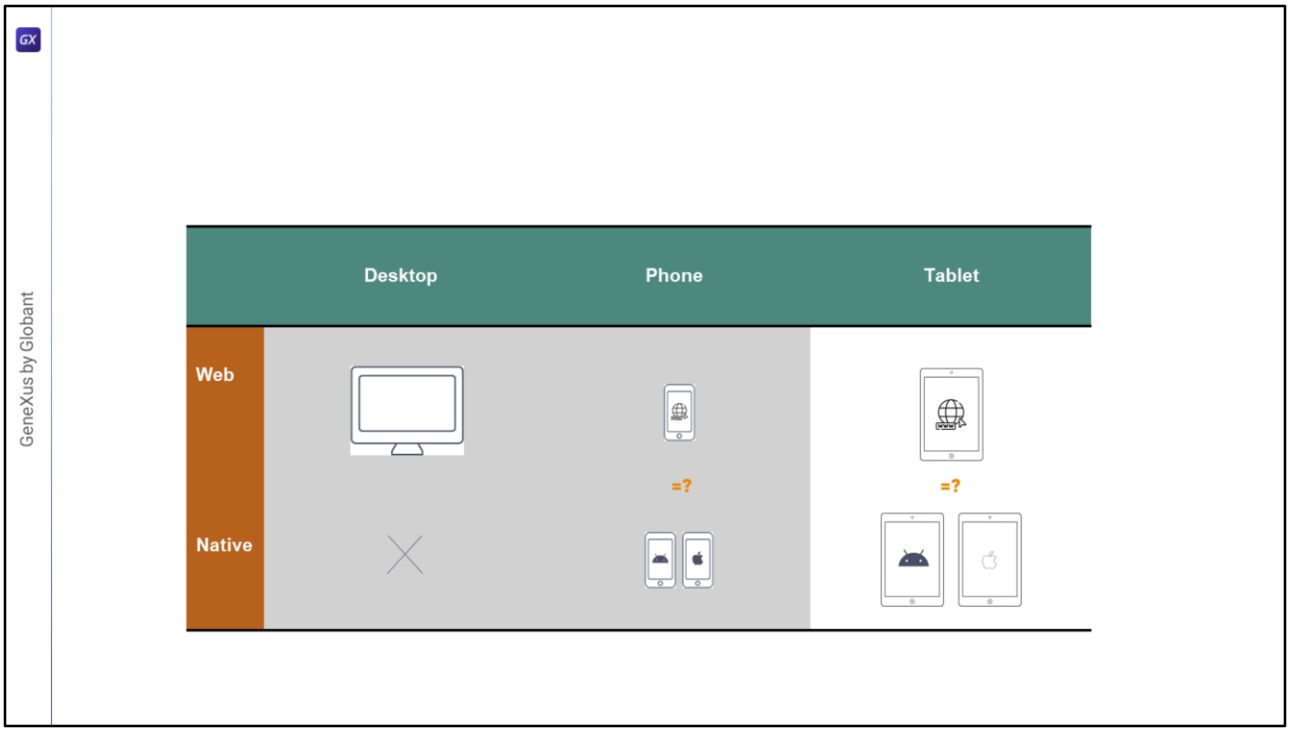
GeneXus by Globant

From Travel Agency Web to Native Mobile

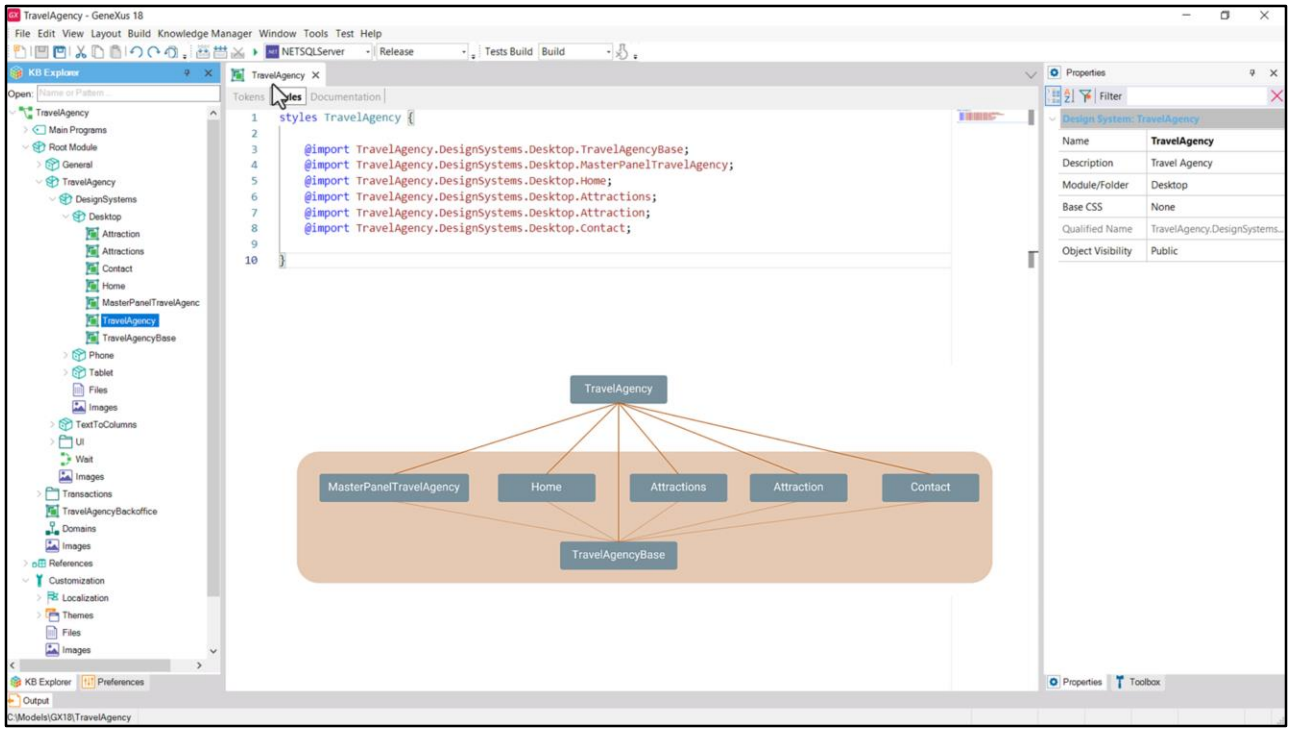
Cecilia Fernández



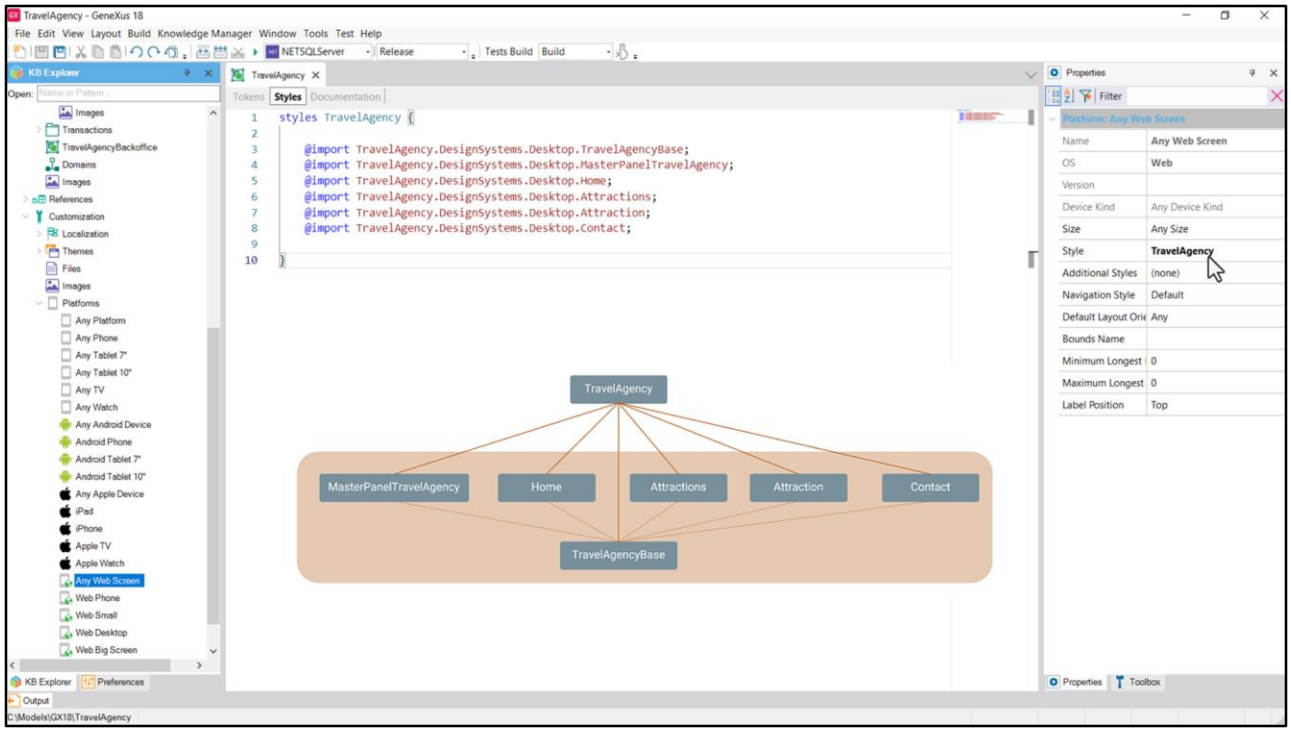
In the previous video, we stopped before showing you some of the consequences of focusing on the Angular solution without taking into account the differences compared to the native application.

GeneXus by Globant

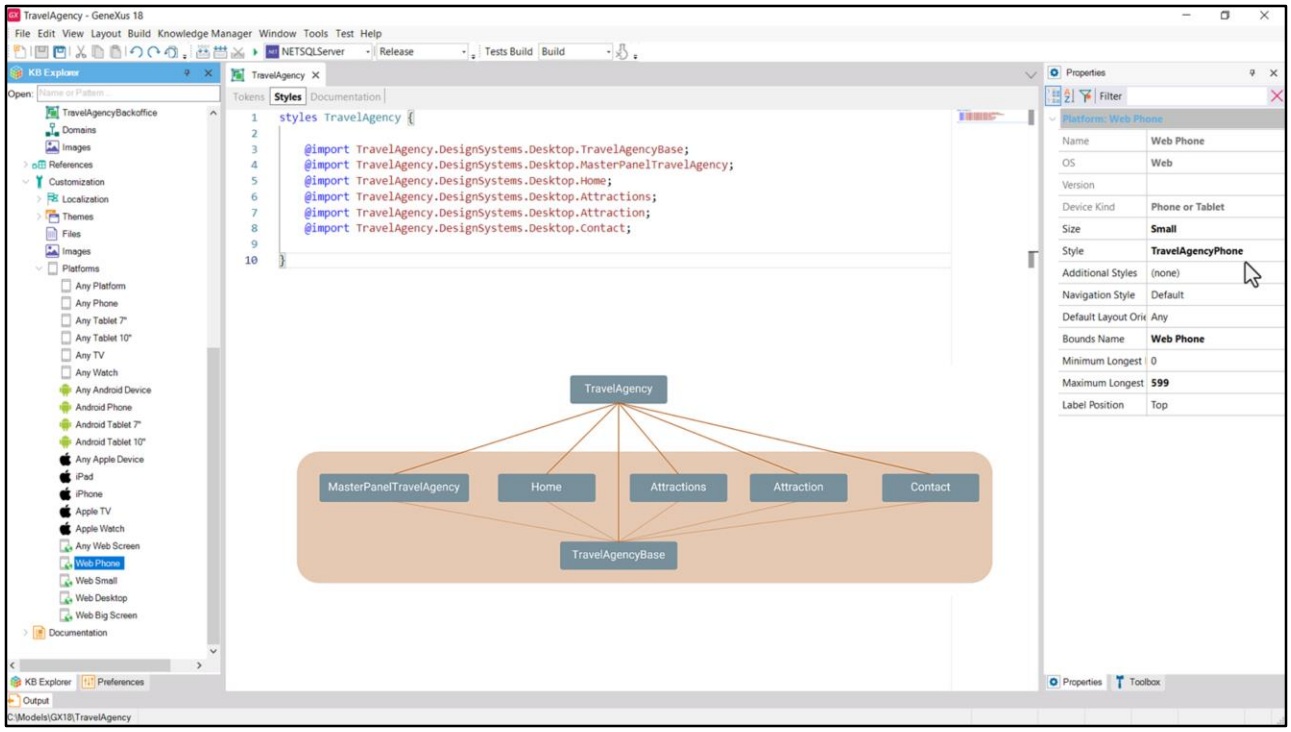
Design System Objects



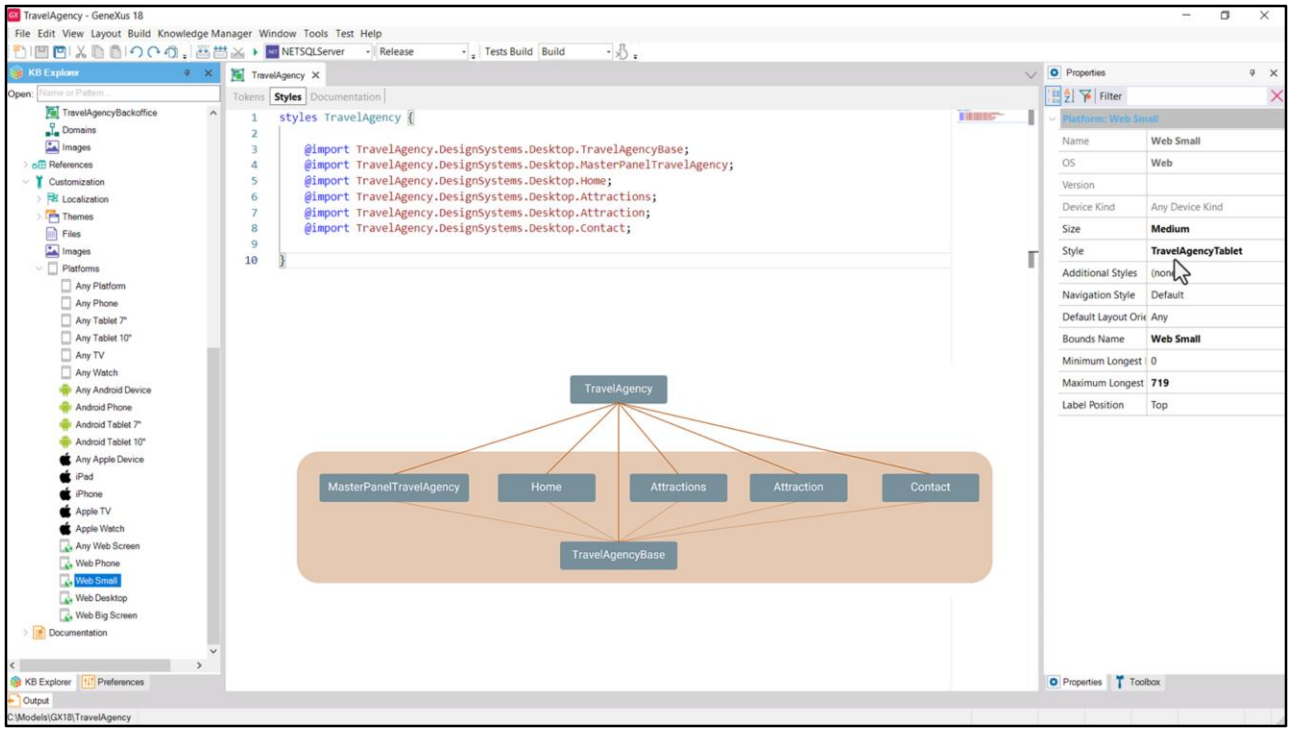
Remember that we had defined a DSO tree to implement the style of the Desktop screens of the Angular application…

