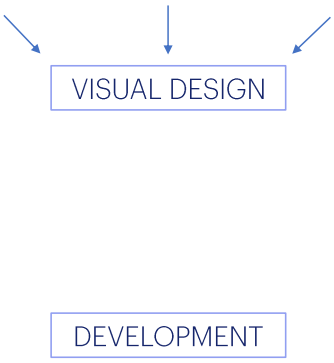
UX Design

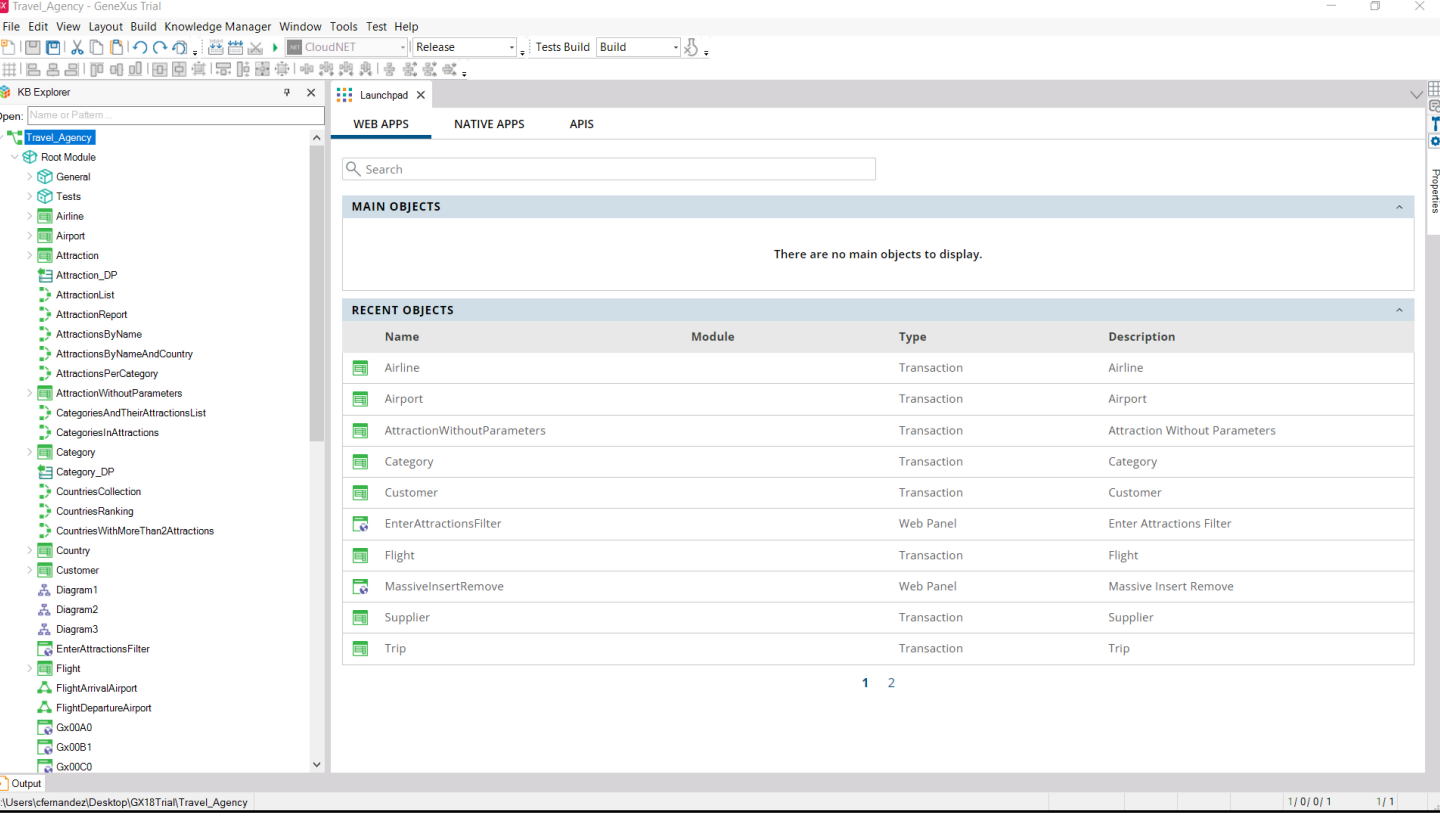
Introduction

GeneXus™



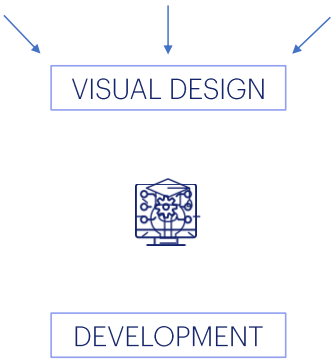
So far, we have focused only on the data model and behavior, and we have not been concerned at all with the visual design of the application.

We could say that we have been doing only development tasks.



Thus, we have created and developed a set of objects: some transactions, some lists, some web panels created automatically by applying the Work With pattern, and others that we created from scratch.

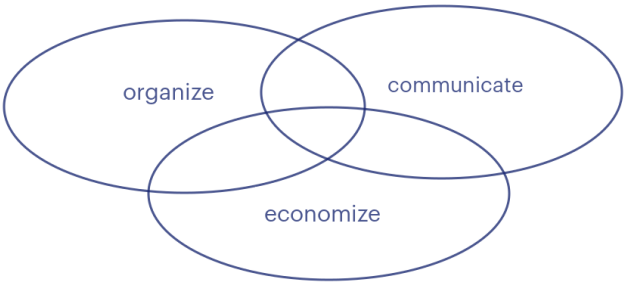
However, we don't have an organized application yet, and we haven't even started to envision it that way.



That is because we have been learning the basics of GeneXus, focusing on some parts, but, understandably, this is not the approach we will take when we start developing a real-life application.

UX DESIGN

DEVELOPMENT



In fact, the development of an application does not start with the development itself; it begins after a very important previous stage, which is the **user experience design**. There, the conceptual structure of the application as a whole is organized first. The objective is to obtain a product that meets the expectations of both the user and the business, and that communicates exactly what it should in the most effective and economical way possible.

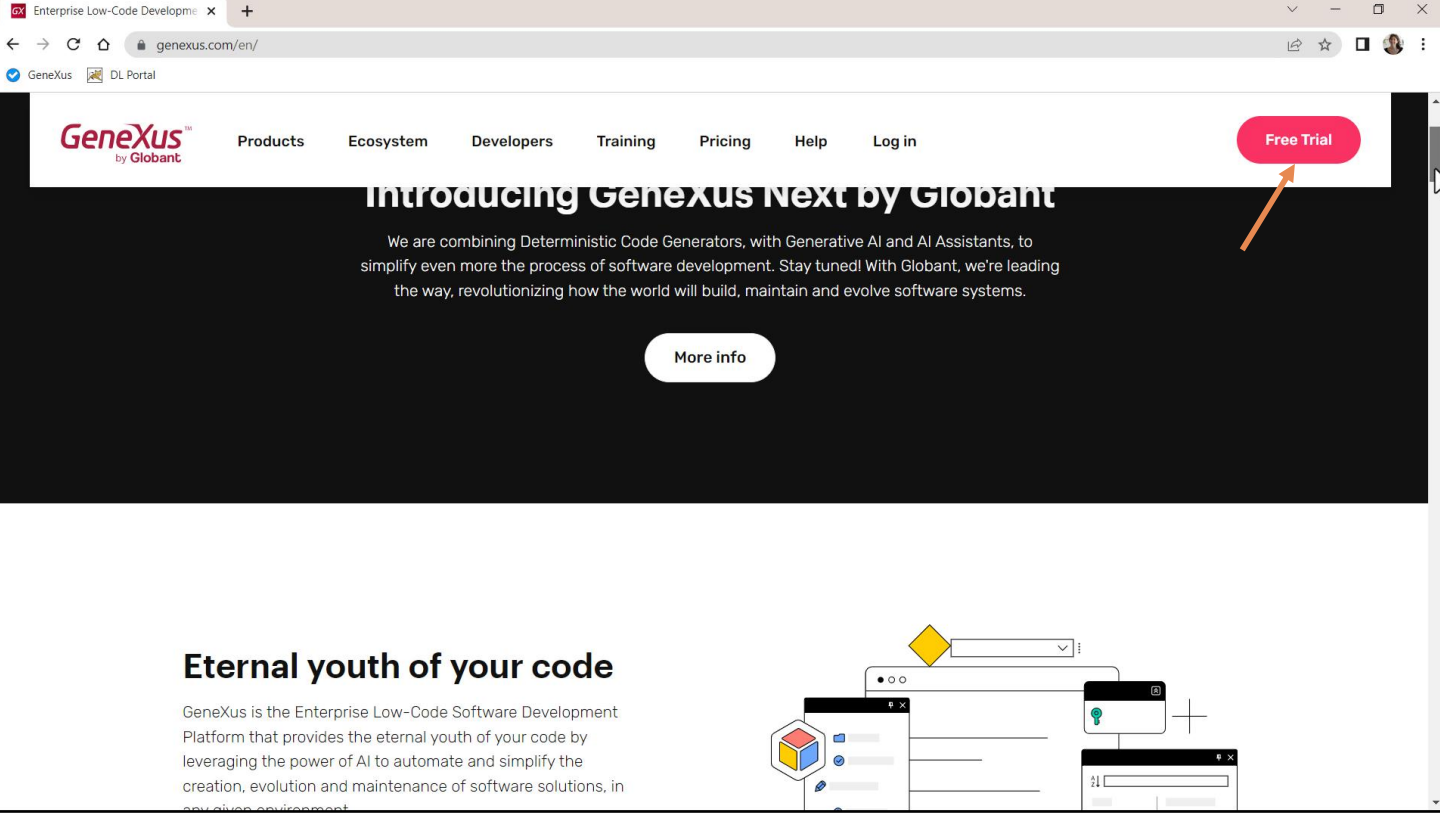
UX DESIGN

DEVELOPMENT

5	Surface	VISUAL DESIGN		
4	Skeleton	Interface design	Navigation design	Information design
3	Structure	Interaction design		Information architecture
2	Scope	Functional specification		Content requirements
1	Strategy	User needs & Business objectives		

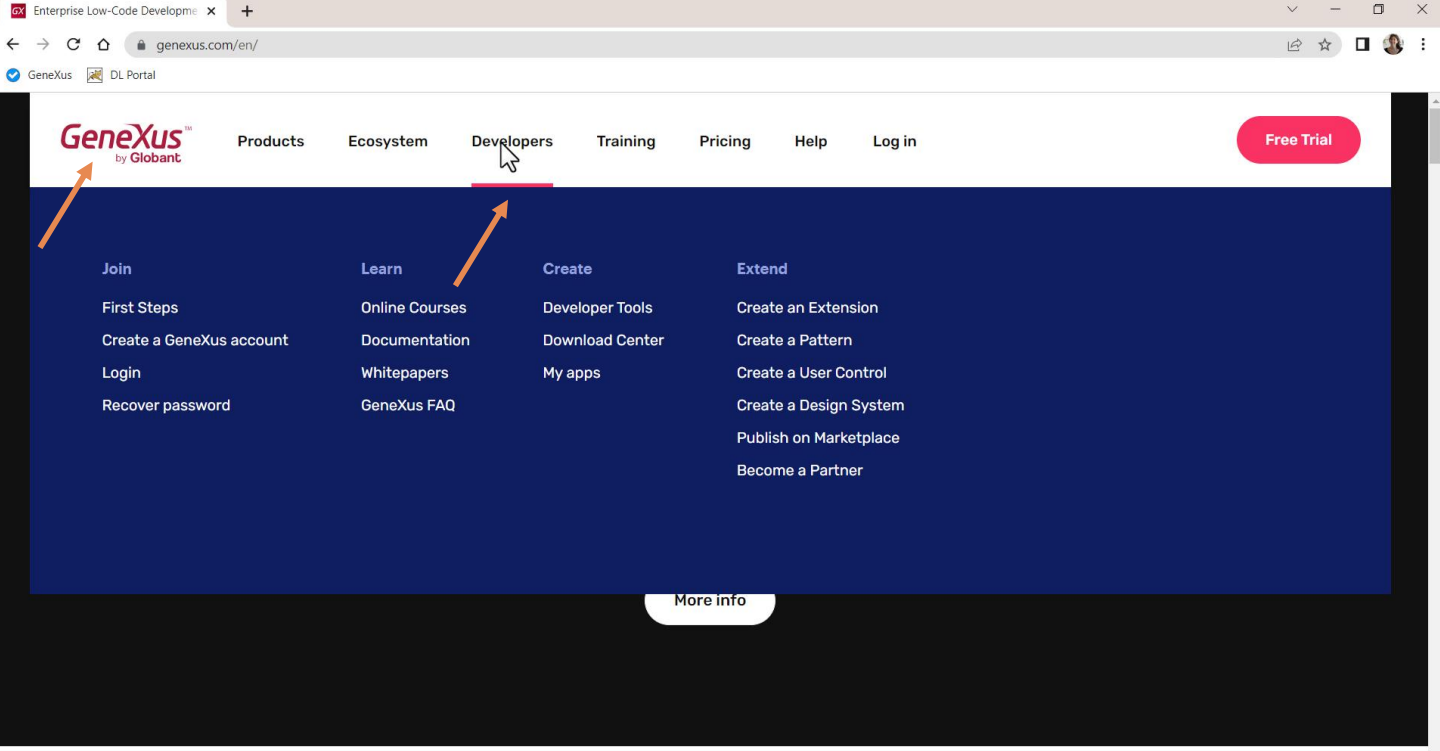
This includes the interface design as well as the navigation and information design. The visual design we see in the finished product is built on this organizational skeleton. Although we will not go into the details of this very important task here, we can realize that we will end up viewing the result of the foundations laid by these previous stages: each one is the basis for the next.

Only with the resulting surface stage will we have the visual design to start the actual development.

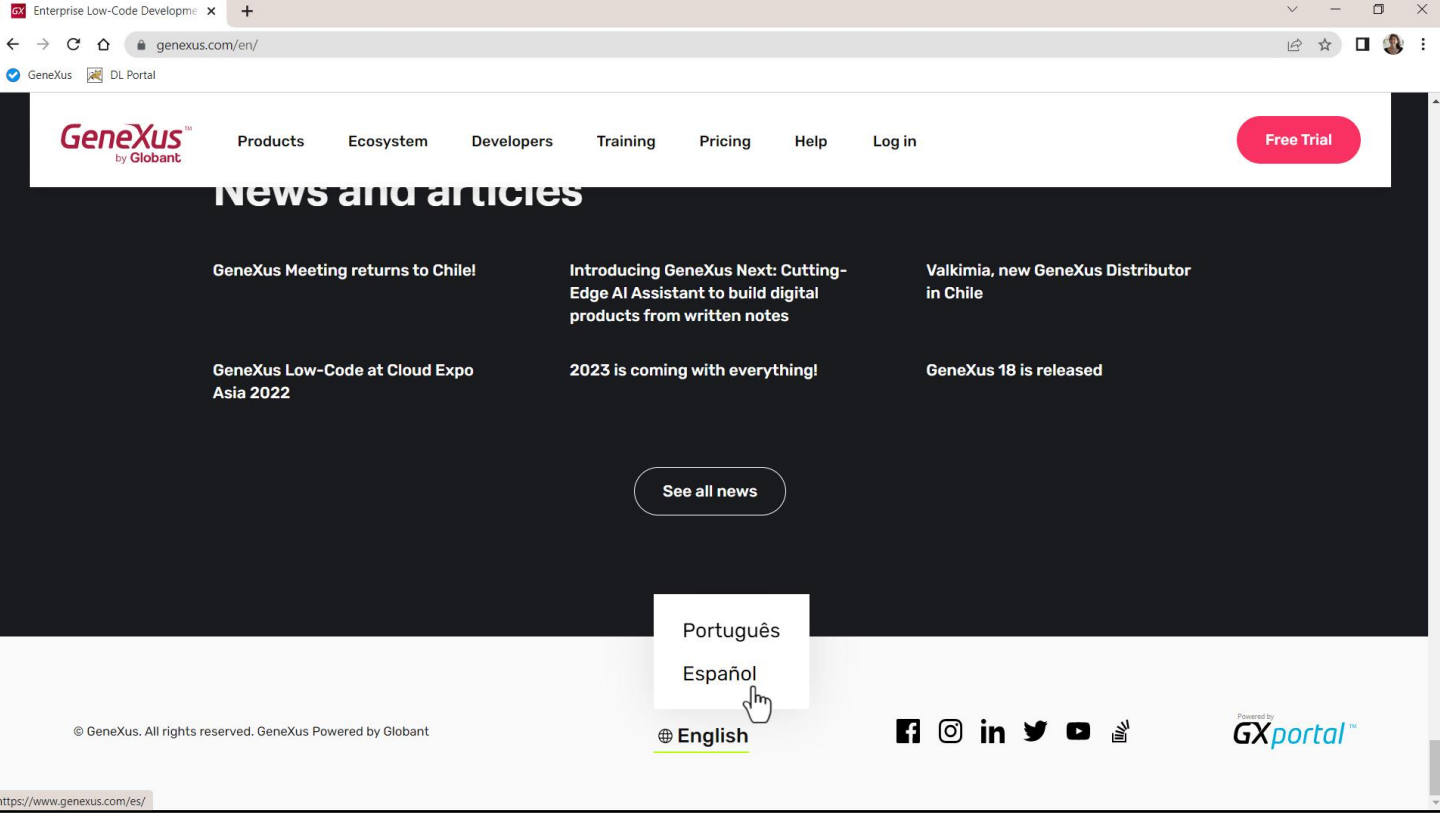


Looking at the GeneXus website we can quickly grasp the **U**ser e**X**perience decisions that were made when designing this web application: a basic principle is to achieve a good contrast, and here we see that it was done with white, black, gray, and red.

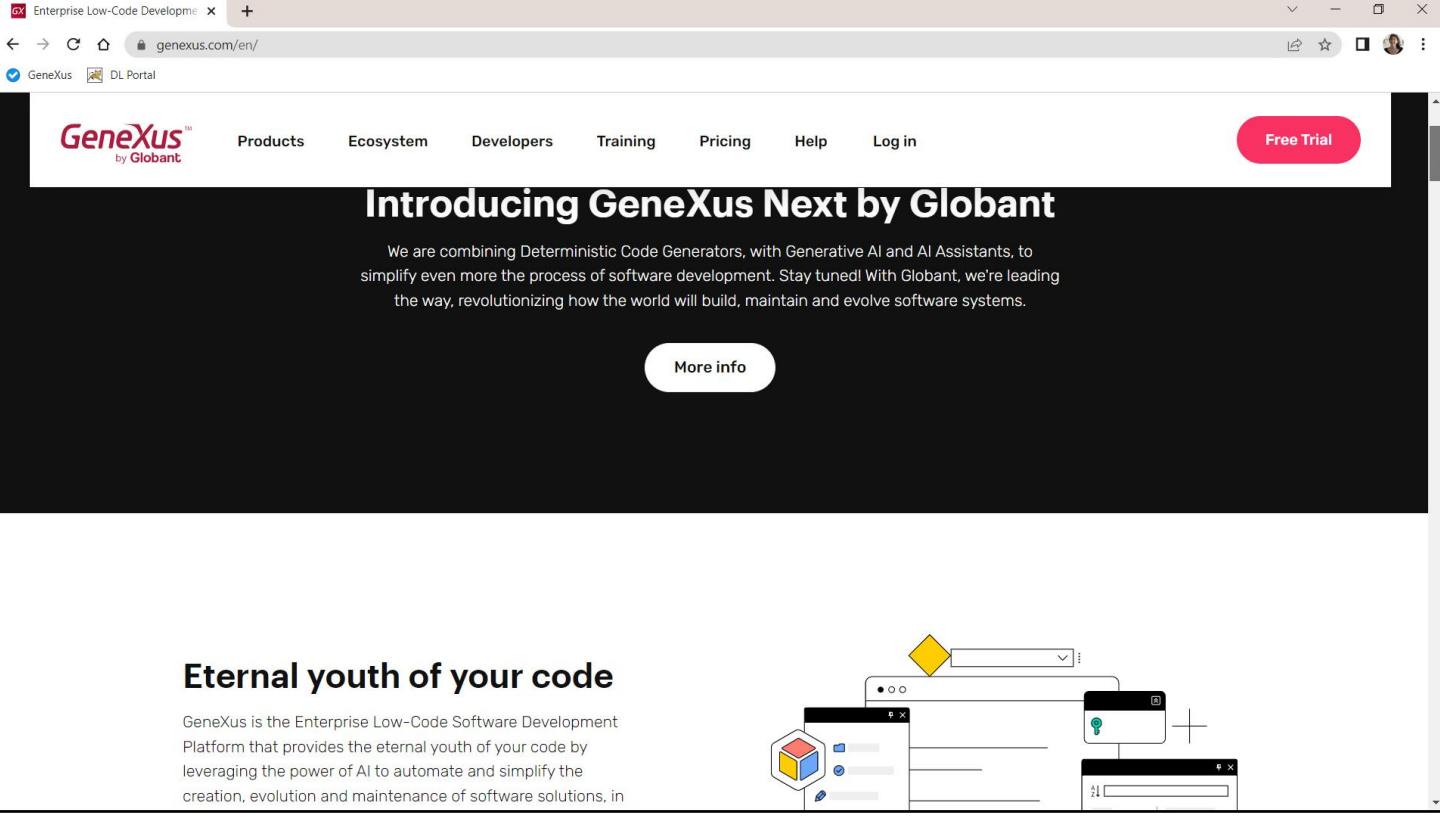
The most important actions that the user can do are shown as buttons, which have a consistent style and therefore can be easily recognized. Their color depends on the background (black or white), but clearly the main color for actions and highlights is this one.



In fact, if we look at the main navigation menu that will appear as a header on all pages, the submenus are displayed when going through the options and this is the color that indicates the active option, which looks similar to that of the company's brand.

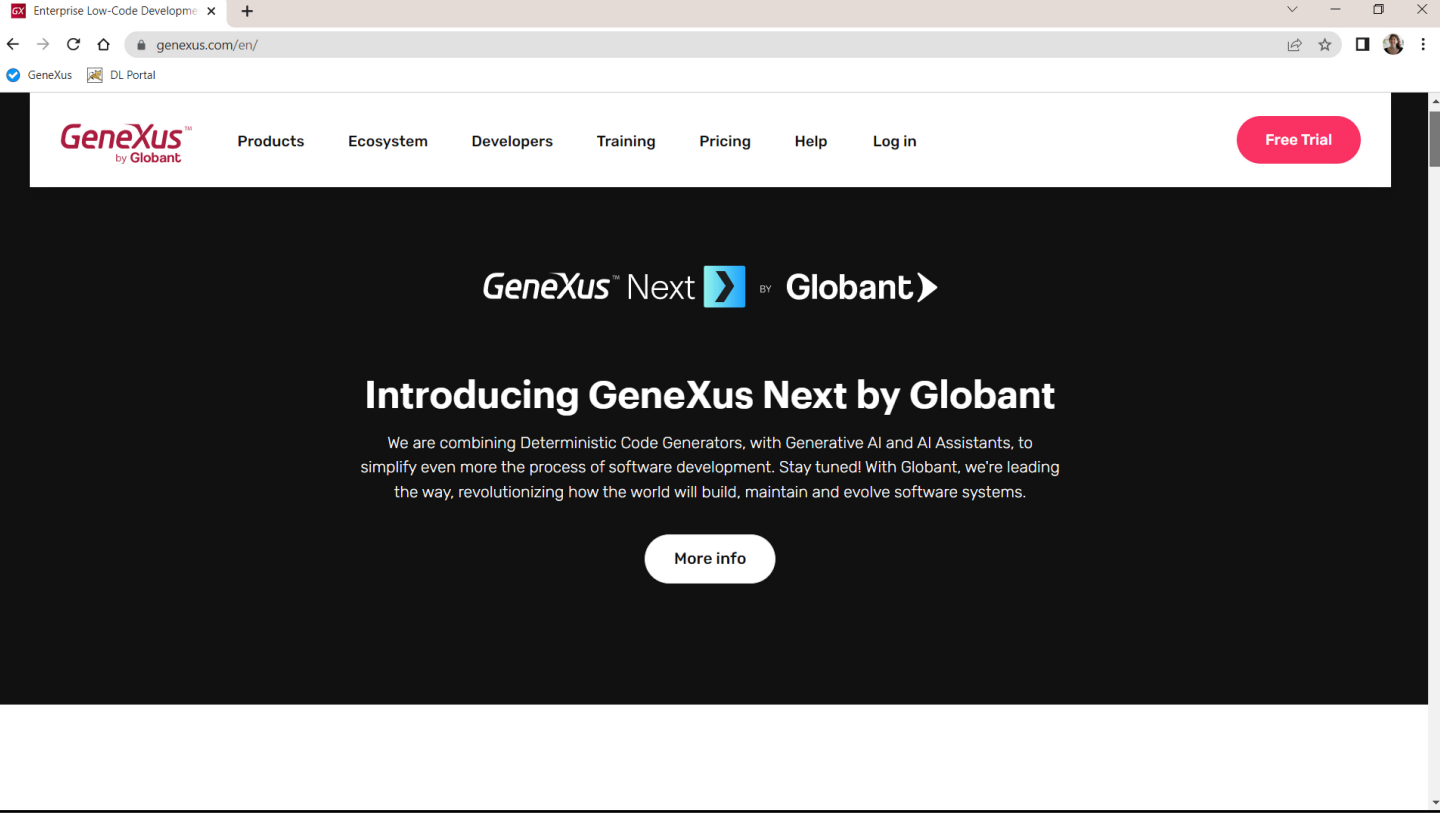


We can also see that there will be a footer repeated on all pages, which is where the language can be changed, following a convention that all internet users are familiar with: the language can be changed in the header or footer.

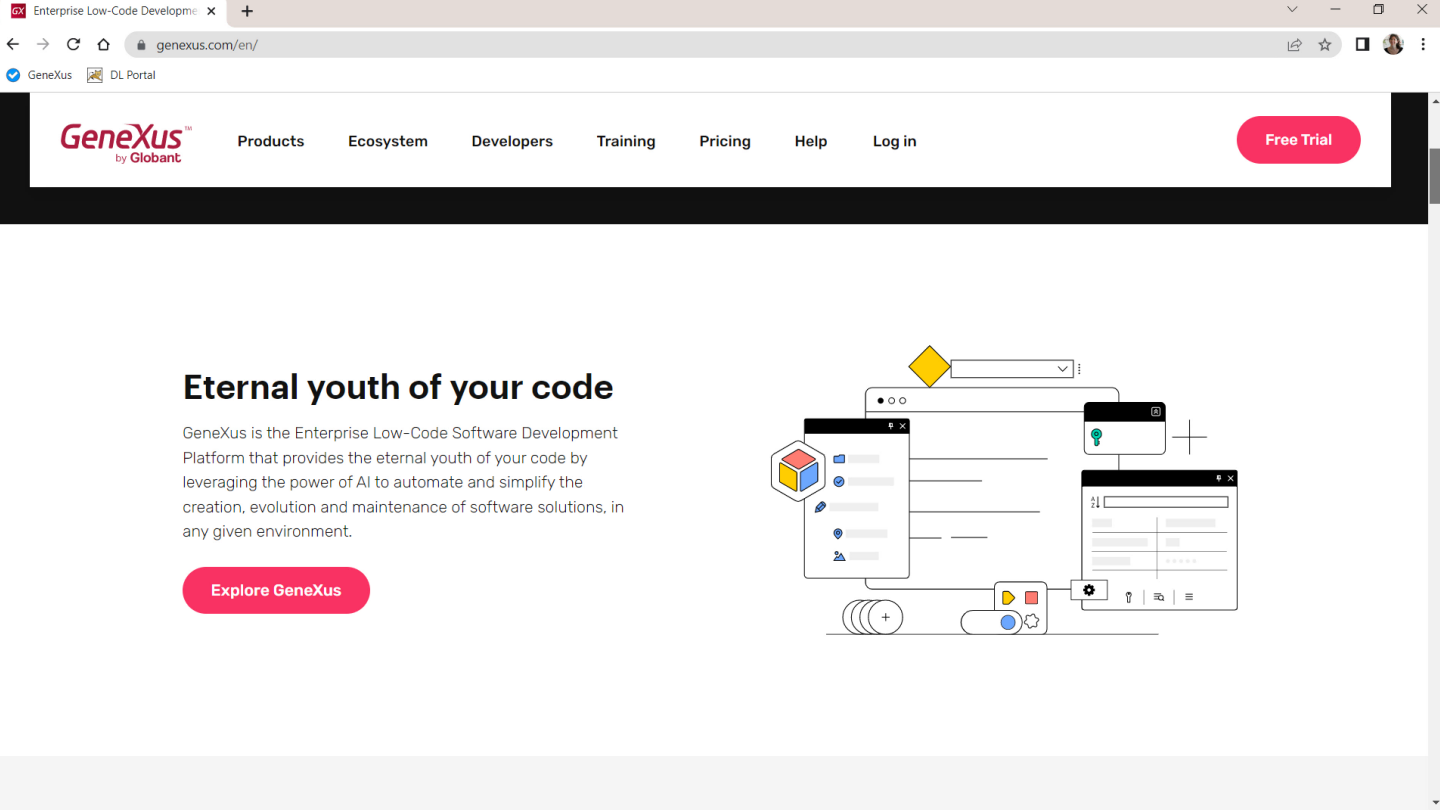


Also, note that the font families are consistent, and that they are used in the same way: the same sizes for the titles, descriptions, and so on.

