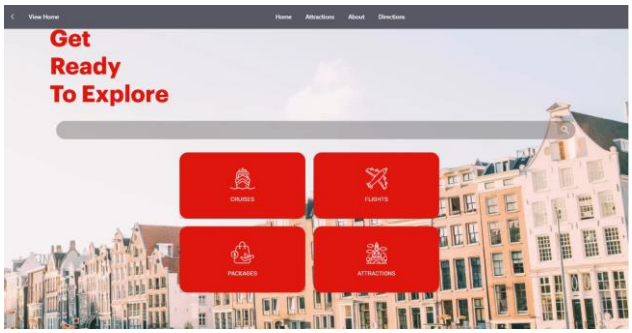
Designing an Angular App

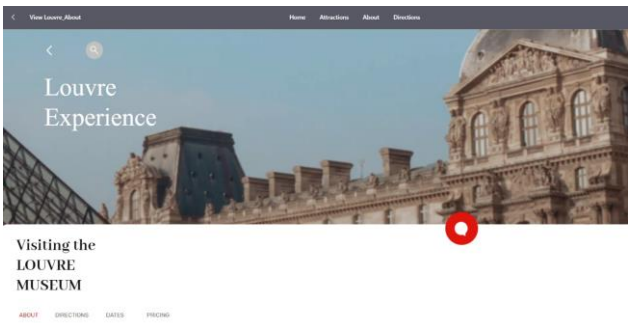
Introduction to Design System Object

GeneXus™

In previous videos, mention was made of an enhancement included in GeneXus called Design System Object, which is an evolution of Theme object and classes. This video shows how to use the object in design definitions for your app.

Design System Concept





- Master Panel
- Panels
- Components
- Stencils
- Controls
- Patterns

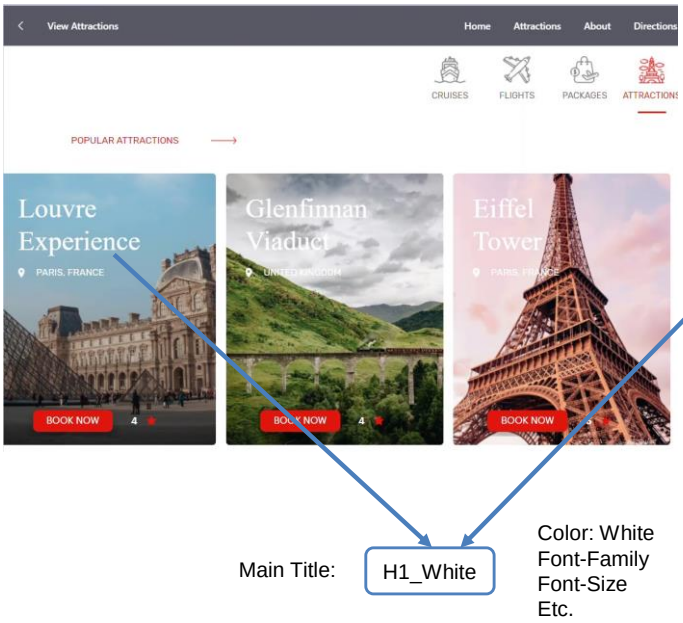
If you analyze the home page in the definitive app you will note that certain decisions remain in several screens, such as the color of buttons matching the title, or the title over a background image, which is also the case in the second screen, as well as the type and aspect of texts that remain the same throughout the various screens.

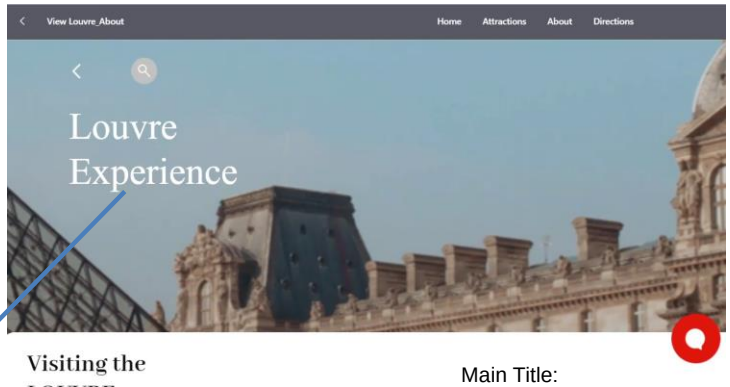
The regularity with a prevailing style and a uniform way of presenting information and standards for the whole app is a Design System in action. To abstract the definitions, you use a series of objects such as Master Panels, Panels, Components, Stencils, Controls, and Patterns.

