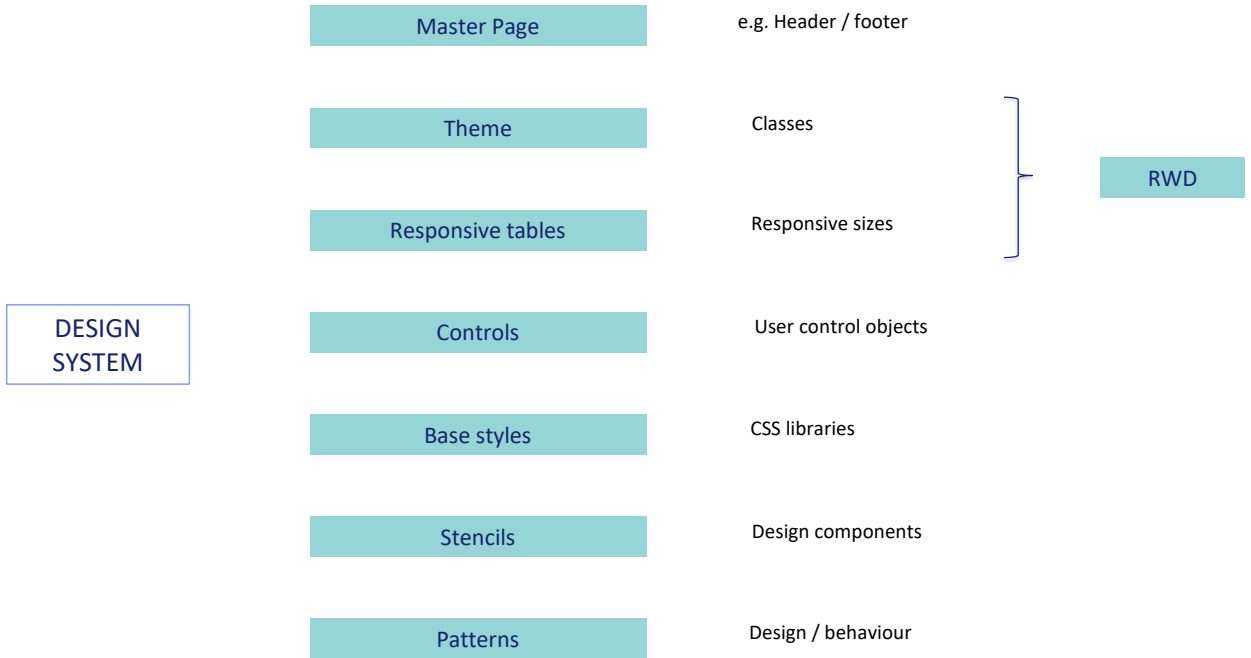


# Design System in GeneXus

Default

*GeneXus™*



In this video, we will superficially cover each of the GeneXus elements involved in the implementation of a Design System for the application.

## Back office

Attraction Without Parameters

Id: 3

Name: Eiffel Tower

Country Id: 2

Country Name: France

Category Id: 2

Category Name: Monument

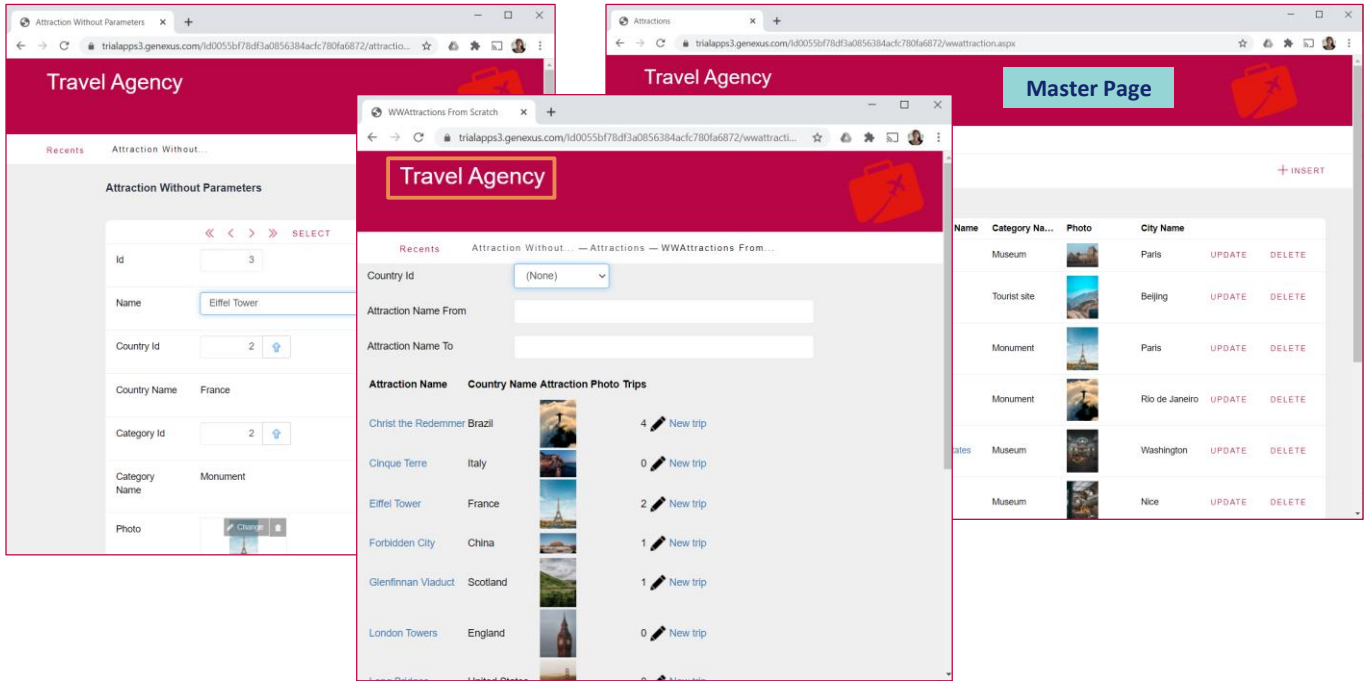
Photo: Change

Attractions

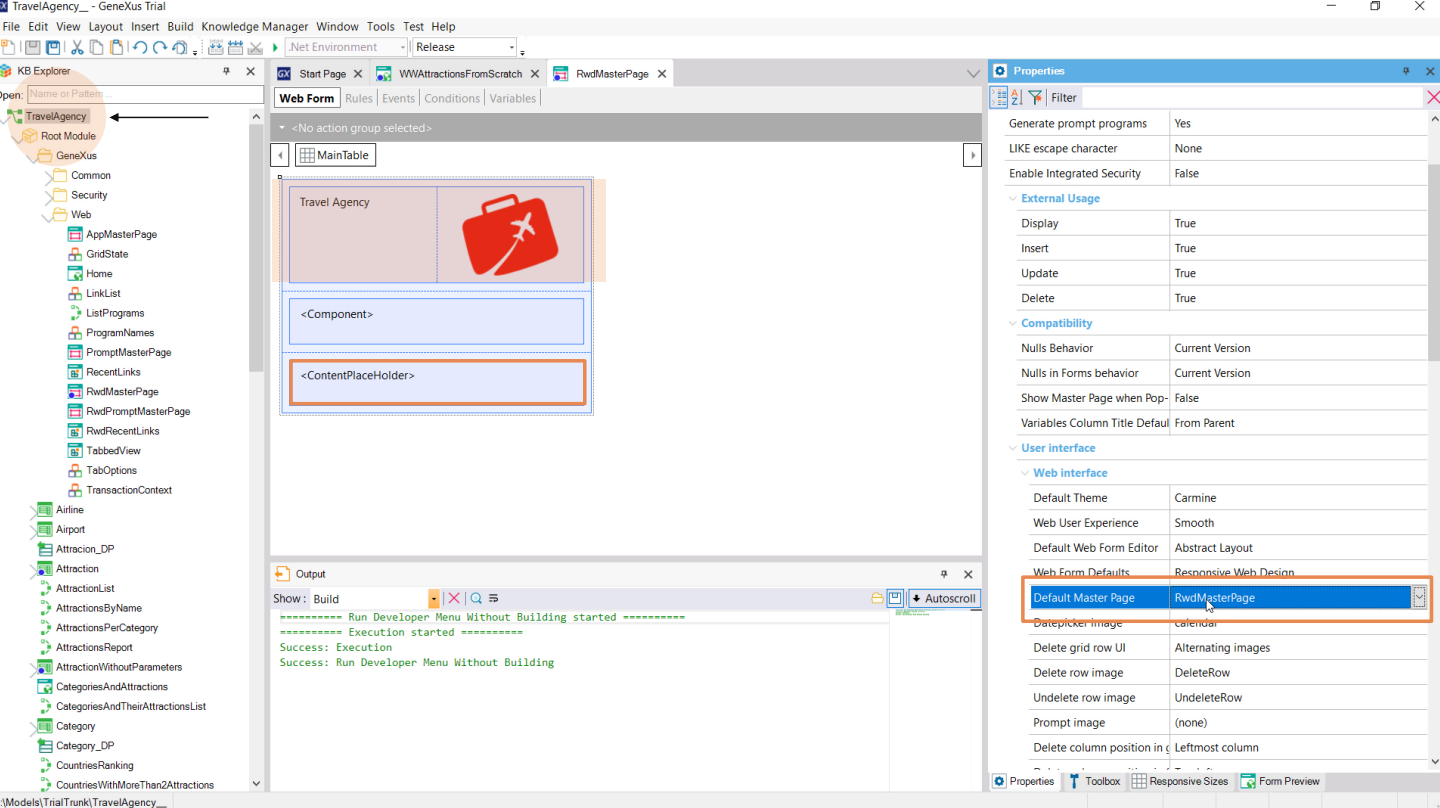
Attractions

Id	Name	Country Name	Category Name	Photo	City Name	UPDATE	DELETE
1	Louvre Museum	France	Museum		Paris	UPDATE	DELETE
2	The Great Wall	China	Tourist site		Beijing	UPDATE	DELETE
3	Eiffel Tower	France	Monument		Paris	UPDATE	DELETE
4	Christ the Redeemer	Brazil	Monument		Rio de Janeiro	UPDATE	DELETE
5	Smithsonian Institute	United States	Museum		Washington	UPDATE	DELETE
6	Matisse Museum	France	Museum		Nice	UPDATE	DELETE

If we pay attention to the transactions we have been using, and to the panels created by the Work With pattern, we see that there are elements that give consistency and coherence, without us having worried about achieving this. This means that GeneXus is already doing something for us in terms of a predetermined Design System, even if it is elementary.

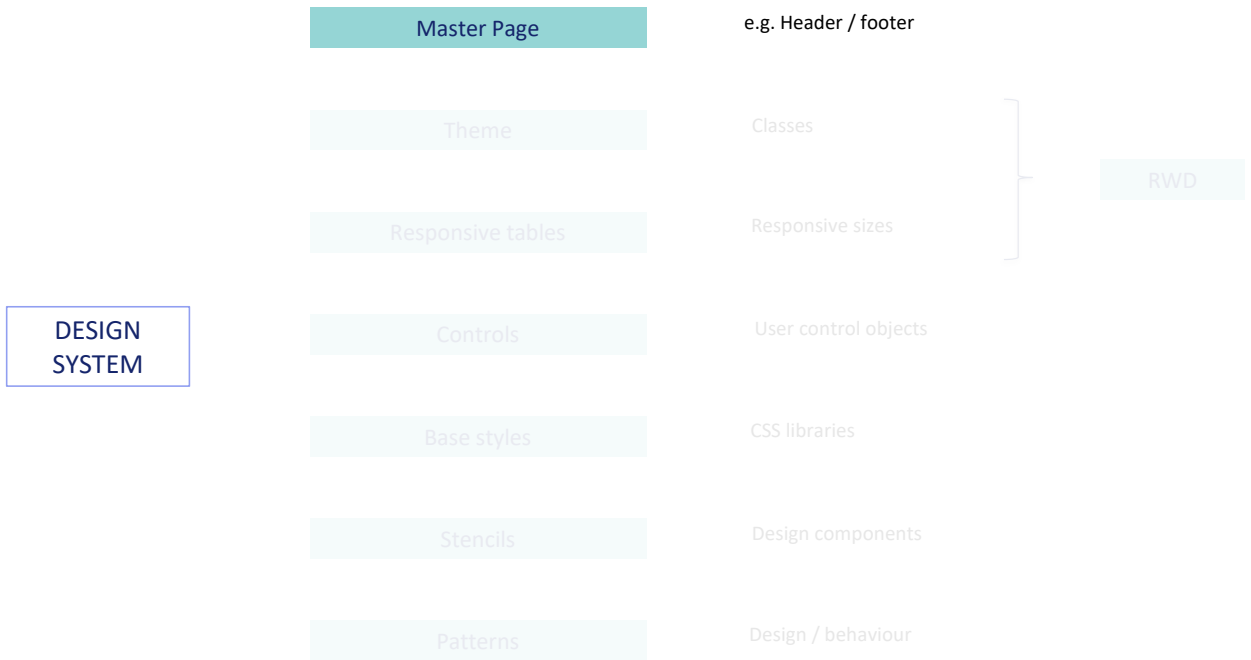


For example, we see that all screens (even the ones made from scratch, like this web panel) have a common header. In this case we have customized it, changing the default text to "Travel Agency" and replacing the previous image with this other one.

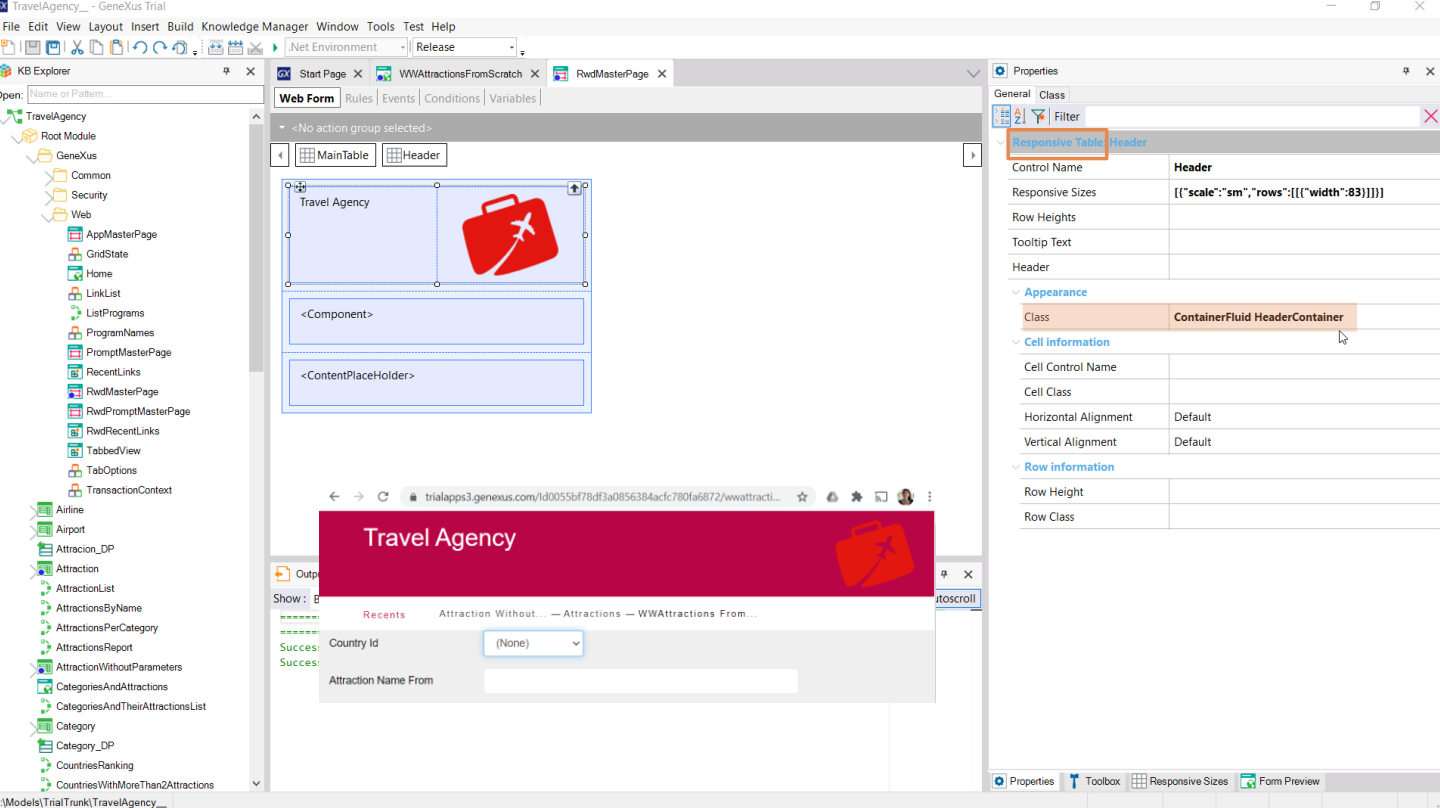


The way to achieve it is with the web Master Page object (Here we can see the changes made). Remember that later on the various screens of the application will be loaded into this control, the content placeholder.