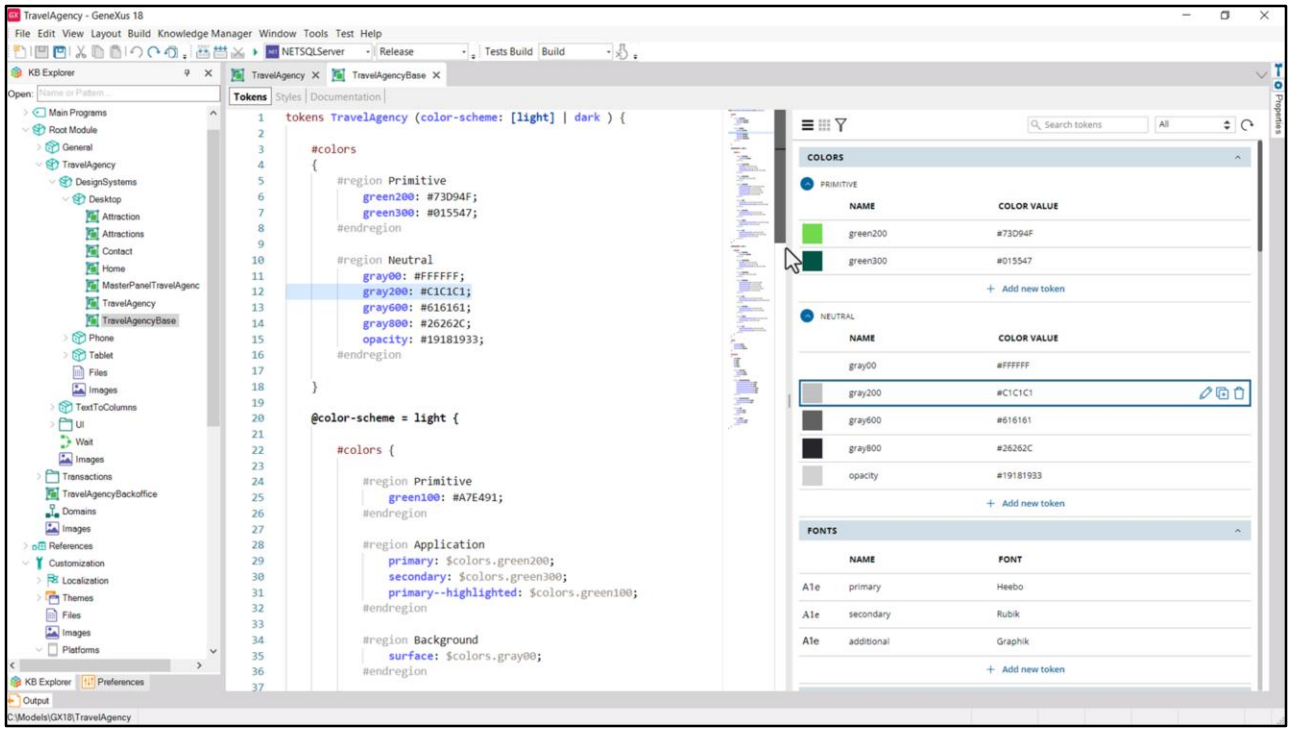


...where this was the root DSO, the DSO we indicated for the platform, both as default (for all platforms), and as default for all Angular screens. That was going to be inherited for Web Desktop and Web Big Screen.

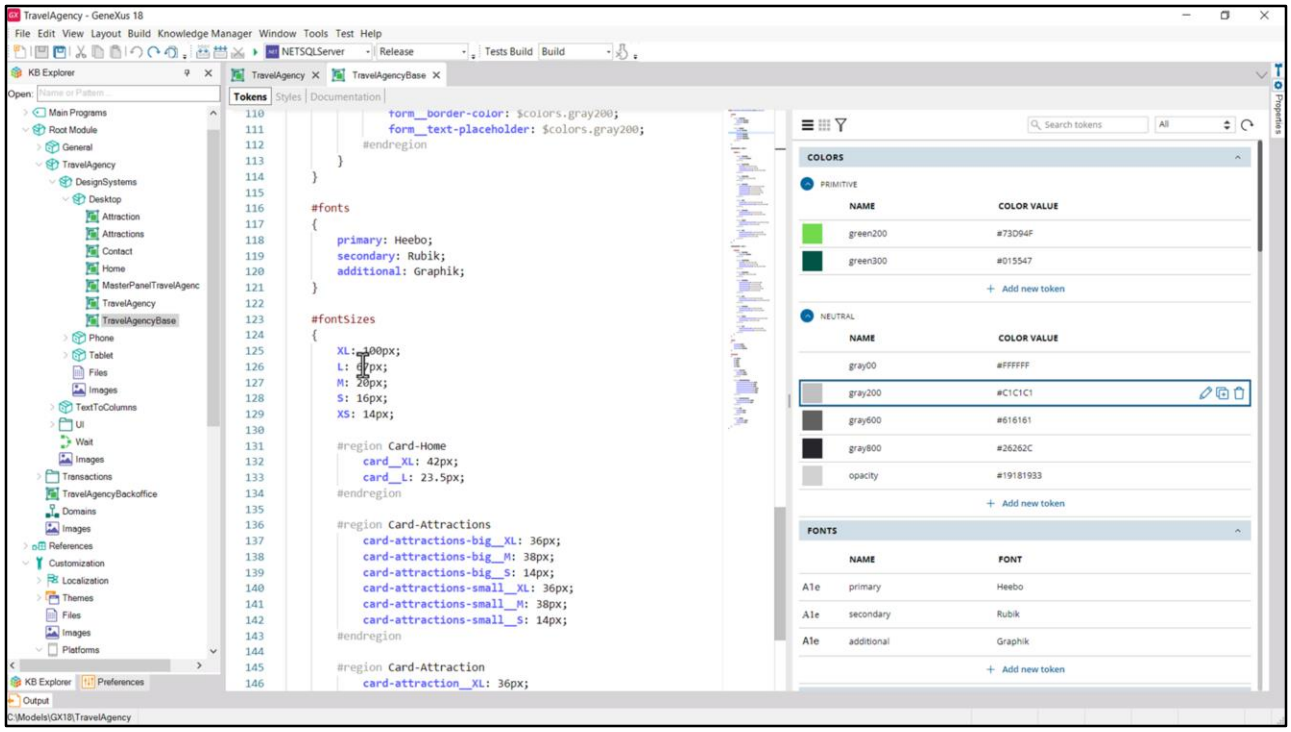


For these others we had customized them with these other DSOs.

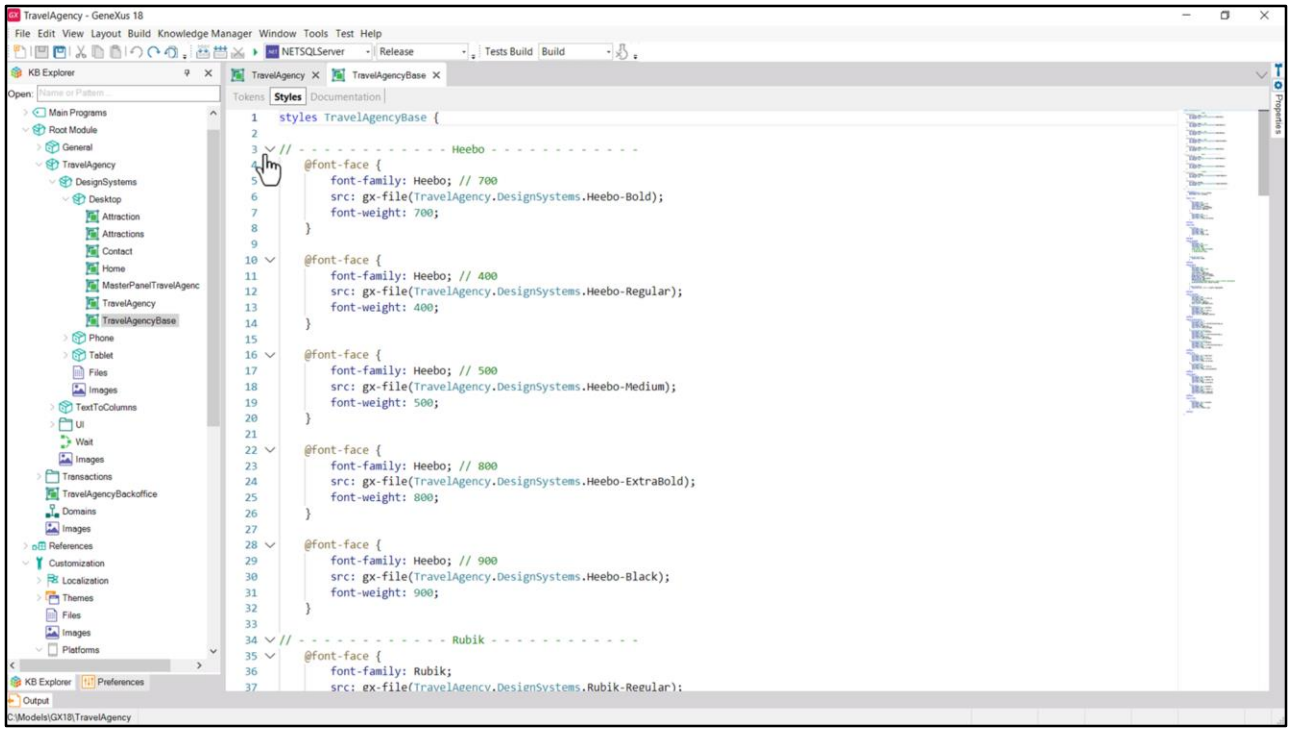

TravelAgency - GeneXus 18
File Edit View Layout Build Knowledge Manager Window Tools Test Help
NETSQLServer
Release
Tests Build
Build
KB Explorer
Open:
Name or Pattern ...
TravelAgencyBackoffice
Domains
Images
References
Customization
Localization
Themes
Files
Images
Platforms
Any Platform
Any Phone
Any Tablet 7"
Any Tablet 10"
Any TV
Any Watch
Any Android Device
Android Phone
Android Tablet 7"
Android Tablet 10"
Any Apple Device
iPad
iPhone
Apple TV
Apple Watch
Any Web Screen
Web Phone
Web Small
Web Desktop
Web Big Screen
Documentation
KB Explorer
Preferences
TravelAgency
Tokens
Styles
Documentation
styles TravelAgency {
@import TravelAgency.DesignSystems.Desktop.TravelAgencyBase;
@import TravelAgency.DesignSystems.Desktop.MasterPanelTravelAgency;
@import TravelAgency.DesignSystems.Desktop.Home;
@import TravelAgency.DesignSystems.Desktop.Attractions;
@import TravelAgency.DesignSystems.Desktop.Attraction;
@import TravelAgency.DesignSystems.Desktop.Contact;
}
TravelAgency
MasterPanelTravelAgency
Home
Attractions
Attraction
Contact
TravelAgencyBase
Properties
Filter
Platform: Web Small
Name
Web Small
OS
Web
Version
Device Kind
Any Device Kind
Size
Medium
Style
TravelAgencyTablet
Additional Styles
(none)
Navigation Style
Default
Default Layout Orie
Any
Bounds Name
Web Small
Minimum Longest
0
Maximum Longest
719
Label Position
Top
Properties
Toolbox
Output
C:\Models\GX18\TravelAgency



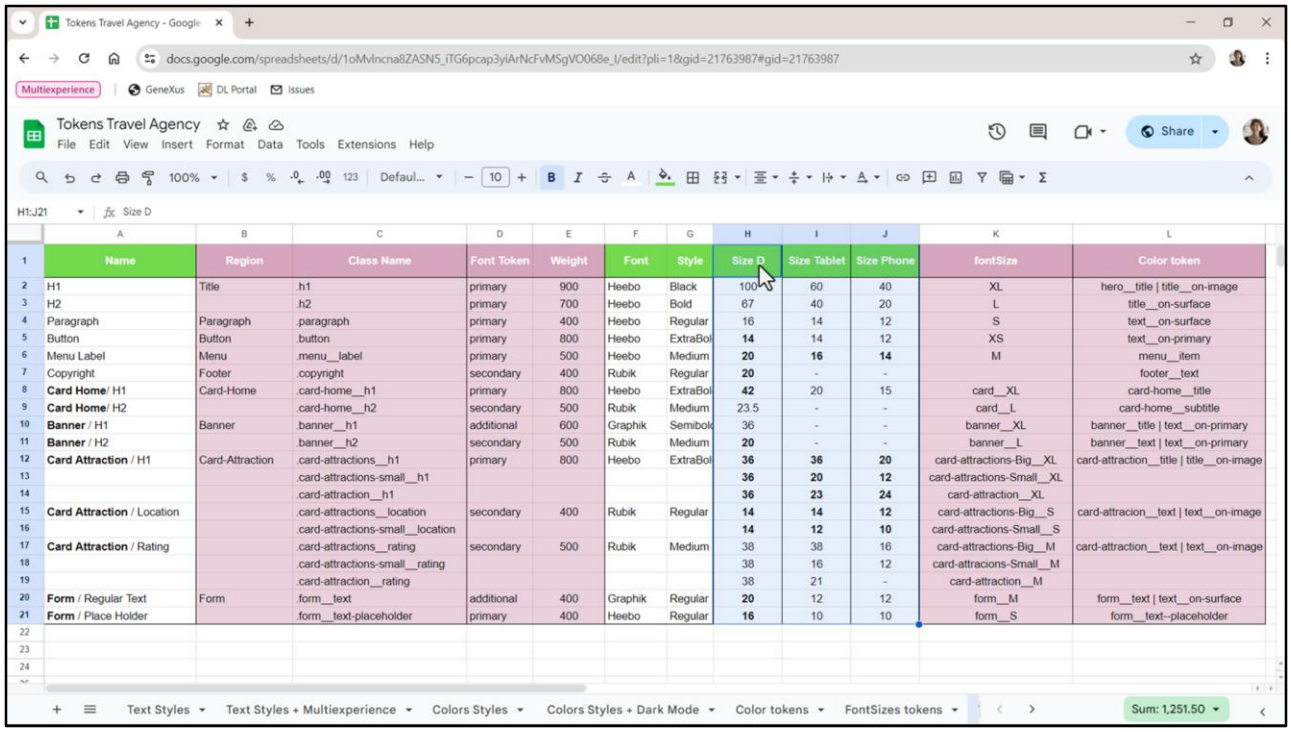
All these DSOs were built on the Base DSO, in which we had defined all the general tokens of the application: both the color tokens (with their light and dark variations)...



...such as those for fonts, as well as those for font size.