The definition of controls on screen, as the main elements of the user interface, is particularly important regardless of the screen they are on. So far, these definitions were made by means of the Theme object that grouped the classes where the aspect of screen controls was defined.

The Design System Object is an evolution of the Theme object that reduces the gap between designers and developers, while preserving the general principle of GeneXus, that is, expressing the most while writing the least.

Design System Definitions





Visiting the LOUVRE MUSEUM

Main Title: H1_Black

Color: Black
Font-Family
Font-Size
Etc.

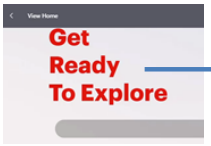
Main Title: H1_White

Color: White
Font-Family
Font-Size
Etc.

For example, consider the white title that appears, in every screen, over the photograph of each tourist attraction. You may abstract that common style by assigning to it the same size, font and color, and calling it, for example, H1_White. “H1” because it is a main title and White because that will be color used in cases that include a background image as in this case.

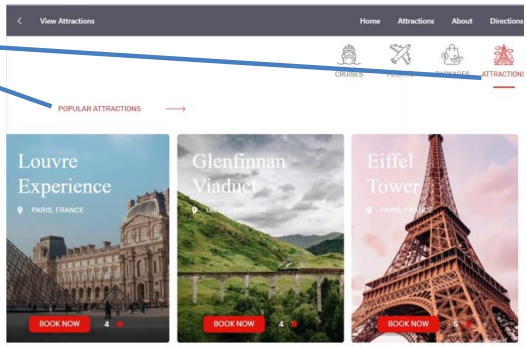
Likewise, you will define this title here as H1_Black.

