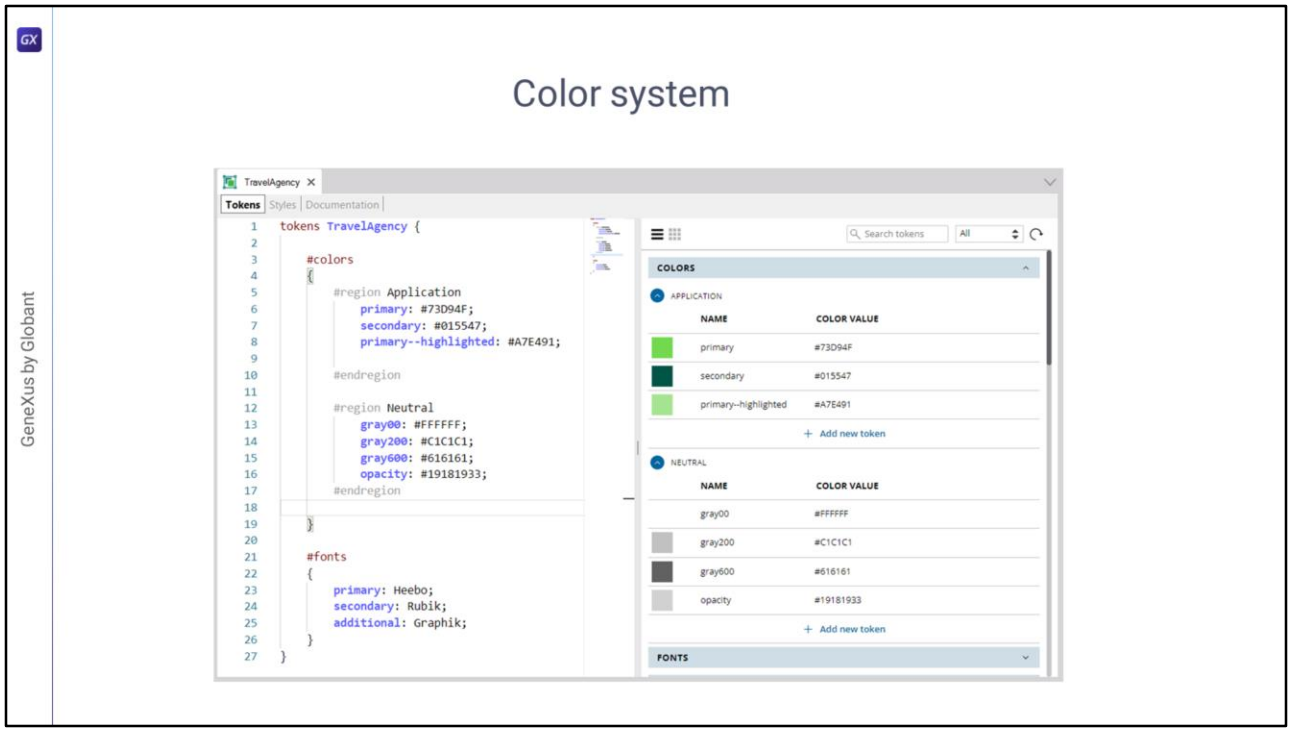
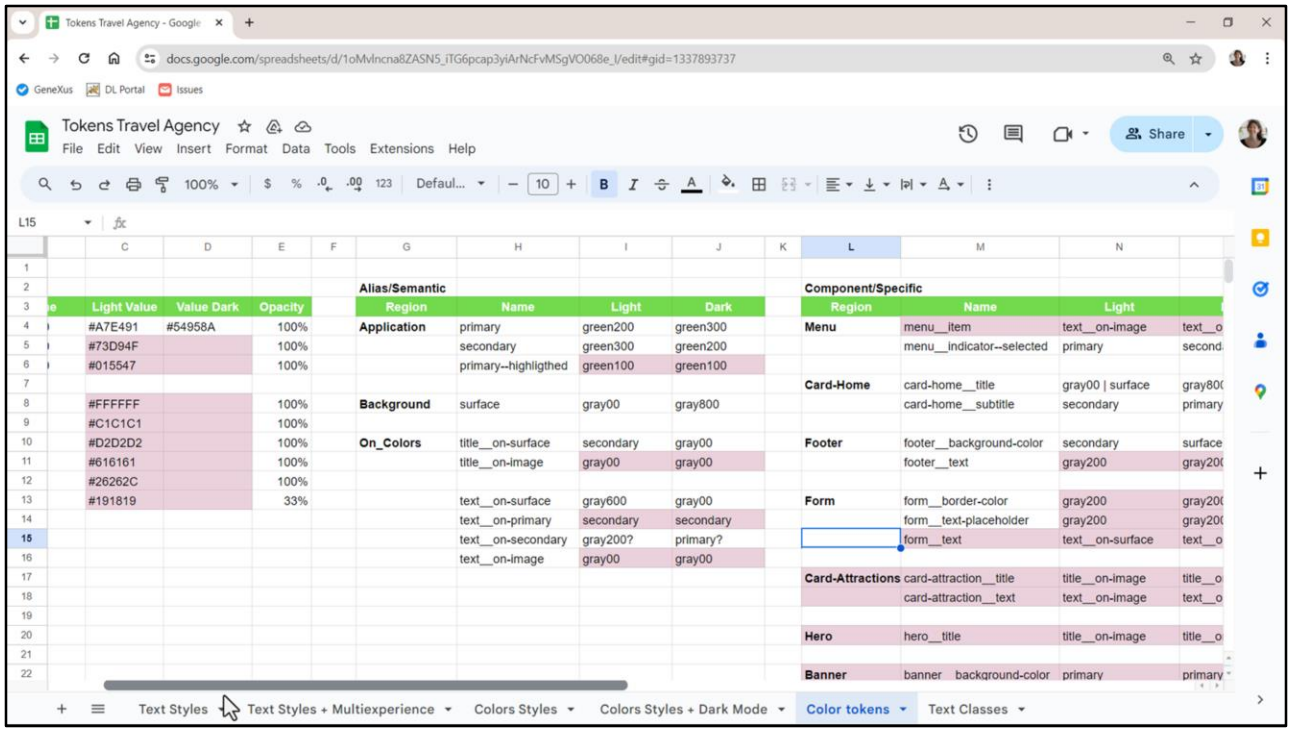
GeneXus by Globant

Color system in GeneXus

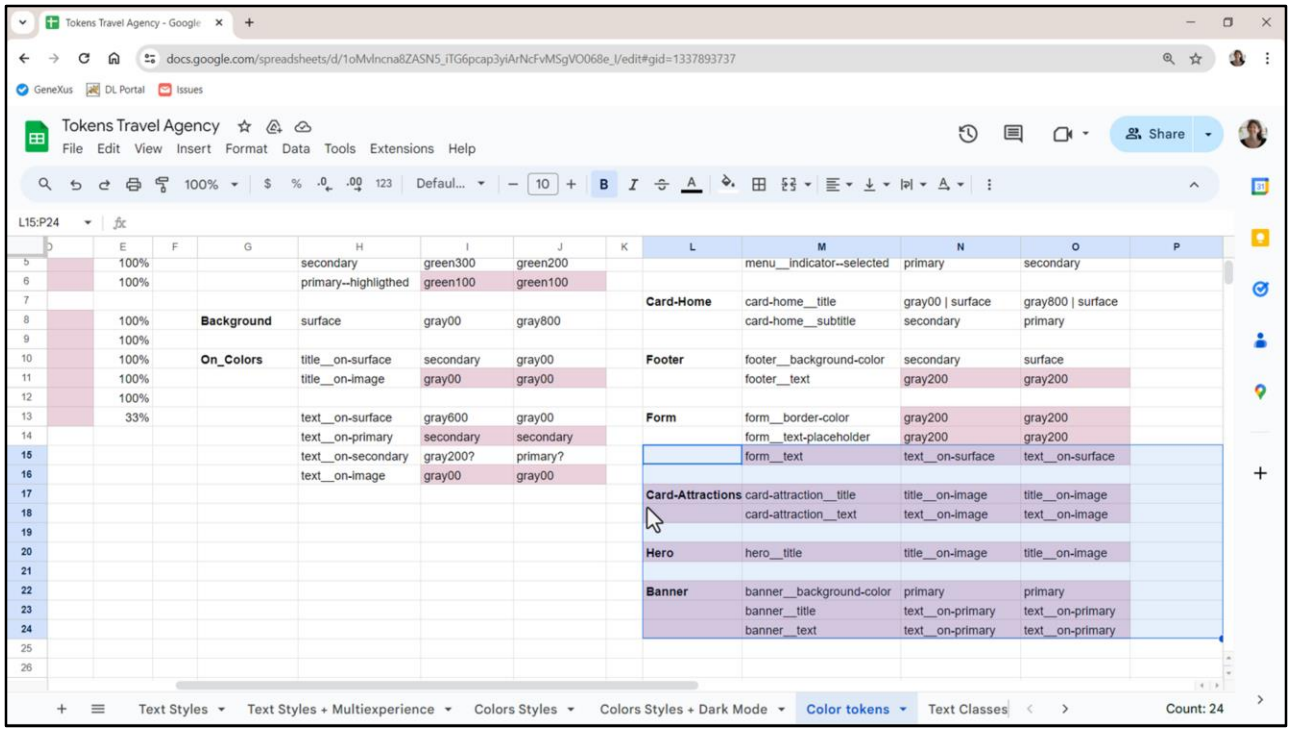
Cecilia Fernández



In the previous video, we had started from these tokens defined in our Design System, which represented the color system of our application in a too basic way. At the end of the video, we had a much more complex color system, but it was also more functionally precise, much more semantic.

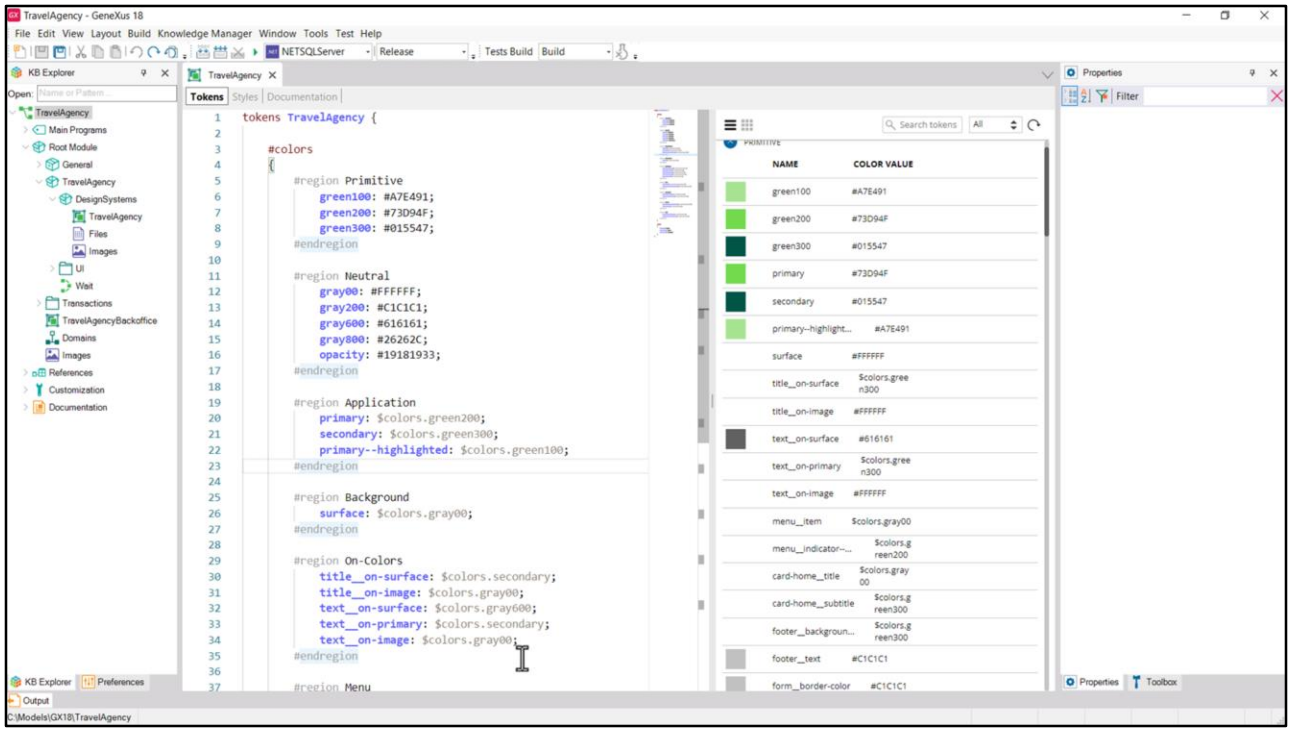


It was the one we had expressed in this spreadsheet.