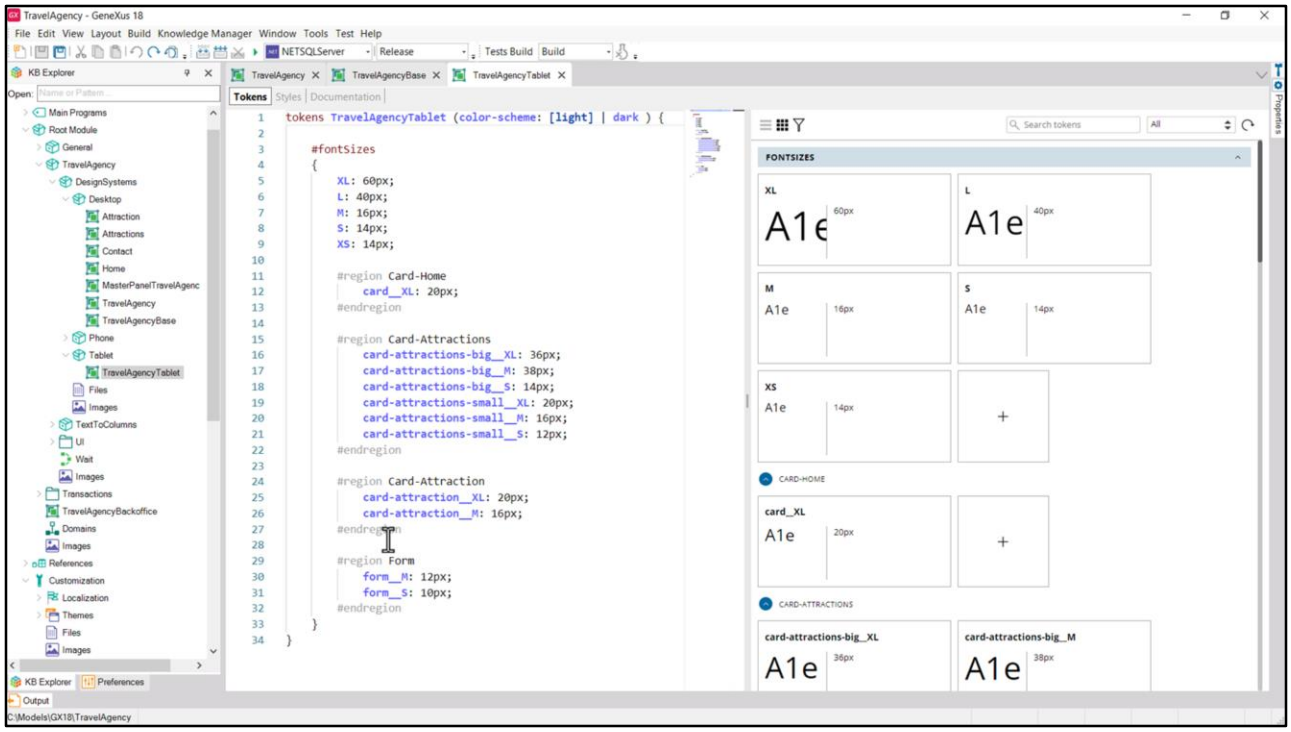


And we had also added there the fonts and typography classes that we had identified in the Preparation stage.



	Name	Region	Class Name	Font Token	Weight	Font	Style	Size D	Size Tablet	Size Phone	fontSize	Color token
2	H1	Title	.h1	primary	900	Heebo	Black	100	60	40	XL	hero__title | title__on-image
3	H2		.h2	primary	700	Heebo	Bold	67	40	20	L	title__on-surface
4	Paragraph	Paragraph	.paragraph	primary	400	Heebo	Regular	16	14	12	S	text__on-surface
5	Button	Button	.button	primary	800	Heebo	ExtraBol	**14**	14	12	XS	text__on-primary
6	Menu Label	Menu	.menu__label	primary	500	Heebo	Medium	**20**	**16**	**14**	M	menu__item
7	Copyright	Footer	.copyright	secondary	400	Rubik	Regular	**20**	-	-		footer__text
8	**Card Home**/ H1	Card-Home	.card-home__h1	primary	800	Heebo	ExtraBol	**42**	20	15	card__XL	card-home__title
9	**Card Home**/ H2		.card-home__h2	secondary	500	Rubik	Medium	23.5	-	-	card__L	card-home__subtitle
10	**Banner** / H1	Banner	.banner__h1	additional	600	Graphik	Semibol	36	-	-	banner__XL	banner__title | text__on-primary
11	**Banner** / H2		.banner__h2	secondary	500	Rubik	Medium	**20**	-	-	banner__L	banner__text | text__on-primary
12	**Card Attraction** / H1	Card-Attraction	.card-attractions__h1	primary	800	Heebo	ExtraBol	**36**	**36**	**20**	card-attractions-Big__XL	card-attraction__title | title__on-image
13			.card-attractions-small__h1					**36**	**20**	**12**	card-attractions-Small__XL	
14			.card-attraction__h1					**36**	**23**	**24**	card-attraction__XL	
15	**Card Attraction** / Location		.card-attractions__location	secondary	400	Rubik	Regular	**14**	**14**	**12**	card-attractions-Big__S	card-attracion__text | text__on-image
16			.card-attractions-small__location					**14**	**12**	**10**	card-attractions-Small__S	
17	**Card Attraction** / Rating		.card-attractions__rating	secondary	500	Rubik	Medium	38	38	16	card-attractions-Big__M	card-attraction__text | text__on-image
18			.card-attractions-small__rating					38	16	12	card-attracions-Small__M	
19			.card-attraction__rating					38	21	-	card-attraction__M	
20	**Form** / Regular Text	Form	.form__text	additional	400	Graphik	Regular	**20**	12	12	form__M	form__text | text__on-surface
21	**Form** / Place Holder		.form__text-placeholder	primary	400	Heebo	Regular	**16**	10	10	form__S	form__text--placeholder

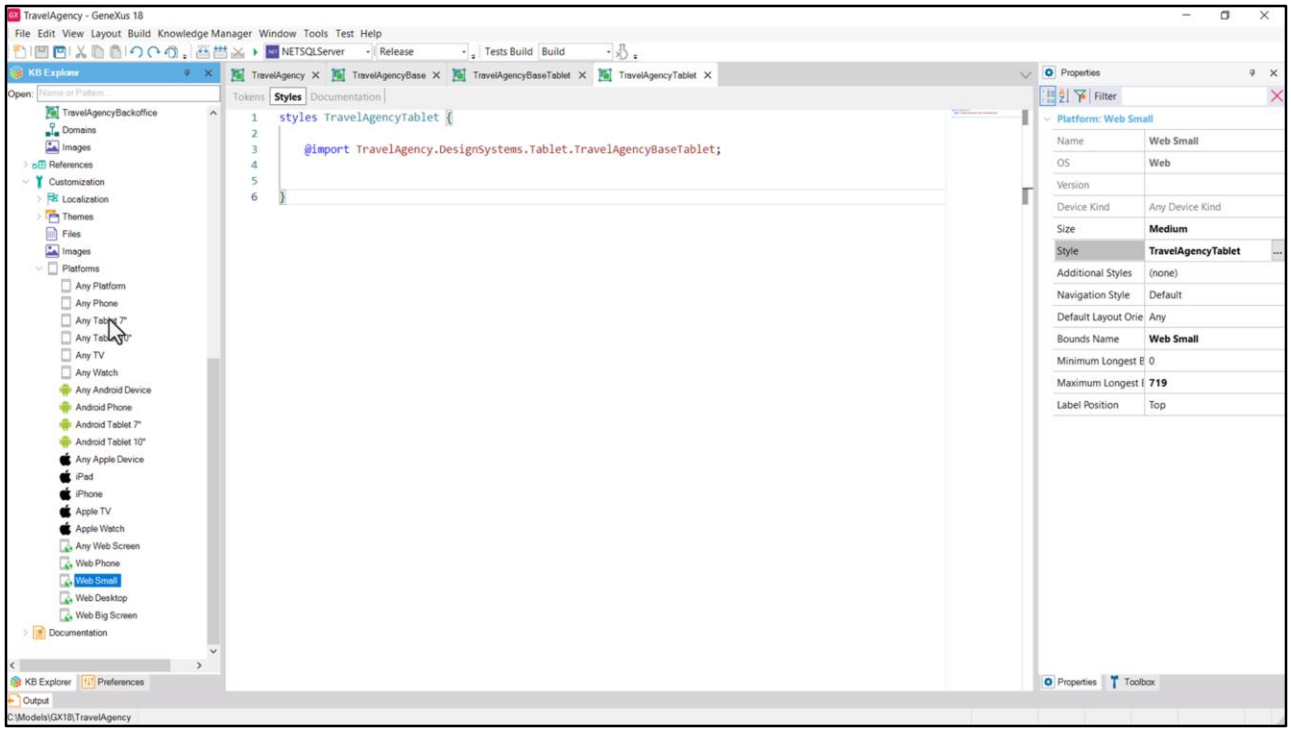
You may remember that then we had already analyzed the variations that the font size tokens would undergo, according to the screen size, and we had also identified some variations for the typography classes.



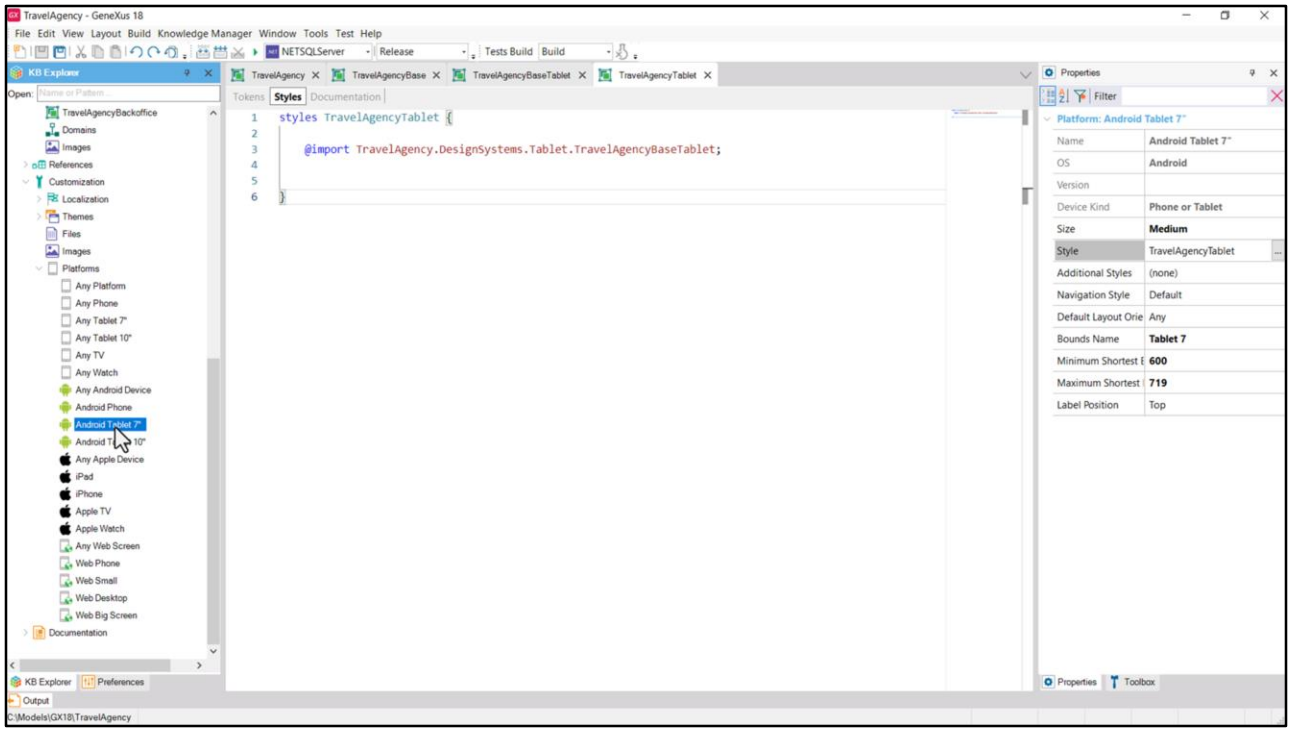
And we had expressed all this in two parallel DSOs.

Thus, for Tablet we had created this TravelAgencyTablet DSO, where we had specialized those variations that we had identified. We should, in fact, call it similarly to its Desktop parallel: TravelAgency**Base**Tablet.

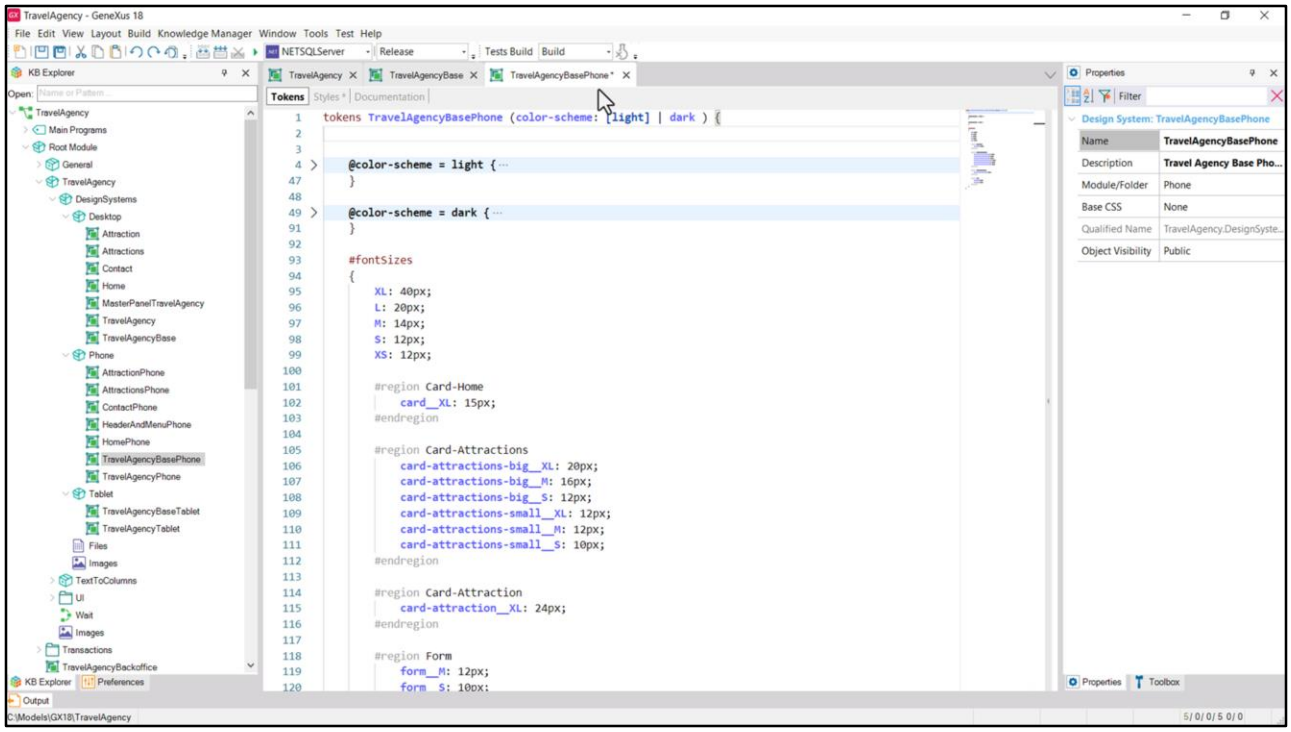
And it should import the TravelAgencyBase, because it specializes it. By importing it, what it does is to change the values to the fontSizes tokens... It also adds classes that were not needed for Desktop but are needed for this other screen size.



Of course, we will need to have the root DSO, TravelAgencyTablet, which is the one we indicate here for Web platform of that size.