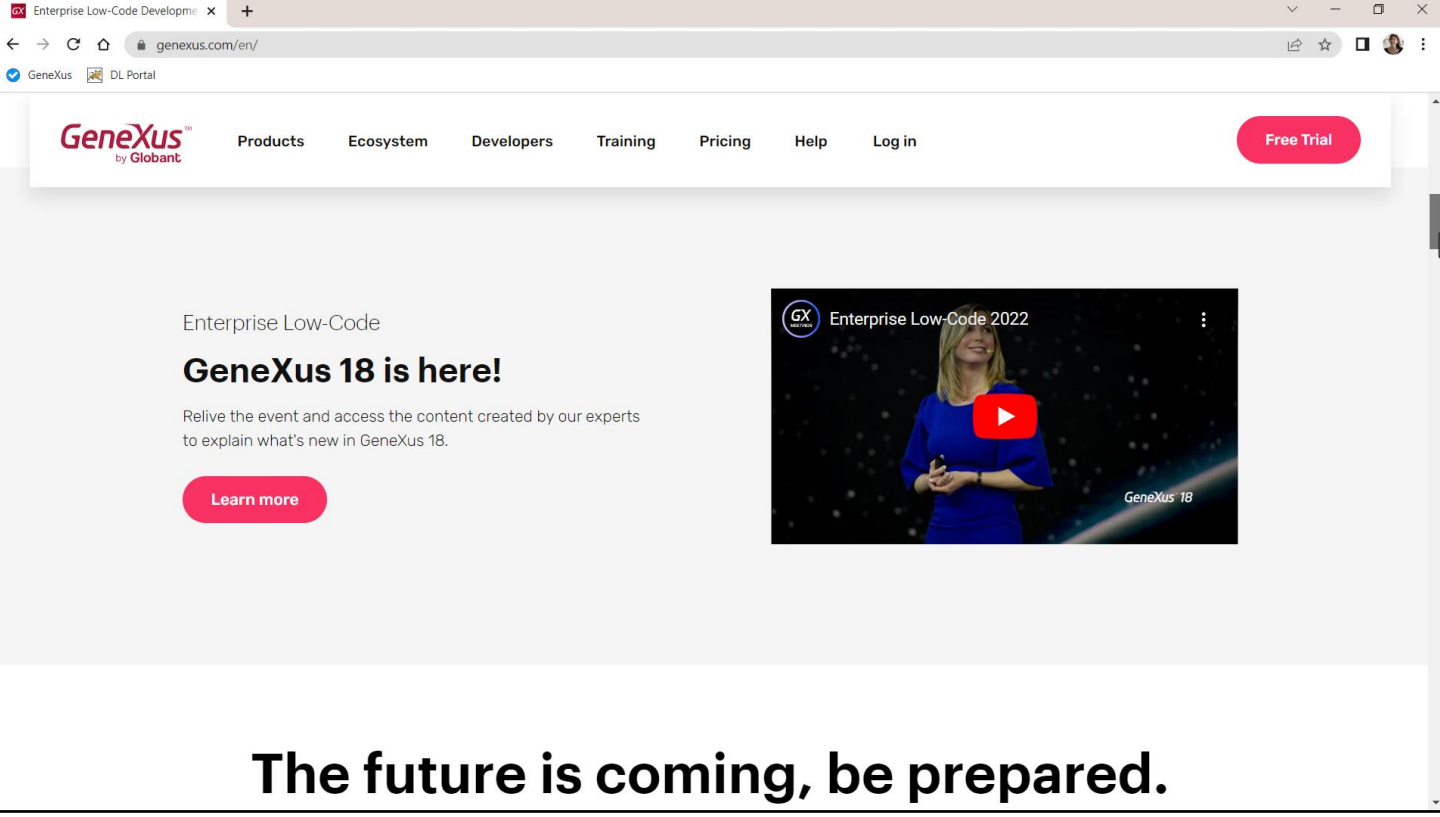
In addition, it is possible to indicate which information is related and must therefore be interpreted as a whole, based on how closely the controls are placed and the spacing in relation to the others, and by alternating the background color.



So, here we have the first set of information....

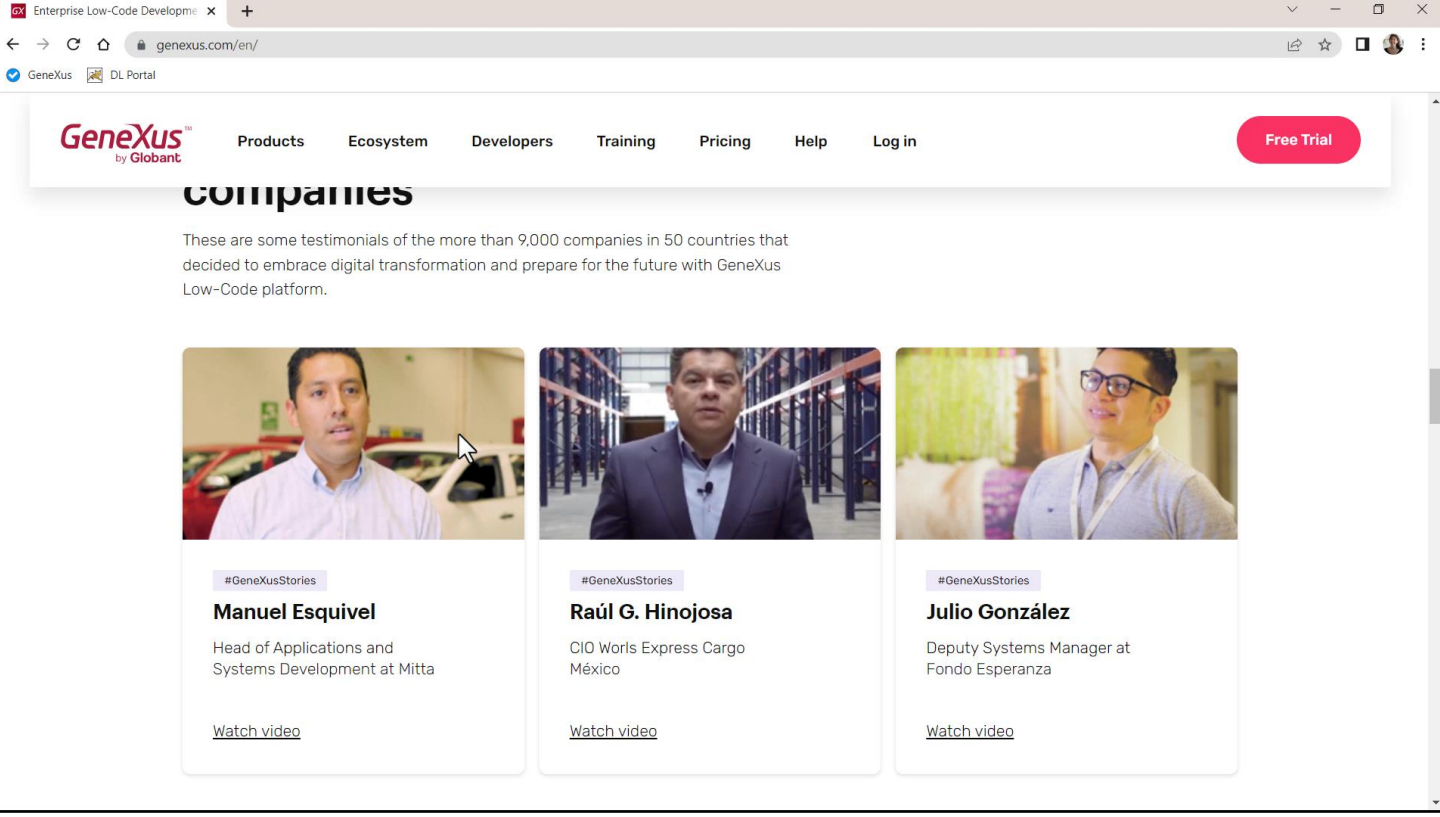


...here is another...

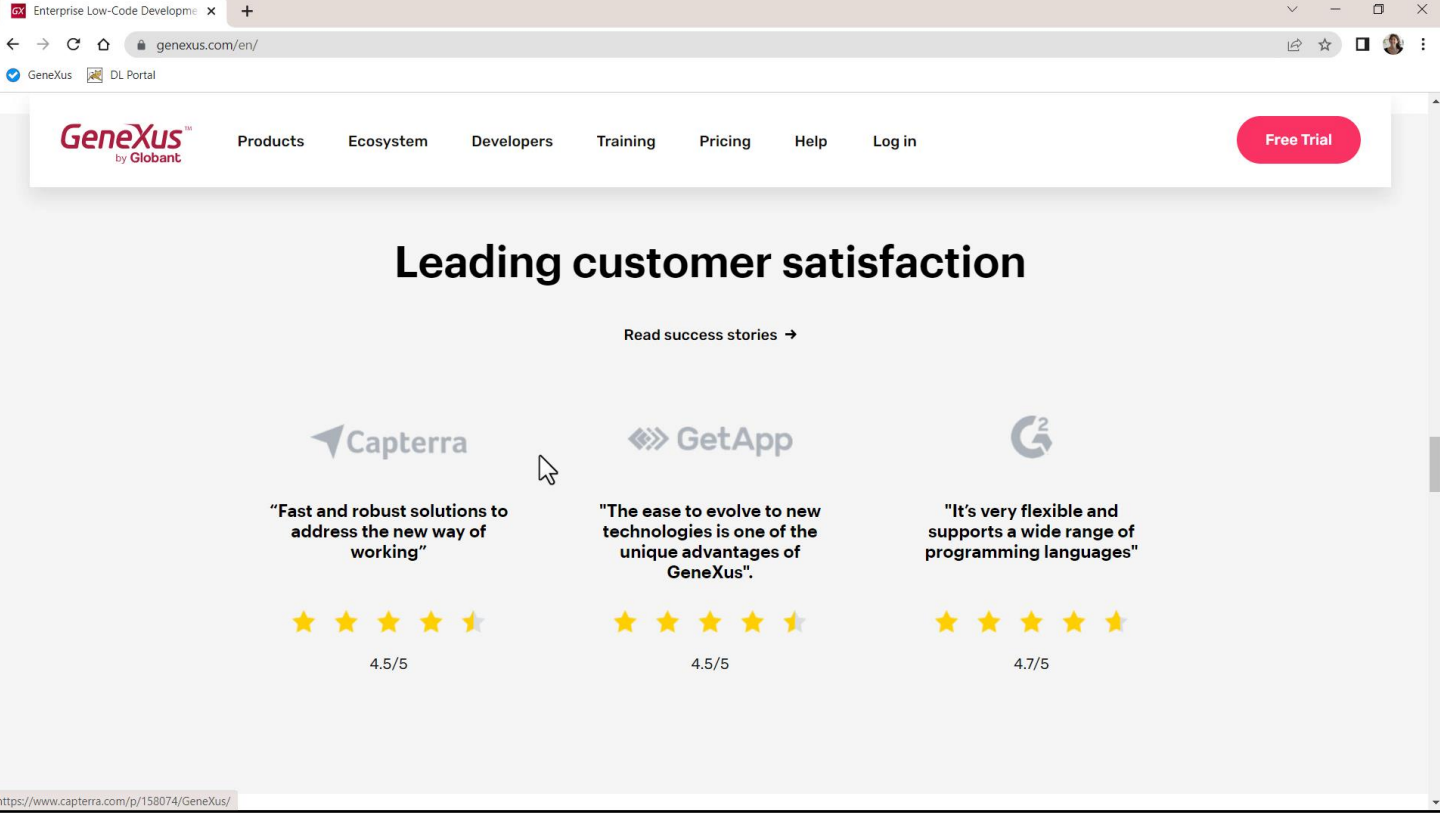


The future is coming, be prepared.

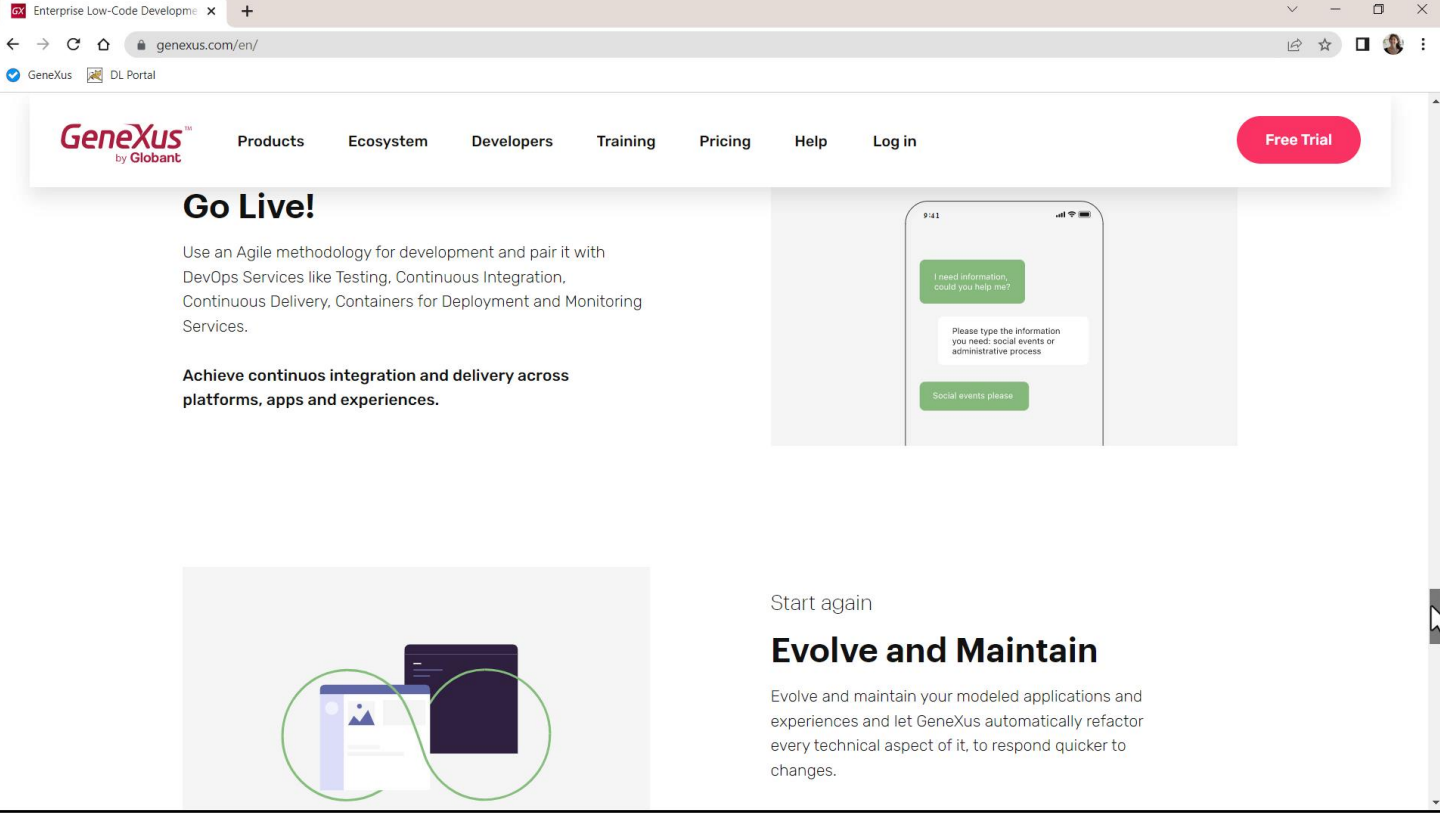
... and another one with a light gray background, and so on.



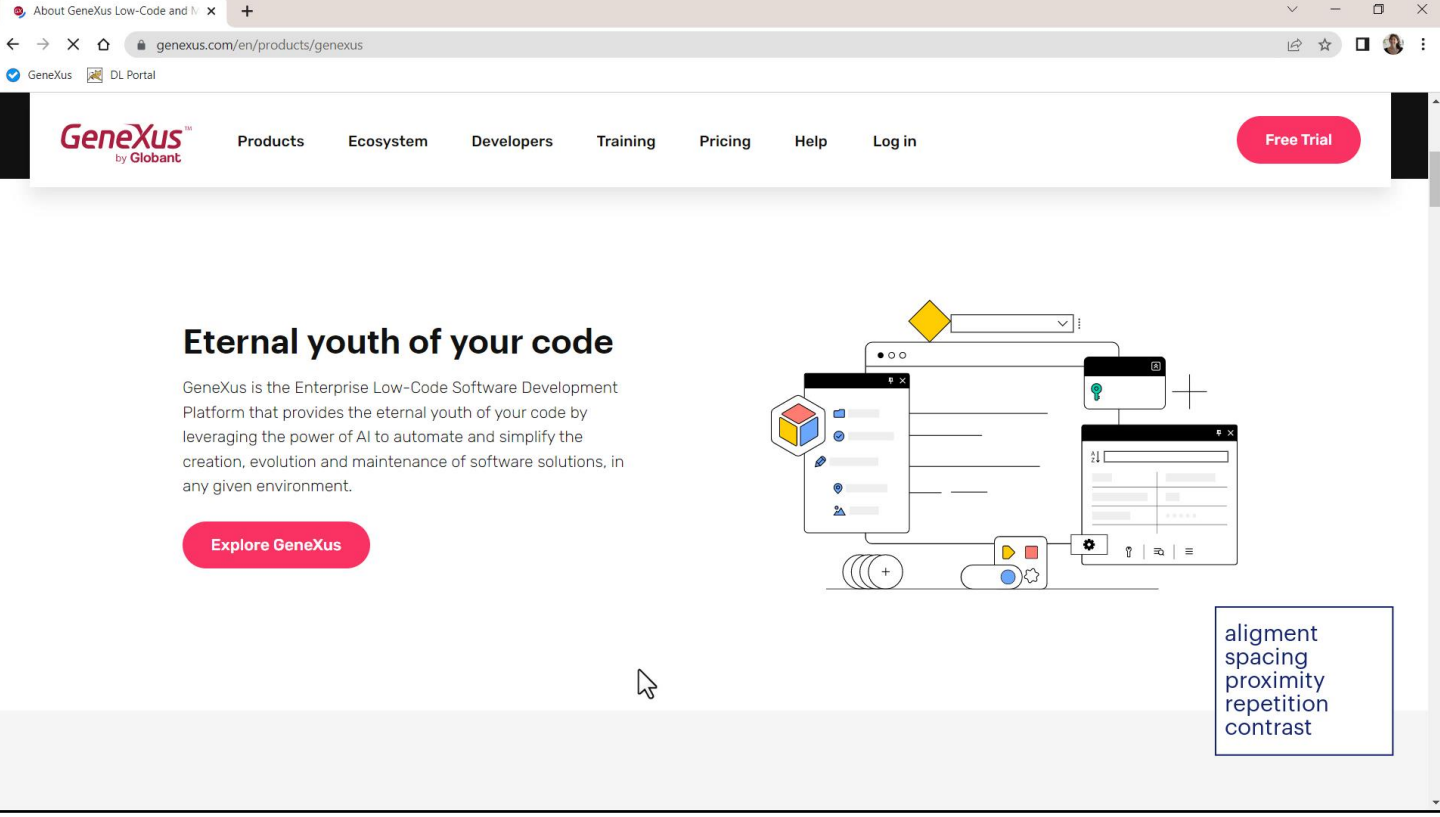
If we look down here, we see that this all corresponds to a block of related information, with this repetitive pattern indicating that there are 3 subunits of information that are at the same level.



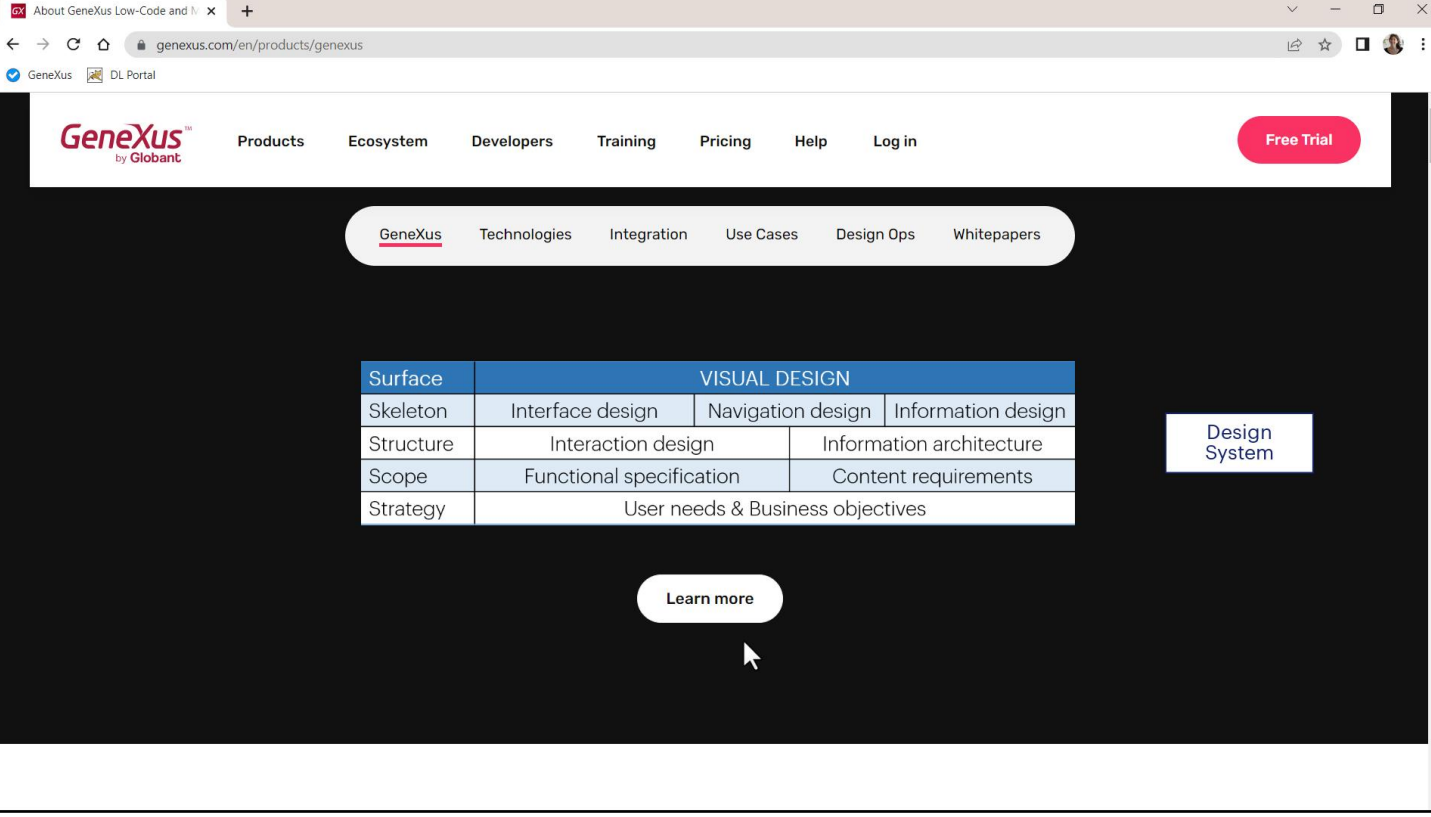
Here we see another block where the user will infer a little bit the same as before. There are cognitive patterns that every user will recognize.



For example, let's look at another one: here we find another block of information, where blocks of information alternate with text on the left, an image, and then the same two things but inverted, to alternate again. Clearly this is a narrative block of information.



Therefore, alignment, spacing, and grouping of controls will be essential and we must know how to implement them in a layout.



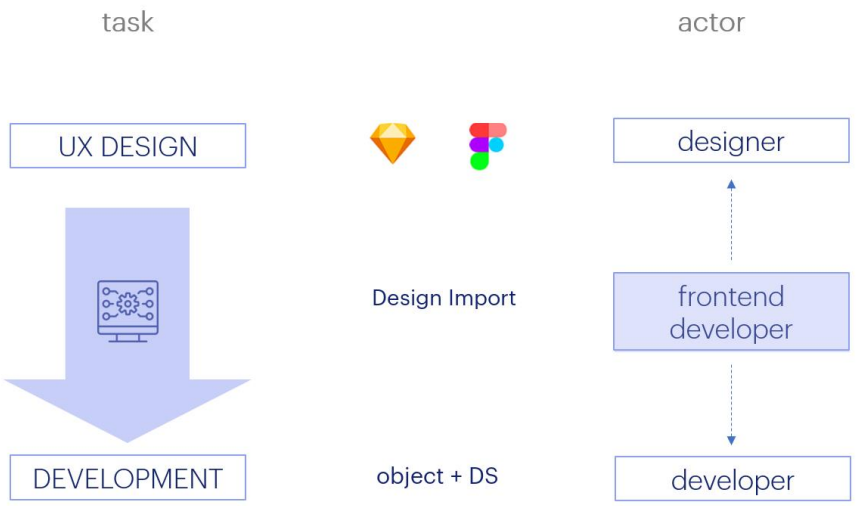
Everything we've analyzed corresponds to both the design skeleton and to the visual design itself, where the choices of colors, fonts, and so on are materialized.

Behind a given design there is a number of fairly abstract decisions aimed at providing standardization, consistency, and ease of use. We speak of Design Systems because the design of an application or family of applications works as an interrelated system.

Until now, we have focused on a single role—the developer—who is in charge of development.

But if we look at this new task, that of user experience design, we find another role that takes the lead in this stage: the designer. He/she usually makes decisions such as the ones we have seen in the example regarding the specification of the Design System.

We would be missing the intermediary role that is the frontend developer, who is in charge of the effective and specific implementation of the design IN the application. In this case, in GeneXus using the tools provided by GeneXus to work with Design Systems.

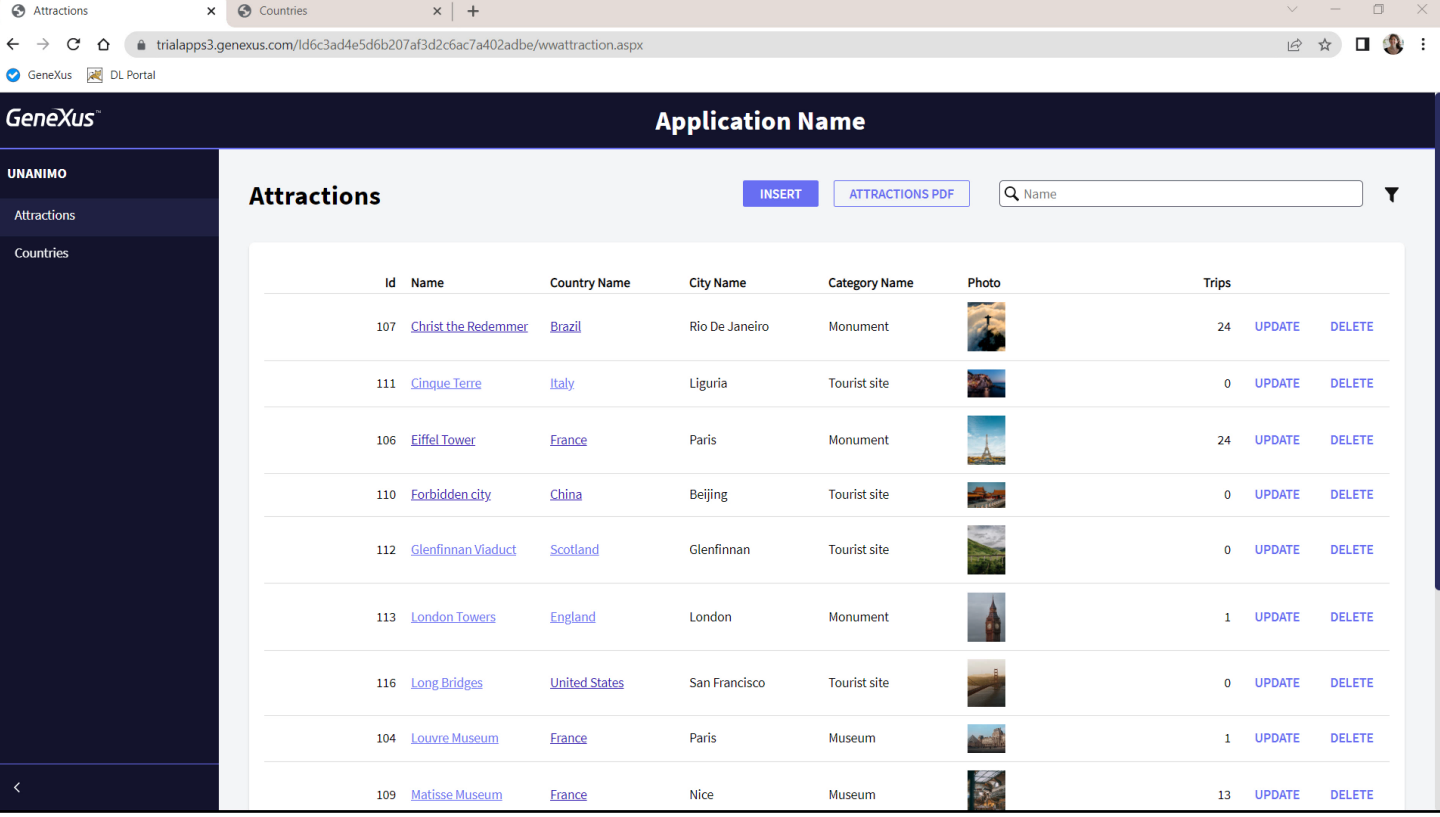


Since we only want to see an introduction here, we will not address the ways to directly import into GeneXus the design created by the designer in his/her preferred tool, but we say that this is possible: the designer designs the screens with a tool such as Sketch or Figma and the frontend developer imports it into GeneXus, which will automatically build the objects with all the design incorporated. There the frontend developer will have minimal work to do.