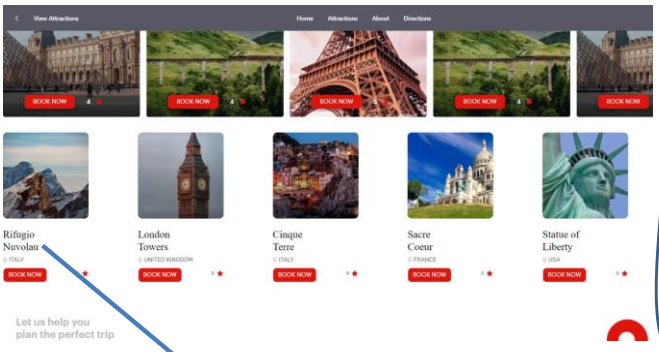
Design System Definitions

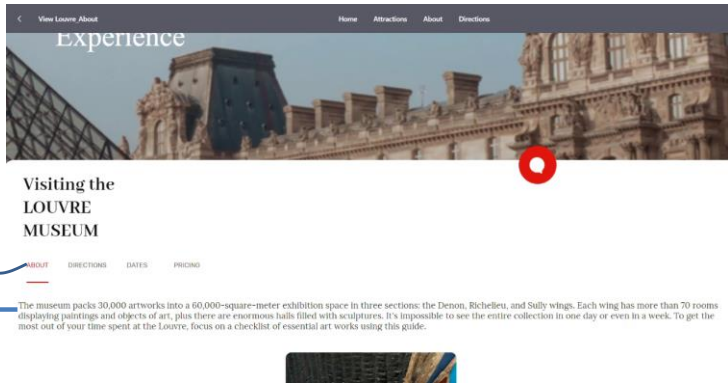


H1_Red

MainActionText







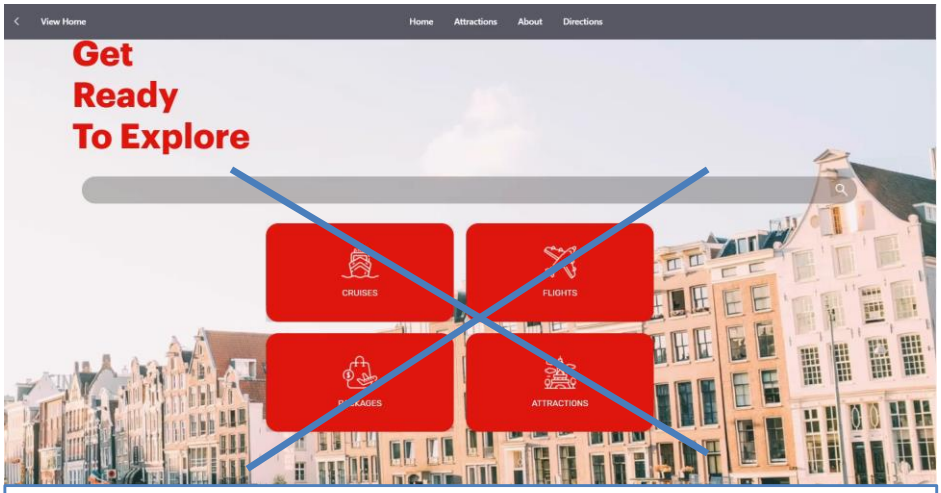
NormalText

Secondary Title: H2

Further abstractions with other texts would be, for example, defining the large title of the first page as H1_Red, this text here as secondary title: H2, and these here in red as texts meant for an action: MainActionText, and this text here as regular or normal text: NormalText.

The names given to those abstractions would then be classes, on which you could base other texts in the application. Identifying these abstractions is in fact modeling the app's Design System.

The Design System Object enables you to concentrate all these abstractions in a single location where you may define the properties of each class.

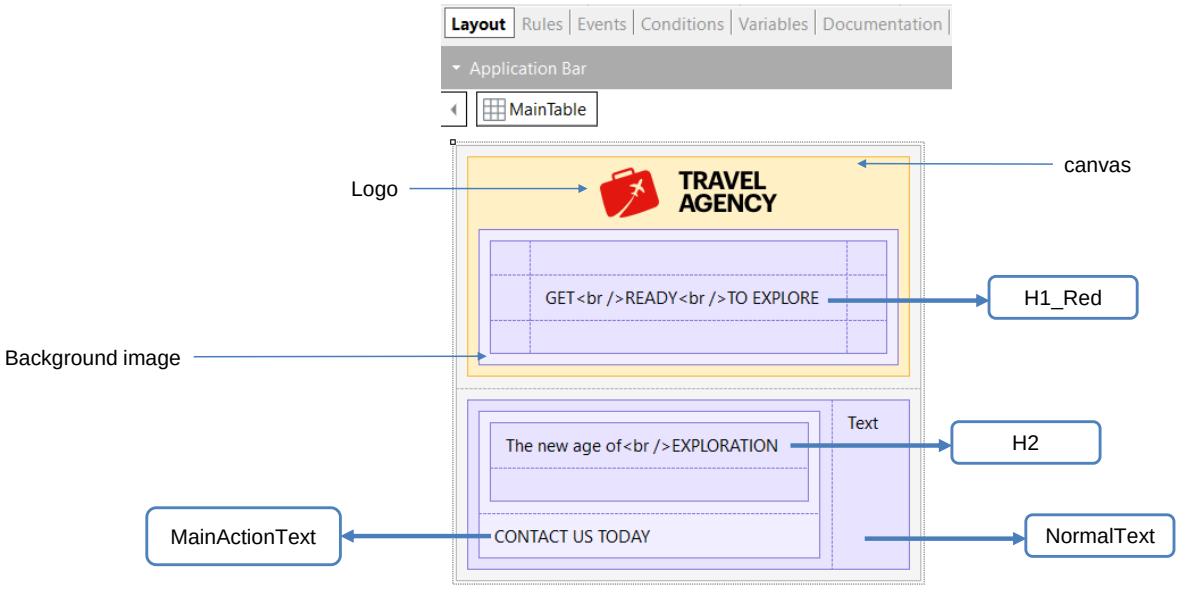


Informative text related to the Travel Agency

Before you start taking these classes to a DSO object, you will create a home panel that will be the startup object in your app, that means, the object that will be executed in the first place and will be the access point to all other screens in the app.