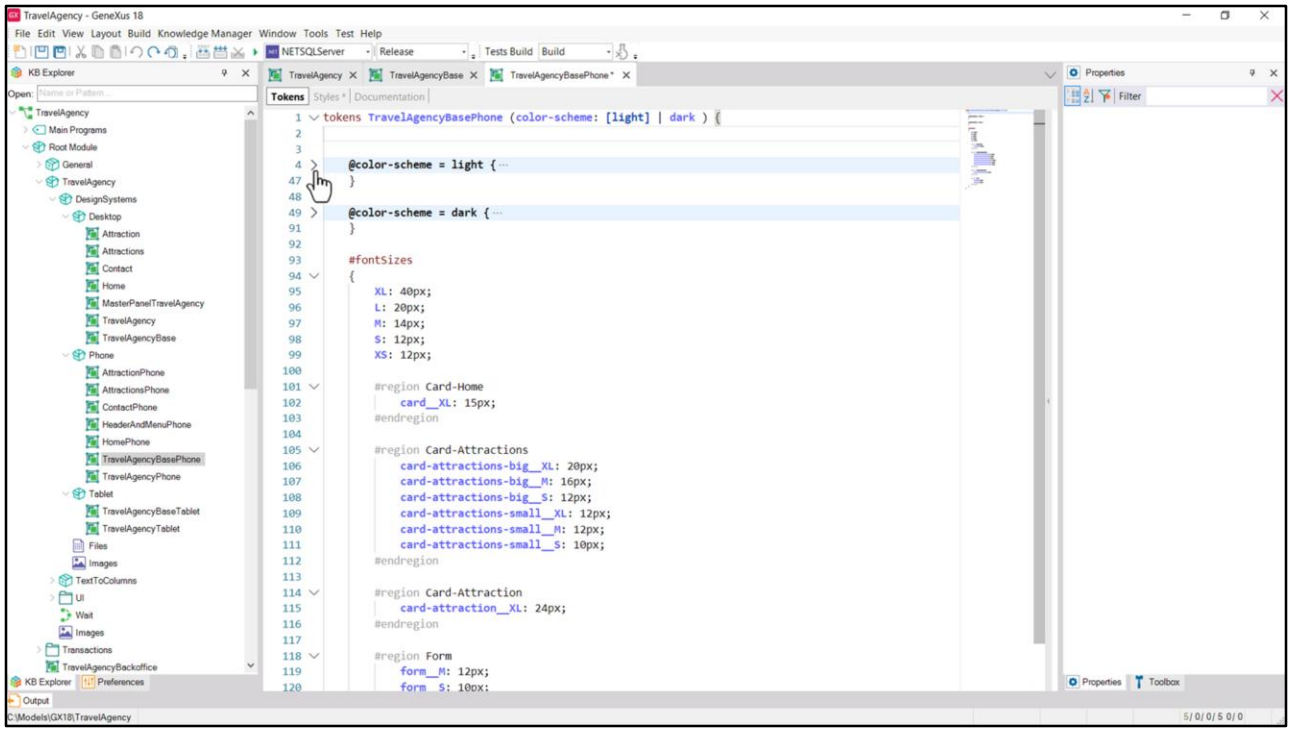


And if we want the same DSO to also work for the native application in this size, we can place it here as well.

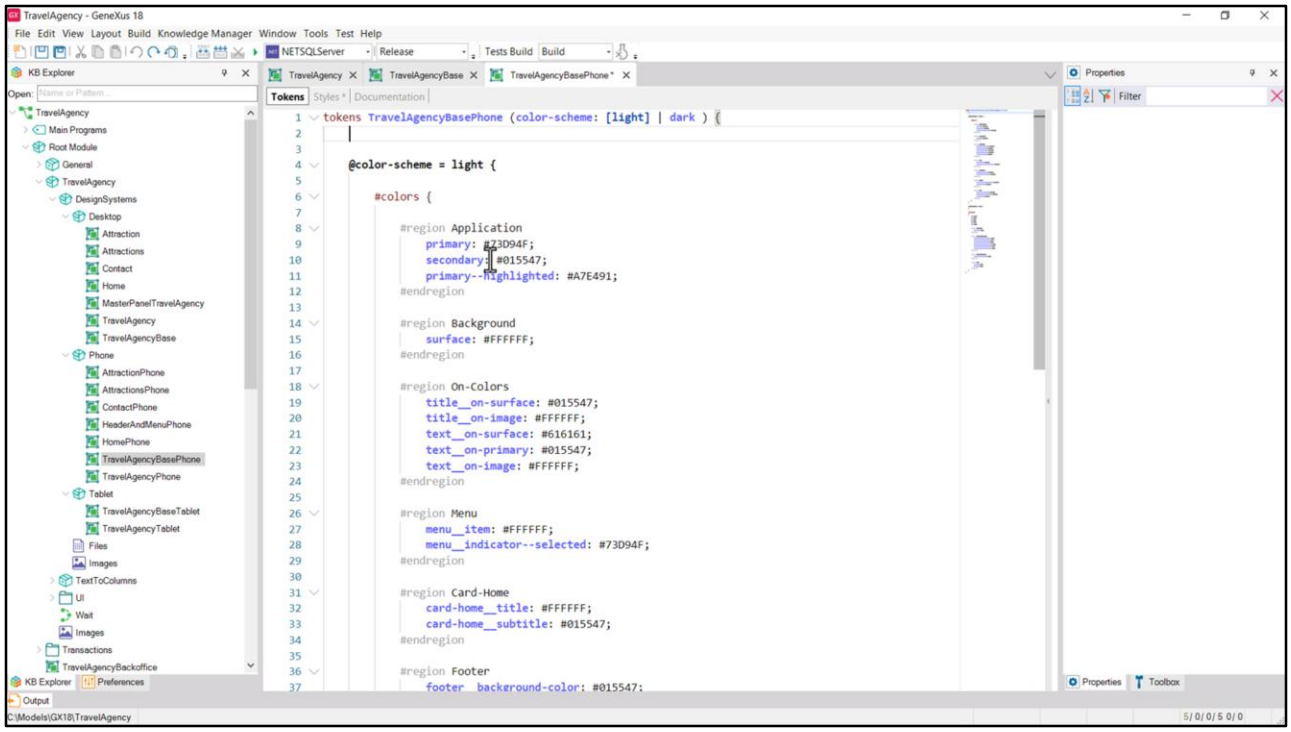


And the same will be true for Phone size. Here I have already created the parallel structure.

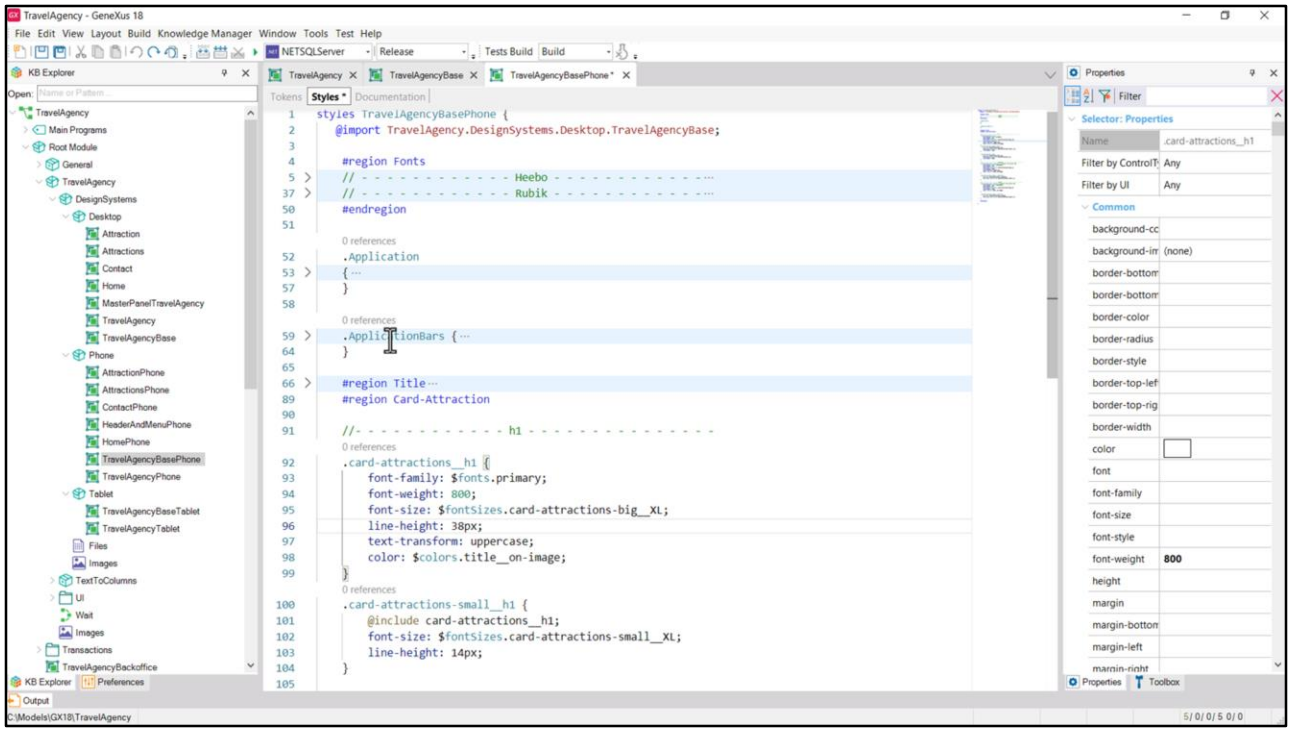
This was the DSO that we had created in the preparation stage (I renamed it), to which I had to modify some things that I need you to see to illustrate the point.



Since it matters to the Desktop analog, only the tokens that change should be here and not the ones that keep the same values, such as the color tokens. Why, then, do all these color tokens appear, instead of just the font size tokens?



Don't mind this. I had to temporarily copy all the Desktop tokens whose value used another token, because this indirection is not working for the native application. I had to assign their absolute values. This will be fixed, so don't worry. This whole section will be gone when this bug is fixed.

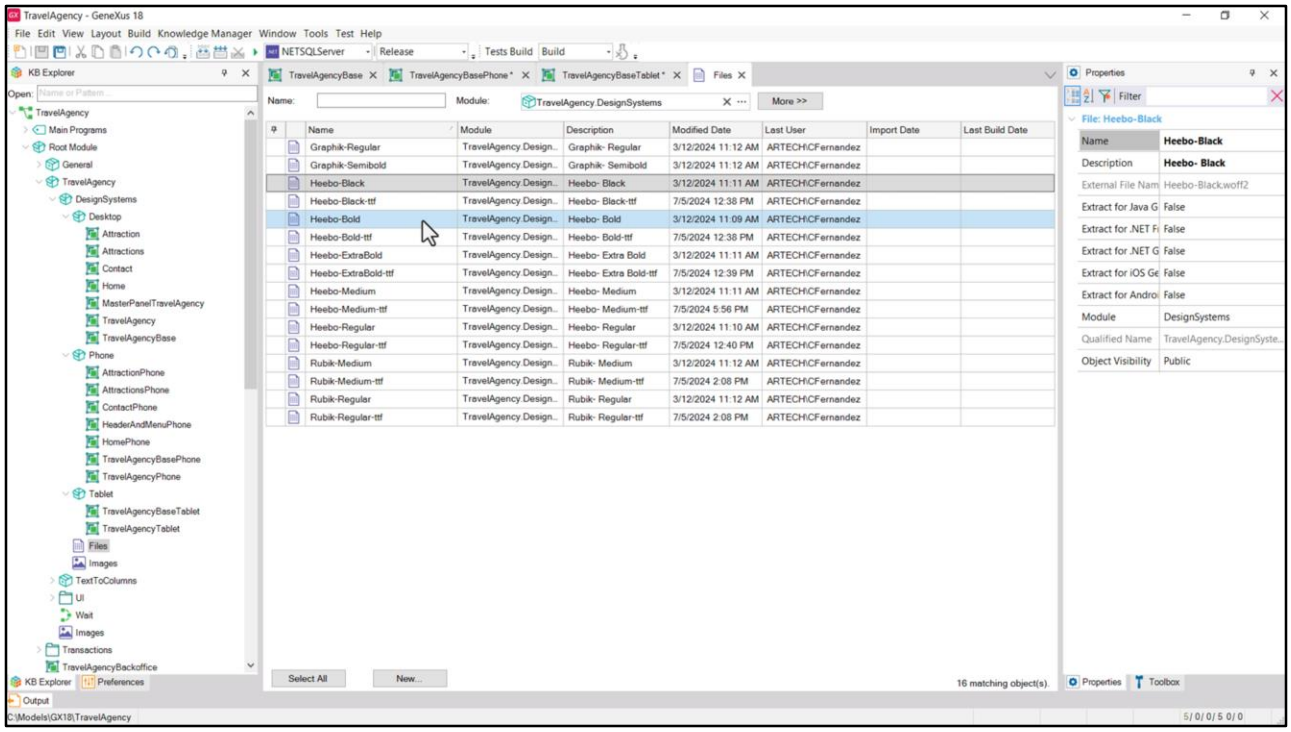


What we do have to worry about is how we define the fonts.

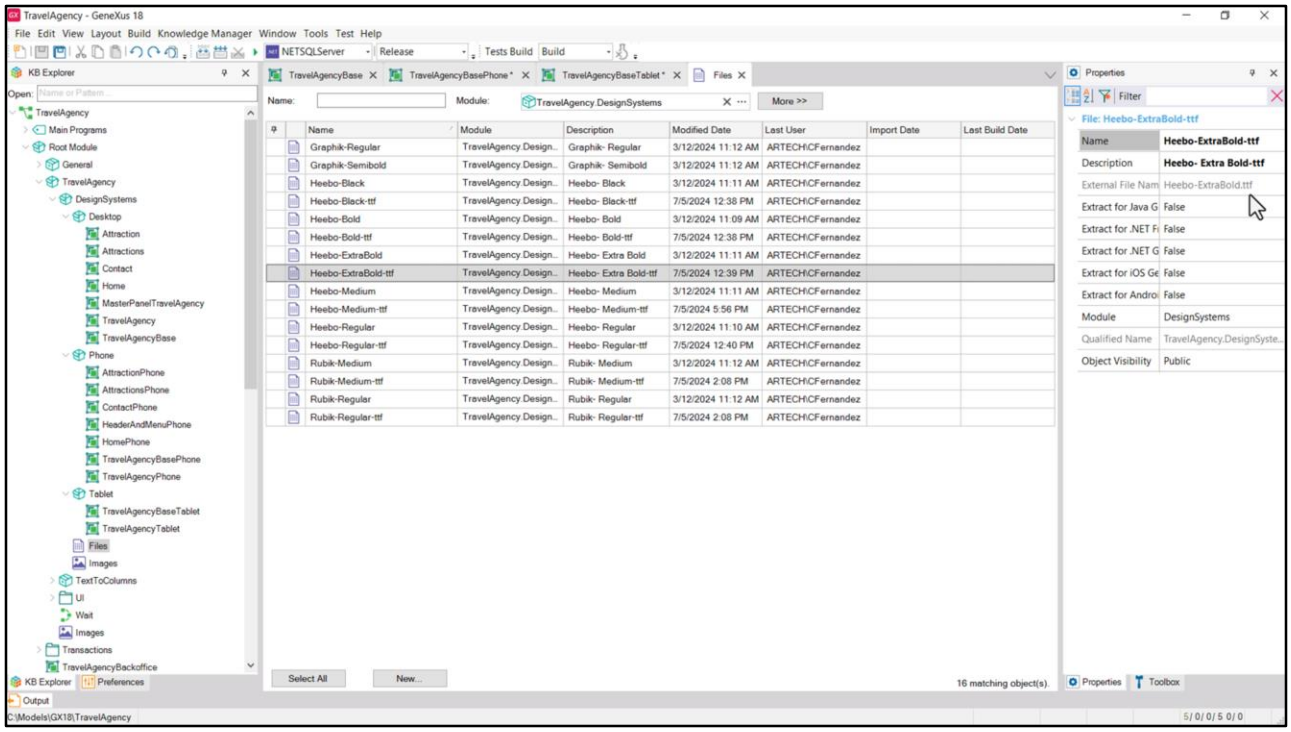
Before: all these classes are the ones we had identified in the preparation stage.

The Application and ApplicationBars classes will have special semantics for native applications, as we will see later. I will not show them now.

Now let's turn our attention to the fonts...