By default, the KB version comes with this predefined Master Page, which means that it will be applied to all web objects that are created, as we had seen.

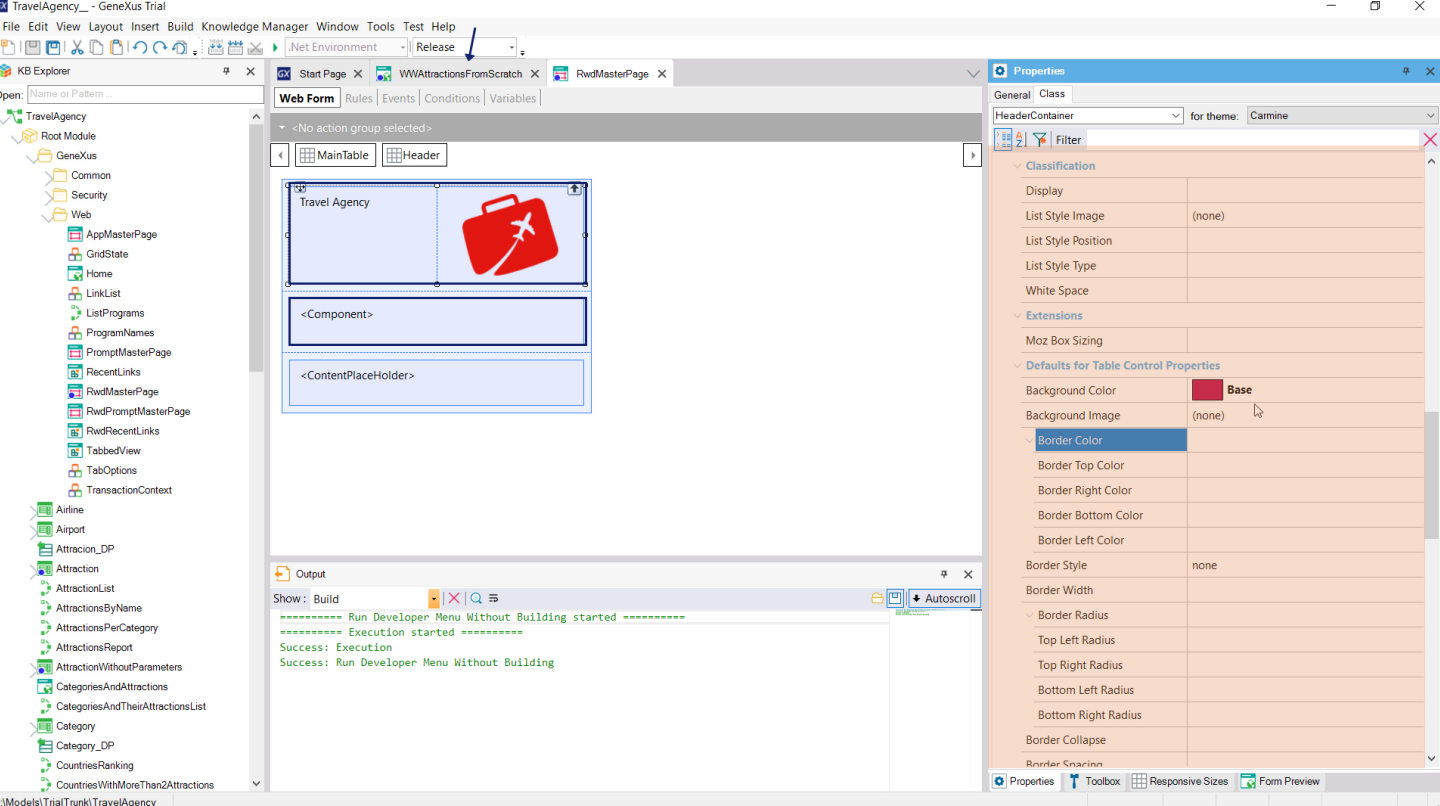


So, on one hand GeneXus offers an object to centralize what will be the common information we want the pages to share, such as a header or a footer.



In addition, we can see that the base color, the one that stands out, is a type of red, which is used not only for the header, but for the buttons and other actions that are given to the user. Where is this configured?

We know this table has a red background at runtime, but we're not seeing it! Note that the control (in this case of the responsive table type) has a pair of defined classes, not coincidentally under the group "Appearance".

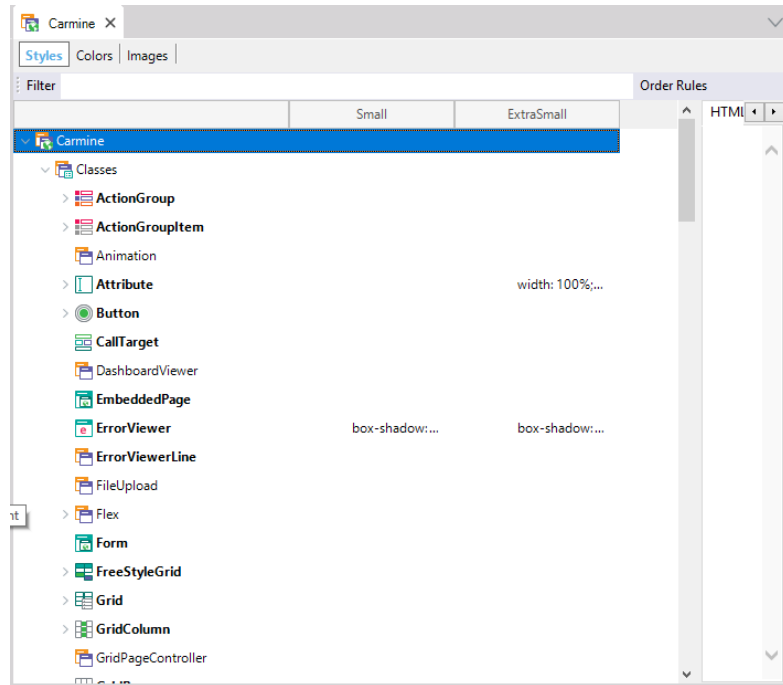


If you select the **Class tab**, you'll see that the properties of the second one of the classes associated with that table are being edited. And there we see defined, among other things, the red background color we had seen!

Classes are useful because they allow you to centralize the design of that type of control (in this case, table). Then, in this case, other tables located in the same or in other forms can share the same definition, so if you want to change, for example, the background color from red to blue for those controls, it is not necessary to do it one by one in each control -that is to say, in each table- but it is done directly in the class and that already affects all the controls that have it associated.

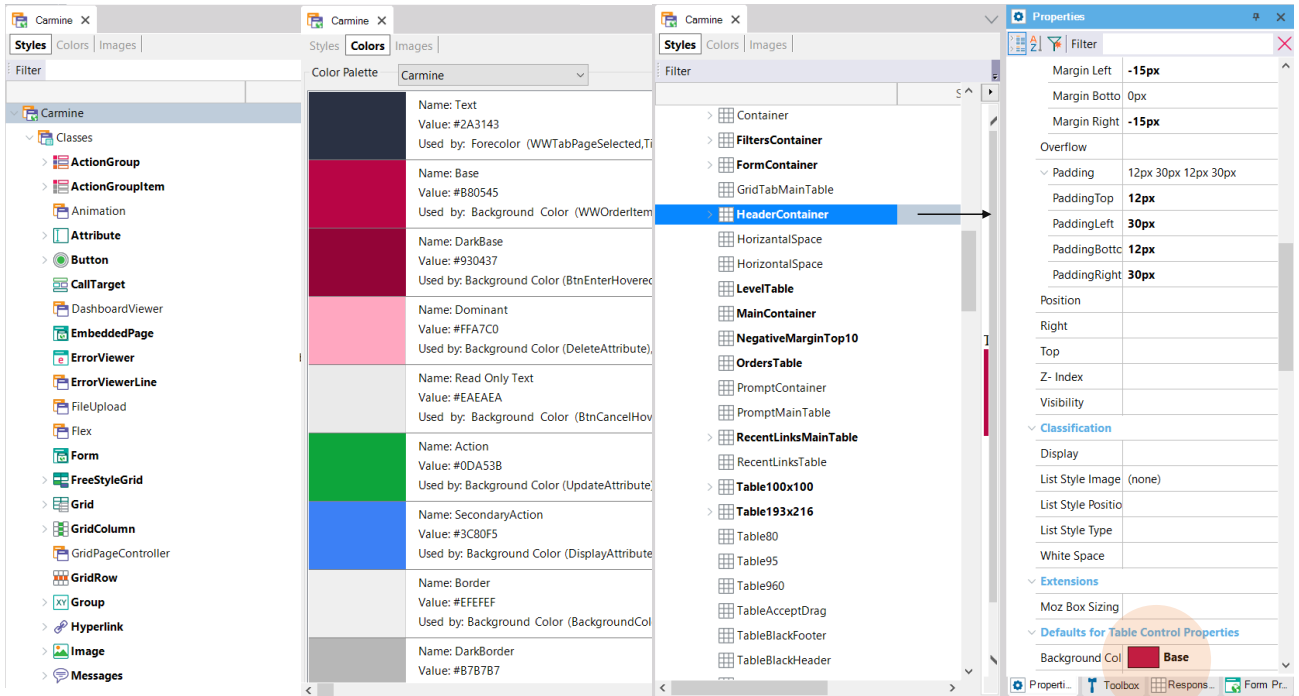


## Theme



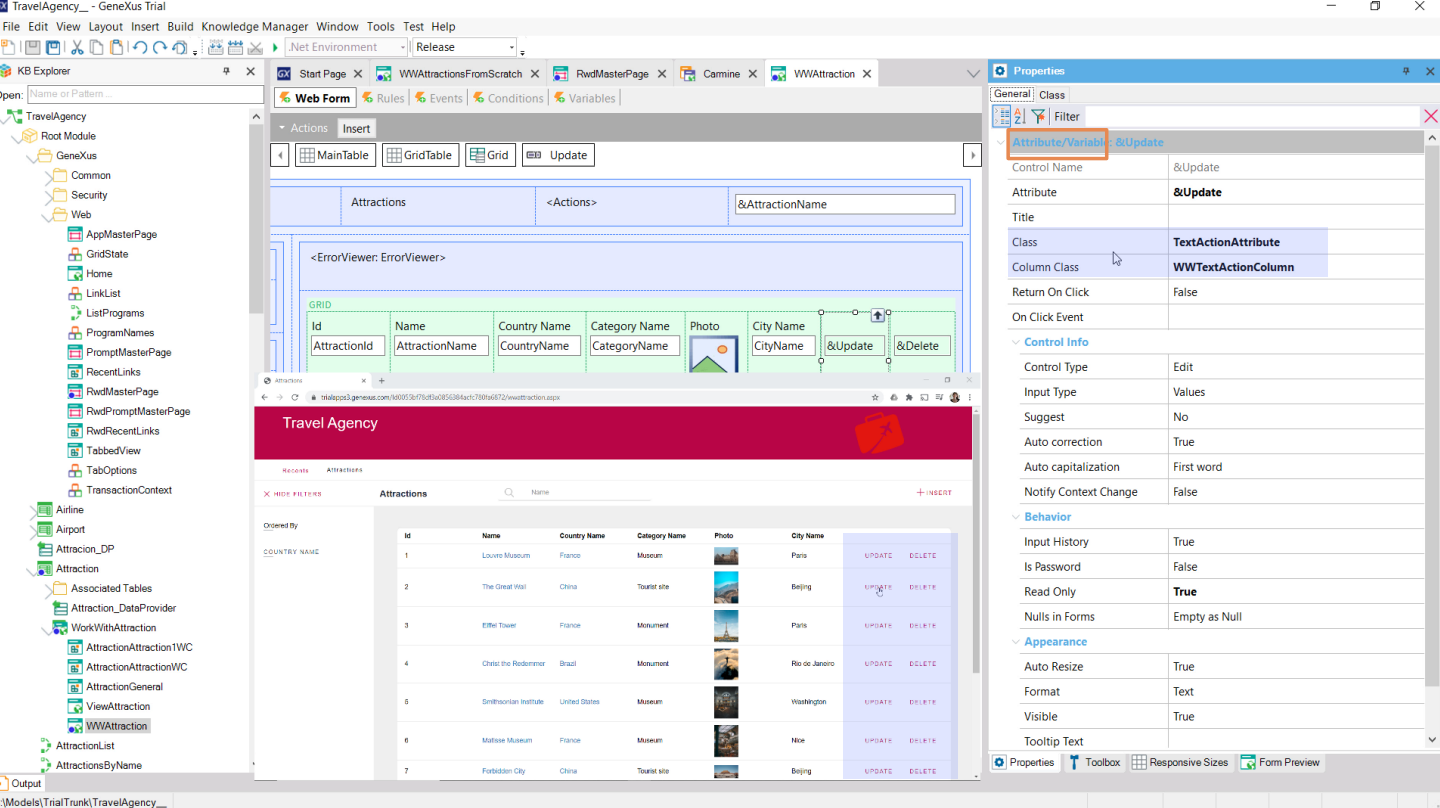
Where are all classes configured? In the Theme object!

Note that just as there was a predefined Master Page, there is also a predefined theme called Carmine, that we have seen associated several times with our web panels.



Note that when opening it there are classes grouped by type of control. This is where everything is centralized. Also, note that the Colors tab shows the color palette of the same name... This is where the Base color is configured.

If we look for subclasses of the Table class, there we find HeaderContainer, one of the two classes that had the Master Page table associated with it. What we change on one side will be changed on the other. That was a shortcut to this place.



If you look at the Update or Delete options of the Work with pattern which, as we know, have the base color at runtime, you'll see that the type of control is Attribute/Variable. Also, that its assigned class is this predefined by the pattern. We can edit its properties from here knowing that what we modify here will be modified at central level, in the Theme object, so that any other attribute/variable control that has this class will also be modified.

In short, every control in a form has individual properties, which can be configured for it alone... And properties that will come from its assigned class, which are those that involve design aspects, and that will be shared by many controls of the same type.

We can already note that in this particular case in which the control is, in addition, a grid column, another class is displayed to be associated with it, which is the class that rules the design and behavior of the column.

Attraction Without Parameters

Travel Agency

Attraction Without Parameters

Responsive table

Id: 3

Name: Eiffel Tower

Country Id: 2

Country Name: France

Category Id: 2

Category Name: Monument

Photo:

Attractions

Travel Agency

Grid

Attractions

Ordered By: COUNTRY NAME

Id	Name	Country Name	Category Na...	Photo	City Name	UPDATE	DELETE
1	Louvre Museum	France	Museum		Paris	UPDATE	DELETE
2	The Great Wall	China	Tourist site		Beijing	UPDATE	DELETE
3	Eiffel Tower	France	Monument		Paris	UPDATE	DELETE
4	Christ the Redeemer	Brazil	Monument		Rio de Janeiro	UPDATE	DELETE
5	Smithsonian Institute	United States	Museum		Washington	UPDATE	DELETE
6	Matisse Museum	France	Museum		Nice	UPDATE	DELETE

And if we look again at the predefined design of the transaction and the Work With [panel](#), we see that the most relevant data is shown in white on grey, with rounded edges.

Responsive table

The image displays the GeneXus IDE interface for a web form. On the left, a table is shown with the following fields: Attraction Without Parameters, <ErrorViewer: ErrorViewer>, <Toolbar>, Id (AttractionId), Name (AttractionName), Country Id (CountryId), Country Name (CountryName), Category Id (CategoryId), Category Name (CategoryName), Photo (with a landscape image icon), City Id (CityId), and City Name (CityName). The table is highlighted with a blue border.

The middle pane shows the Properties window for the selected 'FormContainer' class. The 'Responsive Table: FormContainer' section is expanded, showing the following properties:

- Control Name: **FormContainer**
- Responsive Sizes: `[[{"scale": "sm", "rows": [{"width": ...`
- Row Heights: `.....`
- Header: `.....`
- Appearance:
  - Class: **FormContainer**
- Cell Information:
  - Cell Control Name: `.....`
  - Cell Class: `.....`
  - Horizontal Alignment: Default
  - Vertical Alignment: Default
- Row information:
  - Row Height: `.....`
  - Row Class: `.....`

The right pane shows the Properties window for the 'FormContainer' class, with the 'Background Color' property set to **white**. Other properties include:

- Background Image: (none)
- Border Color: `.....`
- Border Top Color: `.....`
- Border Right Color: `.....`
- Border Bottom Color: `.....`
- Border Left Color: `.....`
- Border Style: `.....`
- Border Width: `.....`
- Border Radius: 8px
- Top Left Radius: **8px**
- Top Right Radius: **8px**
- Bottom Left Radius: **8px**
- Bottom Right Radius: **8px**
- Border Collapse: `.....`
- Border Spacing: `.....`
- Lines Font: `_ _ 400 14px _ Arial`

This is because the table where this data is found in the transaction was assigned this class... which specifies white as the background color and a radius for the four edges...

