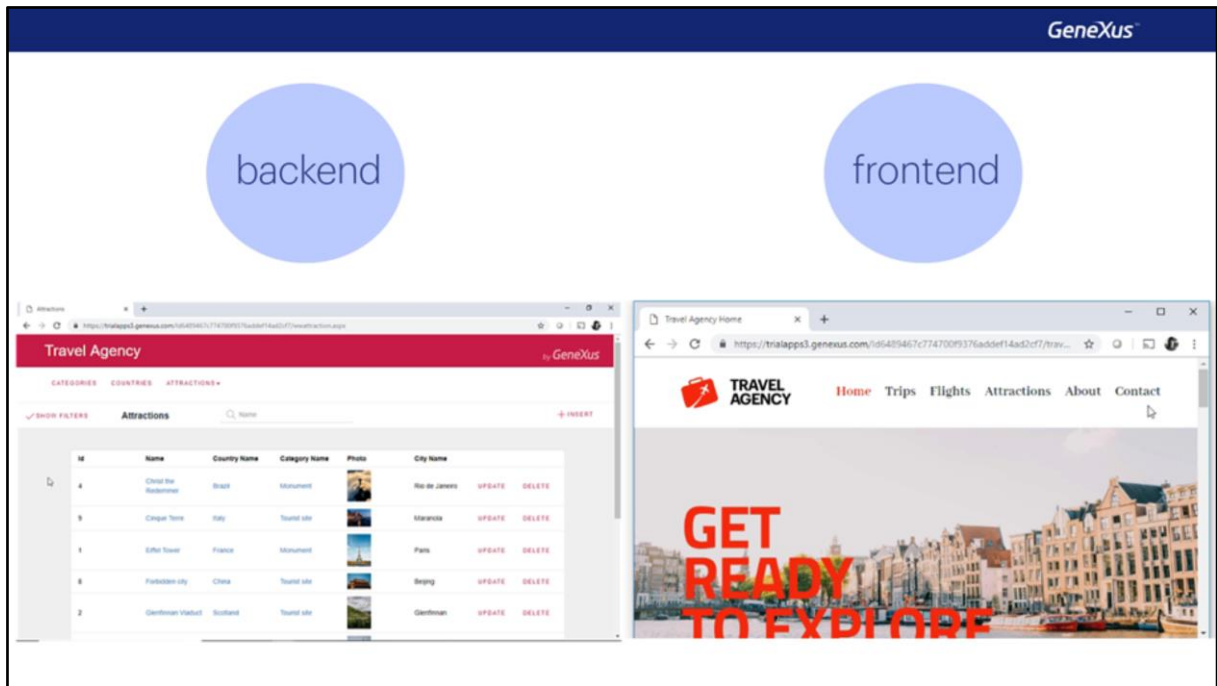


Design System em GeneXus

Default

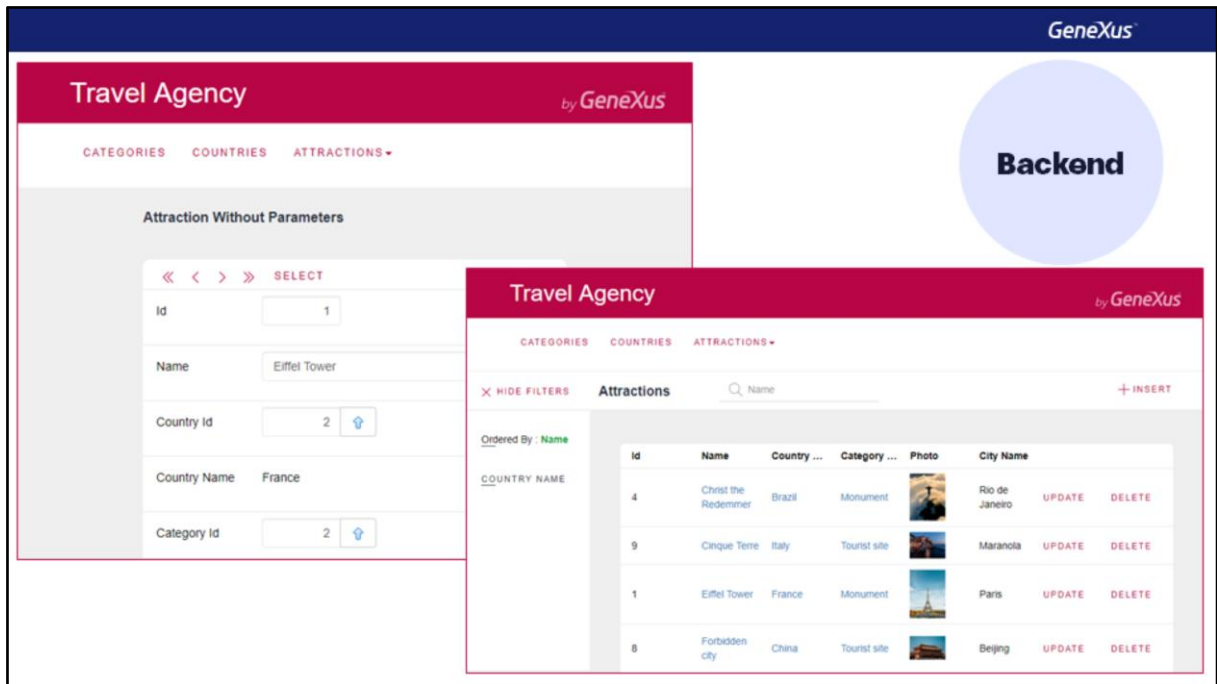
GeneXus® 16



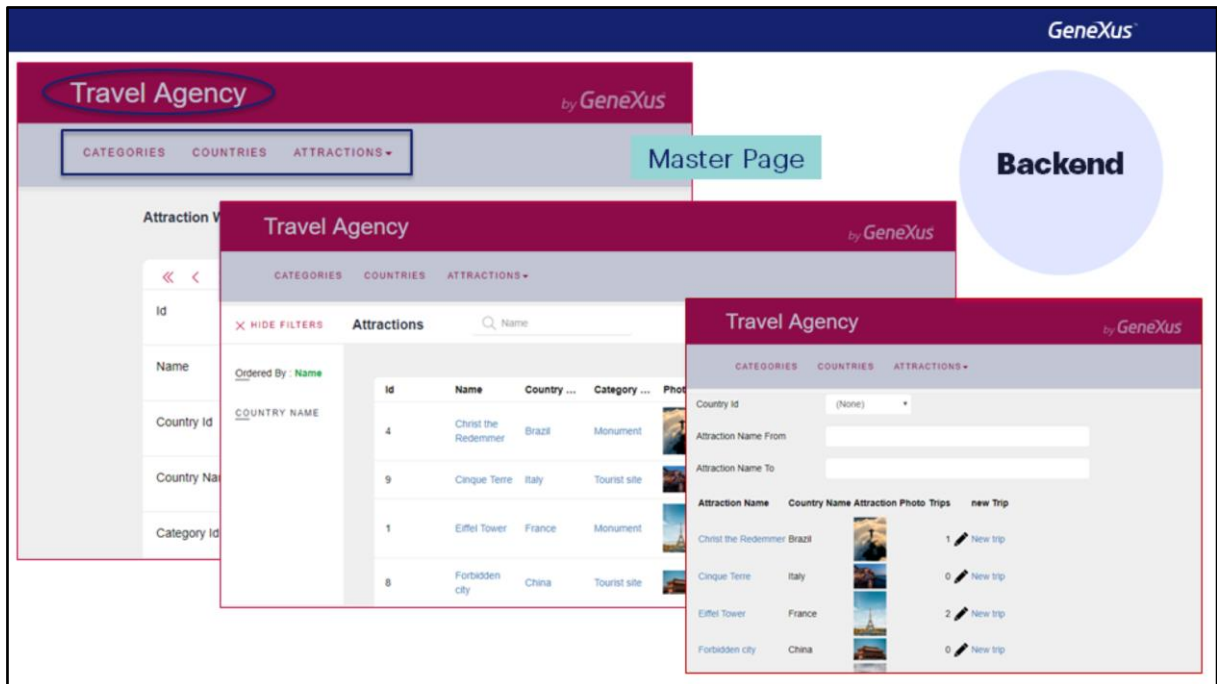
[DEMO: https://youtu.be/l_VA02JfGYQ]

Dissemos repetidamente que até agora estávamos trabalhando em telas do **backend**. No entanto, como veremos no final, o **frontend** é completamente análogo. A maior diferença é o maior esforço que colocaremos em seu design.

Começamos pelo **backend**...



Se prestarmos atenção às transações que estamos utilizando, e aos painéis criados pelo padrão Work With, vemos que existem elementos que dão uniformidade e coerência, sem nos preocuparmos em conseguir isto. O que significa que GeneXus já está fazendo algo por nós em termos de fornecer um Design System predeterminado, mesmo que seja elementar.



Por exemplo, vemos que todas as telas (inclusive aquelas que criamos do zero, como este web panel) têm um cabeçalho comum. Em nosso caso, nós o personalizamos, mudando o texto padrão para "Travel Agency" e removendo a seção "Recents" que estava predefinida (ah, e adicionamos uma barra de menu!). A maneira de obtê-lo é com o objeto web Master Page (Aqui vemos o text block modificado e o menu adicionado).

GeneXus

Master Page

(default na KB)