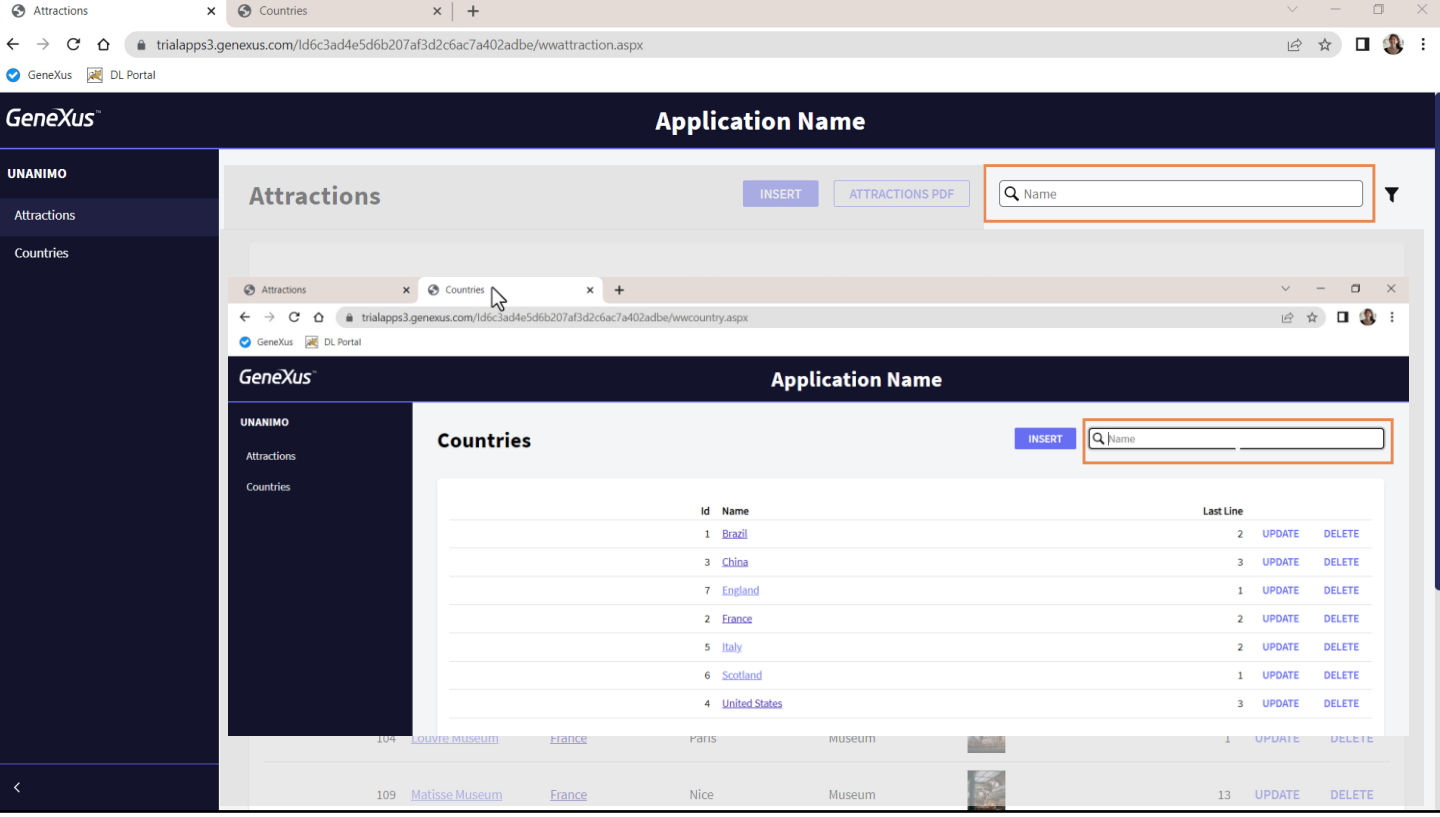
DESIGN SYSTEM

UNANIMO

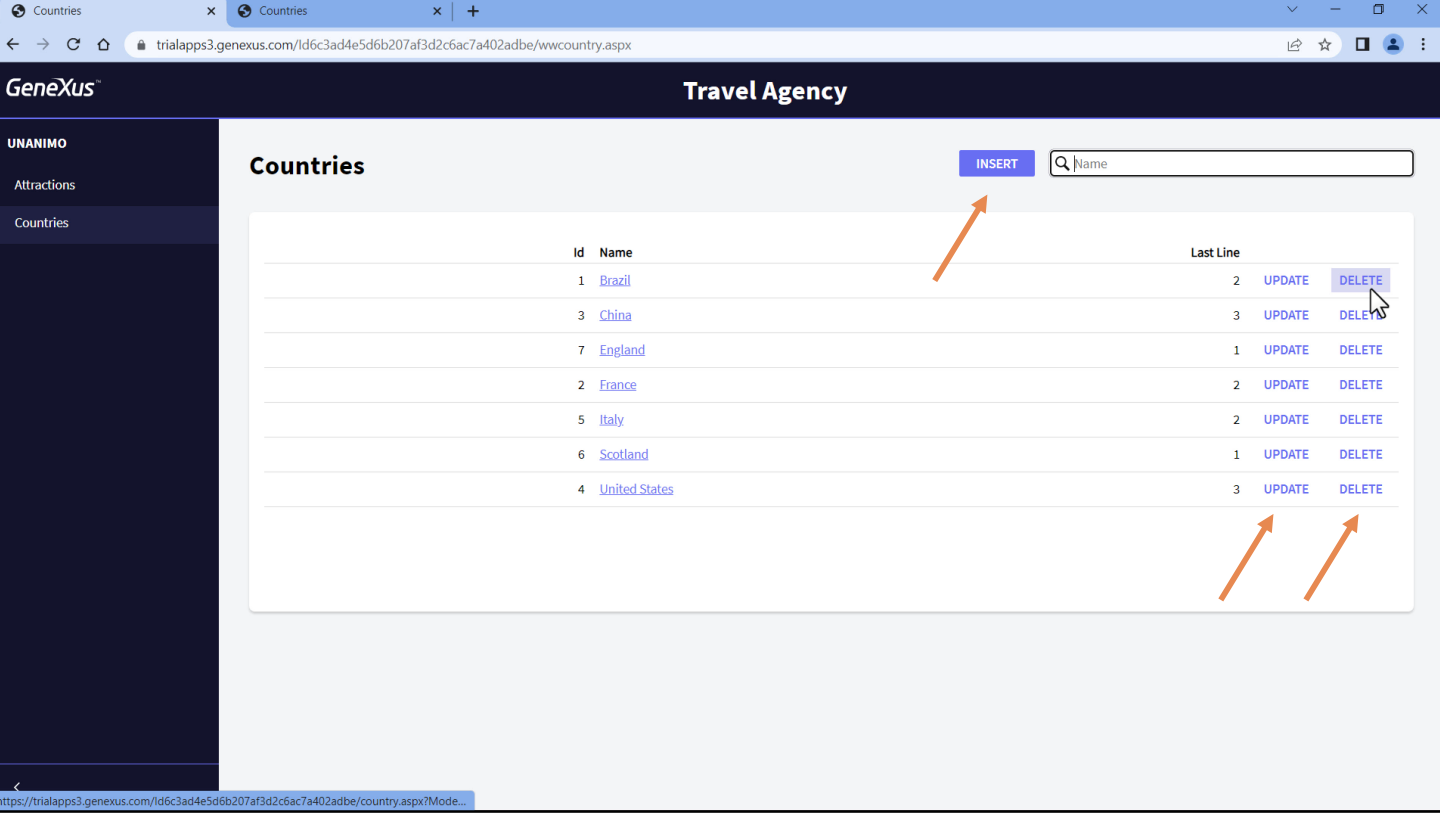
What we will do next is to see how GeneXus already provides a default Design System that we can customize: Unanimo.



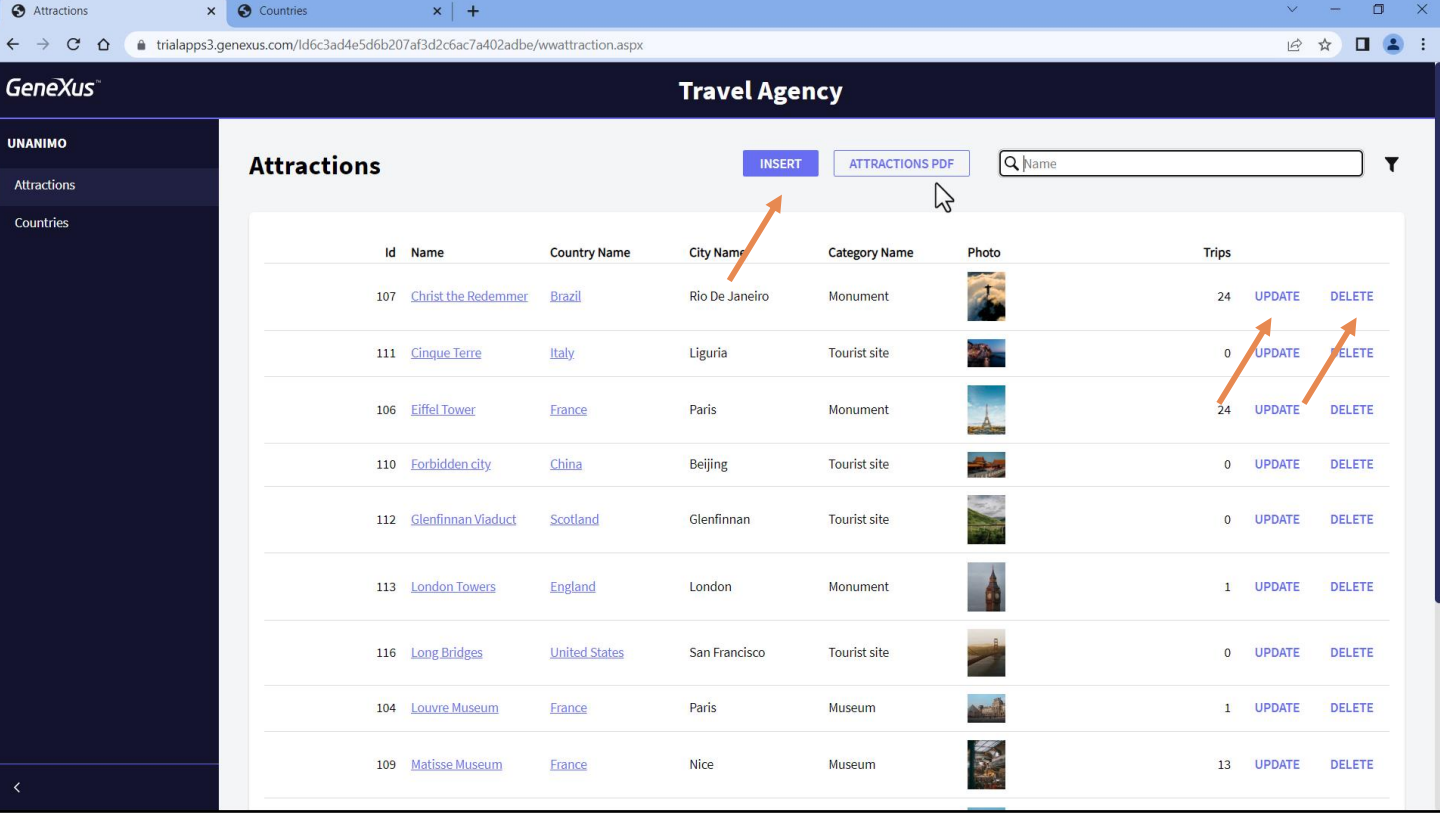
We have seen it in action all the time; for example, let's take a close look at the running Work With. In the central area of the page, there is information from the attractions Web panel created by the pattern, with uniform criteria in relation to the countries Work With.



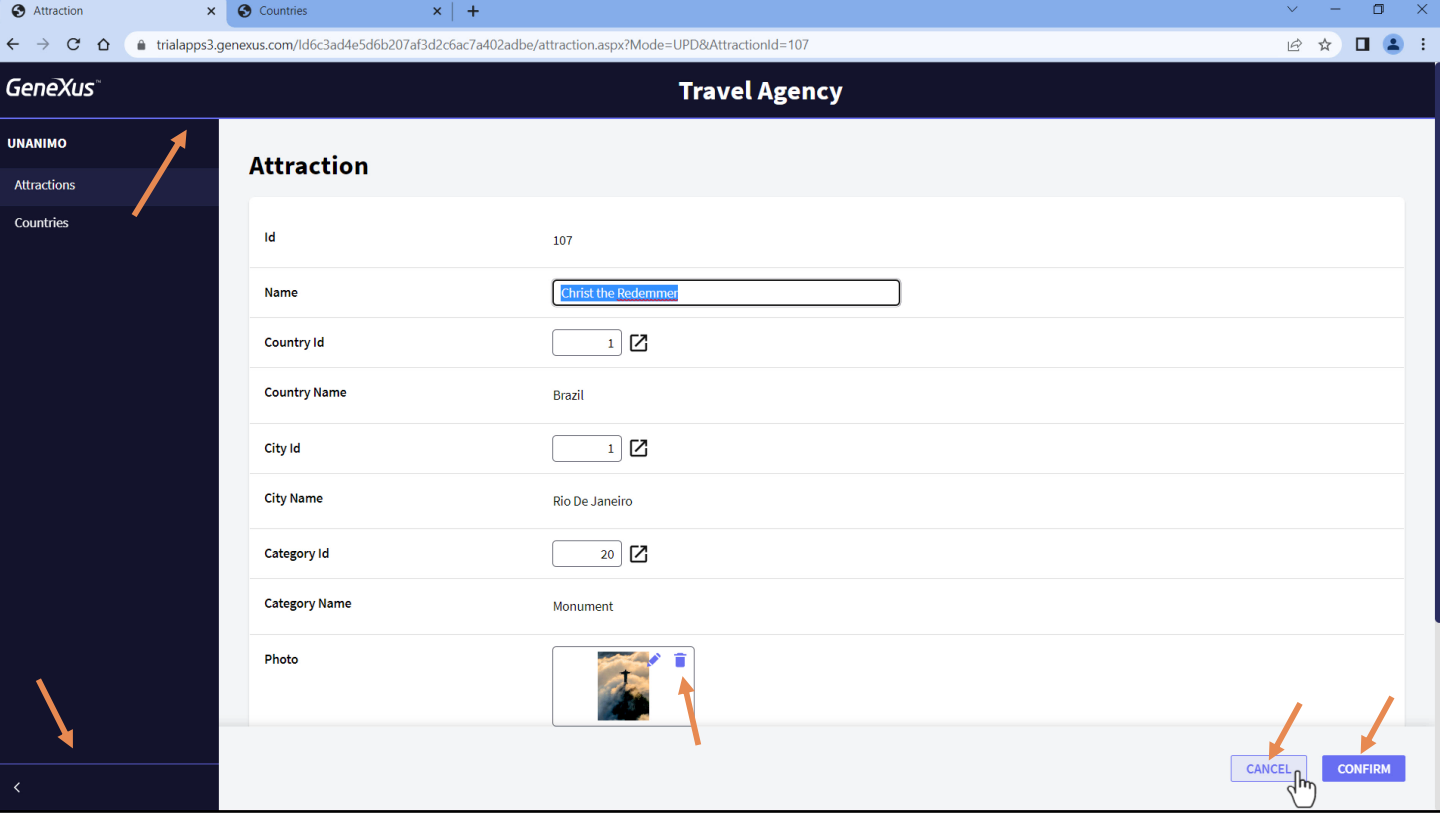
Note, for example, that in both cases a search field is available, where information is filtered by name: here for attractions... and here for countries. Both fields look identical, and they behave in the same way.



We can also notice another similarity: in both screens the same actions are offered to insert, update, and delete, with the same appearance.

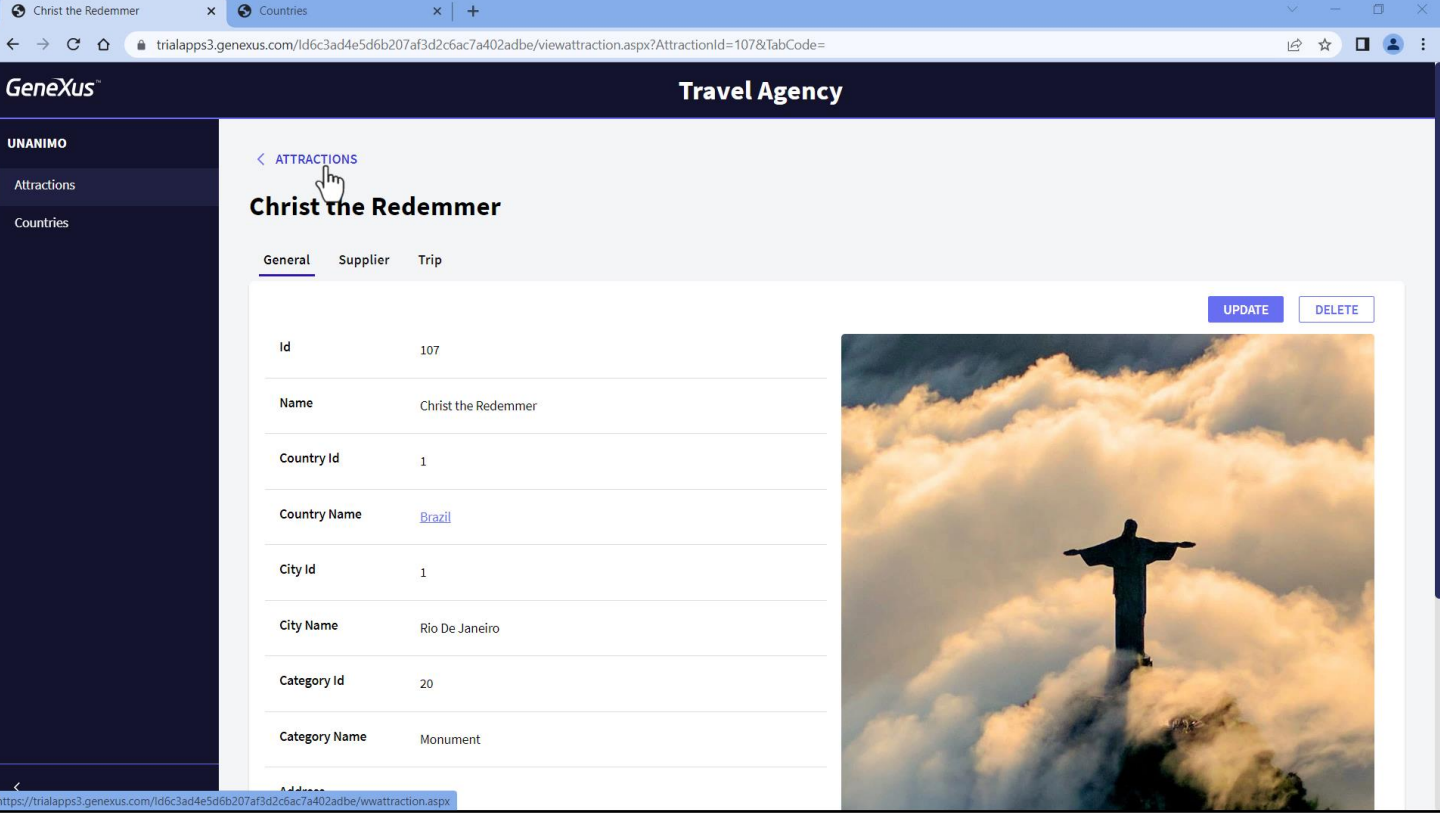


Here we see the same 3 actions, and we had also added another one, outside the grid, to invoke a PDF list. However, note that it is less highlighted than the Insert button. It is a more subtle button. That is a design decision.

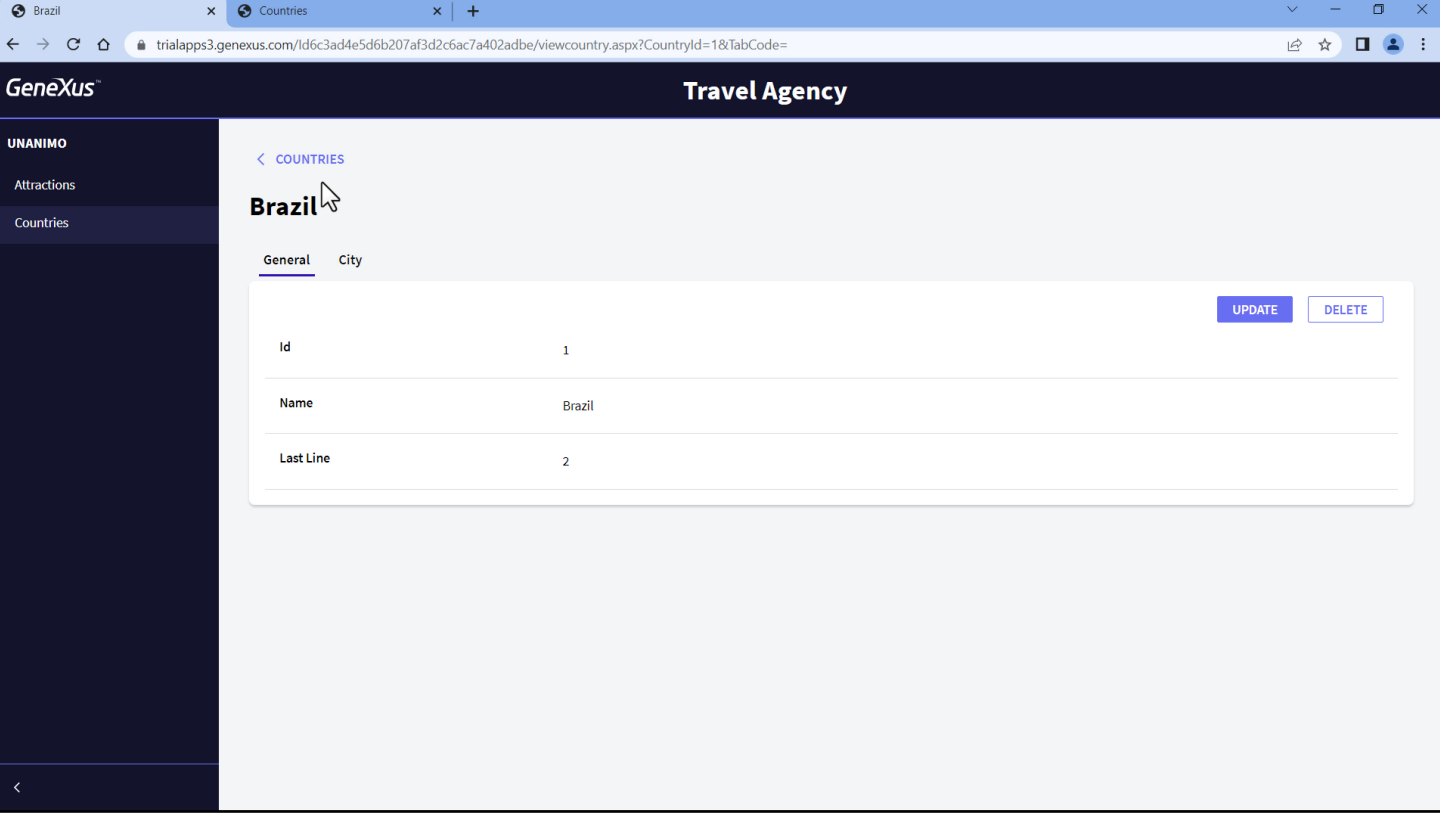


If we go to the transaction to update this attraction we can also see that the Confirm button is the relevant one, and the Cancel button is more subtle.

The actions are displayed with a predominant color, which is also used for other things.

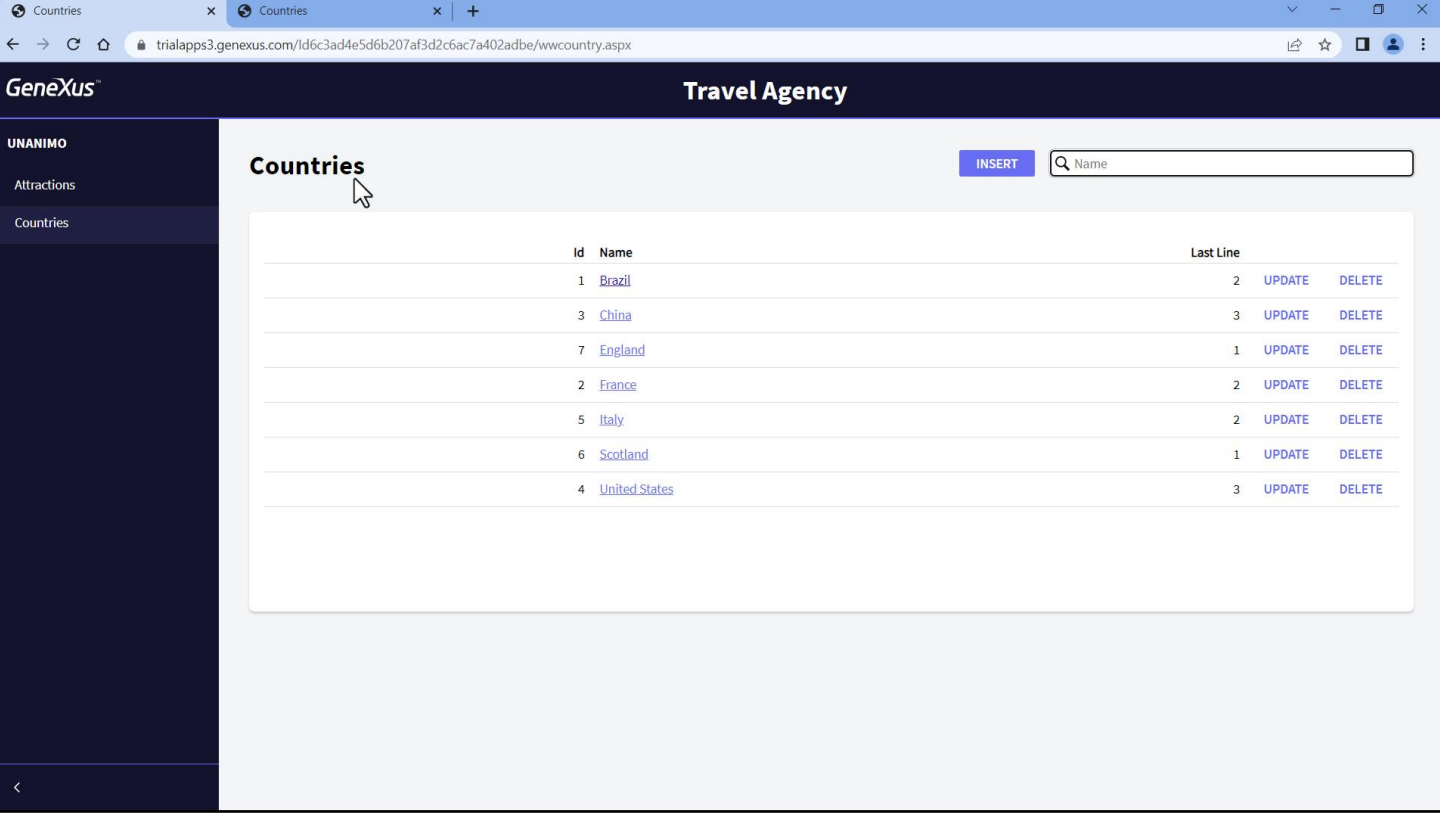


Now let's notice that when we navigate deeper and choose to see one of the attractions, it offers the option to go back to the parent.