The screenshot displays the GeneXus IDE interface for a web form titled 'WWAttraction'. The design view shows a form with a search bar for '&AttractionName', a filter section, and a grid. The grid has columns for 'Id', 'Name', 'Country Name', 'Category Name', 'Photo', 'City Name', and actions '&Update' and '&Delete'. A teal box labeled 'Work With Grid' is overlaid on the design view.

The Properties panel on the right shows the 'Grid' control with the following settings:

- Control Name: **Grid**
- Collection: (empty)
- Base Tm: (empty)
- Order: **AttractionName when ...**
- Conditions: (empty)
- Unique: (empty)
- Save State: **True**
- Data Selector: (none)
- Appearance:
  - Class: **WorkWith**
  - Custom Render: (empty)
  - Empty Grid Text: (empty)
  - Auto Resize: **True**
  - Width: (empty)
  - Height: (empty)
  - Rows: **10**
  - Header: (empty)
  - Tooltip Text: (empty)
- PagingMode:
  - Paging: **One page at a time**
- Layout:
  - Cell Padding: **1**

The right-hand Properties panel shows the 'WorkWith' class settings:

- Background Attachment: (empty)
- Background Color: **white**
- Background Position: (empty)
- Background Repeat: (empty)
- Opacity: (empty)
- Box: (empty)
- Classification: (empty)
- Extensions: (empty)
- Defaults for Grid Control Properties:
  - Background Color Style: **None**
  - Background Image: (none)
  - Forecolor: **Grid Title**
  - Border Color: (empty)
  - Border Style: **none**
  - Border Width: (empty)
  - Border Radius: **8px**
  - Top Left Radius: **8px**
  - Top Right Radius: **8px**
  - Bottom Left Radius: **8px**
  - Bottom Right Radius: **8px**
  - Border Collapse: (empty)
  - Border Spacing: (empty)

...and the Work With grid was assigned this other class, which also specifies white as the background color, and the same values as the other for the radius.

The image displays the GeneXus IDE interface. On the left, the 'Web Form' design view shows a form with two input fields for 'Attraction Name From' and 'Attraction Name To'. Below them is a grid control named 'Grid1' with columns: 'Attraction Id', 'Attraction Name', 'Attraction Photo', 'Trips', '&update2', and '&newTrip'. A 'Total Trips' label is also present. A green box labeled 'Default Grid' points to the grid design.

The 'Properties' window on the right shows the configuration for 'Grid: Grid1':

Property	Value
Control Name	Grid1
Collection	
Base Trm	Attraction
Order	CountryId, AttractionName when not &Coun...
Conditions	CountryId = &CountryId when not &Country...
Unique	
Save State	False
Data Selector	(none)

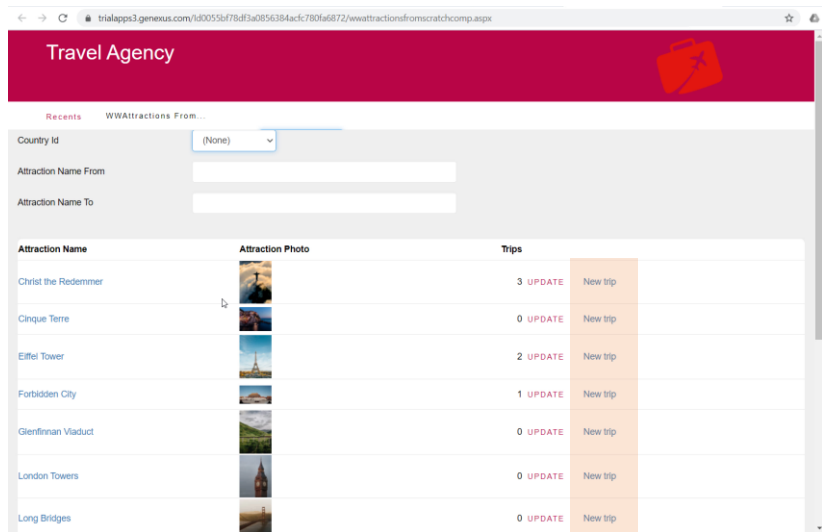
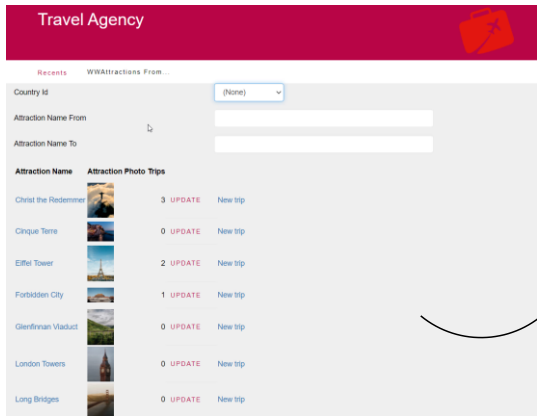
The 'Appearance' section shows:

Property	Value
Class	Grid
Custom Render	
Empty Grid Text	
Auto Resize	True
Width	
Height	
Rows	0

The runtime preview on the right shows a 'Travel Agency' interface with a grid of attraction data. The grid has columns for 'Attraction Name', 'Attraction Photo', and 'Trips'. The data includes attractions like 'Christ the Redeemer', 'Cinque Terre', 'Eiffel Tower', 'Forbidden City', 'Glenfiddich Vodka', 'London Towers', and 'Long Bridges'.

On the other hand, if we look at the grid of the web panel we created, we can see that its class is called Grid, which doesn't have any of this configured... And this is how it looks at runtime, with all the columns next to each other, with the default gray background.

Class: WorkWith

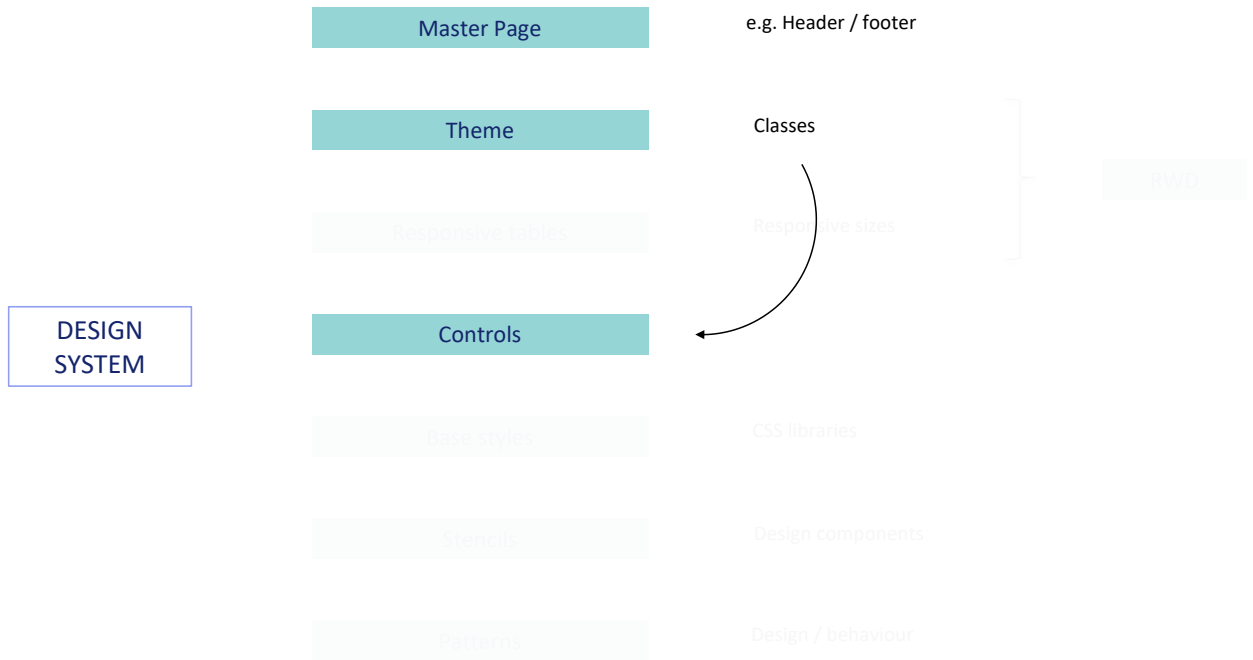


Class	TextActionAttribute
Column Class	
Return On Click	False

Note what happens if we change its class for the same one of the WorkWith grid...

And to maintain the consistency of the Design System, for the update action we had already changed the image for the text block, so that it is identical to that of the Work With, but this grid action should also look like these other ones...

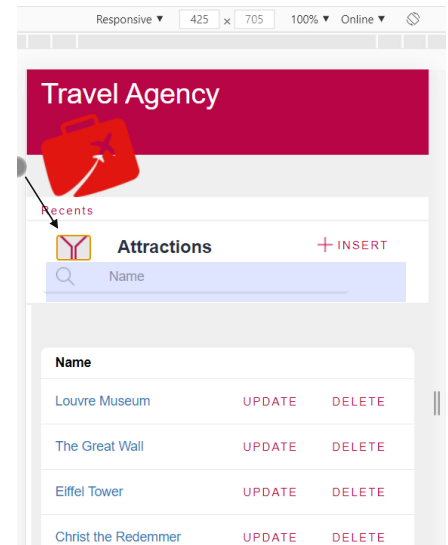
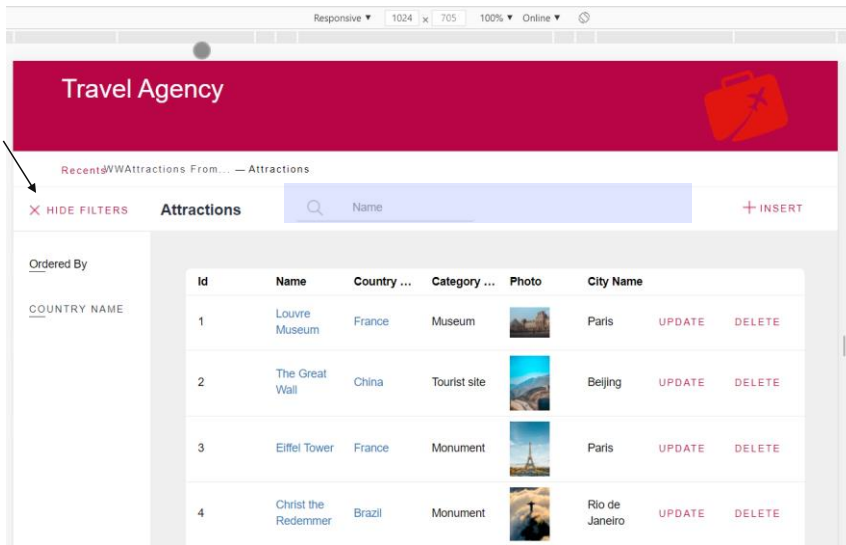




From what we've seen, we can infer that a very high percentage of the Design System is specified in the Theme object, through its classes (the predefined ones and the ones we add). These classes are the ones assigned to the controls of the form, thus making it possible to abstract the design and establish a certain logic, so as to unify it according to the role played by each control.

Therefore, a crucial aspect of design will be to identify the classes that we'll need for the controls, and apply them.

## Responsive Web Design (RWD)



On the other hand, returning to the predefined settings (remember that some browsers offer a display mode –in this case by pressing F12– that allows you to vary the size of the screen to see how it would look)...

We can see that if we reduce the screen of the Attractions Work With enough to match a phone screen, the filters appear differently. And, for example, the field for the user to enter a filter by attraction name appears below the title and not on the right. In addition, the recent links appear as a drop-down menu, instead of being expanded. And the image of the master page is displayed underneath and not on the side.

And with the screen width reduction, almost all columns also disappear except for the one showing the name and Update and Delete actions.

This behavior is known as "responsive," and the web design that considers it as Responsive Web Design. Today, it is not acceptable to design something that cannot be adapted to the screen size.

Part of responsiveness is established through **responsive tables**...

## Responsive tables

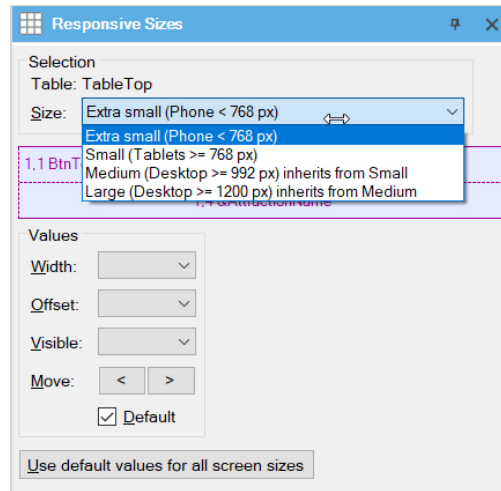
The screenshot displays the GeneXus IDE interface. The main workspace shows a web form design for a travel agency. At the top, there are filter controls: 'Hide Filters', 'Attractions', '<Actions>', and '&AttractionName'. Below this is a grid with columns for 'Name' and 'Country'. A red banner for 'Travel Agency' is overlaid on the grid. Below the banner is a 'Recents' section with a search icon and the text 'Attractions + INSERT'. At the bottom, a table lists 'Louvre Museum' with 'UPDATE' and 'DELETE' buttons.

The Properties panel on the right shows the configuration for the 'TableTop' control. The 'Responsive Sizes' property is set to `{{"scale":"xs","rows":{{"width":...`. The 'Appearance' section shows the class is 'TableTopSearch'. The 'Cell information' section shows the cell class is 'TableTopSearch'. The 'Row information' section shows the row height and row class.

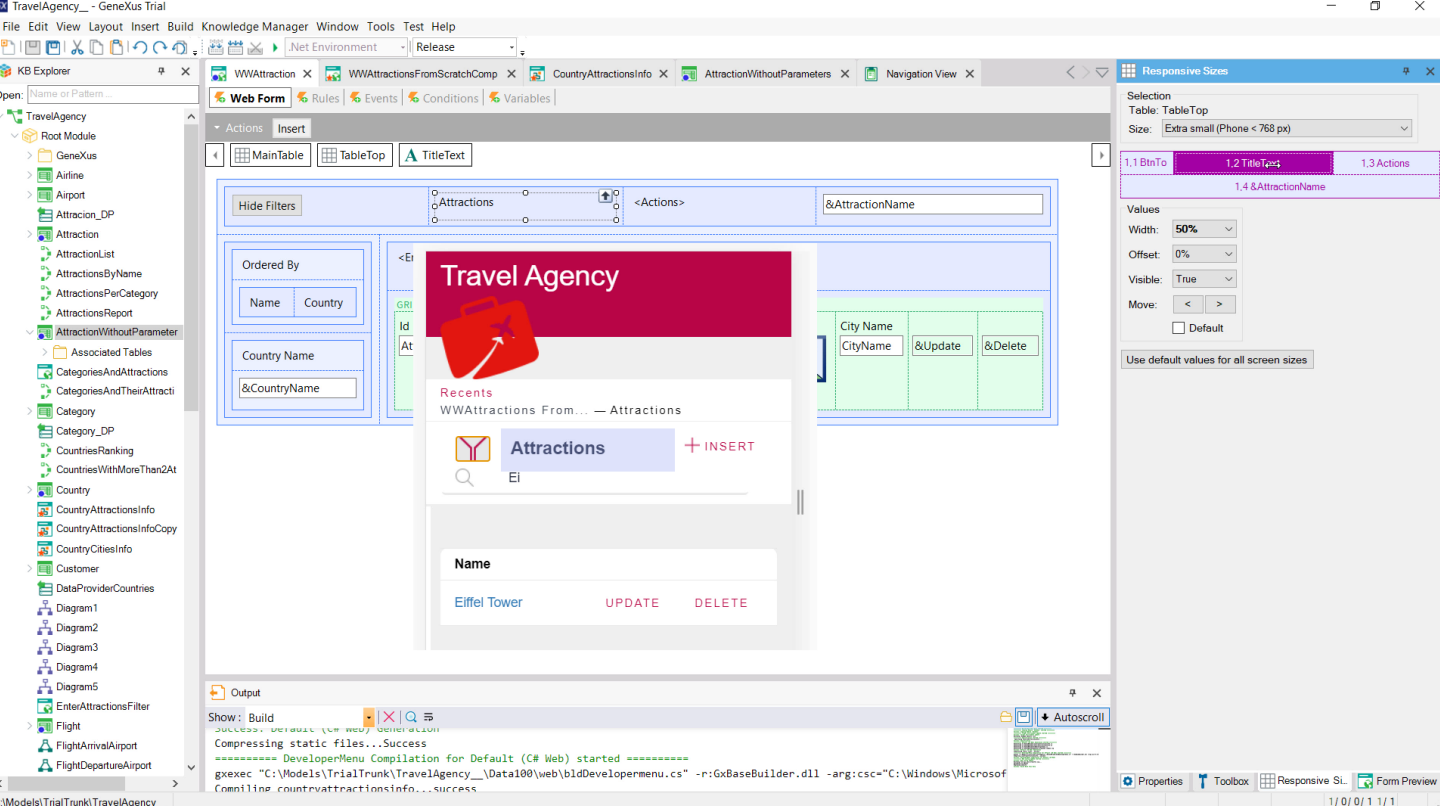
For example, this table [of the Attractions Work With created by the pattern](#), which we see is responsive, contains these four controls. The first one corresponds to the additional filters. The second one corresponds to this textblock. The third one corresponds to the Insert action, and the last one corresponds to the filter variable that we saw.