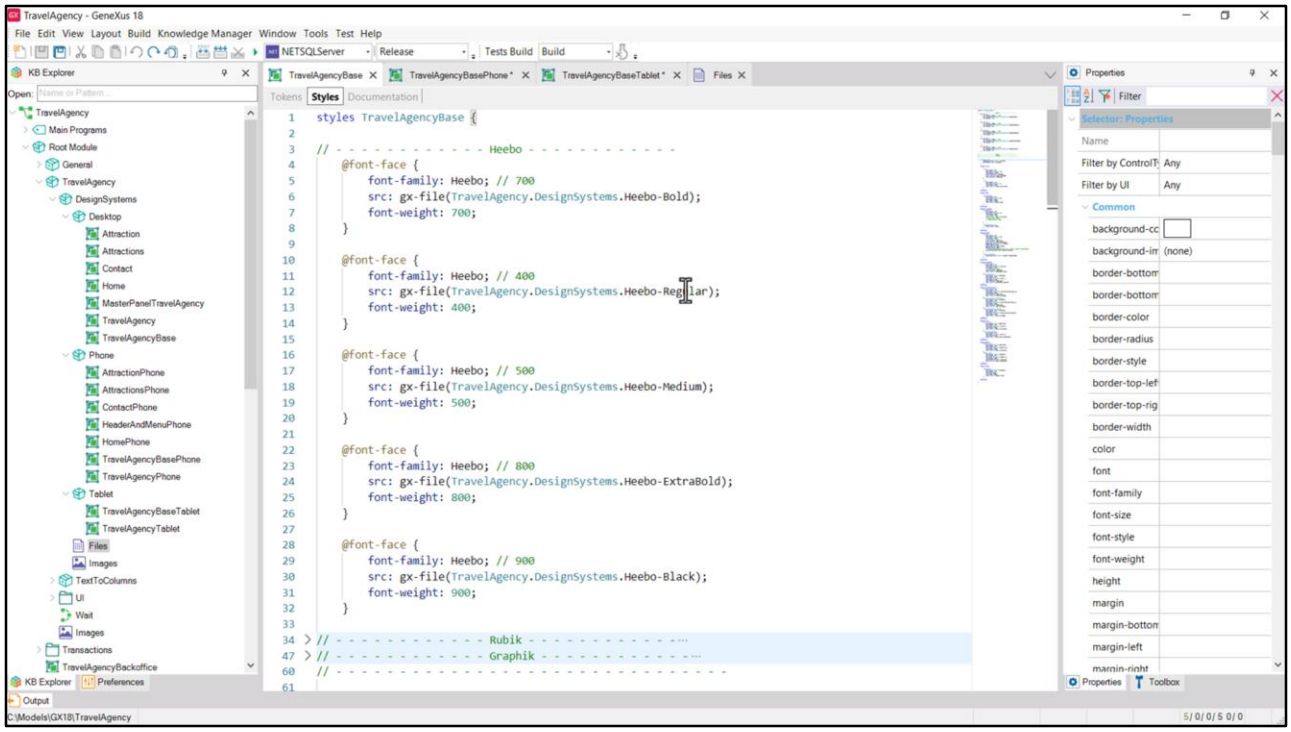
We had inserted as files in the KB the non-default fonts that we were going to use; for example, all the Heebo fonts of different weights, remember? But we had integrated only those of woff2 format, which, as we had said, were the best for Angular. However, as we also said at the time, they cannot be used in native environments.



That's why I added the same fonts but now in ttf format.

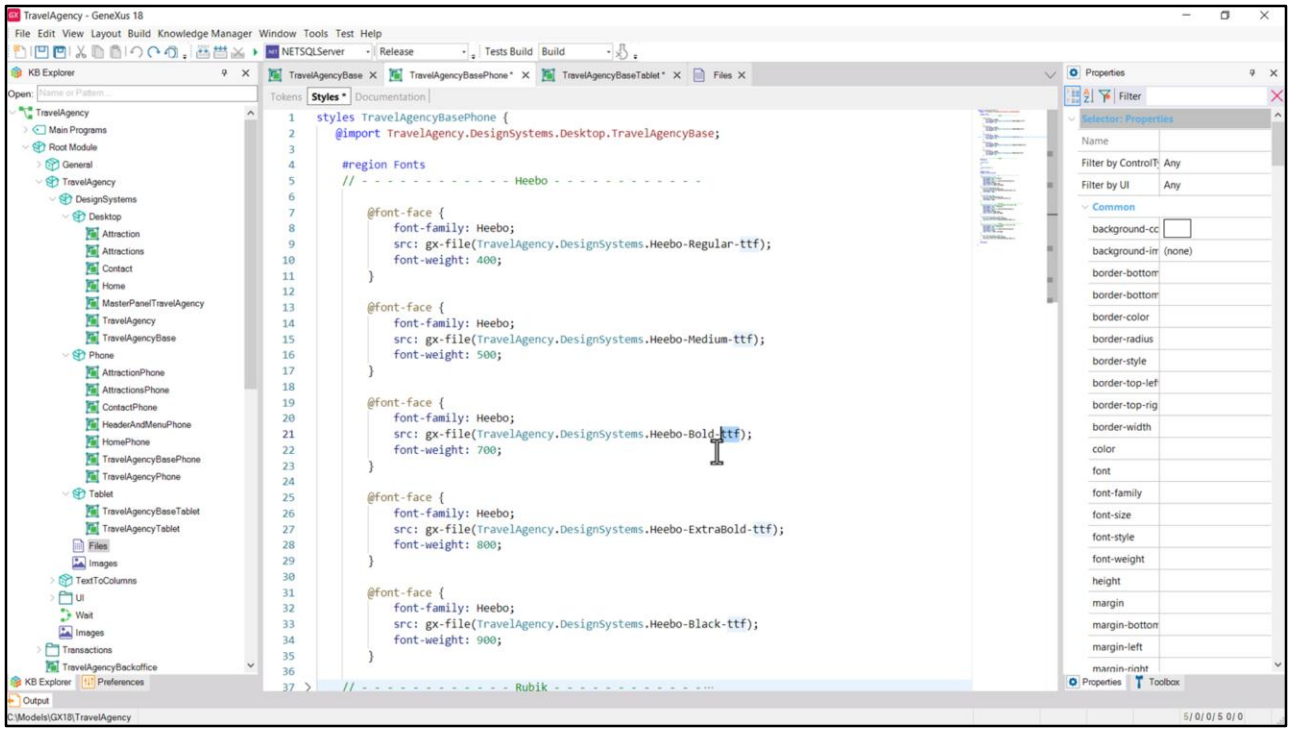
If we had used for Angular the ttf ones instead of the woff2 ones, we would not have this problem, although the ttf fonts are not the best for Angular.

What should we do then in this case where we have the 2 types of fonts?



In the Base DSO for Desktop we had the font-face rules that declared the fonts. Clearly we will have to specialize them for our DSOs for native applications, so that now they take the ttf file and not the woff2.

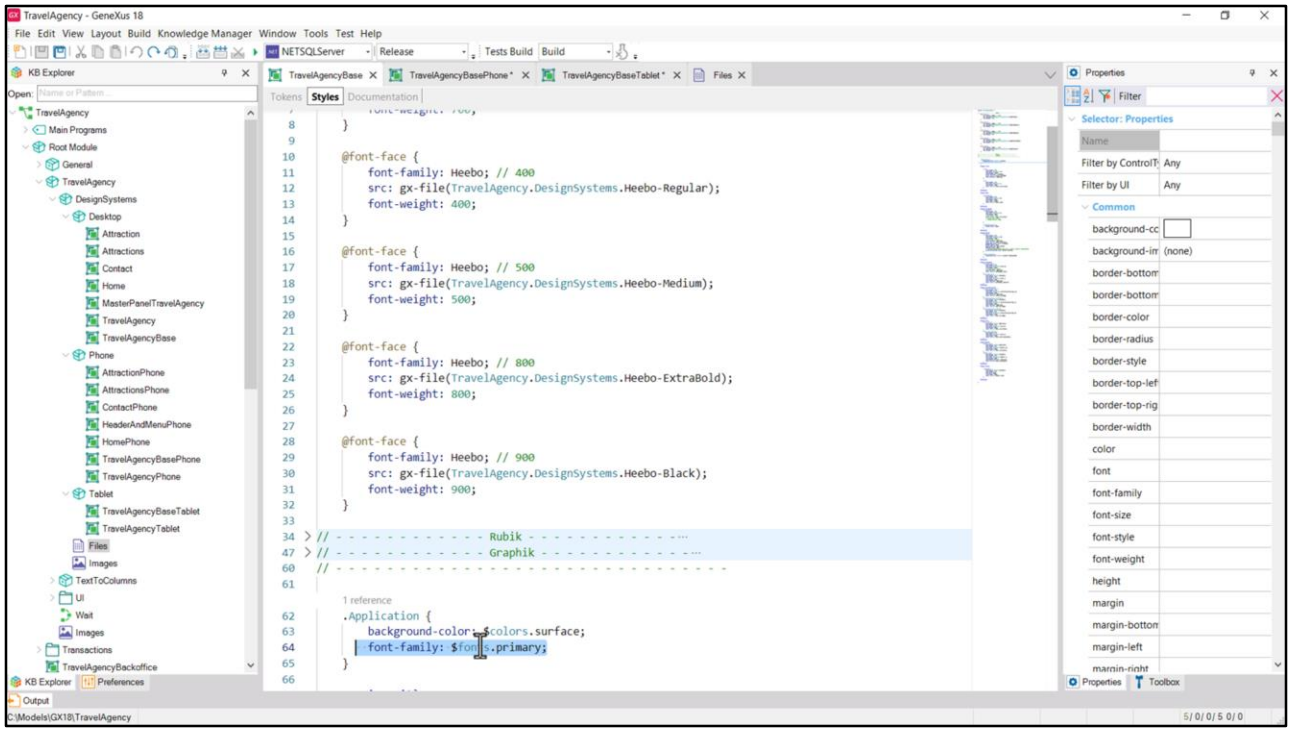
That is, we would have to copy for both TravelAgencyBasePhone and TravelAgencyBaseTablet all these font-face rules and simply change the file to the ttf.



But what if we want to use woff2 for the Angular application in Phone and ttf for the native one in phone? There we will have no choice but to specialize.

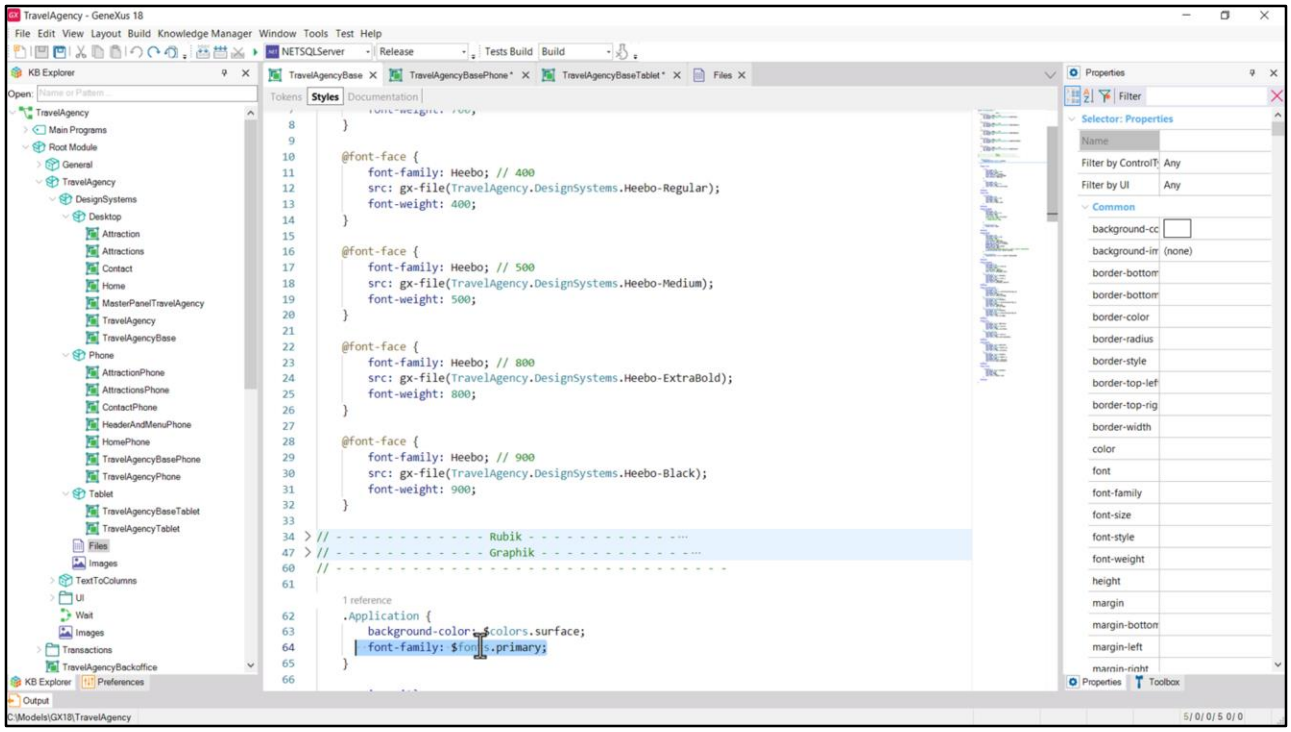
That is to say, to have a DSO tree for the Web Phone platform and another for the native ones.

Then we would specify here the default for Phone that we will want to apply only to the native ones and here its exception, the one for the Web, that will use the woff2 format.



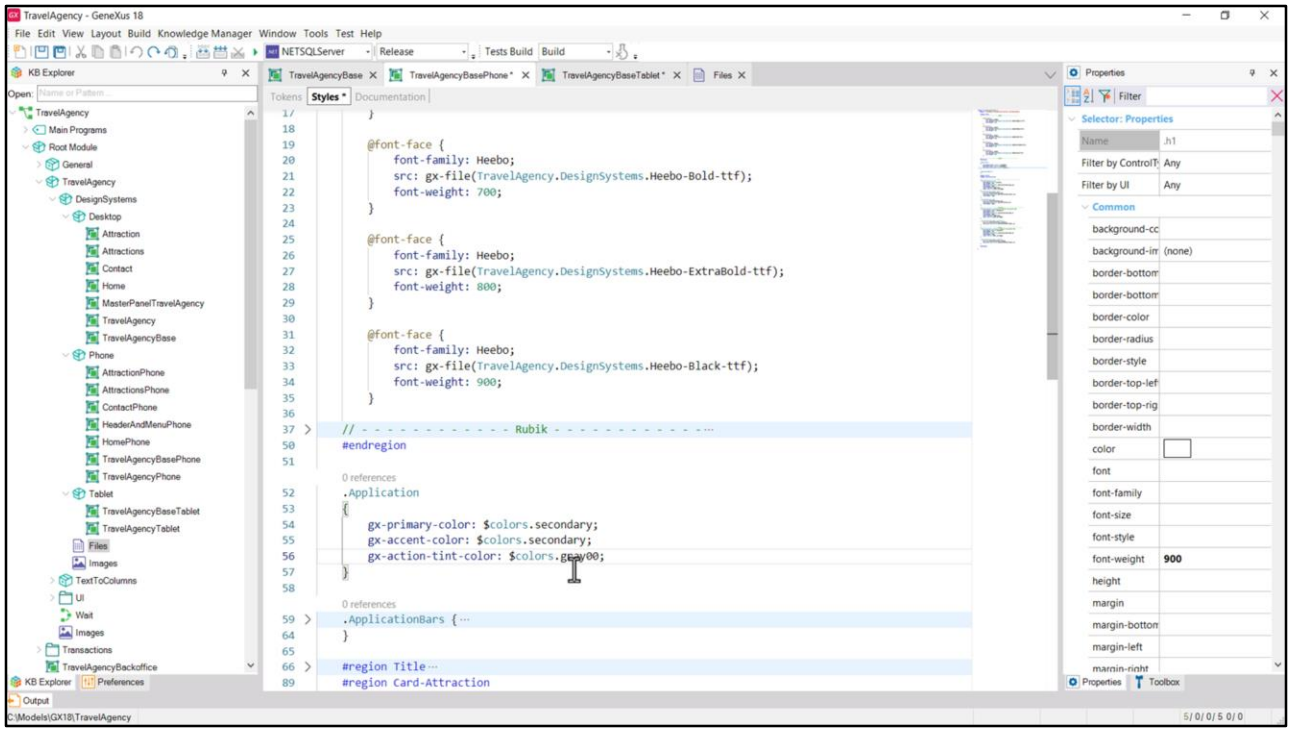
But we have a bigger problem than that (let's assume we use the ttf fonts for Angular as well). The problem is that we had built the solution in Angular taking advantage of the fact that we could specify properties that were valid as default for all classes by specifying them under the Application class, remember? And that's how we decided to call all of them Heebo, regardless of their weight, and then differentiate them by the font-weight property.