The design is quite simpler than the one shown before, so, for the time being you will include some descriptive texts but not the buttons.

Building the app's home page



Here you have the View_home panel already created. Inside it, there is a canvas control, similar to a table, that will enable you to contain objects executed in various layers, one on top of the other. That is the effect desired with the logo and the texts to be shown over the background image.

In the controls of the canvas you will see that now they all have a Zorder property that is the numeric value indicating which layer they are on. The greater the number, the higher they are.

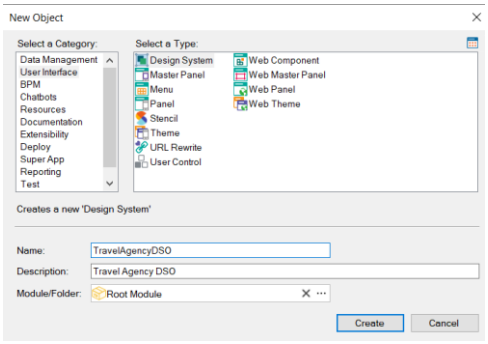
Inside the canvas is a table with its background image property associated with the background image desired. This table includes the Zorder property with value 0, so its background image will be in the lowest part of the canvas. The logo image has Zorder = 1, so it will be seen over the background image.
The textblock with the main title is in a table located inside the first one, so it will appear over the background image as well.

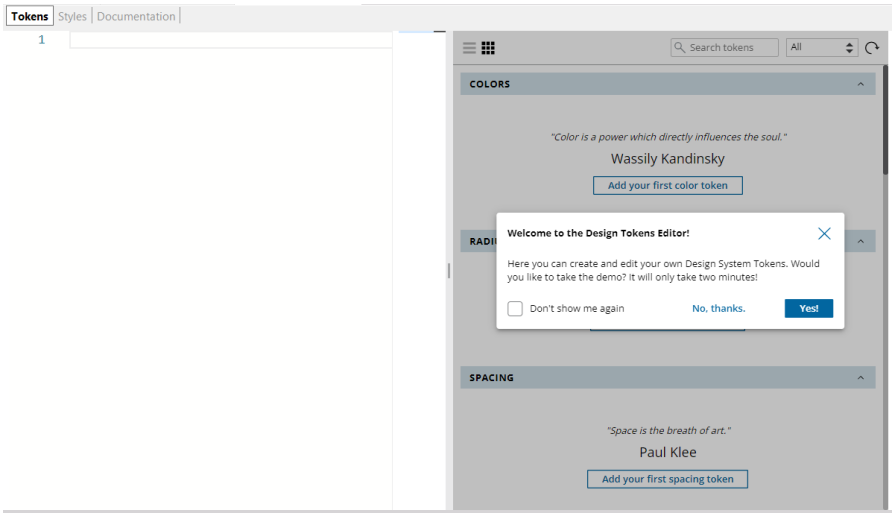
Below the canvas shows a table containing another table on the left with 2 textblocks, one of the them with the subtitle "The new age of exploration", and underneath it the action title "Contact us today". To the right is a textblock with a descriptive text about the travel agency.

The title "Get ready to Explore" will be red, with class H1_Red, the subtitle "The new age of Exploration" will have class H2, the link "Contact Us Today" will be of the MainActionText type, and the descriptive text will be NormalText.

All these definitions will be done on a Design System Object.