5	100%			secondary	green300	green200			menu__indicator--selected	primary	secondary	
6	100%			primary--highligthed	green100	green100						
7								Card-Home	card-home__title	gray00 | surface	gray800 | surface	
8	100%		Background	surface	gray00	gray800			card-home__subtitle	secondary	primary	
9	100%											
10	100%		On_Colors	title__on-surface	secondary	gray00		Footer	footer__background-color	secondary	surface	
11	100%			title__on-image	gray00	gray00			footer__text	gray200	gray200	
12	100%											
13	33%			text__on-surface	gray600	gray00		Form	form__border-color	gray200	gray200	
14				text__on-primary	secondary	secondary			form__text-placeholder	gray200	gray200	
15				text__on-secondary	gray200?	primary?			form__text	text__on-surface	text__on-surface	
16				text__on-image	gray00	gray00						
17								Card-Attractions	card-attraction__title	title__on-image	title__on-image	
18									card-attraction__text	text__on-image	text__on-image	
19												
20								Hero	hero__title	title__on-image	title__on-image	
21												
22								Banner	banner__background-color	primary	primary	
23									banner__title	text__on-primary	text__on-primary	
24									banner__text	text__on-primary	text__on-primary	
25												
26												

We had said that these tokens, which were marked in this way and gave more specificity to the system, at the same time made it more complex, so I will not use them this time. I'll move them down here so as not to lose them.

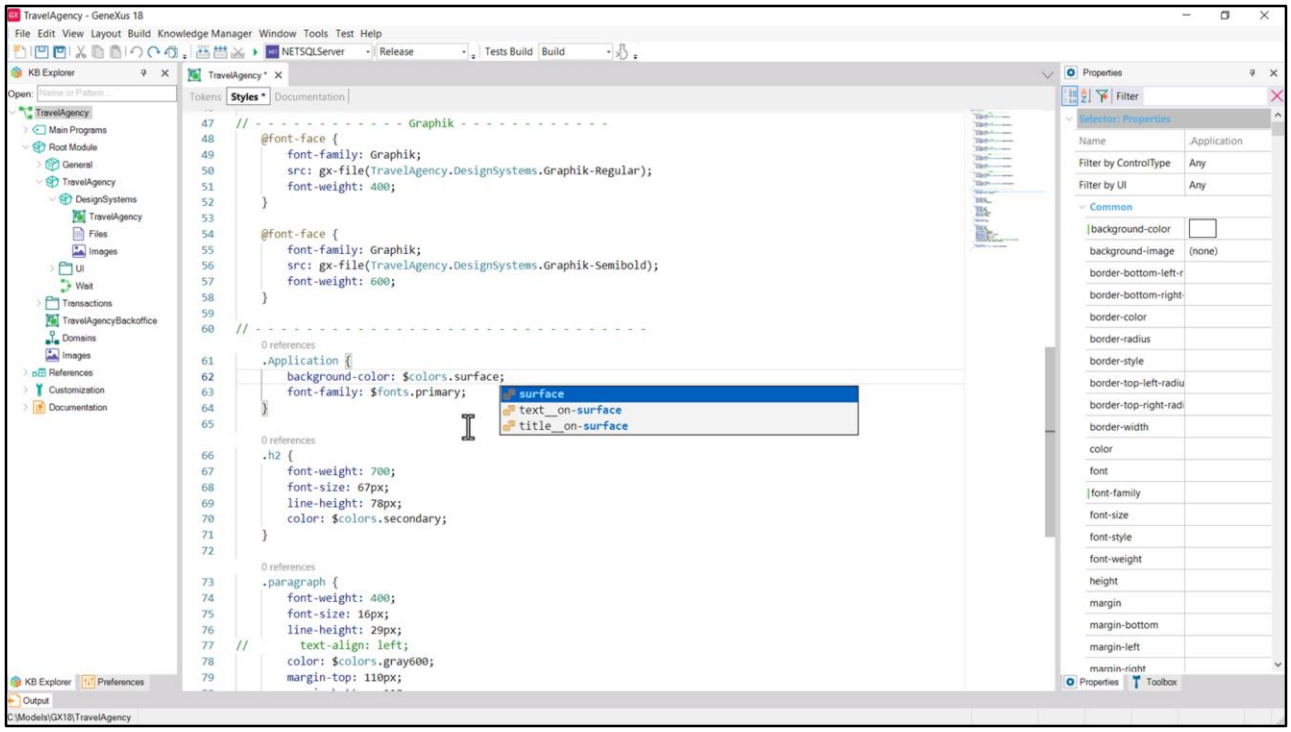


Here I have our DSO with all the tokens required to model the system, and their values for Light mode only.

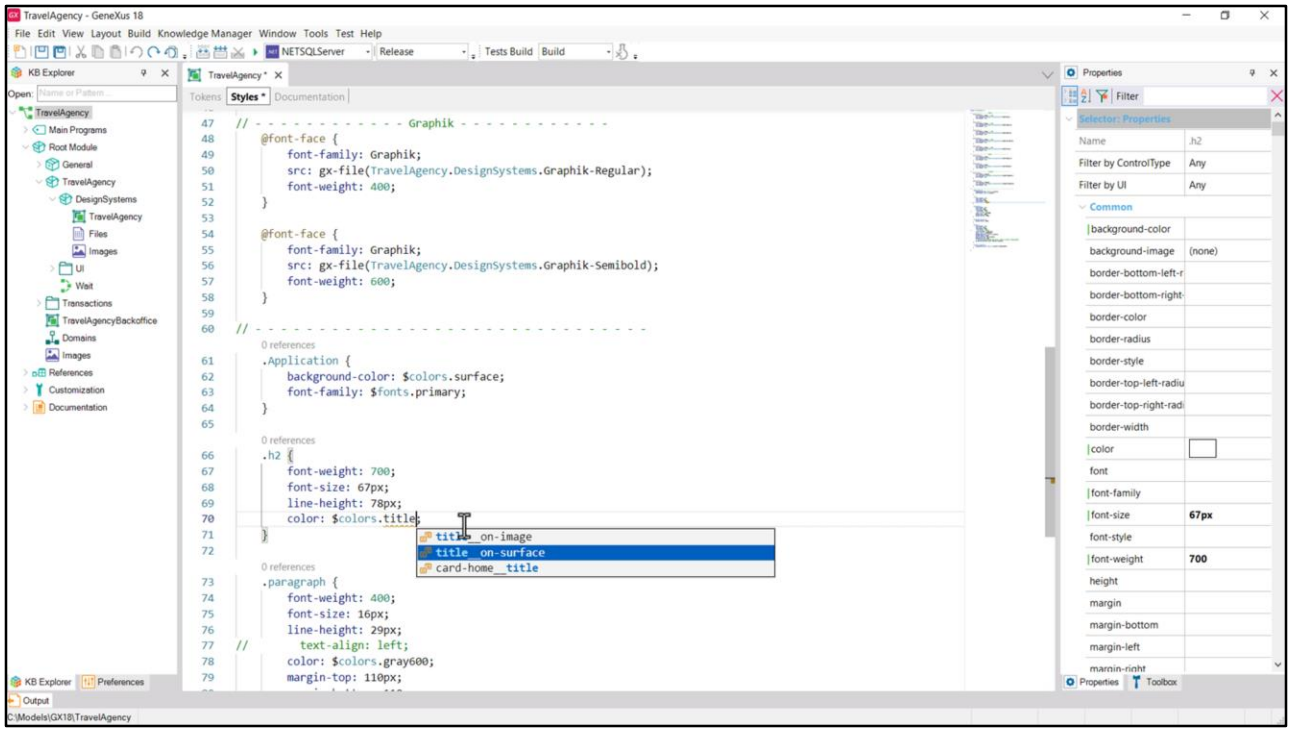
We see that those that we place in the Primitive and Neutral region have absolute hexadecimal values, while all the others' values are references to other tokens.

So far, these are things we more or less already knew. We still have to see how to represent the Dark mode.

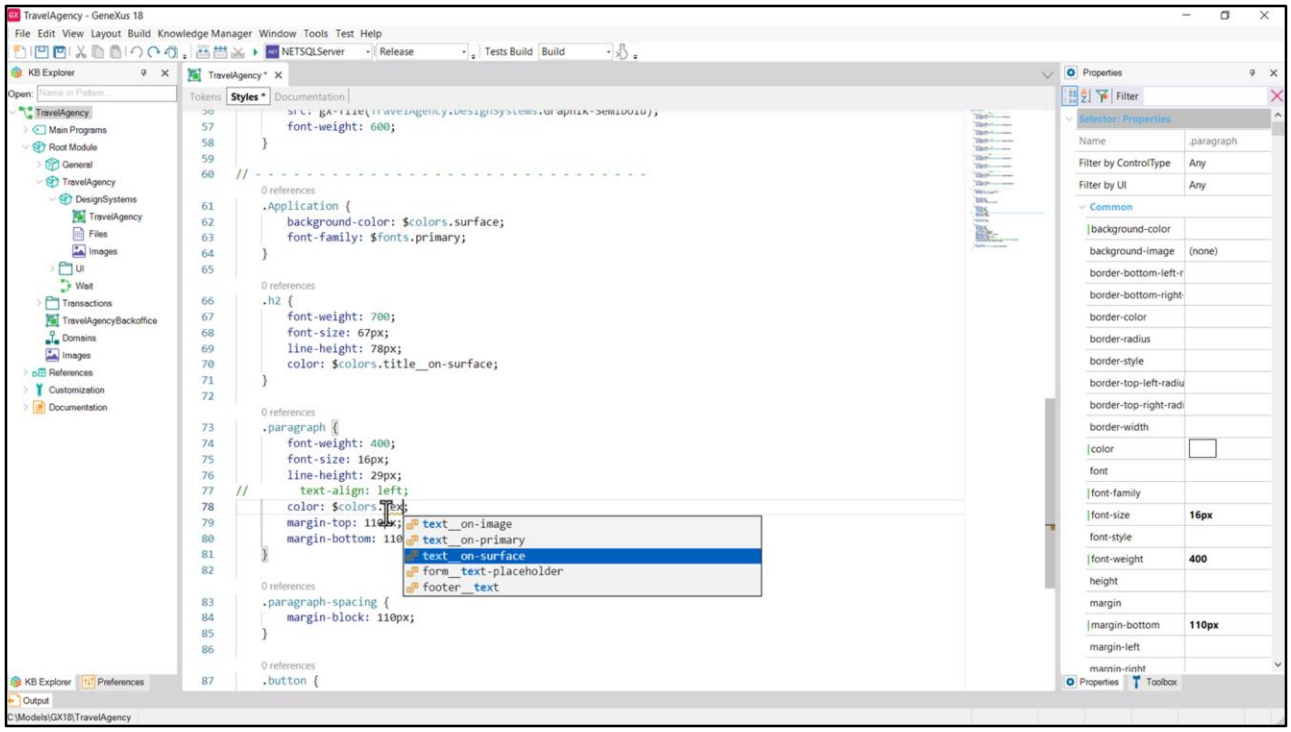
But first... let's modify the colors that we use in the classes, so that they use the new tokens. What colors did we have?



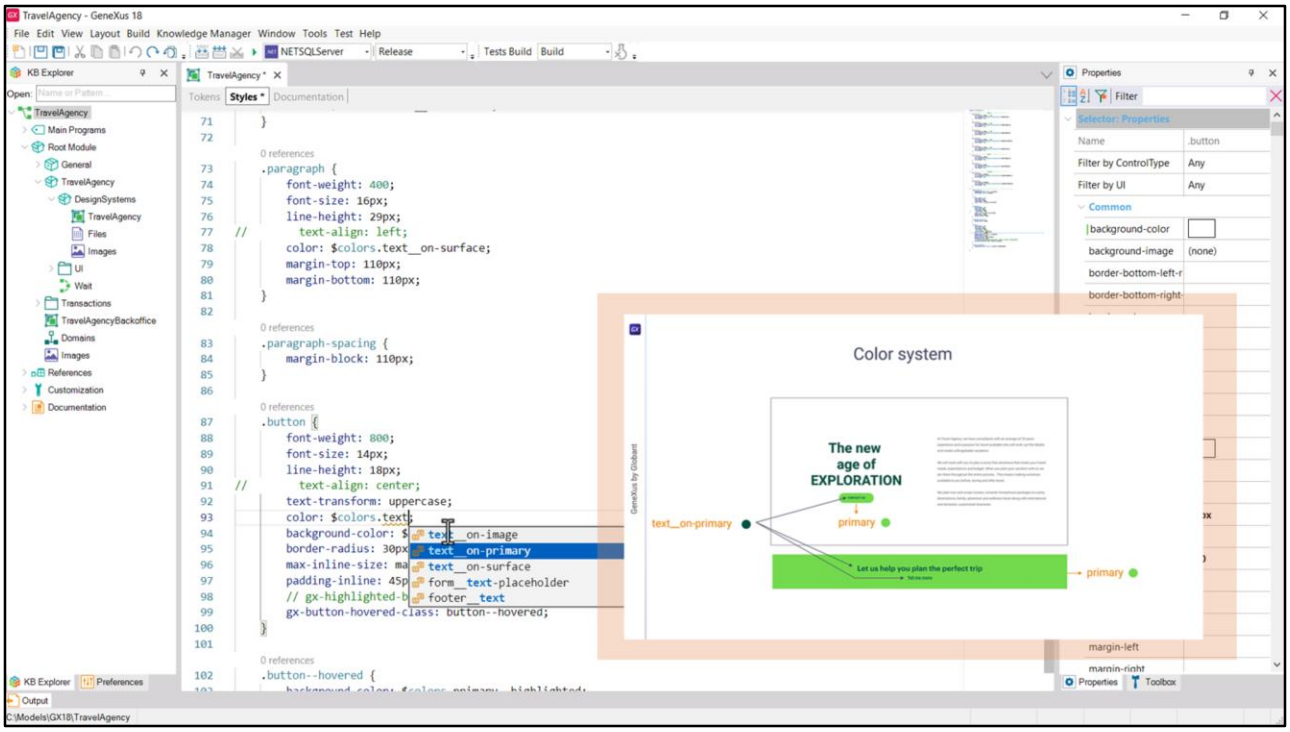
The background-color of the screens, which we had placed inside the Application class, will now correspond to the surface color token...