RwdMasterPage X

Web Form | Rules | Events | Conditions | Variables

NavMenu | Categories | Countries

MainTable

Travel Agency

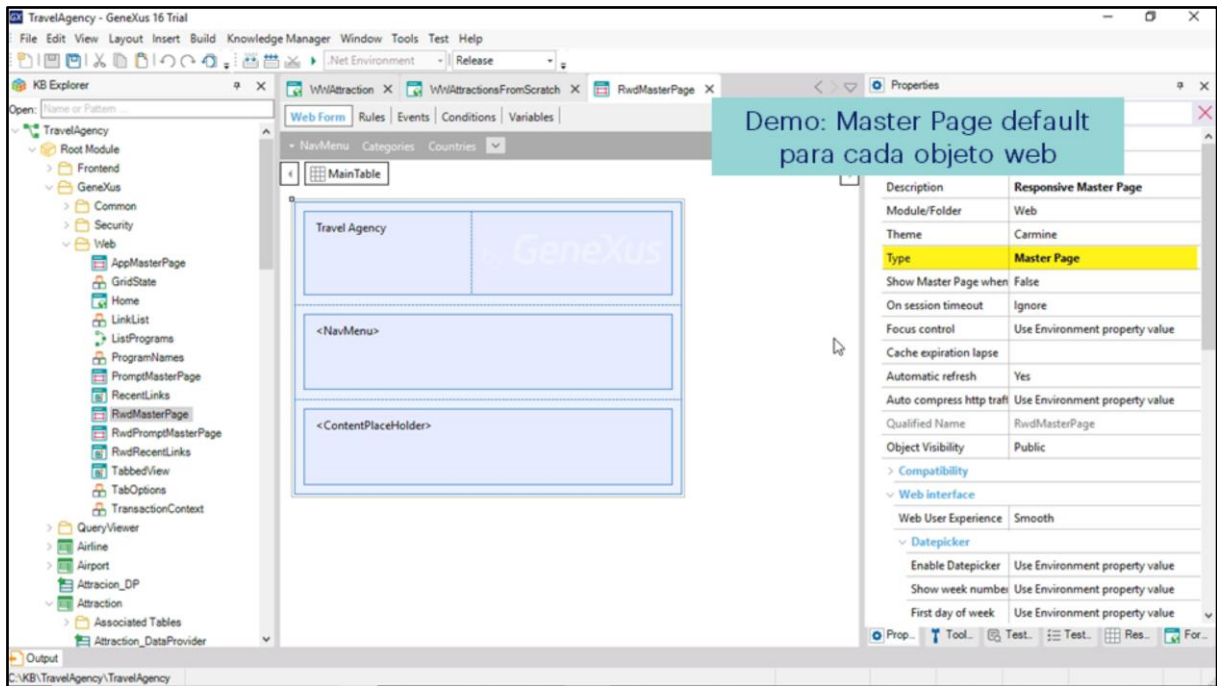
by GeneXus

<NavMenu>

<ContentPlaceholder>

Backend

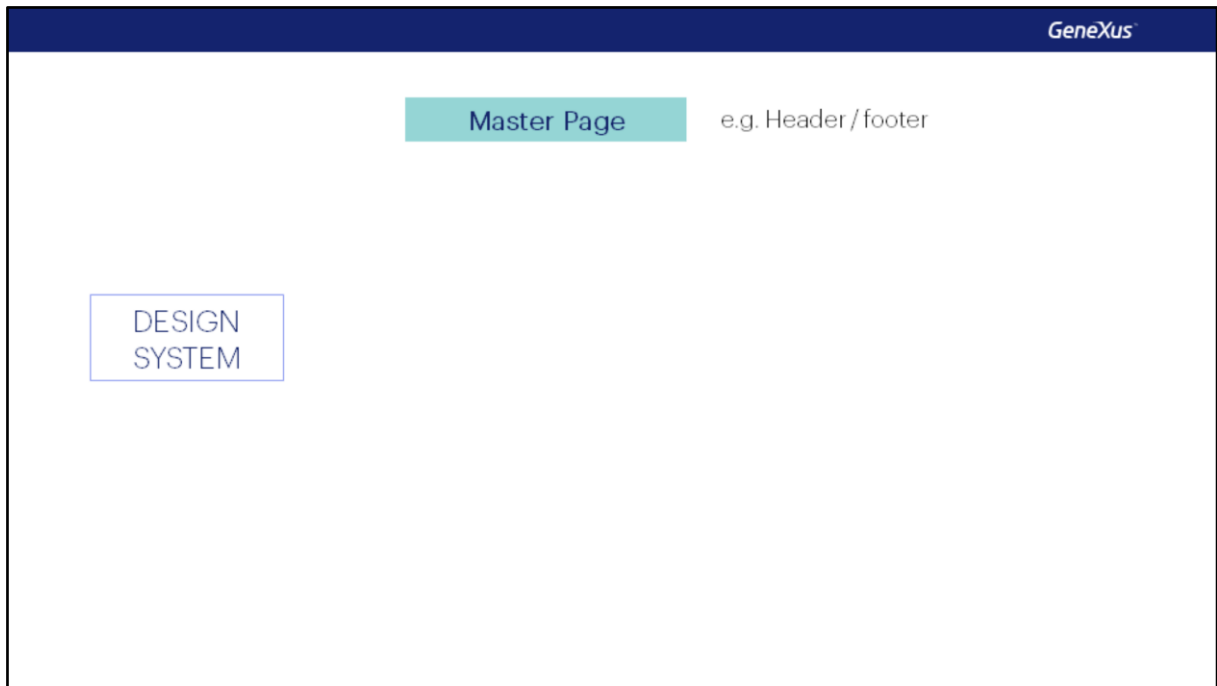
Por padrão, a versão da KB vem com esta Master Page predefinida, o que significa que ela será aplicada a todos os objetos web criados.



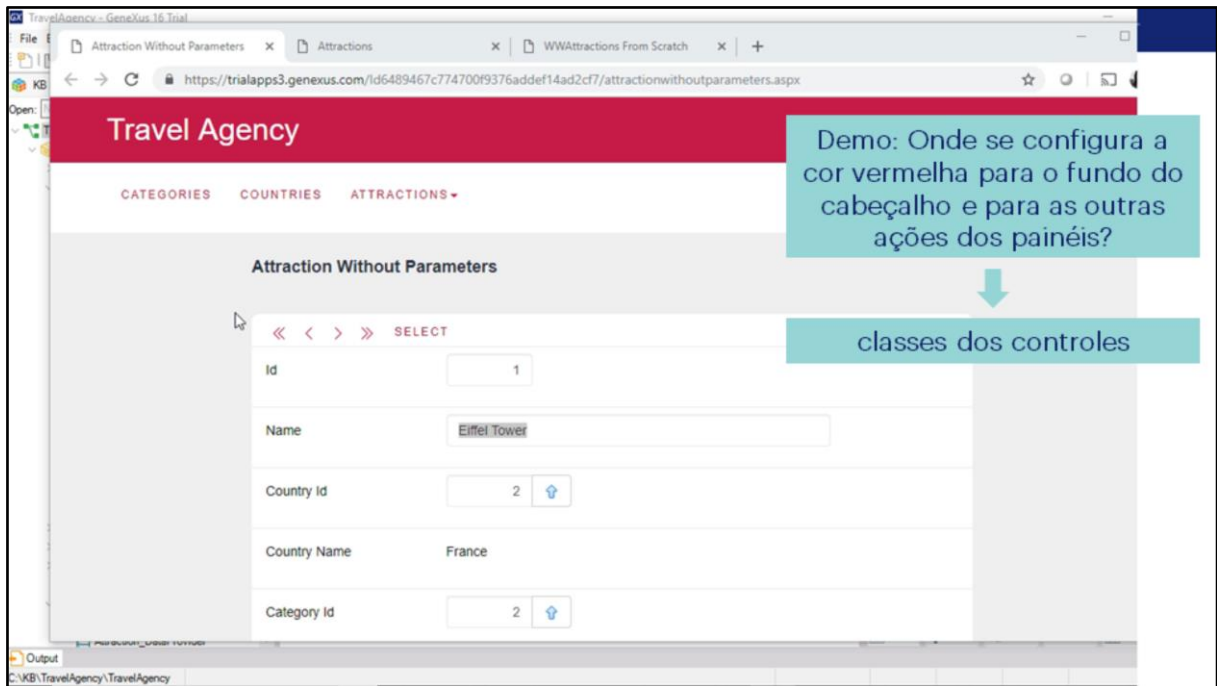
[Demo: <https://youtu.be/PYBRdASQBvs>]