Creating a Design System Object



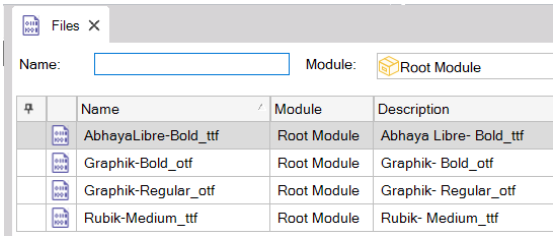


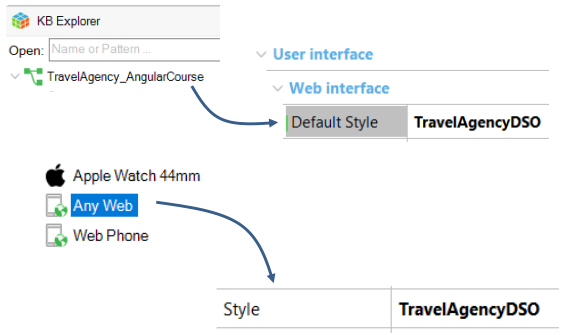
Now you will create a Design System Object called TravelAgencyDSO.

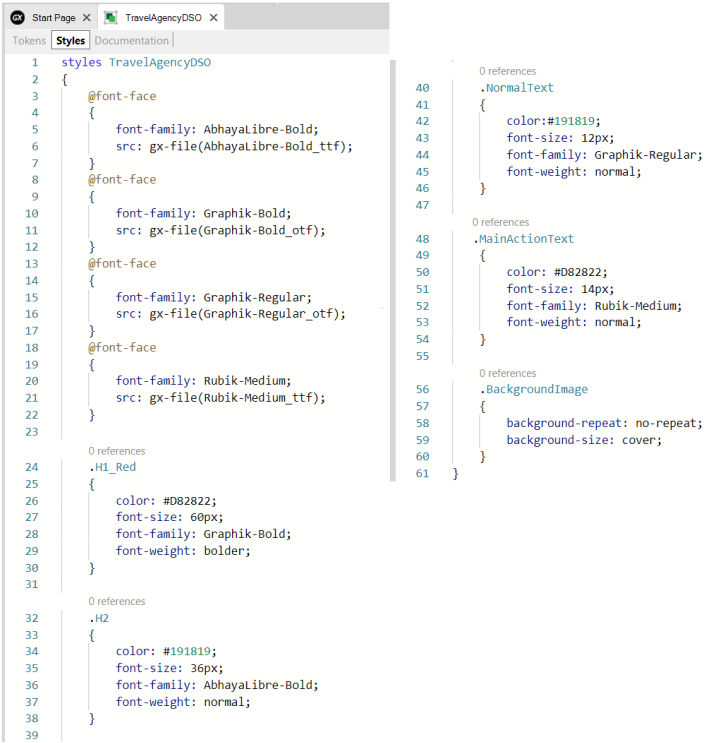
You will see that it is positioned on a tab called Tokens, where it welcomes you to the Design Tokens Editor and invites you to view a help demo.

You will also see that a Styles tab is created, where you will define the classes for the main title, the subtitle, and so on, all seen before.

Defining style classes







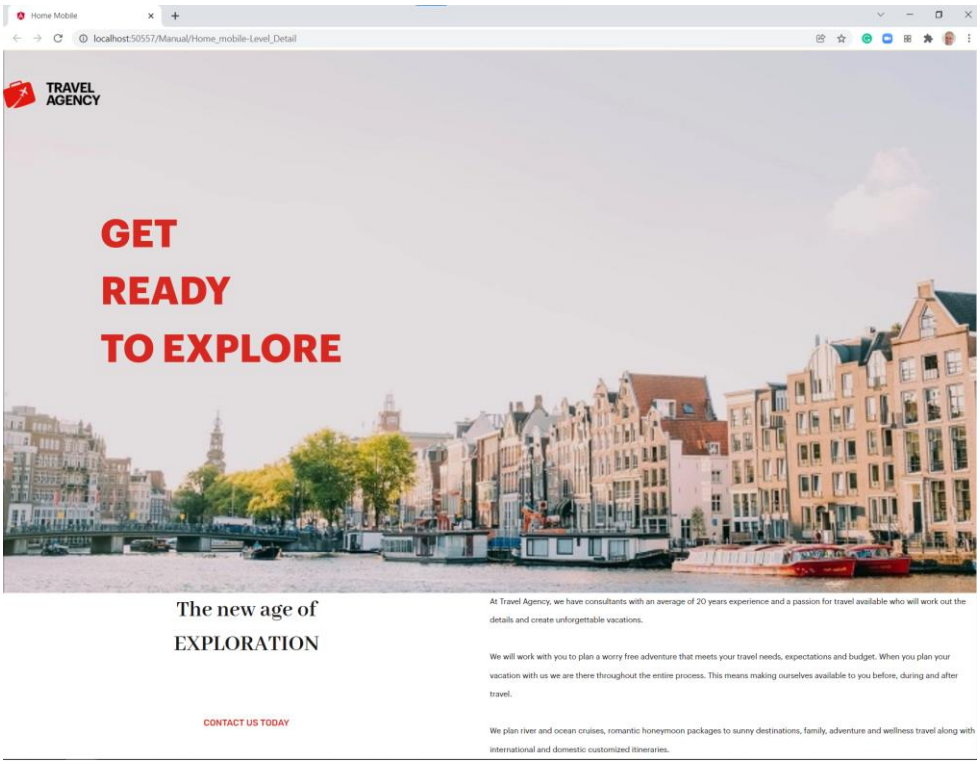
The first thing is to write the structure that will contain the styles.
You will create the class H1_Red that will be used to define the main title.