The h2 class is that of the title on the Surface, and its color is the token title__on-surface.



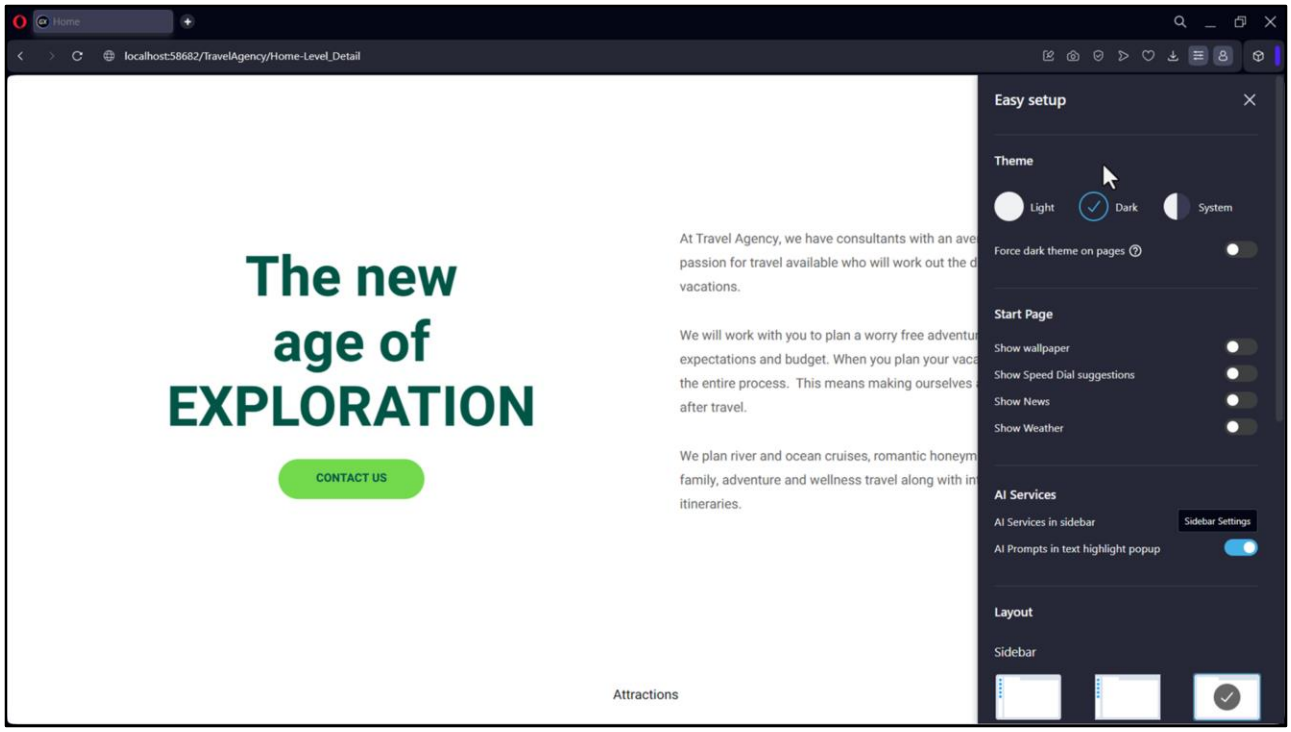
The paragraph class corresponded to the common text on the surface, so here we replace it with the text__on-Surface color token.



And for the button, we have the correct background-color, because it is primary as background color.

And the color should be text__on-primary.

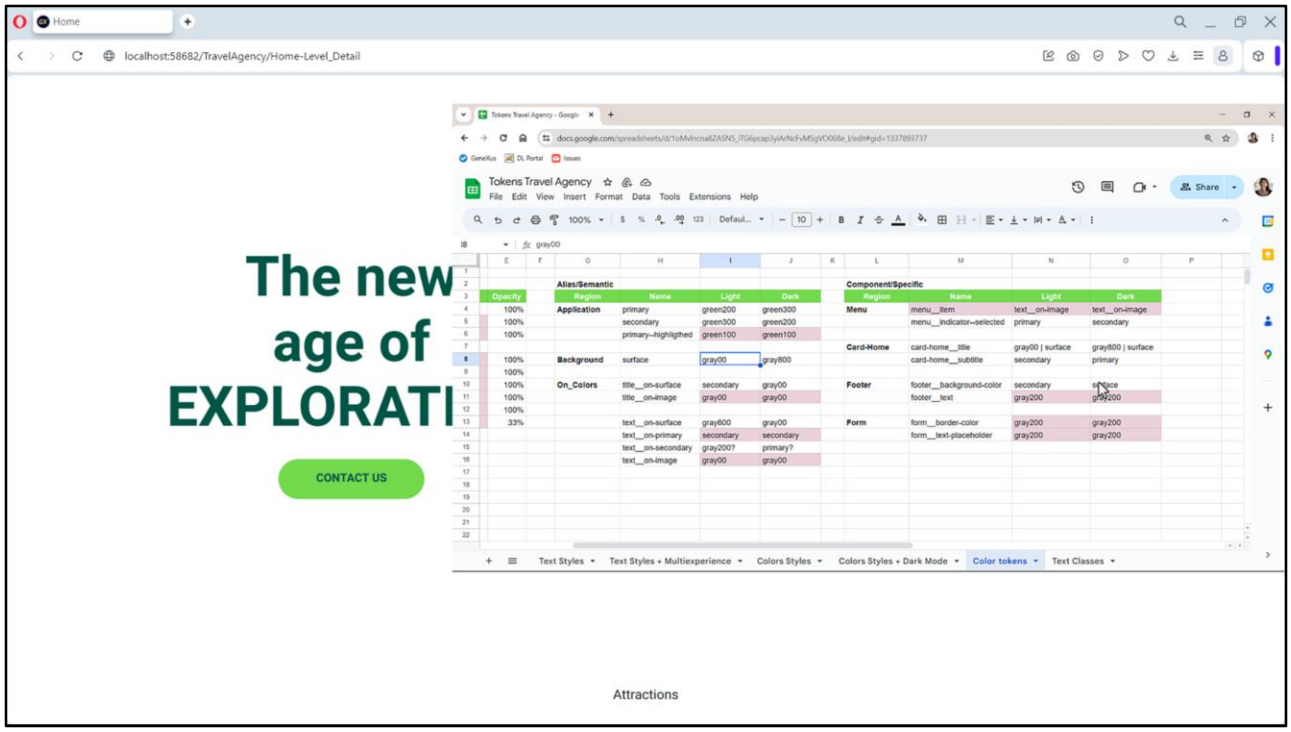
With the hover action on the button, the background-color changes to primary-highlighted, which is correct.



Let's run it to confirm that everything looks as expected.

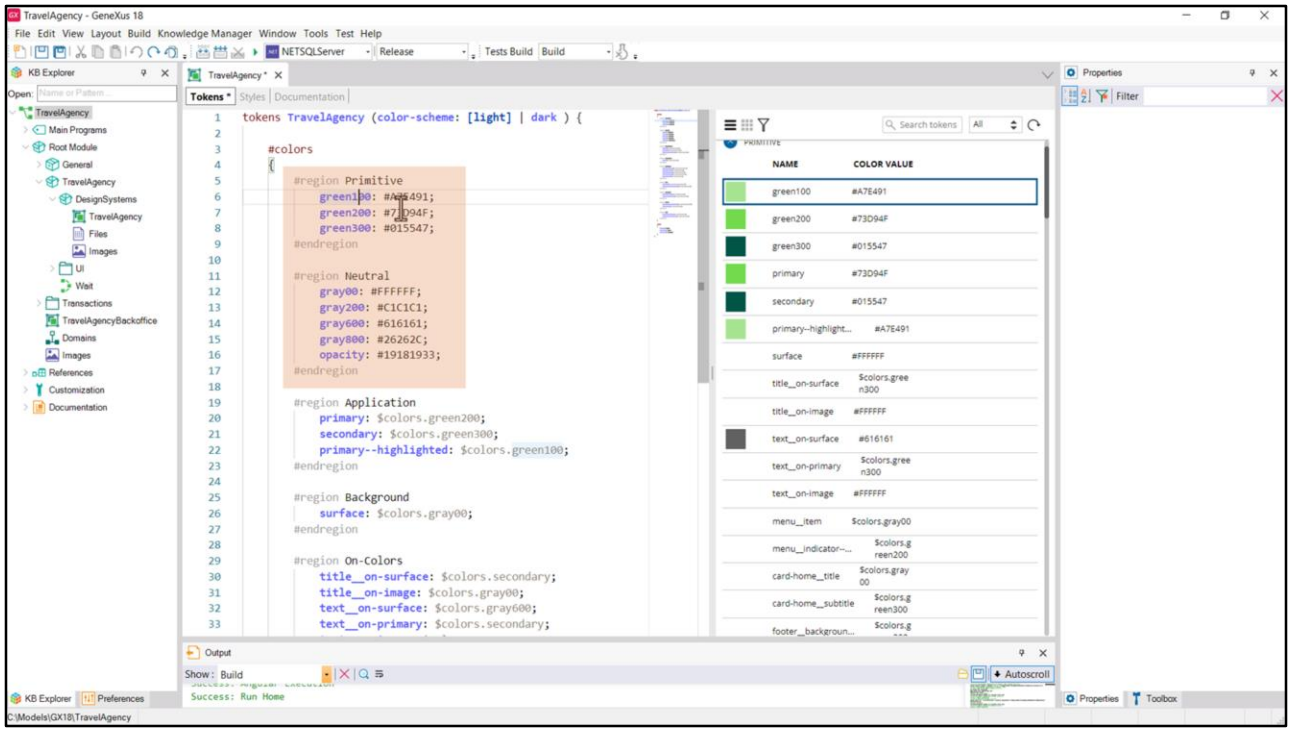
I'm going to copy the URL to run the application in the Opera browser that allows me to switch between Light and Dark mode more easily.

Note that if I choose the Dark mode, what we see is exactly the same as in Light mode.



Well, let's not delay any longer the question we are eager to answer from the beginning: how do I specify the changes in the color tokens in order to make them vary by mode?