Assim, se observamos as propriedades da transação que vimos em execução, vemos que tem esta definida como sua Master Page, a default. E o mesmo para o Work With e para nosso painel.

Veremos que a Master Page contém este controle que indica que aqui é onde o conteúdo de cada página web será carregado.



Então, por um lado GeneXus já nos oferece um objeto Master Page para centralizar quais serão as informações comuns que queremos que as páginas compartilhem.



[DEMO: <https://youtu.be/vkybJzp3uEO>]

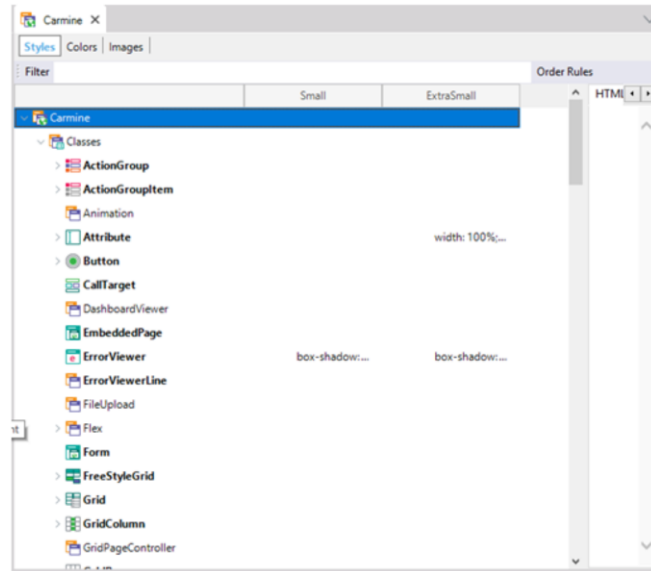
Por outro lado, podemos ver que a cor base, a que se destaca, é um tipo de vermelho, que é utilizado não apenas para o cabeçalho, mas também para os botões e outras ações que são fornecidas ao usuário. Onde se configura tudo isso?

Sabemos que esta tabela tem um fundo vermelho em execução, mas não o estamos vendo! No entanto, observemos que o controle (neste caso, do tipo **tabela responsiva**) tem um par de **classes** definidas, não casualmente sob o grupo "Aparência". Se formos para a **guia Class**, vemos que estão sendo editadas as propriedades da segunda das classes associadas a essa tabela. E vemos que existem coisas definidas ali tais como margens, preenchimentos e a cor de fundo vermelha que vimos!

A vantagem das classes é que permitem centralizar o design desse tipo de controle (neste caso, tabela), e que, em nosso caso, outras tabelas que estão no mesmo ou em outros forms possam compartilhar a mesma definição, o que significa que se queremos mudar, por exemplo, a cor de fundo de vermelho para azul para esses controles, não temos que fazer isso um a um em cada controle, ou seja, em cada tabela, mas se faz diretamente na classe e isso já afeta todos os controles que a tenham associada.