In this way, we were able to define a primary token for the Heebo family, and in the Application class indicate that this will be the default family.



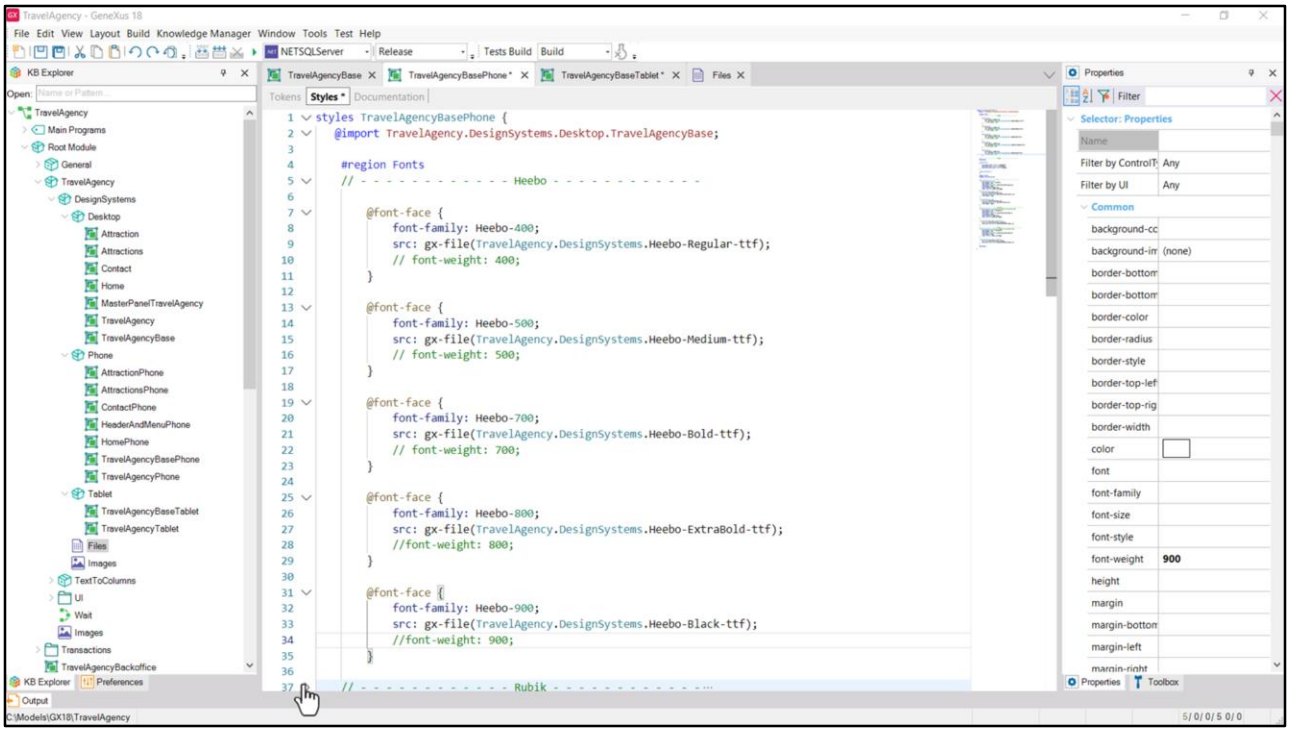
In this way, in the classes that used that family, we didn't have to indicate the family, but only its weight. Here there are two assumptions that don't work in the native world with GeneXus (at least not for the moment):

One: that the properties specified in the Application class will be valid by default for all classes. This is true for the Angular world because that class is applied to the body tag of all HTML pages...



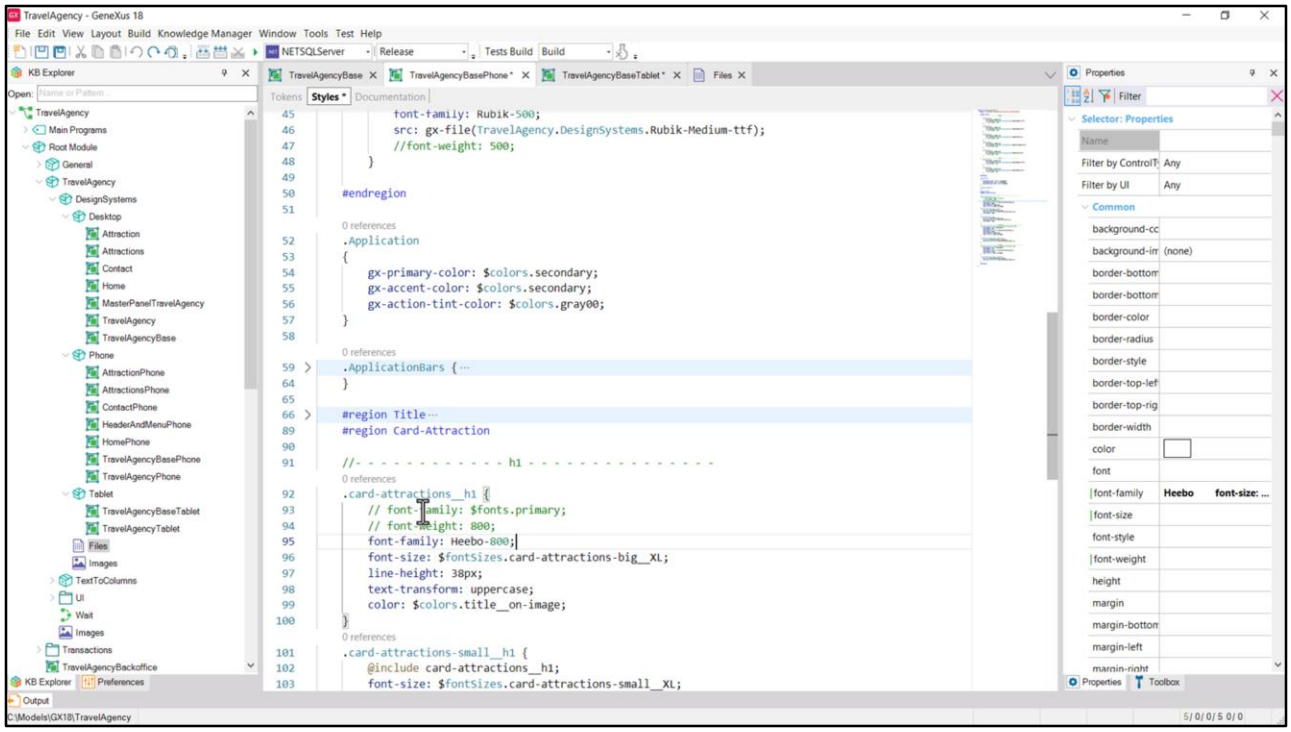
But in the native world that class only controls some general behaviors of the application, like the colors of the Application Bar, actions, and so on. So if I place the font-family property here, it will not be valid by default for all the controls of all the screens.

And second and even more important: in Angular, since the font family names are repeated, what identifies the font is the name and weight pair. This is not the case (at least for the moment) for native applications. What identifies the font is the family name.

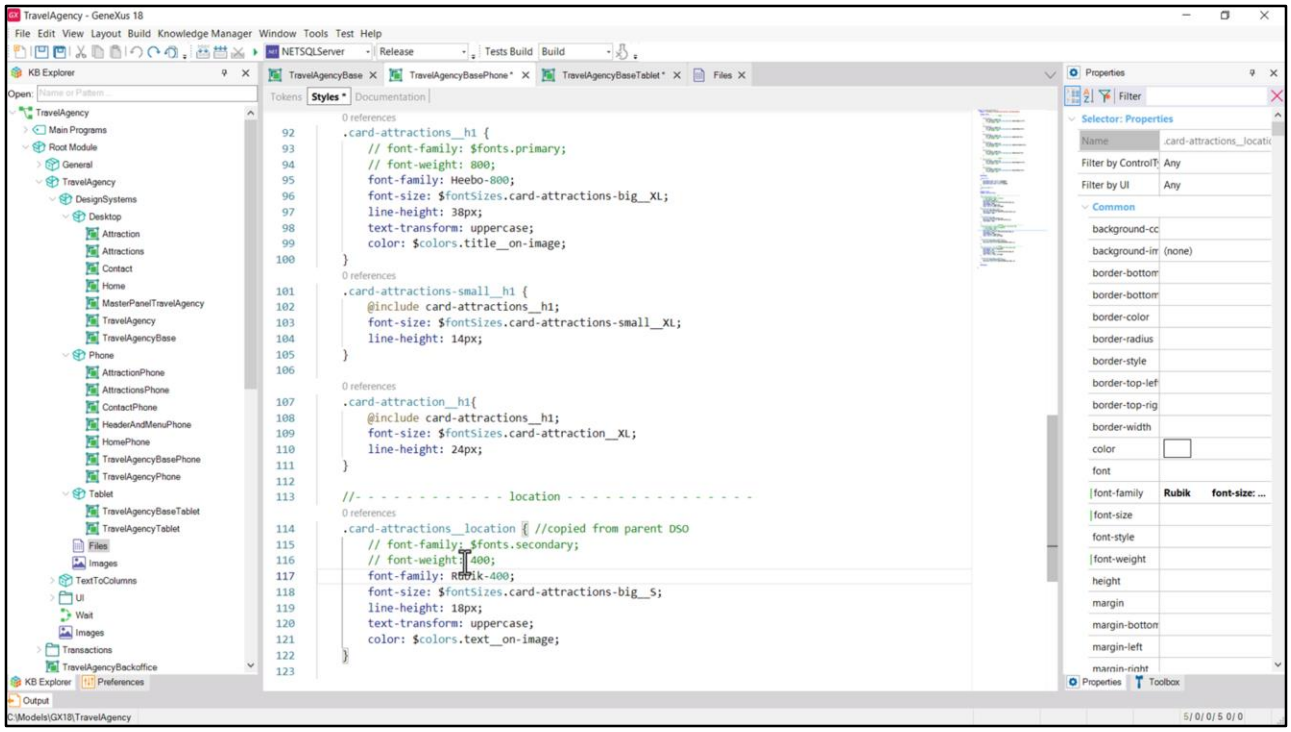


So we have to give different names to the Heebo fonts.

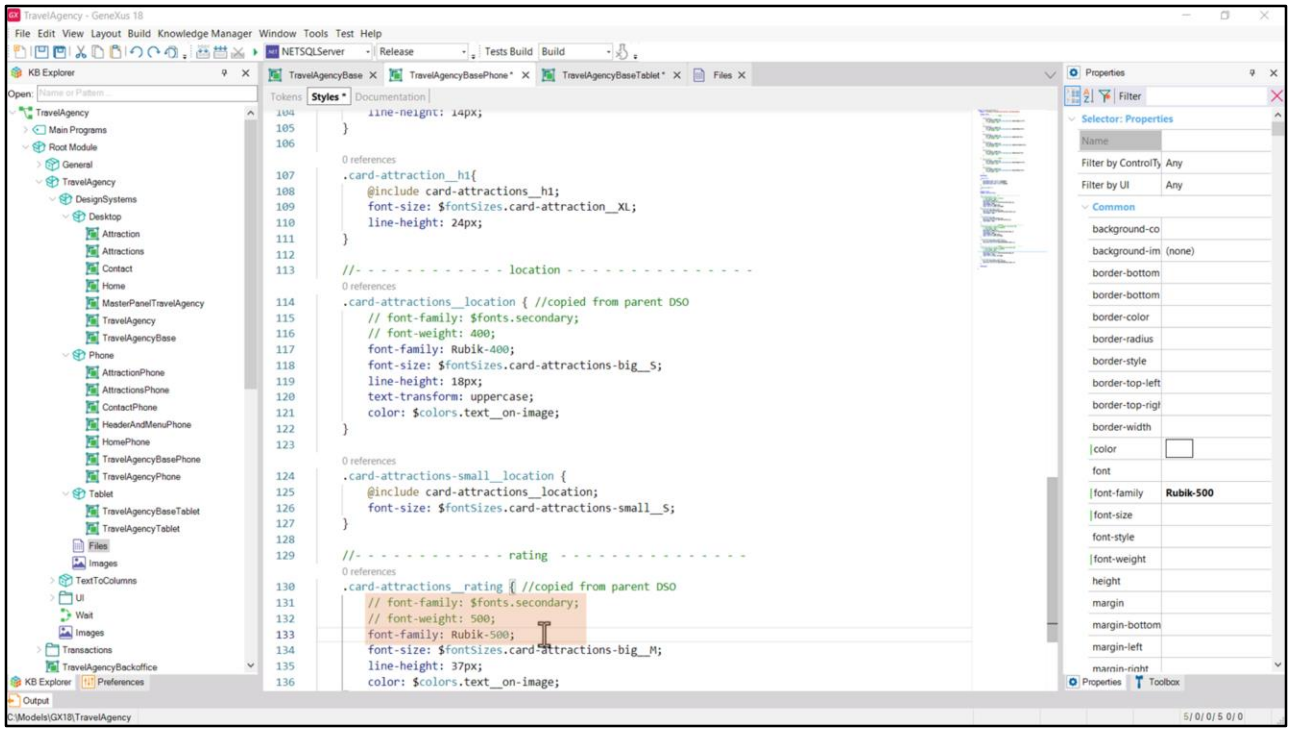
And the same goes for the other families.



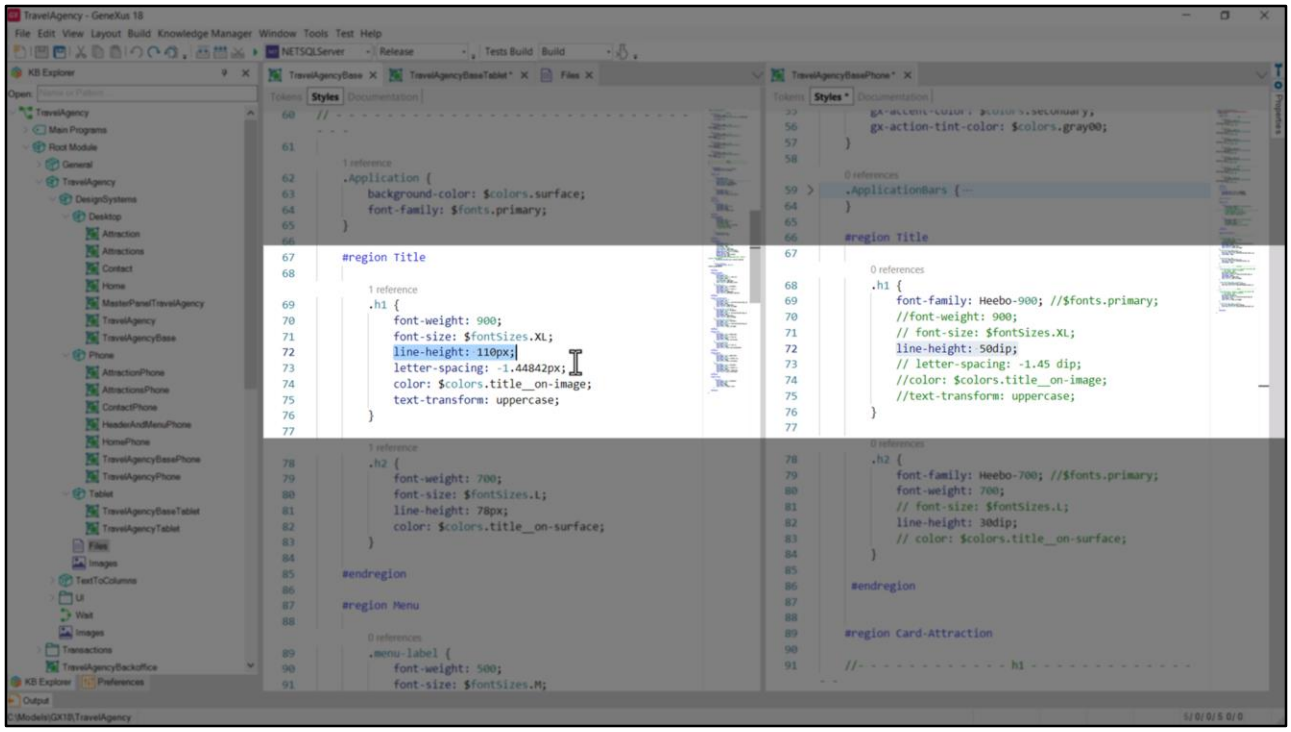
But this means that we have to redefine all the typography classes. For example, these ones...