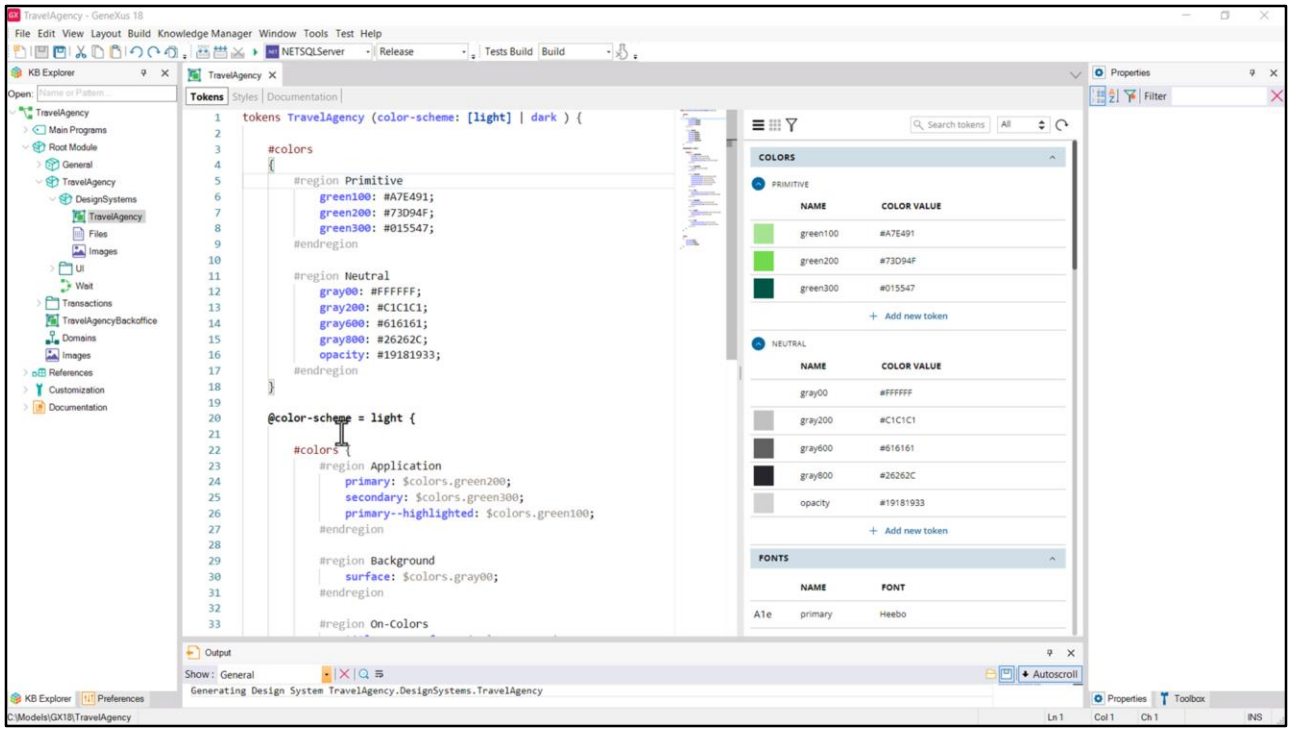
There we find the options of the set of tokens.



To this set of tokens called TravelAgency we can specify variations, and we do it in parentheses. In this case, we want to vary some of the color tokens according to the color scheme — our option will be called color-scheme — which can take two values: light and dark. We can specify with straight brackets which will be the default option.

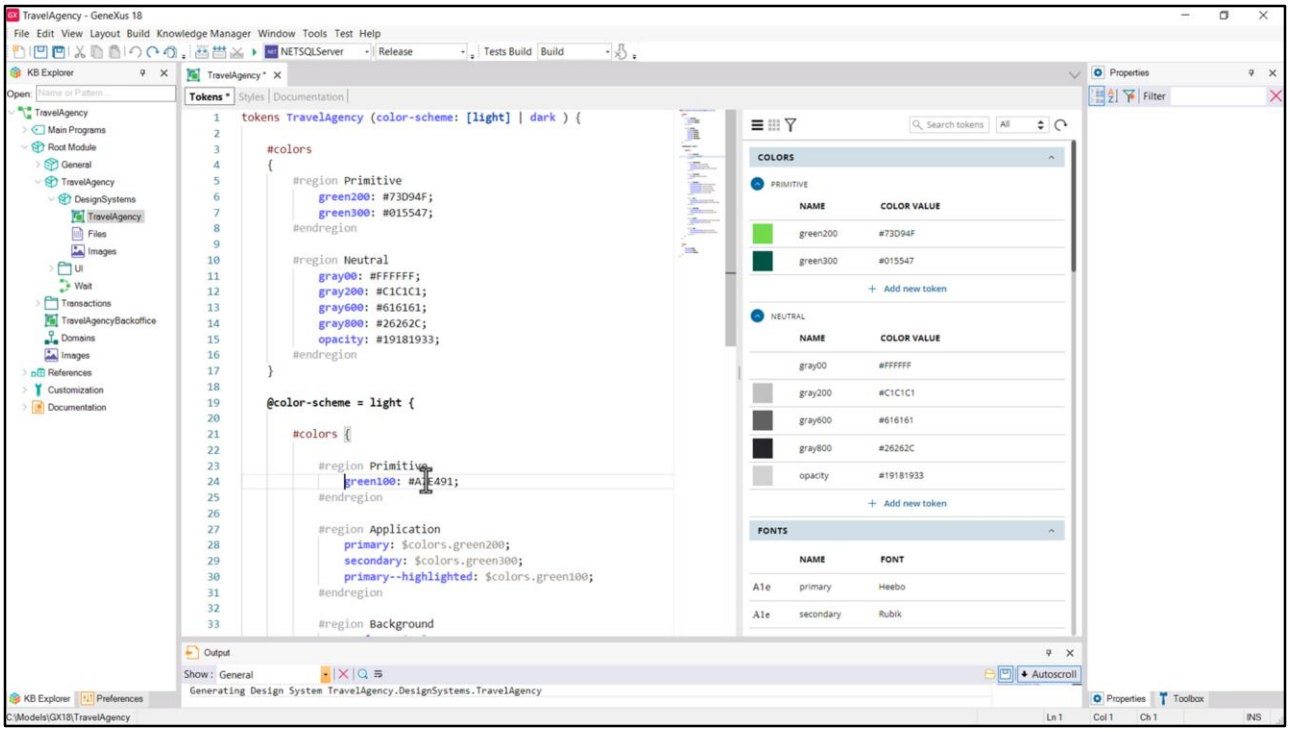
And then we still have to specify what values the tokens take on for one value of the option and for the other.

For example, the values of these tokens: for which color scheme are they valid? Only the first one varies and the others do not vary, so they would be for both. Therefore, we leave them as they are.



From there we can specify that all the ones indicated here, which are the ones in levels two and three that we've identified in our template, will be given values for both options.

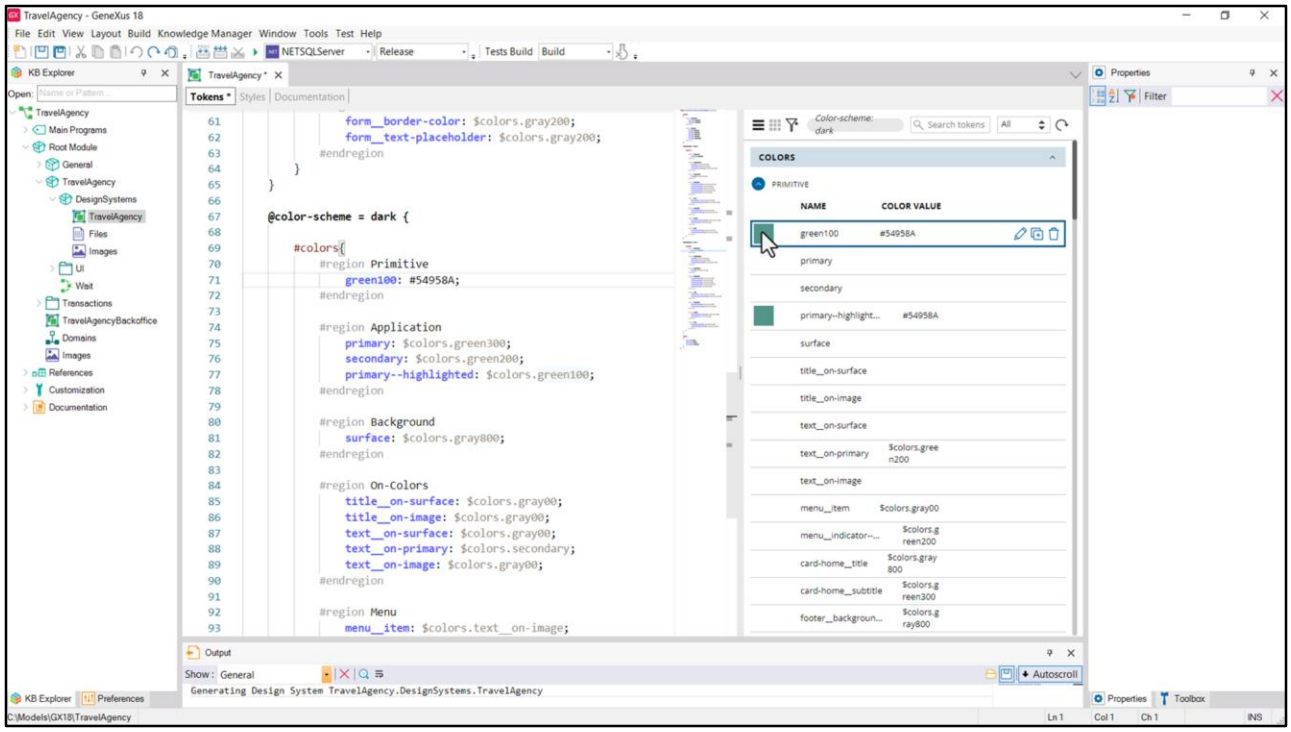
So we set all of these as assignments for the color-scheme light value... like this. Note that we have the universal color tokens, and here we have for the light option, the definition of this color token set. They will be valid, then, only when that is the chosen option.



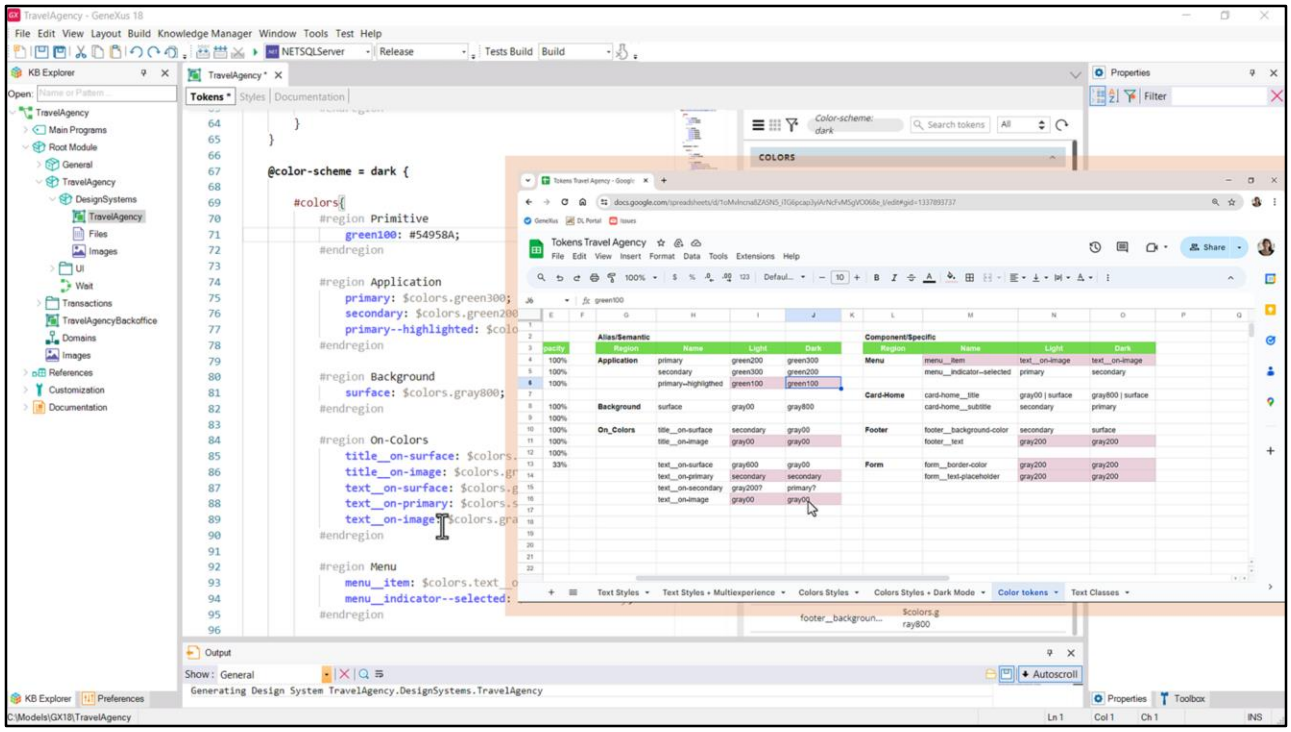
We are going to add, in here, the Primitive region. And there we're going to copy this token, because we knew that this was going to vary by mode. So this is going to be the value of the token for the light mode.

To sum up what we've done so far: we defined the color-scheme option, which takes one of these two values. The default value is this, light. Here we have color tokens that do not vary according to the mode. They will always correspond to these values. And here we are defining the color tokens when the color-scheme is light, right? We're indicating, then, the values of all those tokens.

What's left to do? Well, to define the values of the tokens when the color-scheme is dark.