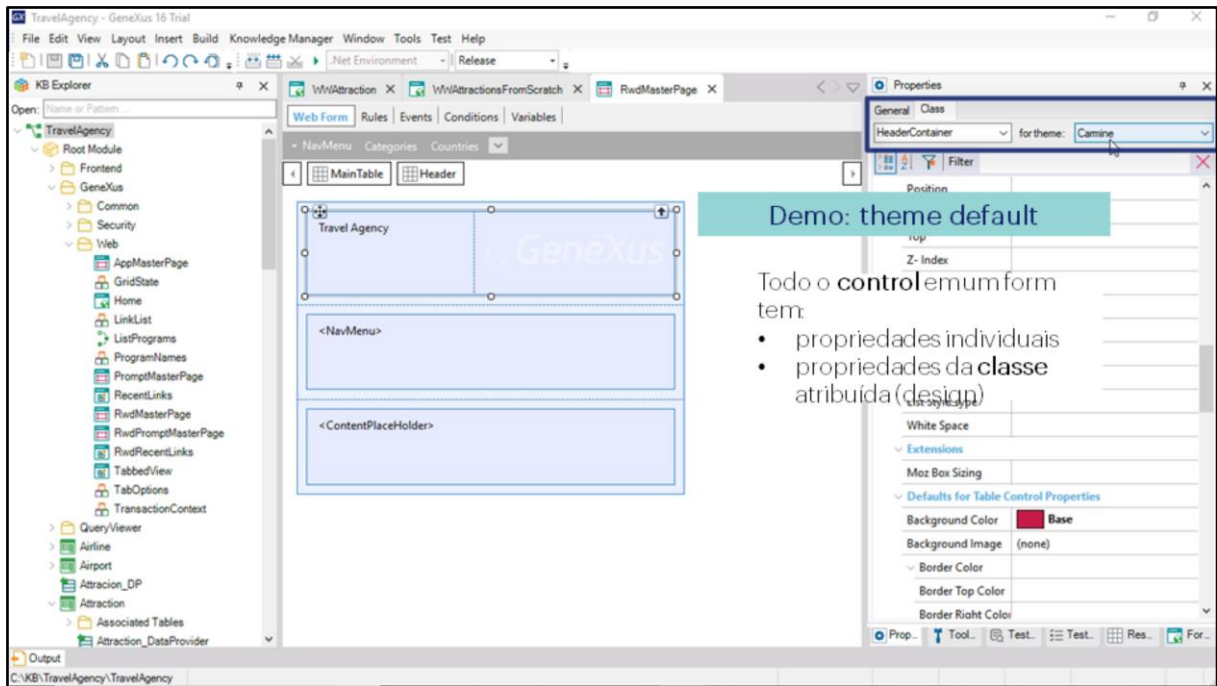
Theme

(default na KB)



Backend

Onde estão todas as classes definidas? No objeto Theme, Tema! Vemos que, assim como havia uma Master Page predefinida, há também um tema predefinido, chamado Carmine.



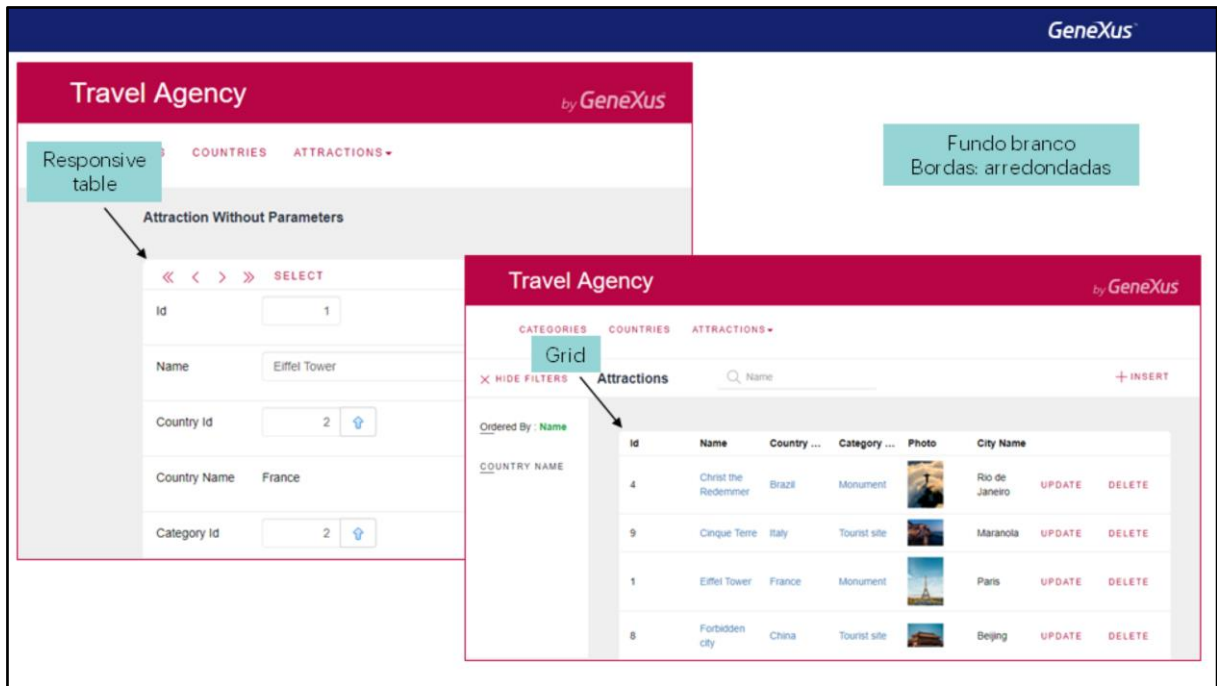
[DEMO: <https://youtu.be/tkdOB-JPSgg>]

Como veremos mais adiante, criamos um nosso que será o que utilizaremos para o frontend. Em nosso backend, usamos o default. Vemos que ao abri-lo, aparecem classes agrupadas por tipo de controle. Aqui é onde está tudo centralizado. E também podemos ver na guia Colors, que mostra a paleta de cores com o mesmo nome... Que é onde está definida a color Base.