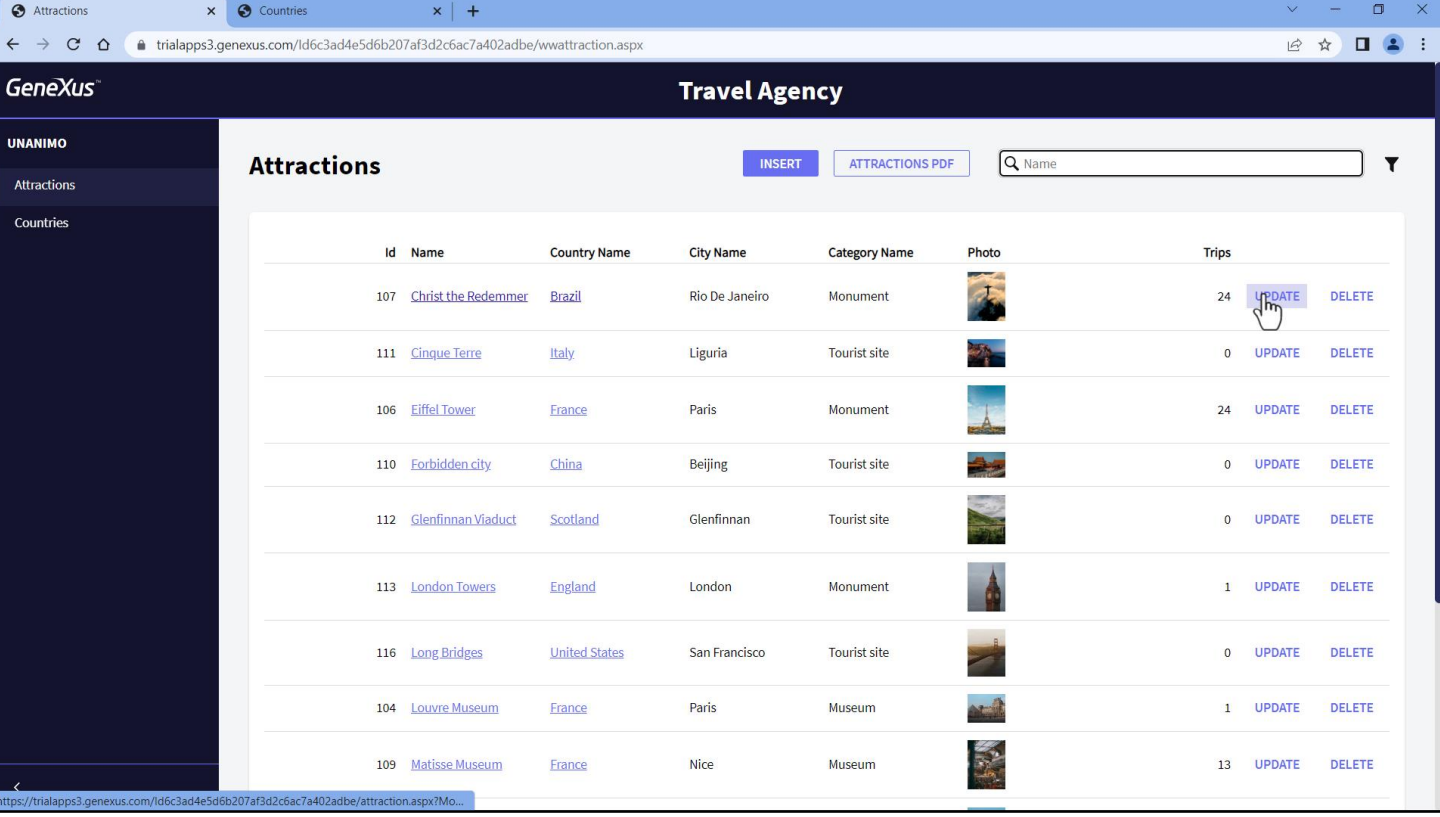


And this navigation is also a consistent pattern. Throughout the application we will expect this pattern of returning to the screen from which we arrived at the current one.

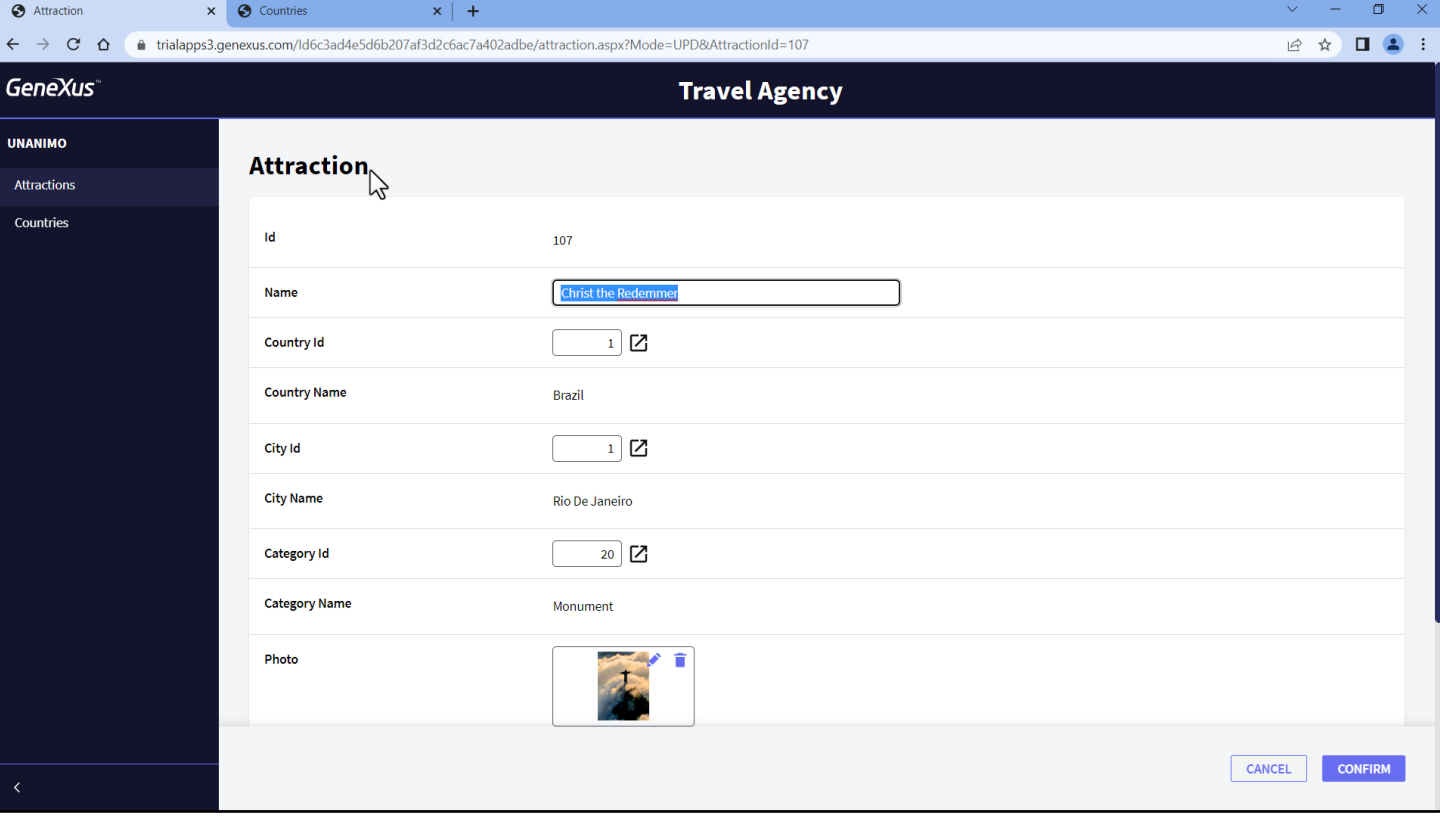
Now let's notice that all screens have text that clearly indicates where we are. Here we are in Brazil's information screen. Let's take a look at its appearance.



Here we are in Countries, with the same appearance.

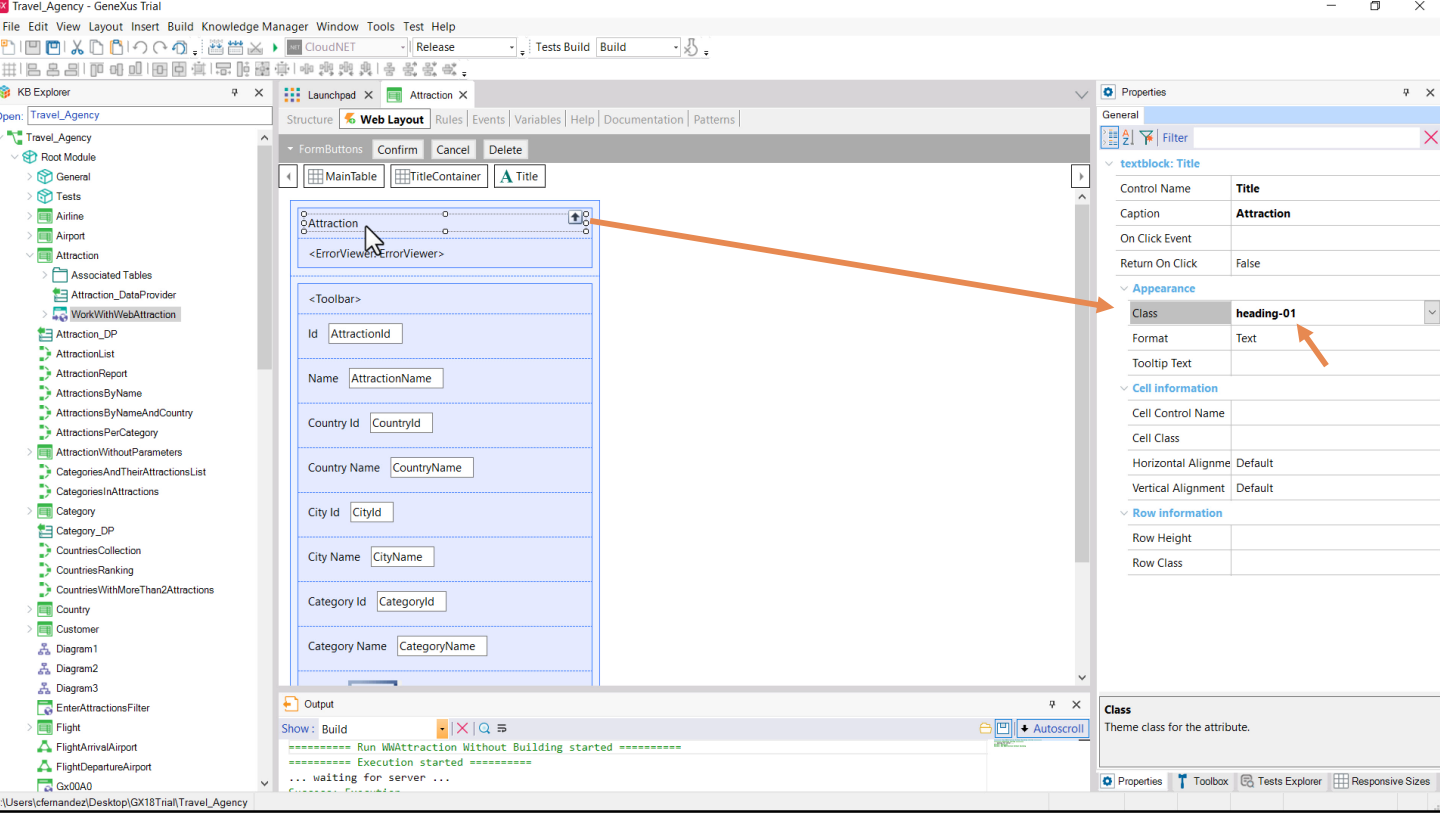


Here in Attractions. And if we choose to update one...

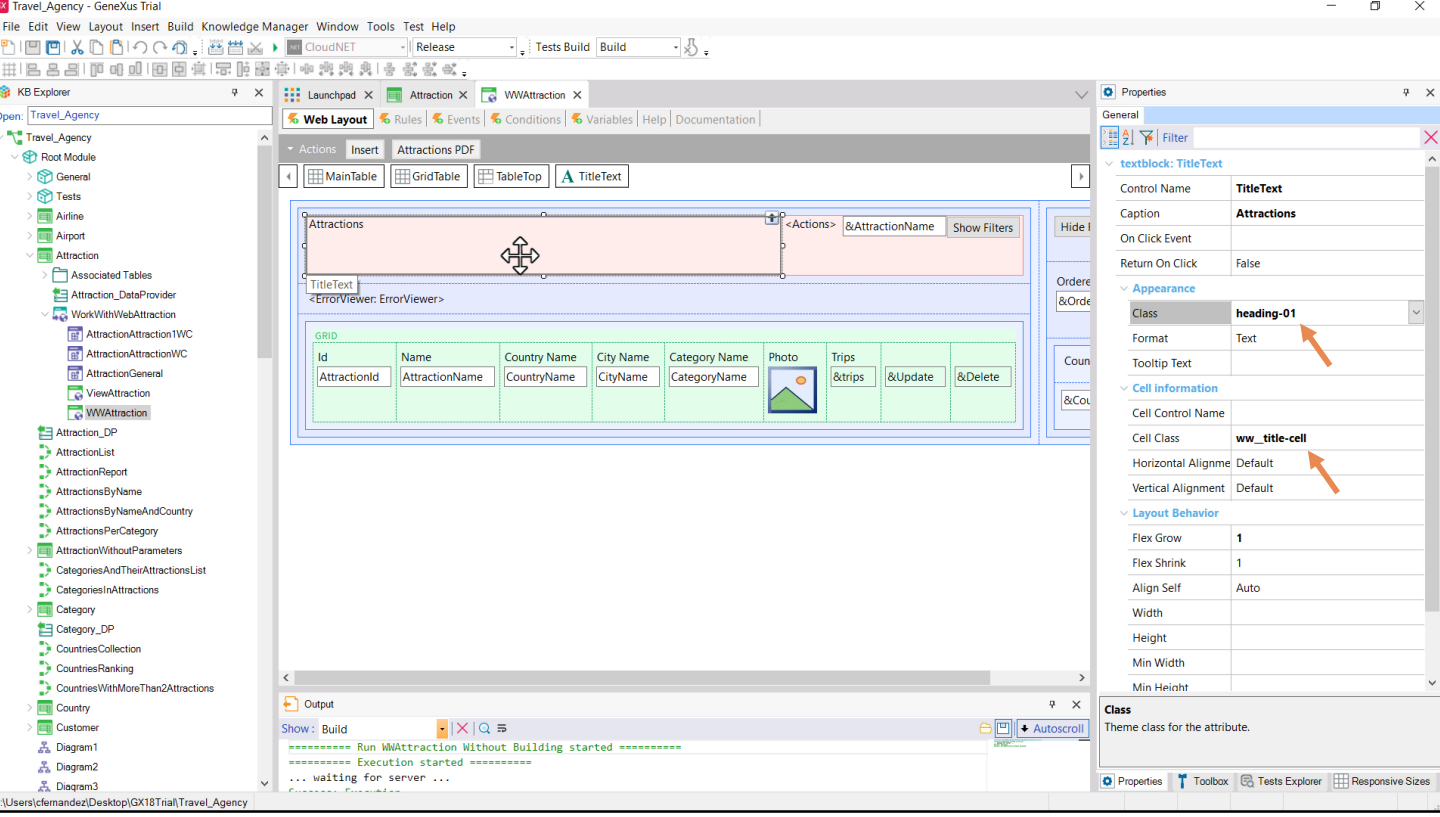


...here we are in the Attraction transaction.

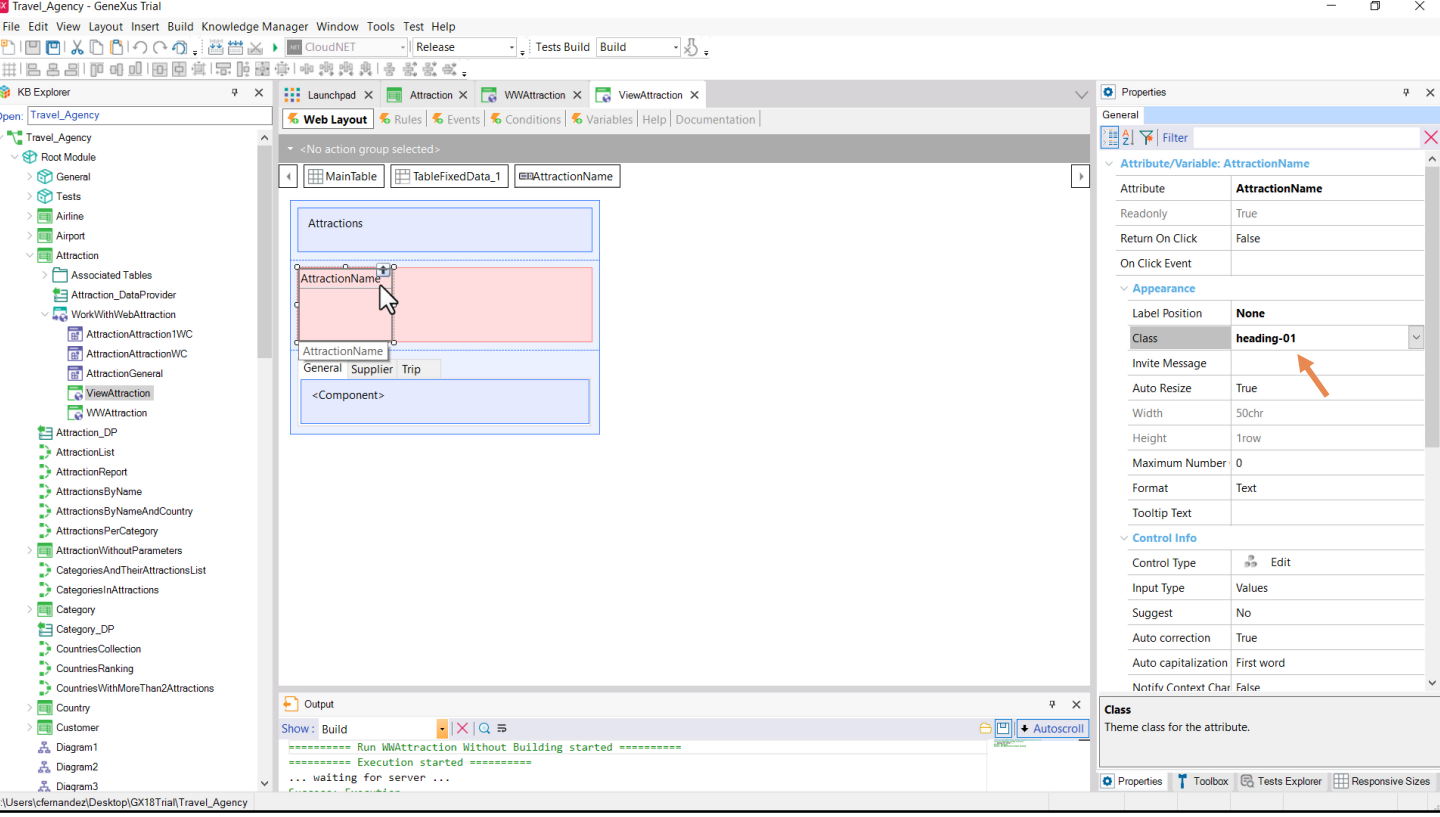
The question that comes up: where has the appearance of these texts been defined (that is, their font, font size, color)? And so, what if we want to change it, for example, to red? How do we go about it?



Let's edit the transaction layout, and let's see the text block that implements the text that we see at runtime. If we look at its properties this one appears: Class. The class with this name has been associated with it. It is here, in this class, where most of the design properties will be defined.

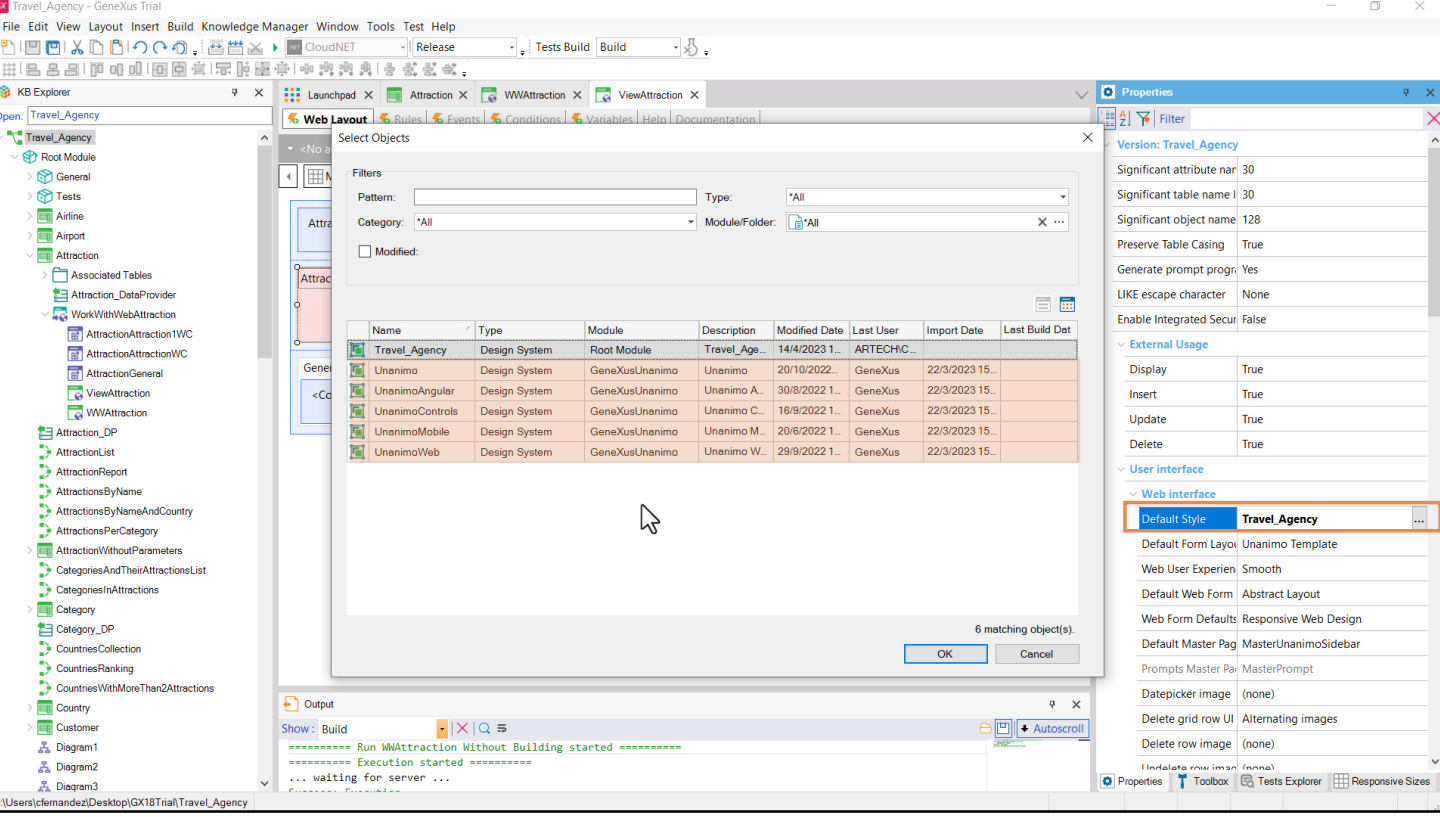


If we go to see the text in the attractions Work With, it is also a text block control, which also has the same class assigned to it. It also has a class assigned for the cell where the control is located in the table. For now, we will not pay attention to it.



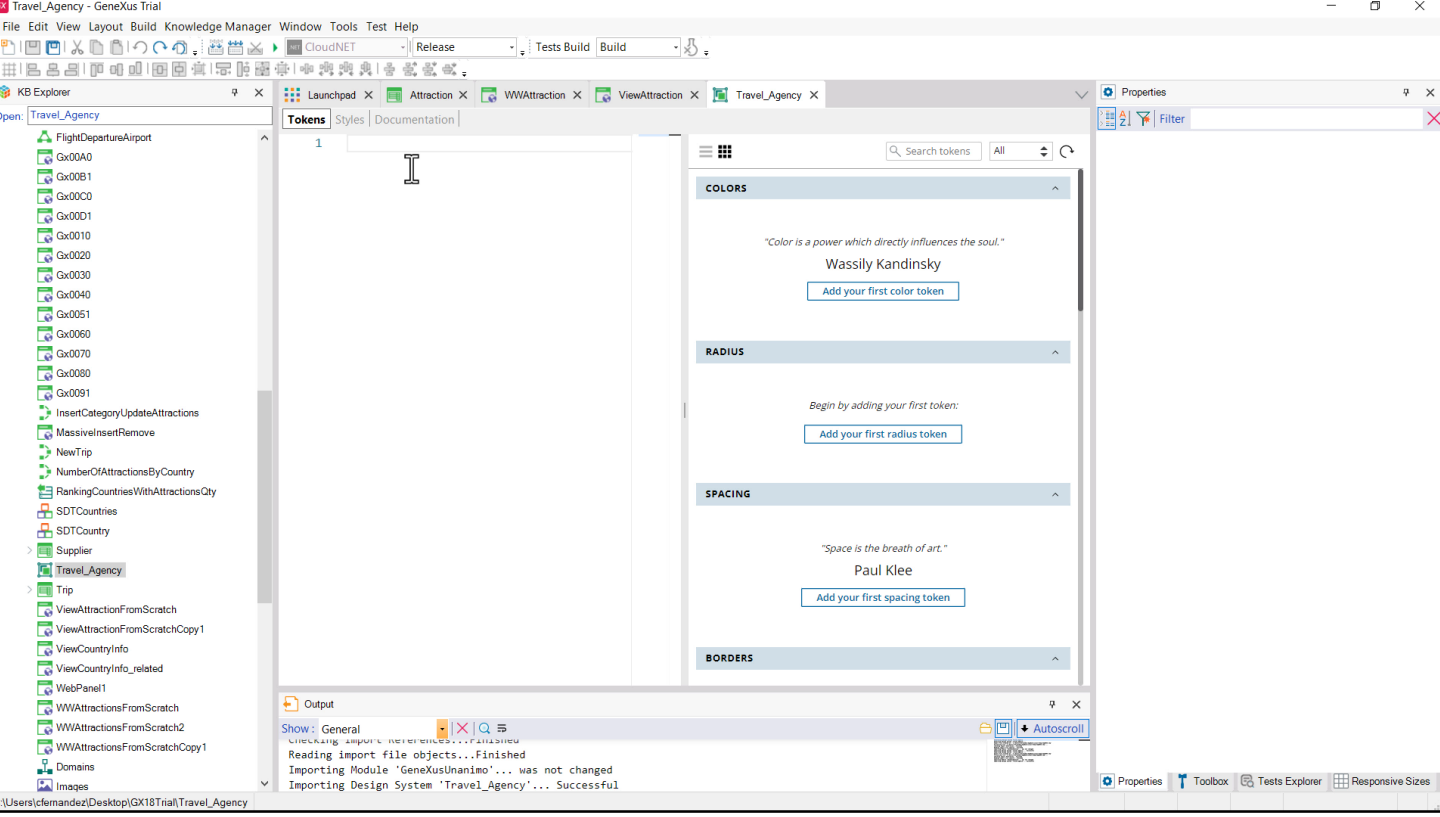
If we now see the view of an attraction, it is the AttractionName attribute control that marks the page where we are, and it is not a coincidence that it has this class associated with it.

The question now is this: where are the style definitions of that class located?

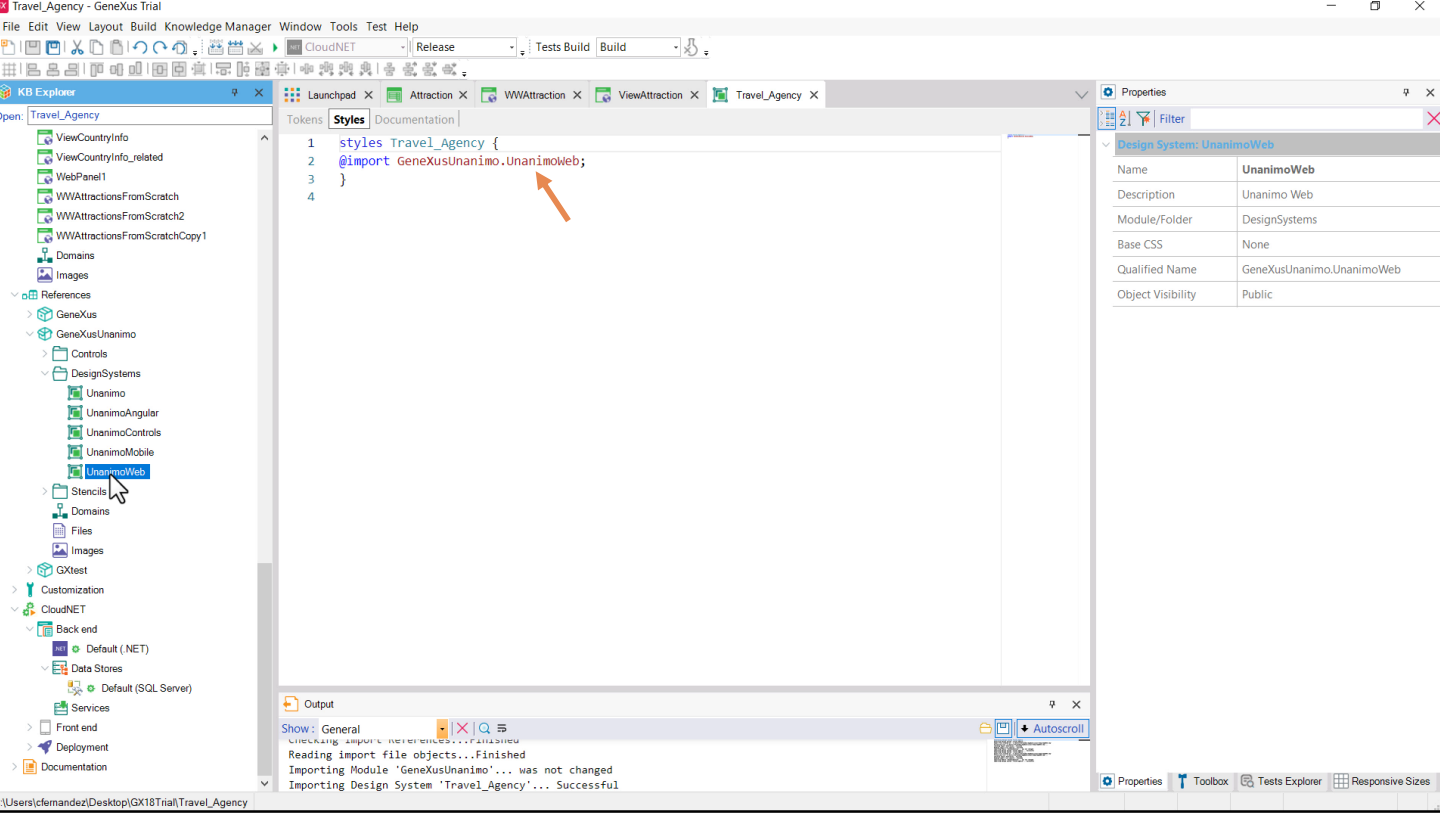


If we go to see the properties of the version of the KB we are working with, among them we see one, Default Style. It has an associated object, which happens to have the same name as the KB. When we open the window to make a selection, we see that the types of objects of the KB that could be specified here are of the Design System type. In our KB we have one that is created by default in the root module, and that we can modify: it is the default one, precisely. In addition, there are these other ones in the GeneXusUnanimo module that come by default in every KB we create, and will be read-only objects: that is to say, they can be used but not modified.