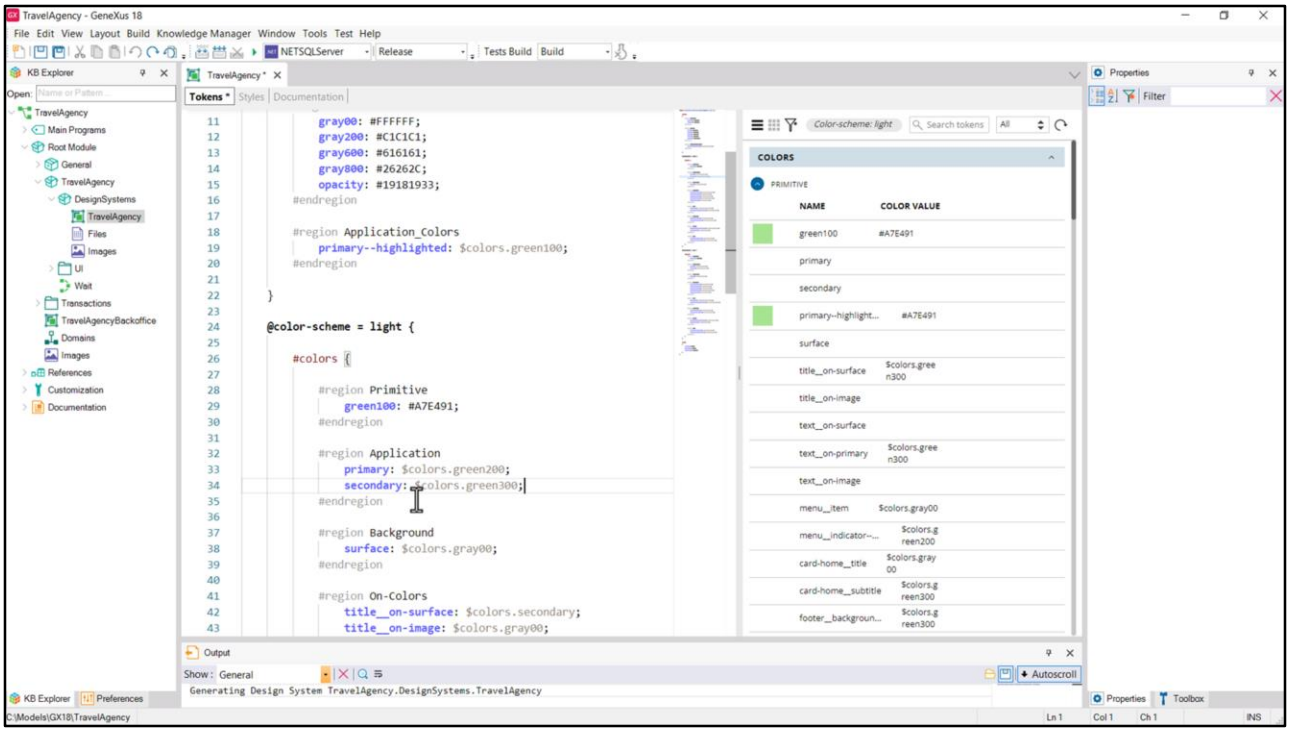


And that, I set it here below. For the color-scheme dark... what's the value of the green100 token, which we see is this other light green, primary, secondary, and all the others...



...whose values, clearly, I took from the spreadsheet, from the Dark columns of each table.

OK, here you can see the point of having marked the tokens whose values were the same in light and dark mode.

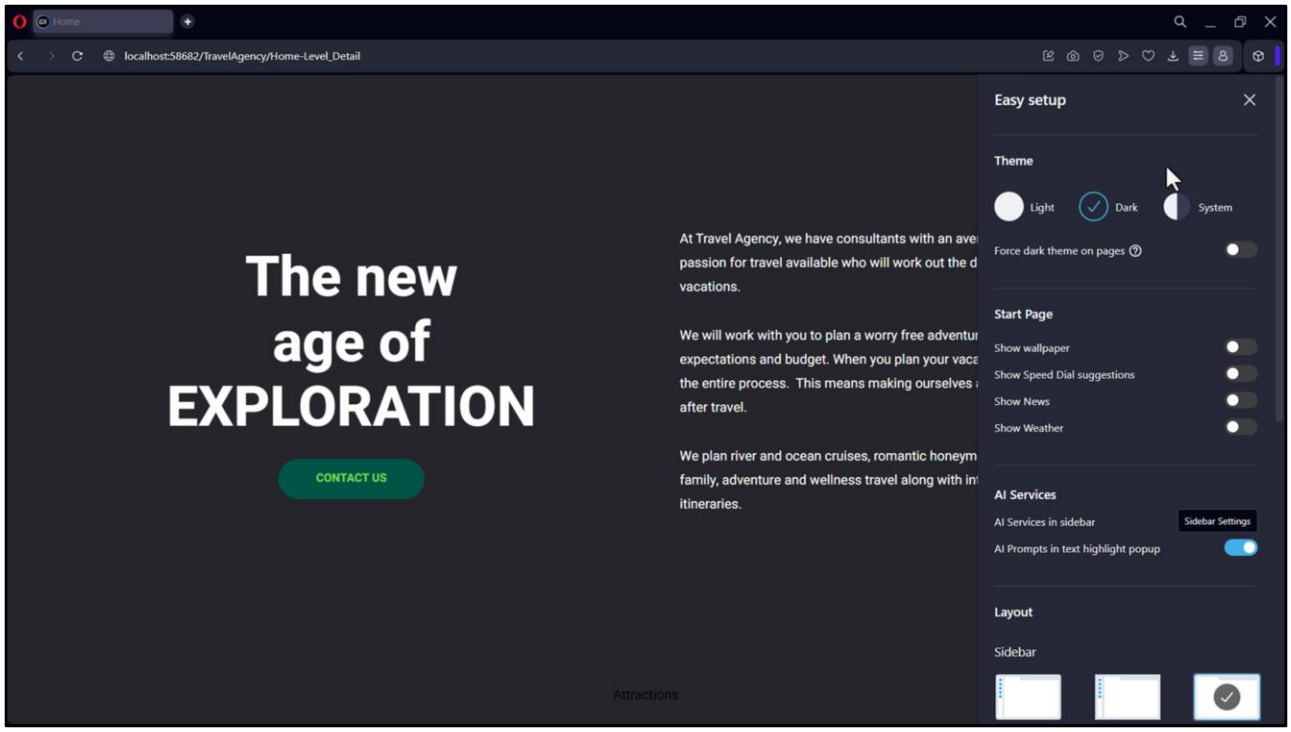


Because I could define these only once, at the beginning, where I have the tokens that don't vary, by color-scheme. So, I move, for example, primary-highlighted to here, and remove it from both sections, light and dark.

Doing it that way has advantages: there will be no inconsistencies; and it also has disadvantages: the tokens are more scattered.

For example, I will have the primary and secondary variables by color-scheme, but if I want to look for primary--highlighted I won't find it there, filtering by light or dark.

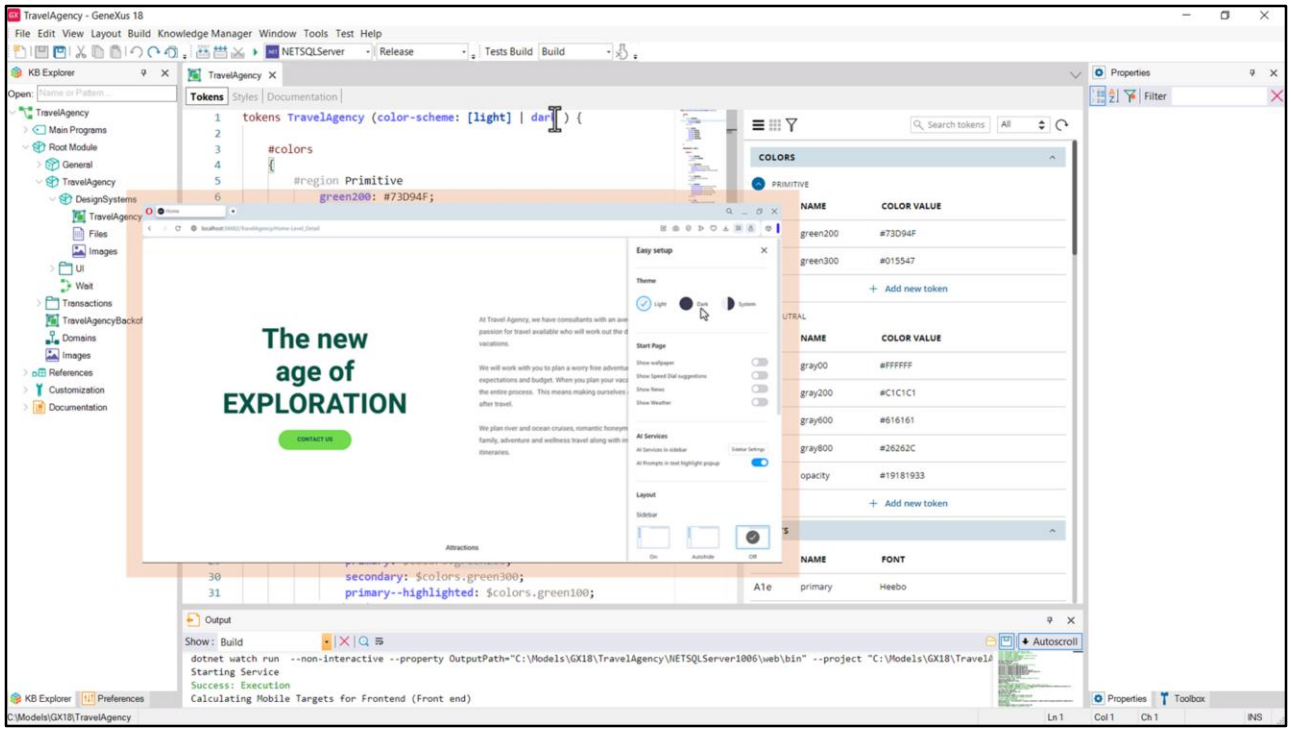
Once again, this decision is at the fontender's own risk. I'm going to leave them as I had them for now.



OK, let's try it now.

Here we have the Light mode. And we see, if we switch to Dark, how the colors were modified, just as we wanted.

If we now go back to Light...

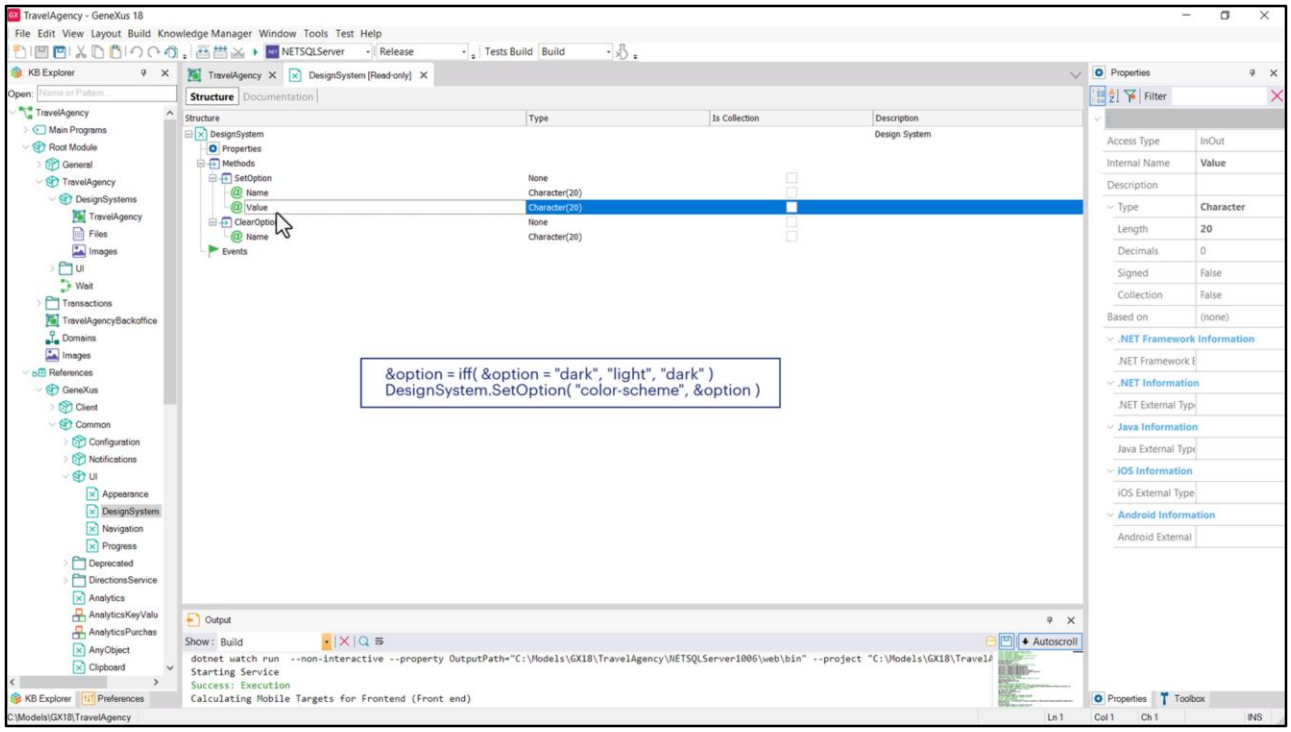


At this point it would be convenient to make some remarks.

