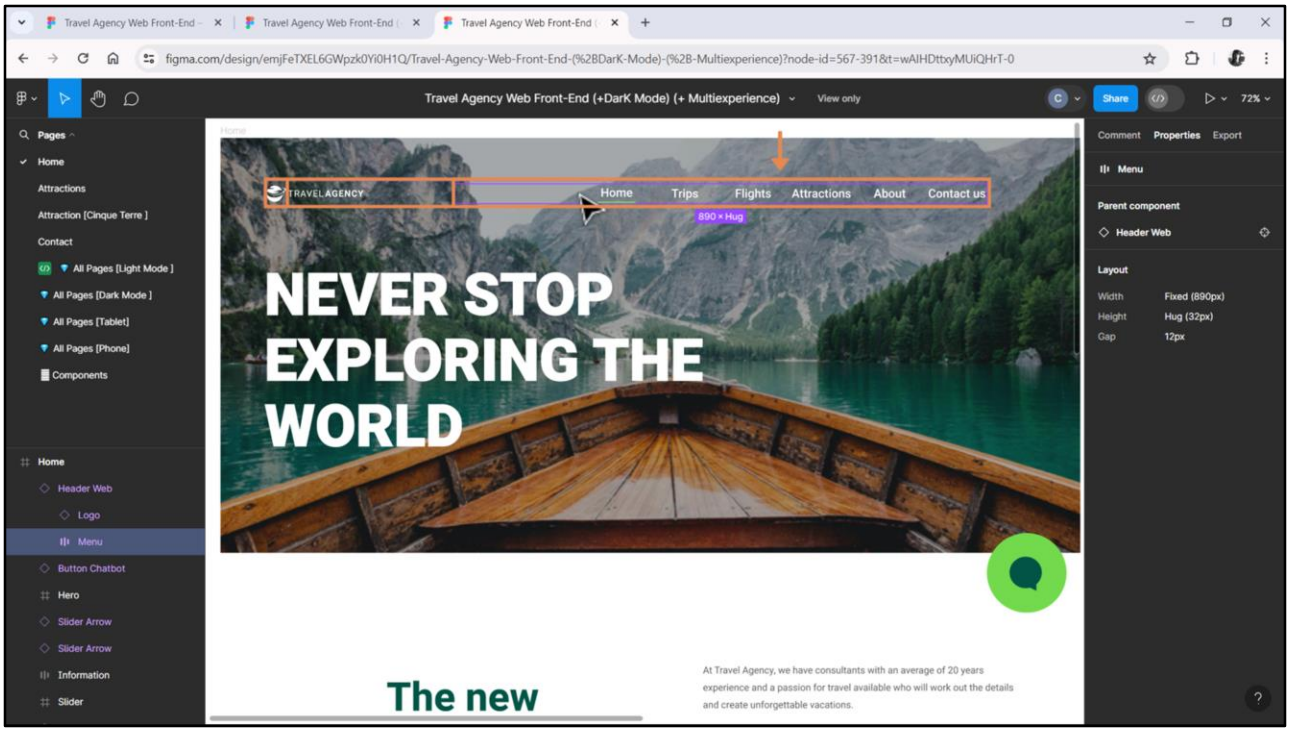
We can also see that Chechu grouped both components in this “Web Header”, also in a container with autolayout (we can see it clearly in this gap that appears here). And if we extract the CSS properties we finally verify it.

And here we see, graphically, that gap between the two elements: the menu and the logo.

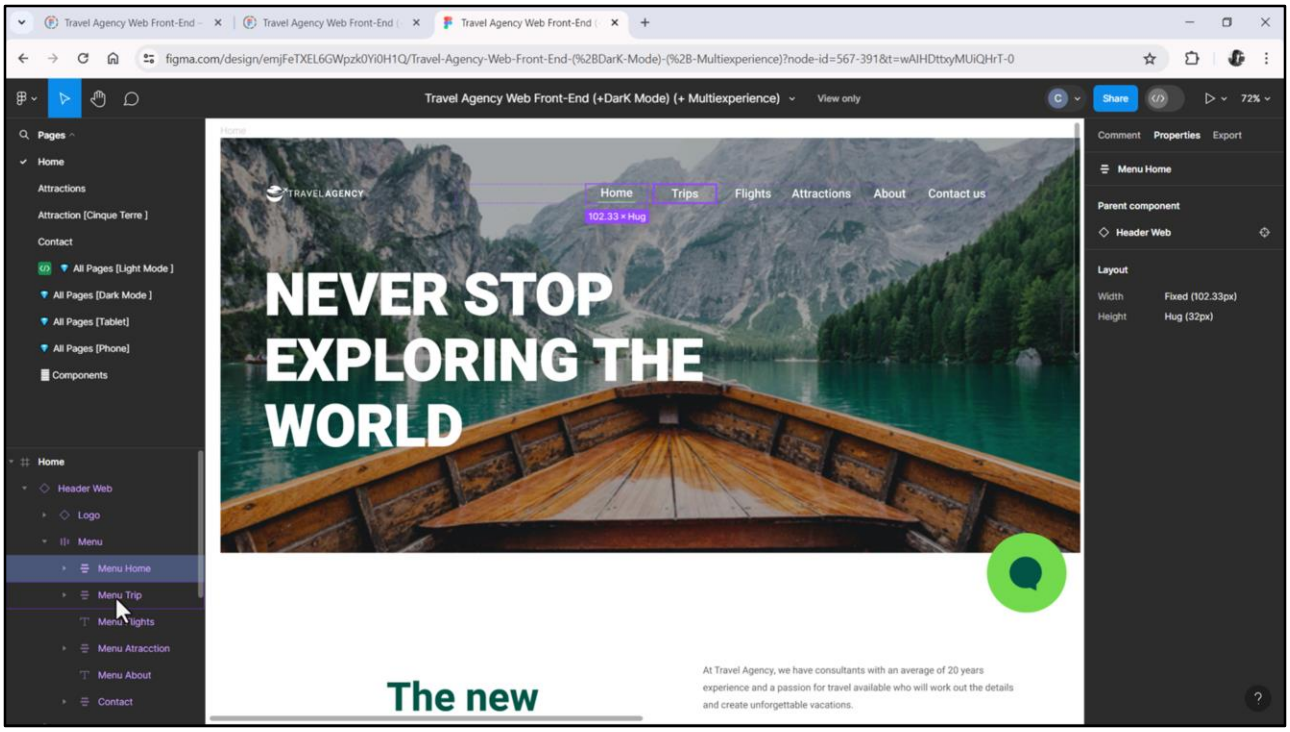


But we are not going to implement it as a flex, but as a table, because, as we can see, the logo is aligned on the left with this title. So it will be easier to use a table to align them.

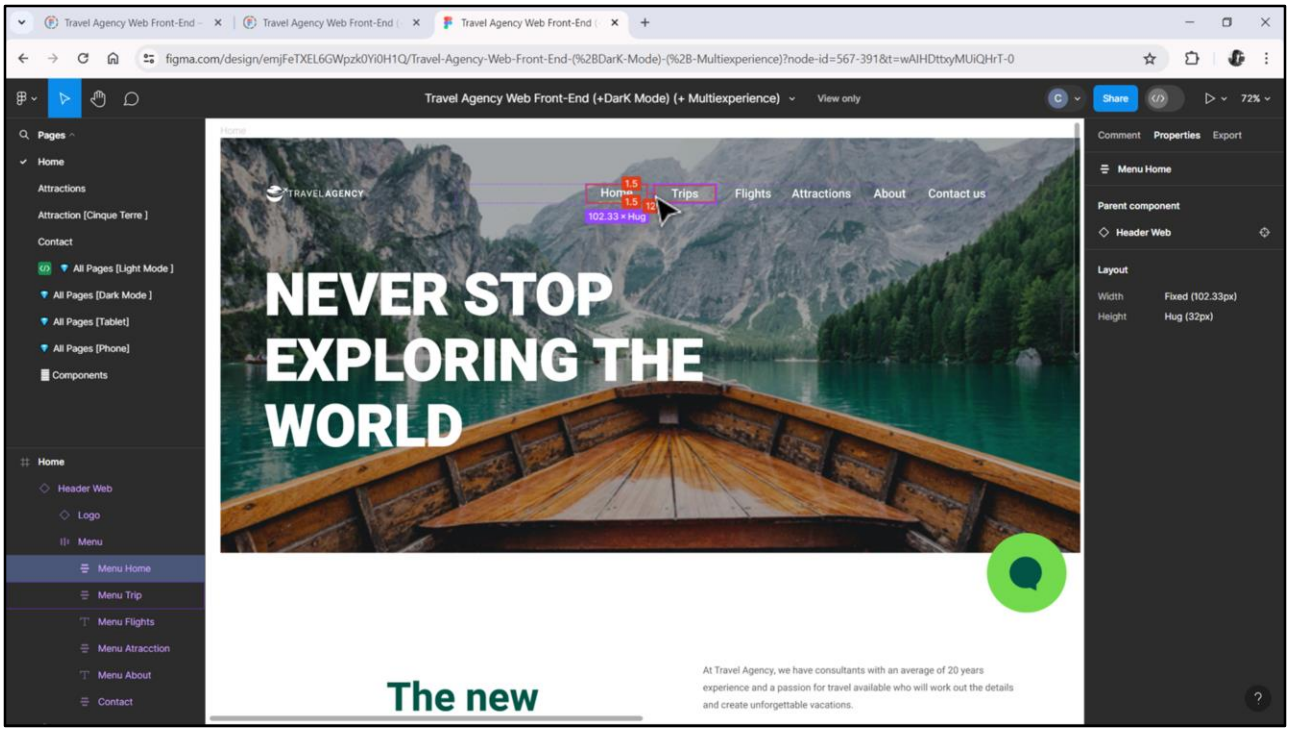
The table will have 3 rows, the middle one for spacing. Anyway, for the moment let's make it a single row, thinking only about the logo and menu. Later we will add the other two.



We can always imagine many ways to implement a layout. I'm going to choose to place the icon in one column, the words “Travel Agency” of the logo in another column, and a third column for the menu itself, which I will choose to implement just like Chechu did, with a flex.

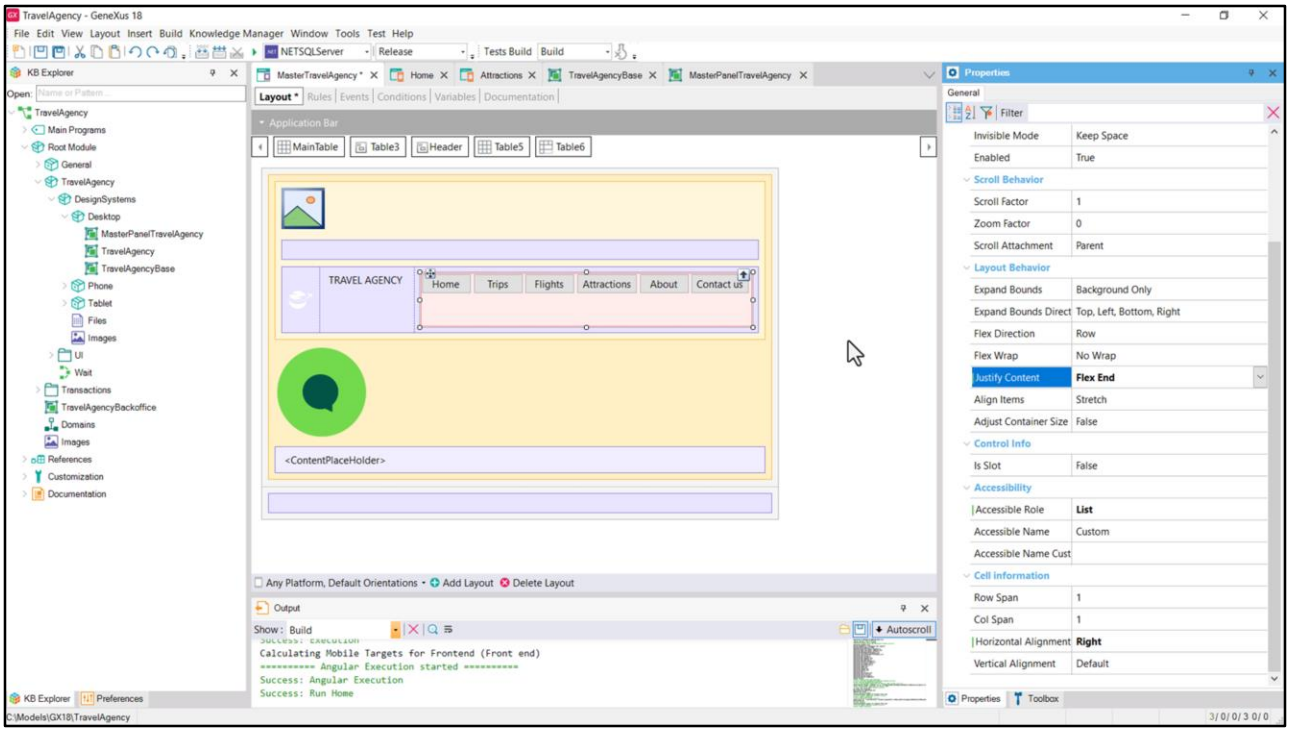


It will be a succession of buttons in horizontal direction: one for each action. They will be buttons and not textblocks because of what we said when we talked about accessibility. I prefer a flex in this case rather than a table, because, precisely, it gives me more flexibility: each button will occupy the space it needs, according to the number of letters in the text (it is particularly important if we want the application for several languages). And I can also distribute them evenly through the gap property applied to the flex class.



Here Chechu didn't realize (and I'm sure because I've just asked her) that what she did was to set fixed widths for the spaces of each text and that was what she distributed evenly, at 12 pixels from each other, but in this way, note that the space between the text Home and Trips is much bigger than between Attractions and About, for example. And this is a mistake.

In addition, if I want, in the event that the screen width is not enough to contain all the texts, instead of a horizontal scroll bar, to make a wrap and show some of the texts, of the options, let's say, of the menu, in a second line, then I can't get that with a table and I can get it with flex.



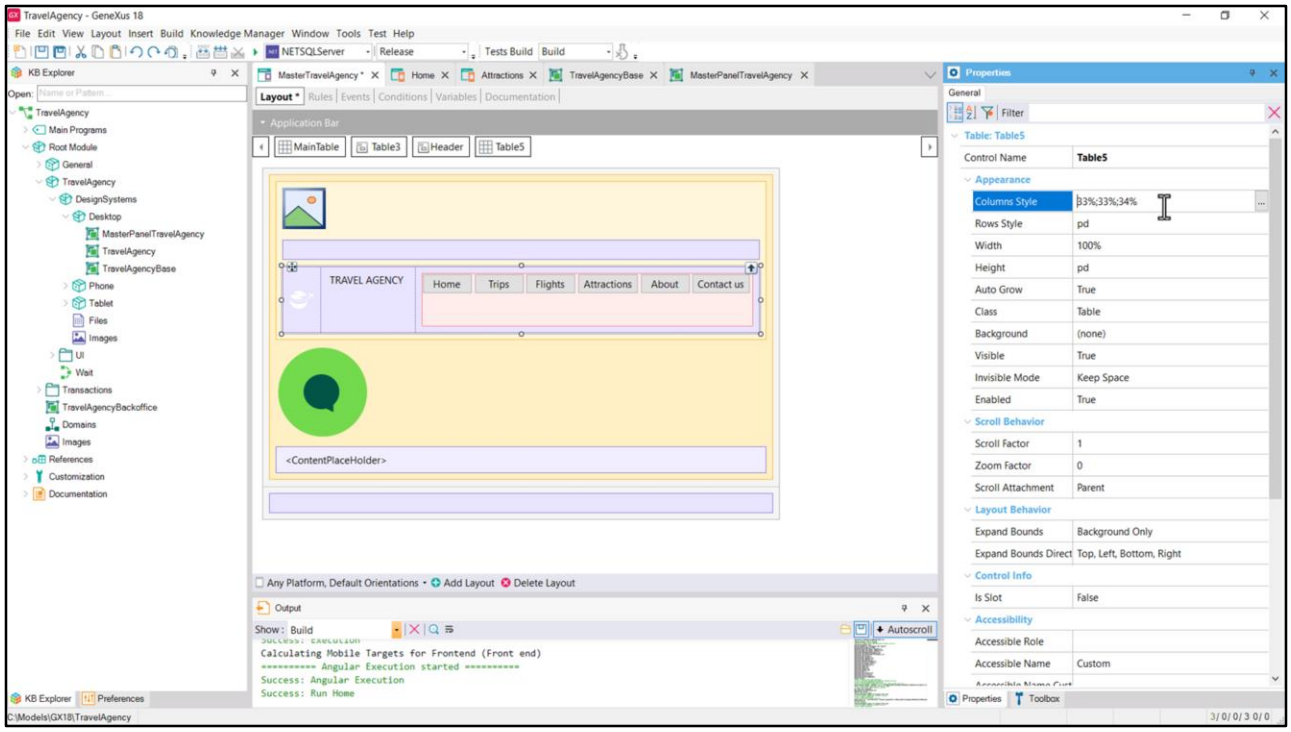
Let's start by inserting a table with 3 columns.

In the first one, we will place the Travel Agency icon. In the second one, we will place a textblock with the caption (for now) TRAVEL AGENCY. And in the third one we will place a Flex container.

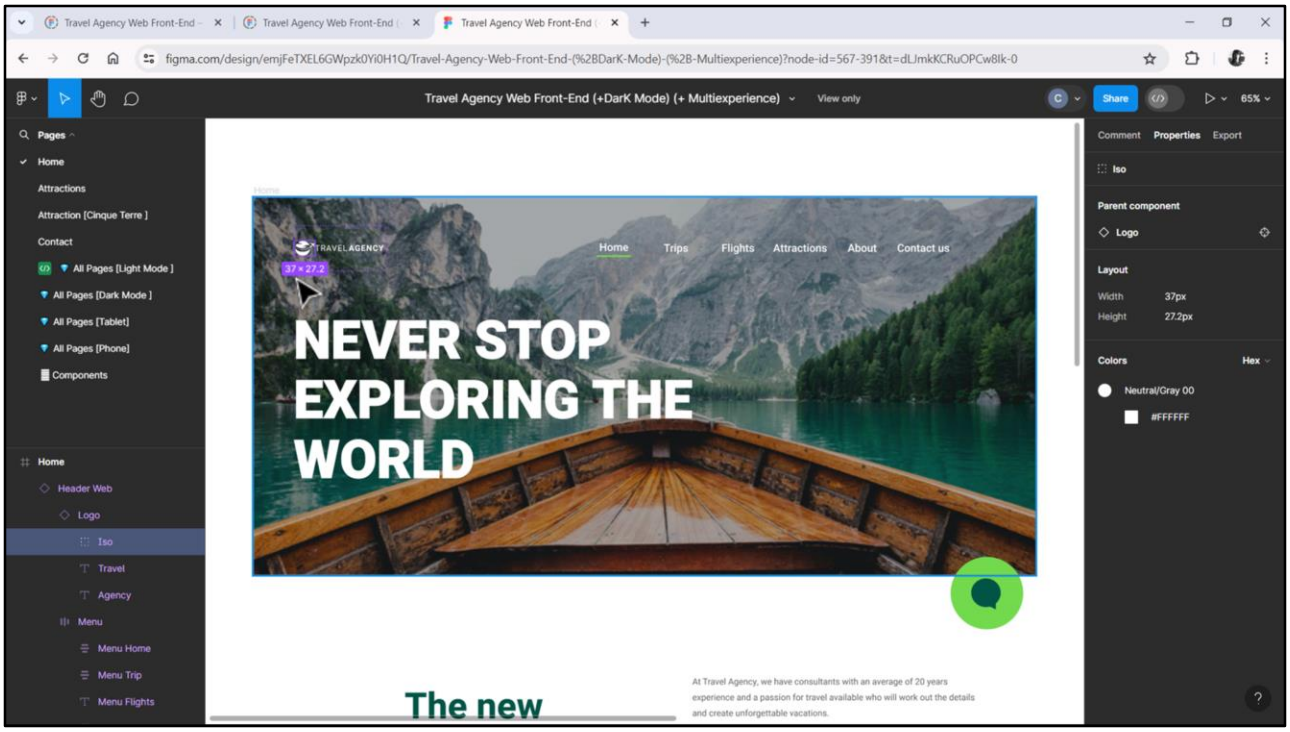
In this flex we will insert the 6 buttons: one for each action. To this flex container we are going to set the Accessible Role: List, to indicate precisely that it will be a list of items. We can already give it horizontal alignment on the right.

We can see that the direction of the Flex is the correct one, a row. In Wrap for now let's leave the default option that is not to wrap. Let's change the justification of the content to Flex End, so that all the buttons are justified in relation to the end of the flex and not the beginning, so they leave the free space in front and not at the end.

The alignment of the items regarding the other axis, y, we will want it centered, but for now I will leave the default; we will see that later.

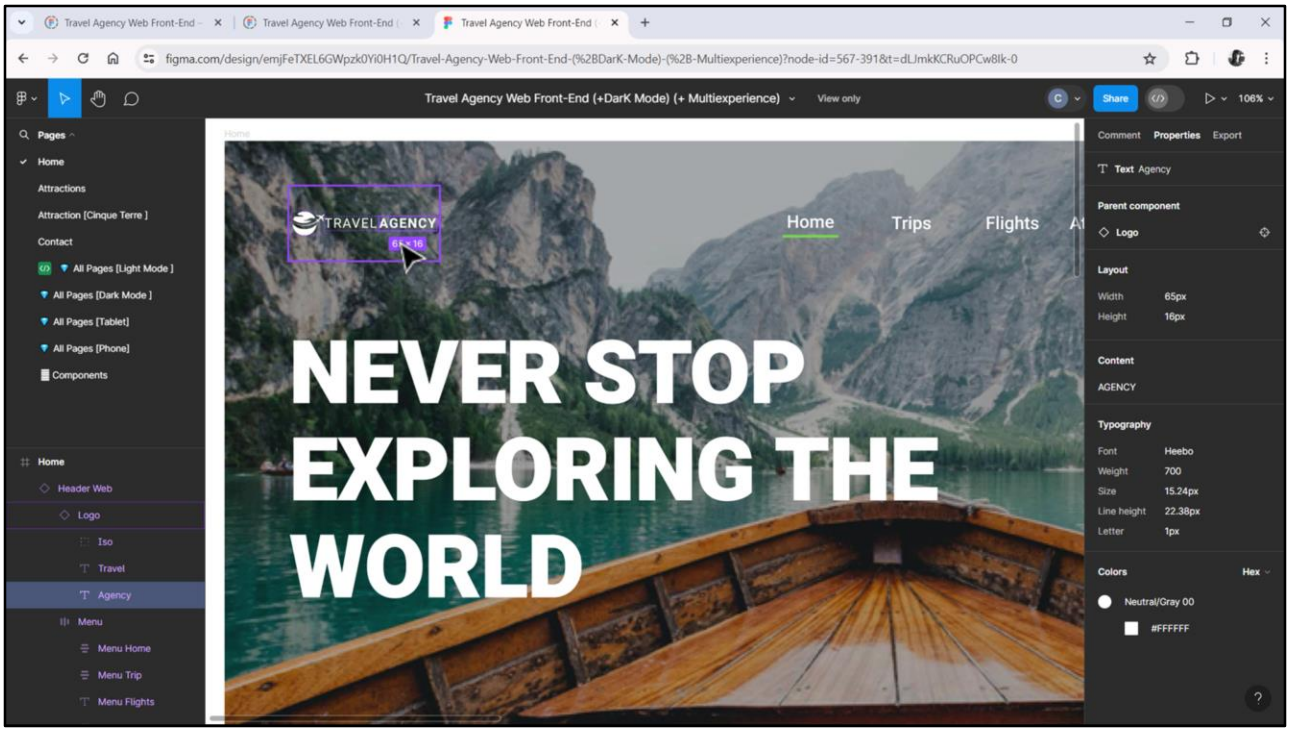


Now let's set the table dimensions: columns style, rows style, Height.

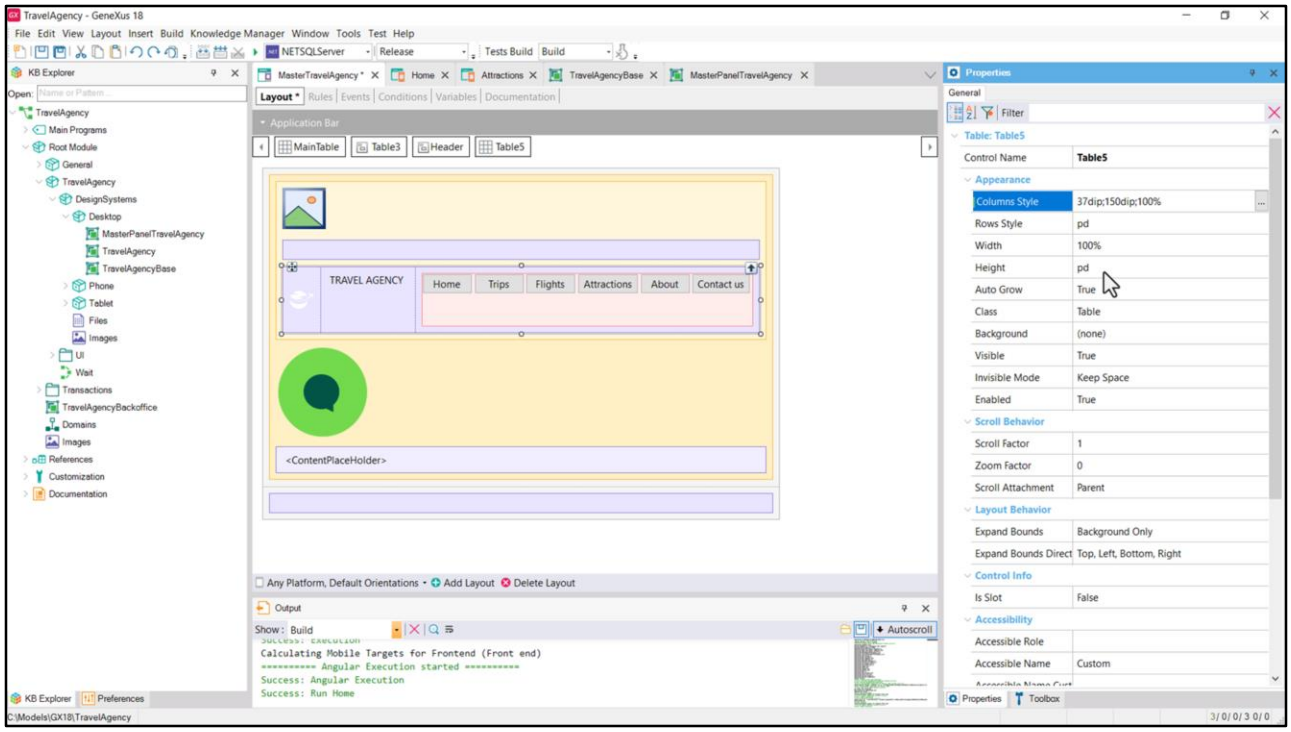


We could set the height of the table to 43 dips.