Let's look for our default object, the Design System that will govern most of the design of the controls and layouts of our KB. We open it.

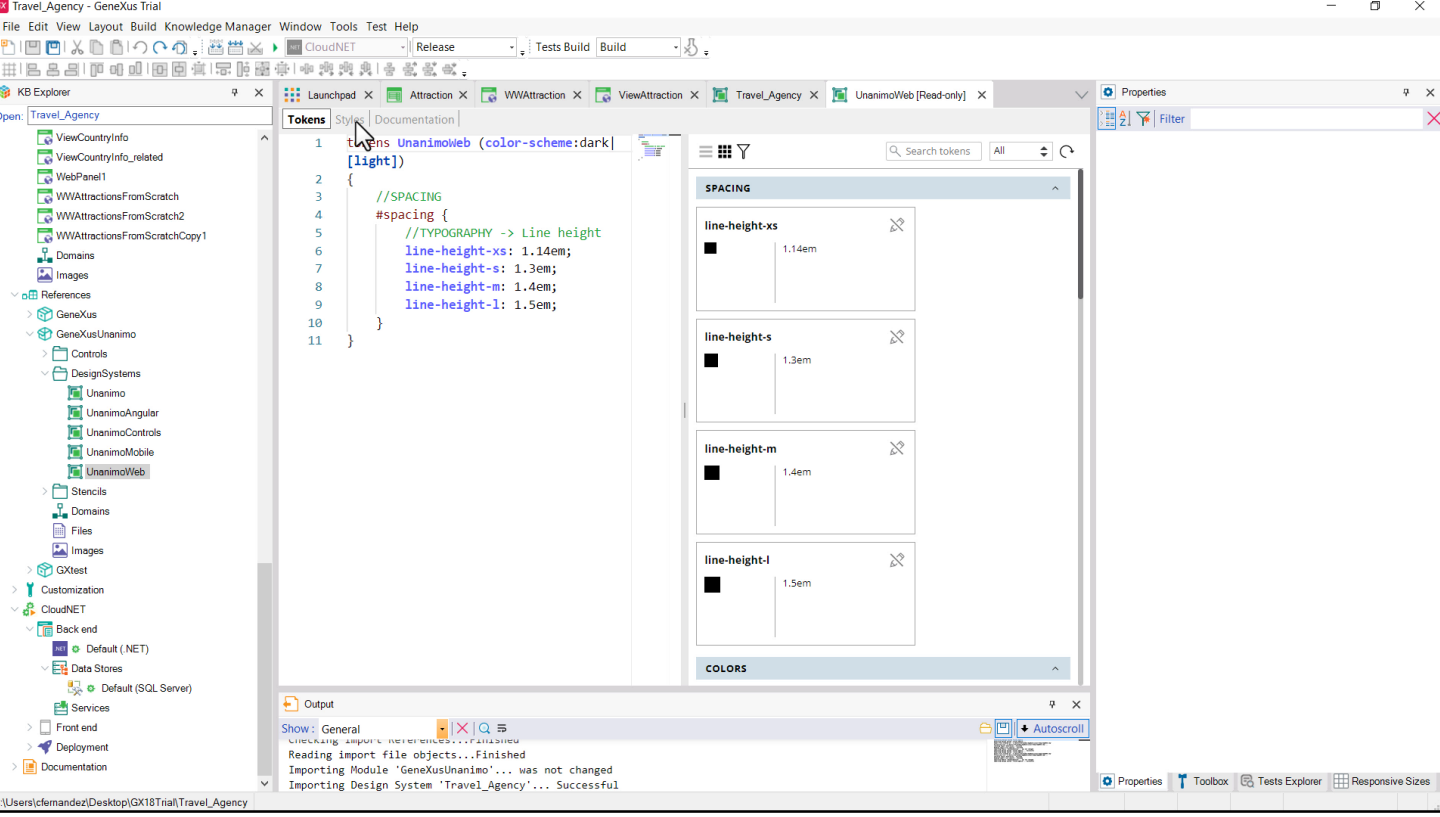


We have this Tokens tab, and later we will see what it is used for...

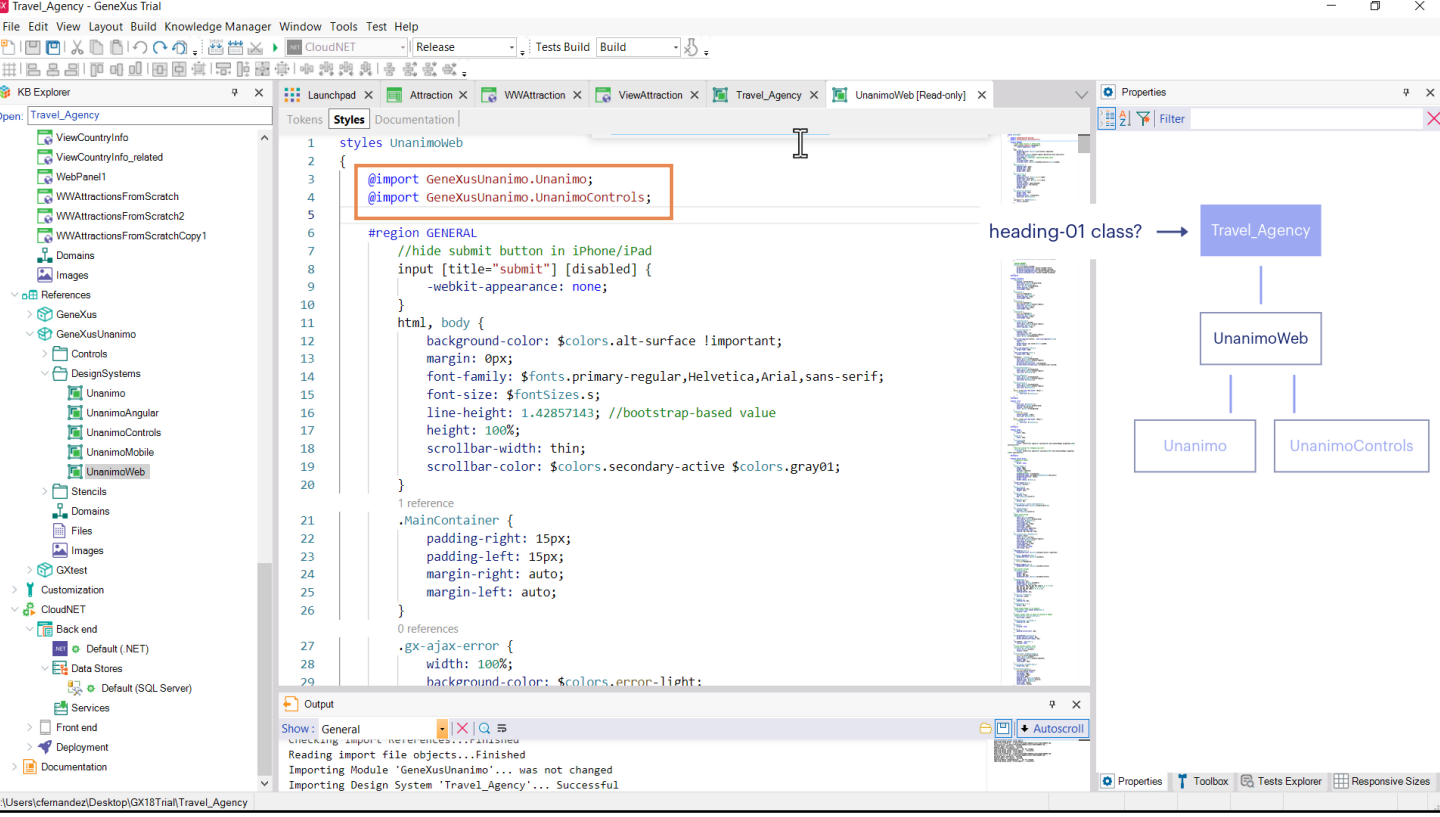


...and this other one, Styles, which is the most relevant one. It seems to be empty, but it is not. Note that it is asking to import everything that comes from this other Design System object, which is one of the Design System objects that come with every KB and that implement the Unanimo Design System.

It can be found in the GeneXusUnanimo module, under the References node.

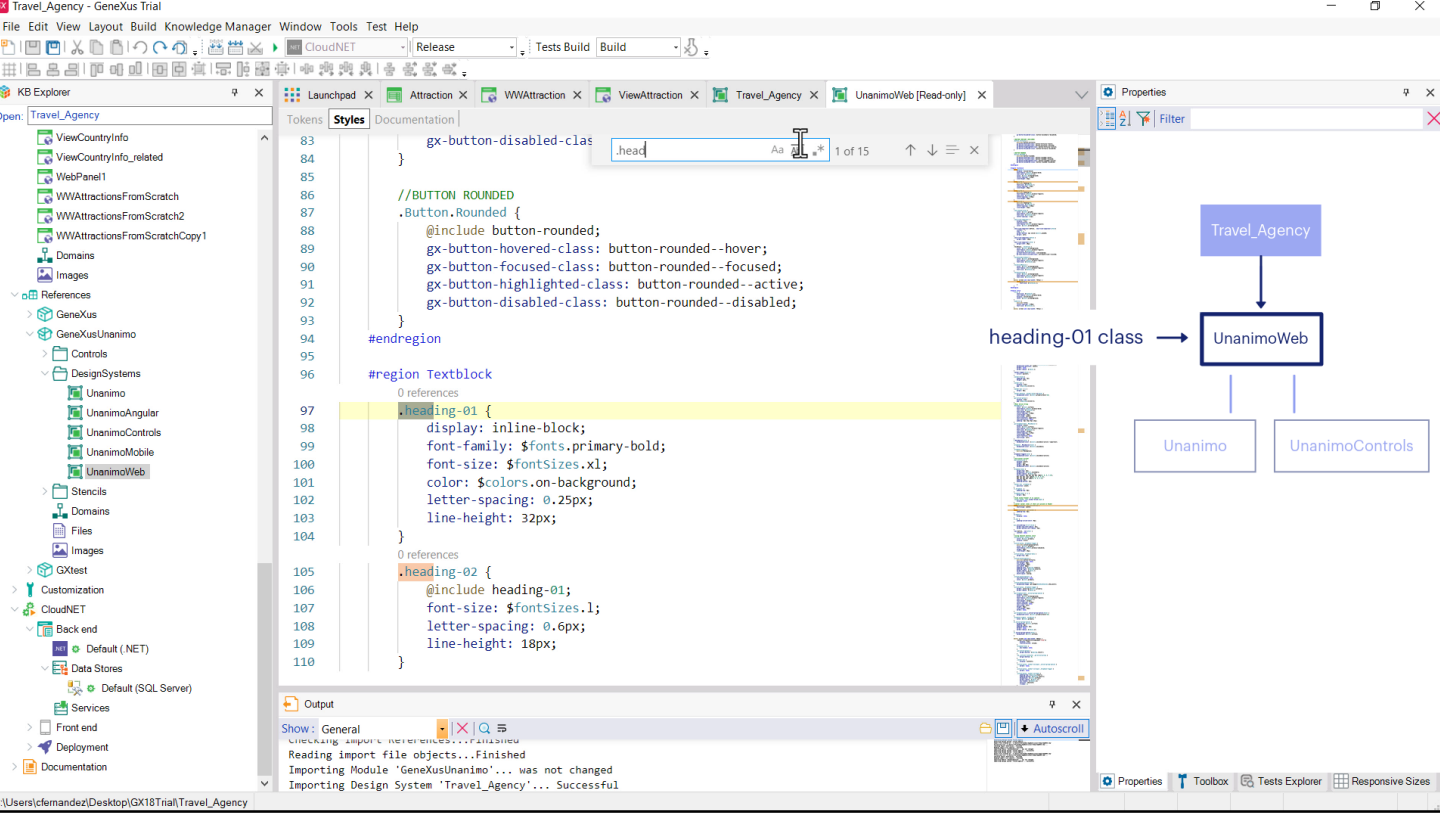


Here the Tokens section is no longer empty.

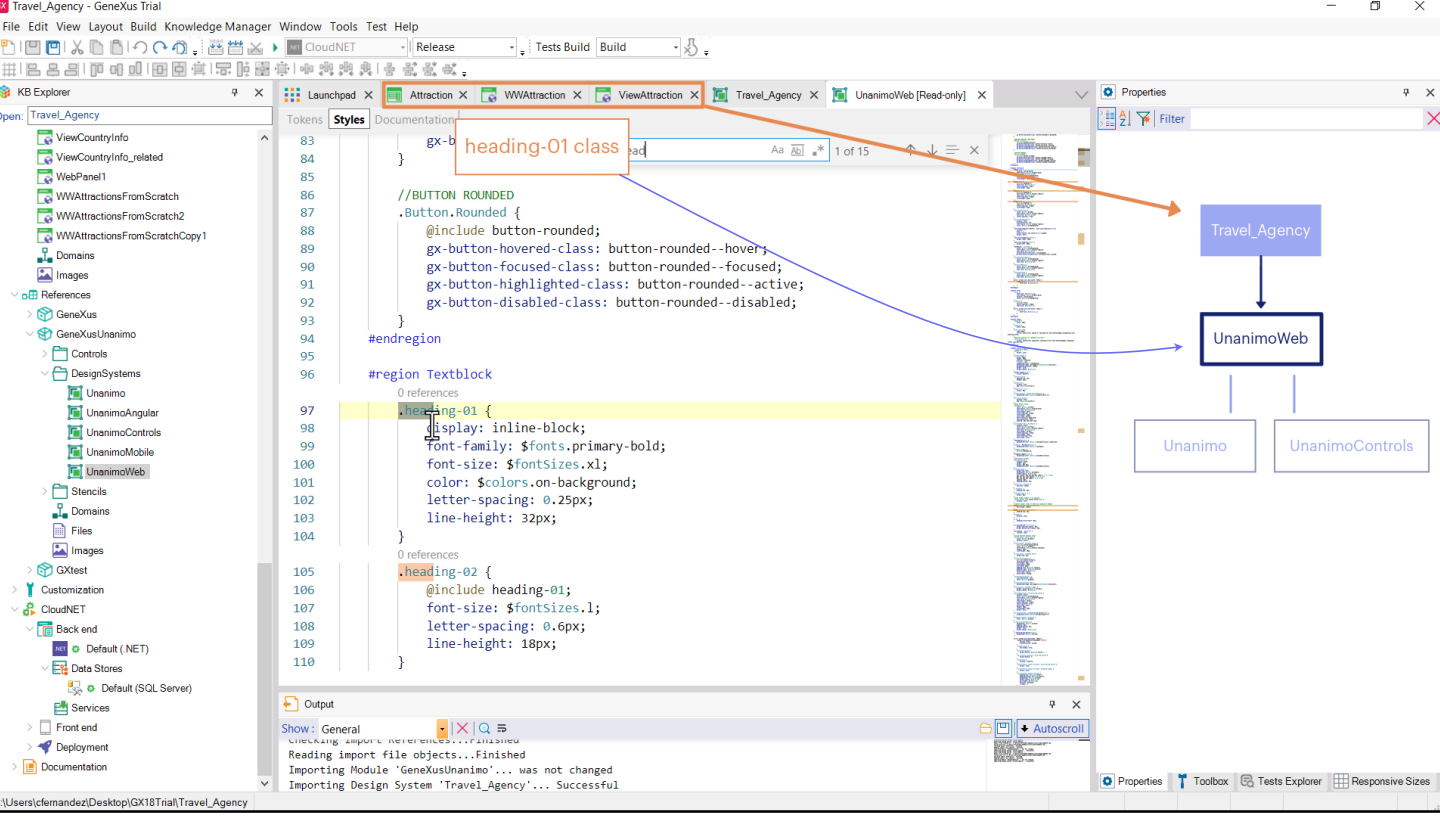


And the styles section is much less so. This is where the classes that are later associated with the controls are defined. We see that this DSO (Design System Object) is not starting from scratch either. It is importing all the definitions of these two. Therefore, we have the DSO of our KB built from this DSO tree.

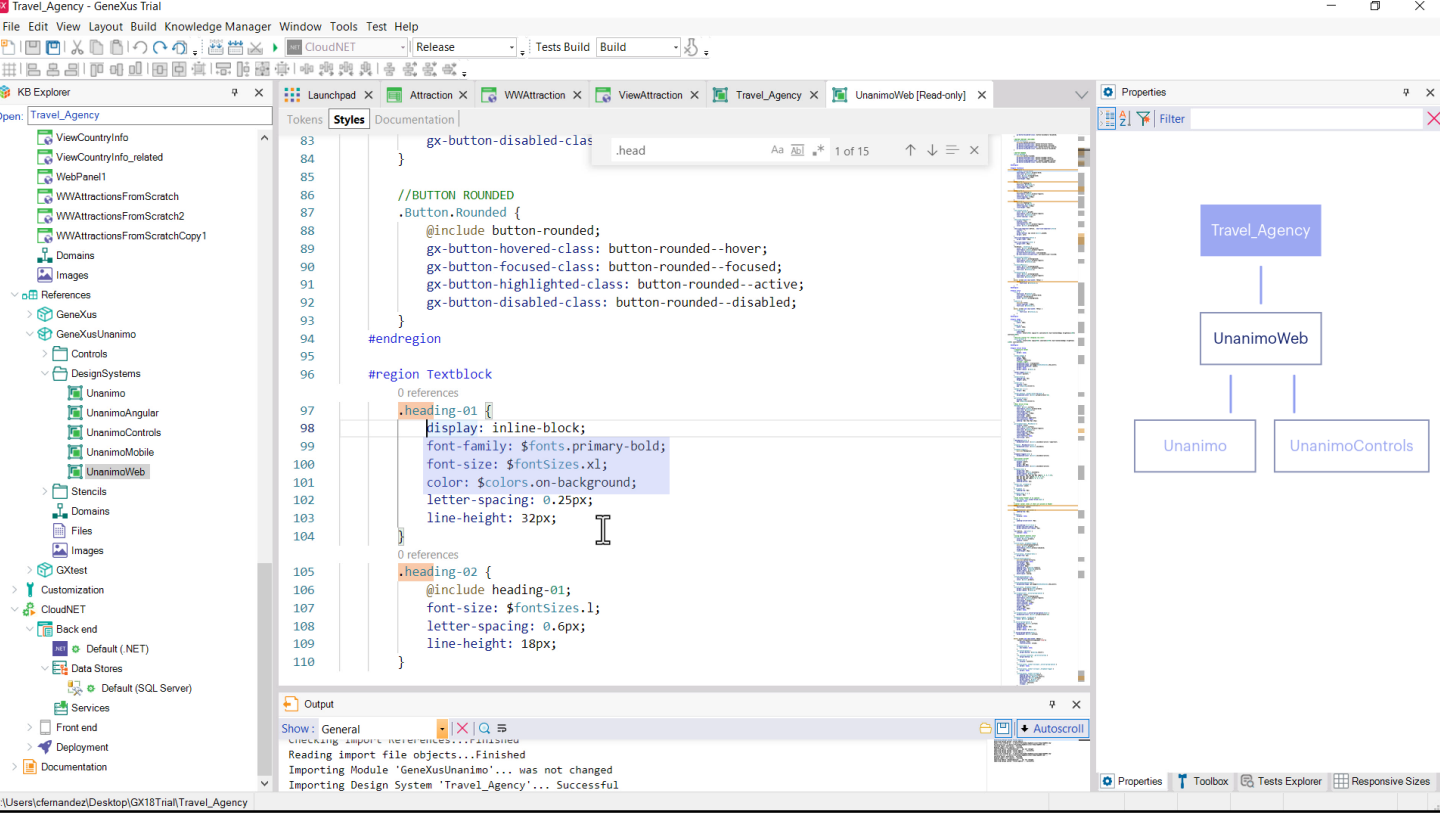
We are looking for the characteristics of the heading-01 class....



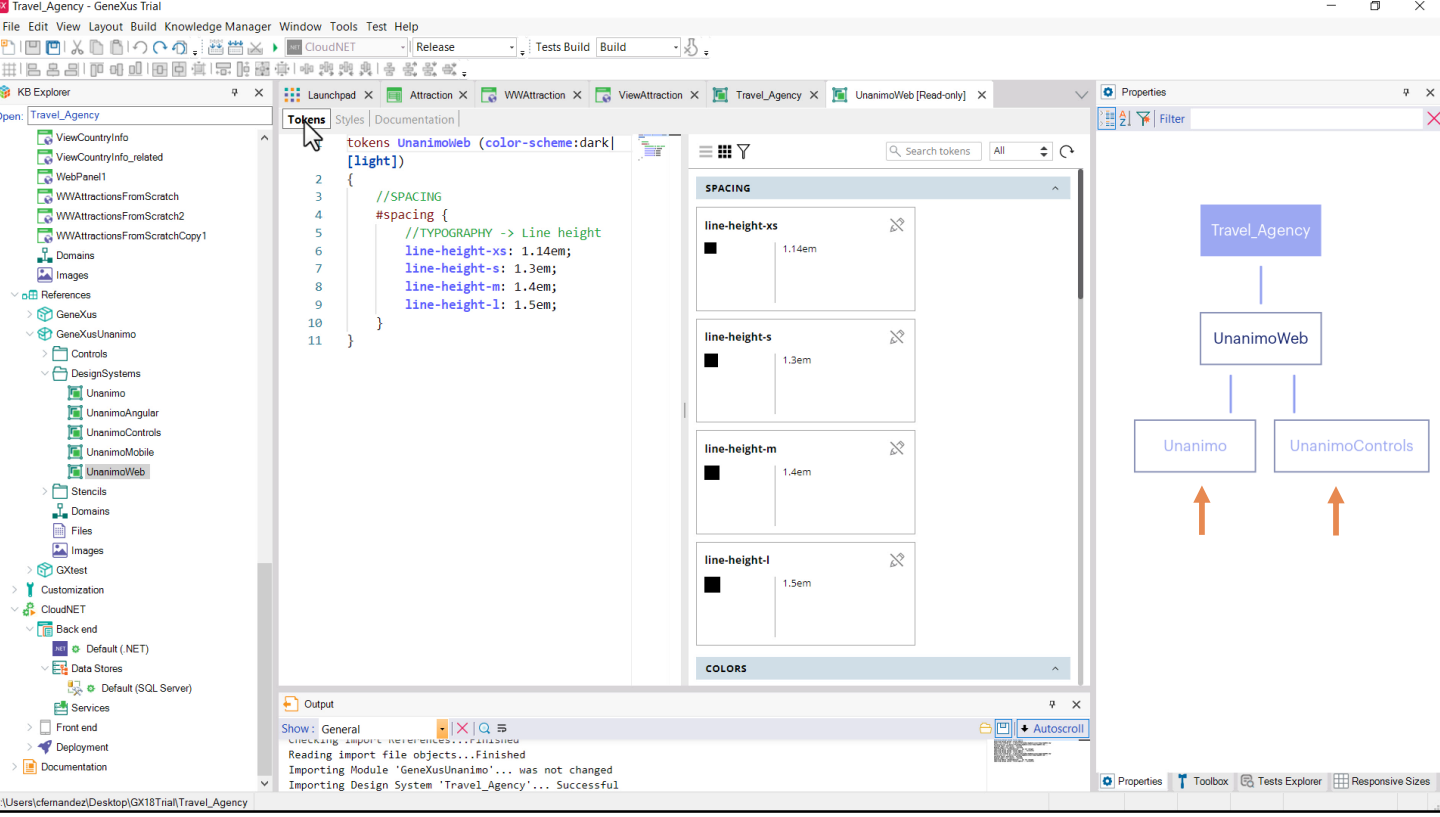
...so we can search for it by pressing Ctrl + F. Classes are selected with a period. Here we find it.



Before going on, let's review what we were doing: the textblock or attribute controls of web panels and transactions whose function was to show a title on each screen to indicate where we are had this associated class. These web panels and transactions have a Design System object defined by default. It is that of the KB, so they will look for the class properties there. But for now this object does not have its own classes. It only has the ones imported from this DSO. It is as if they were copied. That is why we look for them here.

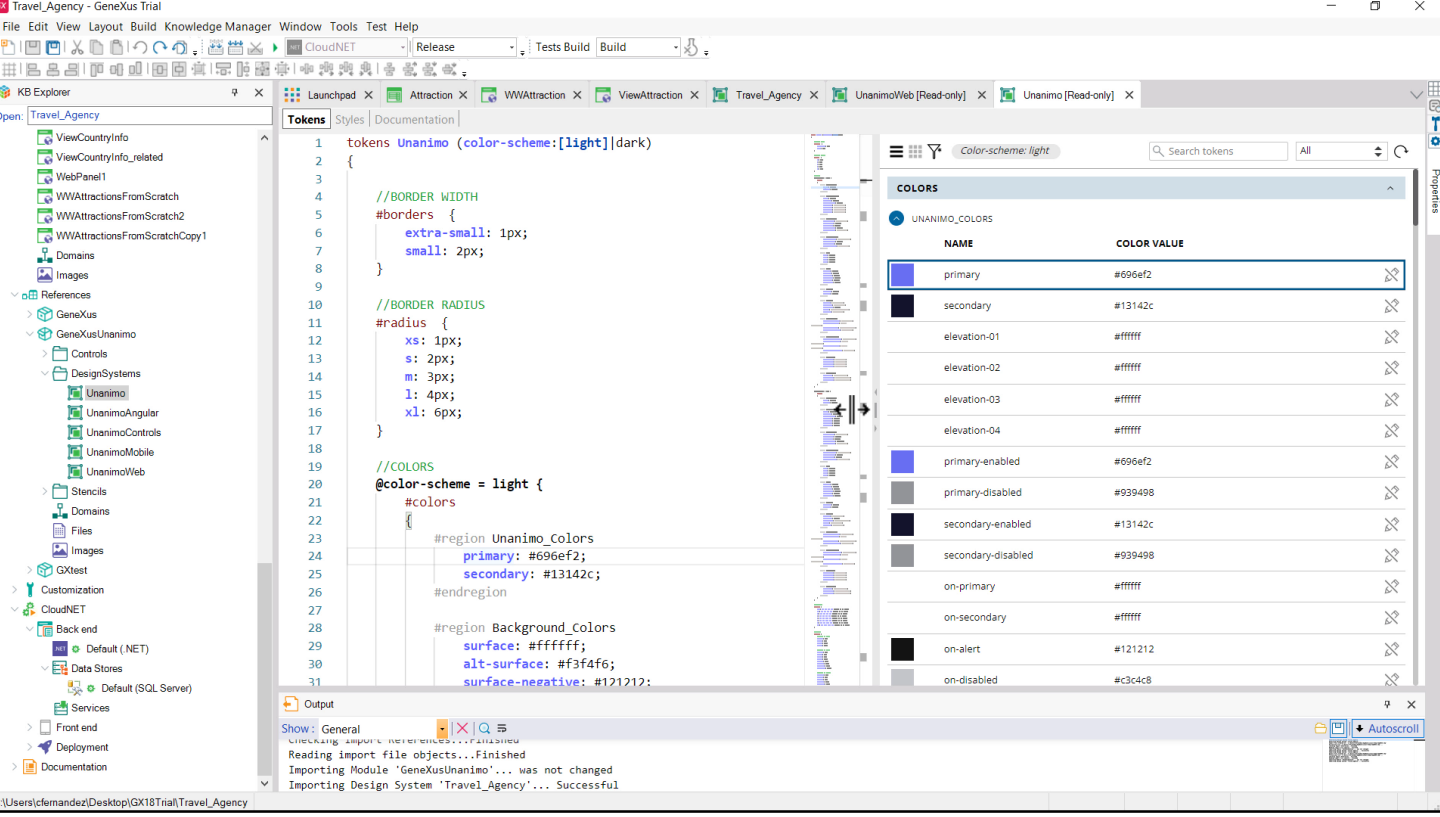


Let's look at the properties it has configured. They include the font family, font size, and color. What do the dollar signs before fonts, fontSizes, and colors mean? They are references to **tokens**. And what are tokens? Names given to values. In programming, we call them constants. For example, let's analyze the color property. We are indicating that the color will be taken from the value of the color constant named "on-background." Where should we look for it? In the Tokens tab.