If we edit the table properties, we find this one labeled “Responsive Sizes.”

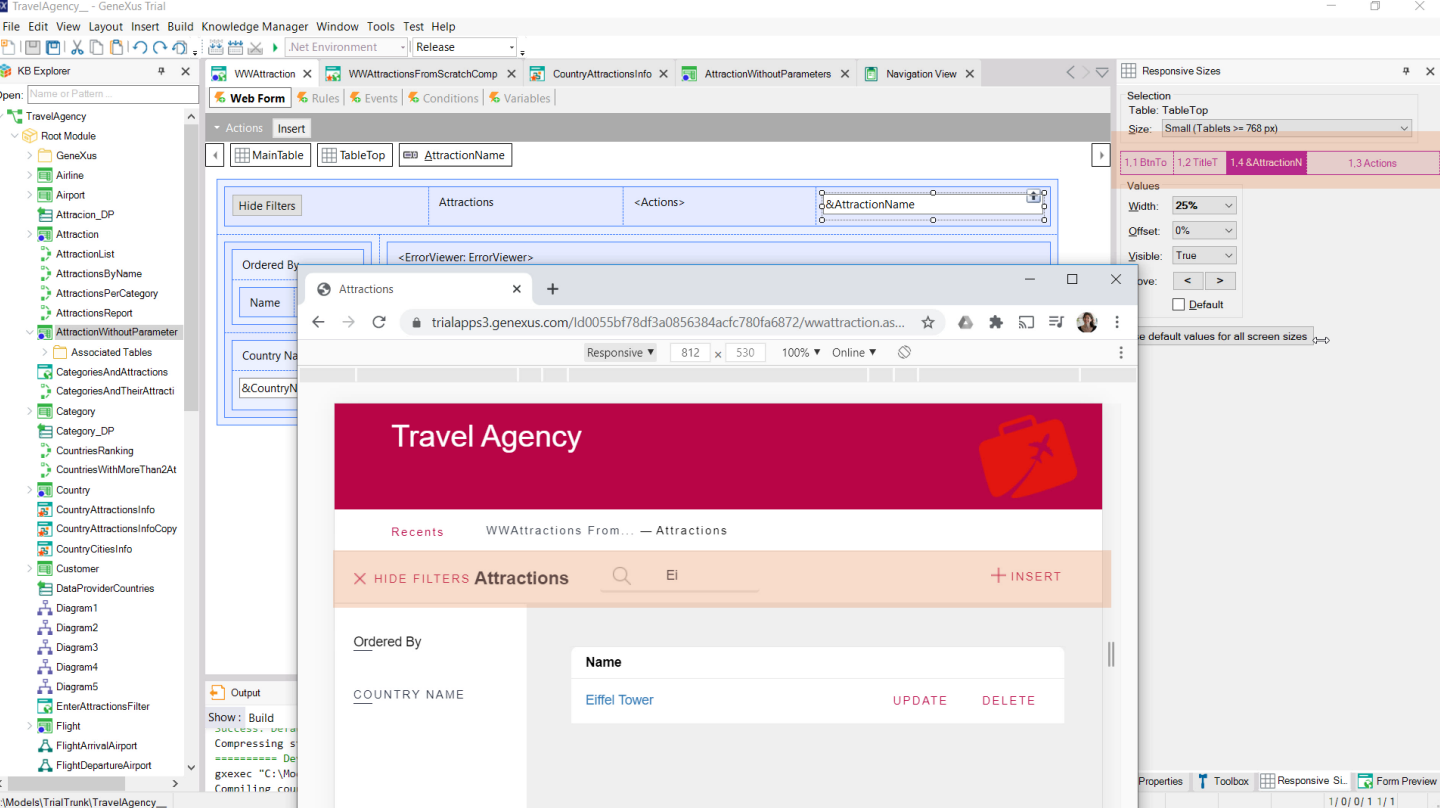
## Responsive tables



Here you establish how you want to display the table contents according to four screen sizes: **Extra small**, which corresponds to phones; **Small**, corresponding to tablets. **Medium** that corresponds to notebook or PC screens smaller than 1200 pixels; and **Large** corresponding to bigger sizes.



Here we can see that if the size is **Extra small**, the button to enable the other filters will be displayed first, taking up 17% of the screen width; then to its right will come the title, which will take up 50% of the remaining width, and finally the actions, which here will be only the Insert action, taking up the remaining 33%. The filter variable will appear underneath all of that, taking up 100%.

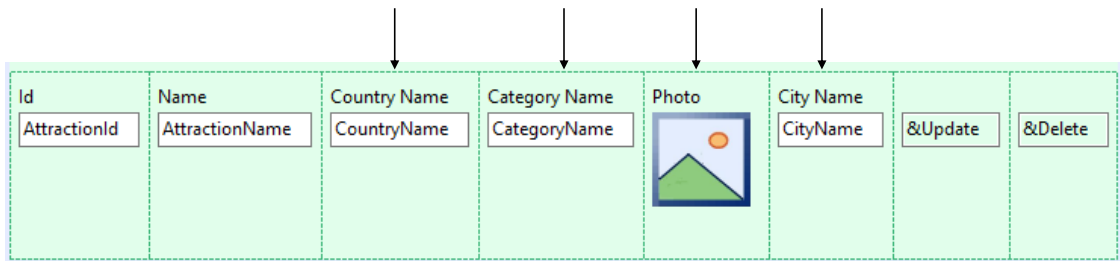



Meanwhile, if the size is **Small**, it will be all in one row, with the button, title, variable and action, in that order.

## Responsive Web Design (RWD)

- Responsive Sizes of Responsive tables
- Theme Rules: Small, Extra-small and Default

i.e.: to hide grid columns if Extra small or Small size



Id	Name	Country Name	Category Name	Photo	City Name	&Update	&Delete
AttractionId	AttractionName	CountryName	CategoryName		CityName		

We've said that an important part of responsiveness, the most general one, is achieved by making use of these tables and properties.  
But another part, a bit more specific, is achieved through the **classes**.

The screenshot displays a web form design in a development environment. The main area shows a grid table with columns: Id, Name, Country Name, Category Name, Photo, City Name, &Update, and &Delete. The 'Country Name' column is highlighted with a red circle. The Properties window on the right shows the following settings for the 'Country Name' attribute:

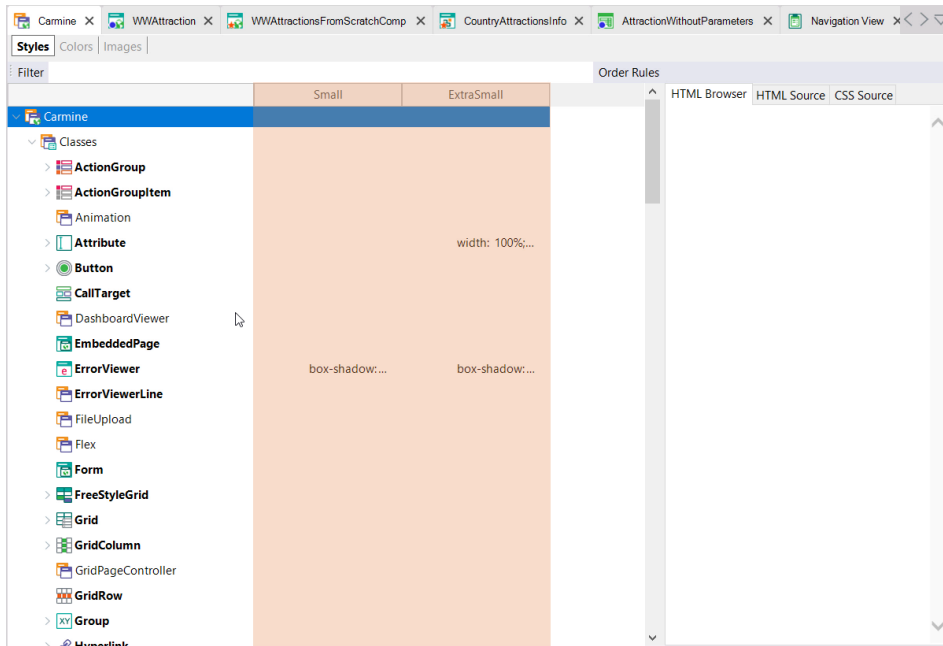
Attribute	Country Name
Title	Country Name
Class	Attribute
Column Class	WWColumn WWOptionalColumn
Return On Click	False
<b>Control Info</b>	
Control Type	Edit
Input Type	Values
Suggest	No
Auto correction	True
Auto capitalization	First word
Notify Context Cha	False
<b>Behavior</b>	
Input History	True
Is Password	False
Read Only	False
Nulls in Forms	Empty as Null
<b>Appearance</b>	
Auto Resize	True
Format	Text
Visible	True
Tooltip Text	
Invite Message	

For example, hiding grid columns depending on screen size.

Note that the class of the attraction name column is WWColumn, but that to the other columns the class WWOptionalColumn is added.



## RWD: Theme rule columns

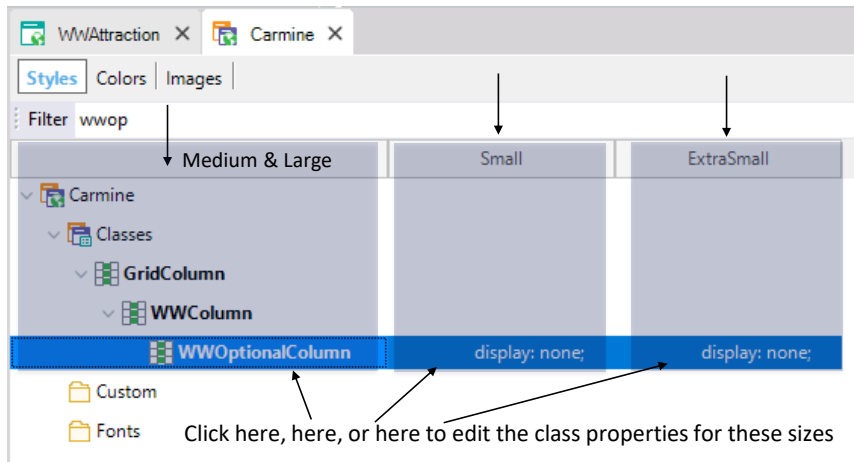


If you select the theme, you'll see these two columns that allow you to vary the values of the class properties, according to the size of the screen.

By default, only two are set: Small and ExtraSmall. But, actually, there are three, because the values in the default column correspond to Medium and Large.

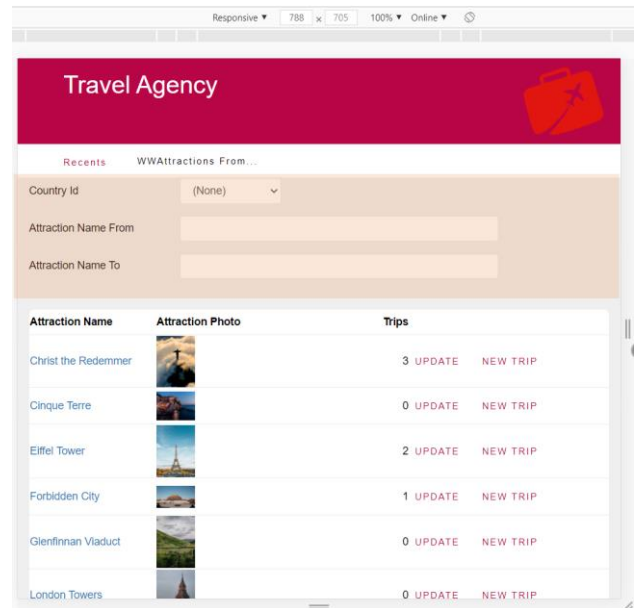
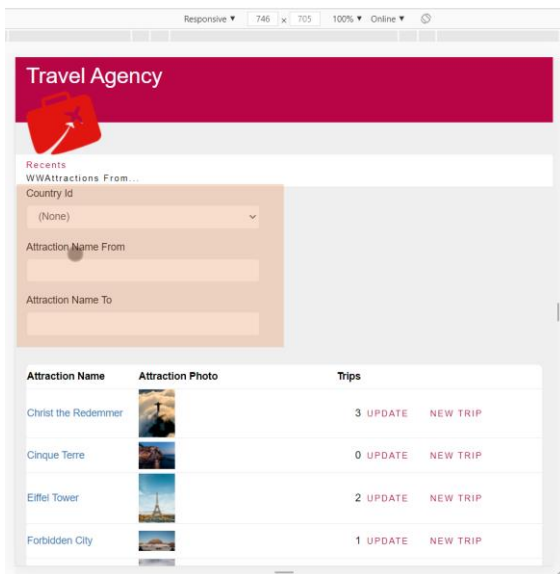
## RWD: Theme rule columns

- Theme Rules: Small, Extra-small and Default



So, even if the Display property of this class doesn't say anything, it means that it will be displayed at runtime for Medium and Large screen sizes. For Small and ExtraSmall it has the "none" value. That's why they are not displayed.

## How to improve responsiveness in our web panel



However, in the web panel we implemented from scratch we do not have this grid-responsive behavior. By decreasing the screen size we still see the same columns.

The only predefined responsiveness we have is the one that comes from the Master Page, where for Extra-small size the image is at the bottom and not on the right, and the list of recent links appears as a vertical menu; for our web panel itself, the one that involves the filter fields, which are taking up 100% of the width with their label on top, for Small sizes onwards they are on the left.

## How to hide columns when Small or Extra-small







**Travel Agency**

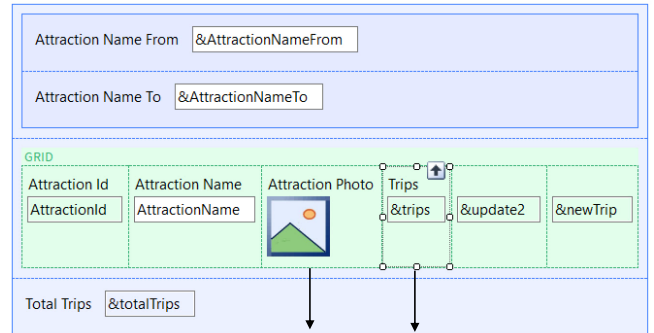
Recents WWAttractions From...