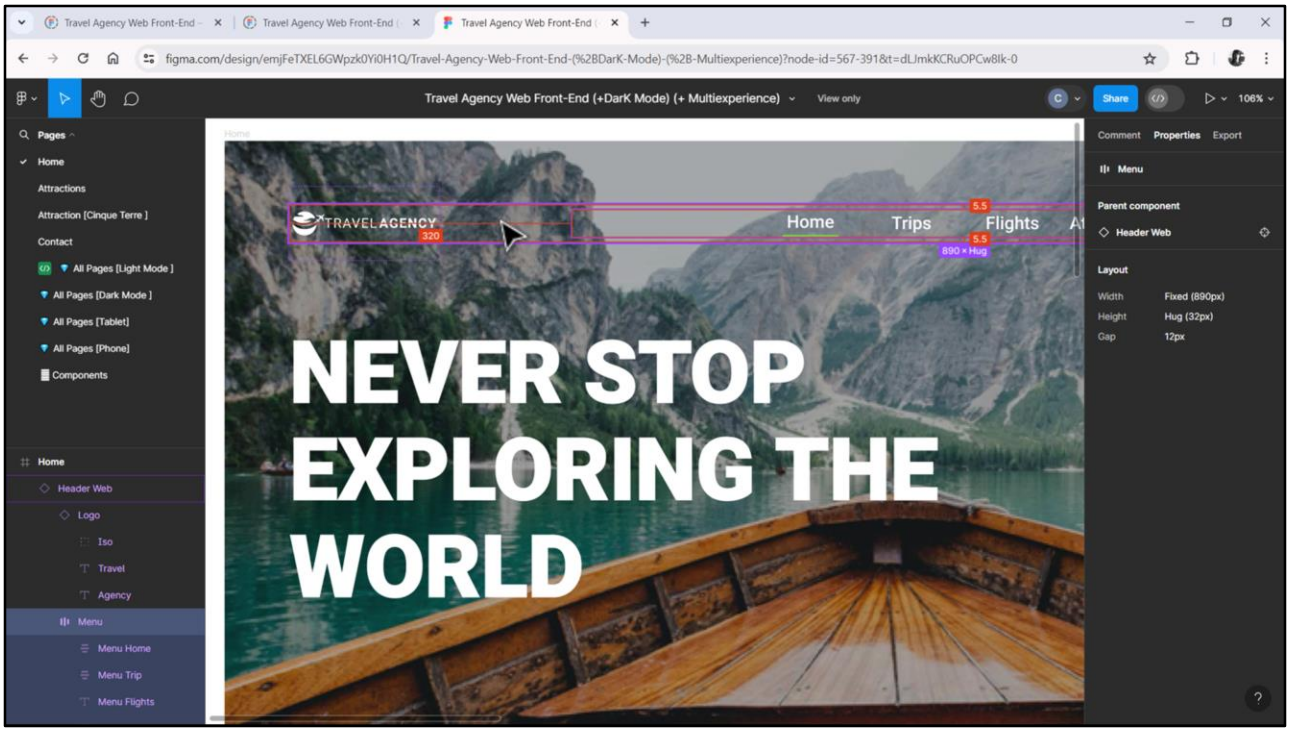
And then... the icon will be 37 dips wide, so we'll give that width to column 1.



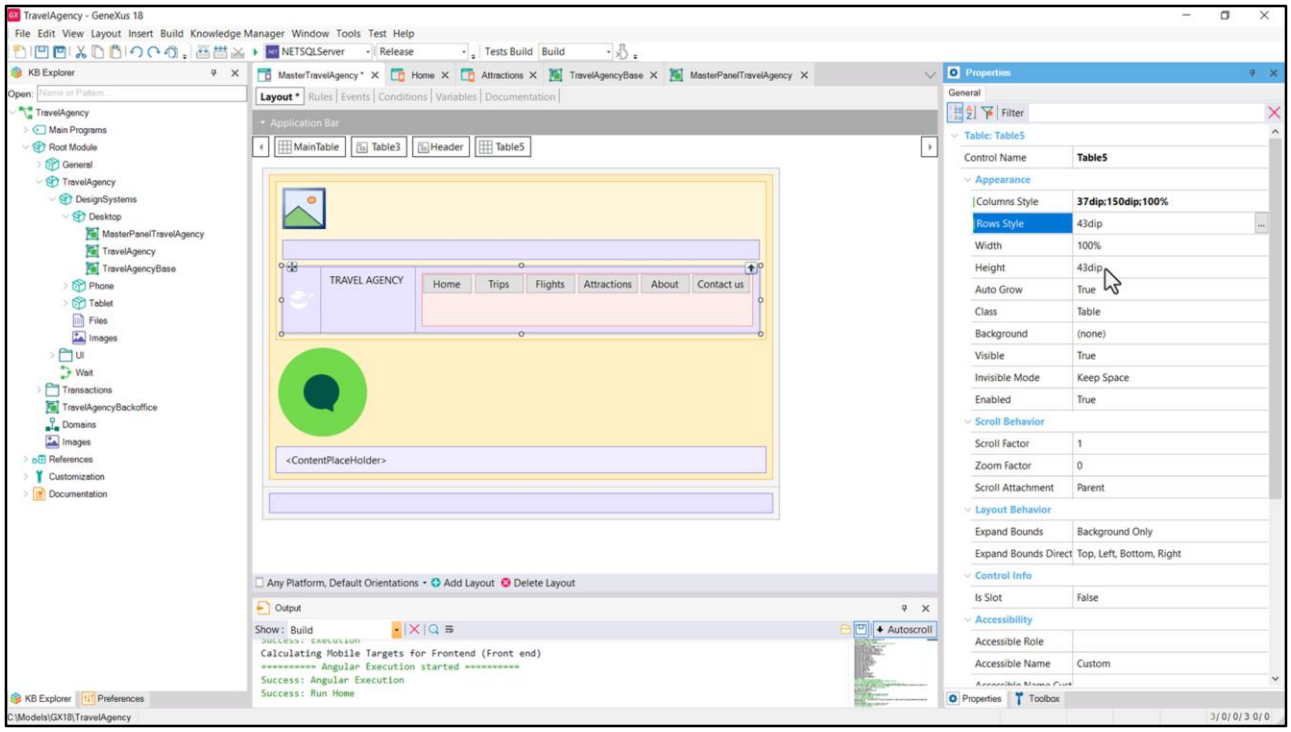
Then, pasted, without spaces, is the text Travel Agency, which adds 62 + 65, 127. So we could give column 2 a width of 150 dips, and column 3 100% of the remaining width...



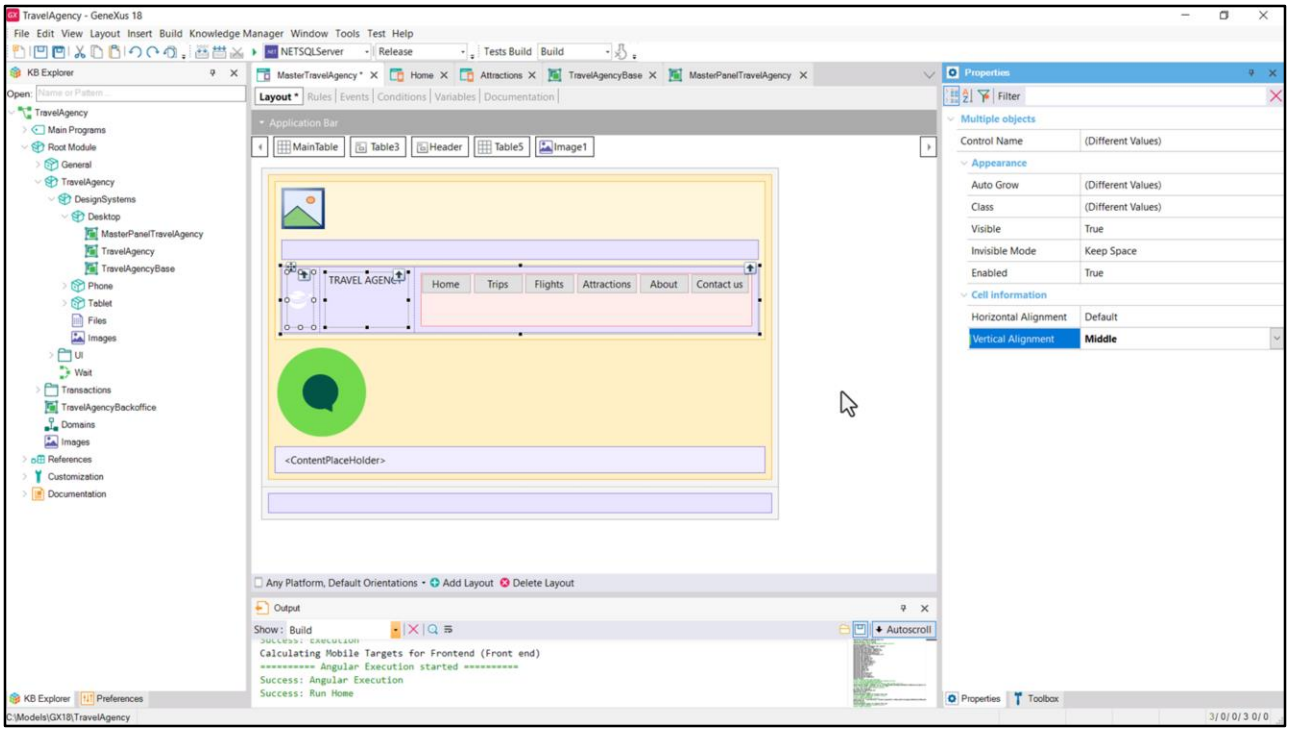
So... 37 dips, 150 dips, 100%... What about the row or table height?



It would be this one, of 43 dips. And we see that all the controls are vertically centered in relation to their edges.

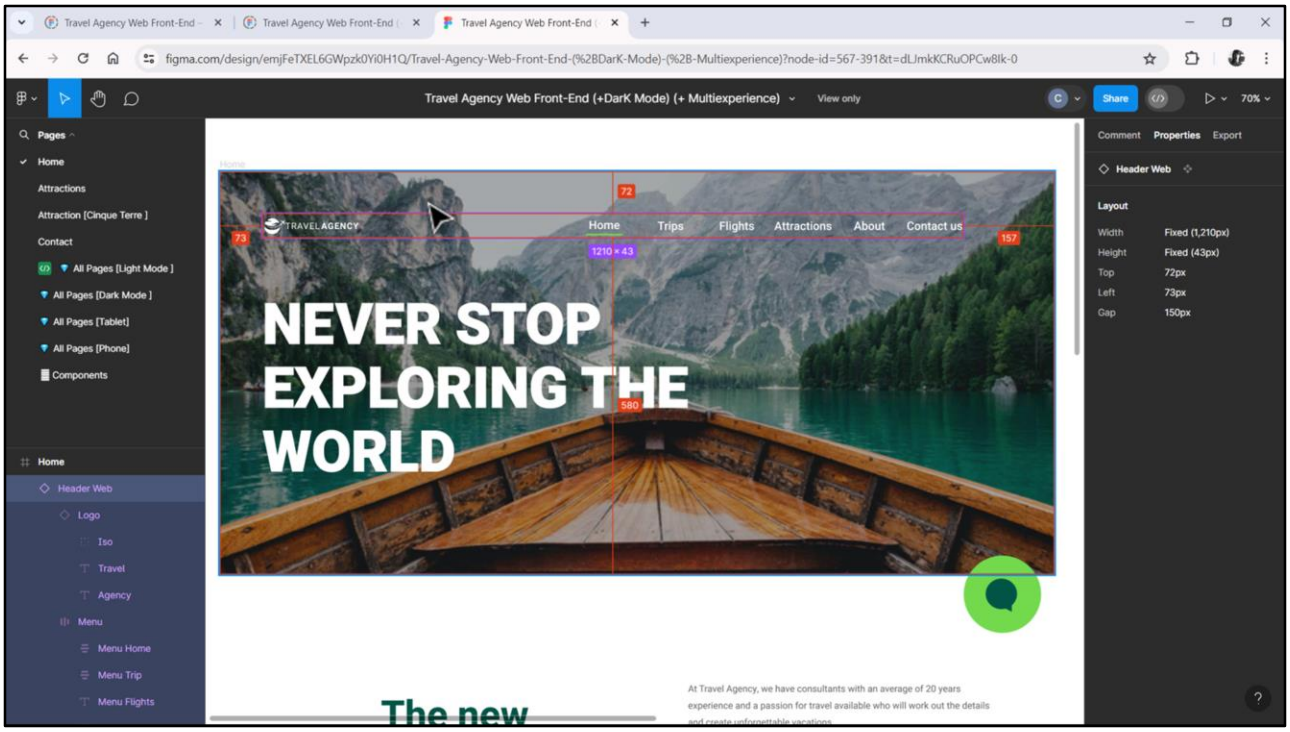


Ok, so I place the 43 dips here. Automatically the Height is going to be that same size...

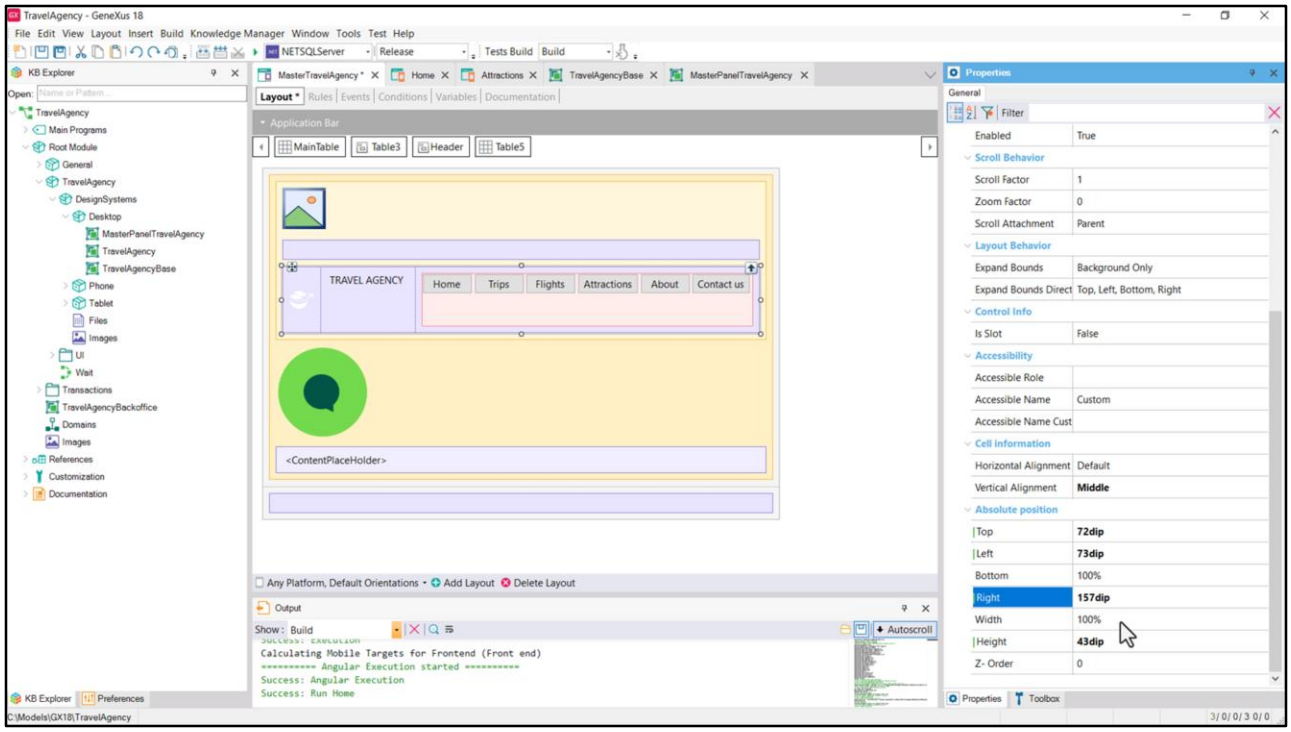


And then I select the three elements... here the flex... and I set the Vertical Alignment property for the three to Middle.

Now, where is this table located inside the canvas that contains it? We have to provide its absolute position.

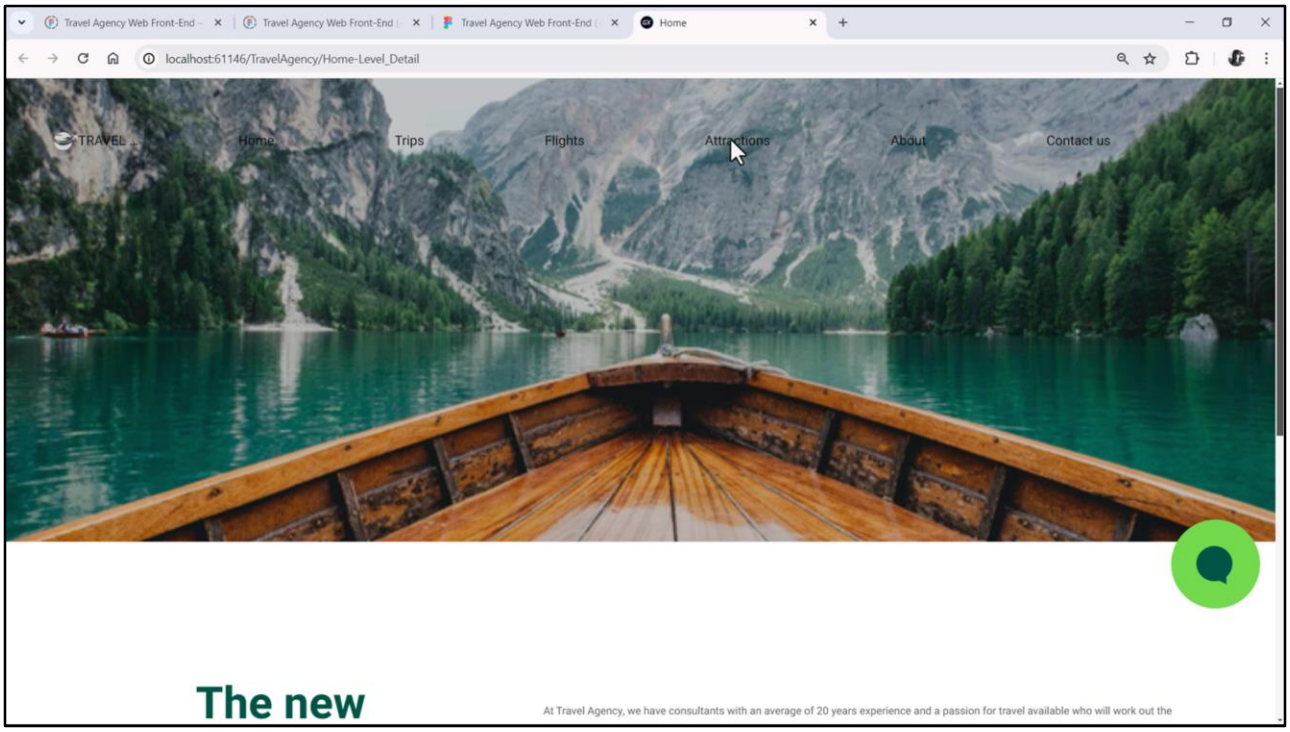


And it is from the top 72 dips, from the left 73 and from the right 157. This is enough. From the bottom it will be the remaining 100% of the canvas height.

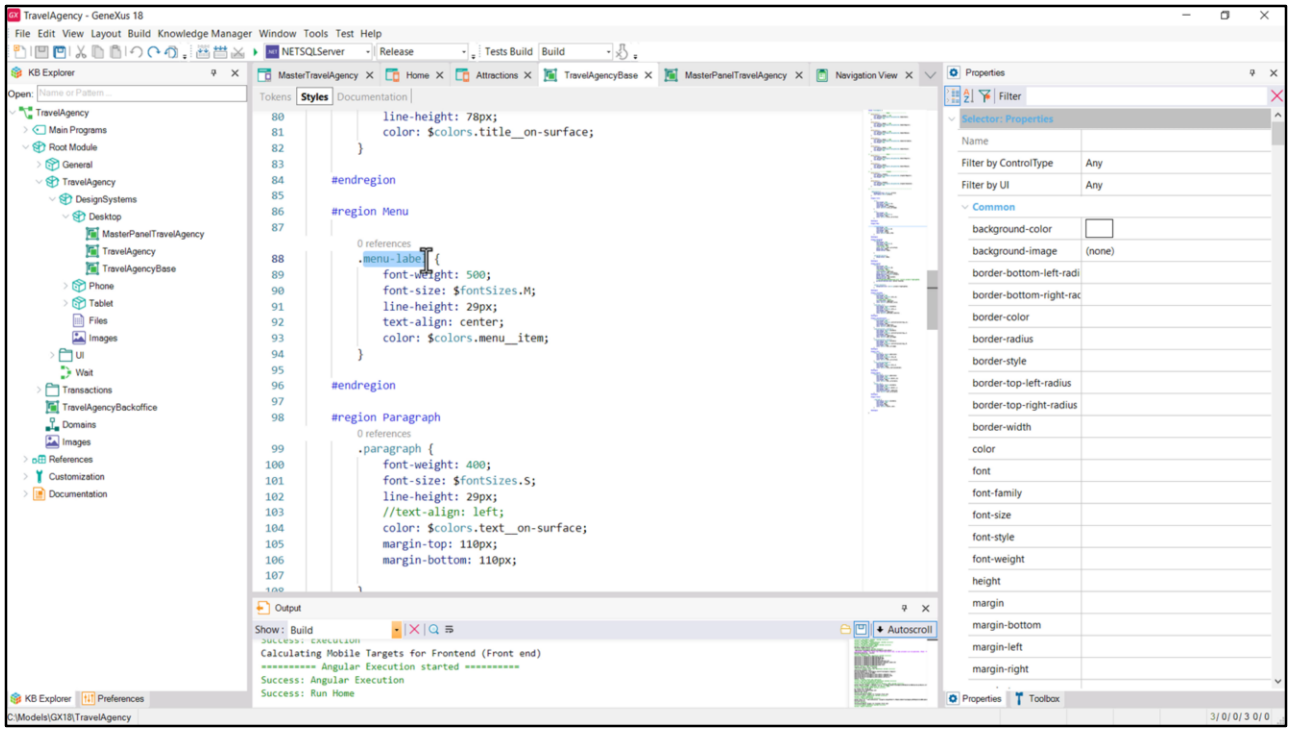


So... top is 72 dips, the height of the table was 43 dips, and so the bottom is the remaining 100%.
Left 73 and Right 157.

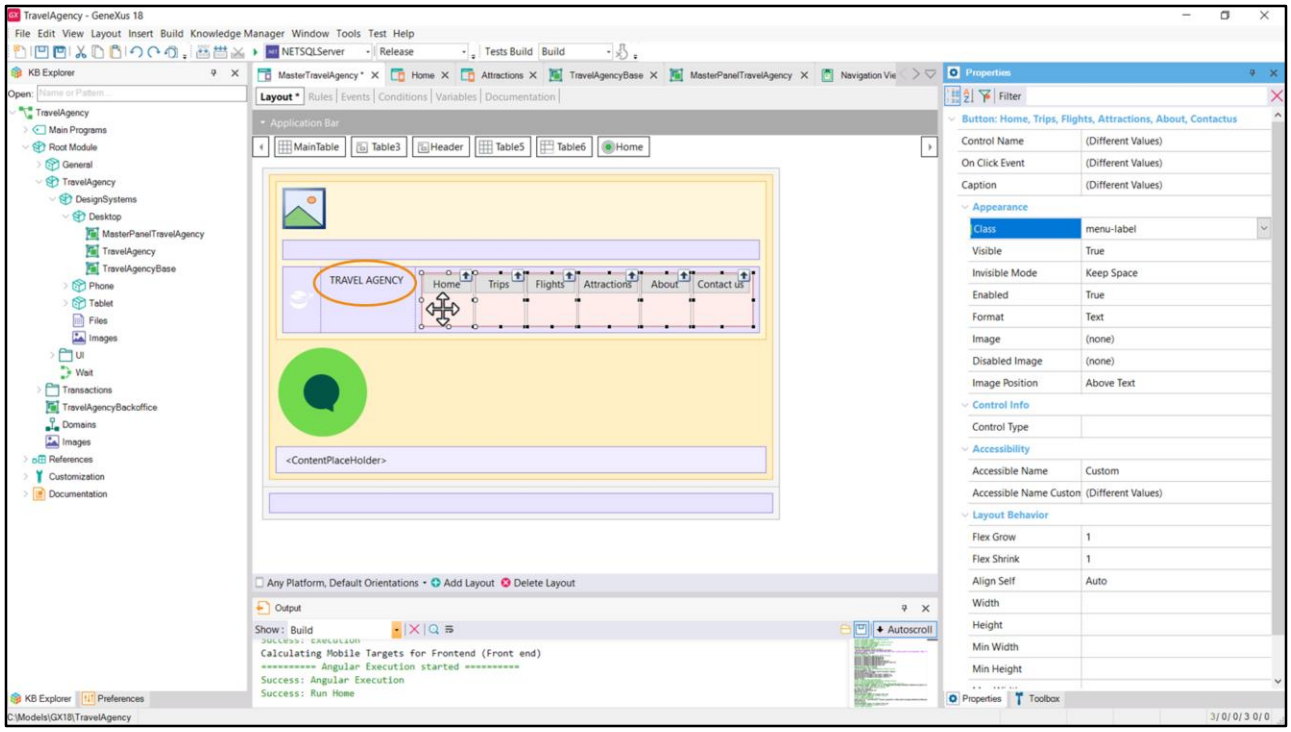
Let's try what we have done.