This secondary font is Rubik... so we have to make this change...



And here is another one...

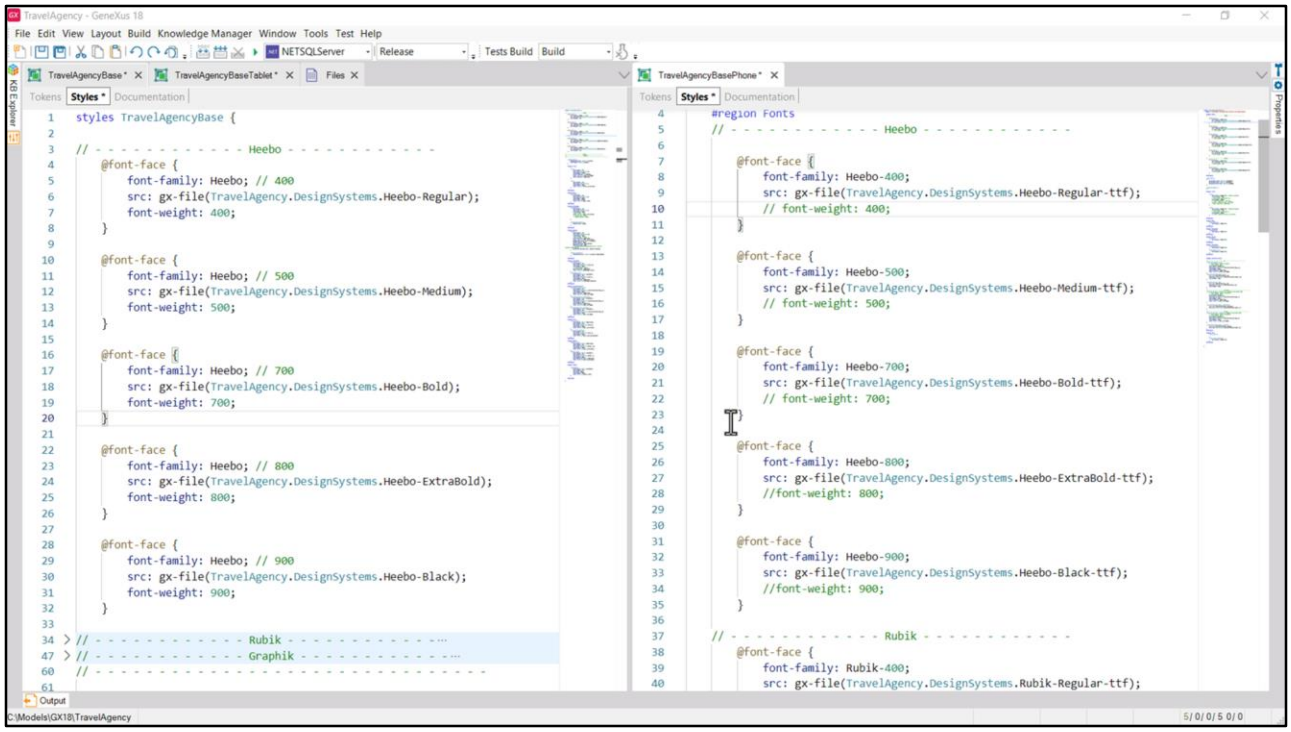


But not only that, but also, the classes that we didn't have to modify because they came from the TravelAgencyBase DSO will also have to be modified.

Here we see, and I take the opportunity to show you, how we have the class h1... to which we could also remove the font-weight...

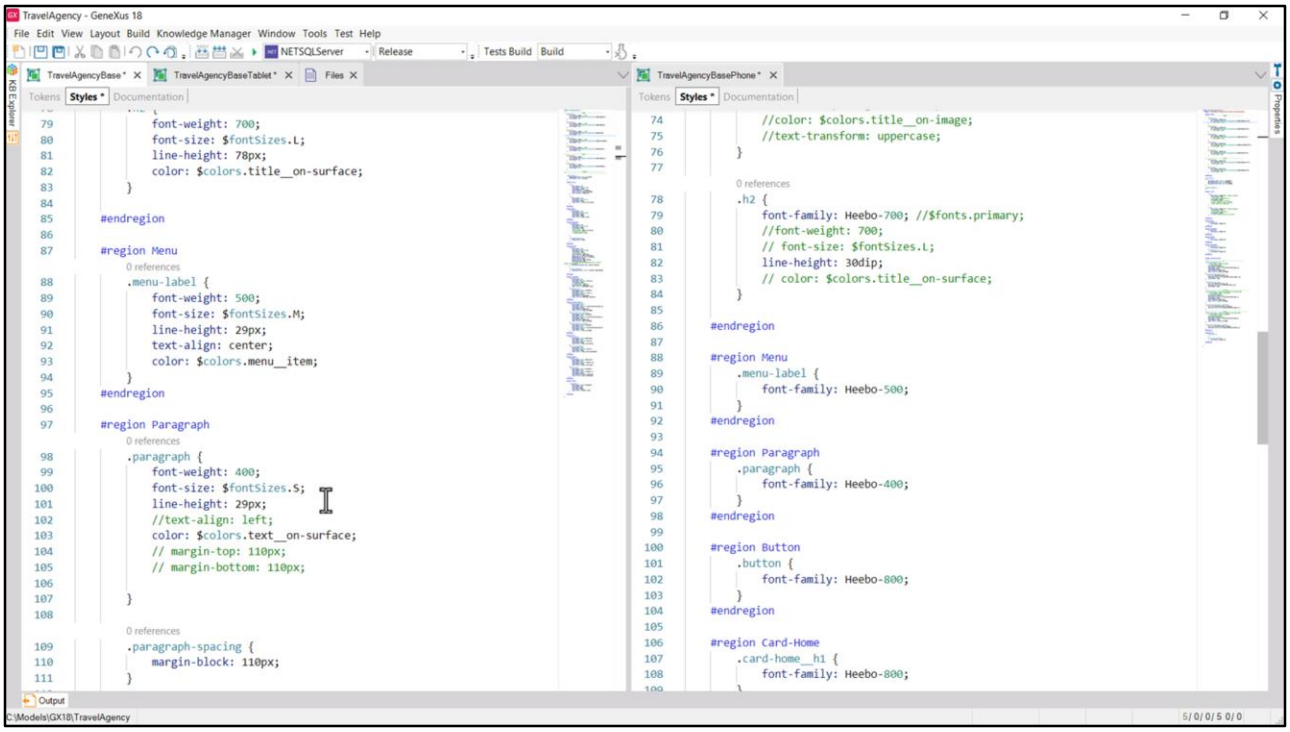
Note two things: first, that if in this DSO that imports this other one I want to change some properties and/or add others to a class, I don't have to copy all those that don't change. Those will be valid.

That's why I left all these commented, because they are the same; I don't have to indicate them again here.
But I do have to indicate those that vary or are added. In this case the line-height changes between Desktop size and Phone size. But if it didn't change, the point I wanted to show you is that having ignored this difference between Angular and native applications regarding font definitions makes it necessary to specify all the typography classes in the child DSO, even if they don't change anything, just to be able to indicate the font family correctly. And this is expensive.

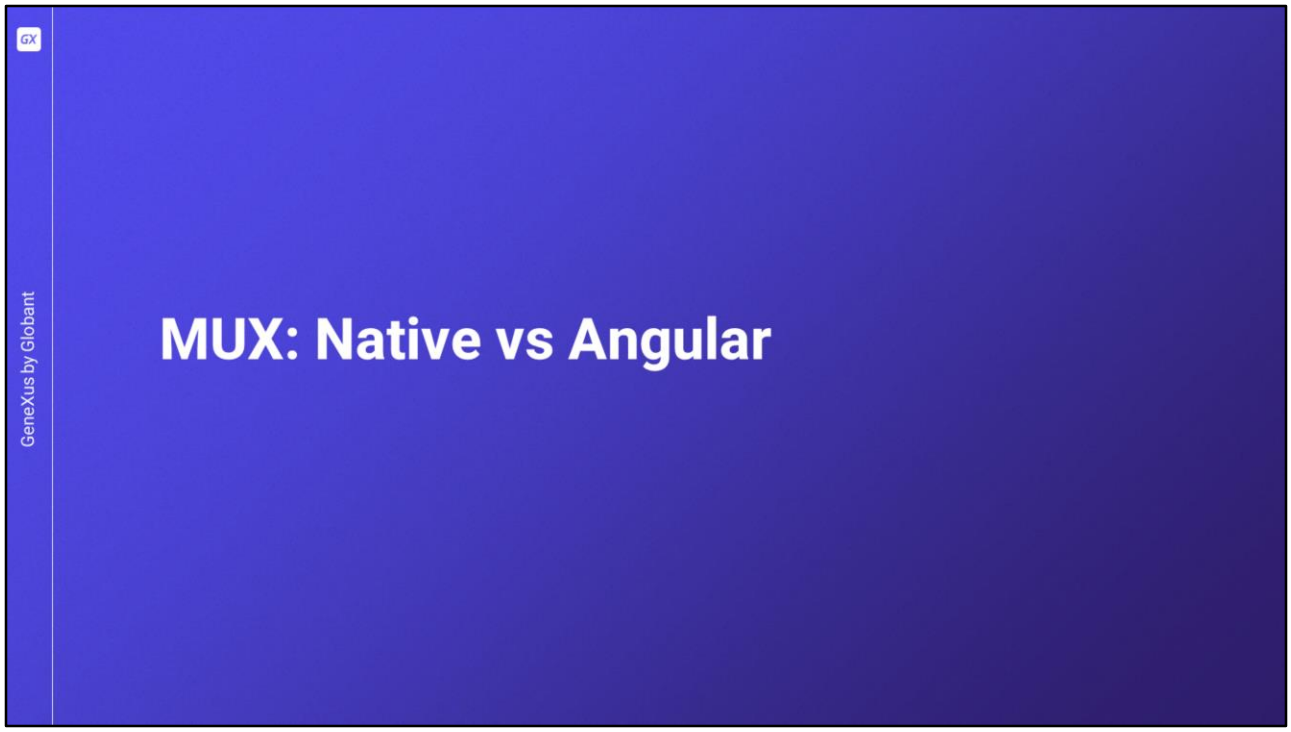


If we had known beforehand we wouldn't have chosen this solution that we implemented in Angular. If we don't want to touch the DSO for Desktop Angular we will have to select all these classes, at least to change their font-family indication.
This is how the changes would look like: we have to declare the font-face rules to identify the font family uniquely by name...

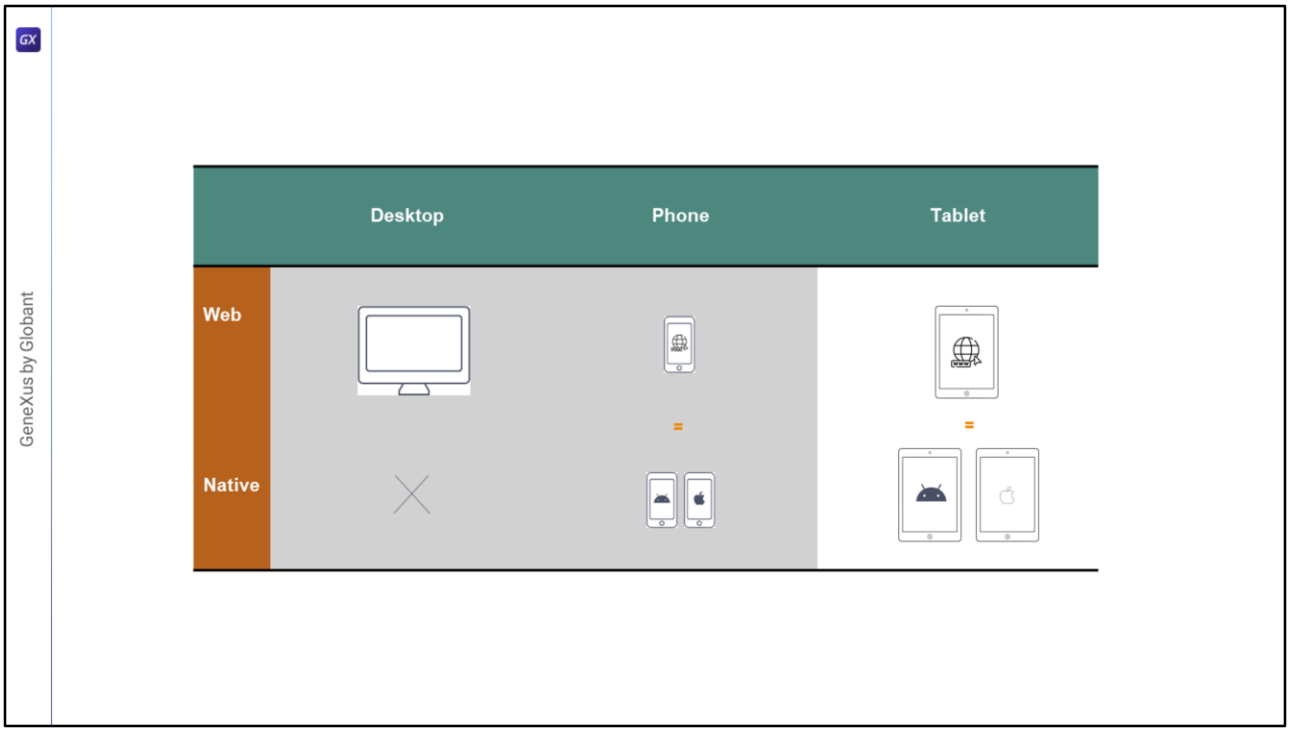
And then we have to add all the typography classes, even if only to indicate the font family by name.