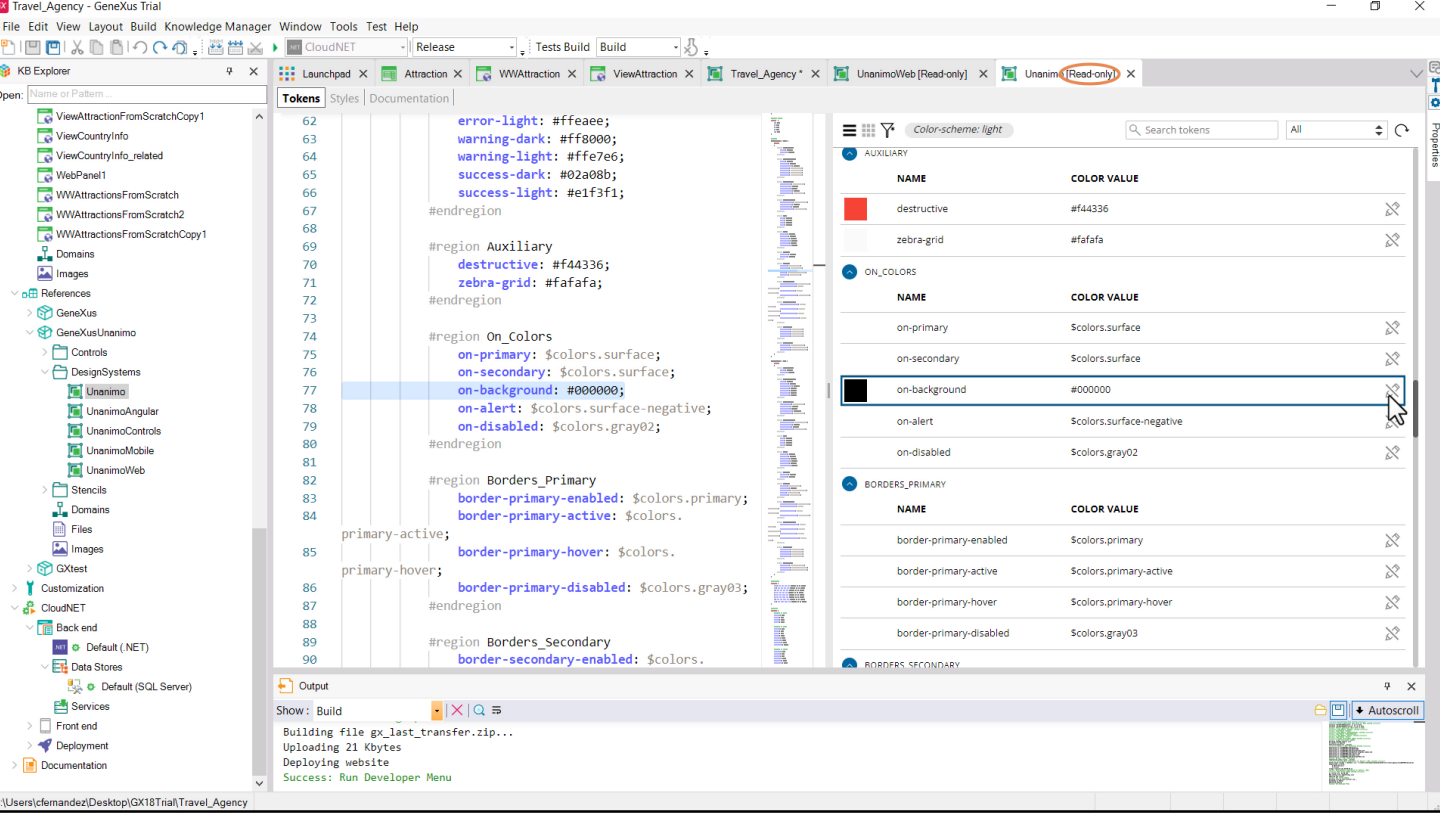


But it is not here! So?

Oh, that's right, this DSO is importing two others. Among those that matter are the tokens. Let's look for our color token in one of those two DSOs, because it will have to be in one of them.



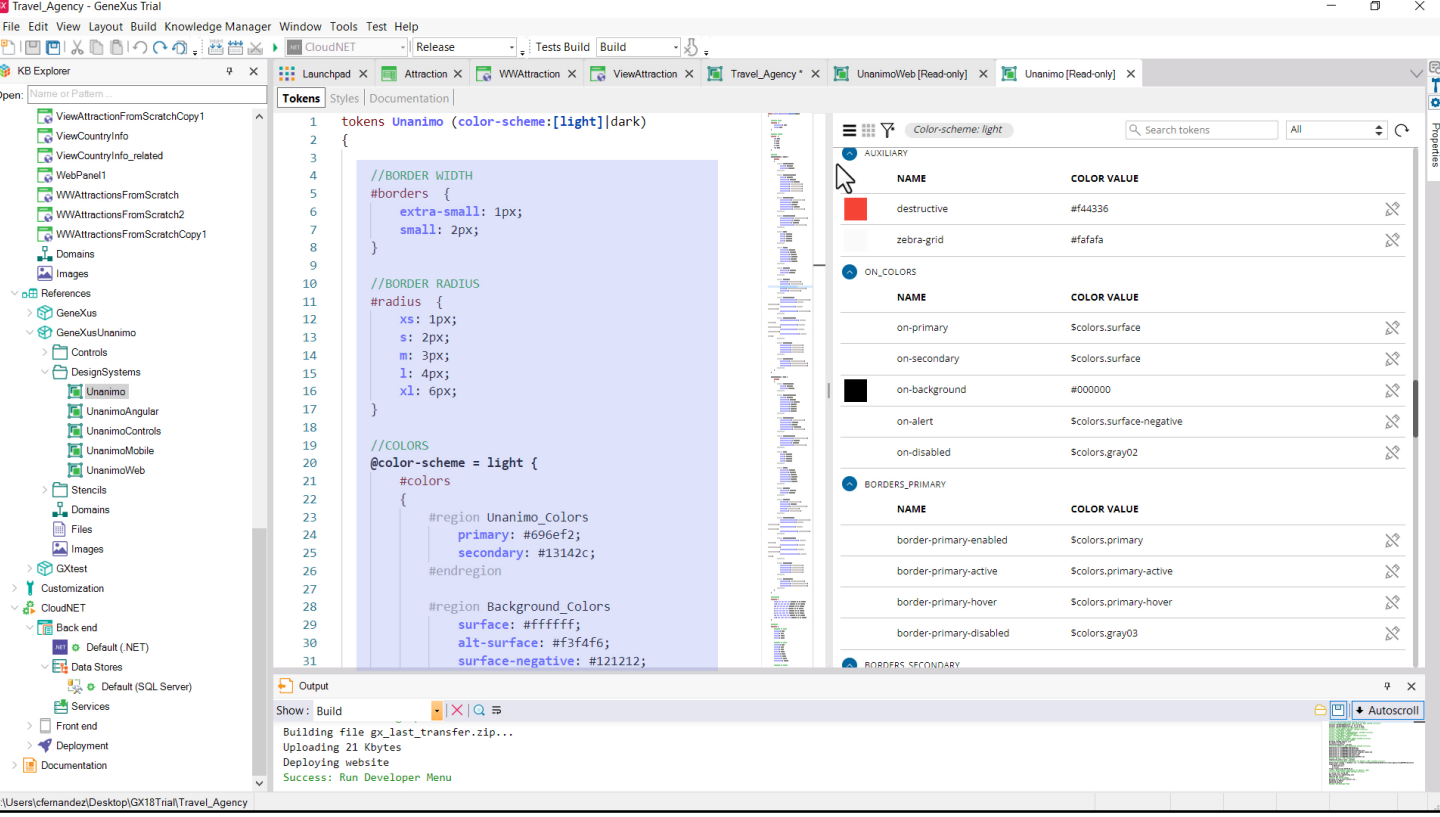
Here the Tokens tab appears very crowded. It is possible that here we will find what we are looking for. Let's see that the editor is double. And so, if we select this color token, the one named “primary,” in the graphical editor on the right we see a color that on the left is defined with a hexadecimal value. This color looks familiar, doesn’t it? It is the one we have seen repeatedly in the actions and buttons. It has not been named “primary” just for the sake of it.



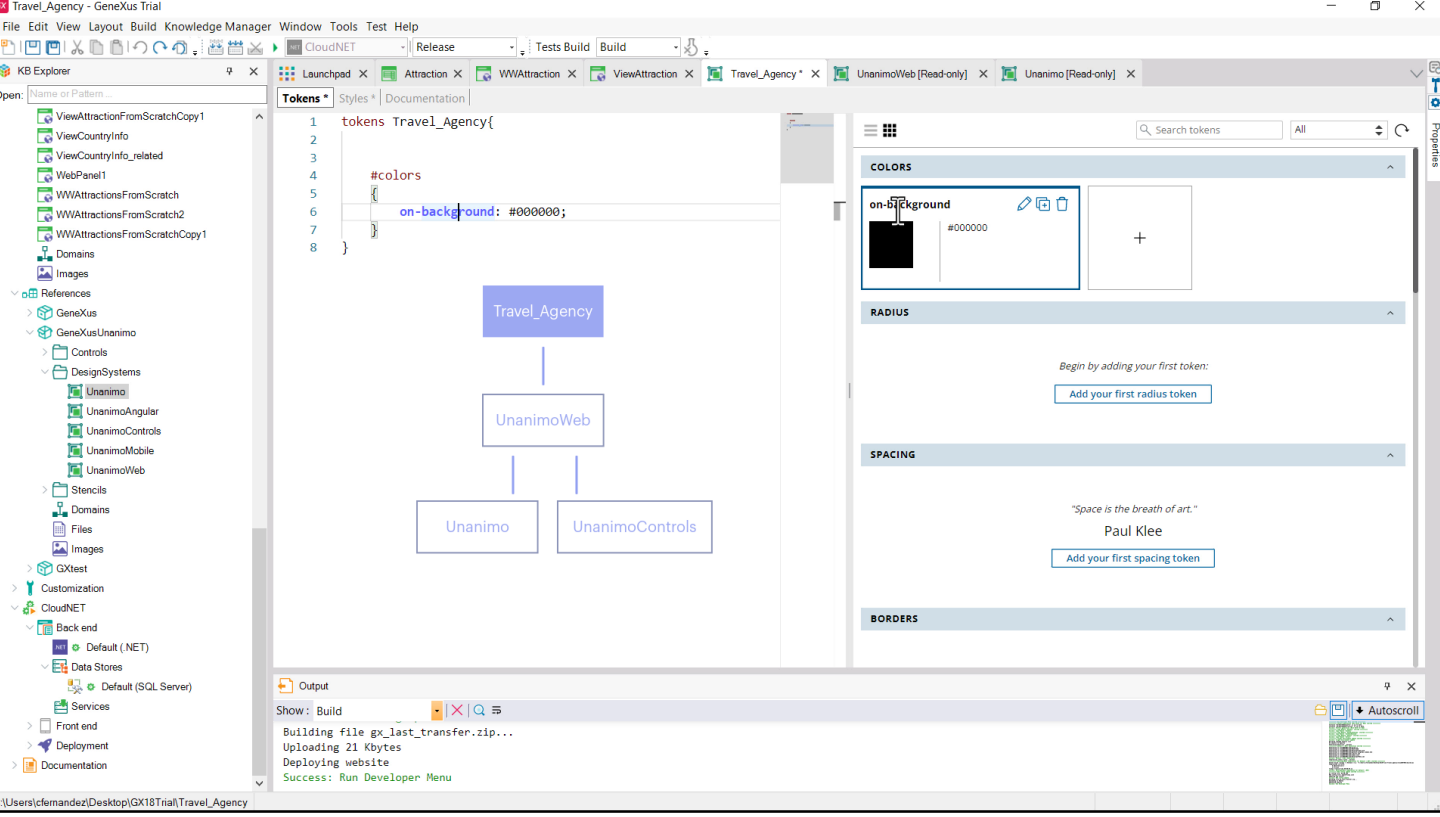
The one we are looking for is the color token named “on-background.” If the token was given a semantically clear name, we can assume it is the color that will be used to contrast against the background. Here we have it.

What happens if we change the value of the token, so that it goes from black to red? Supposedly, with that we will be able to change the color of the texts that we were interested in, those of the heading_01 class. Let’s try it.

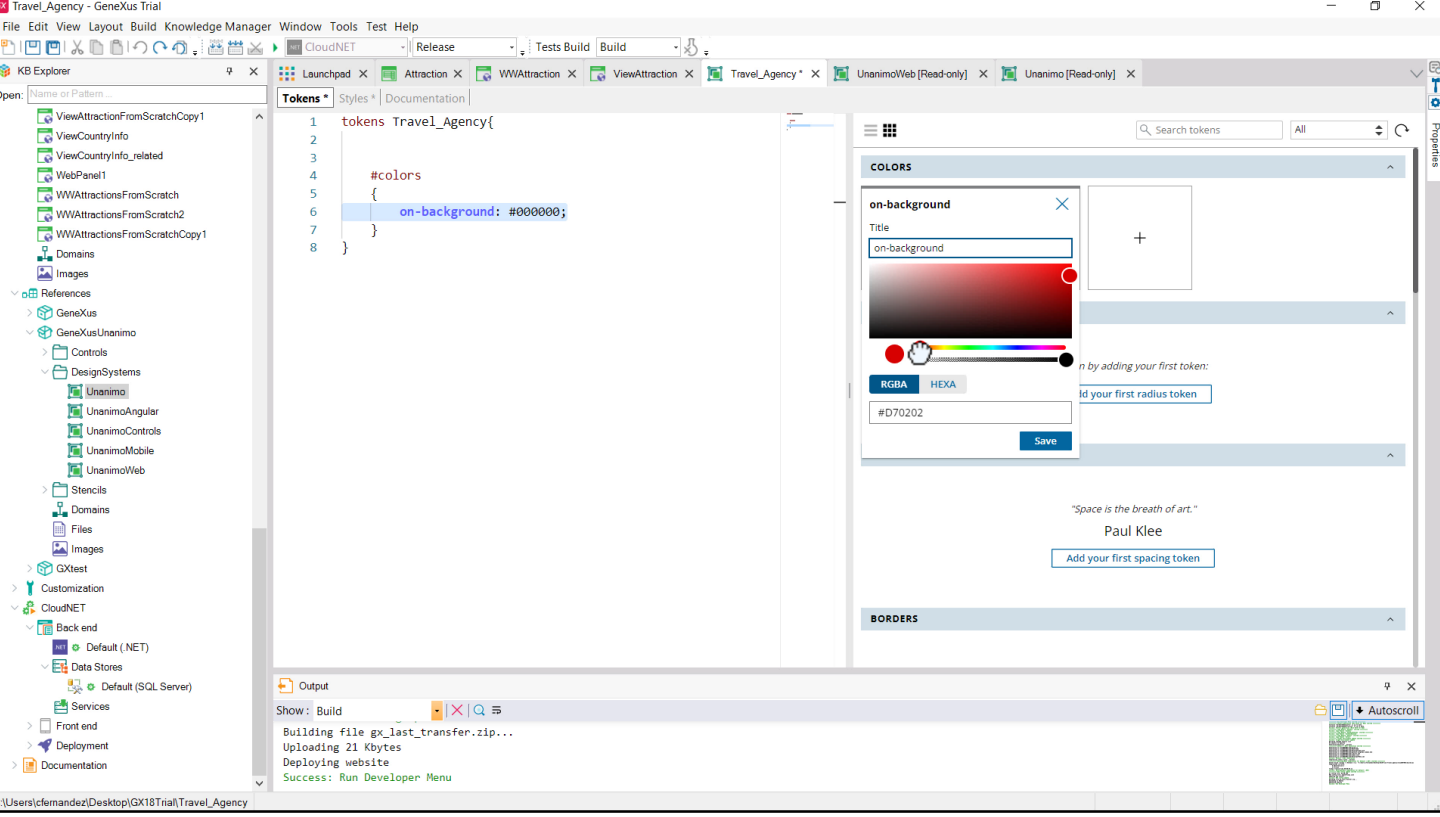
We want to modify its value, but we do not have the change enabled. The reason is that this object is read-only. So, this customization will have to be done in our Design System object, that of our application. Let's copy the definition of the token.



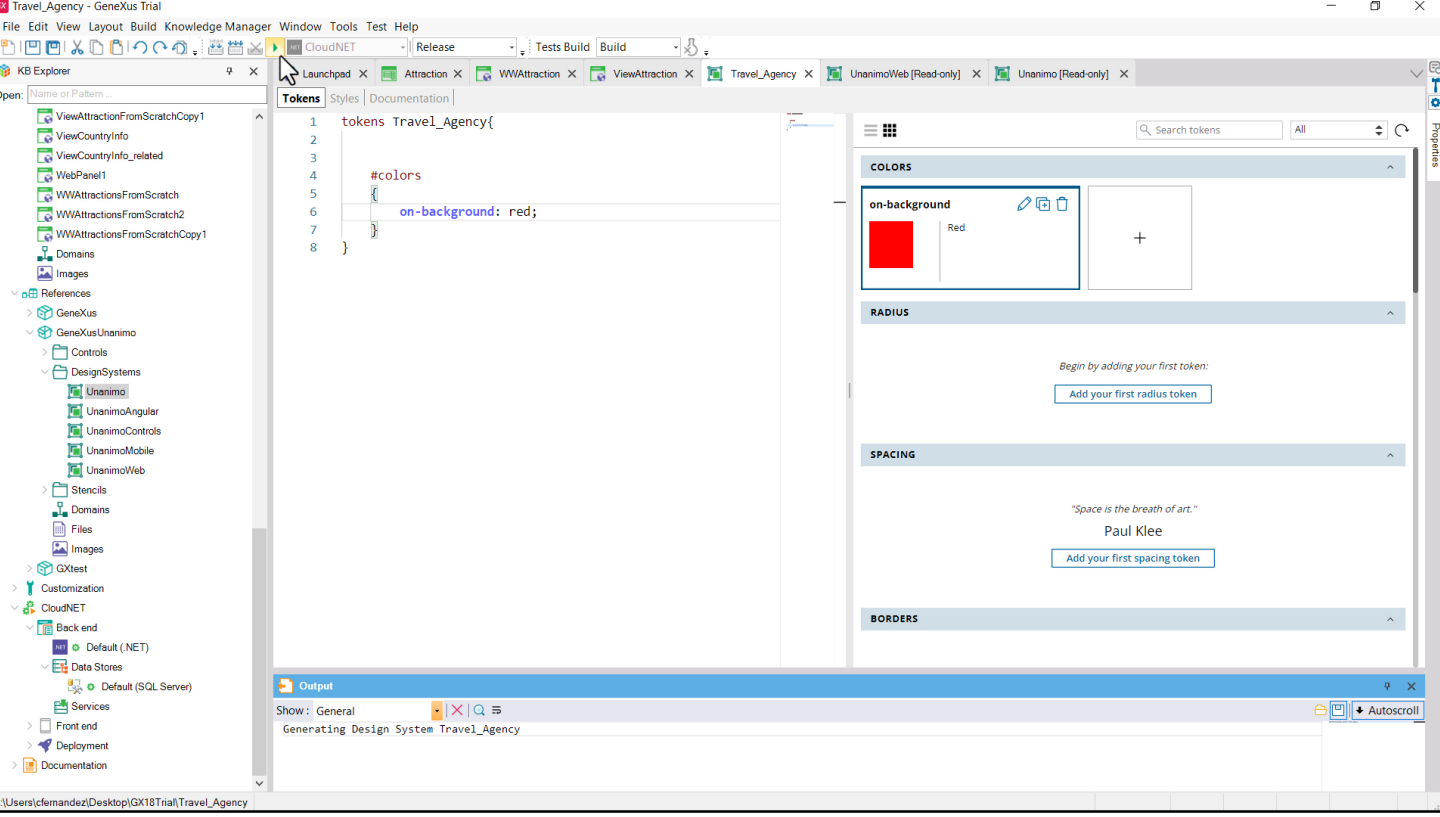
Note that tokens are specified within a set of all the tokens of the object. We will not go into details about the light and dark options here. We will only say that the token definitions can vary according to these parameters.



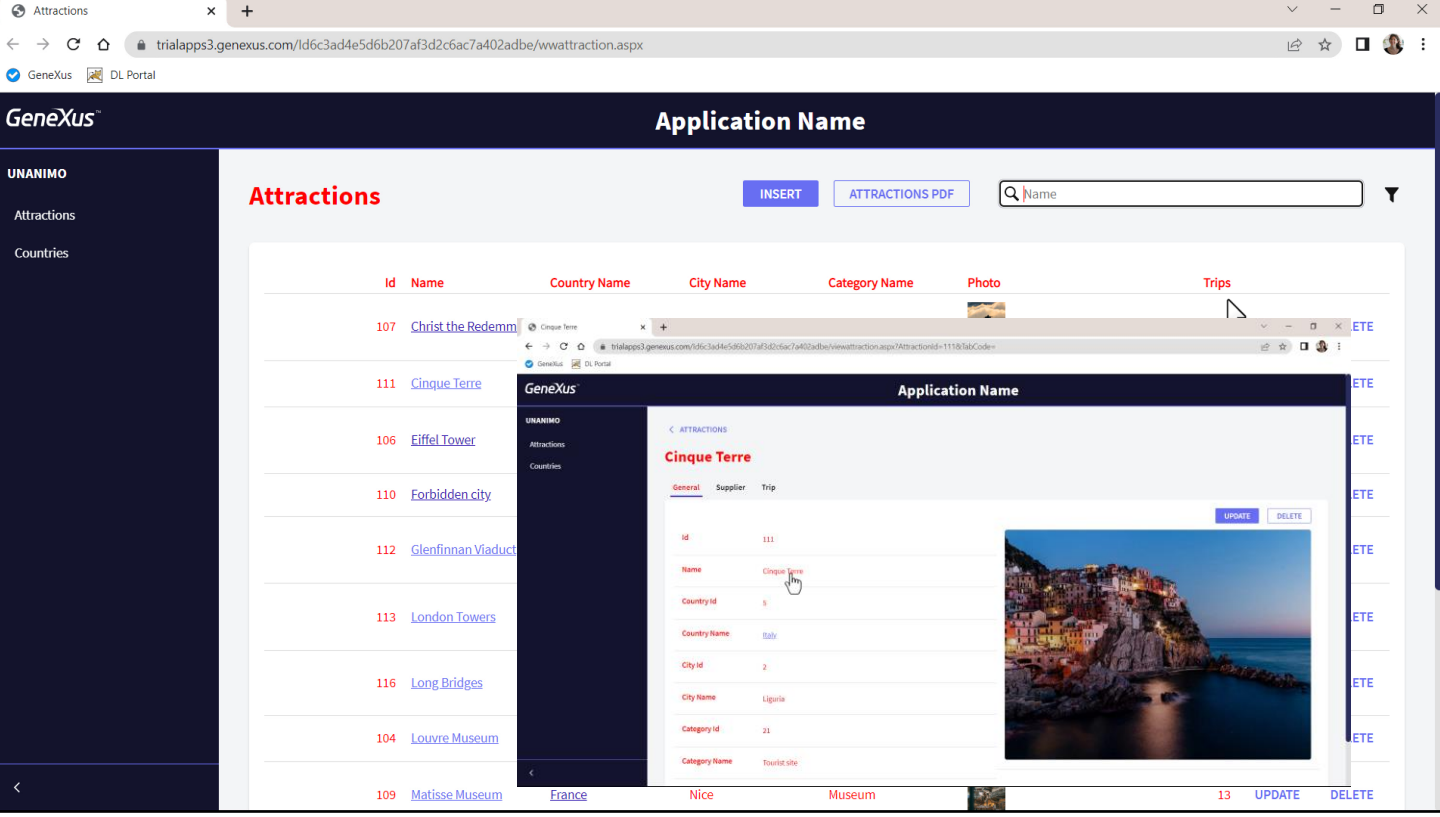
Therefore, in the Tokens tab of our Design System object we define the set following the syntax. What we do is simply overwrite the color token that will be imported along with all the others of the DSO that in turn imports those of the other ones.



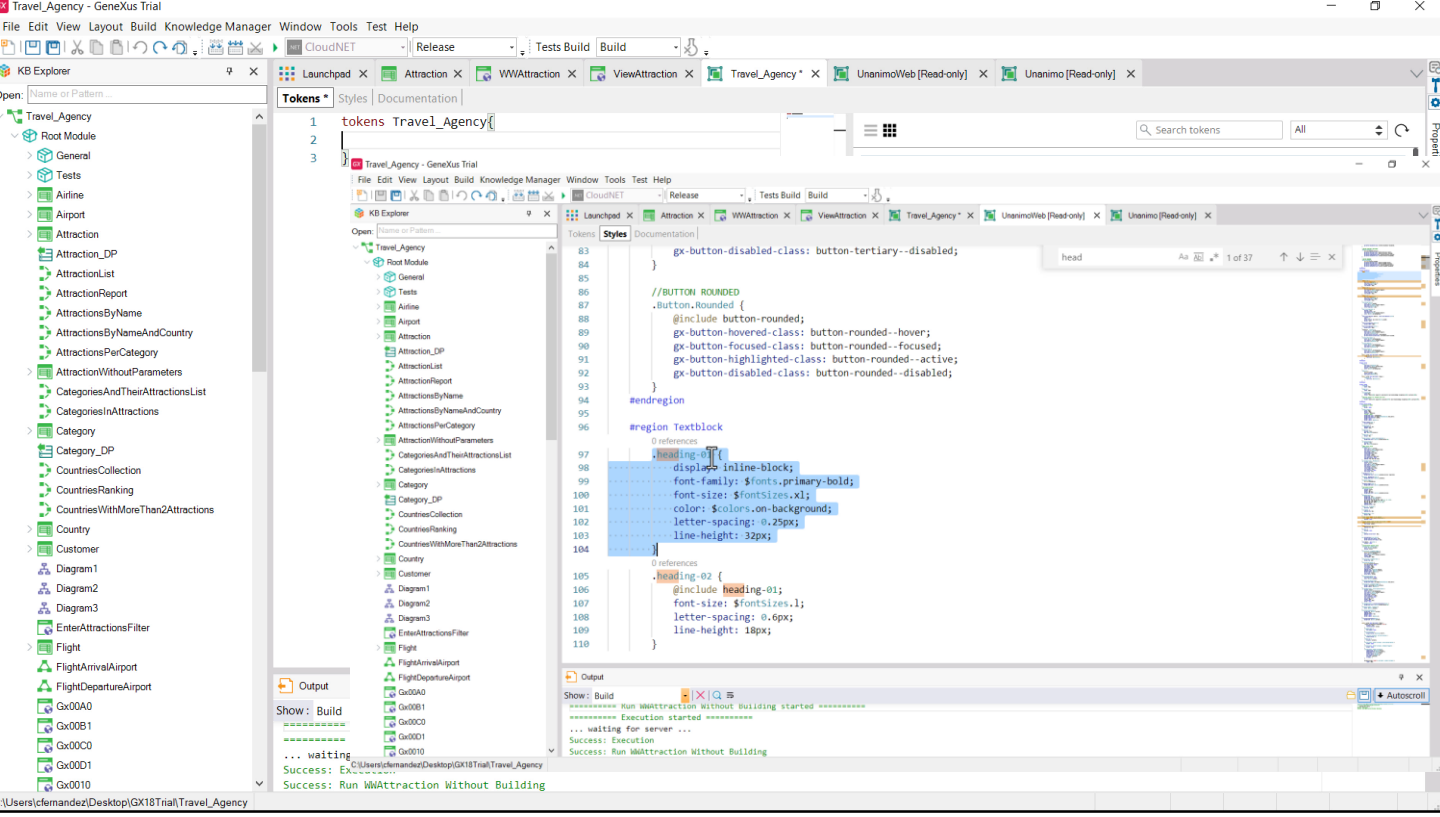
We change the color value here, with this hexadecimal...



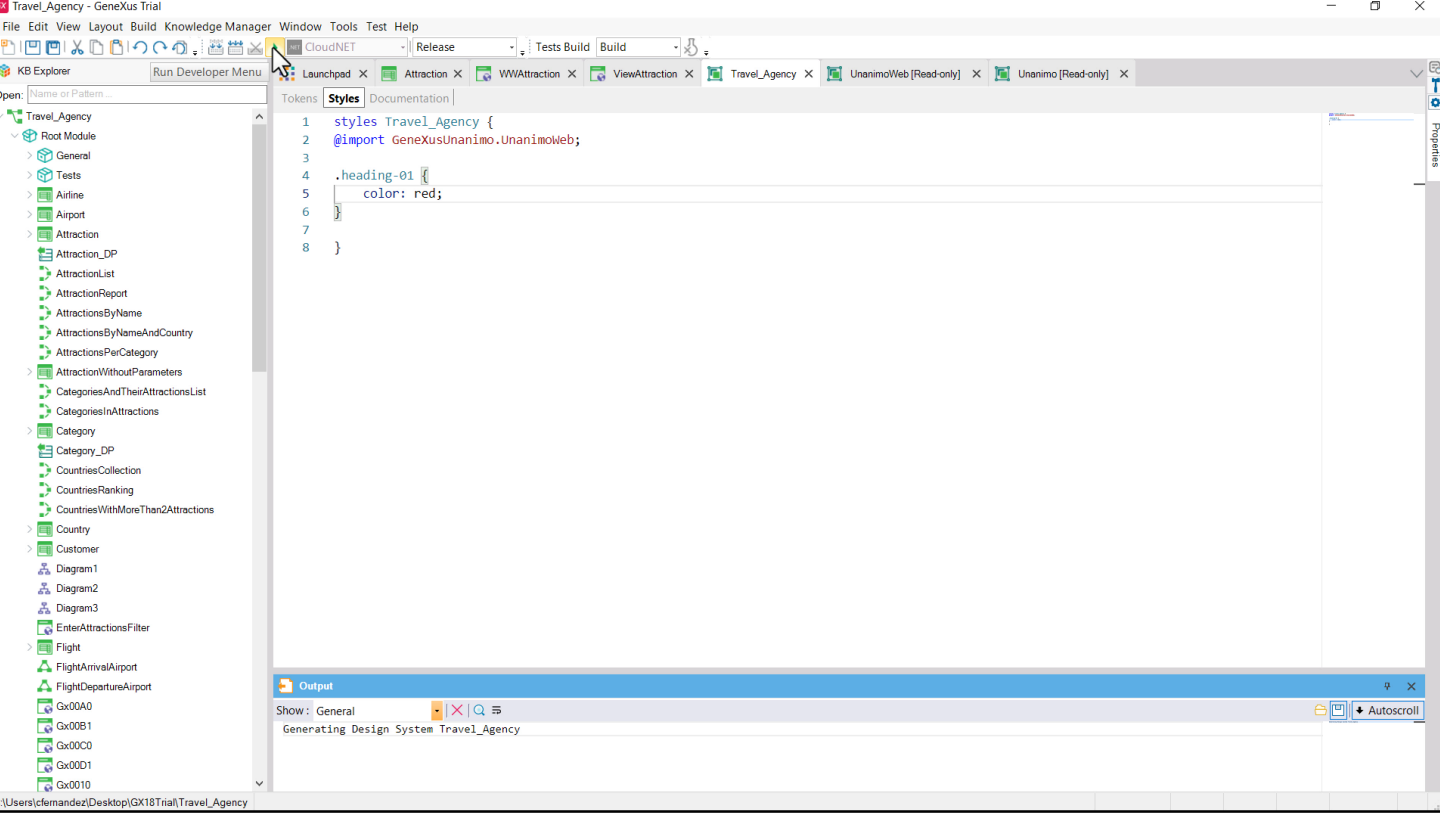
...or, directly, with the value red. Let's save and run.