Well, we have a few things to work on, don't we? To begin with, the typography styles of the texts...

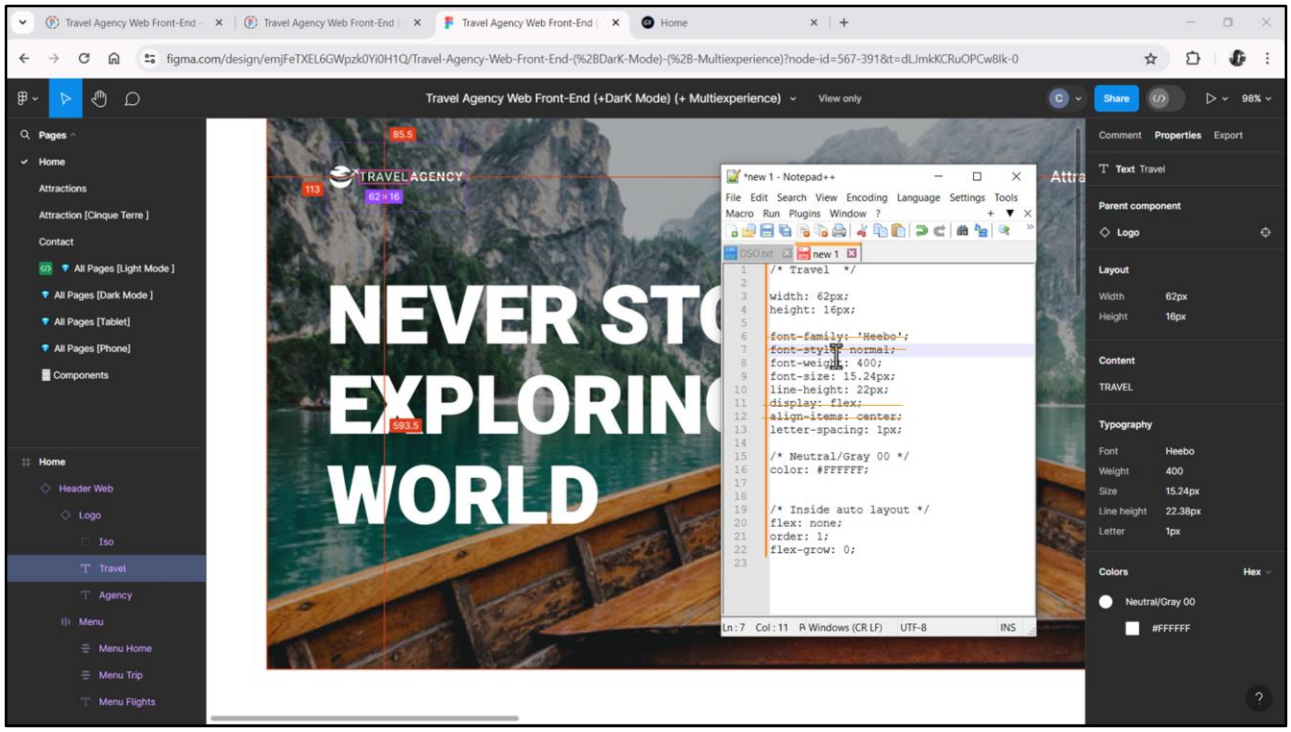


In the base DSO we had the style of the menu labels…



So let's begin by assigning this class to all the buttons of the menu. We select all of them, and we change the class for this... we see that indeed it was changed in all of them.

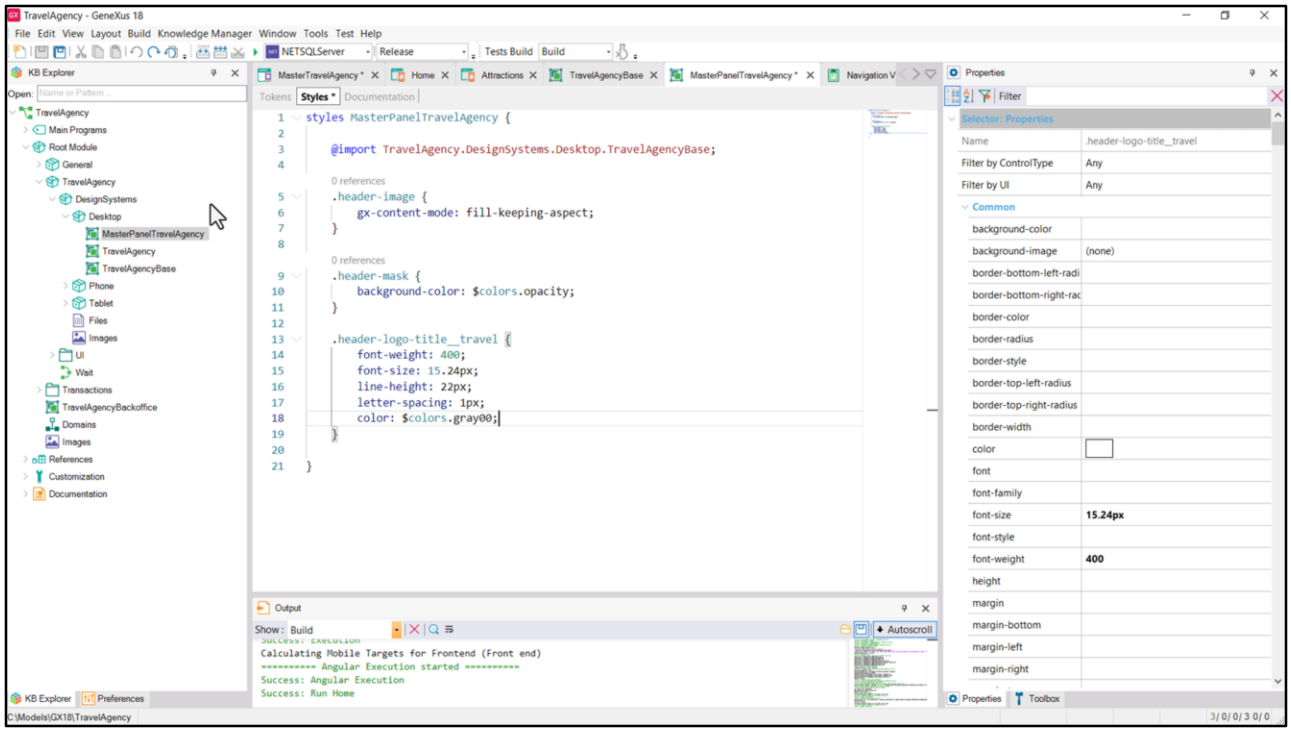
We had not defined a class for the typography style of these texts...



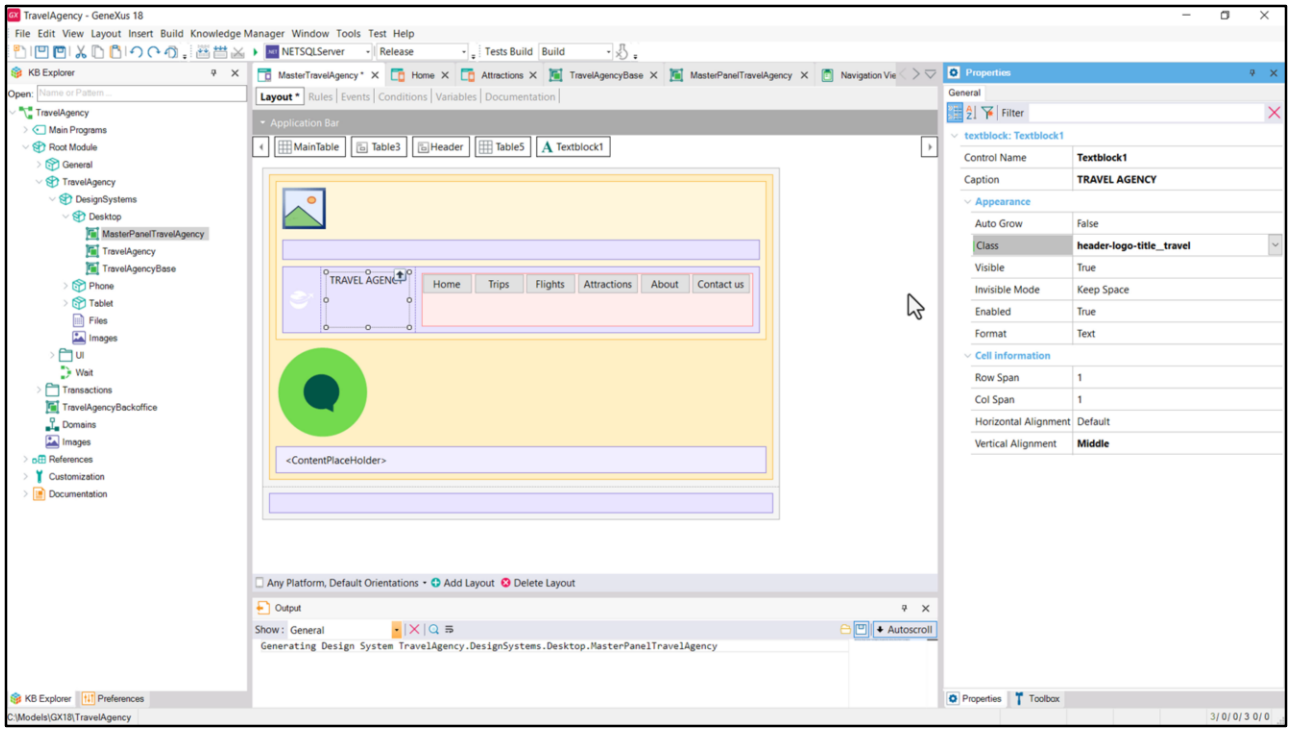
We see that Chechu hadn't created a style... and that the only difference between one text and the other is the font weight. One is 400, the other is 700.

Well, let's copy the CSS because we will have to define a class for these texts. I remove these properties that will be those of the default font, so I'm not going to need them... I remove these two that are from the element in Chechu's flex, and the one that I will need to add besides these 4 will be the color of our gray00 token.

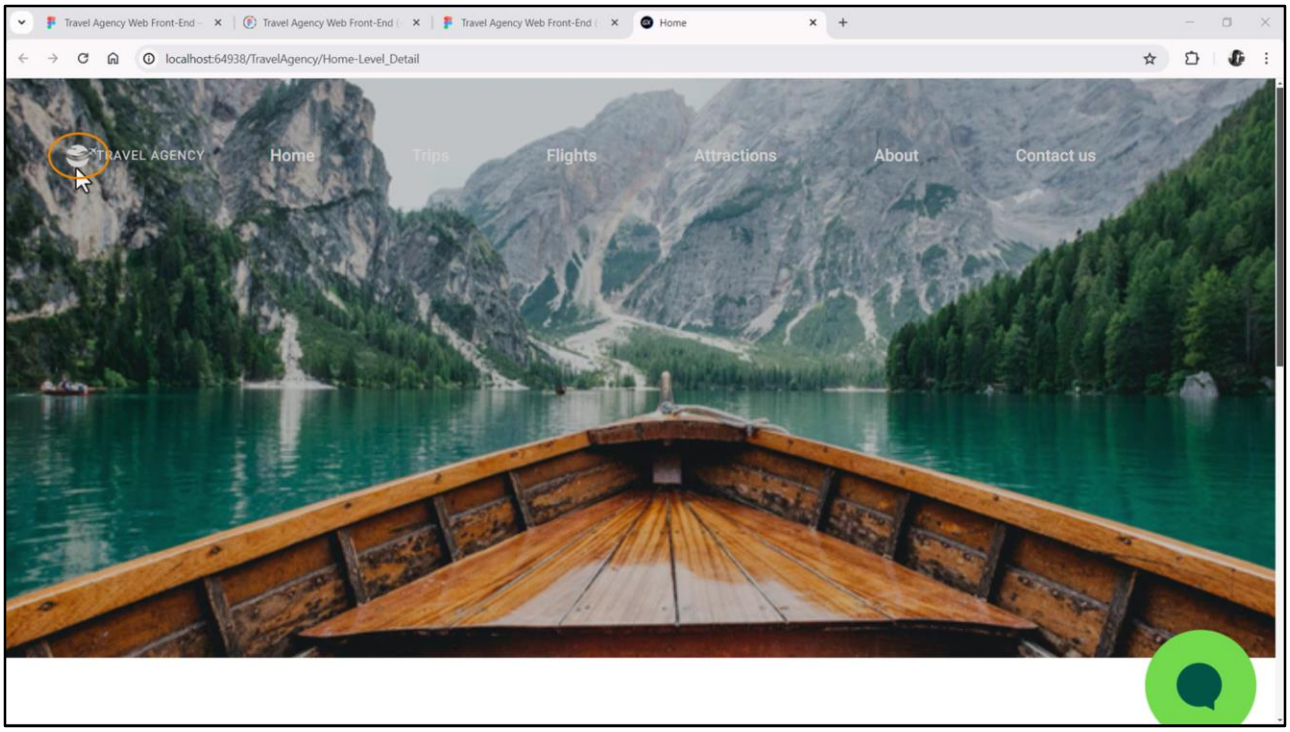
I copy these 4...



And in the DSO of the Master Panel I add a class that I will call...like this. Later we will see why the name contains “travel”...
I paste the properties... I add the color one...



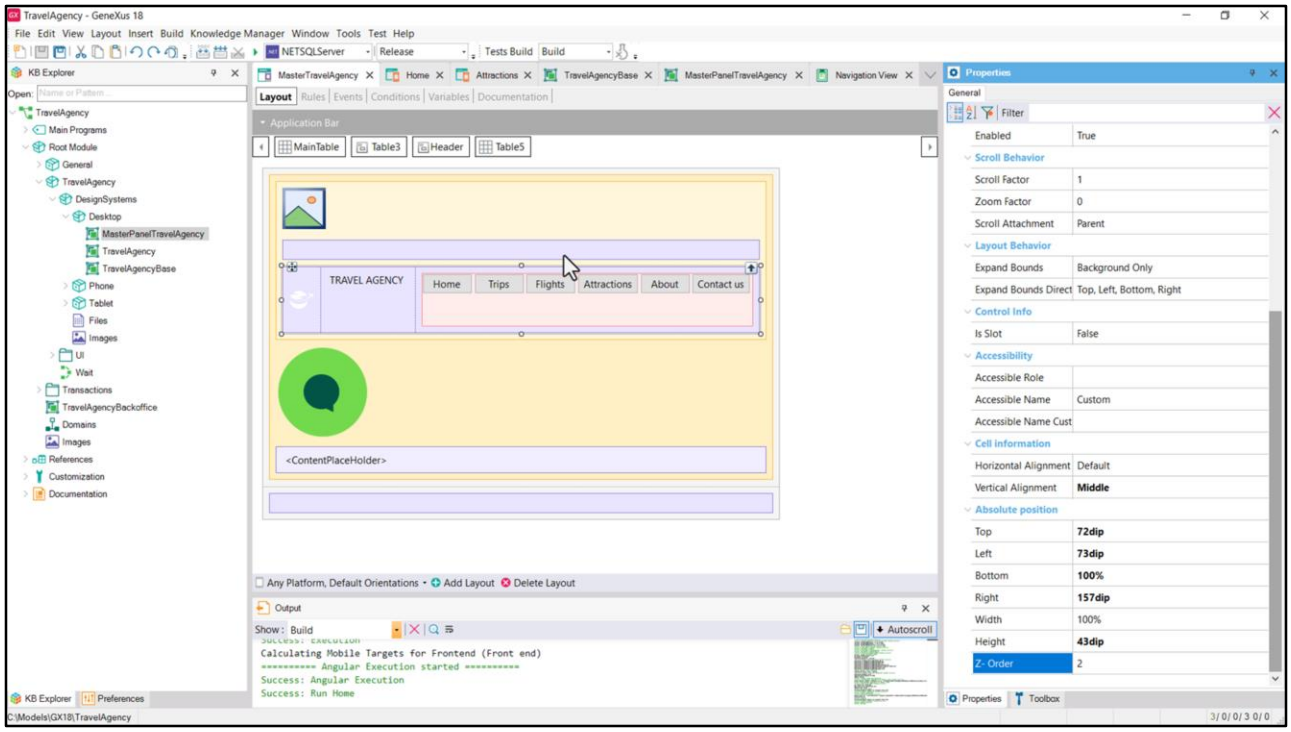
...And I associate the class with the textblock.



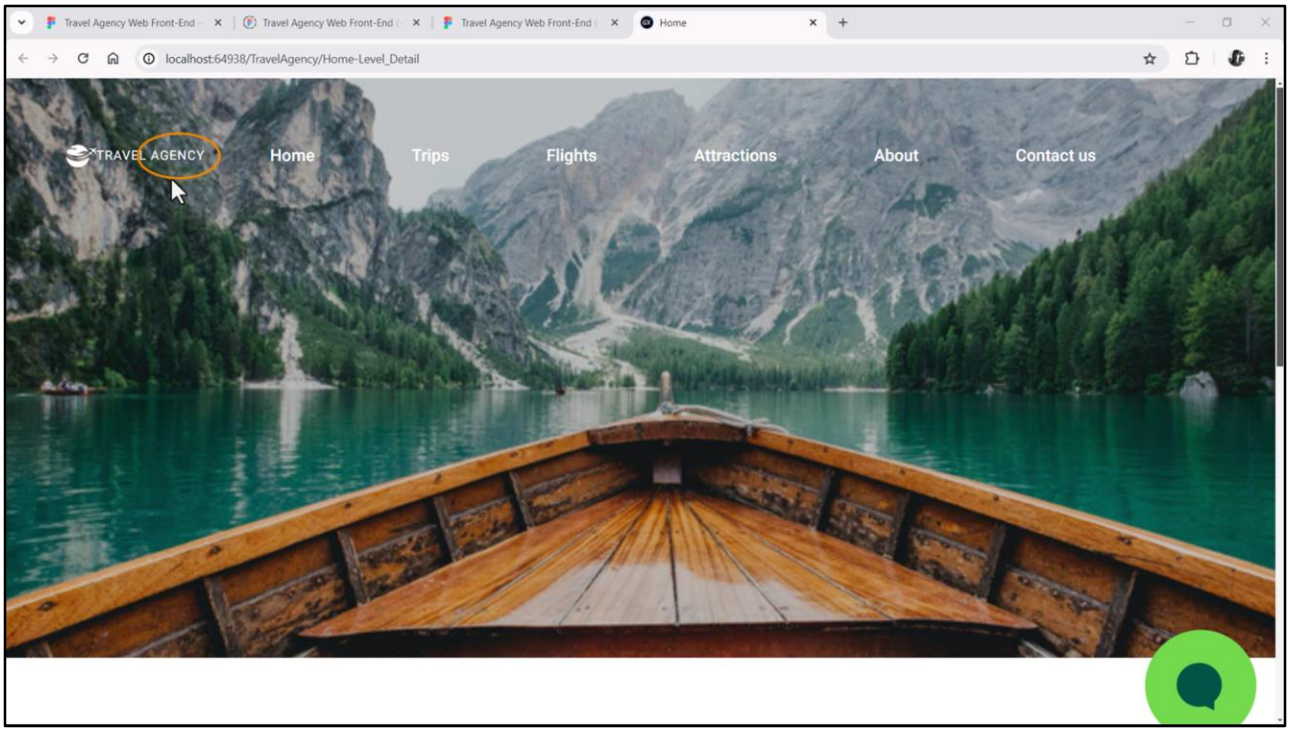
Let's give it a try...

If this icon does not appear, you can do a “Rebuild all” and try again. Or reinsert the icon in the KB. When we moved the icons module in the Assets video, I don't know if you remember, they may have been internally misplaced there. It's fixed as I'm telling you.

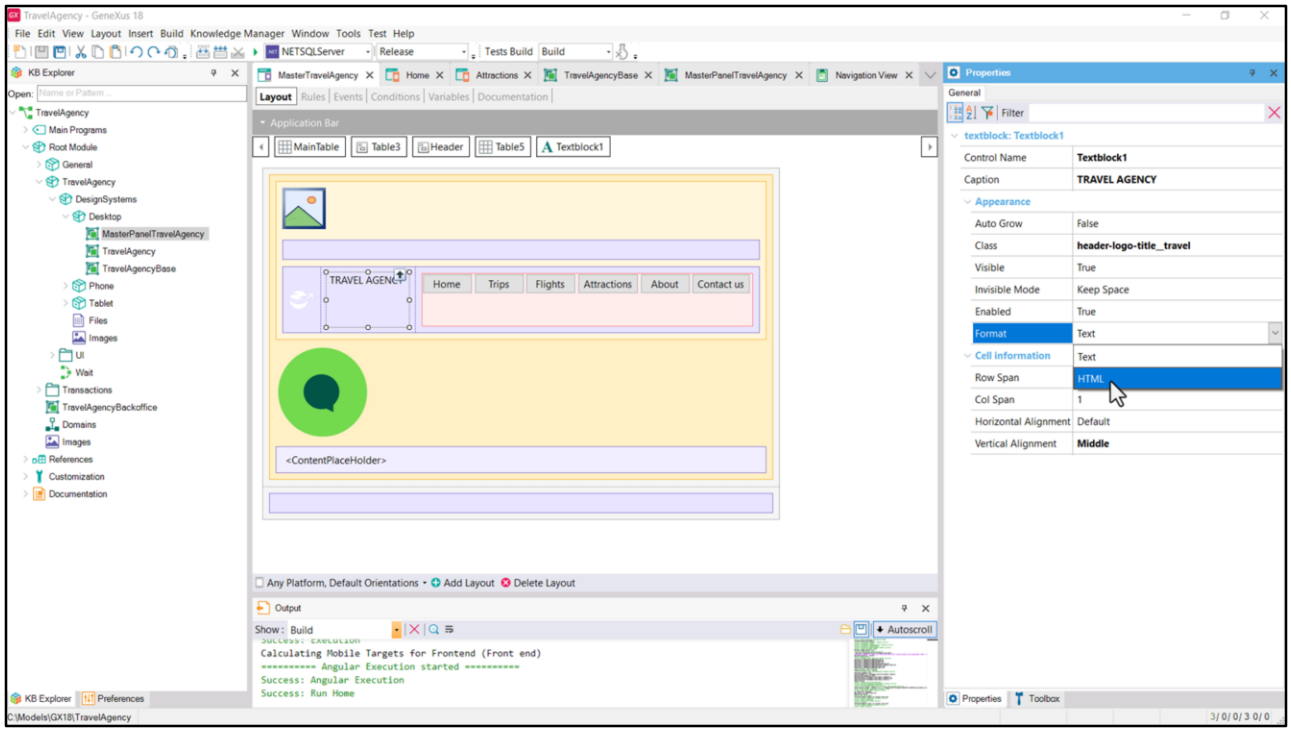
Something strange is happening... These texts are looking gray and not white... The reason is that they are below the mask.



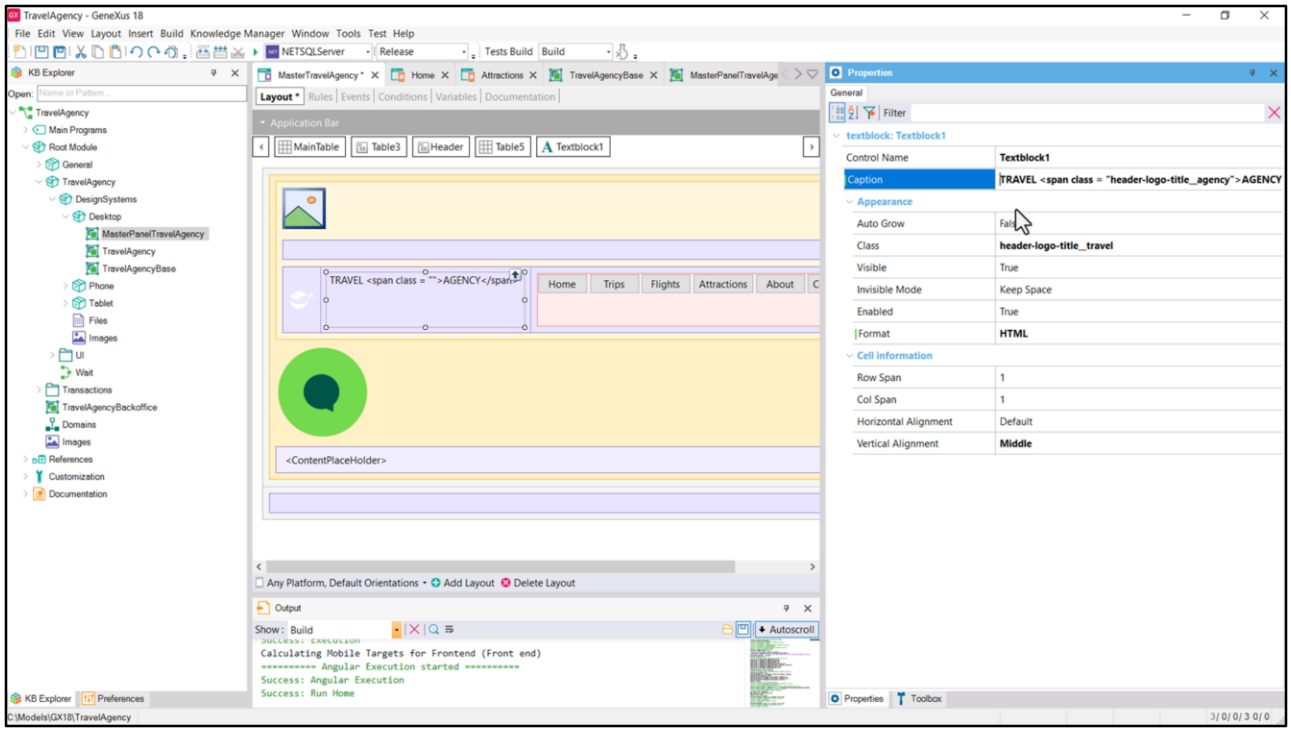
I had left the default Z-order for this table, 0, and the mask was set to 1. I am going to set it to 2, so that it is above. Let's try it now...



Good. Now we have to work on the menu that is not right, but we also need to change the font weight of this text, so that it looks 700 and not 400. Let's start there, and then we can devote the rest of the time, more calmly, to the menu itself.