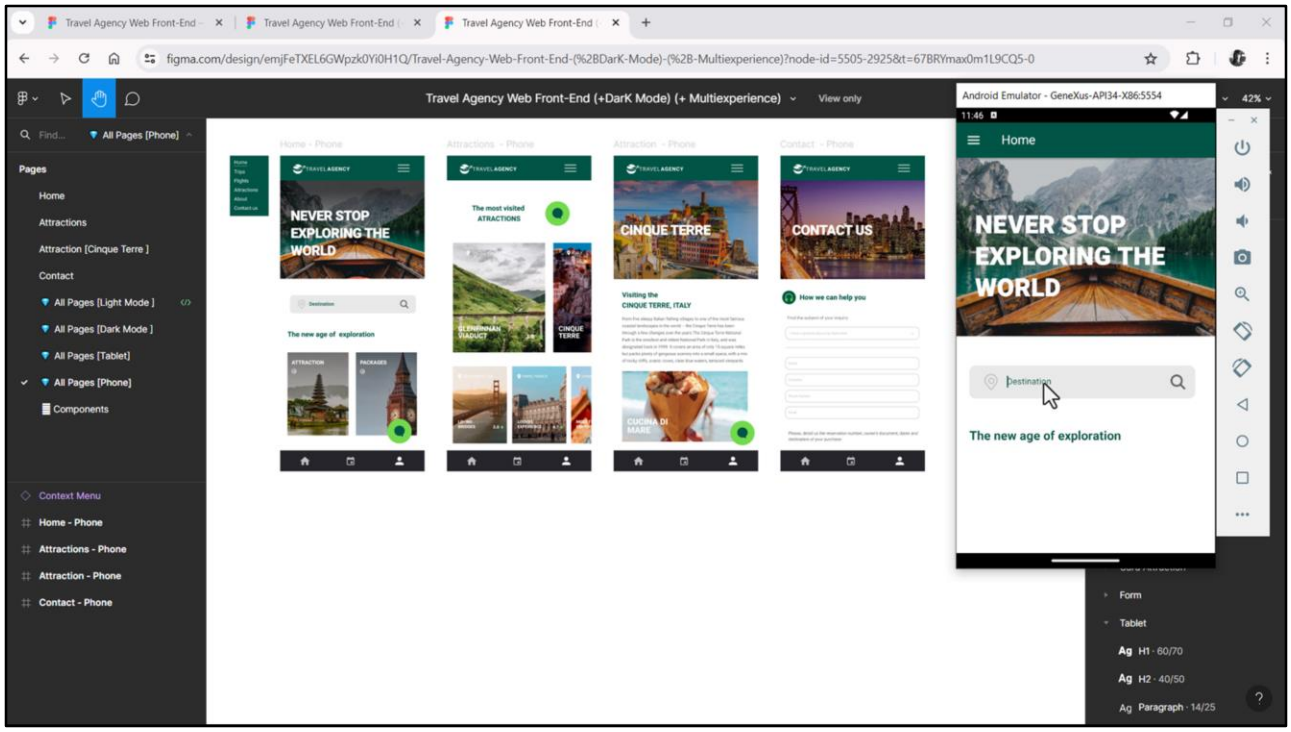
I changed something else in these ones, and I didn't analyze these others to see if I should change anything else: I only changed the font family. But note that I had to do it one by one with all of them.



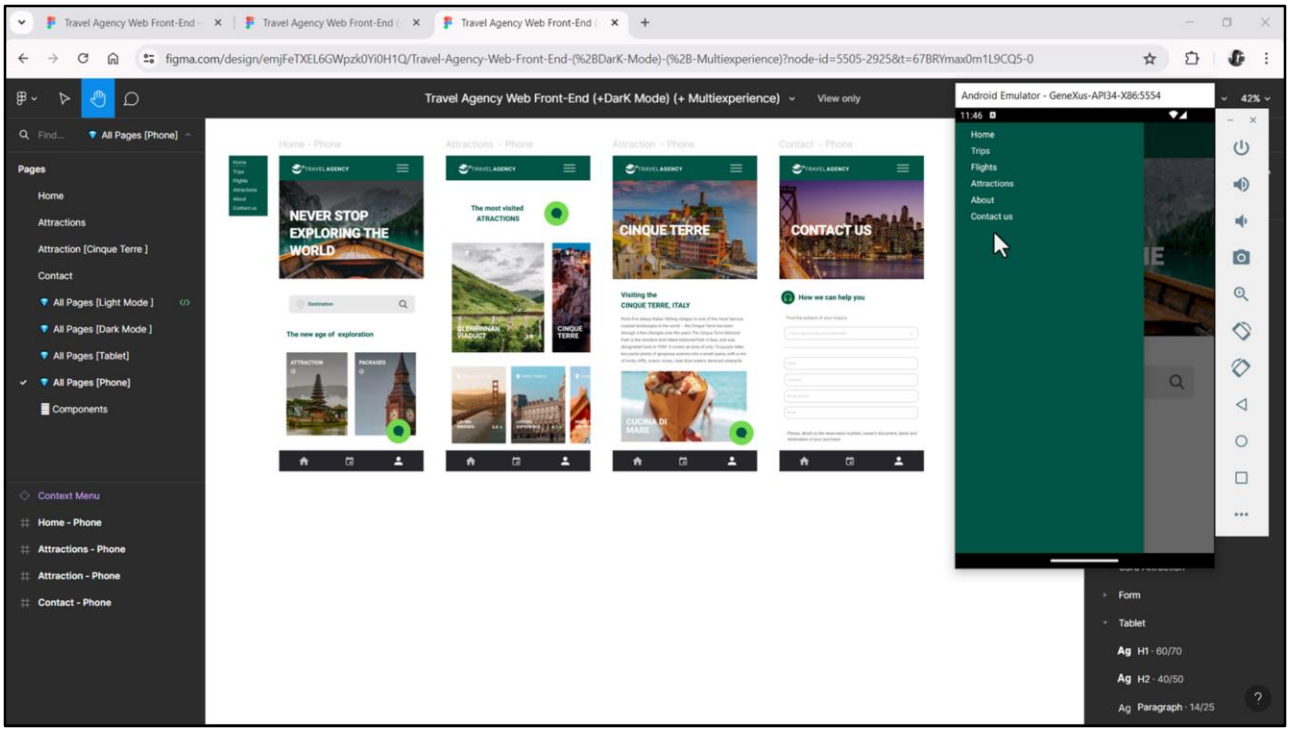
This was probably troublesome for you. It was just an example of the obstacles that can emerge when implementing multiexperience and that can be reduced to a minimum if you know the differences from the beginning of the development.



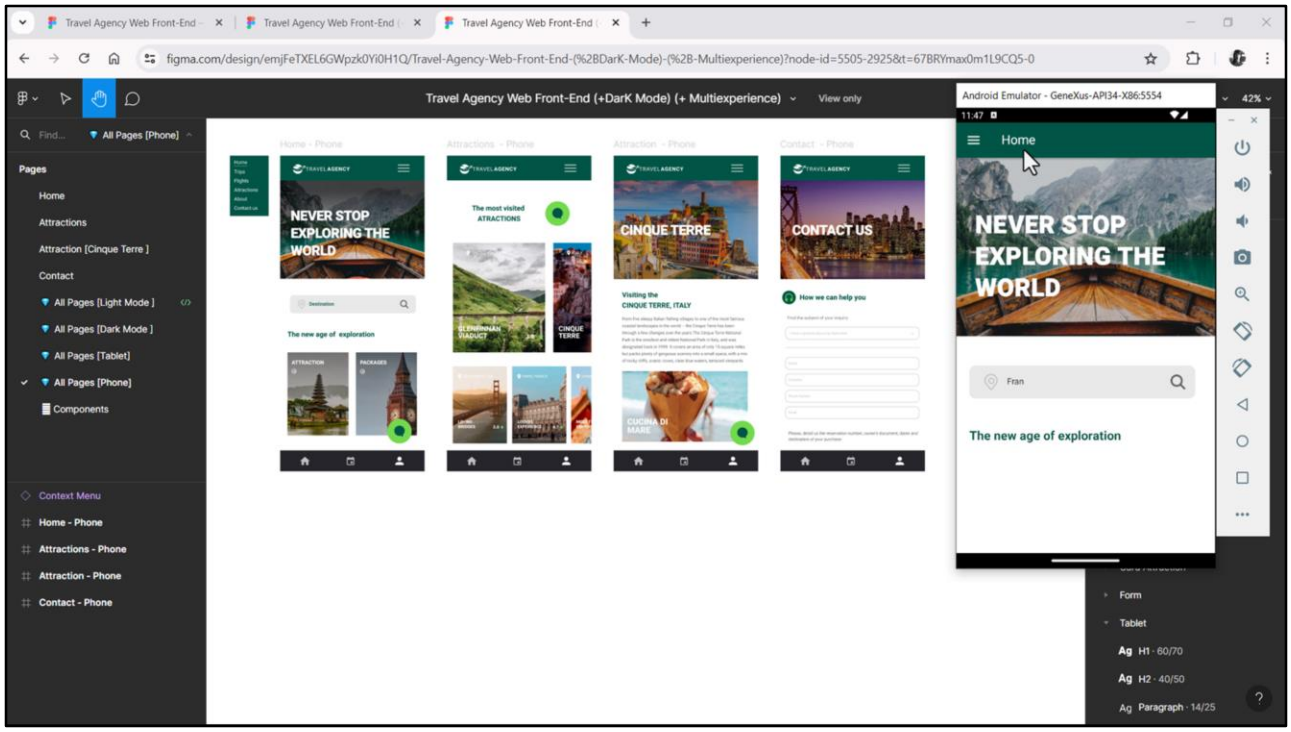
The reason is that native applications haven't been built on the CSS paradigm and this will have an impact, although the aim is to bring the two worlds as close as possible in GeneXus.



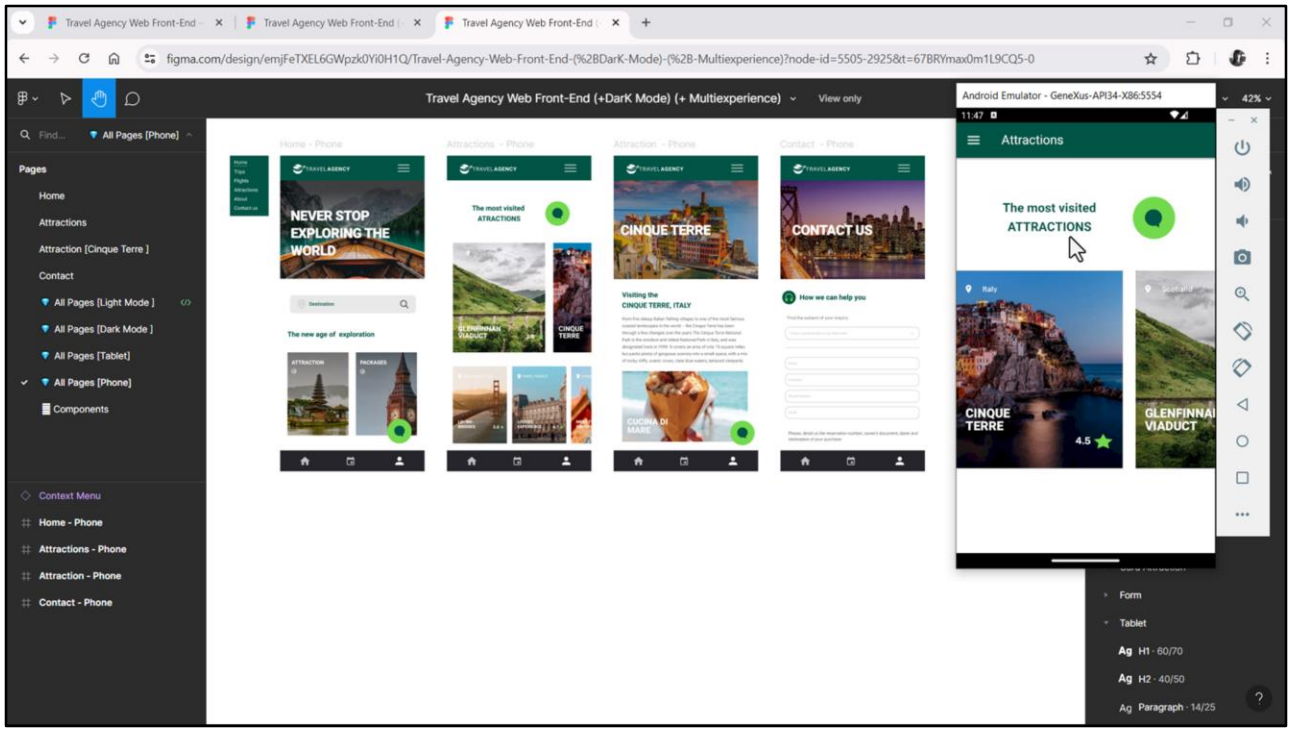
To inspire you, I advanced quickly in my KB so that you can see in this emulator how the screens of the Android application for Phone look like.



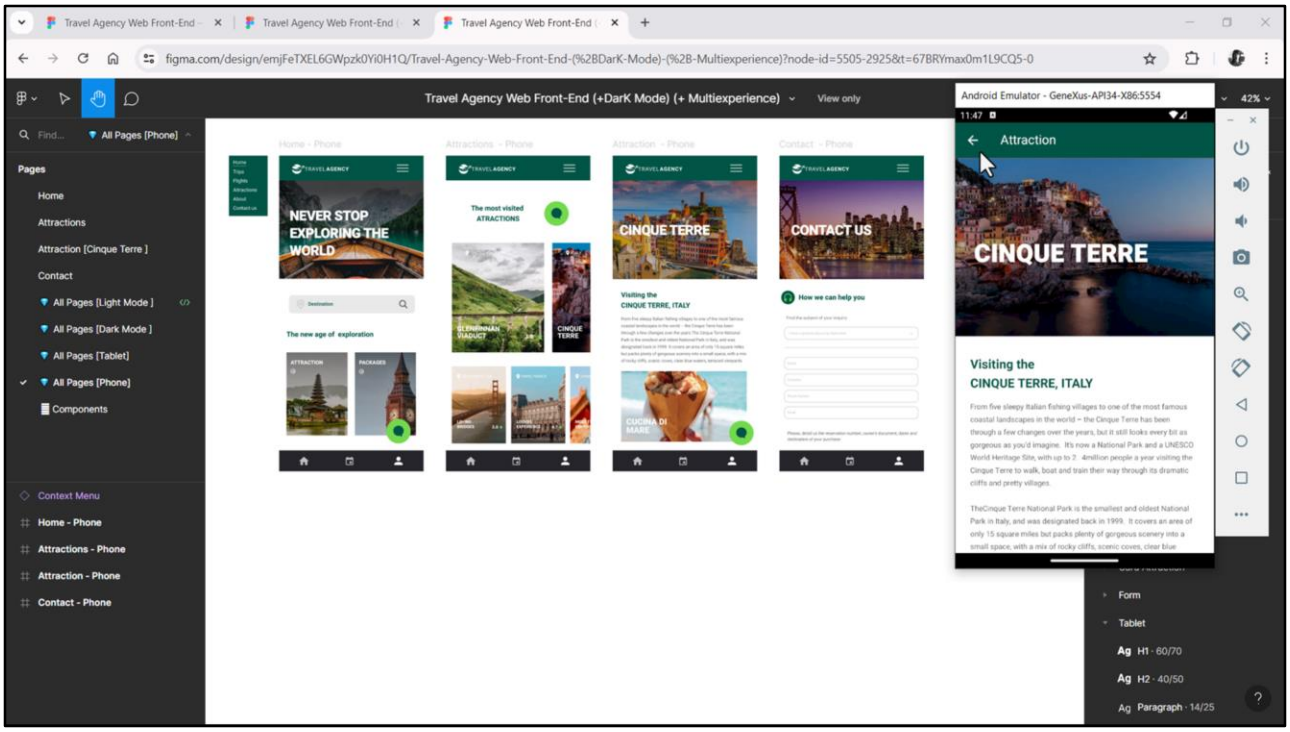
For example, we see the hamburger menu (on the left, as is the standard in their operating system, and not on the right, as proposed by Chechu's design)...



We see as title in the Application Bar the name of the object being loaded, and not this icon and text that I will show you later how to change in the simplest way.



Note that in the Attractions panel I removed the Header, as indicated in the design, and that for now I implemented the carousel as a horizontal grid (to make it quick and easy)...



...and that if I tap on an attraction it takes us to the Attraction panel and automatically, and also according to the operating system's own mechanisms and Android's design guides, the back button appears to return to the object that called it.

Okay, I'll stop here so that it doesn't become boring and we'll continue in the next video.


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