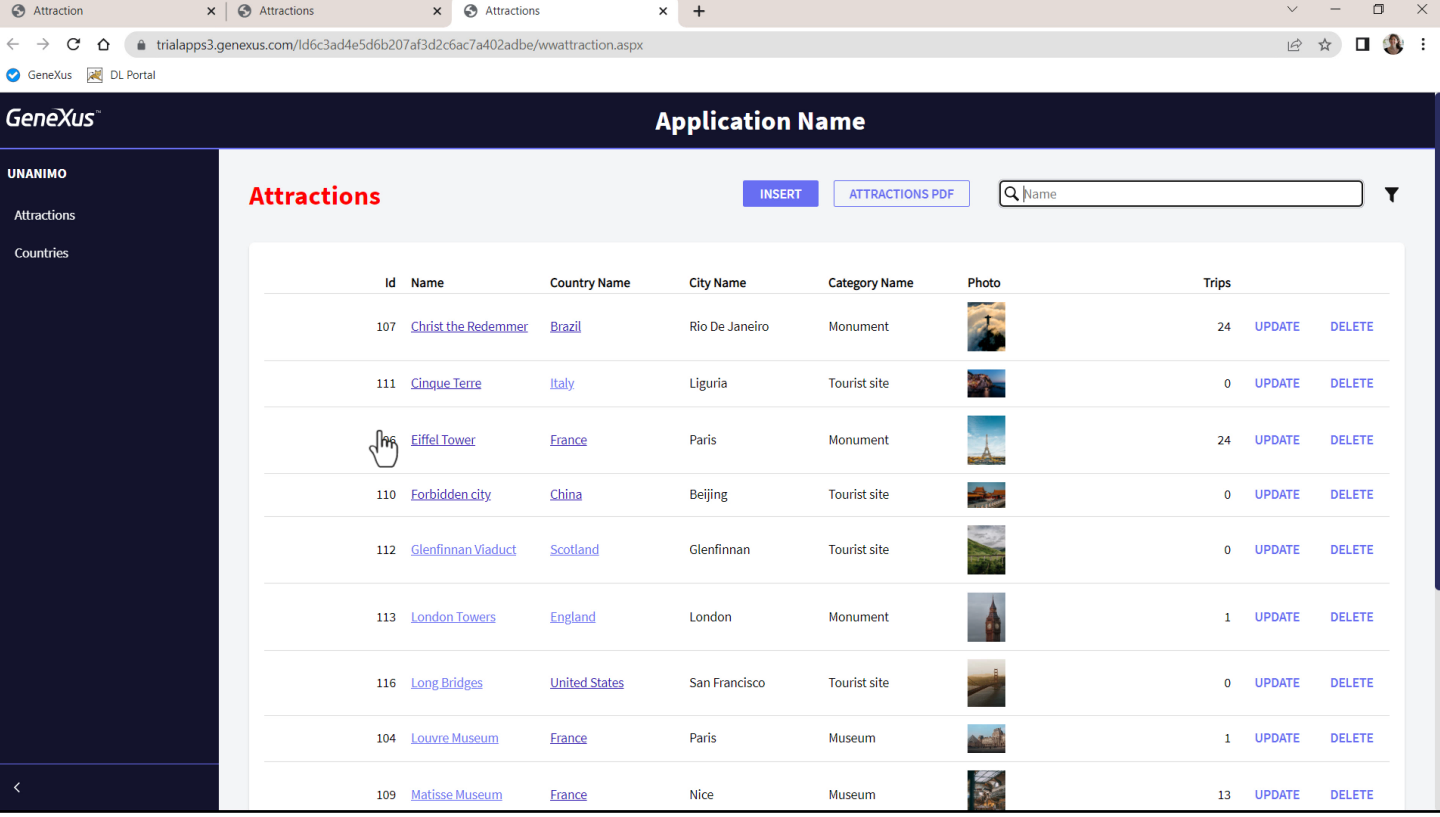
We managed to change the color of the title of each page, but not just that. We changed the color of many more controls than we wanted. It is that, of course, the "on-background" color token is used in the class of these controls, but obviously it is also used in many others.



If we only want to change it for that class, then... what we have to overwrite is the class (here we can't, it's read-only). So we copy it...



...and paste it into our DSO. But actually, since it is imported we don't need to overwrite everything, but only the property that we are interested in changing. We save and run...



We press Ctrl + F5 so that it is refreshed... and now it does. Only the color of the controls that have the heading-01 class associated with them changed.

Attraction | Attractions | Christ the Redemmer

trialapps3.genexus.com/Id6c3ad4e5d6b207af3d2c6ac7a402adbe/viewattraction.aspx?AttractionId=107&TabCode=

GeneXus | DL Portal

GeneXus

Application Name

UNANIMO

- Attractions
- Countries

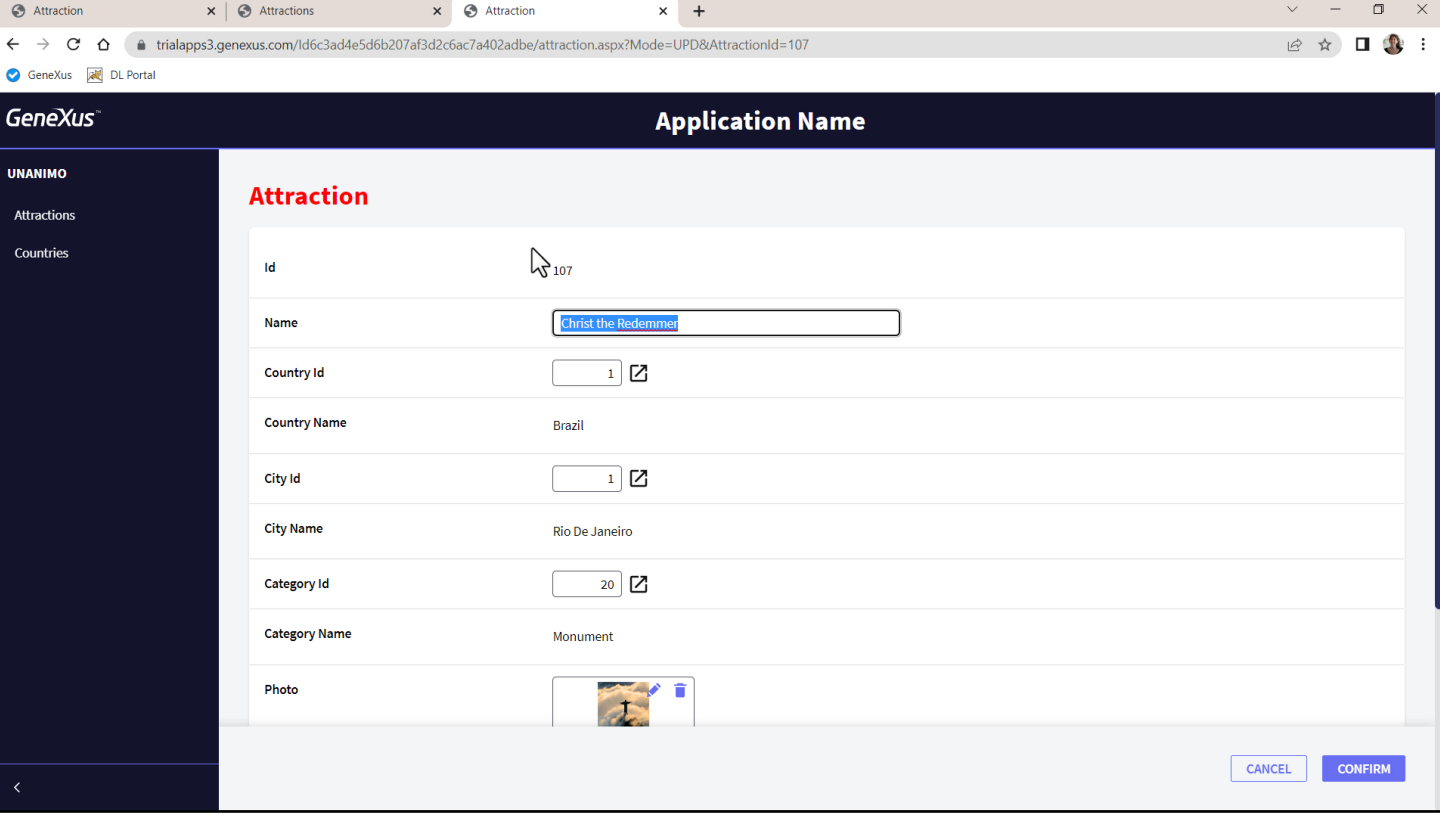
ATTRACTIONS

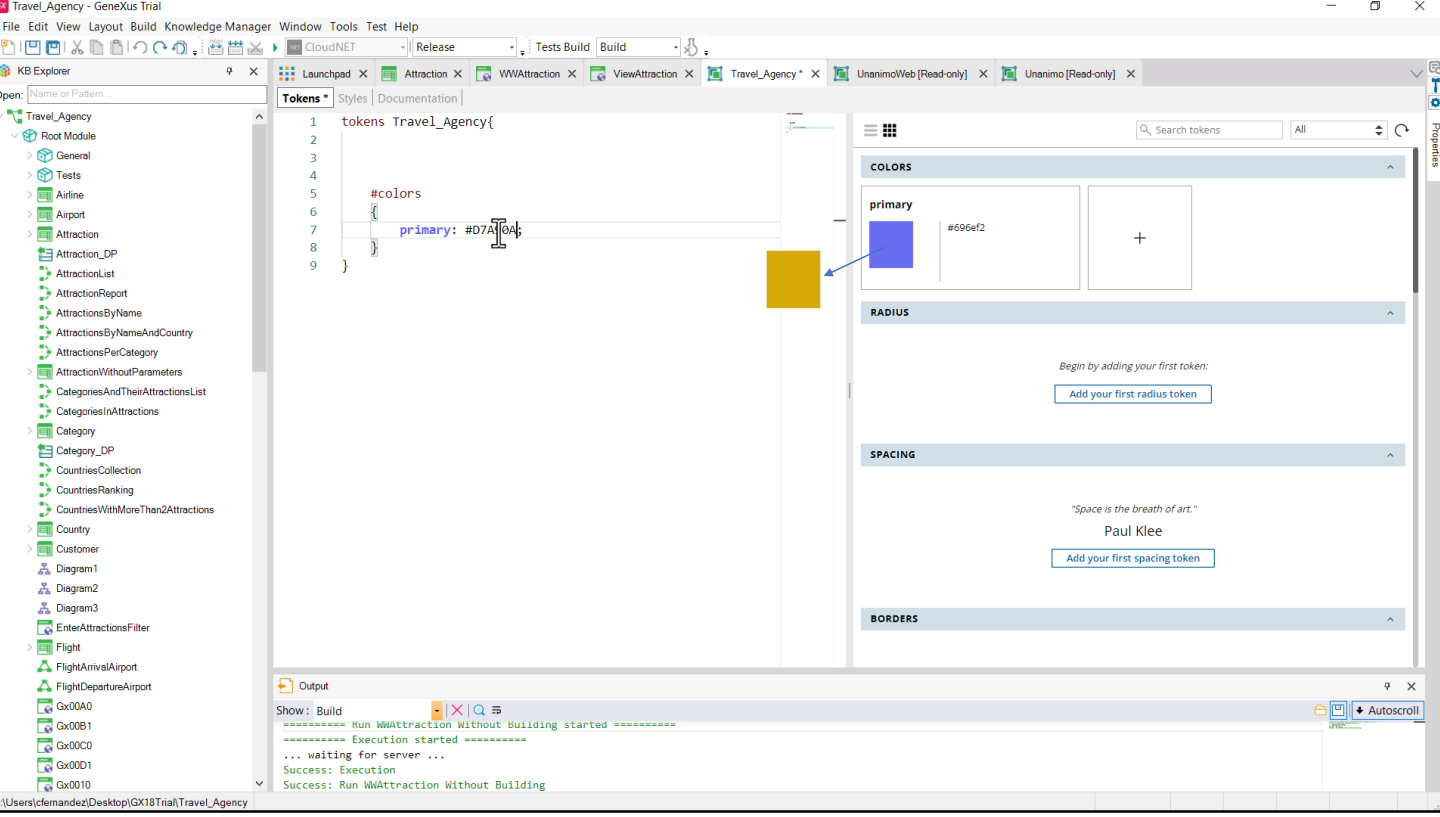
Christ the Redemmer

General | Supplier | Trip

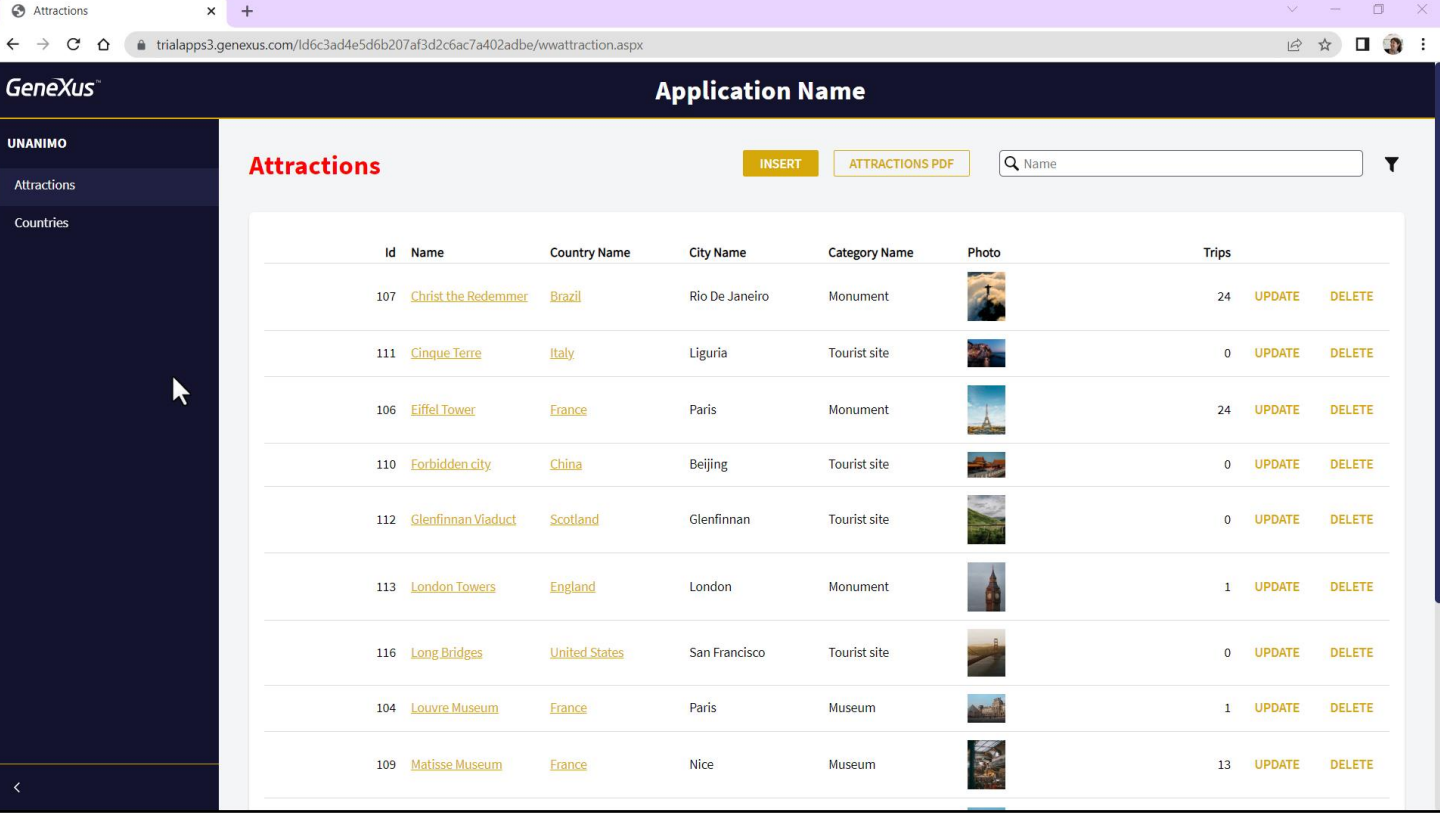
UPDATE | DELETE

Id	107
Name	Christ the Redemmer
Country Id	1
Country Name	Brazil
City Id	1
City Name	Rio De Janeiro
Category Id	20
Category Name	Monument

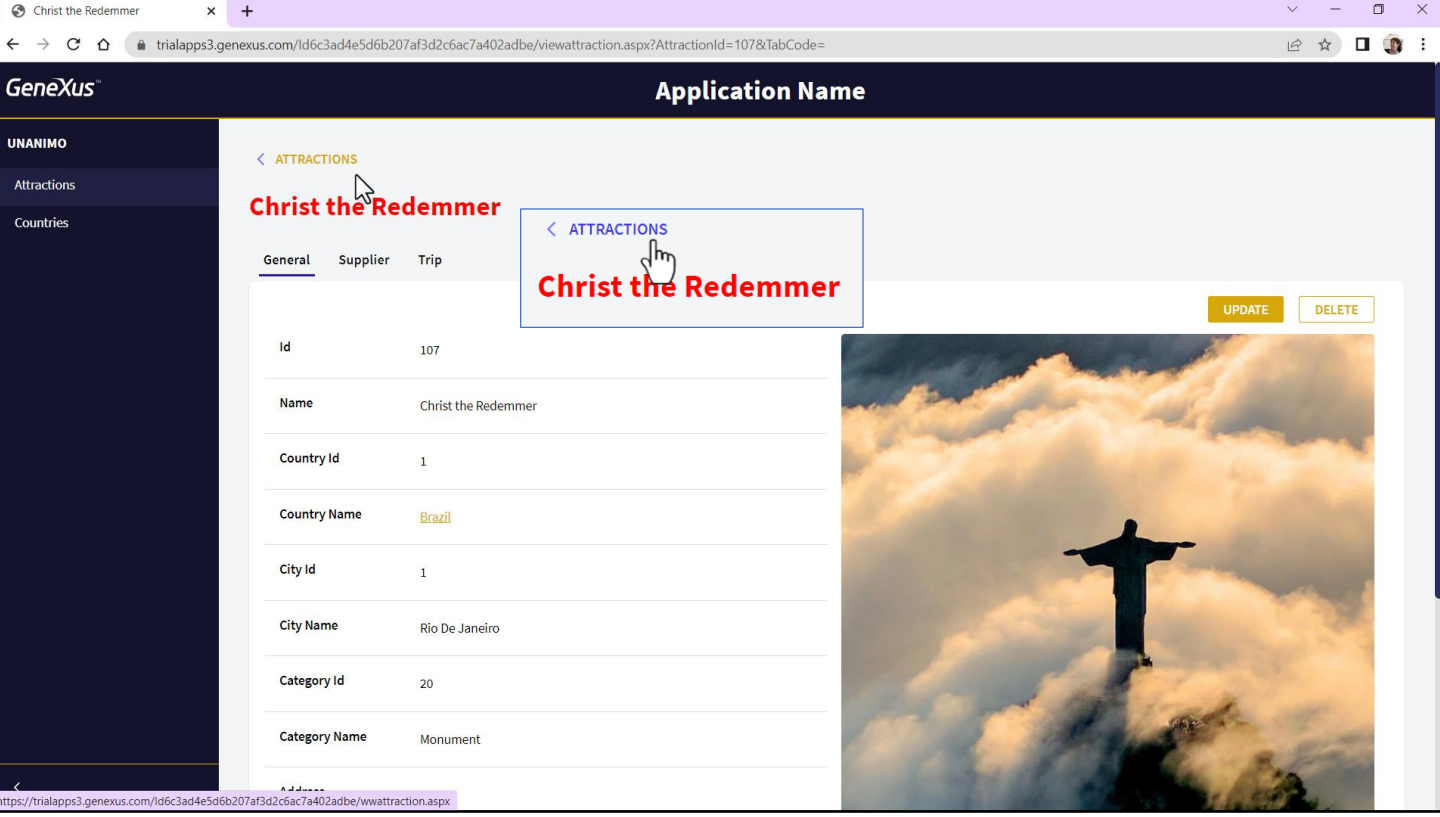

Attraction
Attractions
Attraction
trialapps3.genexus.com/Id6c3ad4e5d6b207af3d2c6ac7a402adbe/attraction.aspx?Mode=UPD&AttractionId=107
GeneXus
DL Portal
GeneXus
Application Name
UNANIMO
Attractions
Countries
Attraction
Id
107
Name
Christ the Redemmer
Country Id
1
Country Name
Brazil
City Id
1
City Name
Rio De Janeiro
Category Id
20
Category Name
Monument
Photo
CANCEL
CONFIRM



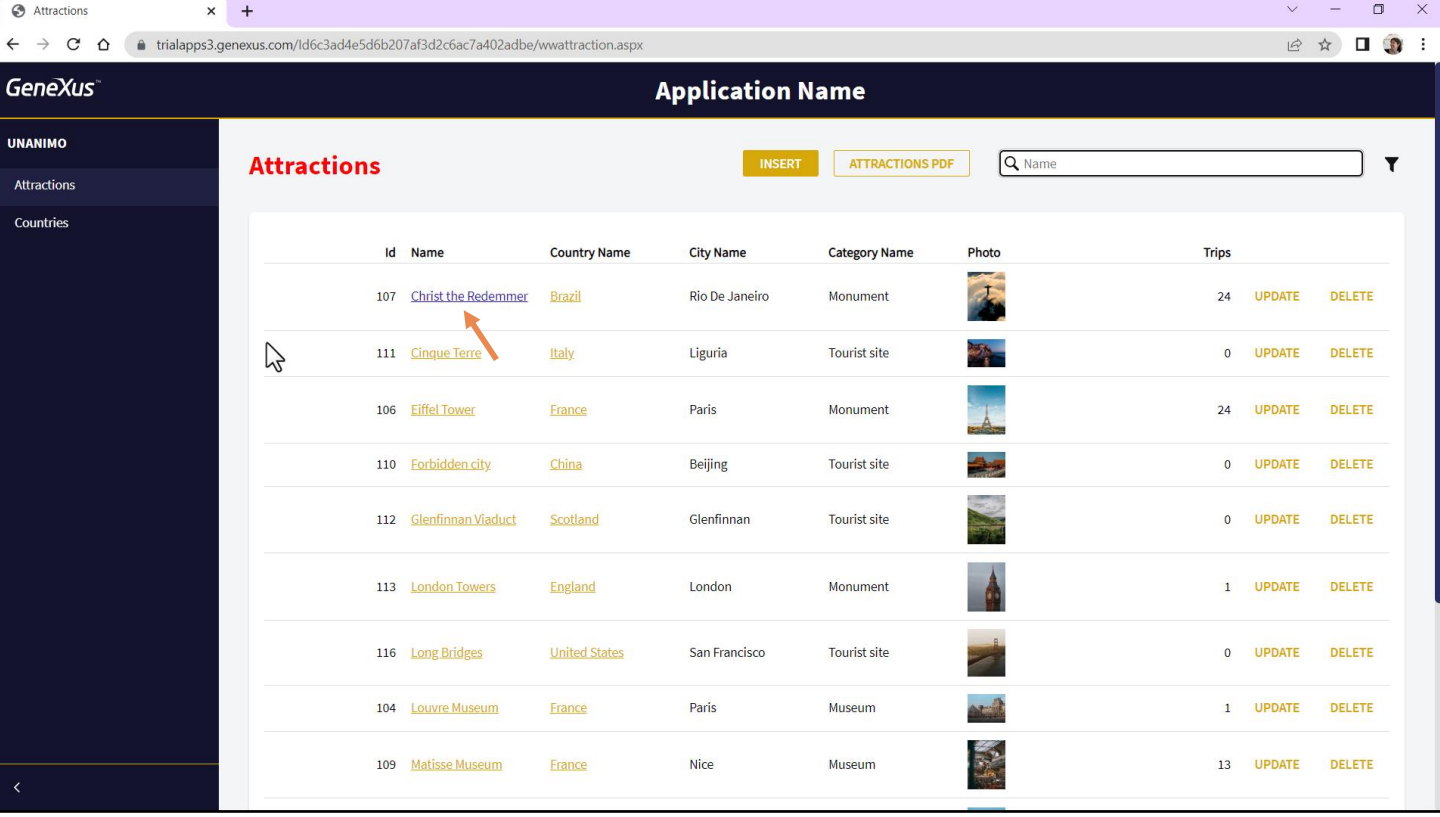
Considering everything we've seen so far, what should happen if we change the value of the “primary” color token?



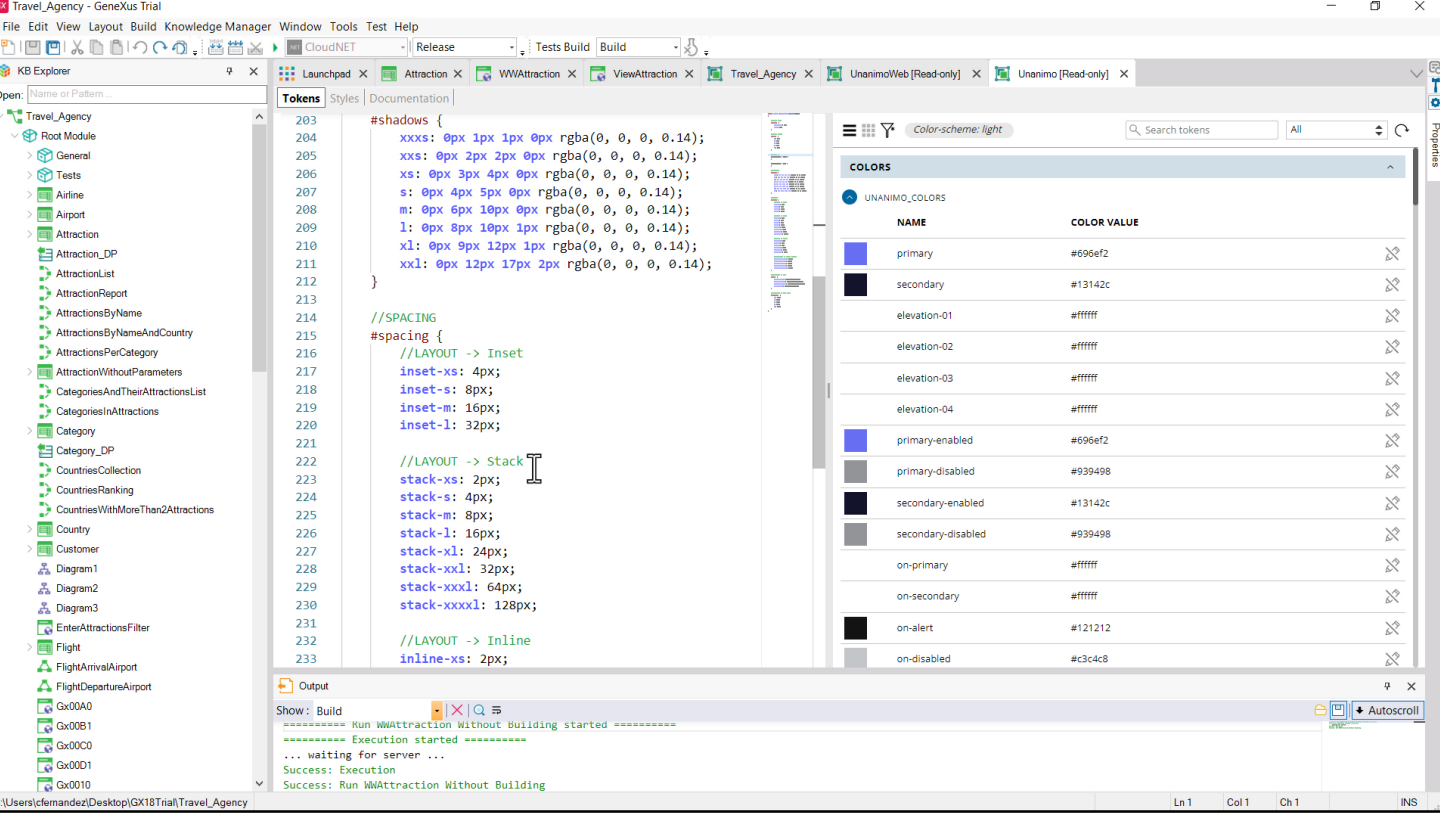
Note how the color of the buttons and actions in general have changed.