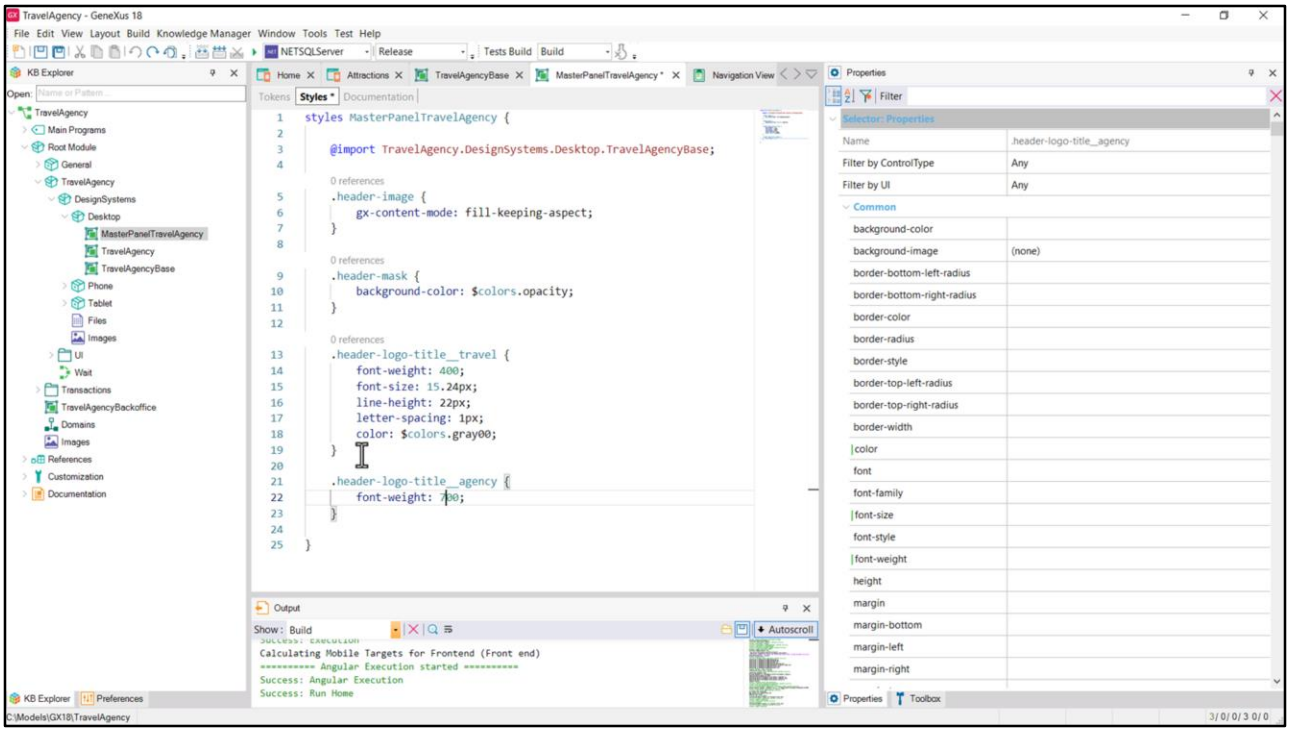


As we are in the Web application, we can change the format of the text to HTML, so that the Caption can be interpreted as HTML. This will allow us to assign another class to the text AGENCY through the HTML span tag.

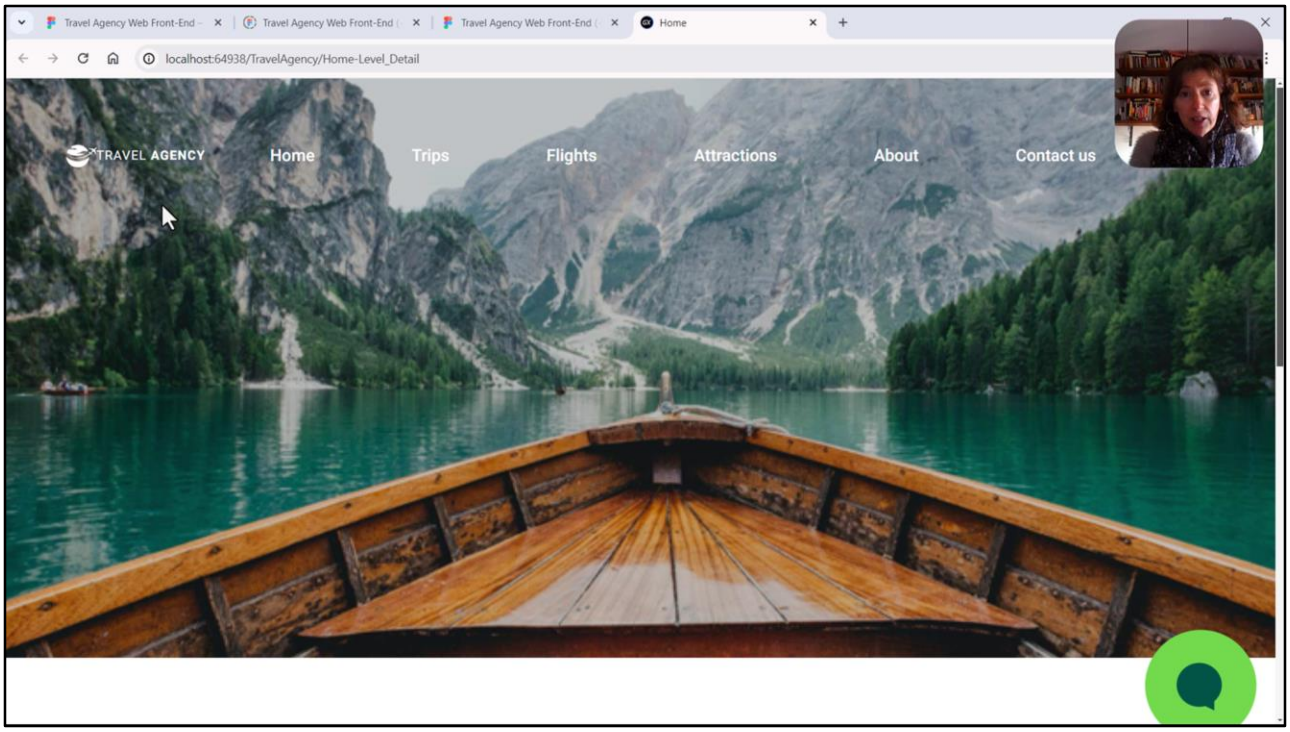


We have the class of the textblock, “header-logo-title__travel”, and we will assign to AGENCY the class that we will create, “header-logo-title__agency”, to be able to change the weight of the font to the word AGENCY. That's all we want to do. We want everything else to take the properties of “header-logo-title-travel”.

Then we copy this text...



...We come to DSO, and we are going to specify this class. The only thing we're going to do is change the font weight: from 400 to 700.



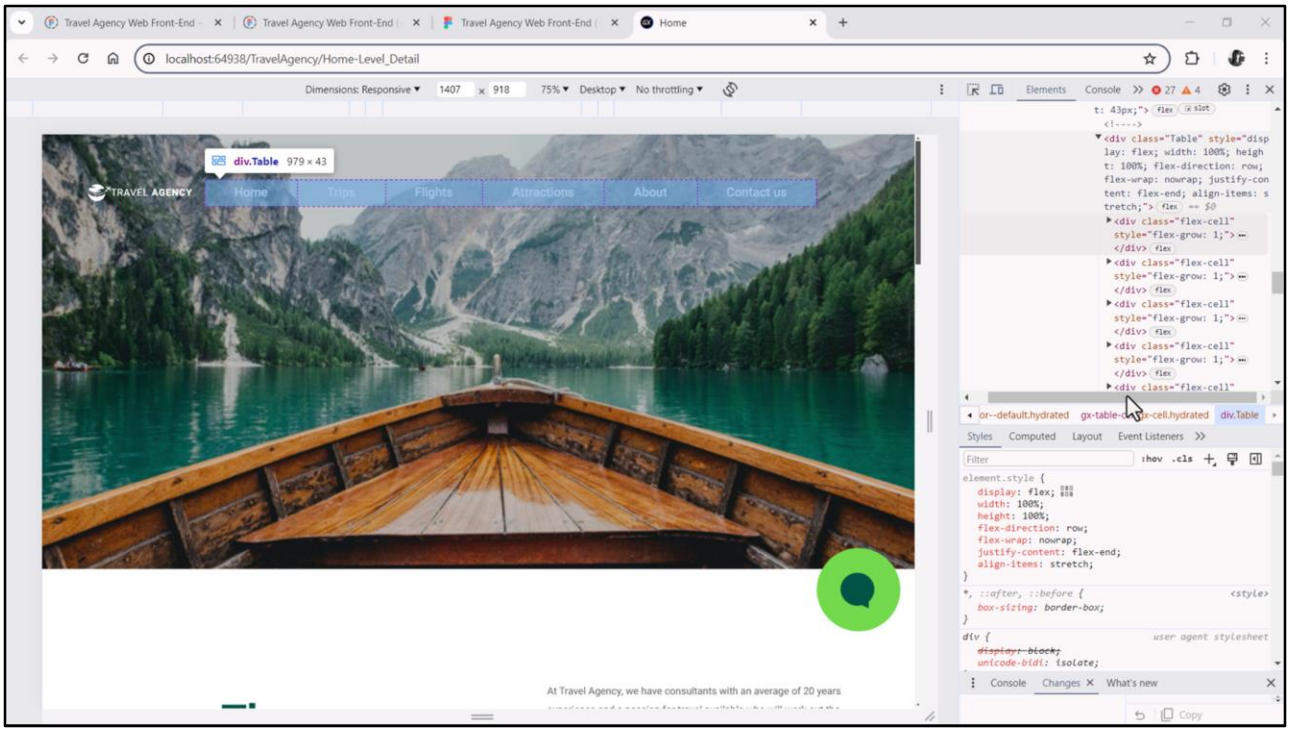
And here we can see it.

What we have just seen doesn't mean that Angular will allow placing any HTML in the textblocks and in the editable fields with HTML formatting. Actually, it will remove any definition of a style attribute that is placed there. But what it does allow is what we did, using DSO classes inside that HTML.

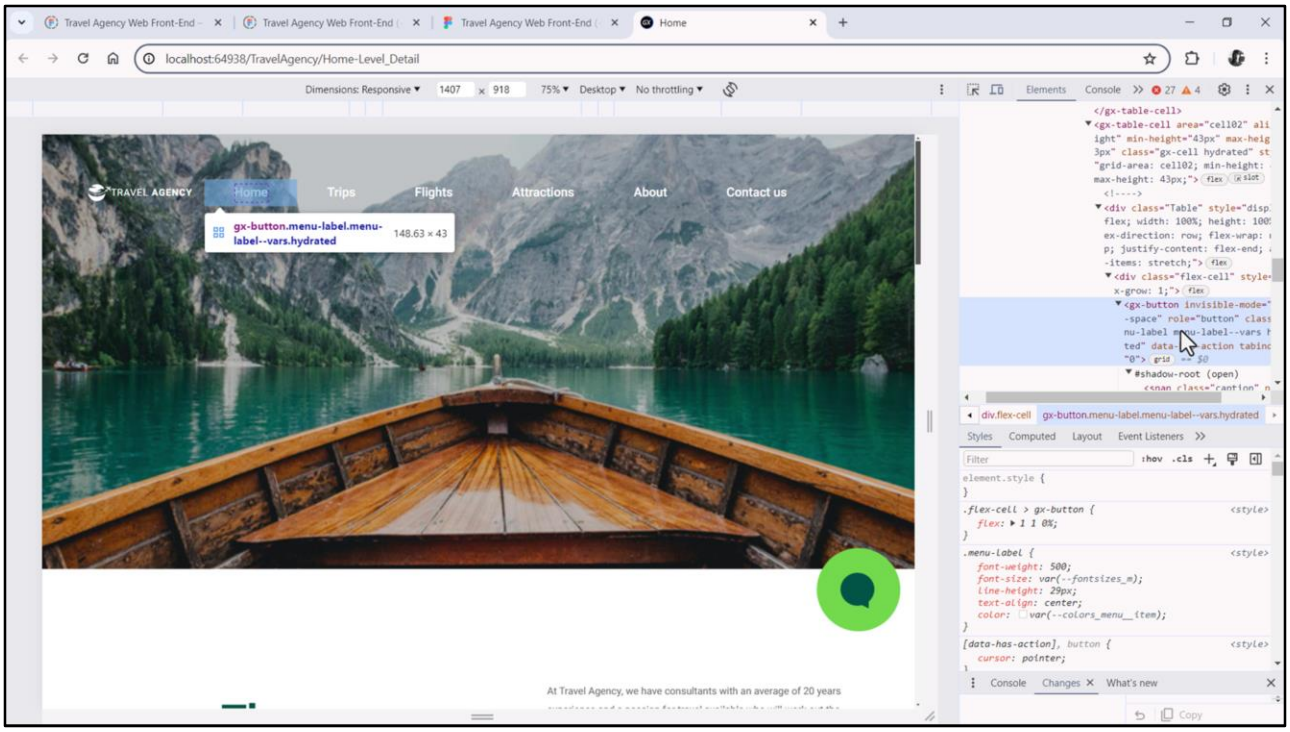
Quite the opposite of what happens with Android or Apple: they don't allow DSO classes to be used within the HTML and they don't remove the style attribute.

So, in principle, for the solution to work for both cases, that is to say for Angular and also for Android and Apple, we would need to add a style attribute where we set the properties of the class that, in our case, we have it in the DSO. In this way, when running Angular, it will remove everything that has to do with the style attribute, it will not take it into account... but it will take into account the class. And in the opposite case, if we execute for Apple or for Android, it will eliminate, disregard the class and it will take into account the style attribute.

Since for desktop size we will only have the Angular application, in this case we don't have to do anything.



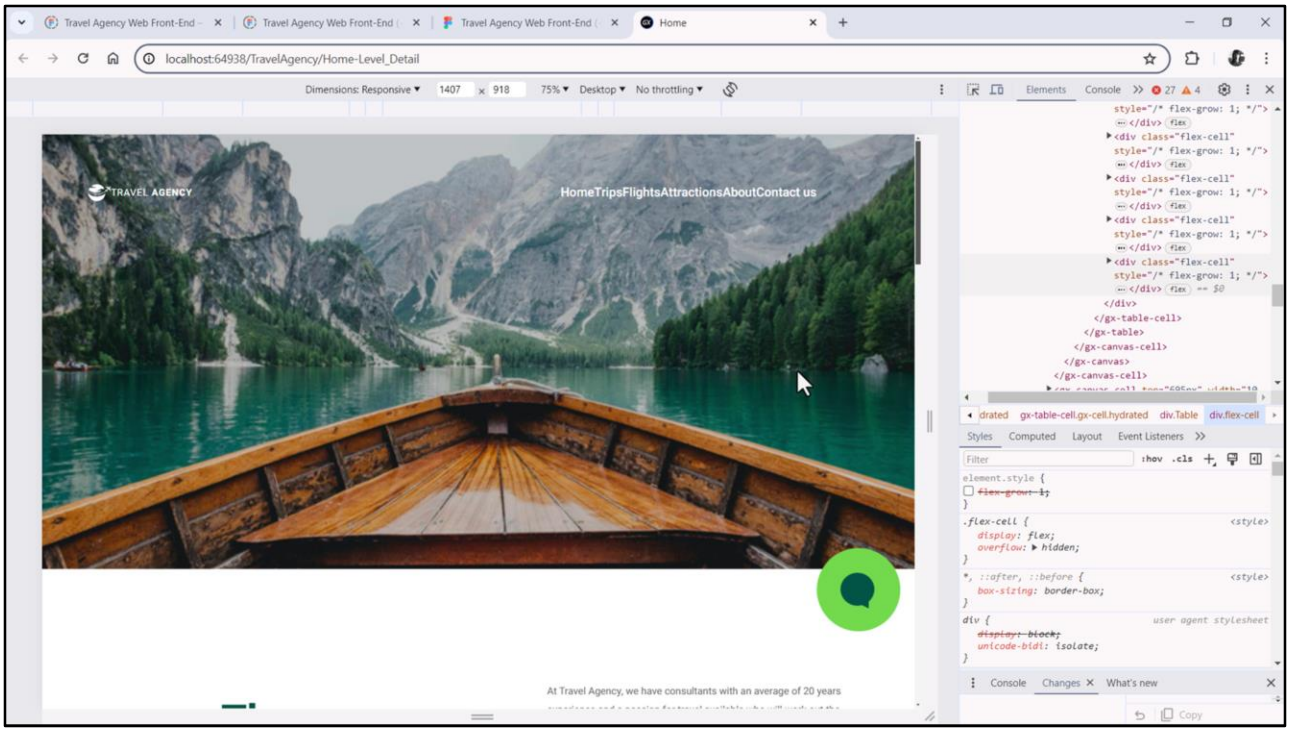
And now, to the menu! If we inspect it as we have it so far, here we see the table that corresponds to the Flex, and that it has a cell for each button.



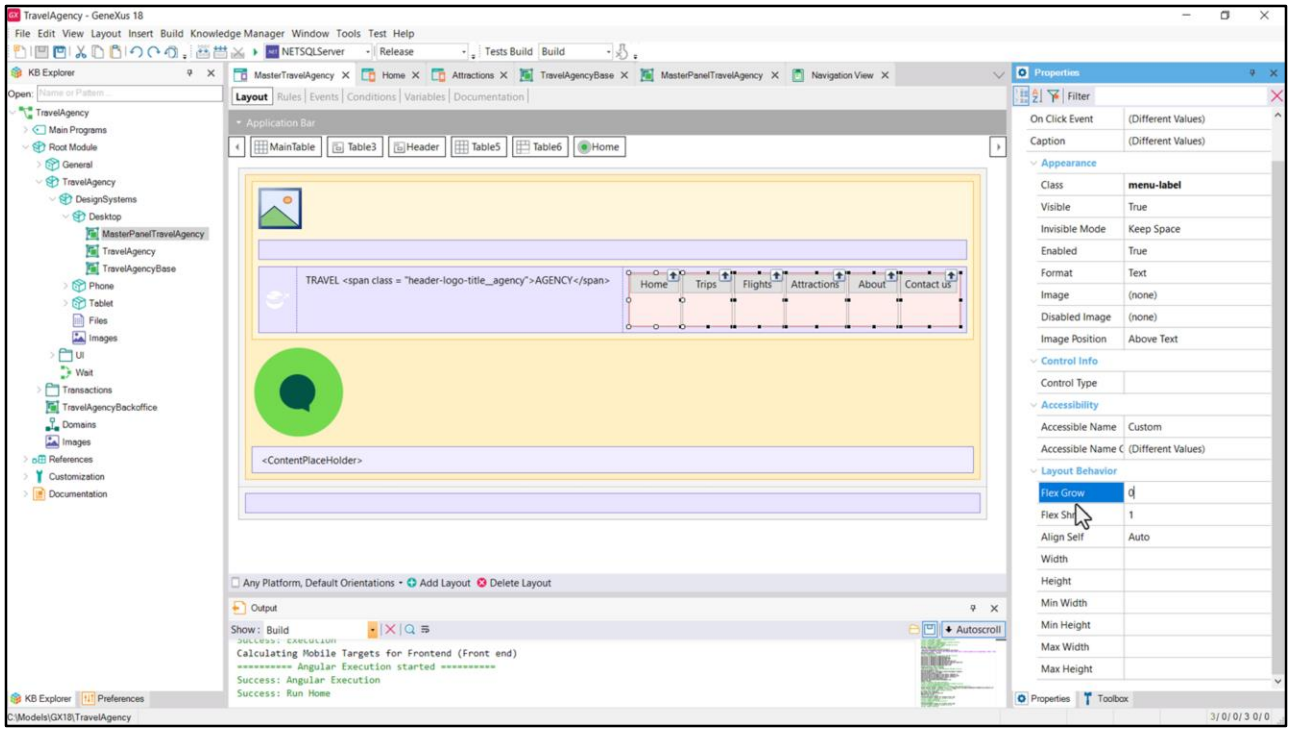
So, here we see the cell of the first button, the button and its caption.

And the same for the other buttons.

Note that it allows each button to grow so that it occupies equally all the free space of the flex.



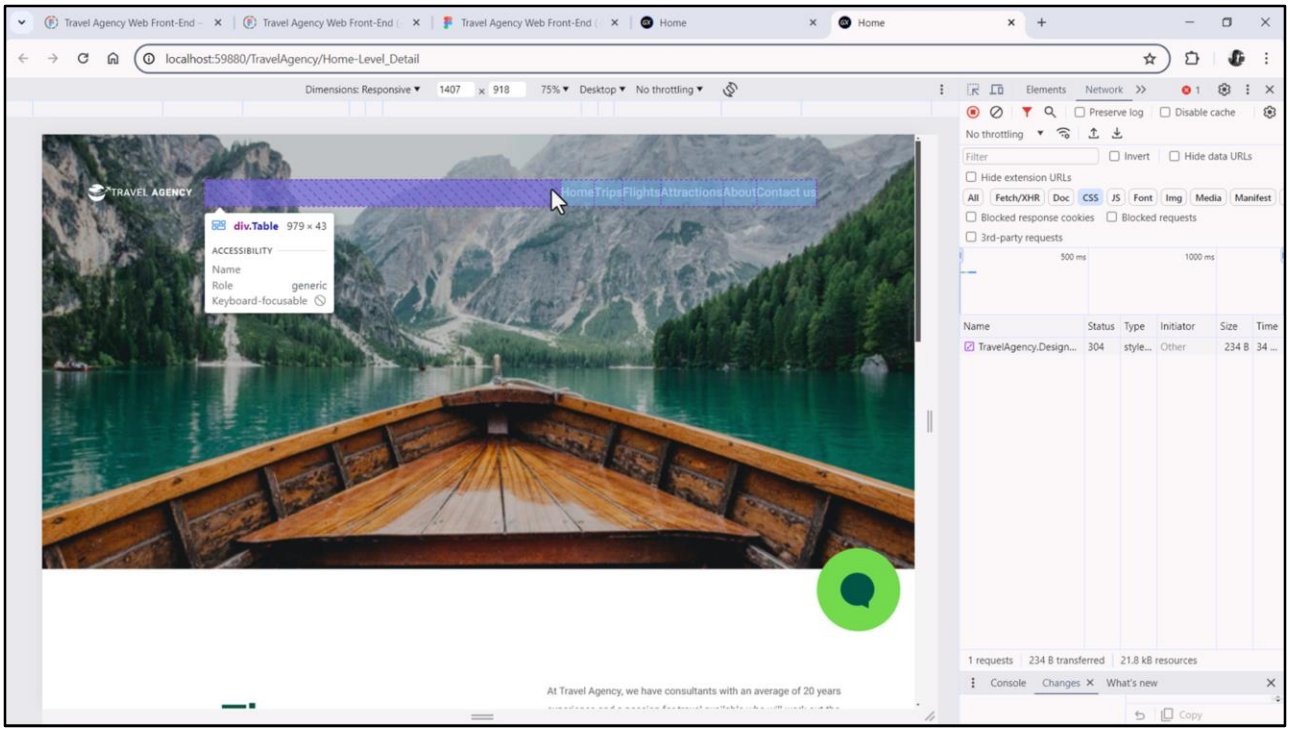
This is due to the Flex Grow property at 1, of each one of the buttons. Let's see what happens if we remove them...



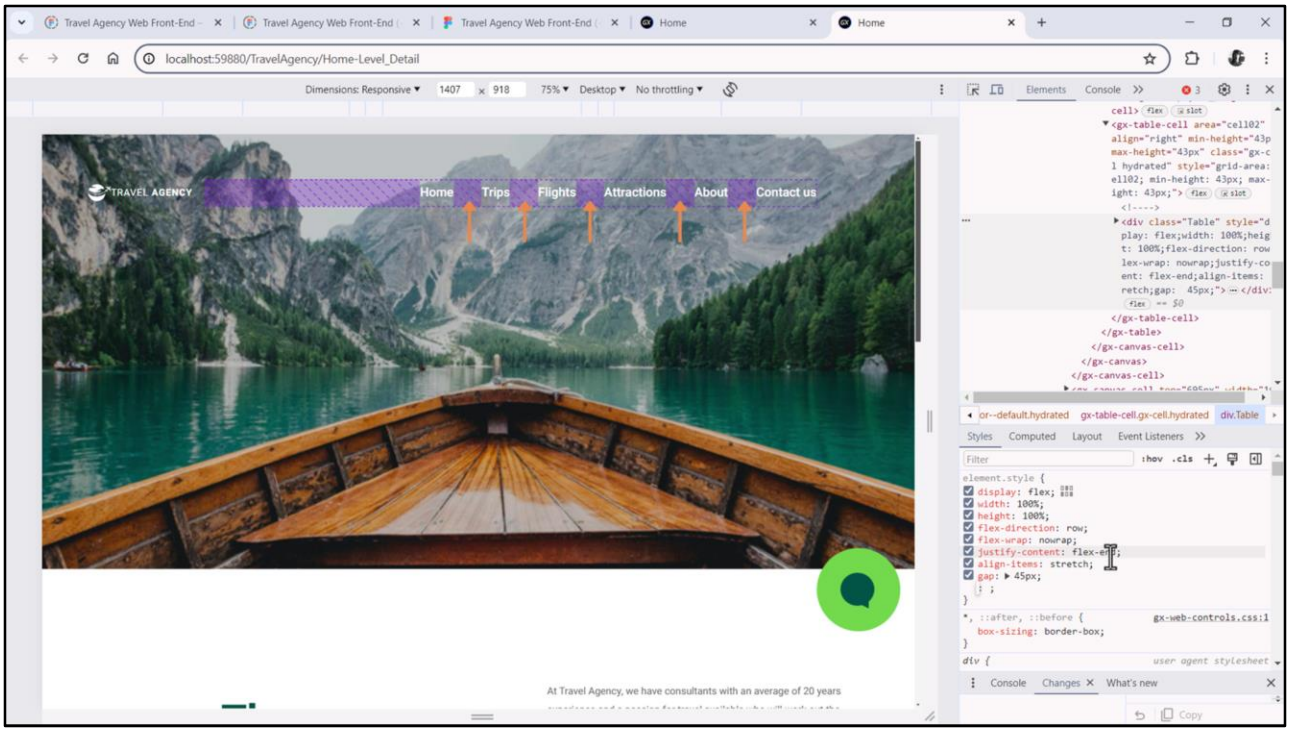
In GeneXus we see that although the content is justified according to the end of the flex, each button will have by default the Flex Grow property set to 1. Let's see the description of the property...