Country Id (None) ▾

Attraction Name From

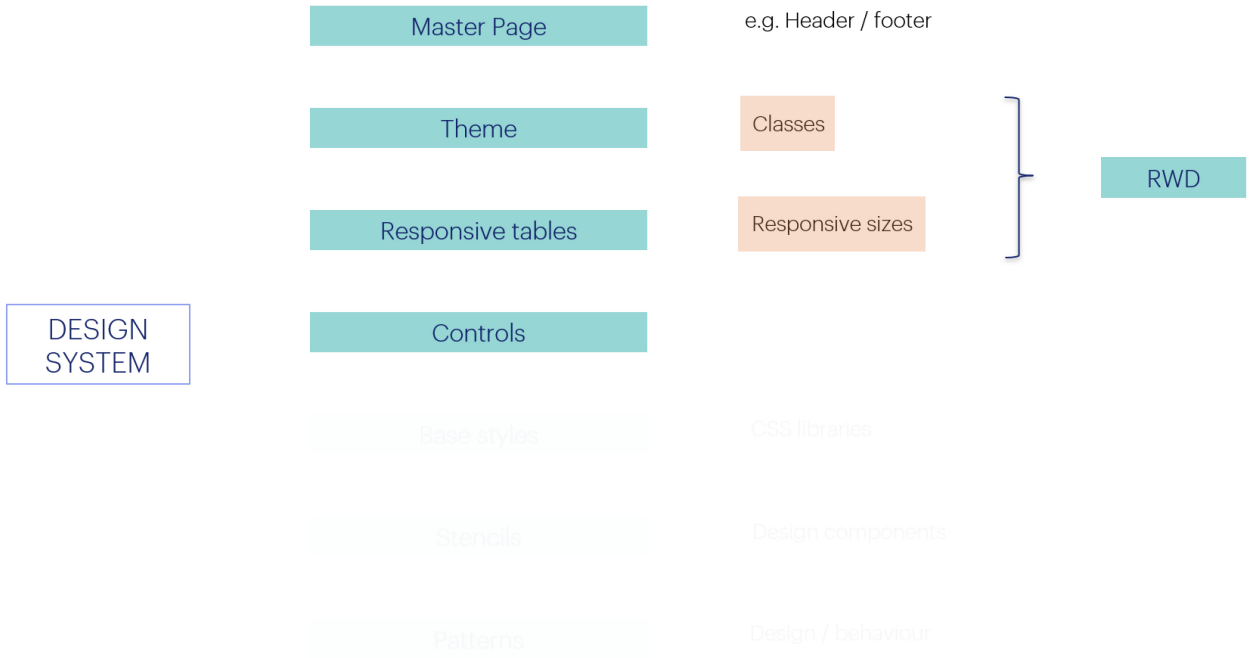
Attraction Name To

Attraction Name	Attraction Photo	Trips	
Christ the Redemmer		3	UPDATE NEW TRIP
Cinque Terre		0	UPDATE NEW TRIP
Eiffel Tower		2	UPDATE NEW TRIP
Forbidden City		1	UPDATE NEW TRIP
Glenfinnan Viaduct		0	UPDATE NEW TRIP
London Towers		0	UPDATE NEW TRIP

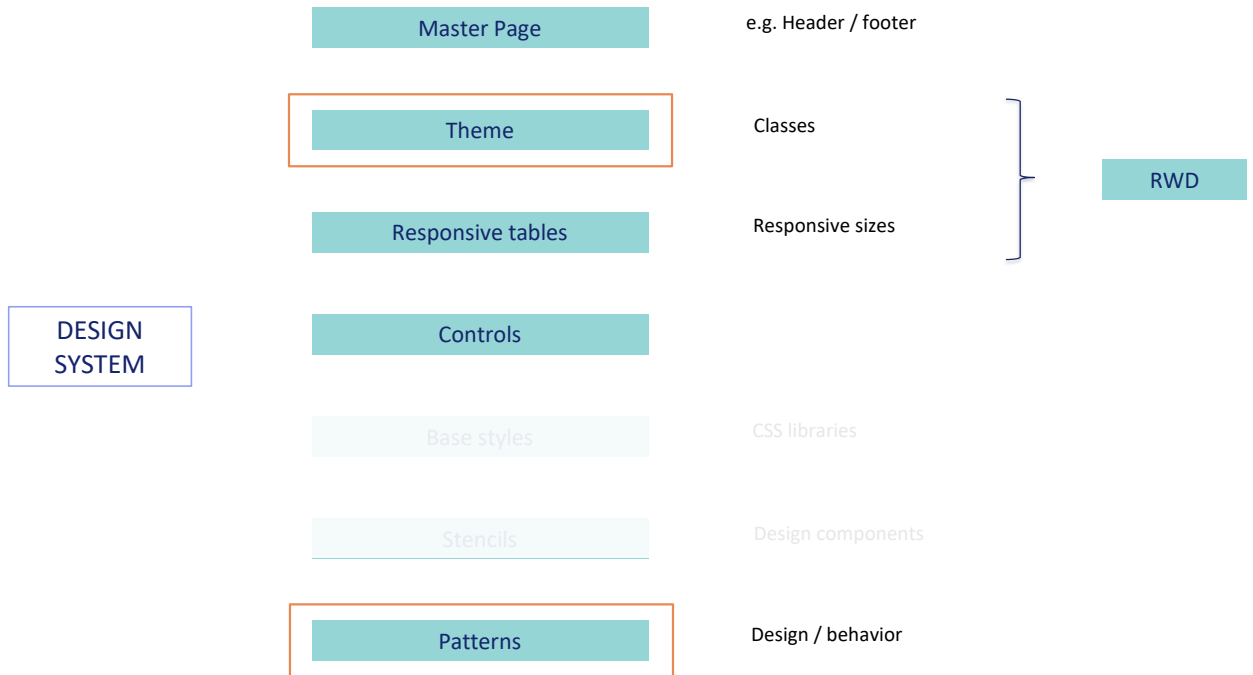


Column Class: WWOptionalColumn

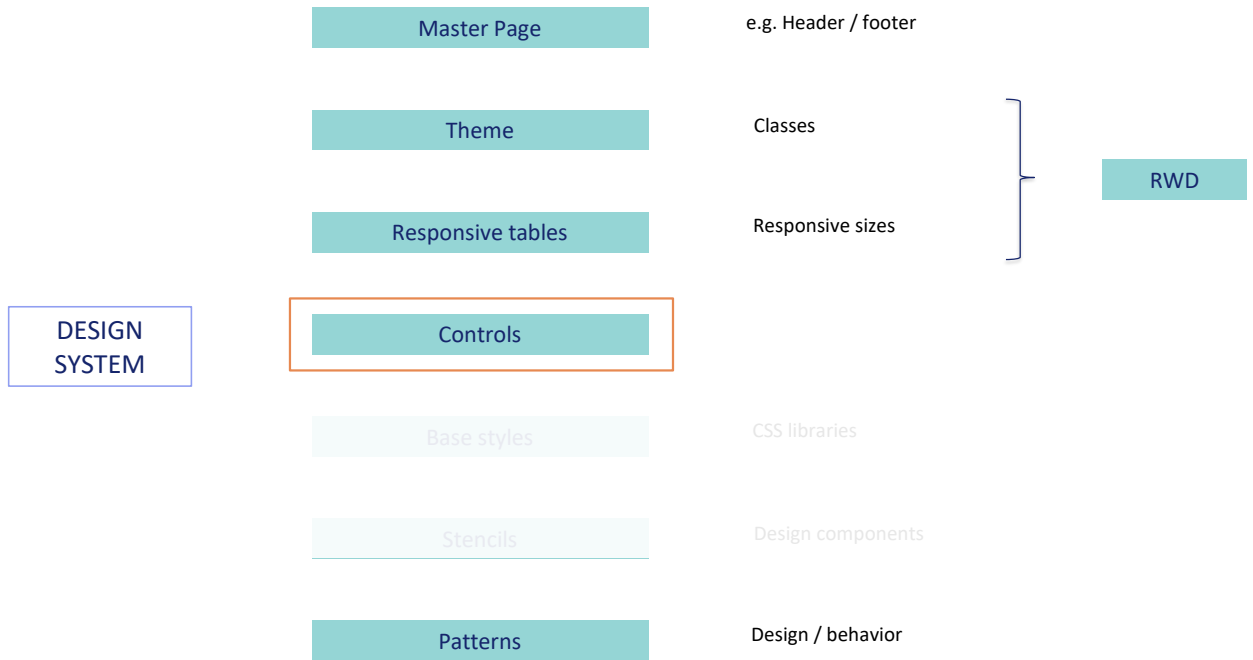
Therefore, to make sure that the grid of our web panel in Small size doesn't show other columns than those of the name of the attraction and actions, we change the classes of the columns according to what we've just seen.



In summary: another important part of the Design System has to do with responsiveness, which is achieved both by varying the positions and visibility of the responsive table controls, as well as by varying the class properties according to size as well (through these columns of the Theme we saw).

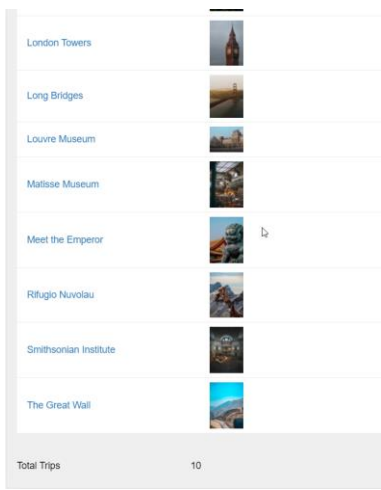


To achieve all this, we didn't have to do almost anything. We only had to replicate what the Work With pattern did automatically, together with the theme. For this reason, we say that GeneXus provides a basic, predefined Design System that we can use in our [web](#) panels.

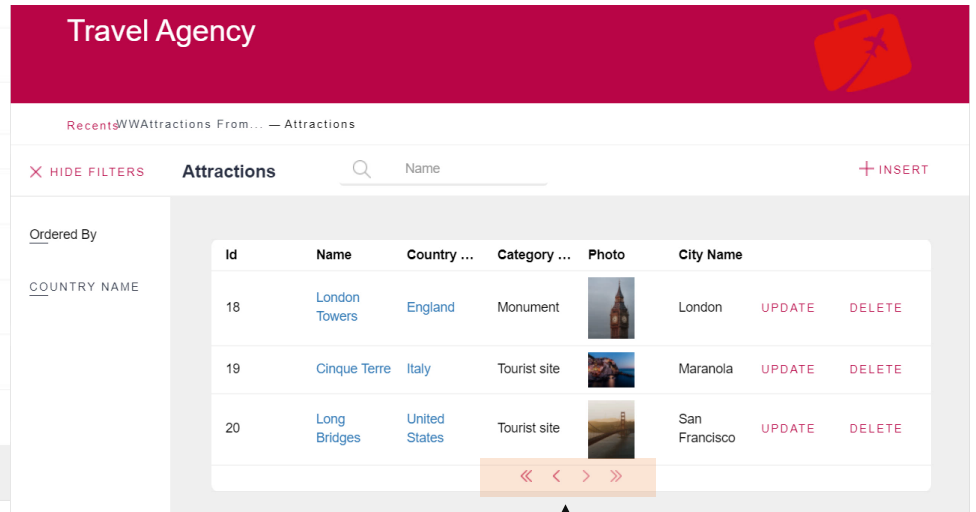


Another part of all this will have to do with the controls we use.

## Grids: design and behavior



Our web panel: no grid paging

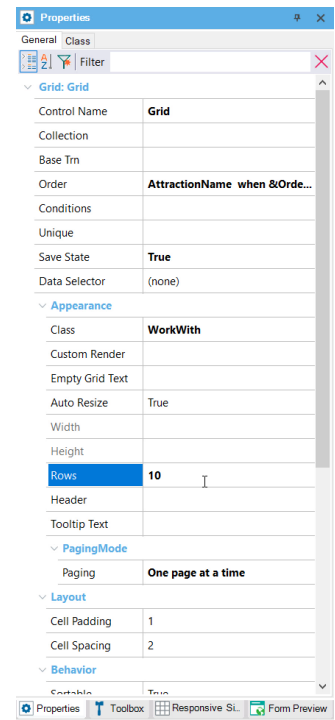
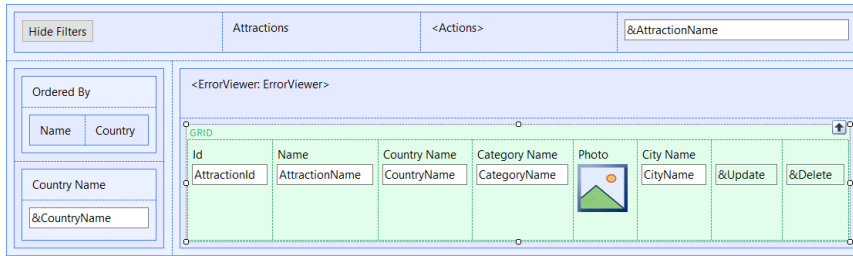


Work With web panel: grid paging

For example, in our web panel, where we replicated that of the Work With, we don't have the paging that the Work With has.



## Grid Paging



If we look at the grid properties, we see one named Rows, which has the value 10, along with the PagingMode property that has this value. Let's do the same on our grid.

## Grid Paging

Attraction Name From

Attraction Name To

Attraction Id	Attraction Name	Attraction Photo	Trips	Total Trips
AttractionId	AttractionName		&trips &update2 &newTrip	&totalTrips

Properties

General Class

Filter

Grid: Grid1

Control Name	Grid1
Collection	
Base Trn	Attraction
Order	CountryId, AttractionName w...
Conditions	CountryId = &CountryId whe...
Unique	
Save State	False
Data Selector	(none)

Appearance

Class	WorkWith
Custom Render	
Empty Grid Text	
Auto Resize	True
Width	
Height	
Rows	0
Header	
Tooltip Text	

Layout

Cell Padding	1
Cell Spacing	2

Behavior

Sortable	True
Allow Drop	False

Properties

General Class

Filter

Grid: Grid1

Control Name	Grid1
Collection	
Base Trn	Attraction
Order	CountryId, AttractionName w...
Conditions	CountryId = &CountryId whe...
Unique	
Save State	False
Data Selector	(none)

Appearance

Class	WorkWith
Custom Render	
Empty Grid Text	
Auto Resize	True
Width	
Height	
Rows	10
Header	
Tooltip Text	

PagingMode

Paging	One page at a time
--------	--------------------

Layout

Cell Padding	1
Cell Spacing	2


Behavior

Sortable	True
Allow Drop	False

The value 0 indicates that all rows will be loaded. By changing its value to 10, we are instructing it to paginate, which means to bring 10 records from the database at a time. Note that the Paging property shows the same value it had for the Work With.

## Grid Paging

## Travel Agency






Recent VW Attractions From...

Country Id: (None) ▾

Attraction Name From:

Attraction Name To:

Attraction Name	Attraction Photo	Trips		
Rifugio Nuvolau		0	<a href="#">UPDATE</a>	<a href="#">NEW TRIP</a>
Smithsonian Institute		1	<a href="#">UPDATE</a>	<a href="#">NEW TRIP</a>
The Great Wall		0	<a href="#">UPDATE</a>	<a href="#">NEW TRIP</a>

« < > »

Total Trips: 1

Let's generate this object and try it.

Here, we see that all the tourist attractions had been loaded, but if we refresh now, we see the paging option in action.

## Grid Paging: infinite scrolling

The image shows a development environment with a Properties window on the left and a web application preview on the right.

**Properties Window (Grid1):**

- Control Name: Grid1
- Collection: Attraction
- Order: CountryId, AttractionName w...
- Conditions: CountryId = &CountryId whe...
- Appearance: Class: WorkWith
- PagingMode:
  - Paging: Infinite scrolling
- Layout: Cell Padding: 1, Cell Spacing: 2

**Web Application Preview (Travel Agency):**

Country Id: (None)

Attraction Name From: [Input Field]

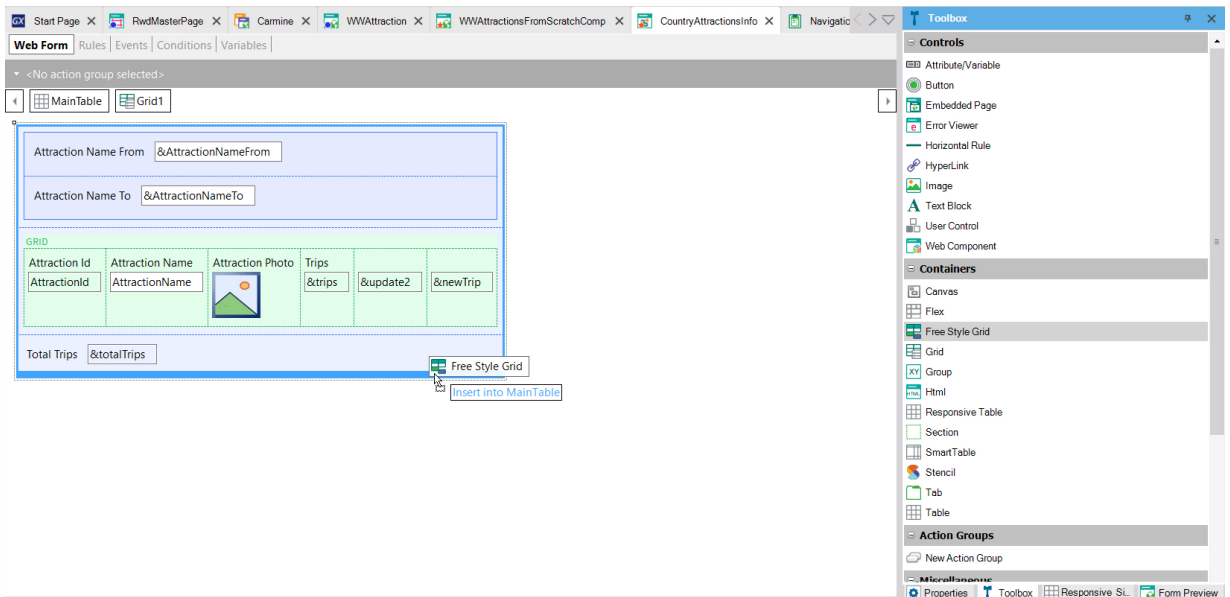
Attraction Name To: [Input Field]

Attraction Name	Attraction Photo	Trips	UPDATE	NEW TRIP
Christ the Redemmer		3	UPDATE	NEW TRIP
Cinque Terre		0	UPDATE	NEW TRIP
Eiffel Tower		2	UPDATE	NEW TRIP
Forbidden City		1	UPDATE	NEW TRIP
Glenfinnan Viaduct		0	UPDATE	NEW TRIP

Loading...