Let's start with the simplest thing: this, which looked like magic, actually corresponds to a correct systematization where we assigned, to the color properties of the classes, both background-color and color, semantic color tokens by function. And we vary those color tokens by color scheme: light or dark.

In addition, this worked just like that, without us having to specify anything at the programming level because the browser already understands that what it calls light and dark theme there corresponds exactly to the values that we call light and dark, for the option that we call exactly color-scheme in our DSO.

The reason is that we gave it exactly that name. It is not mandatory, we could have called the option in any way, but if we had done so, the browser would not have understood that it is the same thing, and would not have changed the colors as it did.



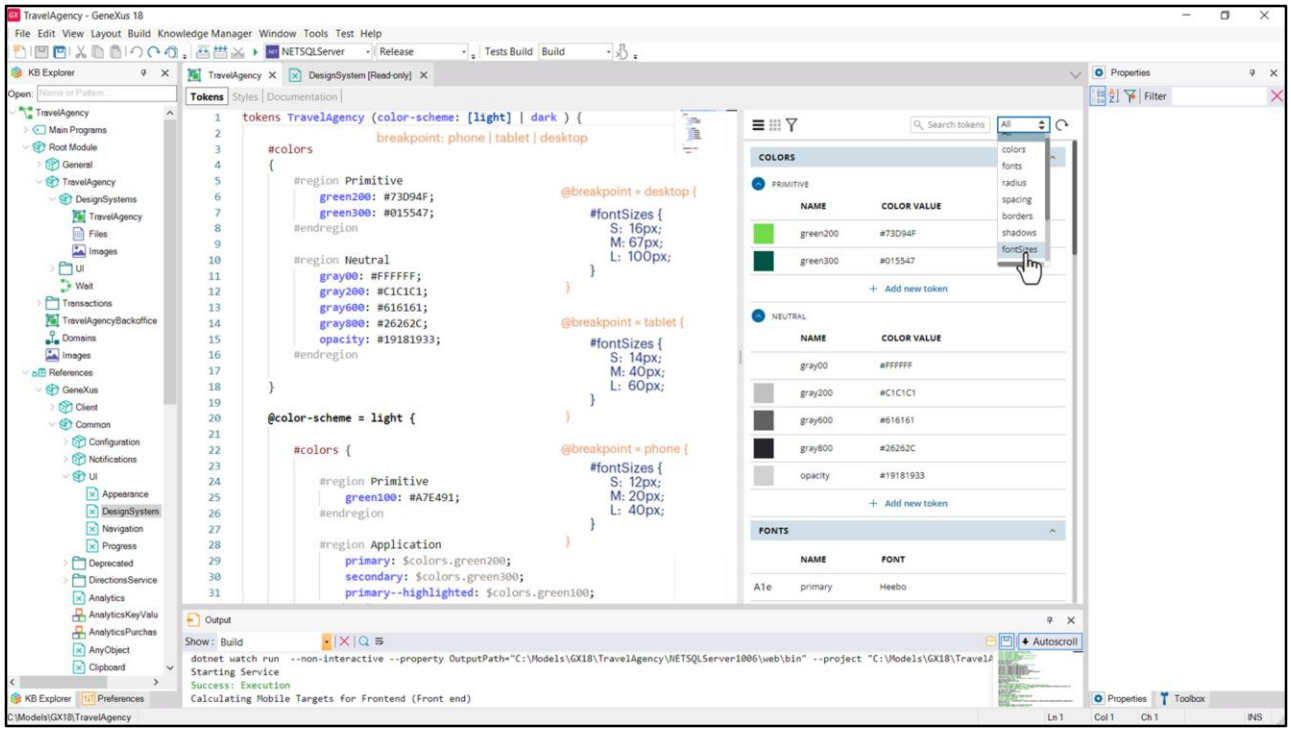
We have, however, a way to specify at the programmatic level which of the option values we want to apply at a given time.

It will be helpful, for example, if we want to give the user the option at the application level, and not at the browser level, to choose the color scheme.

For this purpose, the DesignSystem external object is included in the GeneXus module that comes installed with every KB.

When we open it, we can see a SetOption method offered precisely to set one of its possible values for a particular option (in this case, the only one we have is the color-scheme).

So, for example, we could enter this code in the event associated with a layout control for modifying the color-scheme. If in the variable that we inserted, option, there was a "dark" value and it is changed to "light" and vice versa (that is what the iff command does), and then the "color-scheme" option of the DSO associated with the object is set.



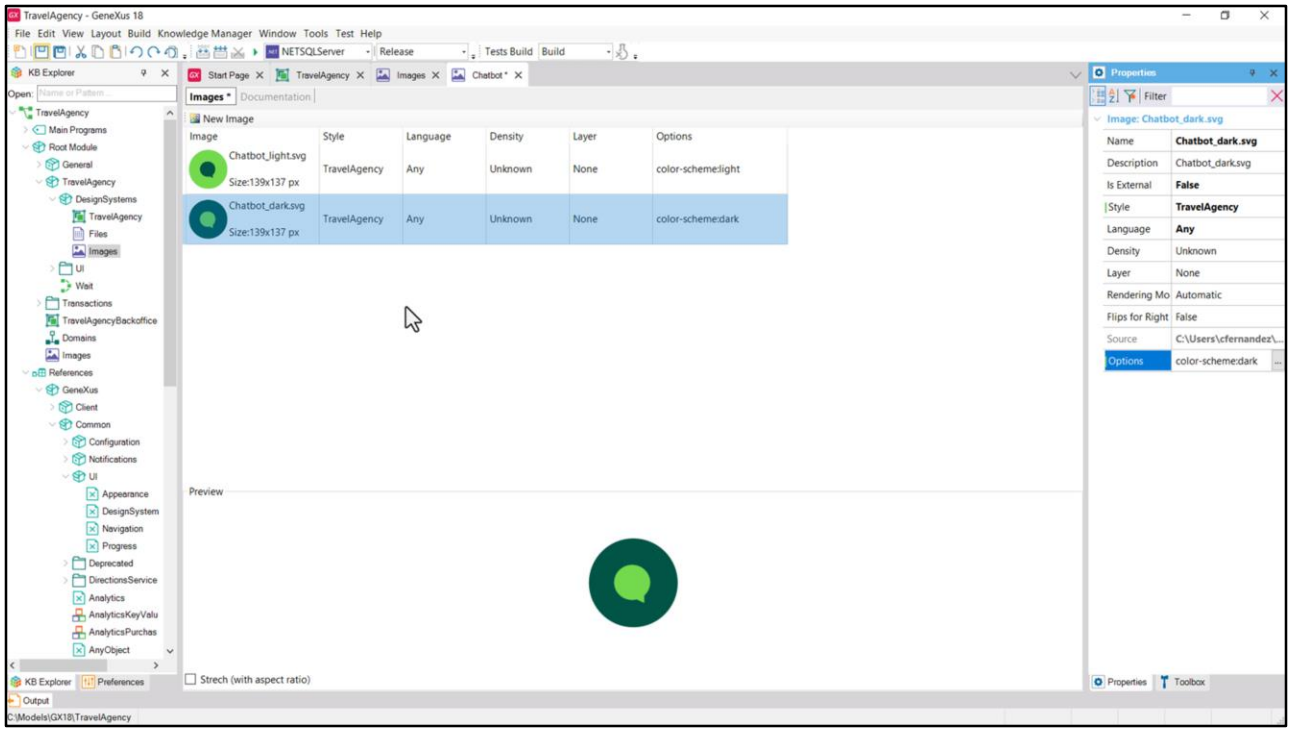
Of course, this means that we can have several options per DSO, not just one, and it doesn't have to be only for varying color tokens.
For example, we could add an option besides this one, separating it with a comma, or instead of this one, to vary the tokens by screen size.

Consider, for example, if we had spacing or font size tokens defined. Suppose we had S, M, L tokens for font sizes. For the different sizes, Small, Medium and Large.

We might want to vary their values according to the screen size, for which we can define a breakpoint option that takes the values phone, tablet and desktop, and from there define that for the Desktop breakpoint these are the values of these 3 tokens, but these other ones for the other breakpoints.

Is it convenient to use tokens for the font sizes in our system?

We will analyze that in the next video.

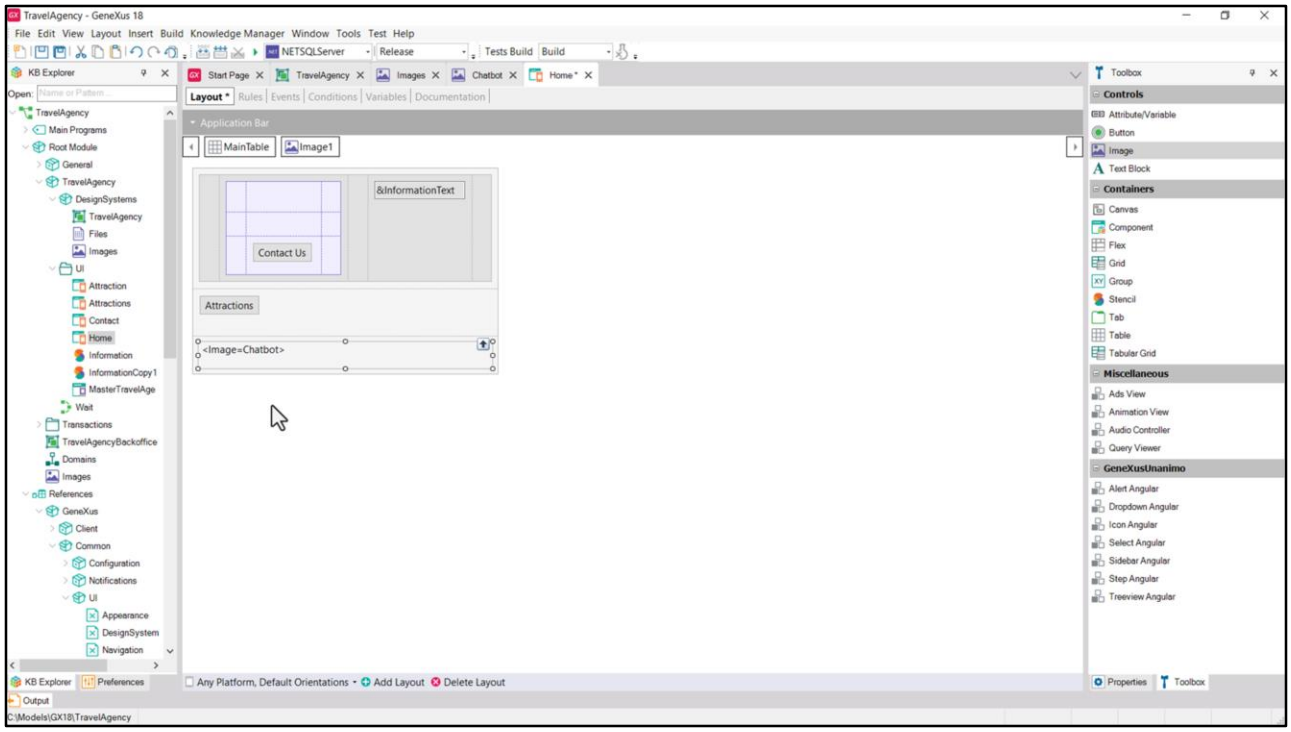


What is left for me to say now is that since this option appeared in our DSO, we can understand that dimension that appeared at the image level.

If I now associate our DSO to both variations as Style... let's begin with this first one... we see that the Options property appears. It will offer me to give a value to each one of the options that this DSO has specified. In this case for now we only have one, color-scheme.