It determines how much this child will grow if there is free space to distribute among the other items in the line. If the item has a positive value for this property, the free space will be distributed proportionally among all the items that have a positive value.

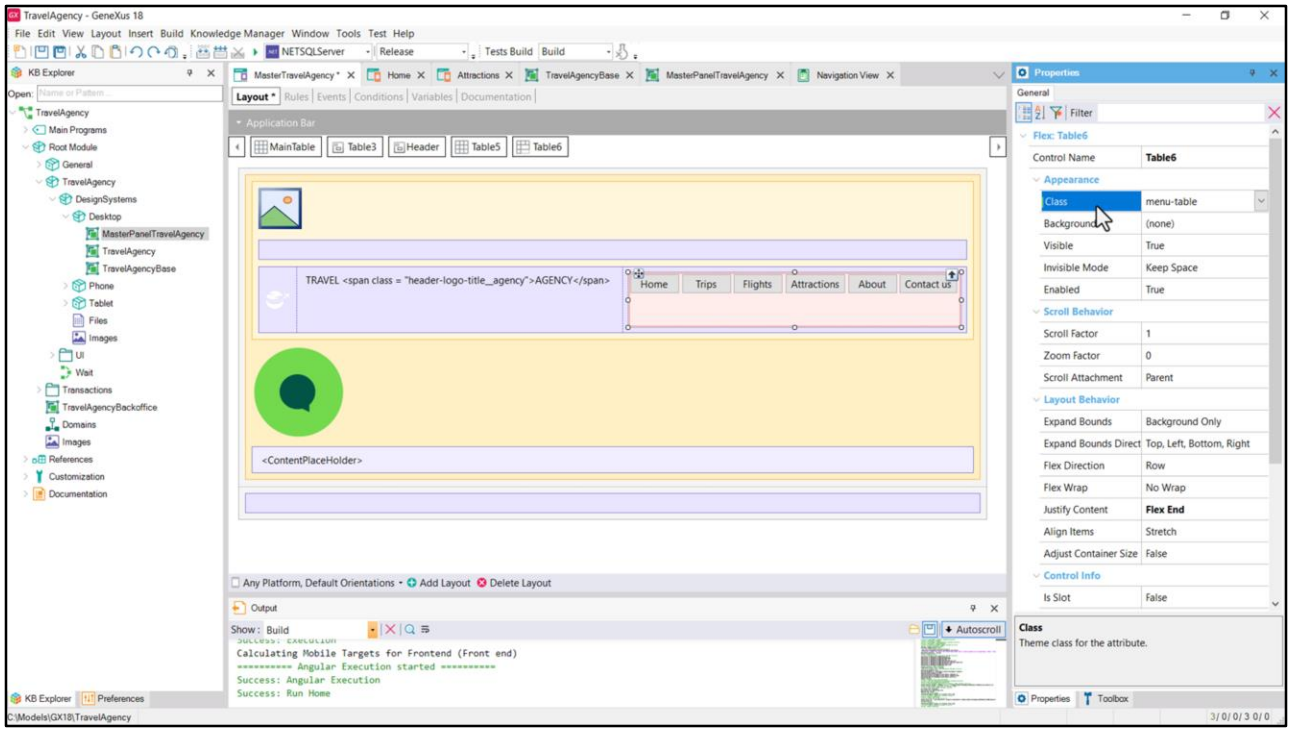
They all have the same value by default, 1. So let's select all the buttons and change their value to 0, so that they occupy exactly their space and no more. Let's run it...



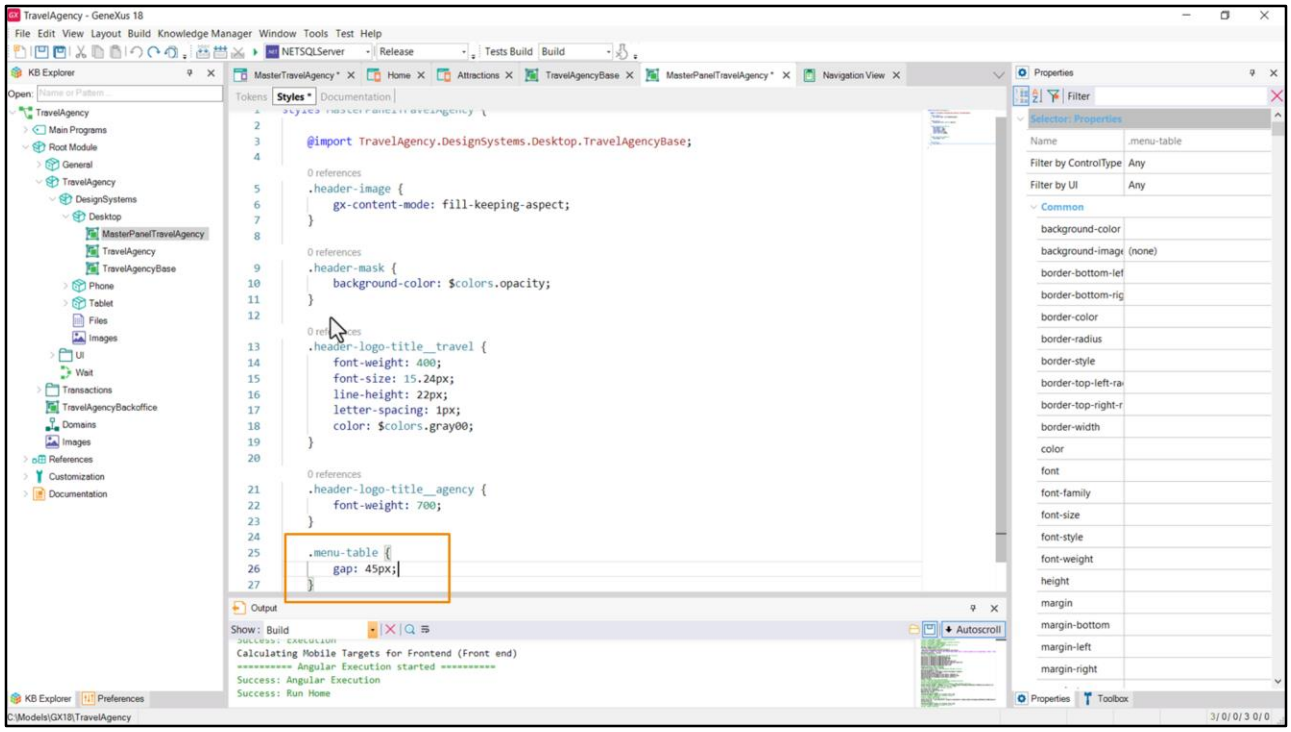
If we now inspect it... we see all the buttons next to the end of the flex, and no spaces between them.



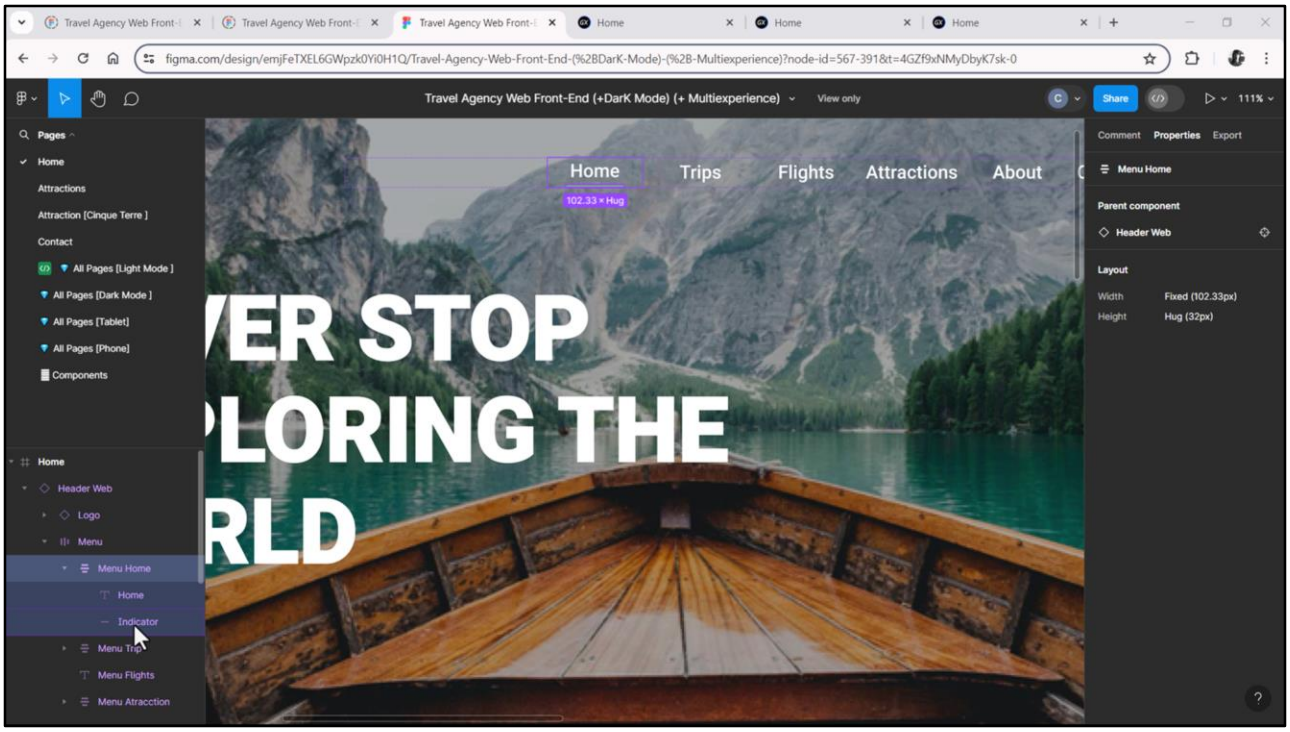
To give them a uniform spacing so that the distance between the end of one caption and the beginning of the other is the same, let's try the gap property, with 45 pixels, for example. Well, it looks good. There we see clearly how they are justified against the end of the flex, that's why there is all this free space in front, and how this gap of 45 works between them.



This is a flex level property... and we don't have it as a static property of the control, so we will have to specify it in a class, which I will call menu-table.

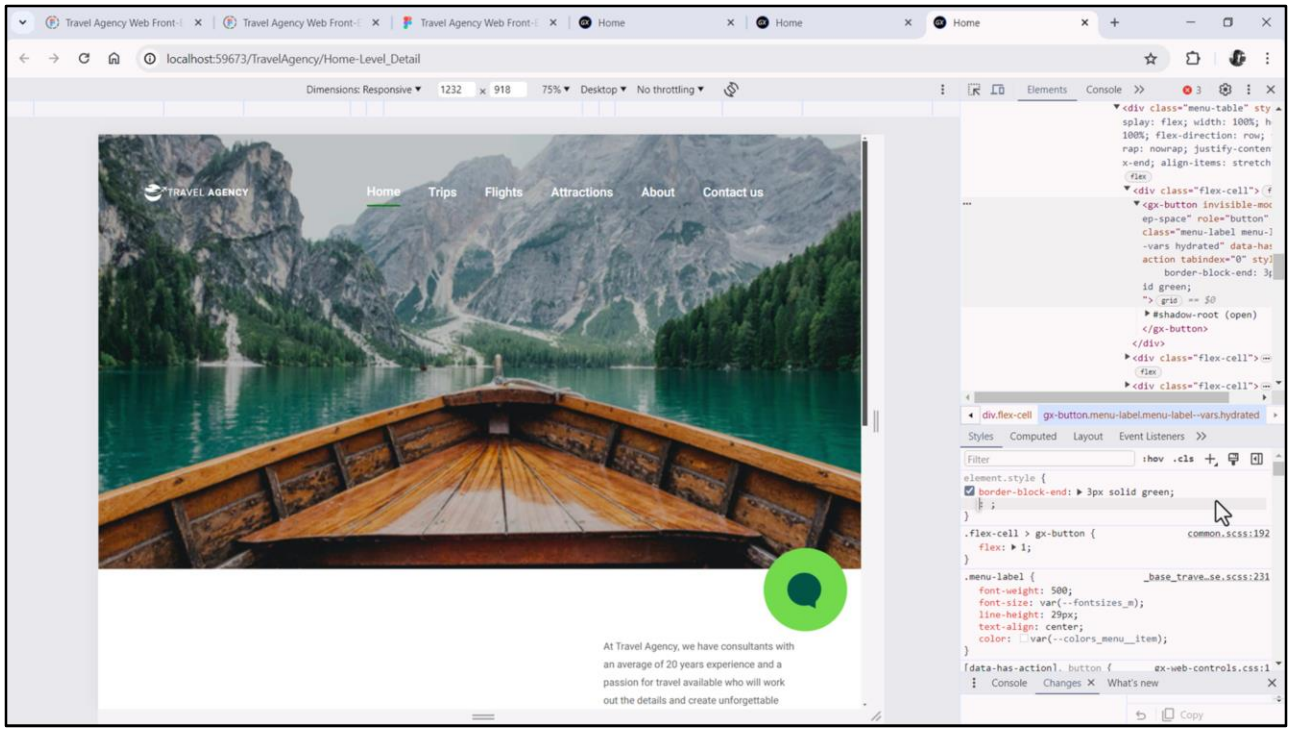


And there I place the gap property.



Now we would have to think about how to implement the indicator of the selected option.

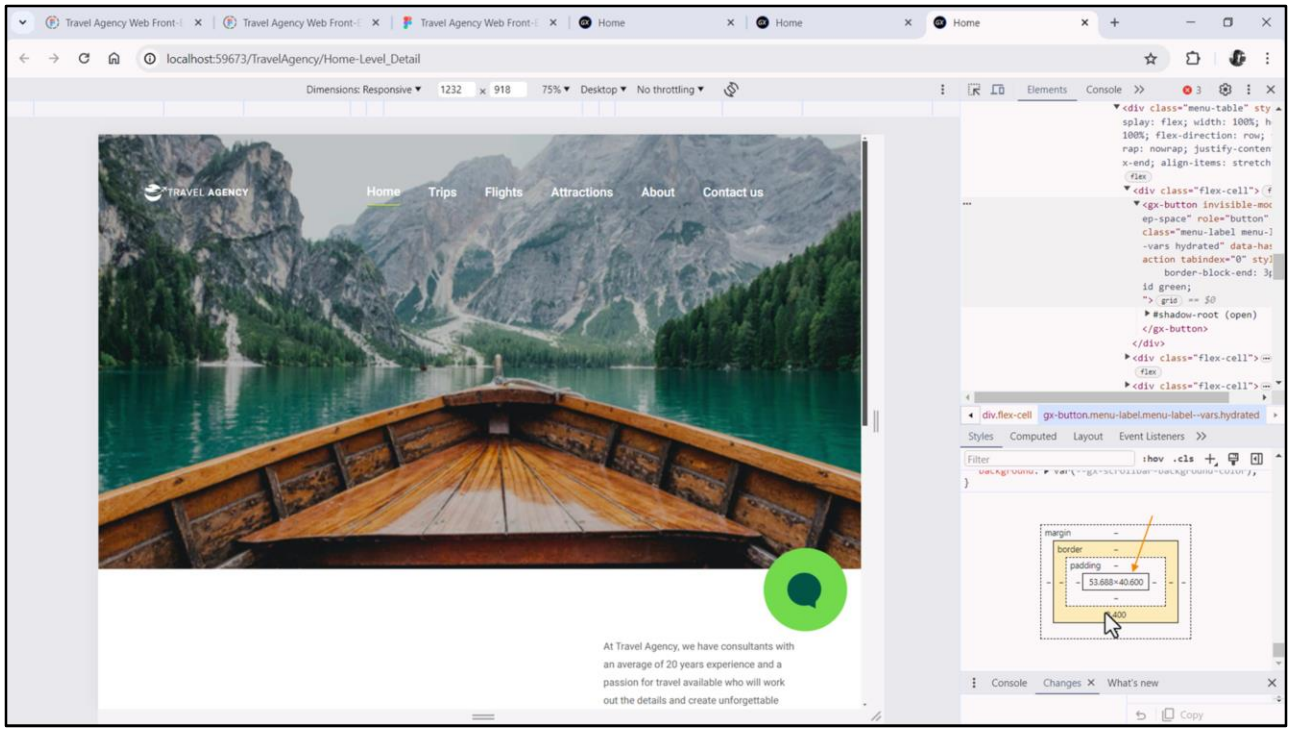
In Figma, we see how Chechu implemented each option as a vertical flex, with the text above and the indicator as a line below.



Since this is an action we use a button in GeneXus, and not text, and like any control, it has 4 borders, one on each side.

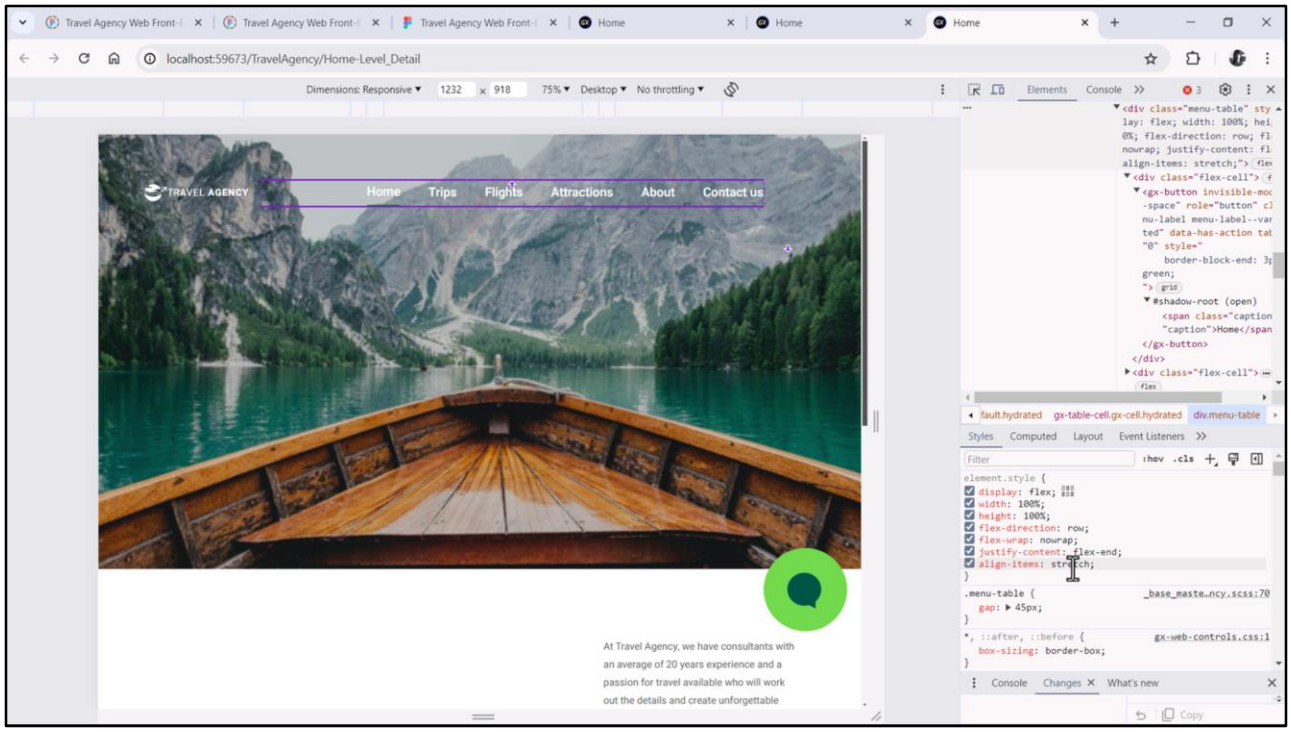
It will be enough for us that the borders at the top and at the bottom have a positive value but are transparent for all the buttons, and on the other hand the one at the bottom has color only for the selected button.

Let's try with this element... the property is border, and I will use the logical one and not the physical one, so it will be border-block, with which I will be referring to the two borders in vertical direction: the beginning and the end. If I want only the end one then I enter end. And there I specify that it will be 3 pixels, for example, with a solid line, and the third parameter would be the color, for example, green.

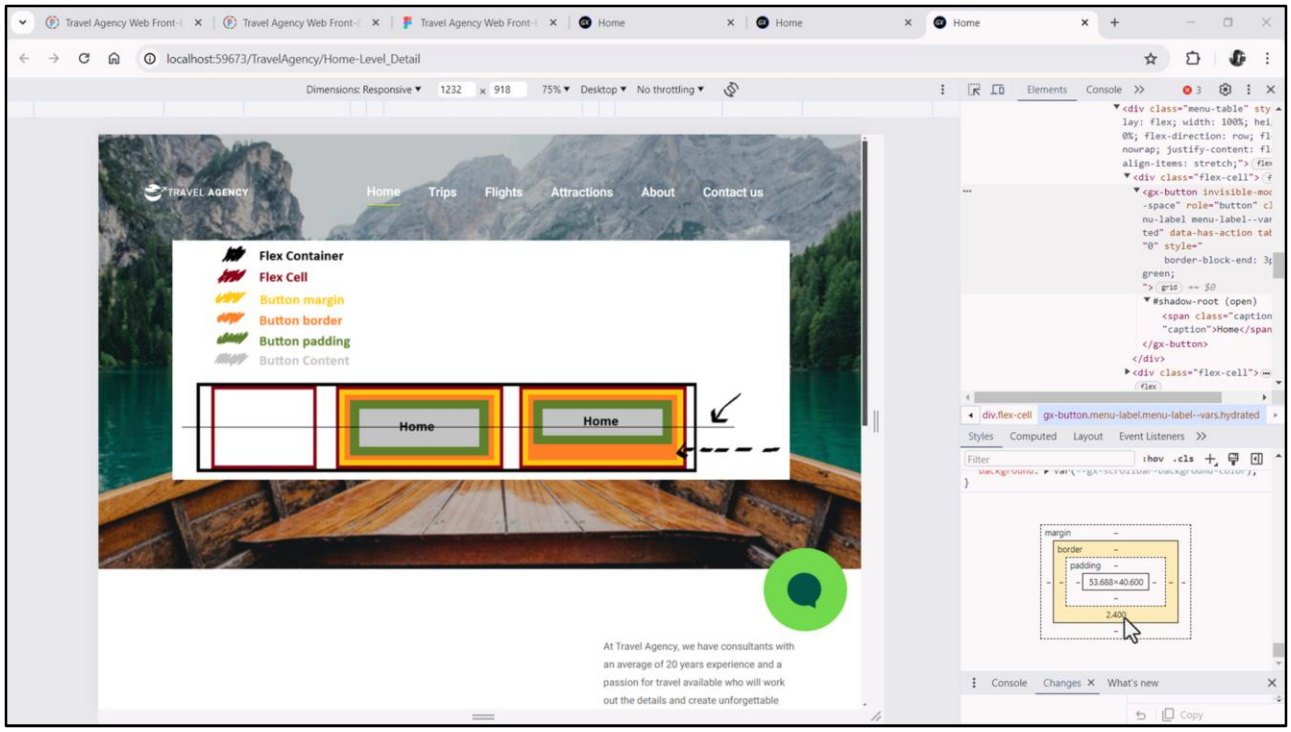


This 2.4 instead of 3 is a floating point precision error. It usually occurs if the screen is zoomed in multiples other than 100%, which is the case. Let's not pay attention to it now.

Note that by giving 2.4 to the border, the height of the button will be of this value, so that together they add up to 43 pixels of the height of the flex.

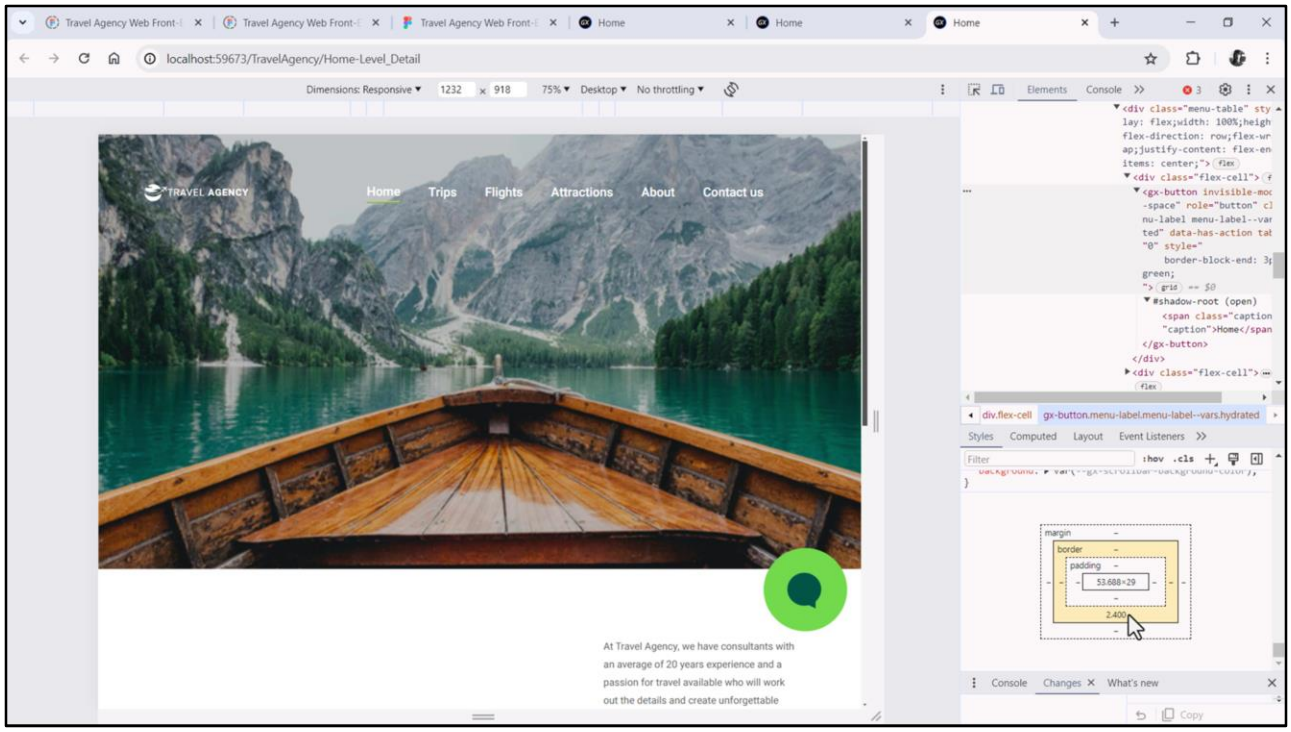


What is the problem with this? We see that the flex has the align-items property with the default value which is Stretch. This causes each item to stretch to vertically occupy the entire height of its cell, which in turn will occupy the entire height of the flex. In this case, 43 pixels.