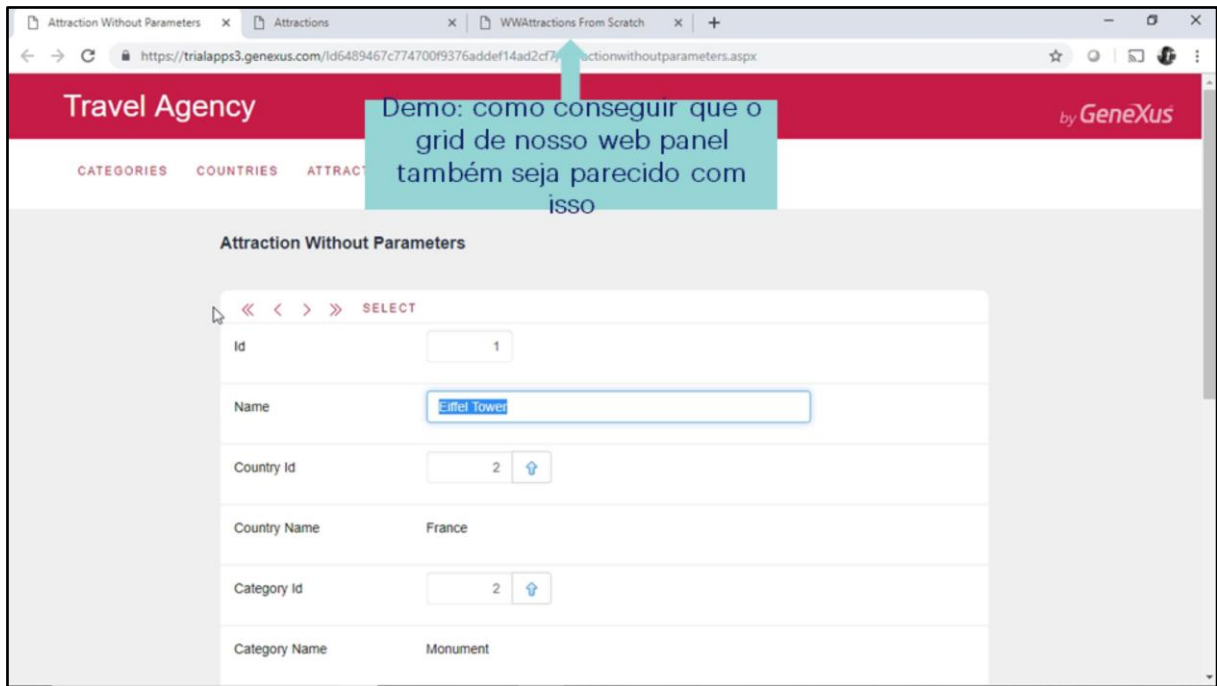
Se formos ver as opções Update ou Delete do pattern Work with, que sabemos terem a color base em execução, vemos que o tipo de controle é Attribute/Variable. E que tem atribuída esta como classe, predefinida. Podemos aqui mesmo editar suas propriedades sabendo que o que modificamos aqui será modificado no nível central, no objeto Theme, então qualquer outro controle atributo/variável que tenha esta como classe, será modificado.

Em suma, todo controle em um form tem propriedades individuais, que podem ser configuradas apenas para ele... E propriedades que virão da classe que tenha atribuída a ele, que são aquelas que envolvem, mais do que qualquer outra coisa, os aspectos de design.

Já observamos que neste caso particular em que o controle é, além disso, uma coluna de um grid, aparece outra classe para associar a ele, que é a classe que governa o design e comportamento da coluna.



E se observamos novamente o design predefinido da transação e o Work With, vemos que os dados mais relevantes são mostrados em branco sobre cinza, com bordas arredondadas.



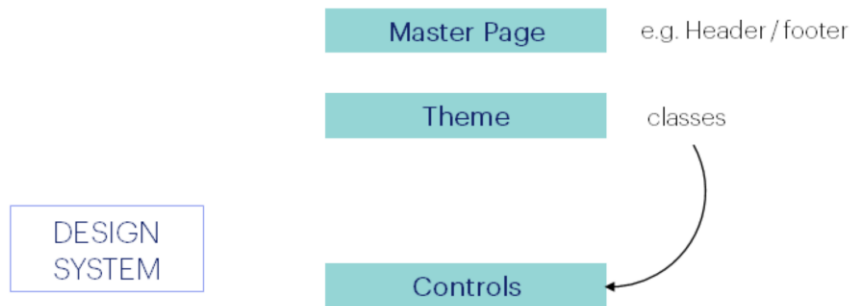
[DEMO: <https://youtu.be/XhDqve5E74>]

Isso ocorre porque a tabela onde esses dados se encontram na transação foi atribuída esta classe... que especifica como cor de fundo a branca e um raio para as quatro bordas...
...e para o grid do Work With esta outra classe, que também especifica branco como cor de fundo, e os mesmos valores que a outra para o raio.

Por outro lado, se formos ver o grid do web panel que nós criamos, vemos que tem como classe a de nome Grid, que não tem nada disso configurado... E assim é visto em execução, com todas as colunas coladas, com o fundo default, cinza. Observemos o que acontece se mudarmos sua classe para a mesma que tem o grid WorkWith...

E para manter a uniformidade do Design System, esta ação do grid deveria se parecer com estas outras...

E aqui não deveria estar o título da coluna...



Podemos deduzir do que vimos que uma porcentagem muito importante do Design System é especificada no objeto Theme, através de suas classes (as predefinidas e as que adicionamos). Estas classes são aquelas que são atribuídas aos controles do form, permitindo assim abstrair o design, estabelecer uma lógica e desta maneira padronizá-lo de acordo com a função que assume cada controle no todo.

Portanto, uma parte fundamental do design será identificar quais classes serão necessárias para os controles e aplicá-las.