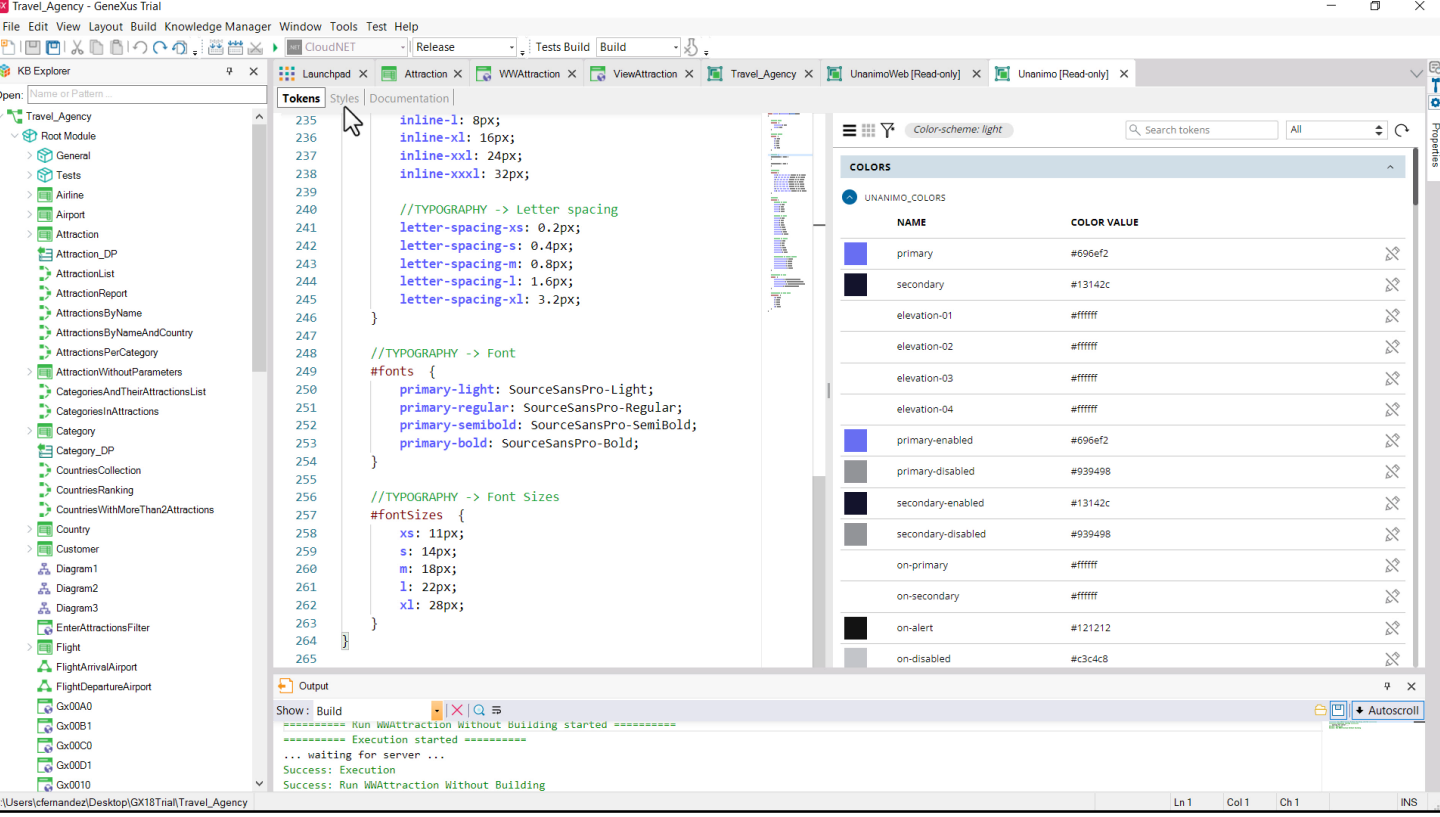
Here we can see that the hover action changes the color and we can assume that this will be a property of the class that controls the hover effect. The same goes for the buttons.

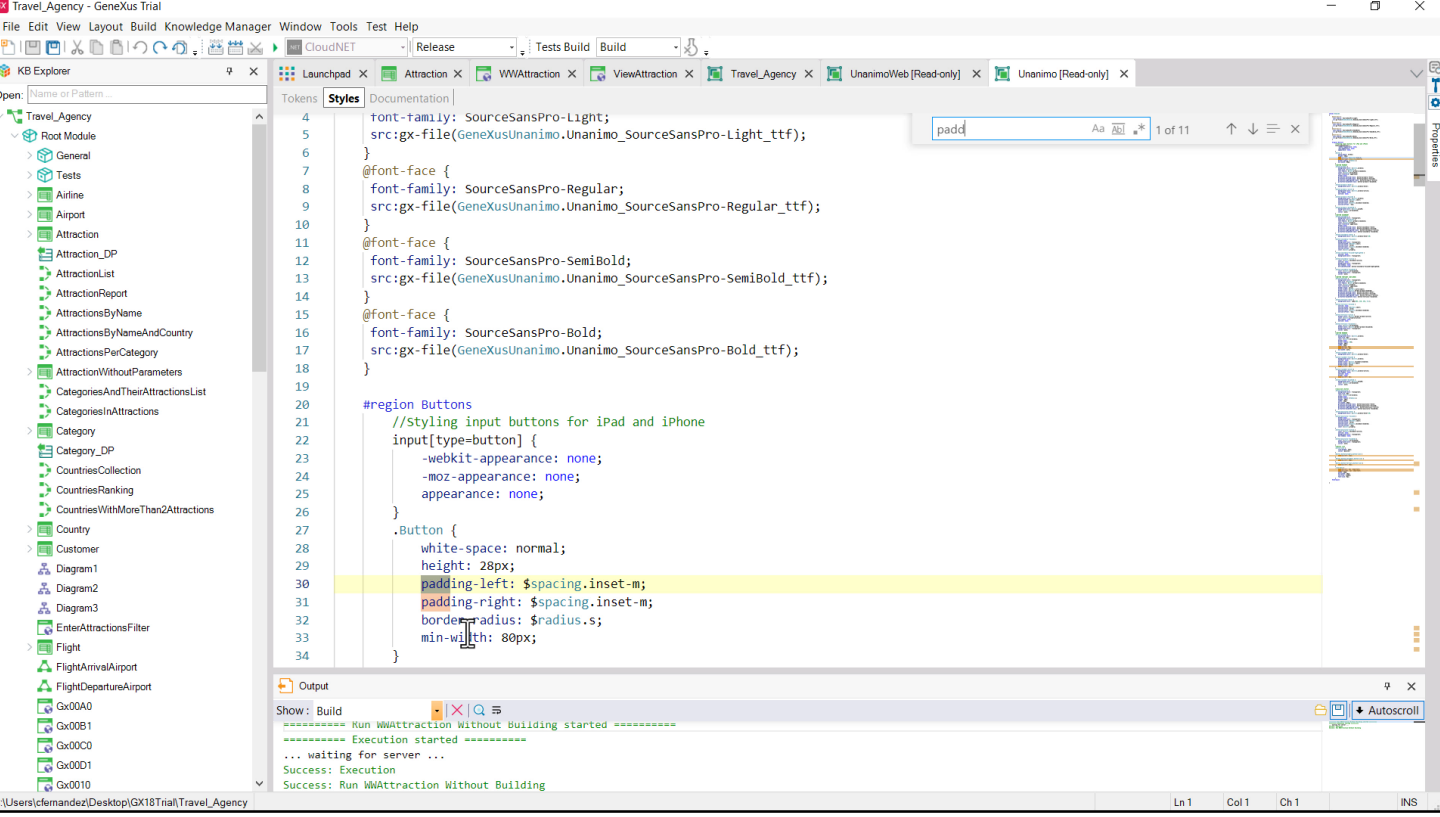


Even here the visited link is shown in this color, something that we can also change in the DSO.

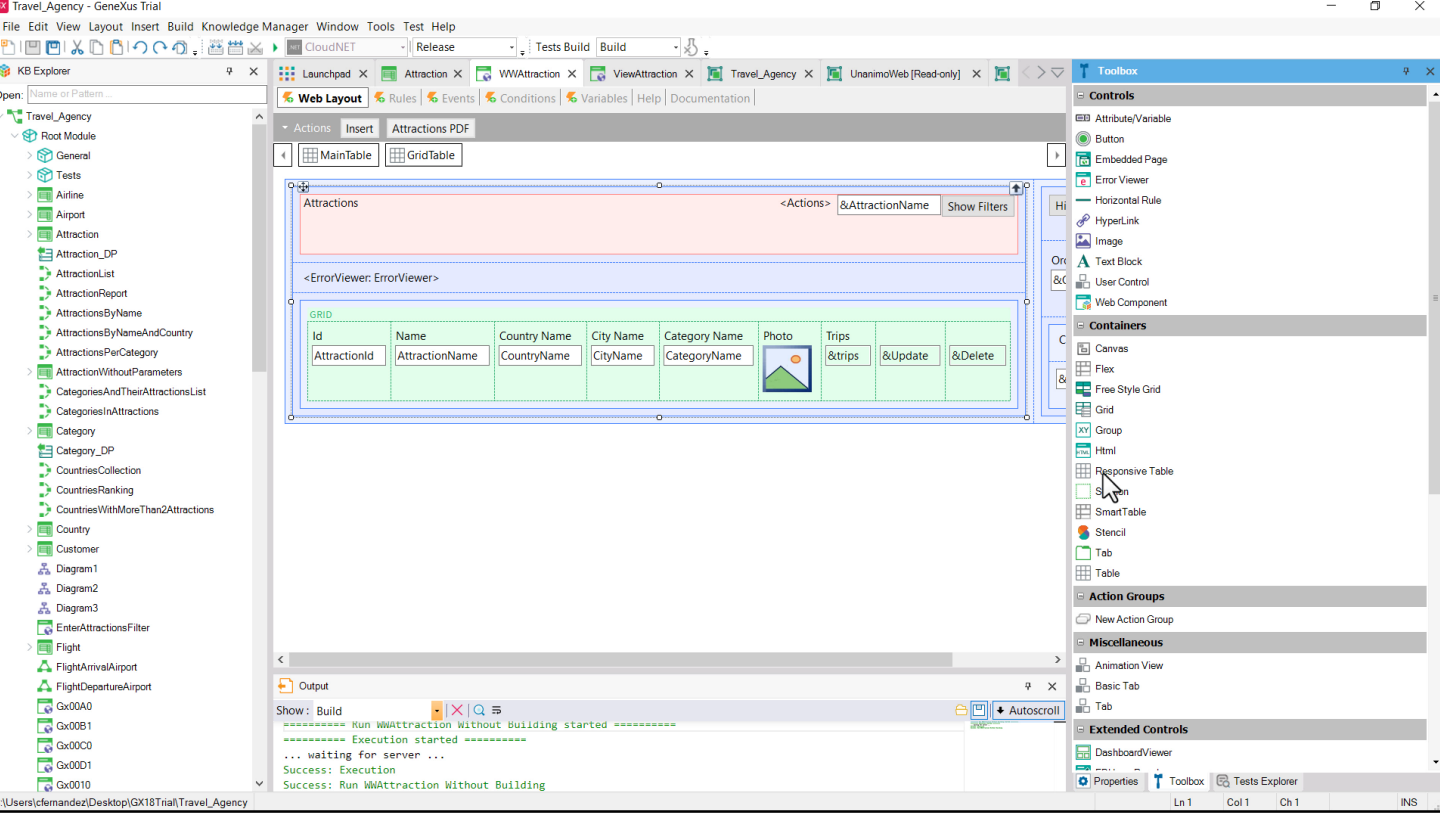


We talked about the importance of spacing, for example, and of defining appropriate font families and sizes to be used consistently in the application. We will not be surprised to find these tokens in Unanimo...


Travel_Agency - GeneXus Trial
File Edit View Layout Build Knowledge Manager Window Tools Test Help
KB Explorer
Launchpad
Attraction
WWAttraction
ViewAttraction
Travel_Agency
UnanimoWeb [Read-only]
Unanimo [Read-only]
Tokens
Styles
Documentation
inline-l: 8px;
inline-xl: 16px;
inline-xxl: 24px;
inline-xxxl: 32px;
//TYPOGRAPHY -> Letter spacing
letter-spacing-xs: 0.2px;
letter-spacing-s: 0.4px;
letter-spacing-m: 0.8px;
letter-spacing-l: 1.6px;
letter-spacing-xl: 3.2px;
//TYPOGRAPHY -> Font
#fonts {
primary-light: SourceSansPro-Light;
primary-regular: SourceSansPro-Regular;
primary-semibold: SourceSansPro-SemiBold;
primary-bold: SourceSansPro-Bold;
//TYPOGRAPHY -> Font Sizes
#fontSizes {
xs: 11px;
s: 14px;
m: 18px;
l: 22px;
xl: 28px;
Color-scheme: light
Search tokens
COLORS
UNANIMO_COLORS
NAME
COLOR VALUE
primary #696ef2
secondary #13142c
elevation-01 #ffffff
elevation-02 #ffffff
elevation-03 #ffffff
elevation-04 #ffffff
primary-enabled #696ef2
primary-disabled #939498
secondary-enabled #13142c
secondary-disabled #939498
on-primary #ffffff
on-secondary #ffffff
on-alert #121212
on-disabled #c3c4c8
Output
Show : Build
Autoscroll
========== Execution started ==========
... waiting for server ...
Success: Execution
Success: Run WWAttraction Without Building

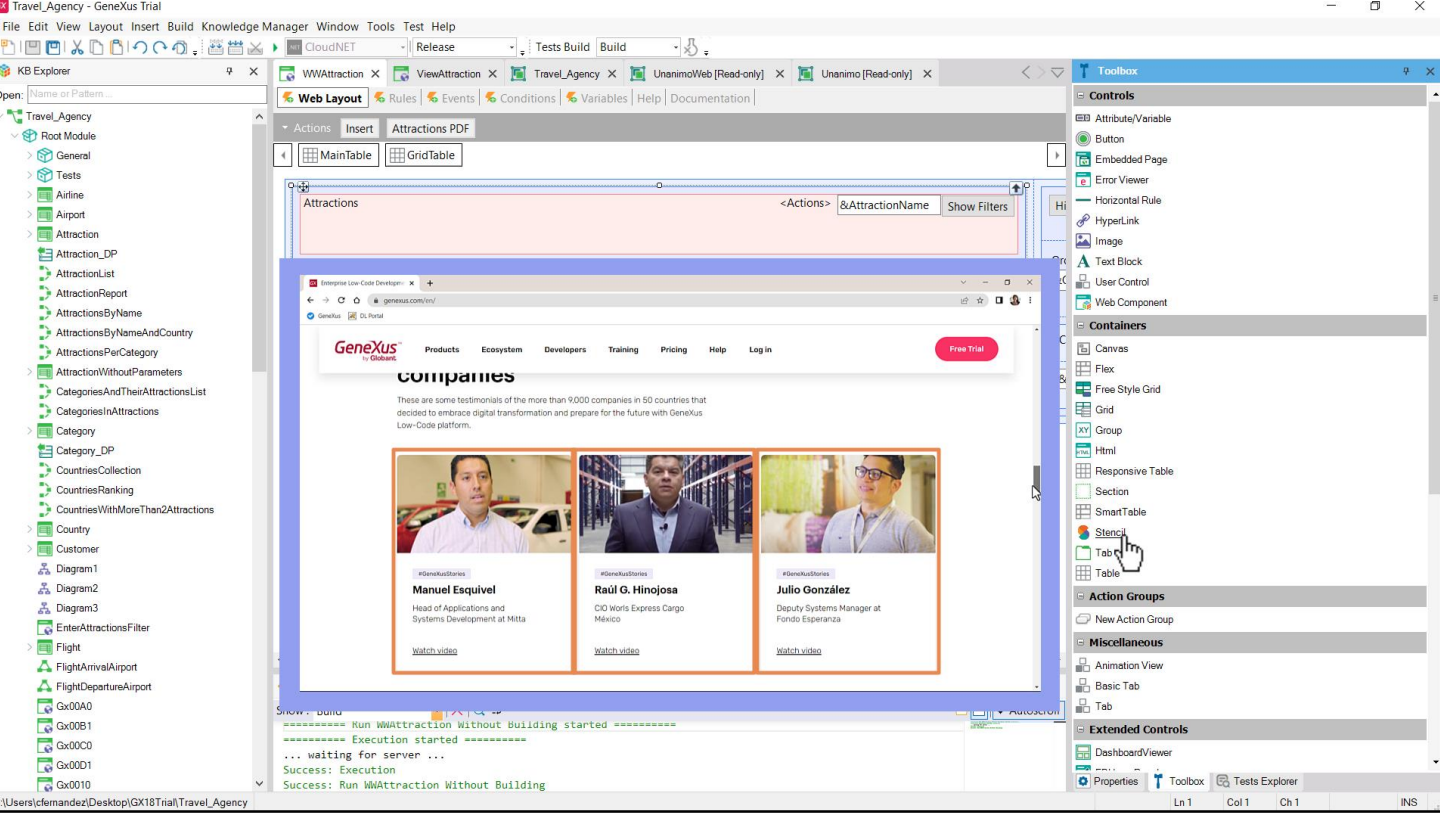


... they will be used in the classes assigned to the properties to give space or to choose fonts.

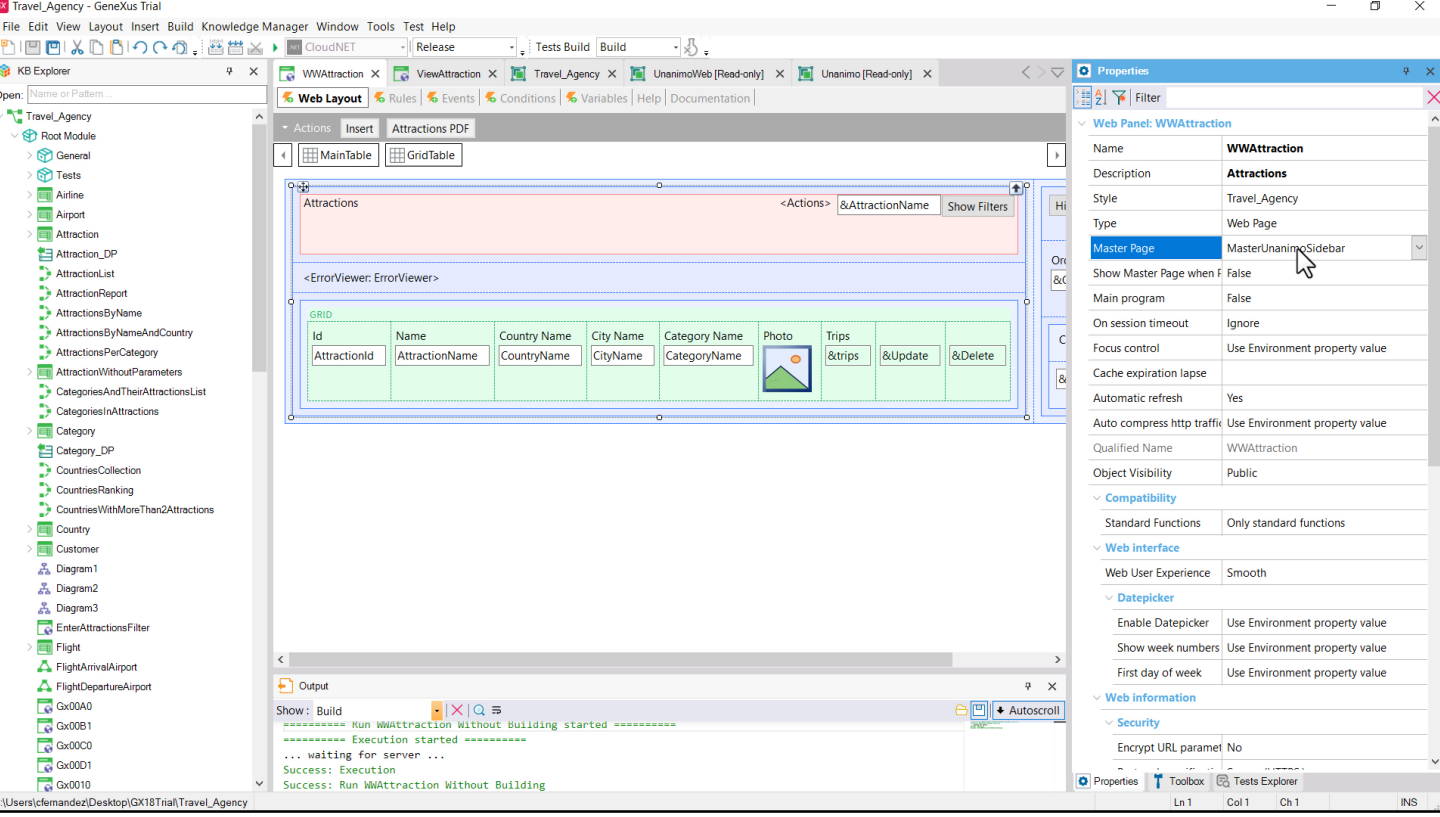


On the other hand, how do you achieve alignment and combine information elements so that they work as a block? That is where GeneXus controls are involved. Mainly the tables.

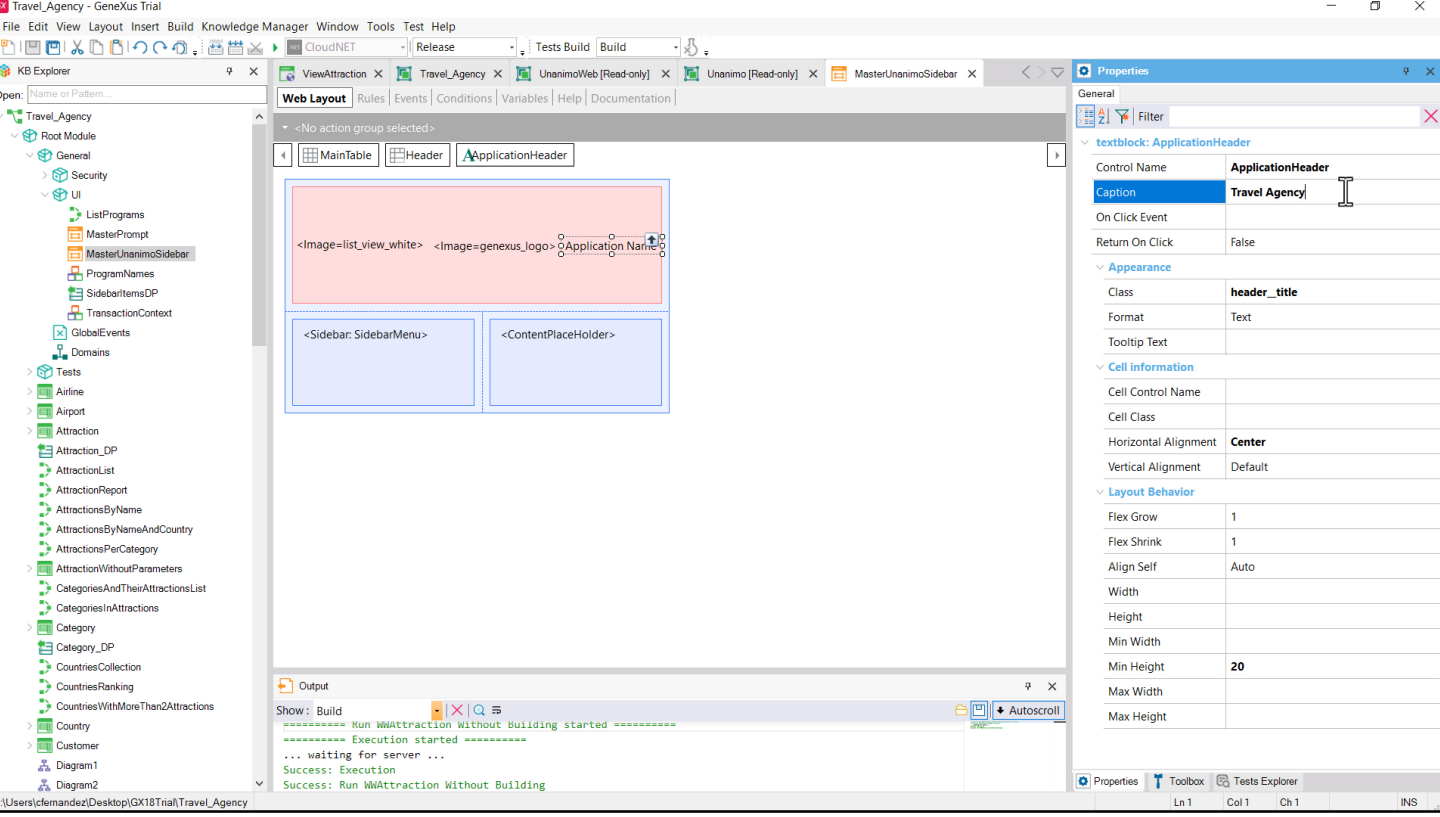
In other words, two key players will be involved: the controls in layouts and their classes in the DSO.



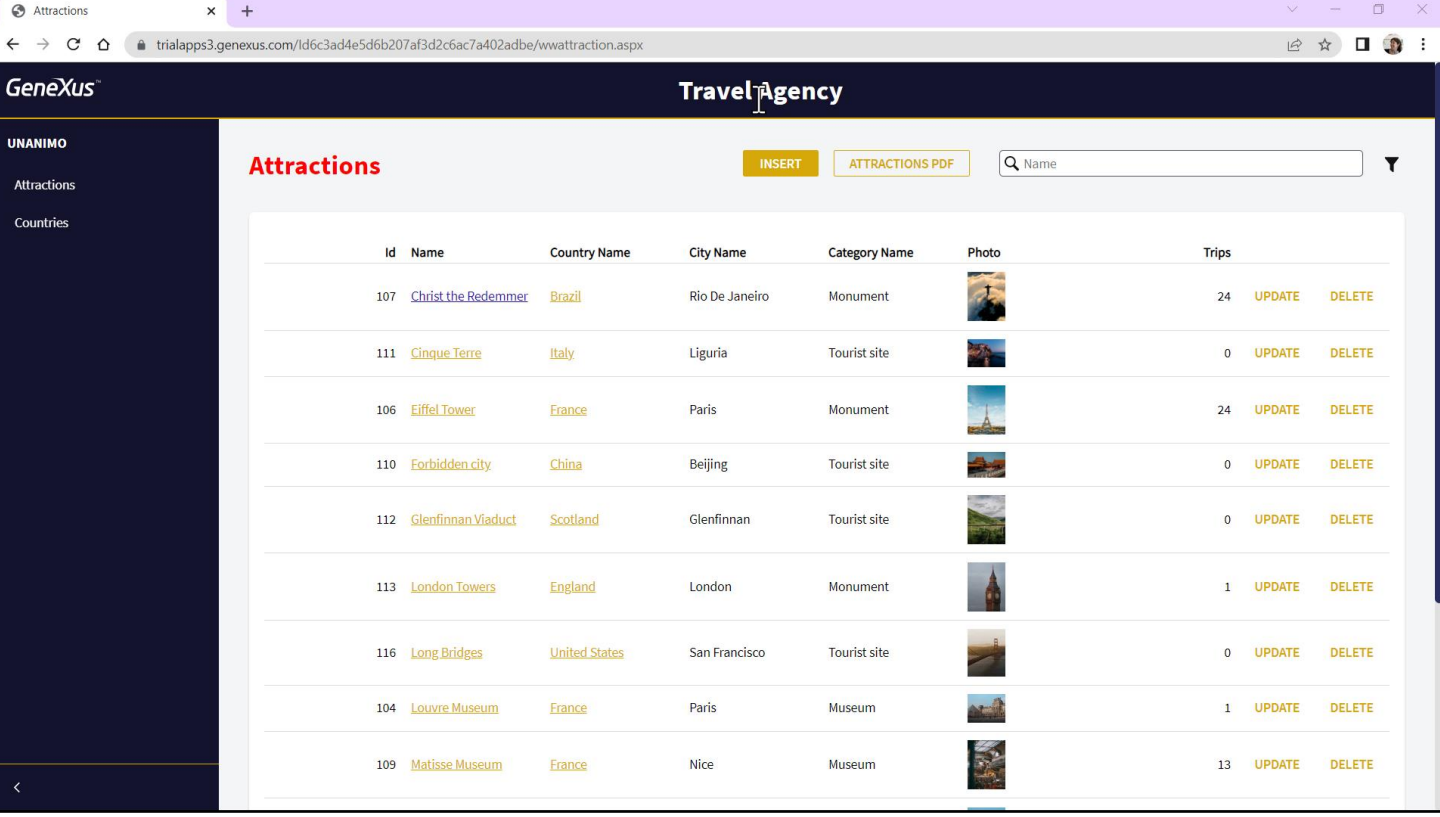
In addition, in order to save resources, we will be able to encapsulate repetitive information units in independent objects that can then be inserted in the layouts, such as stencils or web components.



Even the header and footer are separated in another object, the Master Page.



Here it is. Let's see what happens when we change this text block.



In this way, we start to understand how to use a Design System in GeneXus.

GeneXus™

training.genexus.com
wiki.genexus.com