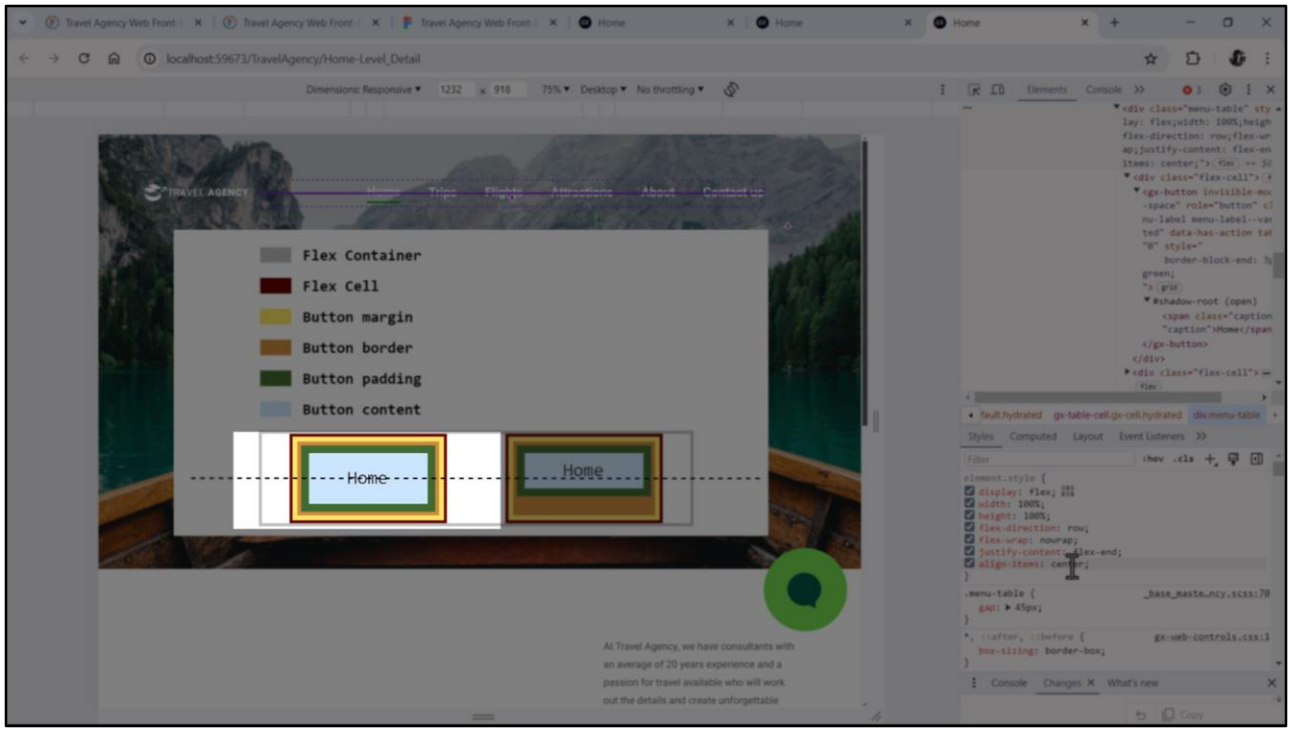


But each button box includes not only the content of the button, but also padding, border, and margin. So, if we leave a lower border of 3 pixels, what we will see is that the text of the button will not be centered in the middle, and therefore it will not be centered in relation to the Travel Agency logo.

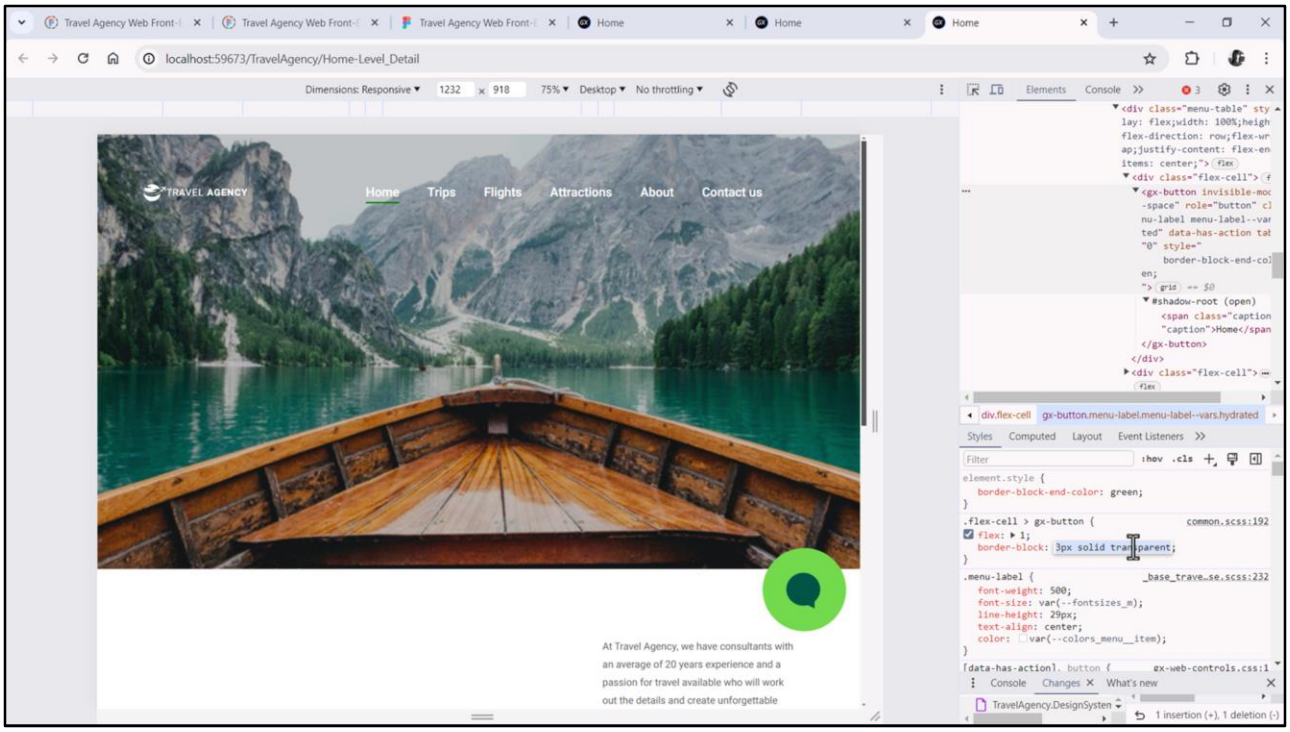
In this somewhat messy drawing I represent this situation with the button on the right, where I'm leaving a bottom border larger than the top one (which in our case is 0: the bottom one is 3 pixels, the top one is 0). And by doing this, the content of the button is not going to be centered. That is the problem.



Changing the alignment of the flex items to center does not solve the issue. The only thing that it does is that the height of the element is not the height of the flex, but it corresponds to the height of the text, of the button, plus padding, border and margin... So it will be 32 pixels instead of 43 pixels.
But as far as center alignment is concerned, we are in the same situation.



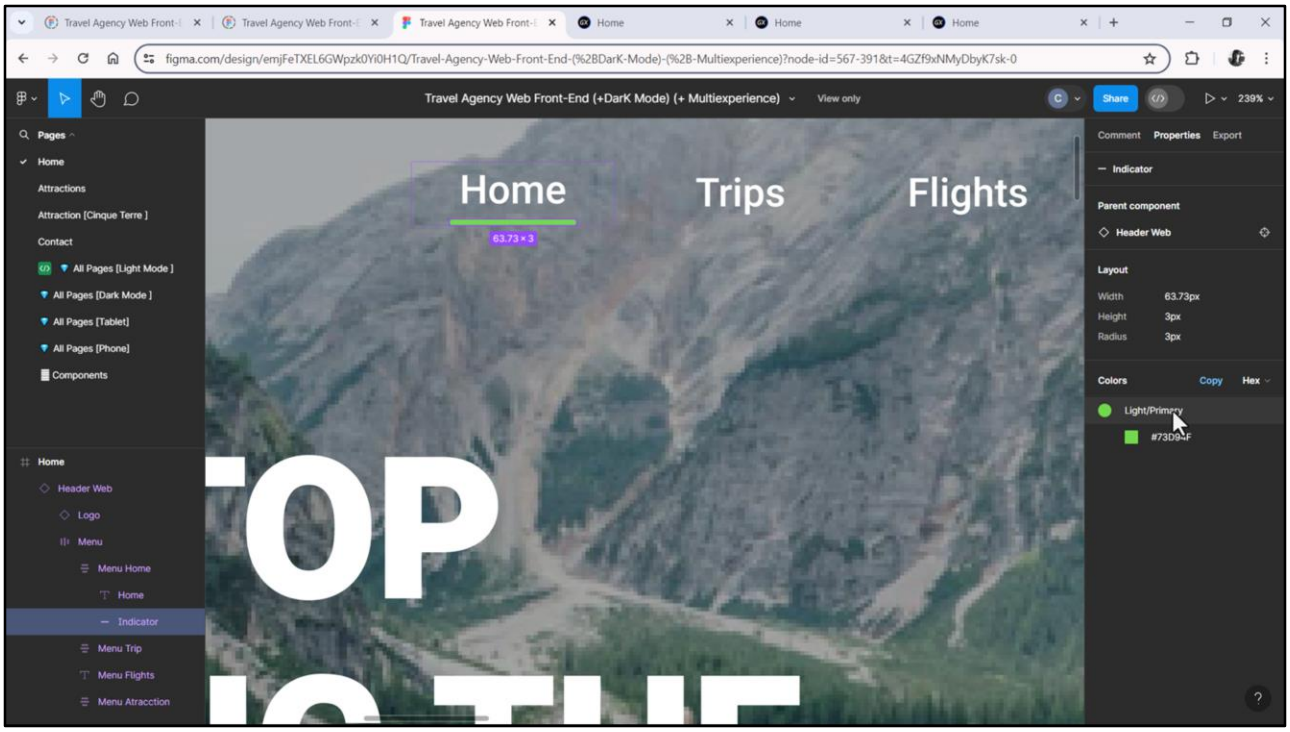
One solution is to give the same thickness to both borders, as in the example on the left. There we will have the text vertically centered. And associate the top one with transparent color.



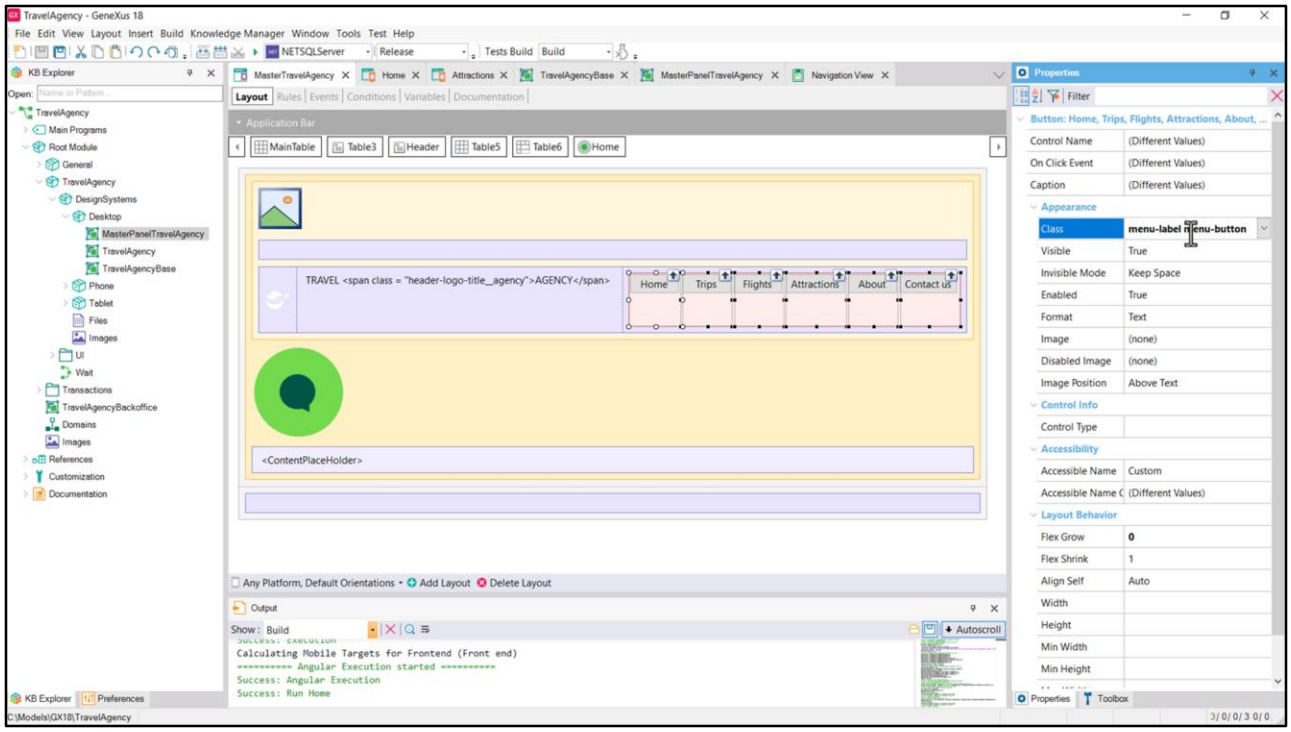
To test it here let's select the element corresponding to the Home button. This property is governing the style of this particular element. In this other selector we have everything that applies to all the buttons of the flex cells, so let's try there to set border-block (that is, it will be valid for top and bottom), of 3 pixels, solid, and transparent color.

And then for the element that is selected, we don't need to repeat this, but just give it green color. So we choose this property... and the value green.

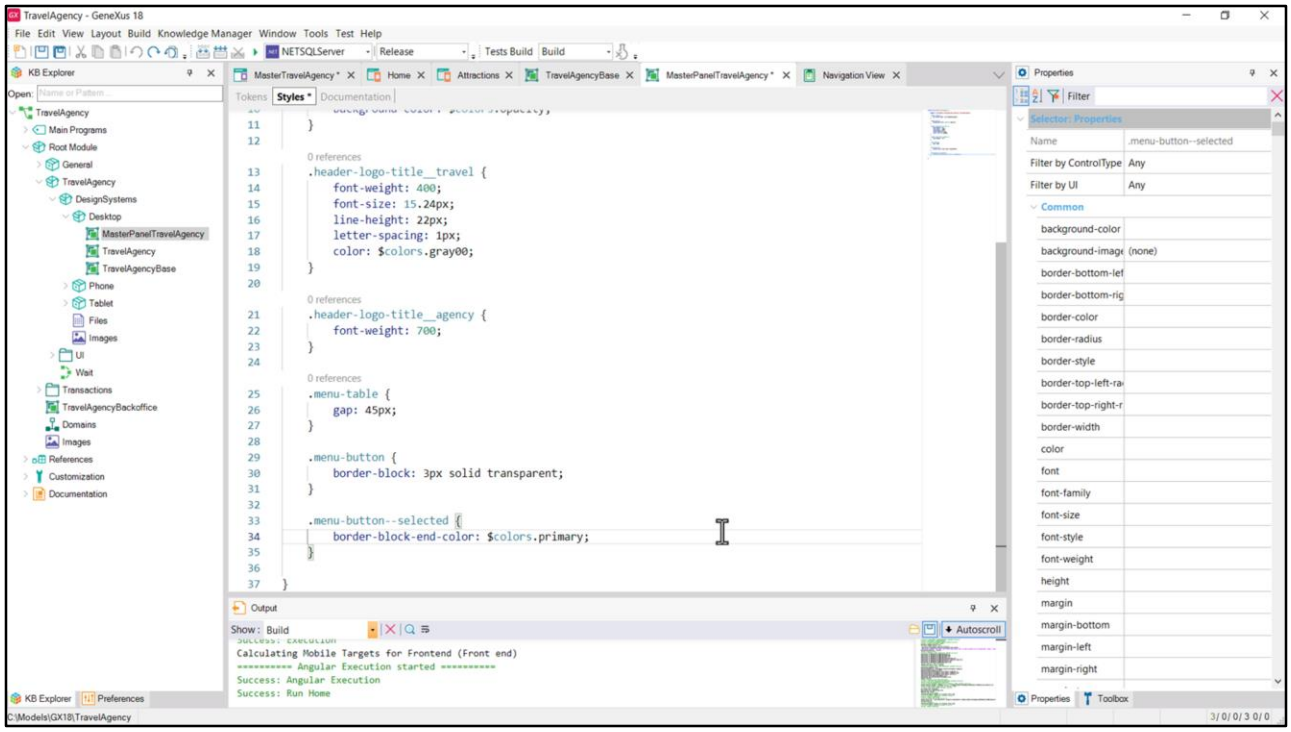
By doing this, we will have applied this property for all the buttons, and in particular for each one, the color of the bottom border is changed from transparent to this green; for the corresponding ones, at the right time.



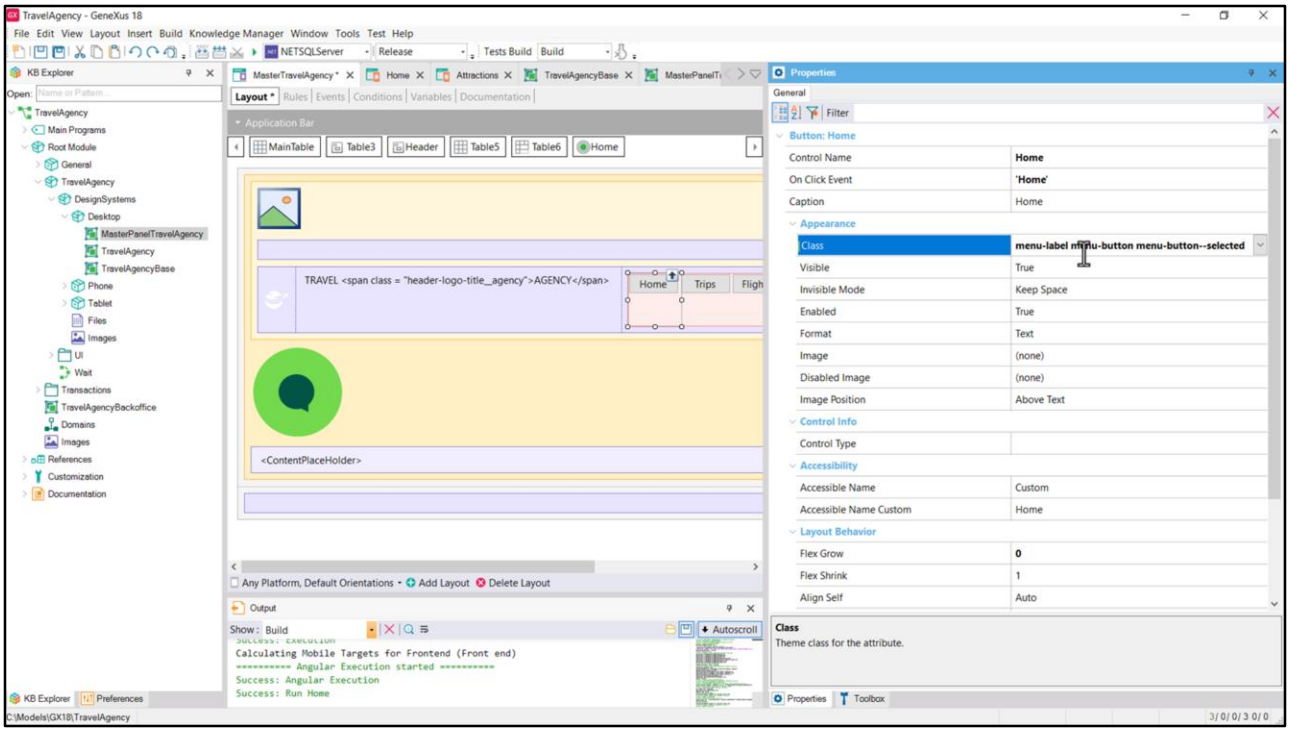
In fact, if we go to Figma... the color will be primary, and the 3 pixels are fine. We will not pay attention to this radius at this time.



Then to the class that all the buttons have, which governs the typography style of the labels, let's add another class that I will call menu-button...

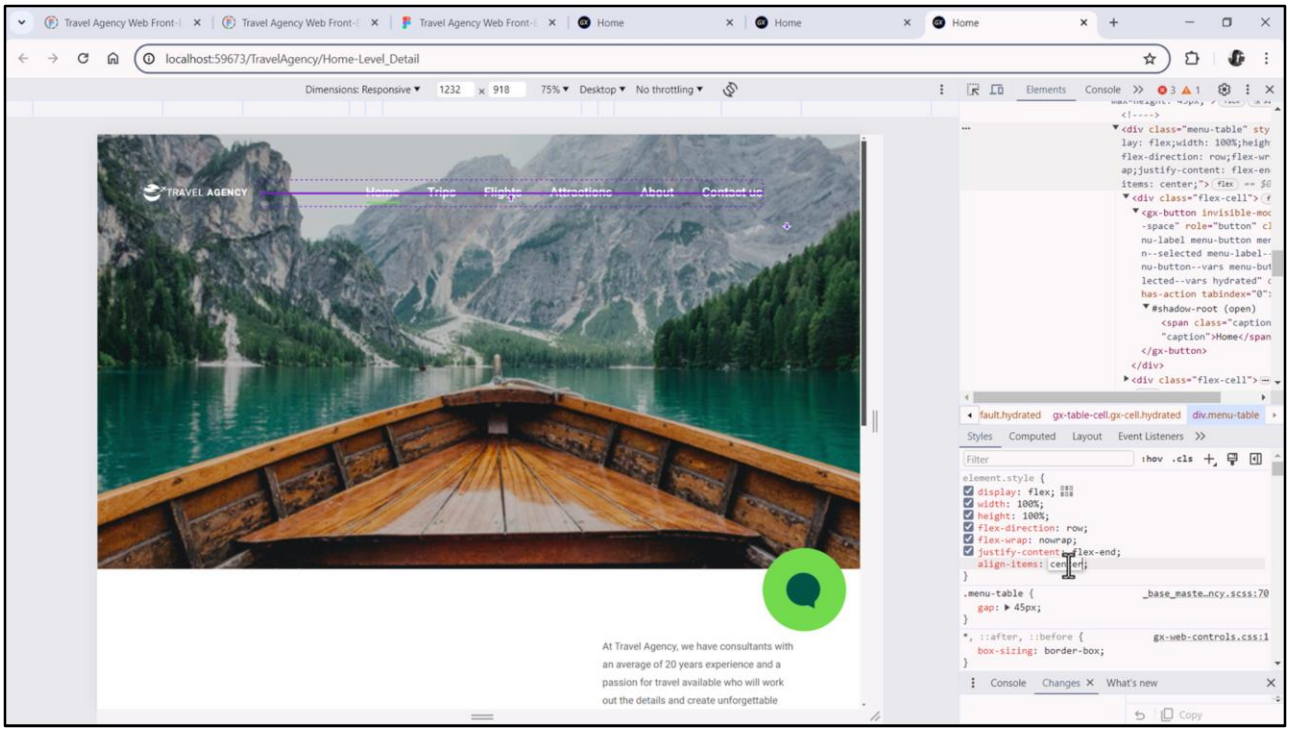


...To which I will assign the transparent border at the bottom and at the top... And then I will create another class to change the color of the bottom border to the primary color.

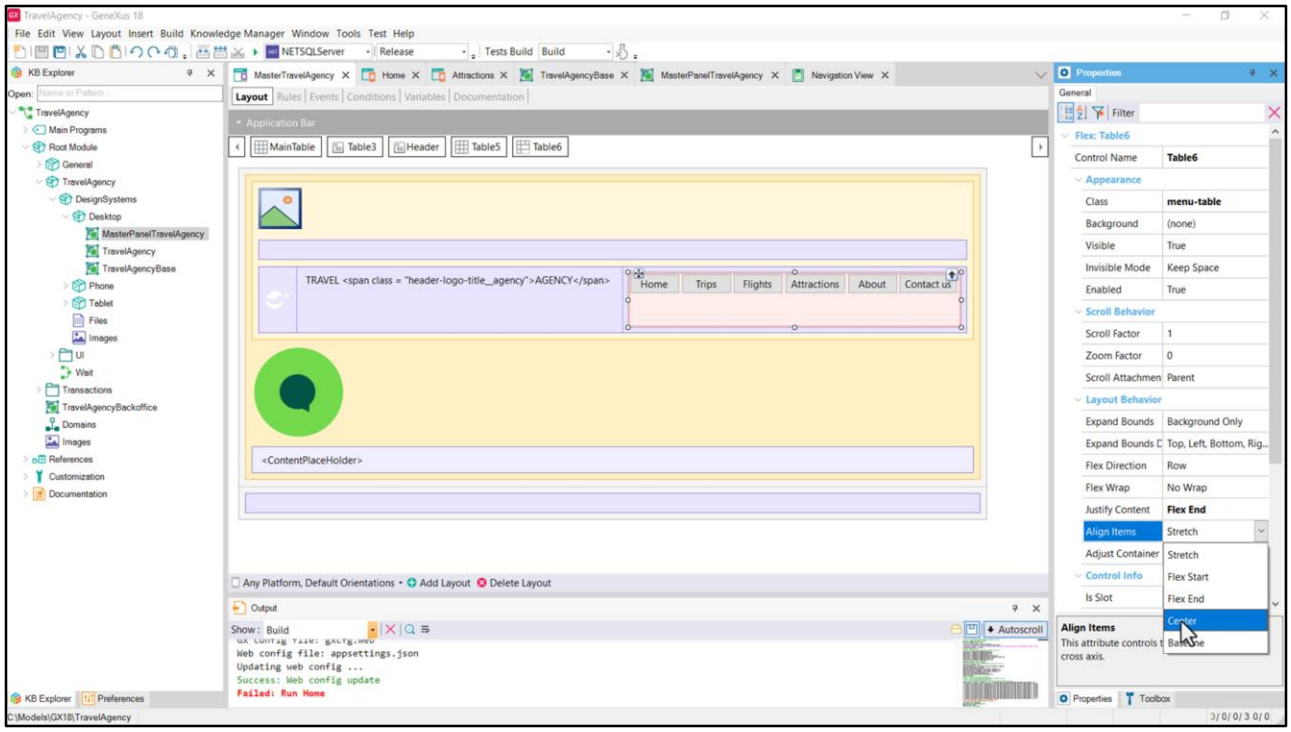


For now I'm going to assign it to the Home button so that we can see the style.