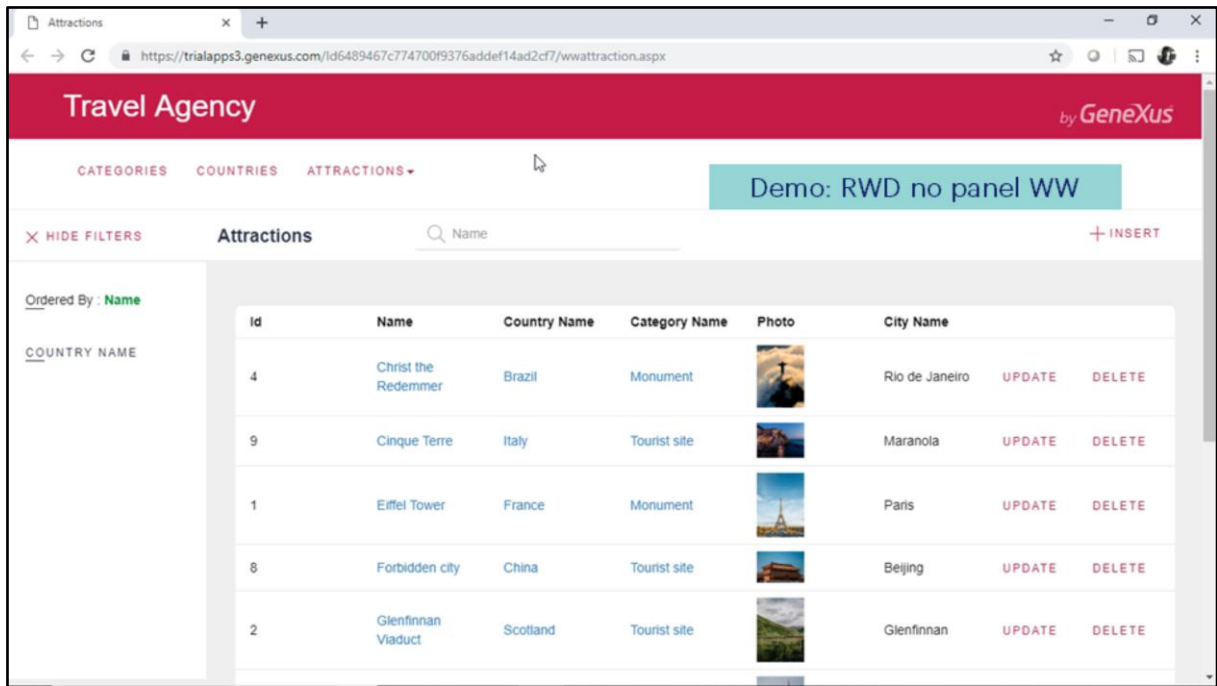
Responsive Web Design

Backend

- Responsive Sizes das Responsive tables

The screenshot shows the GeneXus IDE interface for designing a web form. The main design area displays a form titled 'Attractions' with a table and filters. The table has columns: Id (AttractionId), Name (AttractionName), Country Name (CountryName), Category Name (CategoryName), Photo, City Name (CityName), &Update, and &Delete. The filters section includes 'Ordered By' with 'Name' and 'Country' options, and a 'Country Name' filter with the value '&CountryName'. The 'Responsive Sizes' panel on the right shows the 'Table: TableTop' selected with a size of 'Extra small (Phone < 768 px)'. The panel also shows a table with columns: 1.1 BtnT, 1.2 TitleText, 1.3 Actions, and 1.4 &AttractionName. The 'Values' section shows 'Width: 100%', 'Offset: 0%', 'Visible: True', and 'Move: < >'. A blue arrow points from the 'Responsive Sizes' panel to the table in the design.

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



[DEMO: <https://youtu.be/aRtwlstT7BU>]

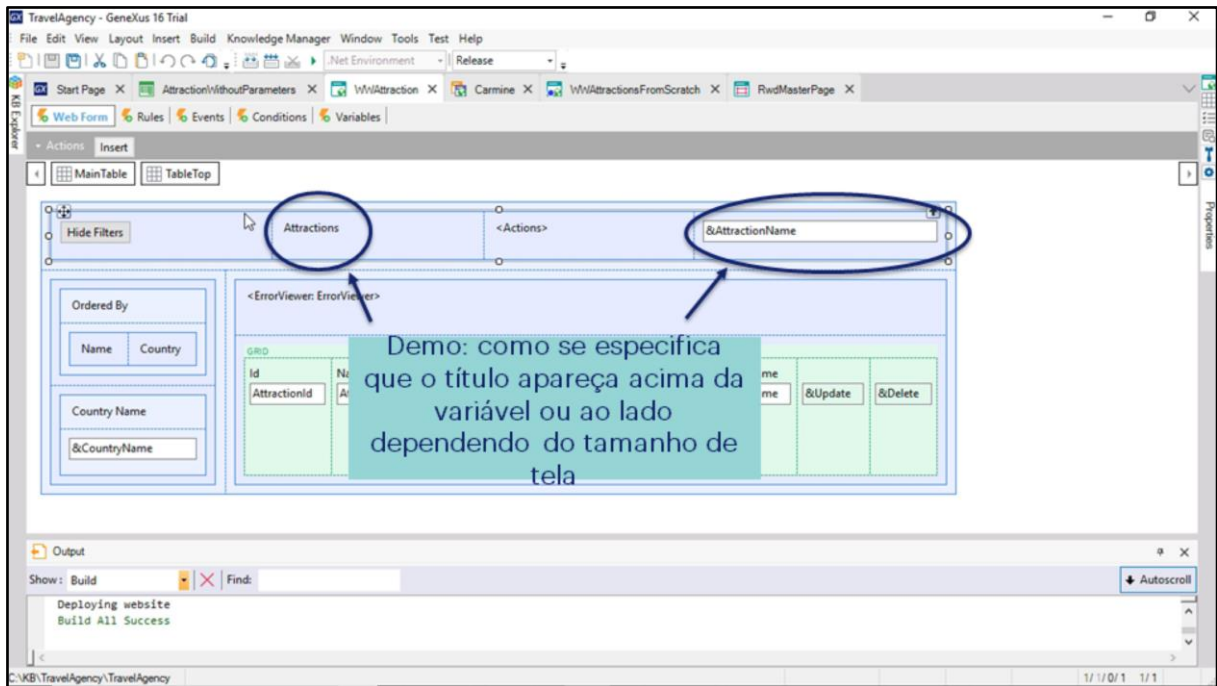
Voltando ao predefinido (lembramos que alguns navegadores nos oferecem um modo de visualização –neste caso pressionando F12– que nos permite variar o tamanho da tela para ver como ficaria)...

Podemos ver que, se reduzirmos a tela do Work with de atrações o suficiente para corresponder à tela de um telefone, os filtros serão exibidos de forma diferente, e o menu que adicionamos aparecerá como o menu típico de hambúrguer. E, por exemplo, o campo para o qual o usuário possa inserir um filtro por nome de atração aparece abaixo e não ao lado.