Now let's change the page size to 5 rows, and change the Paging property to take the Infinite Scrolling value and try it. I change the display for a Tablet, to see it better. There are 5 rows loaded on the grid, but we don't have the buttons to go to the next page. However, when we scroll, we see the Loading message. The next 5 rows are being searched for and loaded on screen, and so on with the last ones.

## Free style Grid



In addition, the information is being displayed in a structured manner, as if it were a table. What if we wanted to see the tourist attractions in a more flexible way?

For example if we only wanted to see a photo and name, but one below the other? For this we have another type of grid, the freestyle one. We drag it to the bottom of the form, start it with the attributes `AttractionName` and `AttractionPhoto`, and move them so that the photo is on top. We remove the label.

## Free style Grid

+ AttractionId



```

Start Event
...
AttractionId.Visible = False
...
endevent

```

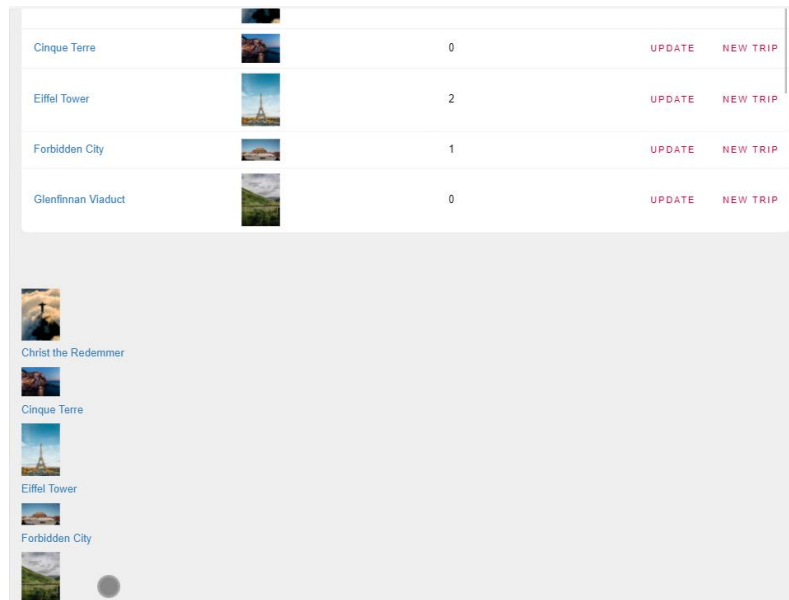
Properties	
General	Class
Control Name	Grid2
Collection	
Rendering Mode	Responsive
Save State	False
Base Trn	Attraction
Order	CountryId, AttractionName w...
Conditions	CountryId = &CountryId wh...
Unique	
Allow Drop	False
Allow Drag	False
Notify Context Chang	False
Tooltip Text	
Width	
Height	
Cell Padding	1
Cell Spacing	2
Control Info	
Scroll Direction	Vertical
Snap To Grid	False
Items Layout Mode	Single
Appearance	
Class	FreeStyleGrid
Rows	<unlimited>
Custom Render	

Note that the grid has the same properties as the standard one to determine the base transaction, Order, Conditions. So we copy them from the other.

Let's remove this variable now, which we are not interested in.

We did not add AttractionId, an attribute that we will need to be loaded, as in the other grid, even if we leave it hidden. This was to be able to send it in a parameter later.

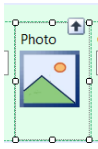
## Free style Grid



Let's try it now. We refresh...

And below our standard grid, we see the new one, without any design, of course. Suppose we want the image to appear much larger. Why is it looking so small?

## Bigger image



Properties	
General Class	
Attribute/Variable: AttractionPhoto	
Attribute	AttractionPhoto
Title	Photo
Class	ImageAttribute
Column Class	WWColumn WWOptionalColu...
Return On Click	False

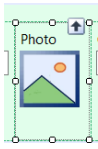
The screenshot shows the GeneXus IDE interface. The main window displays a class hierarchy for 'ImageAttribute'. The 'ReadonlyImageAttribute' class is highlighted in blue. The 'Properties' window on the right shows the 'Width' property set to '50px'. The 'Output' window at the bottom shows the build process: 'Success: Build With This Only'.

If we look at the class of the attribute control in the standard and in the freestyle grid we see that by default it was assigned the same class, `ImageAttribute`, because it is an attribute of `Image` data type. If we go to the theme to look for it, we will see that it has a series of subclasses, among them the one called `ReadonlyImageAttribute`. This will be the class applied to our photo at runtime, because the attribute control in this case is `ReadOnly`; so, even though we see the parent class in the properties, it will actually apply this child. If we look at its properties, we see that it specifies a width of 50 pixels for the image. That's why it looks so small.

If here we changed this property so that the width is, for example, 400 px we will get what we wanted for our grid. However, as a side effect, we will cause that any other control that has this class is also modified. In particular, the image in the `Work With`.



## Bigger image



Properties

General Class

Filter

Attribute/Variable: **AttractionPhoto**

Attribute	<b>AttractionPhoto</b>
Return On Click	False
On Click Event	
<b>Appearance</b>	
Label Position	None
Class	ImageAttribute2
Invite Message	

Start Page x RwdMasterPage x Camine x WWAAttraction x WWAAttractionsFromScratchComp x Properties

Styles Colors Images

Filter

Order Rules

- AudioAttribute
- BlobContentAttribute
- BlobInputAttribute
- CheckBox
- CheckboxLabel
- ComboAttribute
- DescriptionAttribute
- DownloadAttribute
- ErrorAttribute
- FilterAttribute
- ImageAttribute
  - ActionAttribute
  - ProfileImageAttribute
  - ReadOnlyImageAttribute
  - ResponsiveImageAttrib...
- ImageAttribute2
  - BlobContentImageAttrib...
  - BlobInputImageAttribute2
  - ReadOnlyImageAttribu...
- IME\_Active
- IME\_Disabled
- IME\_Inactive

Output

Show: Build

Uploading 27 Kbytes

Deploying website

Success: Build With This Only

Properties

Filter

Top Right Radius

Bottom Left Radi

Bottom Right Rac

Font

FontStyle

FontVariant

FontWeight

FontSize

TextDecoration

FontFamily

Arial

Forecolor

IME Mode

Height

Max- Height

100%

Min- Height

Width

auto

Max- Width

400px

Min- Width

Mouse and Keyboard

Cursor

Text

Letter Spacing

Text Align

left

Text Indent

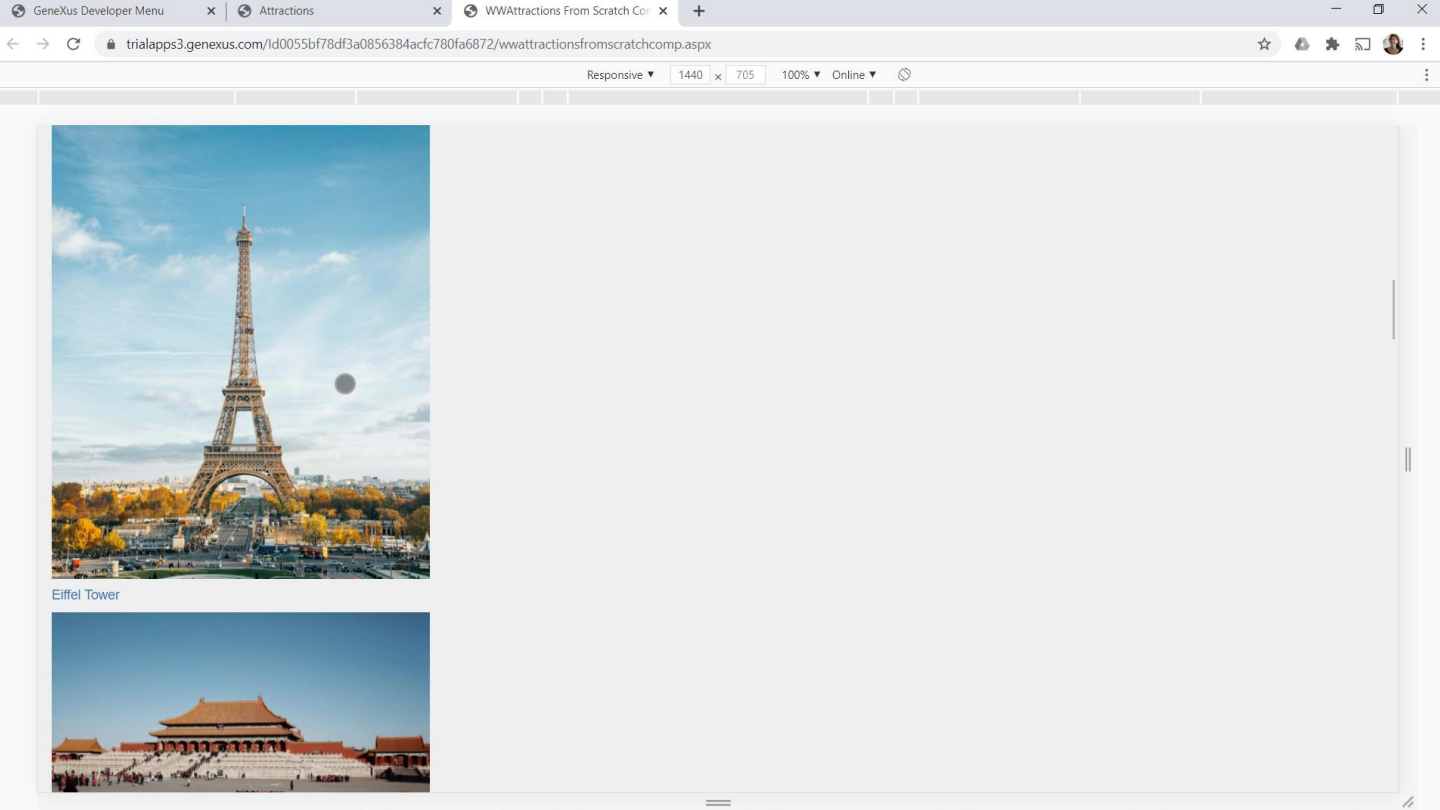
Text Transform

Vertical Align

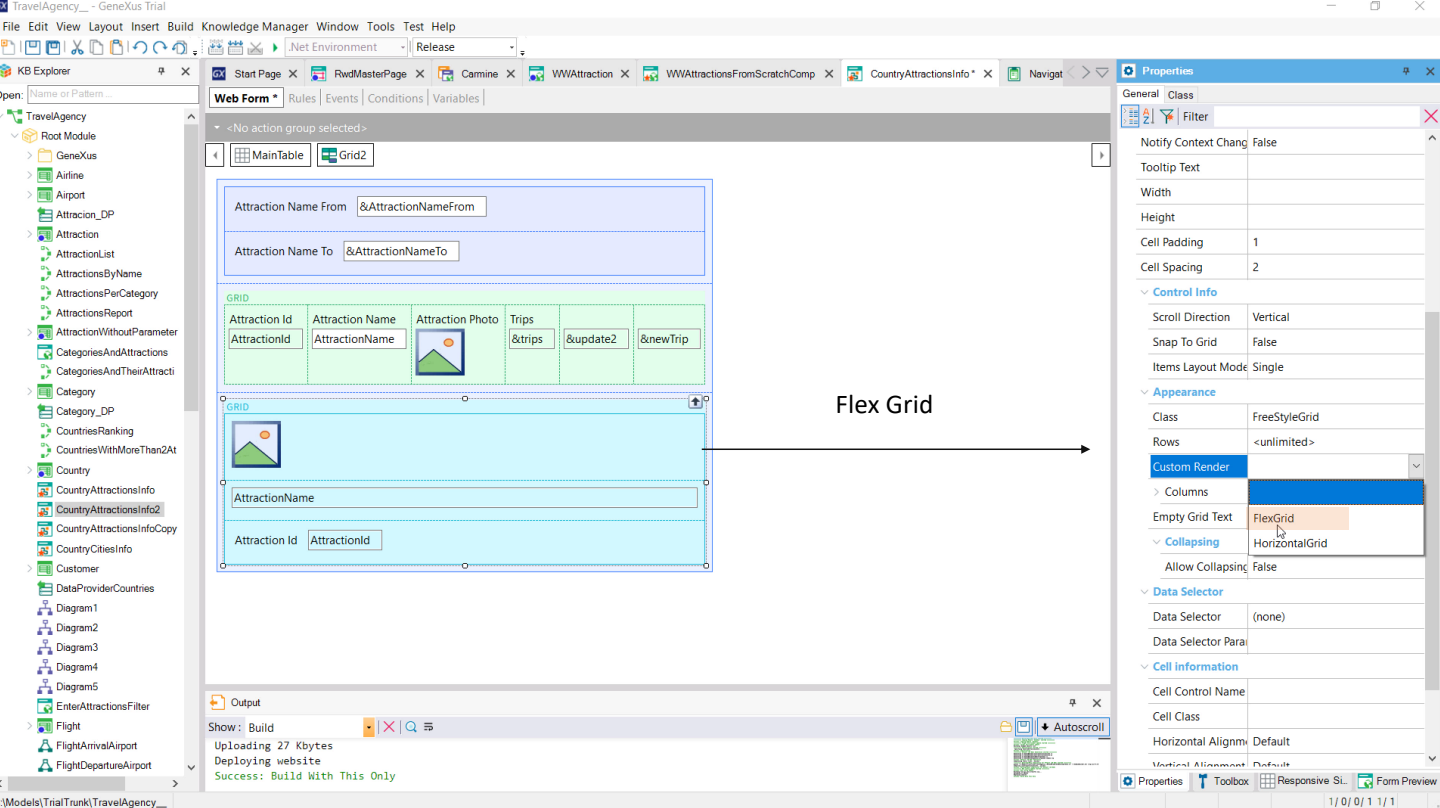
Properties | Toolbox | Responsive SL | Form Preview

We don't want this, so we have created another sister class to this one (here it is already created), which we called ImageAttribute2. It was created automatically with all these subclasses, and what we did was to change these properties for the ReadOnly one. We are setting the image to be displayed in its original size, as long as its width does not exceed 400 px. And in that case, it should be reduced to that size.

So now we will change the class of our AttractionPhoto control for this second grid, to this other one.



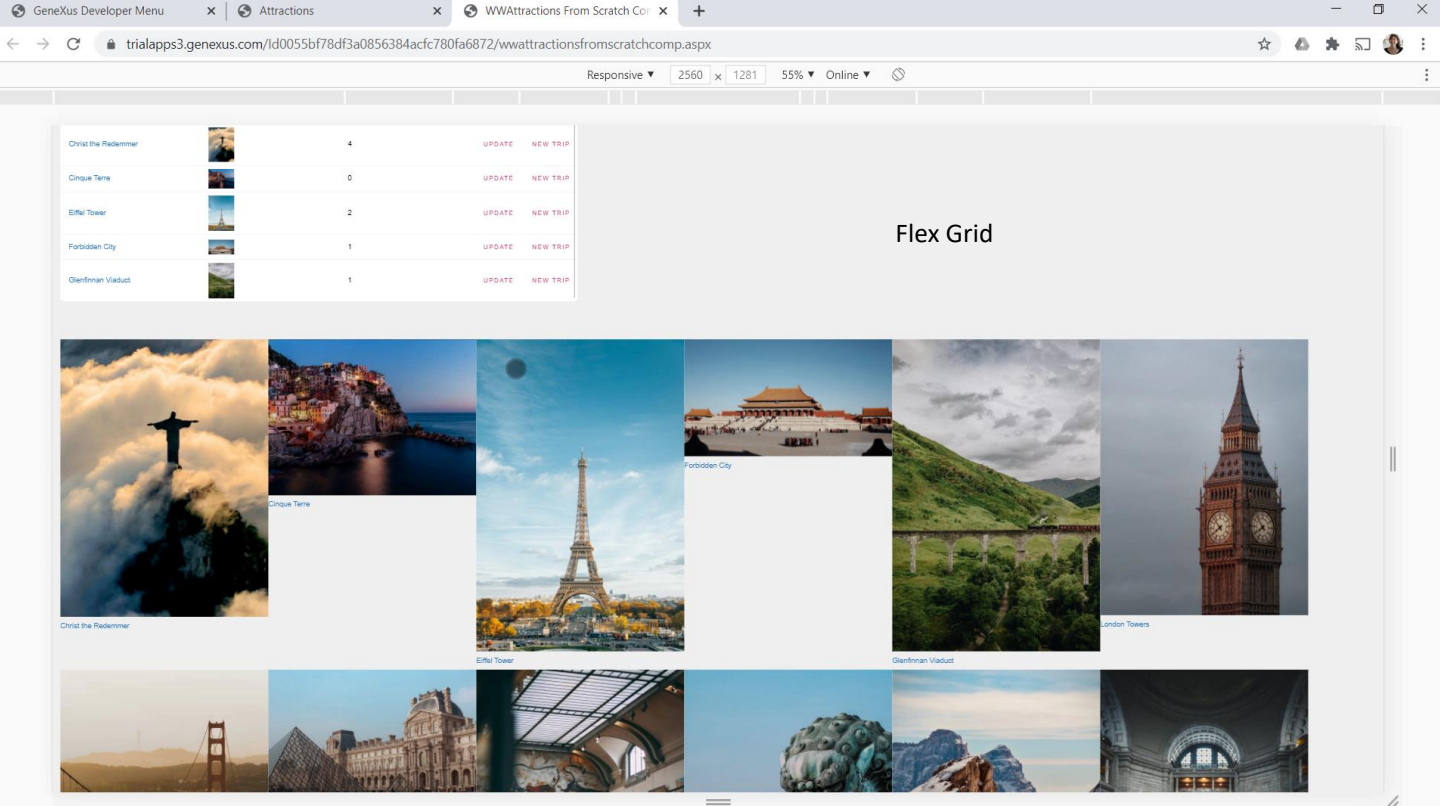
Let's try it at runtime. The difference can be clearly seen.