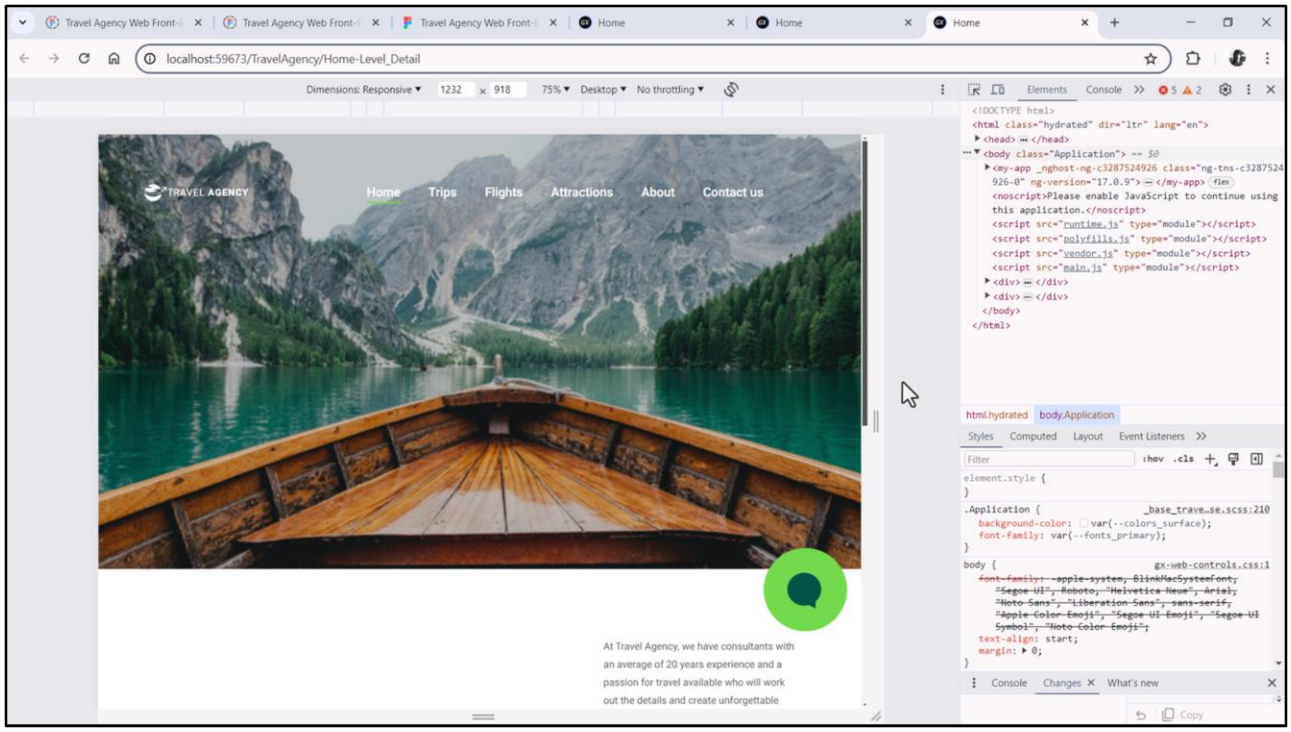
When we study the navigations there we will see how to change it dynamically according to what is loading in the contentplaceholder.



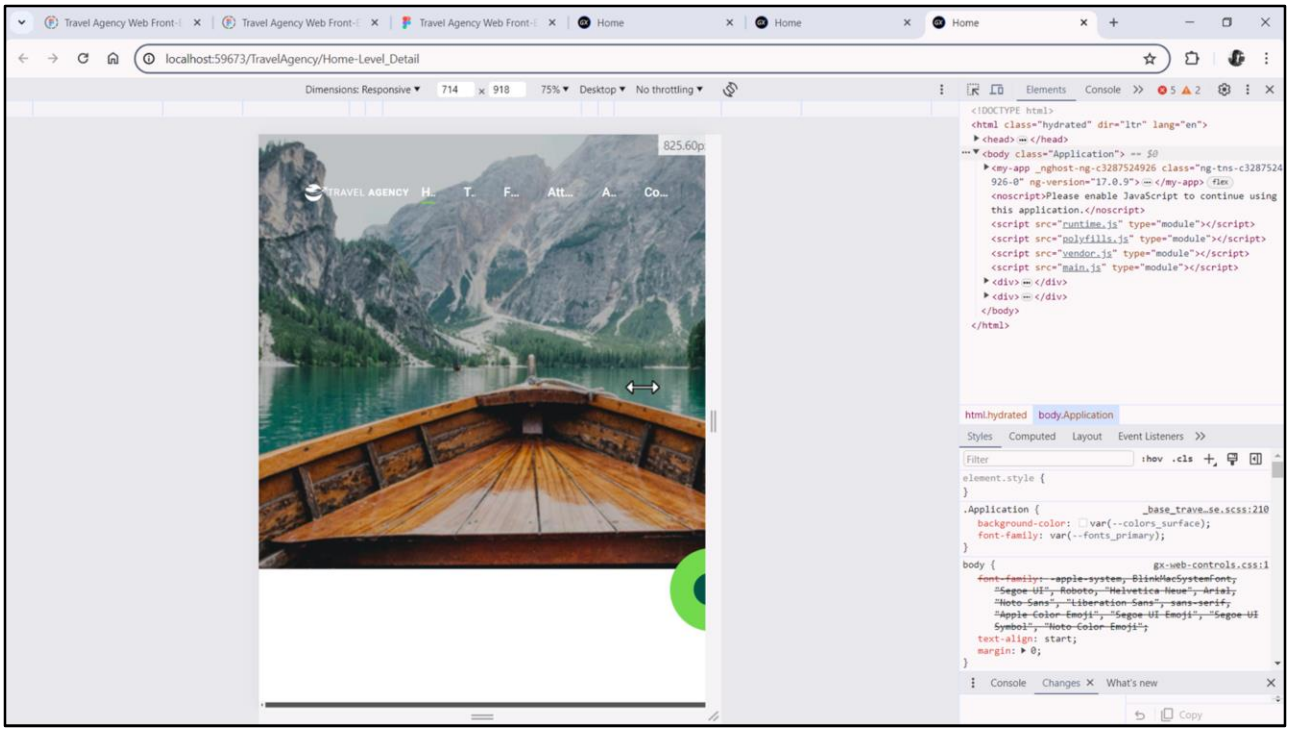
It may seem that there is a great distance between the text and the border line. It's that, remember, the button is occupying the entire height of the cell, which is occupying the entire height of the flex, because we had left the default value to the align-items property, which was stretch. If we change it to center we achieve the minimum height necessary to hold its content plus padding, border and margin. And also that it is centered in the middle.



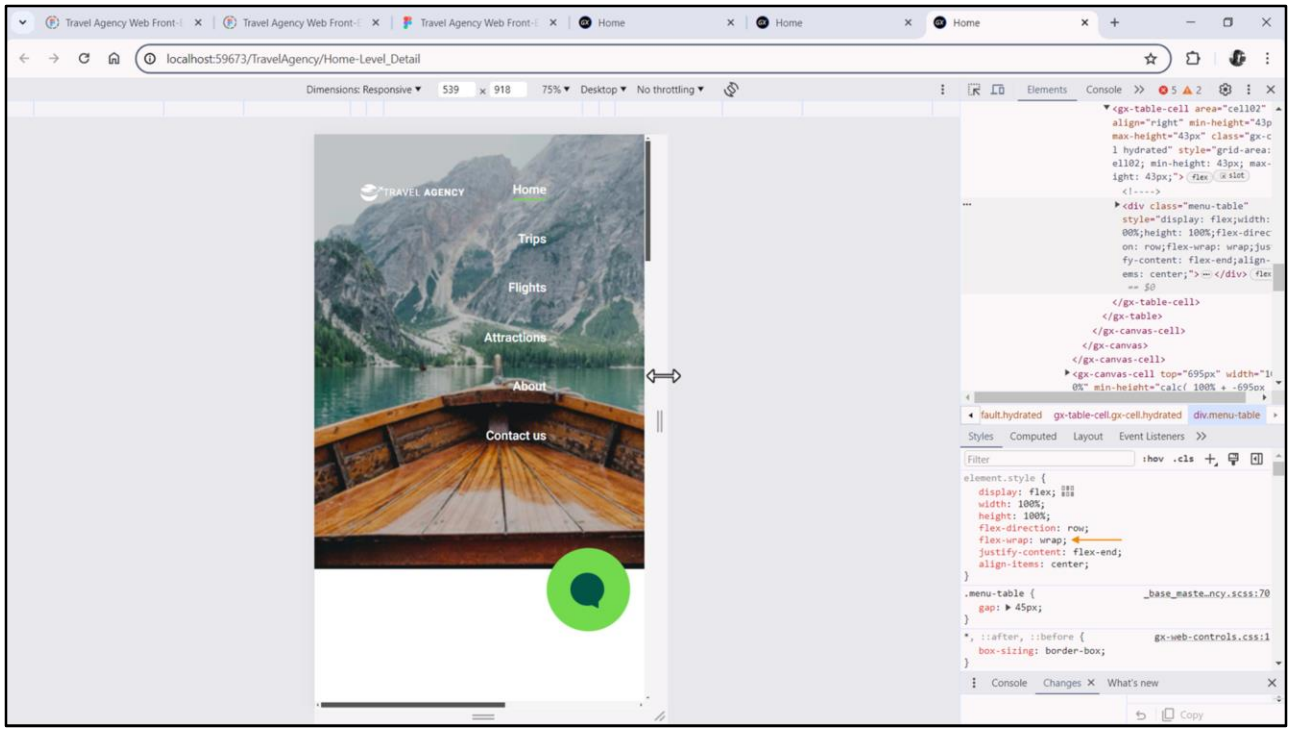
We achieve this with the static property at flex level... (there is a bug in the abstract editor that causes the buttons to be hidden, but they are there. It is only a visual problem. In the GeneXus web editor it is already corrected).



In fact if we run... we see everything as we expect it.

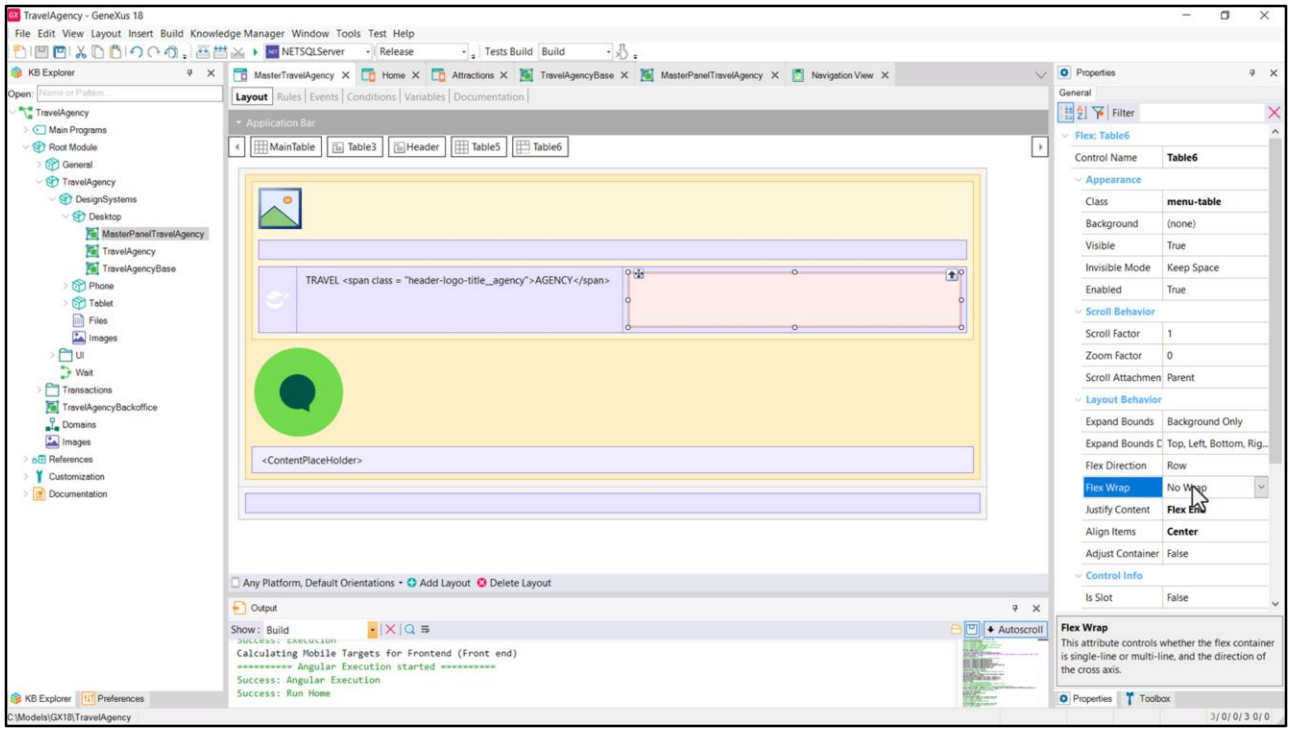


One last thing about the menu: see what happens if we decrease the screen width. The buttons start to get smaller to fit in the space available to them, leaving the gap we indicated, of course.

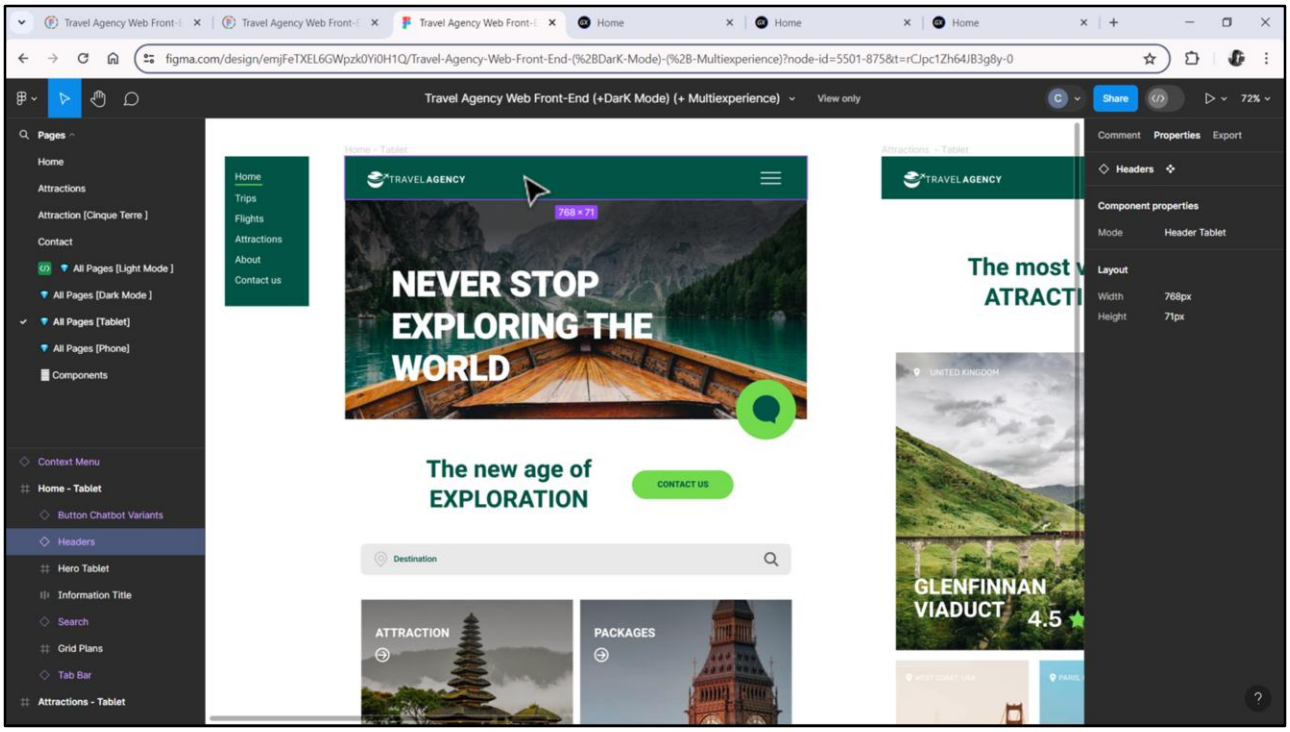


One of the features offered by the flex container, unlike the table, is that it allows wrapping, so that the items that no longer fit in the row are placed in another row.

We are not interested in this for our case, but I wanted to show it anyway.

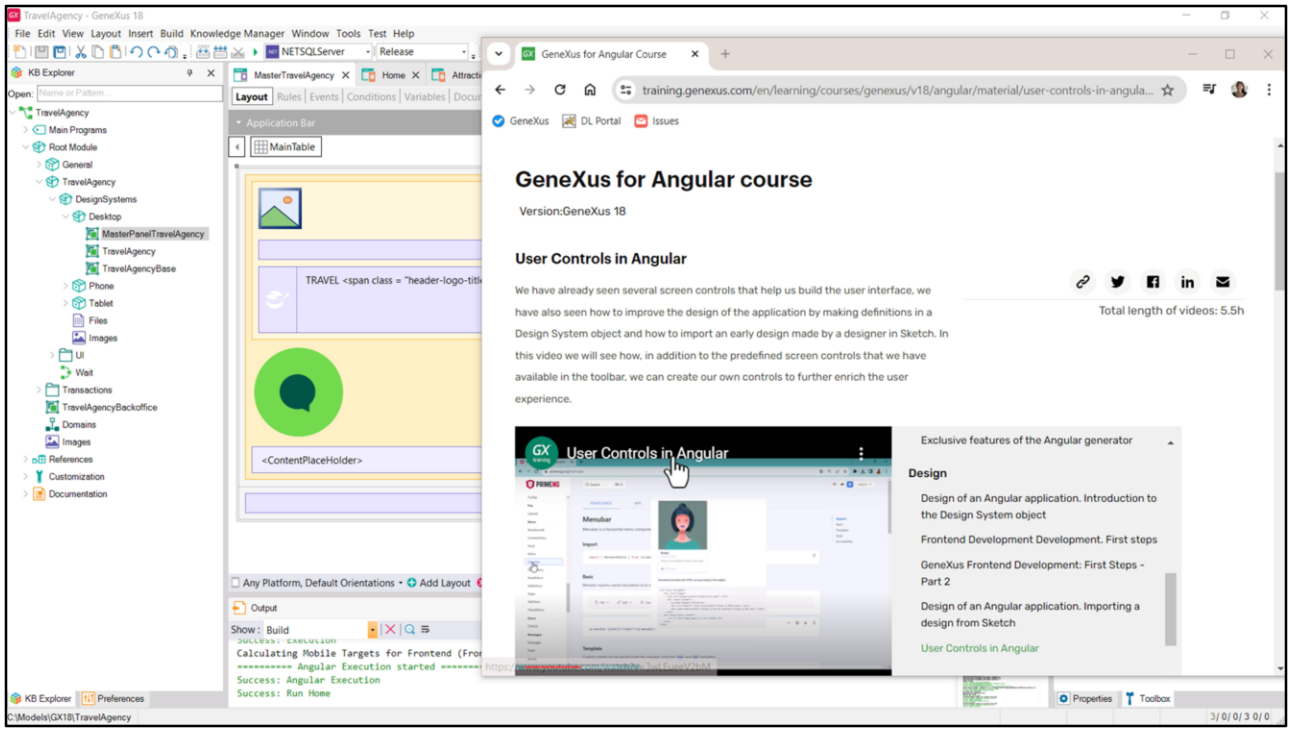


Look where the property is located.



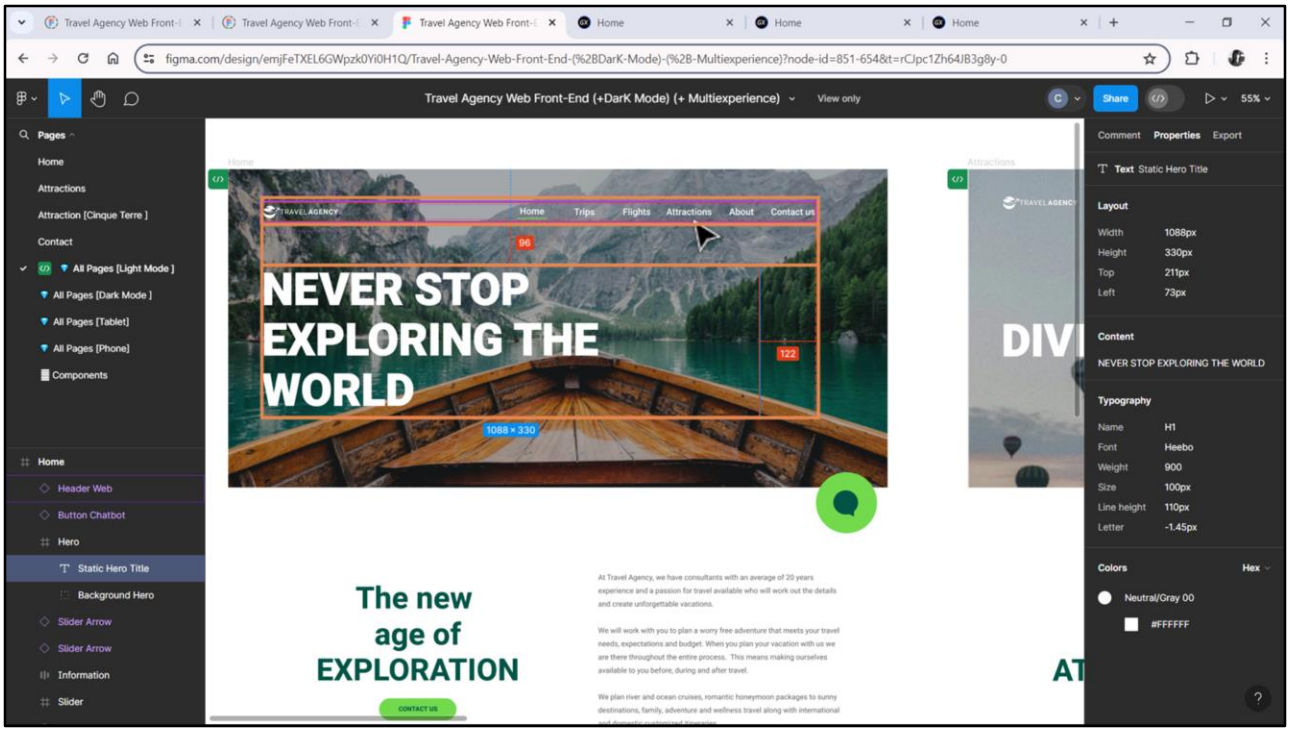
Remember that Chechu put together other designs for the Tablet screen size... where the menu will be the classic hamburger, and for Phone size as well.

We will come back to these breakpoints when we talk about multiexperience in the next module.



I take this opportunity to comment that if we need more sophisticated controls that include design and behavior, we can include them in GeneXus by defining **User controls**, that is, objects in which we can set the HTML and minimally intervene so that it can be used in any GeneXus object with an interface. That is to say, we incorporate it to the KB and it is already available to be used.

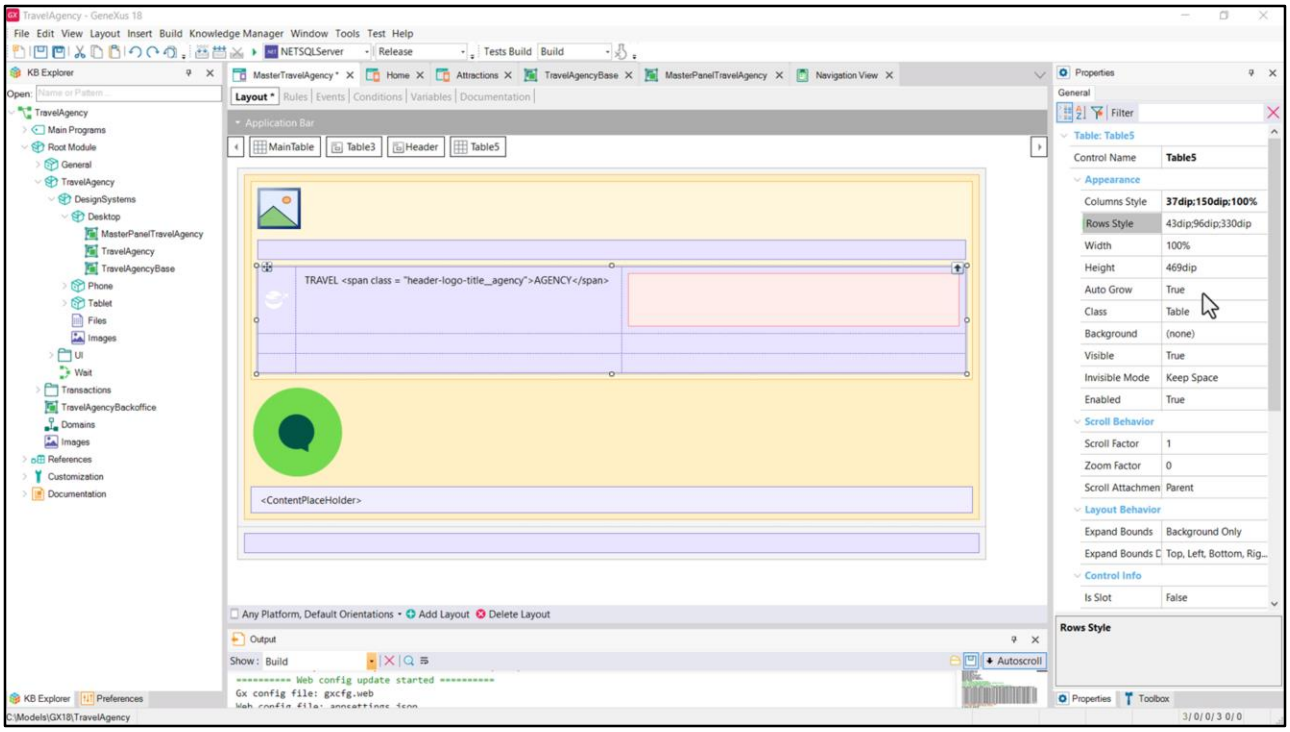
This will be valid for the Web platform (both Angular and .Net and Java).