You must provide the style features for the class. For example, for the color, use a special red, written as hexadecimal. Then define the font size as 60 pixels, and Graphik-bold, indicating an intense font-weight, that is bolder. These properties are identical to the CSS properties, but they are not the CSS properties since some are specific of GeneXus and not defined in this standard. One could think that, when the size is defined in pixels, that will be a fixed value for a platform… however, these pixels are automatically converted into dips. In the case of Angular, it will in fact be 60 actual pixels because 1 dip is 1 pixel.

With a file from designers made in the Sketch tool, you could inspect the file and obtain the exact properties that you should use in your Design System Object.

When the text font is not a standard font, you must load it as a file and then declare it in the DSO. So, add the fonts you have in the file by writing font-family : AbhayaLibre-Bold, semi-colon, and src: gx-file, and in parenthesis indicate the name of the font file: AbhayaLibre-Bold_ttf, and end with a semi-colon.
And then do the same for fonts Graphic-Bold, Graphic-Regular, and Rubik-Medium.
Now you must define the rest of the classes you will need: H2, Normal Text, and MainActionText.
And last, for the background image you will need to set up some properties. To that end, create a class .BackgroundImage and set the background-repeat property with value: no-repeat, and background-size with value: cover.

Now, to base your app's design on the Design System Object created, go to the properties of the KB version, and in the Default Style property assign the design system TravelAgencyDSO.
Then, in the Platform node, since this design will be used in your app generated in Angular, in the platform Any Web, assign the Style property in the DSO created. And execute the panel to view what was done.



With the application running, you will note that all styles have been applied correctly, and each text has the style desired.

Defining Tokens

Tokens | Styles * | Documentation

```
tokens TravelAgencyDSO {

    #colors
    {
        OnSurface: #191819;
        Surface: #FFFFFF;
        Highlight: #D82822;
    }

    // #fontSizes
    // {
    //  MainTitle: 60px;
    //      SubTitle: 36px;
    //      NormalText: 12px;
    //      LinkText: 14px;
    // }
}
```

Start Page | TravelAgencyDSO

Tokens | Styles | Documentation

```
styles TravelAgencyDSO
{
    @font-face
    {
        font-family: AbhayaLibre-Bold;
        src: gx-file(AbhayaLibre-Bold_ttf);
    }
    @font-face
    {
        font-family: Graphik-Bold;
        src: gx-file(Graphik-Bold_otf);
    }
    @font-face
    {
        font-family: Graphik-Regular;
        src: gx-file(Graphik-Regular_otf);
    }
    @font-face
    {
        font-family: Rubik-Medium;
        src: gx-file(Rubik-Medium_ttf);
    }

0 references
.H1_Red
{
    color: $colors.Highlight;
    font-size: 60px;
    font-family: Graphik-Bold;
    font-weight: bolder;
}

0 references
.H2
{
    color: $colors.OnSurface;
    font-size: 36px;
    font-family: AbhayaLibre-Bold;
    font-weight: normal;
}

0 references
.NormalText
{
    color:$colors.OnSurface;
    font-size: 12px;
    font-family: Graphik-Regular;
    font-weight: normal;
}

0 references
.MainActionText
{
    color: $colors.Highlight;
    font-size: 14px;
    font-family: Rubik-Medium;
    font-weight: normal;
}

0 references
.BackgroundImage
{
    background-repeat: no-repeat;
    background-size: cover;
}
}
```

Back to the DSO, you will see certain values such as colors or fonts that could be abstracted and named. You will call Tokens the elements in this higher abstraction level.