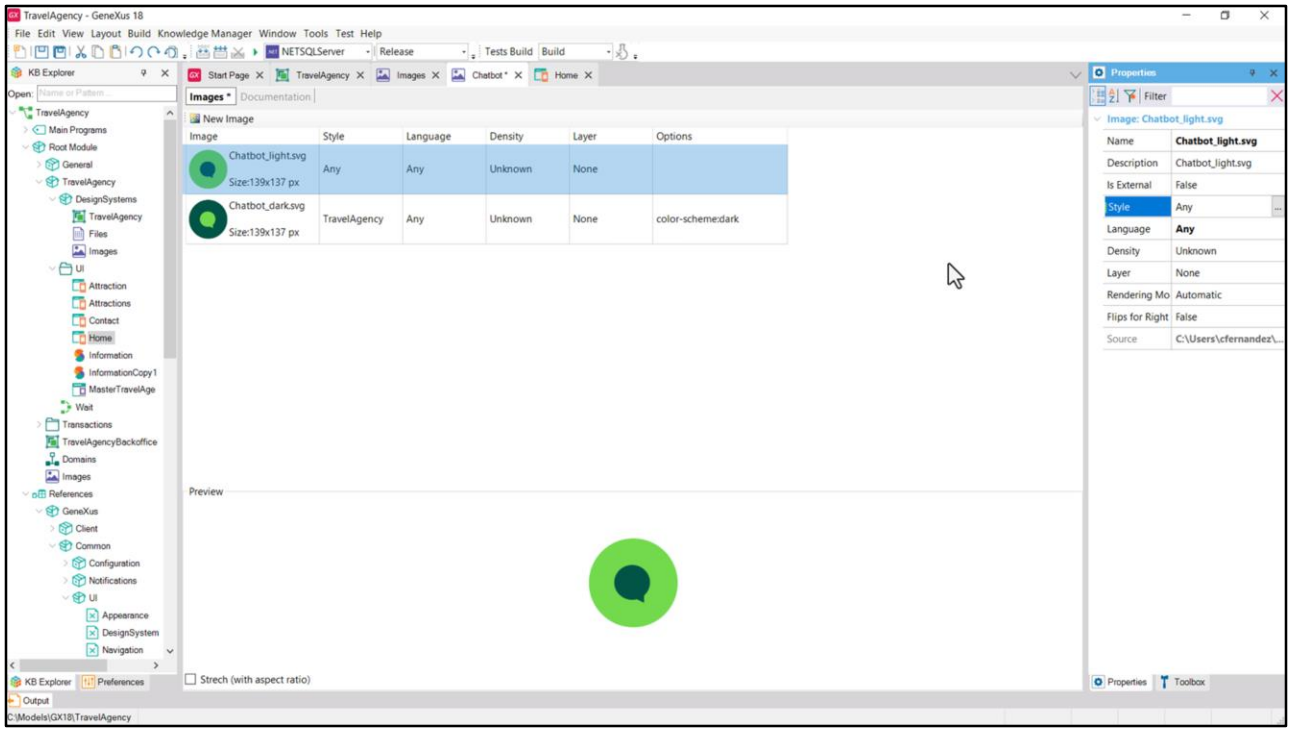
So we will say that the Chatbot image will correspond to this variation only if the color-scheme is Light.

And to this other one instead if the DSO is also TravelAgency, but with Dark color-scheme.

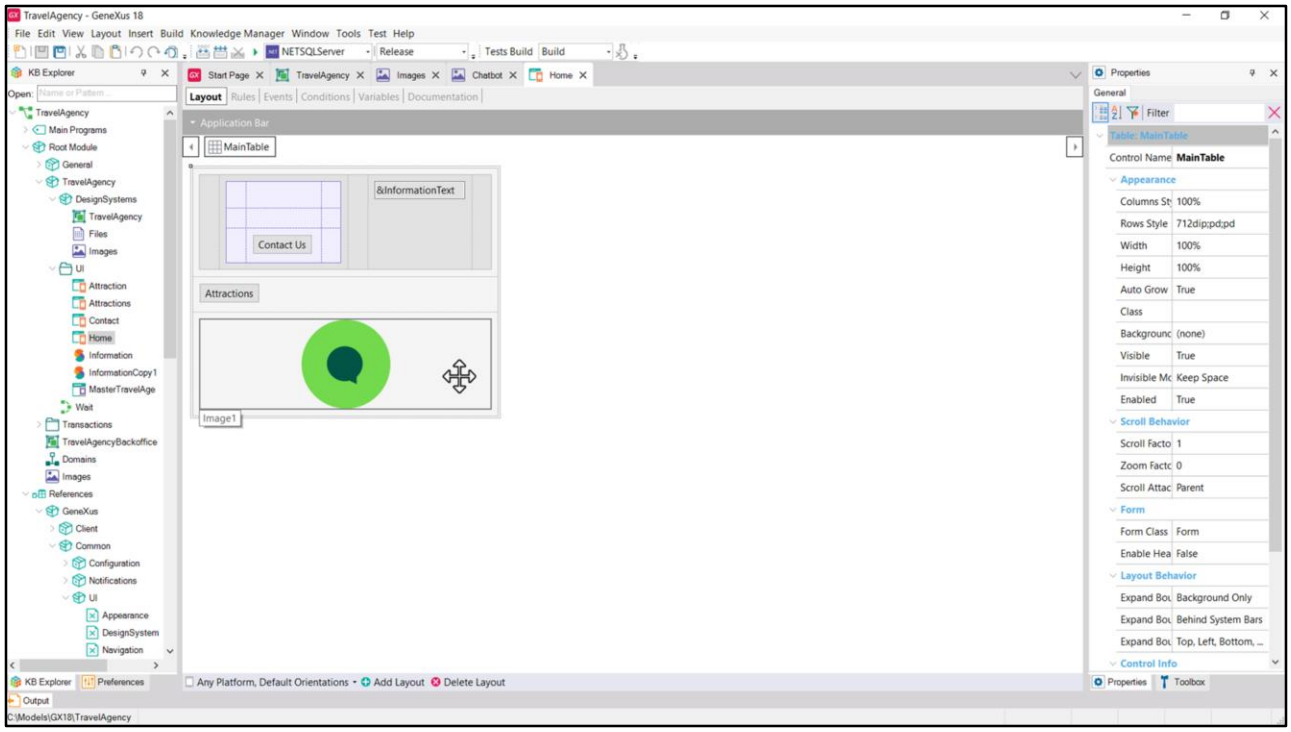


We haven't implemented the Master Panel yet, which will be where we'll have the button with this image, but we could, just to try this at runtime, insert a single image here....

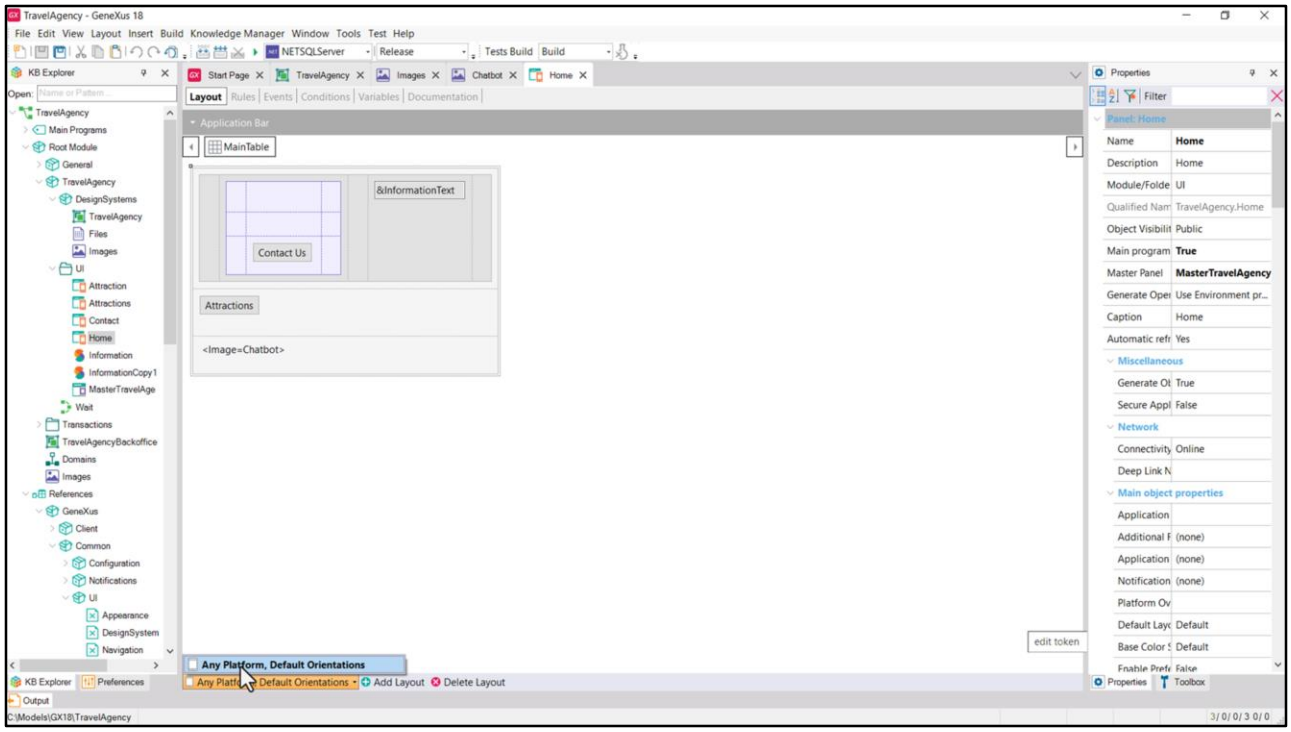
It's not showing the image called Chatbot, we don't see it... it says that's the image that will be rendered, but we can't see it. And saving doesn't change anything.



Let's see what happens if I remove the Style from one of the variations of the image... I leave the Any value, so that it is valid for any DSO..... I save...

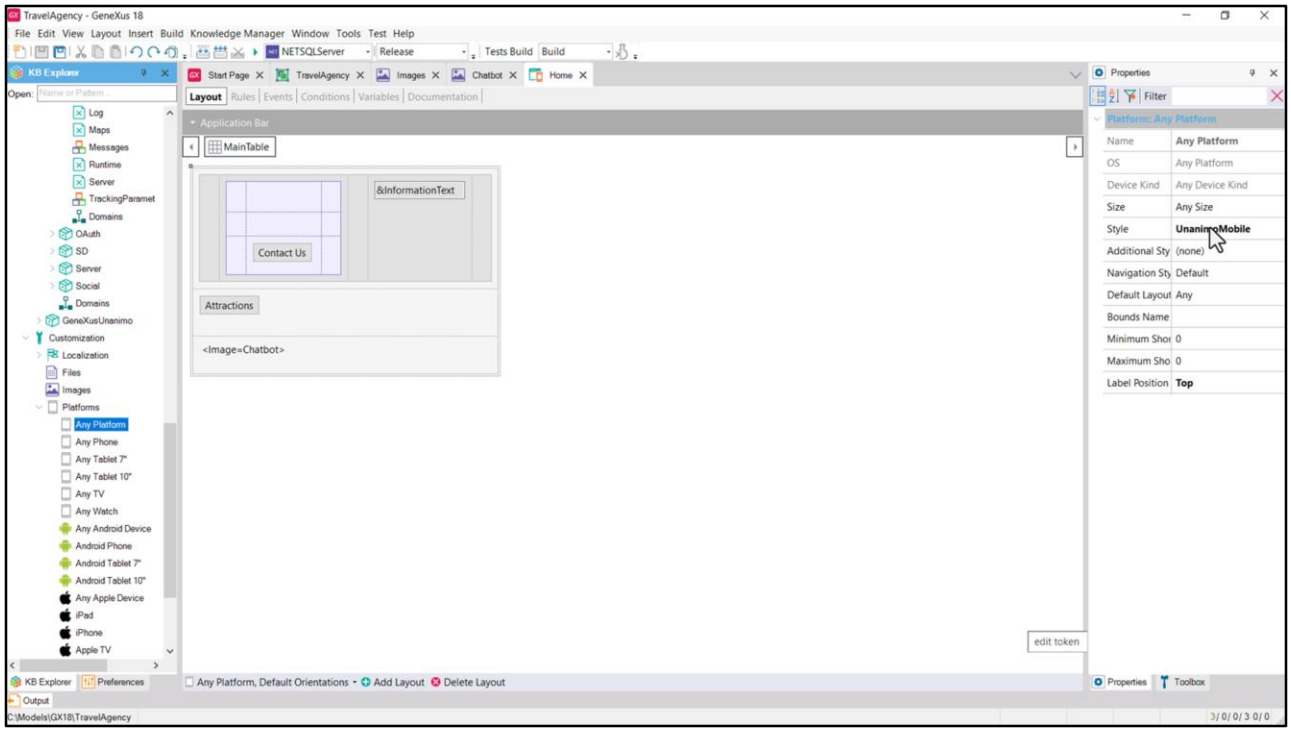


... and now I go back to the panel, close it and open it again. Now I see it!