What if now we want to show the attractions in a way that makes the most of the screen space?

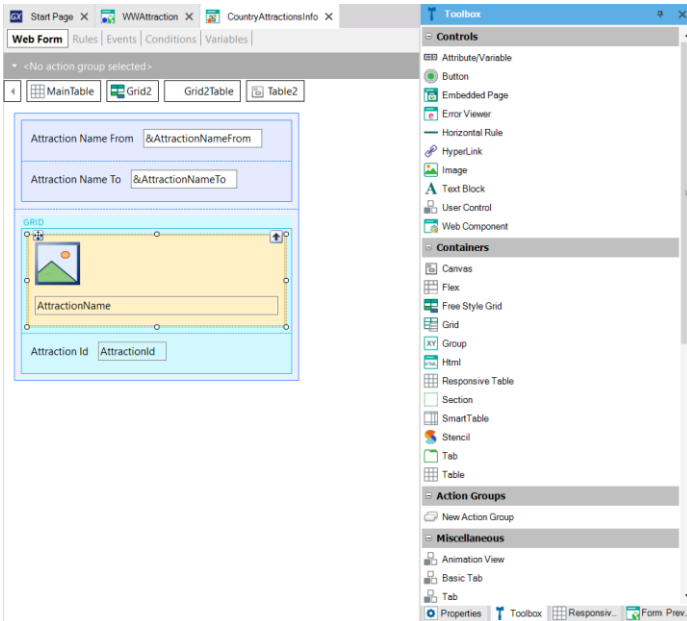
We see that the photos are not of similar size.

Let's go to the grid, and change its rendering to make it a flexible grid; that is, so that its content can take up the empty spaces on the screen like a puzzle. Let's try it.

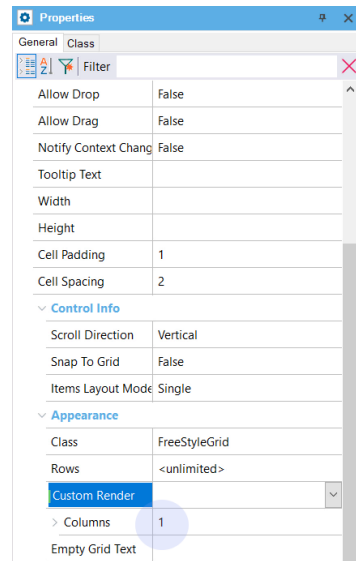


We see that depending on the screen size, and considering that maximum width of 400 pixels, the tourist attractions are arranged in the Row direction to fill the space.

## Canvas for overlapping



## Default FreeStyle Grid

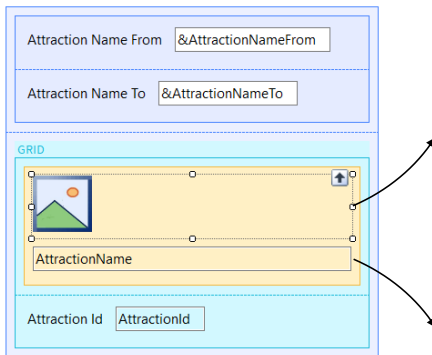


And if we wanted the name of the attraction to overlap the image?

Let's start by deleting the standard grid we had. We don't show it, but we have just deleted the associated events as well.

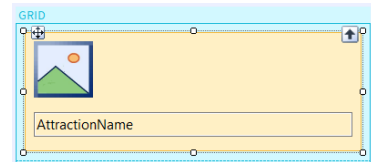
Now we insert a Canvas control into the form, which is a type of table that allows overlapping of the controls it contains. We take in there the two controls that we want to overlap. Let's leave the freestyle grid with the default rendering again, which will list one attraction (with its photo and name) per row.

## Canvas for overlapping



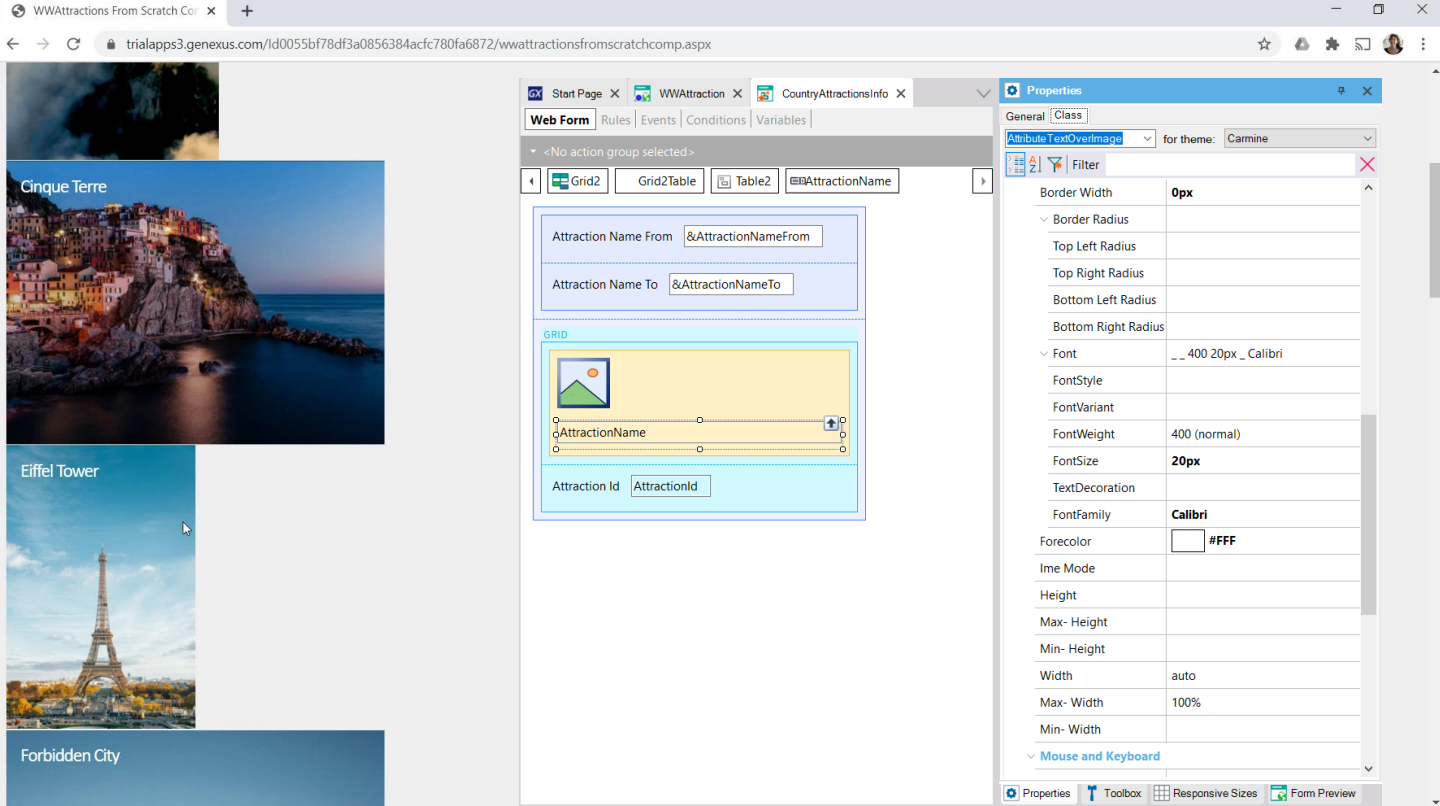
Absolute position	
Top	0px
Left	0px
Bottom	0px
Right	0px
Width	100%
Height	100%
Z- Order	1

Absolute position	
Top	0px
Left	15px
Bottom	0px
Right	15px
Width	100%
Height	100%
Z- Order	2



Properties	
General	Class
Filter	
Canvas: Table2	
Control Name	Table2
Tooltip Text	
Header	
Appearance	
Width	100%
Height	300px
Class	Table
Cell information	
Cell Control Name	
Cell Class	
Horizontal Alignment	Default
Vertical Alignment	Default
Row information	
Row Height	
Row Class	

Note that by placing this control inside the Canvas, these properties have appeared that allow placing the control in relation to the table, as well as setting how much space it will take up and in which layer it will be. We leave all the default values, except for the layer. We write 1, because we want the image to be in the deepest layer, below the next one, that of the name. We will place the name of the attraction in layer 2, above layer 1. We also want this control to have a left margin of 15 pixels, so that it has a margin in relation to the photo. Note that among the properties of the Canvas table, there are those that allow indicating its height and width in relation to the control that contains it. Here we are letting it take up 100% of the width, but we are going to indicate that the height should be, for example, 300 pixels.



And finally, let's change the design of the AttractionName control, so that it looks better.

By default it has the Attribute class associated with it, but we will change it to one that we have created in the theme, with this name. In its properties, we have selected white font color, Calibri font type, 20 pixel size, etc.

Let's try all this at runtime. If we now click on the name of the Eiffel Tower... we see its information.

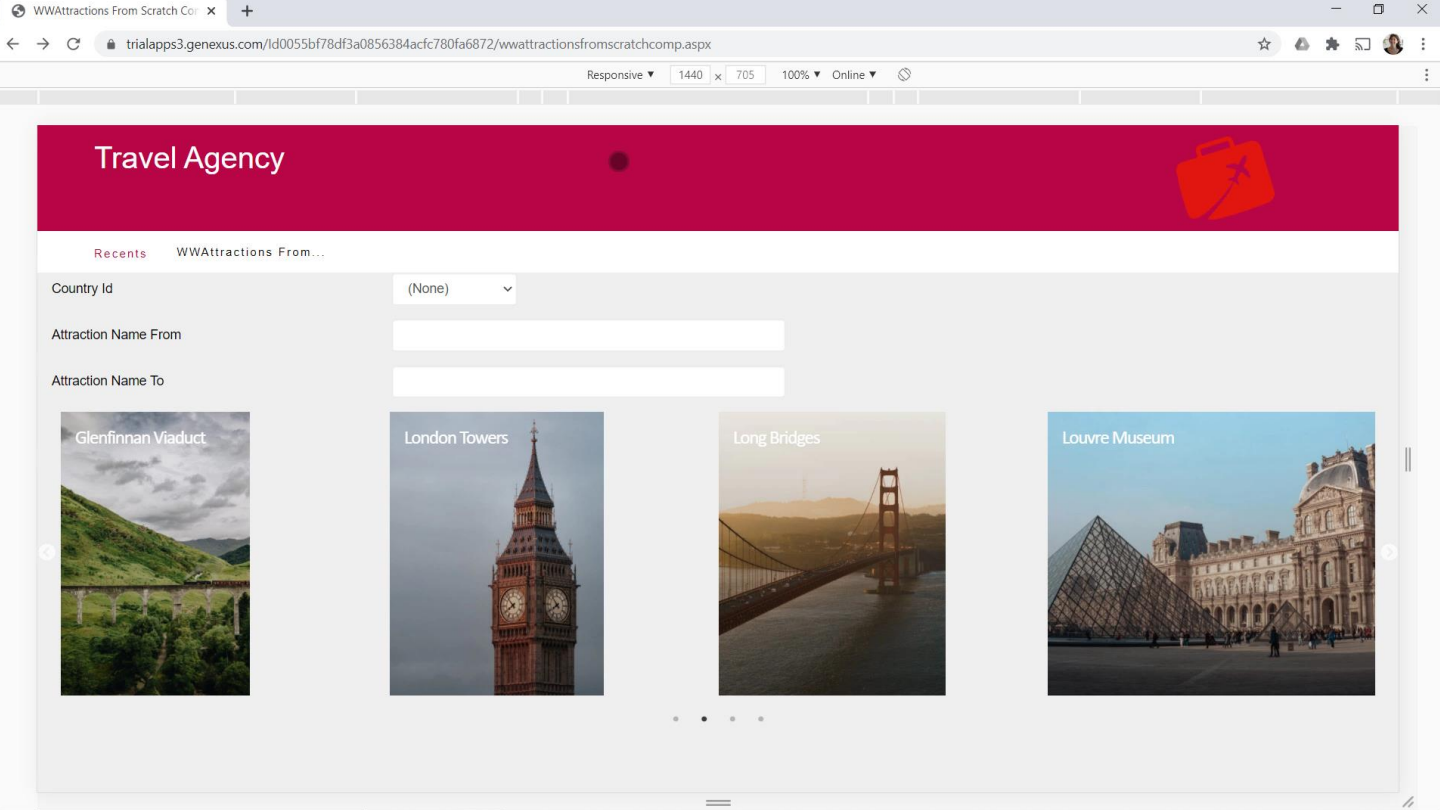
## Horizontal grid

The screenshot shows the GeneXus IDE interface. The main workspace displays a web form with a grid control. The grid is currently rendered as a vertical list. The properties panel on the right shows the following settings:

Property	Value
Cell Spacing	2
Control Info	
Scroll Direction	Vertical
Snap To Grid	False
Items Layout Mode	Single
Appearance	
Class	FreeStyleGrid
Rows	<unlimited>
Custom Render	<b>HorizontalGrid</b>
Columns	1 2 3 4
Extra Small	1
Small	<b>2</b>
Medium	<b>3</b>
Large	<b>4</b>
Empty Grid Text	

And to finish looking at some of the possibilities offered by grid controls, if we now change its rendering so that it is displayed as a horizontal grid... where, for example, for phone size we want it to show only one column, for small size, 2, for medium 3 and for large 4...

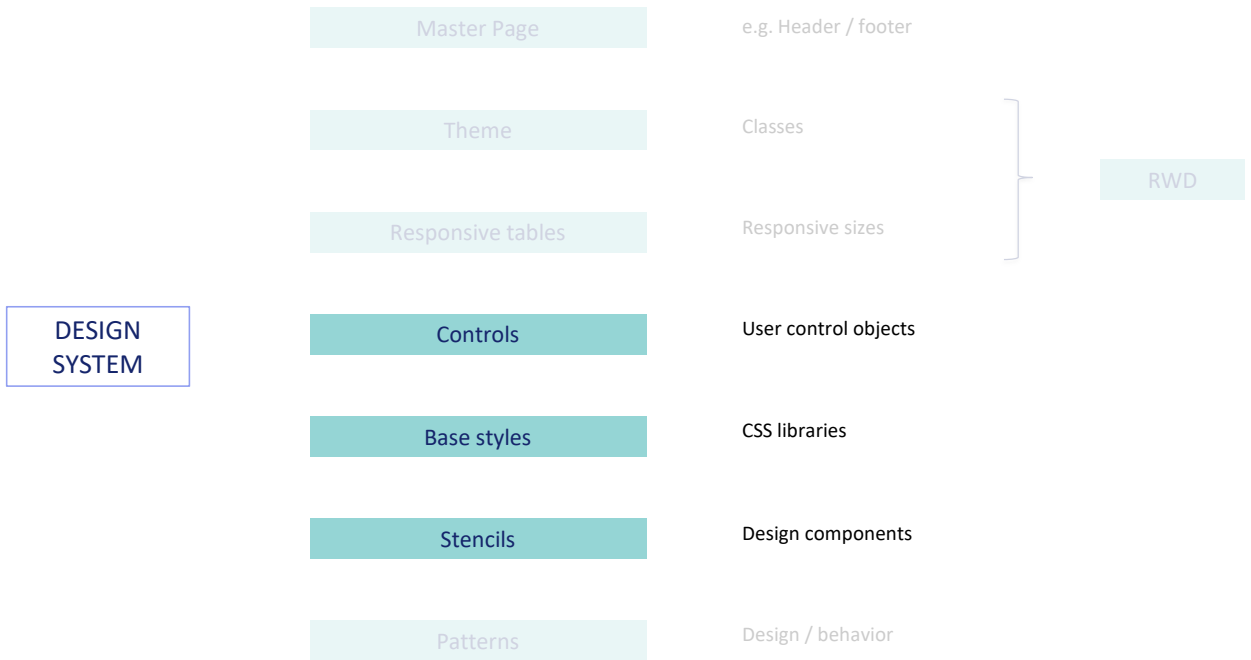




We press F12... Here is the Large size, with 4 attractions per horizontal page...  
Here are 3...  
For a Tablet 2...  
And for a phone one.