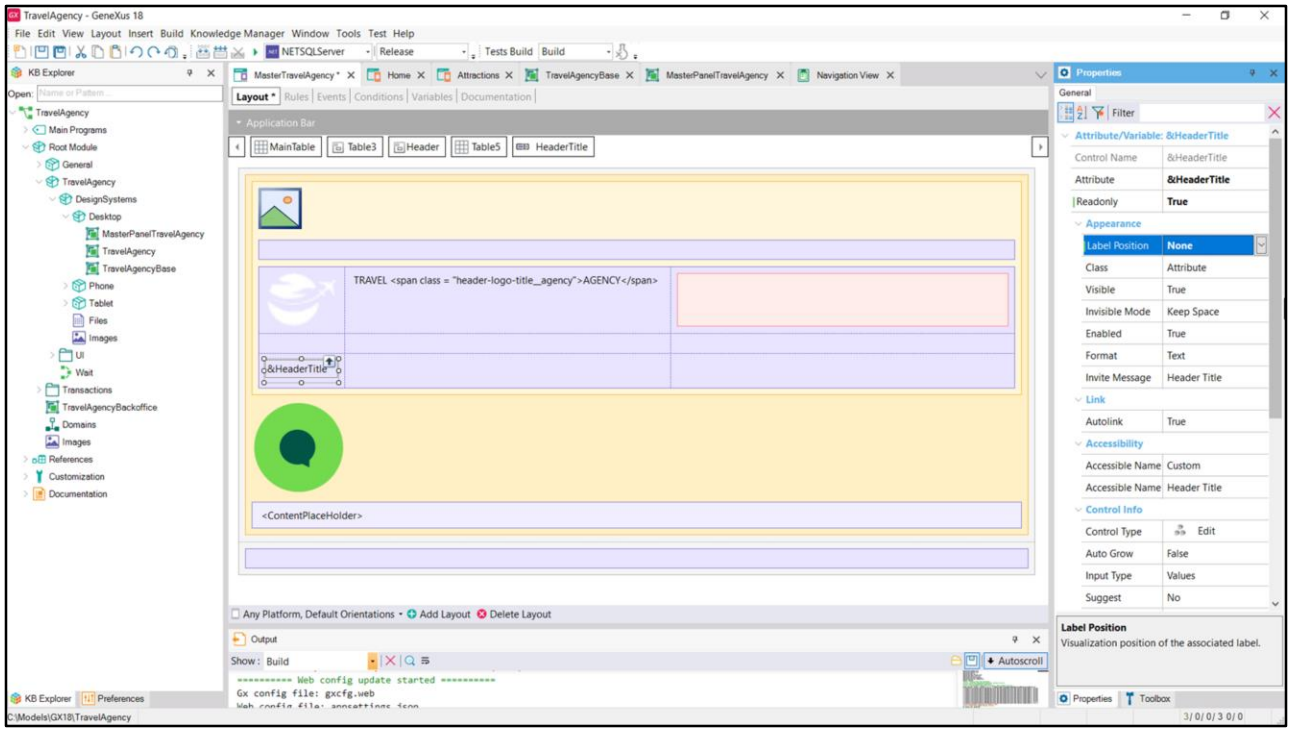
Now let's quickly add what is missing from the Header to finish implementing it: the text that stands out over the image. Remember that its typography style was H1. And let's copy its content to the clipboard.

It is aligned to the left together with the travel agency icon, and that's why we had thought of using the table. Now let's extract the measurements: the height of the row will be 330 pixels, and from that of the menu it will be at this other distance, 96 pixels...

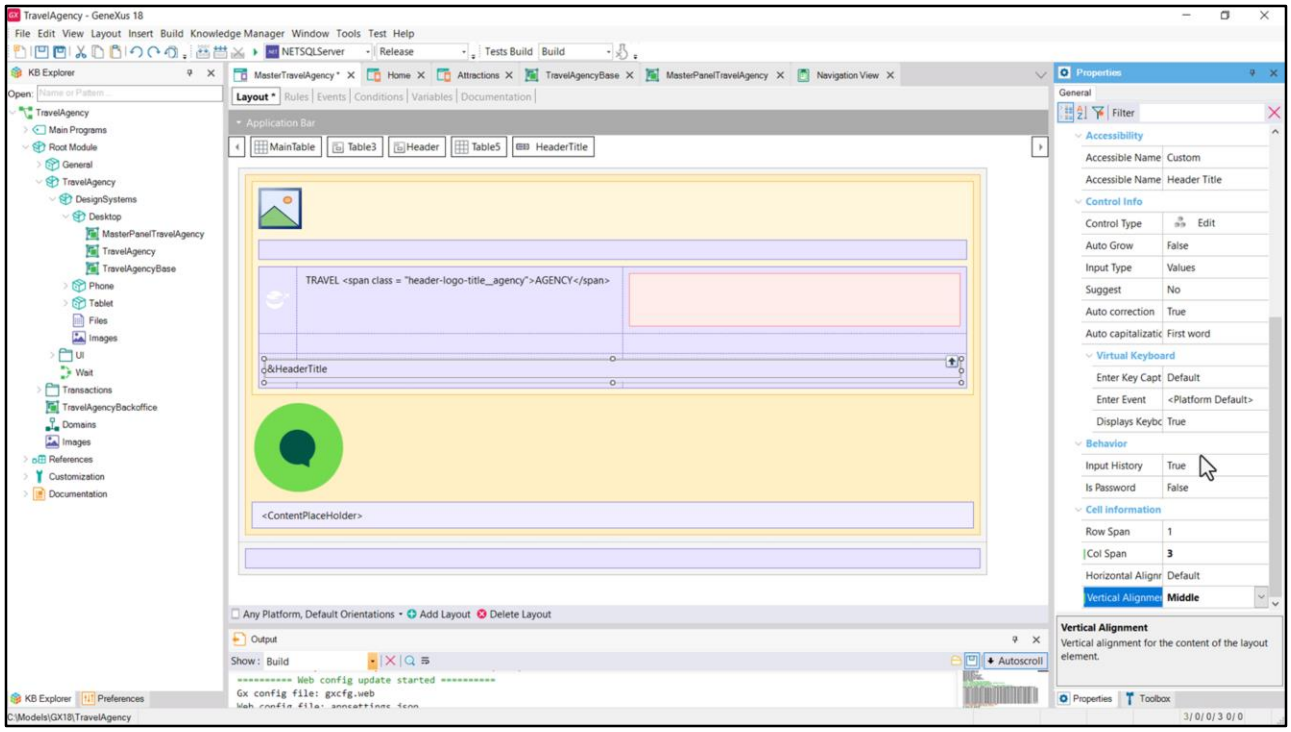
Clearly we will choose to implement the text not as a textblock but as a variable, since its content will vary from screen to screen.



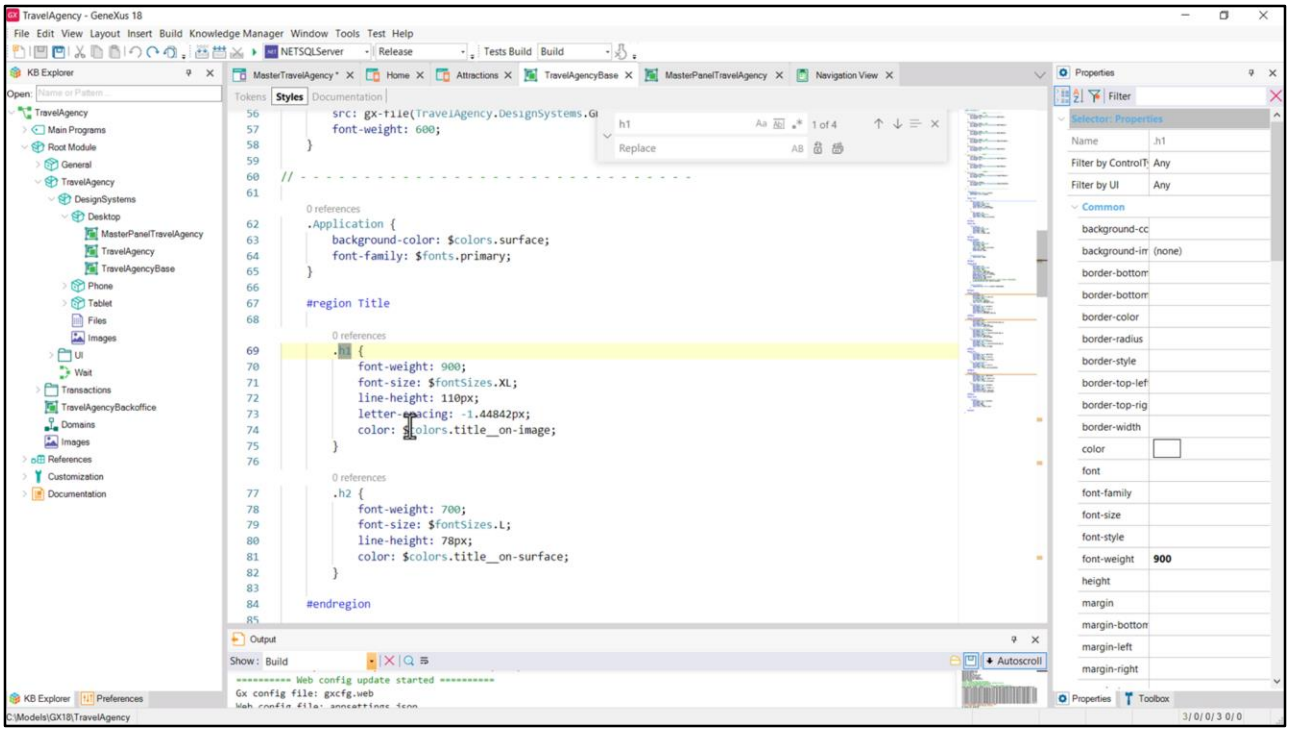
We then insert two rows to the table. The second one will be 96 pixels and the third one will be 330 pixels.



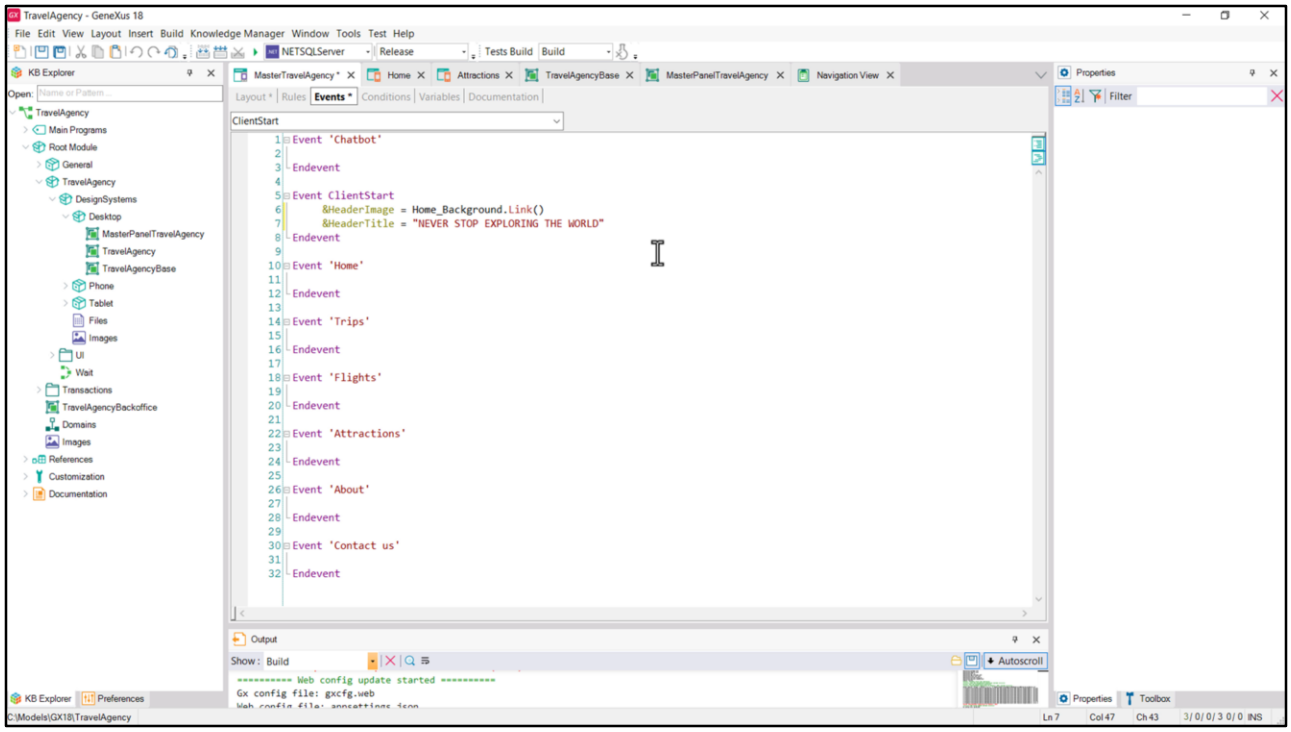
In the third one, we insert a variable control... named HeaderTitle.... and varchar data type.
It will be readonly and without showing the label.



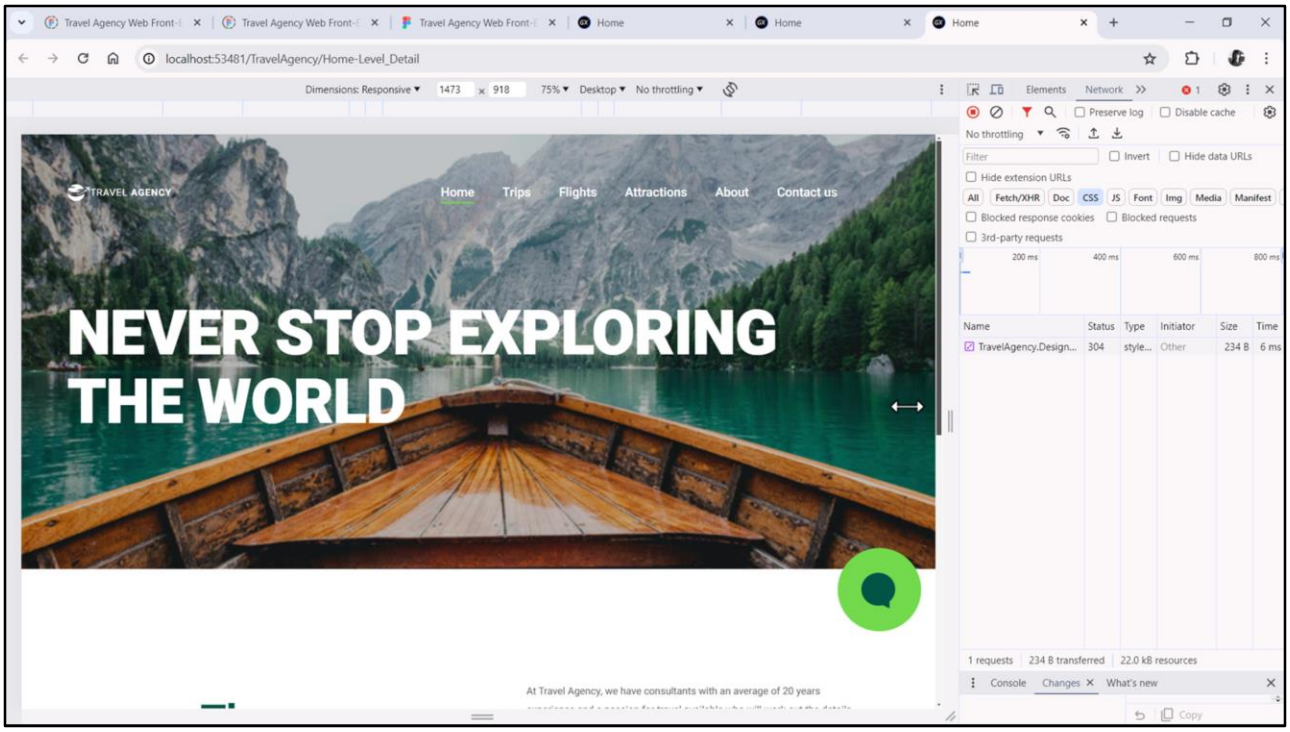
But we will also want it to expand to fill the three columns. And we'll want its content to be vertically aligned in the middle.



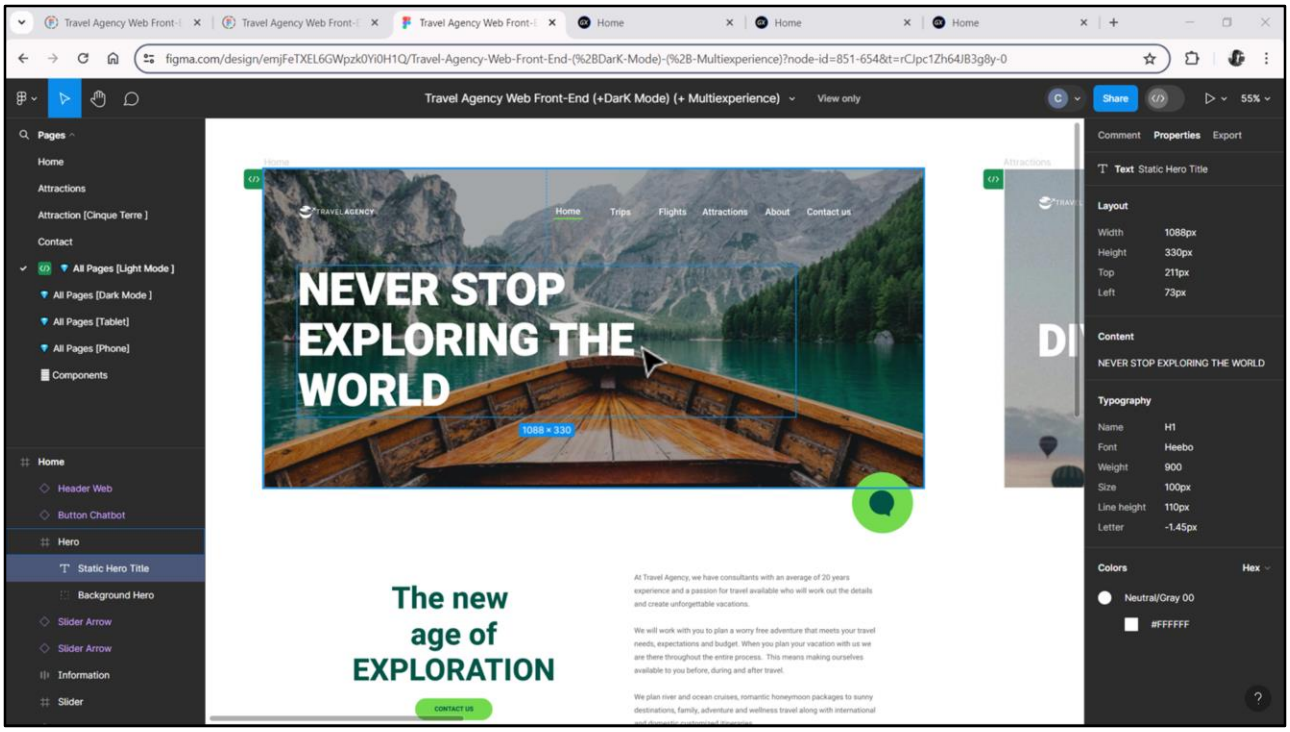
On the other hand, its class will be the one we had called h1 when we introduced all the classes for the typography in the preparation module.



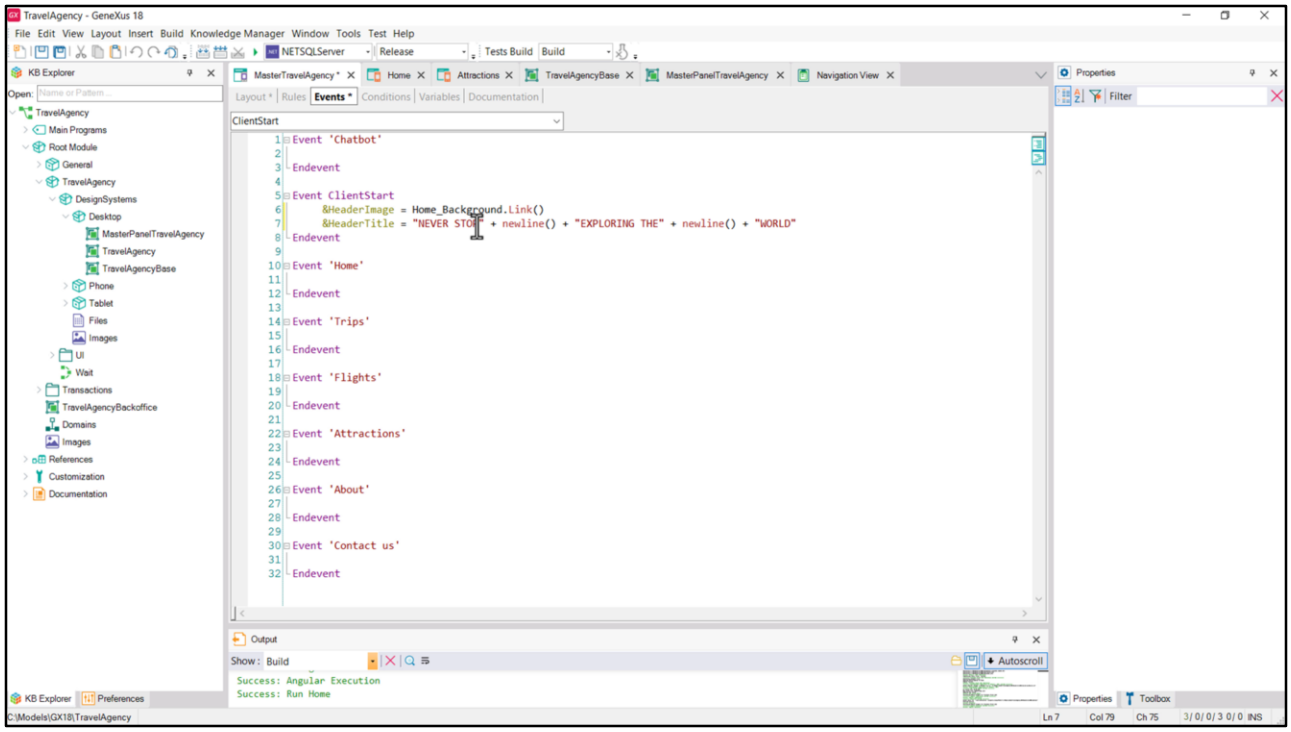
Let's assign a value to it in the ClientStart event. For now we will leave it like this, static.



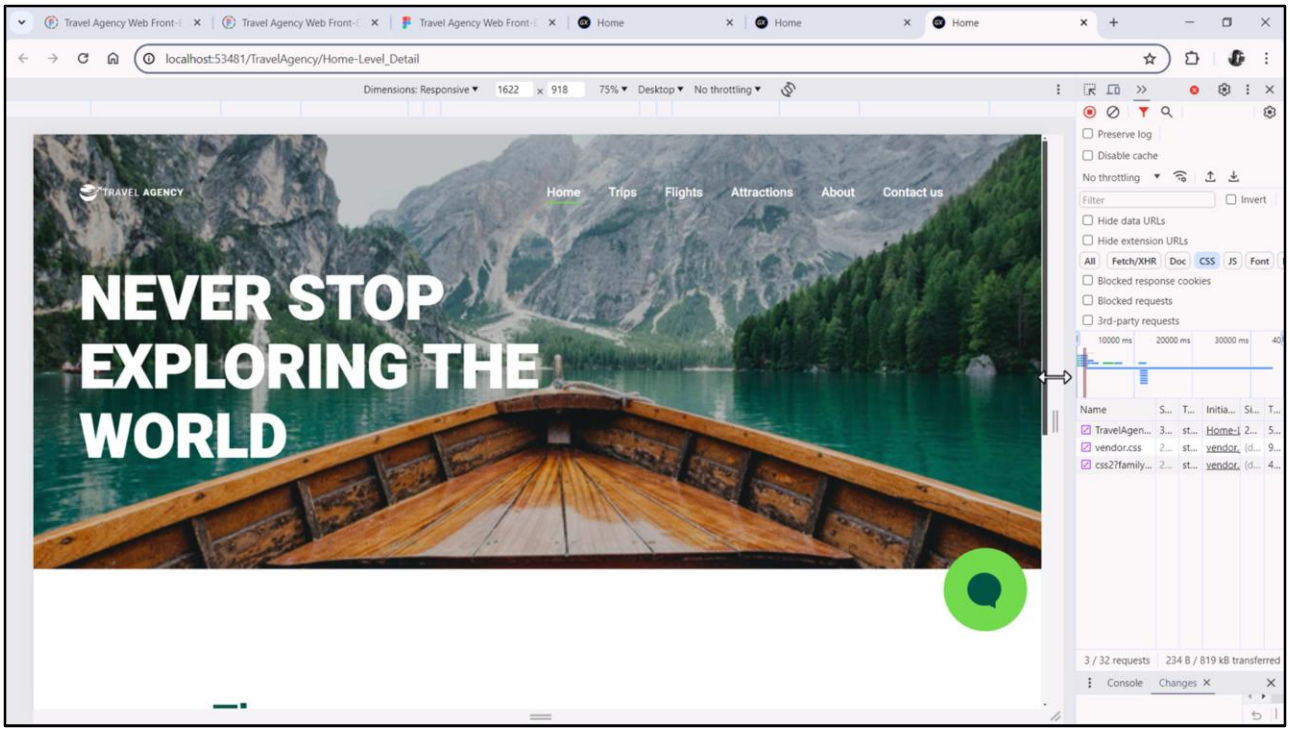
Let's give it a try. If we want to keep it always in exactly 3 lines...



...like here, and exactly these...



... adding newlines() will be enough.



Good, now we are ready to implement the navigations, managing to change the Header according to what is being loaded in the contentplaceholder each time.

We'll continue in the next video.


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