Go to the tokens section to define a token as color constant for the text to be located over a background surface, as in the case where they were located over a background image. To do this, in the Colors section press the “Add your first color token” button and you will see that a structure opens up for you to start writing.

Call it OnSurface and copy the hexadecimal value of the color value previously defined in the style.
You can also define a color for the background, which is a type of white… for example: Surface and a Highlight color for the red texts, as the title H1_red, and for the text that performs an action.

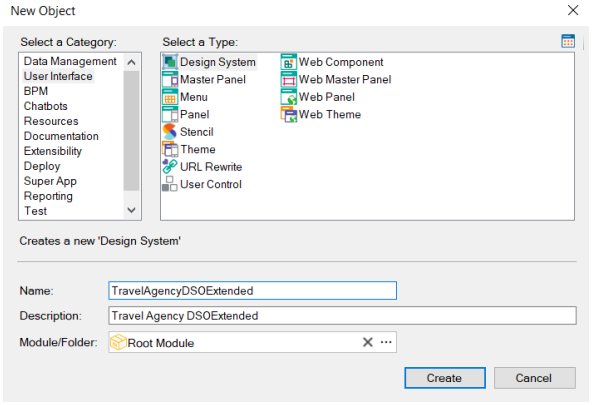
Now you can go to styles to modify the fixed value of color in the corresponding classes changing it to the corresponding token.
If you write the $ sign, you may choose the corresponding token for each case. For instance, for class H1_Red, write $ in color and select the token name colors and then select the value OnSurface.

And to the same for all colors that had fixed values.

So, when you change the token, all styles based on the same token will be changing at once, allowing more flexibility and less chances for errors.

You could also define other token types, such as for the font size. If you filter here by category and select fontSizes you will see that the token is added for you to start writing. You may define a token for the font size in the main title, another one for the subtitle, and so on. And then, just like you did before, you go to styles and change the corresponding token value.

Extending the DSO with another DSO



Tokens | **Styles *** | Documentation

```
1  styles TravelAgencyDSOExtended
2  {
3      @import TravelAgencyDSO;
4
5      //      .H1_Red
6      // {
7      //      color: $colors.Highlight;
8      //      font-size: 60px;
9      //      font-family: Graphik-Bold;
10     //      font-weight: bolder;
11     // }
12
13     .H1_Blue
14     {
15         @include H1_Red;
16         color: blue;
17     }
18
```

One thing possible in a DSO is to import definitions made in another DSO.

For example, you may create a DSO called TravelAgencyDSOExtended and import the DSO built before.

To that end, you will use the @import rule, and the name of the DSO to be imported. In this case, you may reuse classes or tokens defined in the DSO that is imported.

You could also overwrite the main DSO, a property of an imported token or class, and in that case, the one taken will be the one defined in the target DSO. Or you may inherit from a class of the DSO imported and create another based on the previous one.

The @include rule is used here with the name of the class to be overwritten, and you will only add the property that will be changed, while the rest remains with the value of the imported DSO.

To execute the app on a different platform, like a mobile device, you may import the DSO and change only the things that must be adapted to the new platform, reusing all the rest.

To learn more about this topic visit:

https://wiki.genexus.com/commwiki/servlet/wiki?47375

This video sums up how to create and modify a Design System Object, and how to extend it by importing definitions of another DSO.

Another way possible, which will be shown in another video, is to import a full design made by designers, from Sketch.

To learn more about the Design System Object, visit the wiki page:
https://wiki.genexus.com/commwiki/servlet/wiki?47375

training.genexus.com
wiki.genexus.com
training.genexus.com/certifications