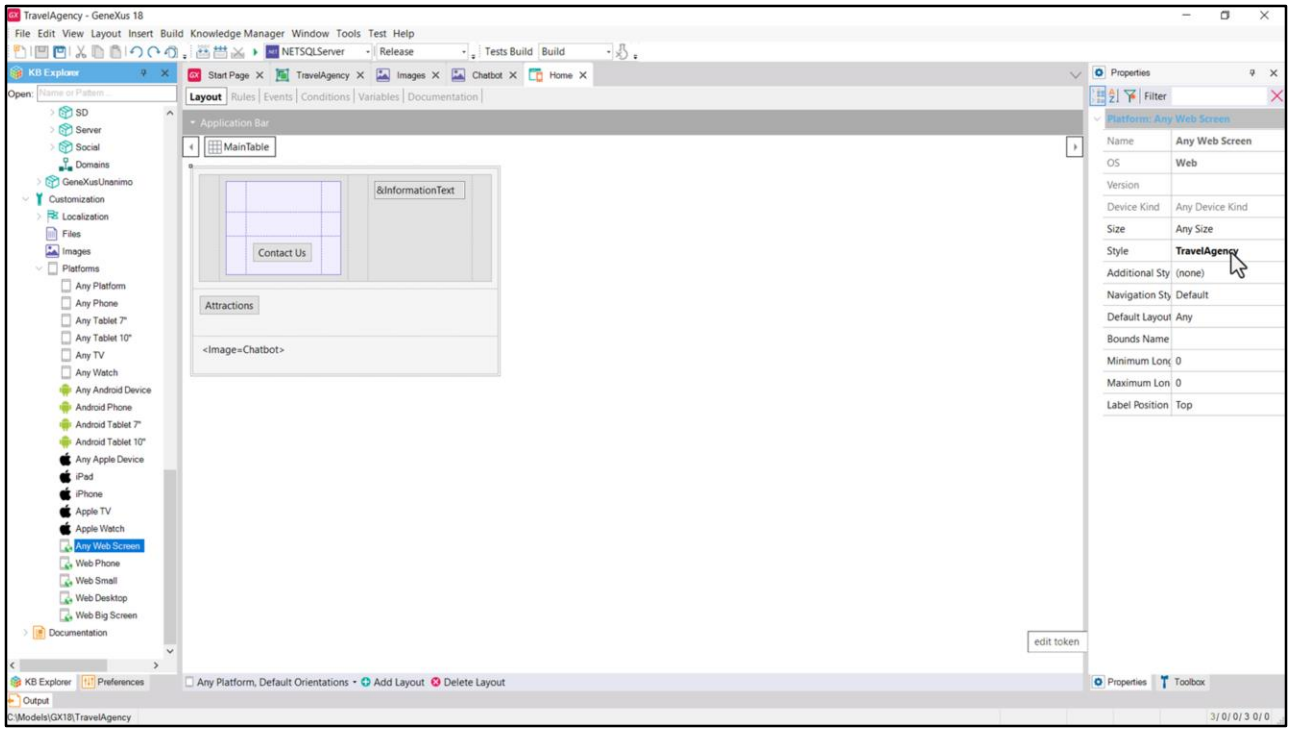


What if I limit it again to the DSO only and in Light mode...? I close and open again, and there it is again!

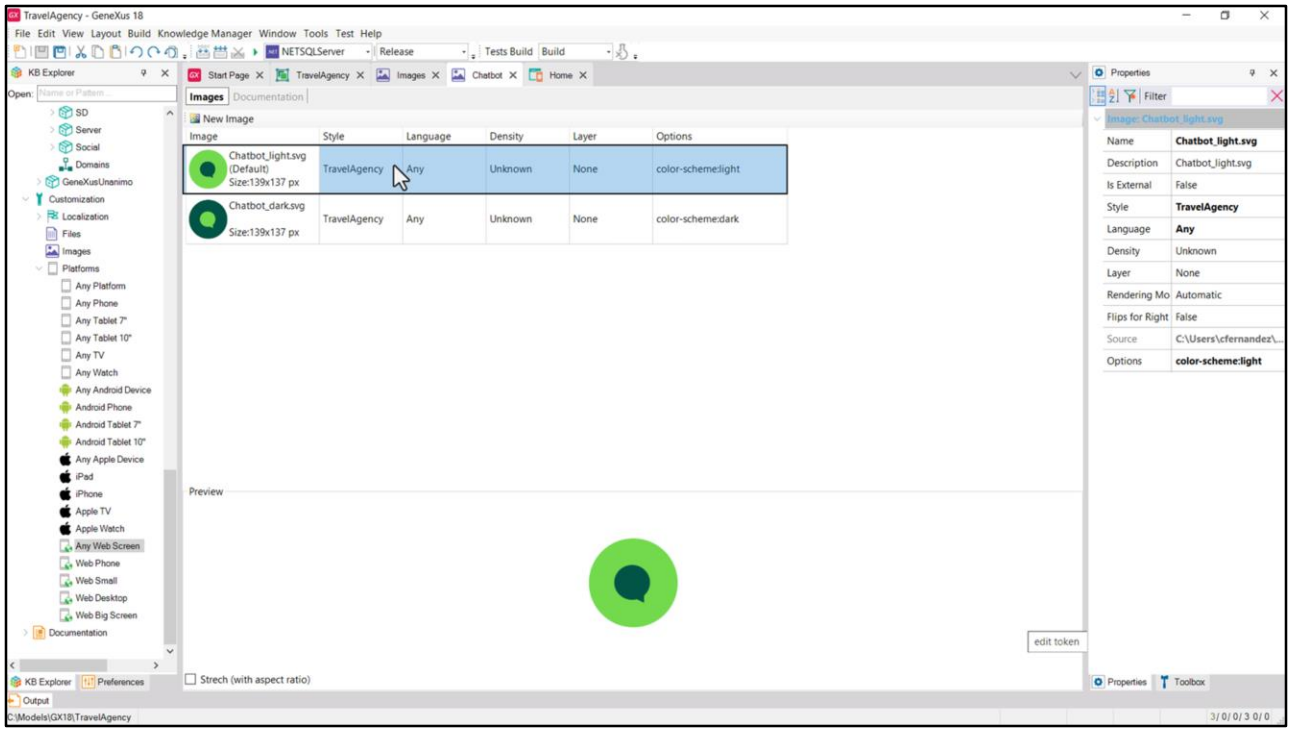
Well, this makes sense because this Panel object works for all platforms, not just for Angular where we are developing.



And if we come to see the DSOs associated with Any Platform, we see that it is UnanimoMobile.

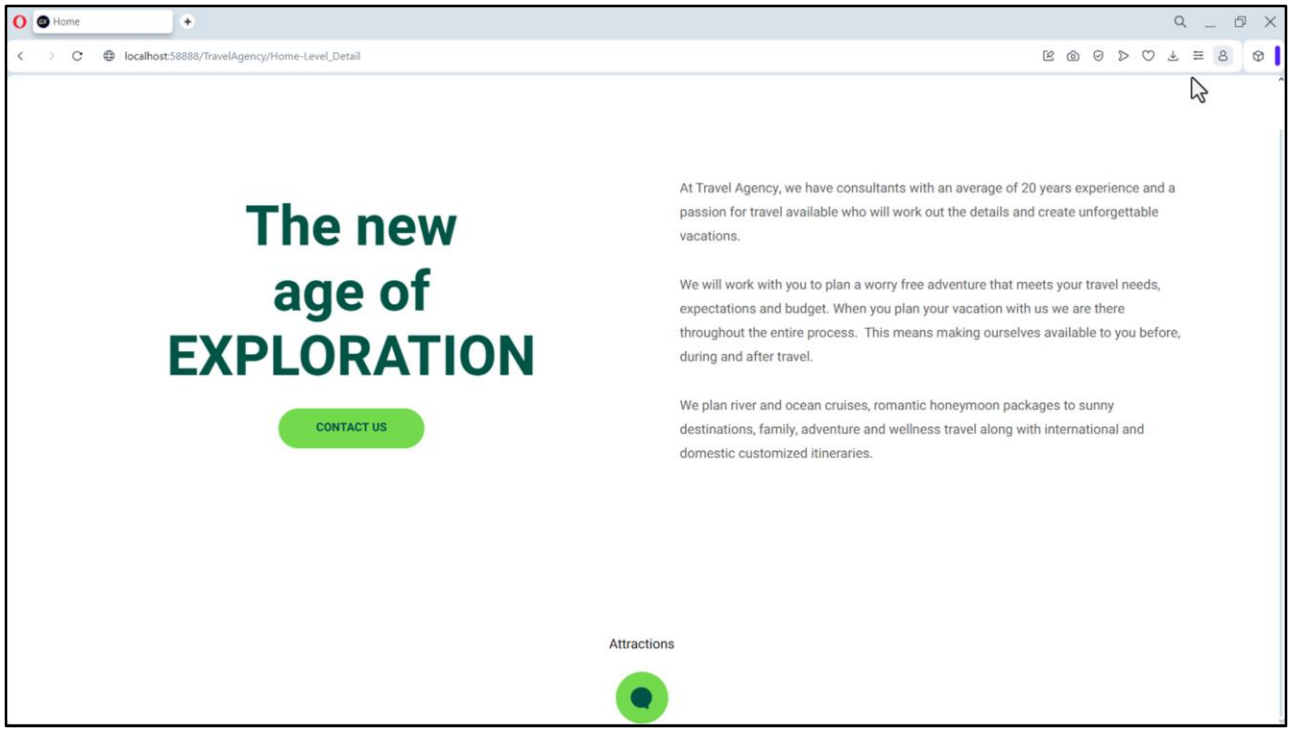


Which is not the TravelAgency we had here.

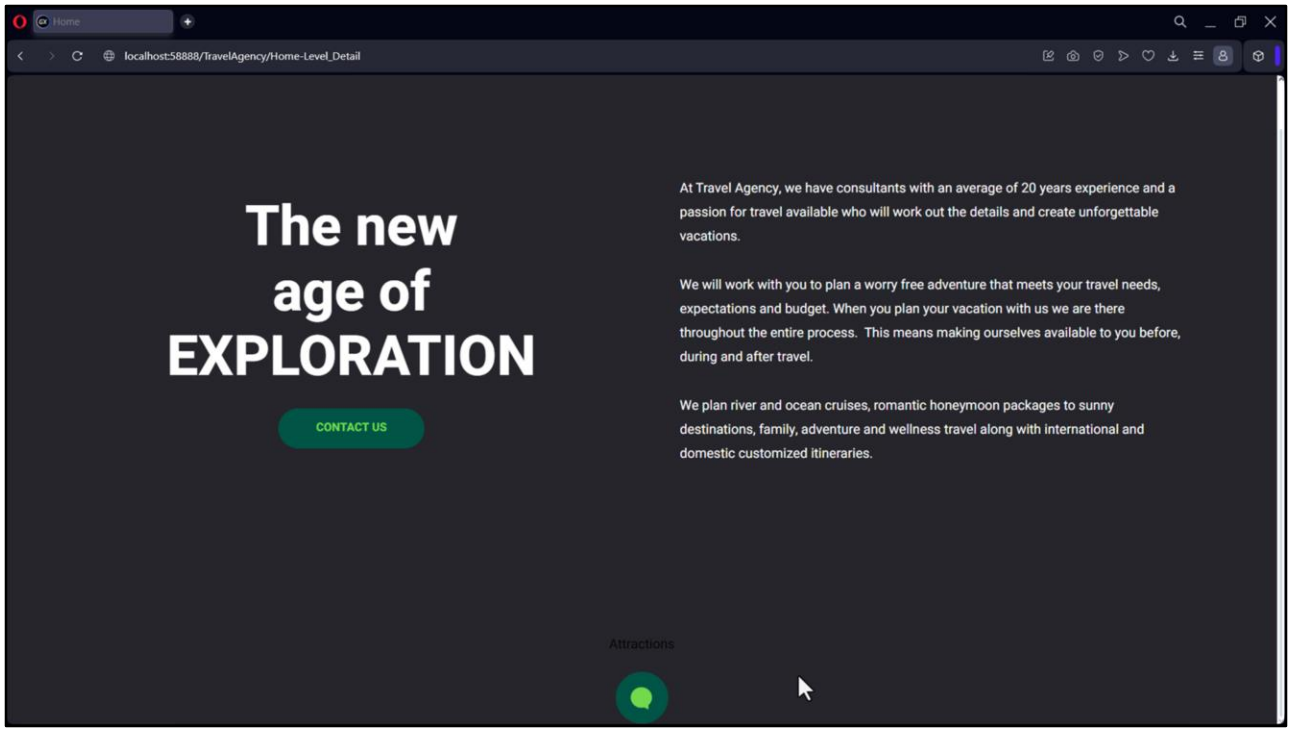


What we are specifying is that it takes one of these two values only for this Design System Object. We haven't specified the value for another Design System Object.

However, note that the same happens even though it appears in this way, when we execute for Angular...



Well, here we see the right image for light mode...



... and let's see what happens in dark mode... very good!

The variations are being made as we expected.

OK, we'll continue in the next video.


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