Of course, here we should insert images of the same size.

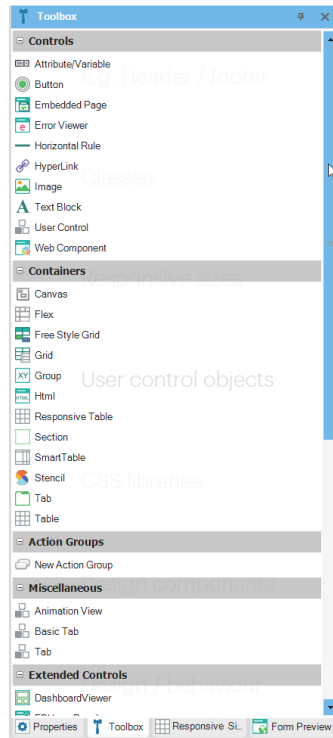
This is just an introduction to all we can do with the controls and classes of the theme.



There are other players, together with those we have seen, that allow using a powerful Design System in GeneXus. We will not study them at this level; only an introduction will be provided.

DESIGN SYSTEM

- Master Page
- Theme
- Responsive tables
- Controls**
- Base styles
- Stencils
- Patterns

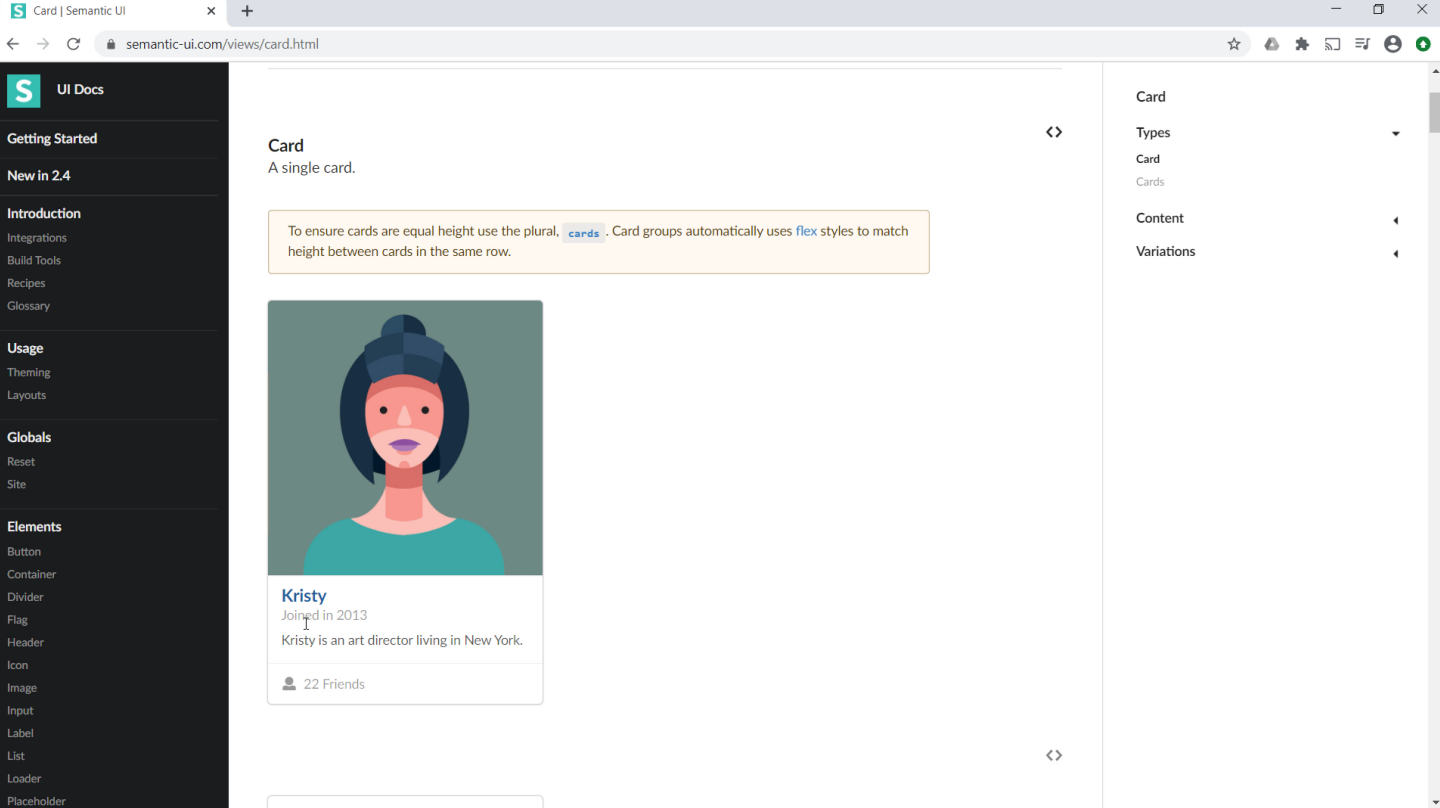


RWD

We can not only use in our forms the controls included in the GeneXus toolbox in the predetermined way...



...but we can also define user controls that can be copied from platforms that offer those controls along with CSS libraries (which, like themes in this case, specify their style –the style of those controls).



For example, this one, the Semantic UI. We may want to use a Card control such as this one.

To do this, you just need to create a User control object (with the same name)...

Card | Semantic UI


semantic-ui.com/views/card.html

## Card

A single card.

To ensure cards are equal height use the plural, `cards`. Card groups automatically uses flex styles to match height between cards in the same row.

Example



```
1 <div class="ui card">
2   <div class="image">
3     
5   <div class="content">
6     <a class="header">{{Name}}</a>
7     <div class="meta">
8       <span class="date">{{Date}}</span>
9     </div>
10    <div class="description">
11      {{Description}}
12    </div>
13  </div>
14  <div class="extra content">
15    <a>
16      <i class="user icon"></i>
17      {{ExtraContent}}
18    </a>
19  </div>
20 </div>
```

`<div class="ui card">`

... copy and paste its HTML code, and change the fixed data to something like names of SDT elements (to be able to make it dynamic; that is to say, to dynamically load that control, with data that you are going to be able to specify, for example, of the database).

Card | Semantic UI

semantic-ui.com/views/card.html

UI Docs

Getting Started

New in 2.4

Introduction

Integrations

Build Tools

Recipes

Glossary

Usage

Theming

Layouts

Globals

Reset

Site

Elements

Button

Container

Divider

Flag

Header

Icon

Image

Input

Label

List

Loader

Placeholder

Kristy  
Joined in 2013  
Kristy is an art director living in New York.  
22 Friends

```
<div class="ui card">
  <div class="image">
    
  </div>
  <div class="content">
    <a class="header">Kristy</a>
    <div class="meta">
      <span class="date">Joined in 2013</span>
    </div>
    <div class="description">
      Kristy is an art director living in New York.
    </div>
    <div class="extra content">
      <a>
        <i class="user icon"></i>
        22 Friends
      </a>
    </div>
  </div>
</div>
```

Screen Template Properties

```
1 <div class="ui card">
2   <div class="image">
3     
5   <div class="content">
6     <a class="header">{{Name}}</a>
7     <div class="meta">
8       <span class="date">{{Date}}</span>
9     </div>
10    <div class="description">
11      {{Description}}
12    </div>
13  </div>
14  <div class="extra content">
15    <a>
16      <i class="user icon"></i>
17      {{extracontent}}
18    </a>
19  </div>
20 </div>
```

User Control: Card

Name	Card
Description	Card
Module/Folder	Root Module
Is Control Type	False
References	
Base Control Type	None
Base Style	SemanticUI
Qualified Name	Card
Object Visibility	Public

Also, indicate from where it will take the CSS –the class design–, and do something else...

The screenshot shows a web browser window displaying a card component. The card features a profile picture of a person with red hair, the name "Kristy", the text "Joined in 2013", a description "Kristy is an art director living in New York.", and "22 Friends".

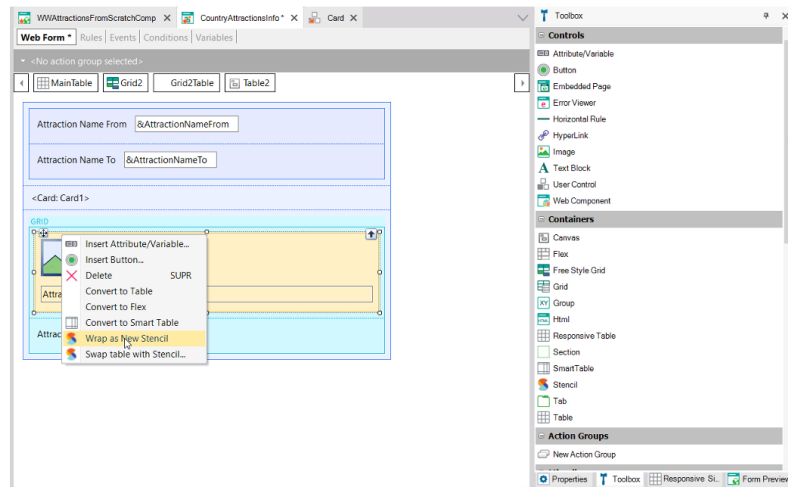
Below the browser window, the HTML code for the card is shown:

```
<div class="ui card">
  <div class="image">
    
  </div>
  <div class="content">
    <a class="header">Kristy</a>
    <div class="meta">
      <span class="date">Joined in 2013</span>
    </div>
    <div class="description">
      Kristy is an art director living in New York.
    </div>
  </div>
  <div class="extra content">
    <a>
      <i class="user icon"></i>
      22 Friends
    </a>
  </div>
</div>
```

Overlaid on the browser window is a design tool interface. The design tool shows a preview of the card component with various elements highlighted. The "Card: Card1" component is selected in the "Controls" panel. The "Toolbox" panel on the right shows a list of controls, including "Image", "Text Block", "User Control", "Web Component", "Canvas", "Flex", "Free Style Grid", "Grid", "Group", "Html", "Responsive Table", "Section", "Smart Table", "Stencil", "Tab", and "Table". The "Image" control is highlighted in the "Toolbox" panel.

In this way, you'll have it available in the Toolbox to use it.





DESIGN  
SYSTEM

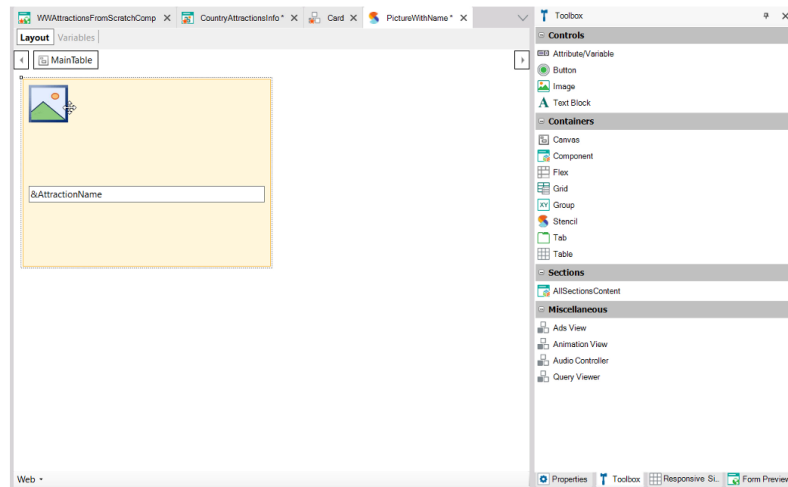
Stencils

Design components

Patterns

Design / behaviour

In addition, GeneXus offers Stencils, which allow repeating the design of the same portion of the screen (a set of controls), in many screens.



RWD

DESIGN  
SYSTEM

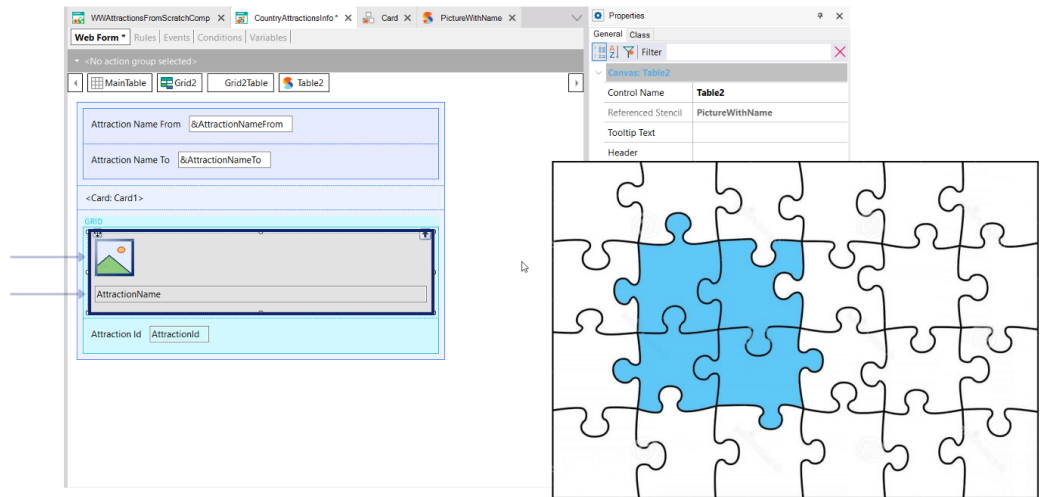
Stencils

Design components

Patterns

Design / behaviour

It's a way to abstract design at a higher level.



DESIGN  
SYSTEM

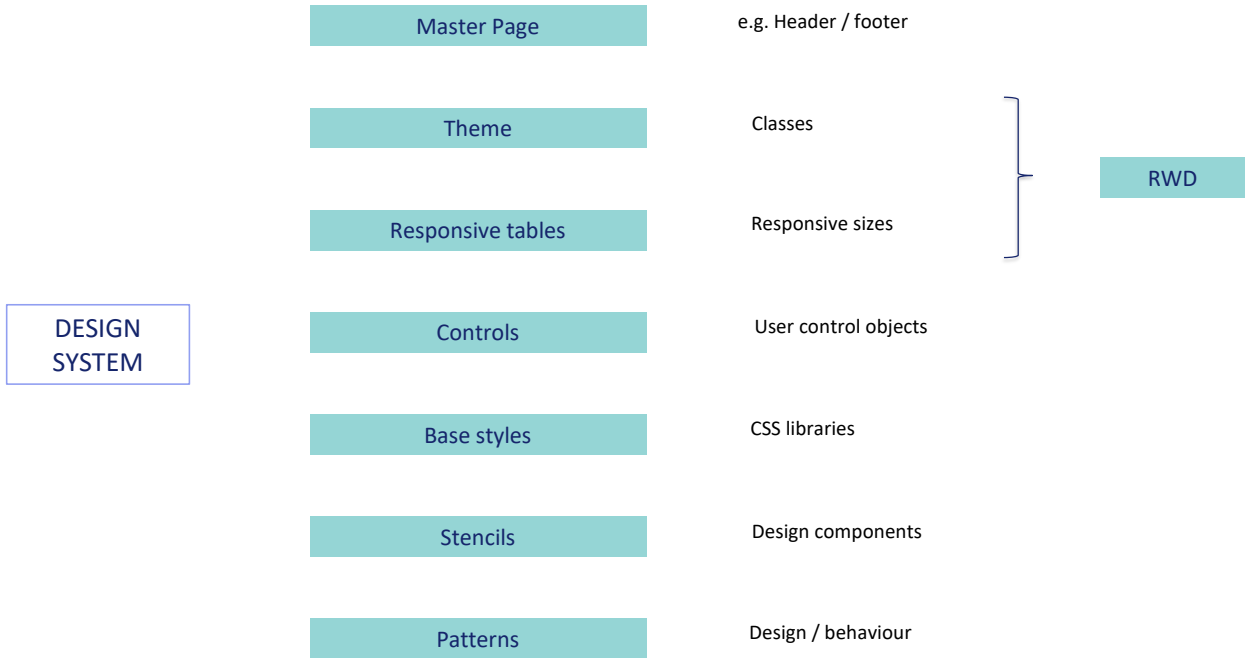
Stencils

Design components

Patterns

Design / behaviour

If you think of the screens of a web application like puzzles made up of atomic pieces (the controls), stencils are a way of grouping together a set of controls whose design will be repeated on many screens. It would be a part made of other smaller parts.



In this way, we have introduced the most important players involved in the implementation of the Design System in GeneXus.

# GeneXus™

[training.genexus.com](http://training.genexus.com)

[wiki.genexus.com](http://wiki.genexus.com)

[training.genexus.com/certifications](http://training.genexus.com/certifications)