E com a diminuição da largura da tela, quase todas as colunas também desaparecem, com exceção da que mostra o nome e as ações de Update e Delete..

Havíamos nomeado este comportamento como “responsivo” e ao design web que o considera como Responsive Web Design. Hoje em dia, não é aceitável um design que não seja responsivo.

Uma parte da responsividade é estabelecida através das tabelas responsivas...



[DEMO: <https://youtu.be/qsbLGSHDzHM>]

Por exemplo, esta tabela, que vemos que é responsiva, contém estes quatro controles. O segundo corresponde a este textblock, e o último corresponde à variável de filtro que vimos. Se editarmos as propriedades da tabela veremos esta, "Responsive Sizes" (tamanhos responsivos). Aqui se estabelece como se deseja visualizar o conteúdo da tabela de acordo com quatro tamanhos de tela: **Extra small**, que corresponde a telefones, geralmente com largura da tela menor que 768 pixels; **Small**, que corresponde a tablets com largura maior ou igual a 768 pixels; **Medium** que corresponde a monitores de notebooks ou Pcs maiores que 992 pixels e menores que 1200 pixels; porque para os maiores ou iguais existe o tamanho **Large**.

Aqui podemos ver que se o tamanho for **Extra small**, a variável aparecerá abaixo do título; enquanto que se o tamanho for **Small**, a variável aparecerá à sua direita.

Dissemos que uma parte importante da responsividade, a maior, é obtida fazendo uso destas tabelas e estas propriedades.

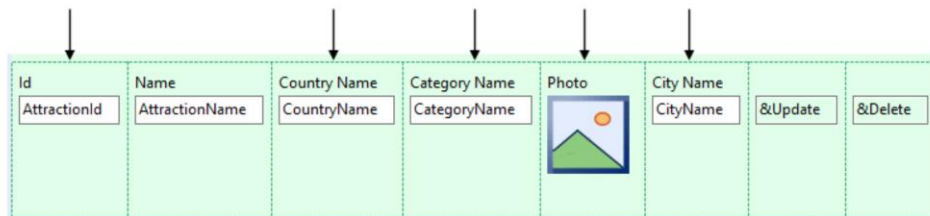
Mas outra parte, um pouco menor, é alcançada através das **classes**.


Responsive Web Design

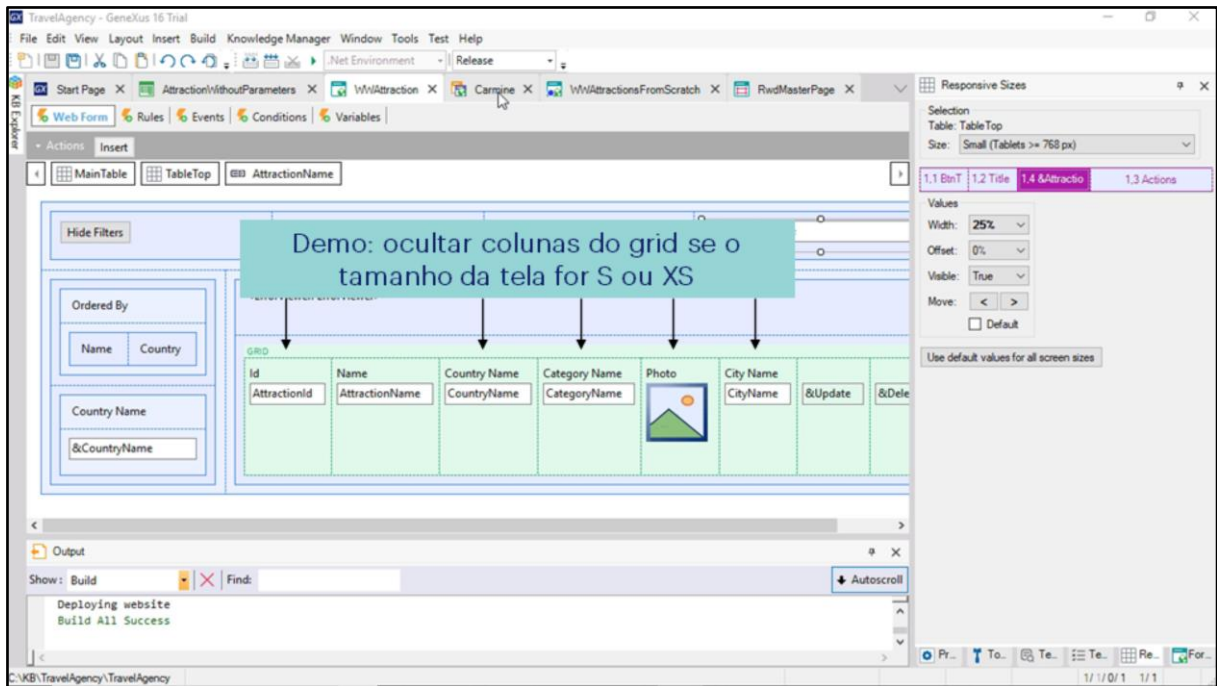
Backend

- Responsive Sizes das Responsive tables
- Rules do Theme: Small, Extra-small e Default

Ex: ocultar colunas do grid se tamanho Extra small ou Small. Para tamanhos maiores não.



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



[DEMO: https://youtu.be/lKGkv7VM_ms]

Por exemplo, ocultar colunas do grid dependendo do tamanho da tela.

Vemos que a classe da coluna do nome da atração é WWColumn, mas que para as outras colunas é adicionada, além disso, a classe WWOptionalColumn.

Se formos ao theme, vemos que estão aparecendo estas duas colunas, que o que fazem é nos permitir variar os valores das propriedades da classe, de acordo com o tamanho da tela.

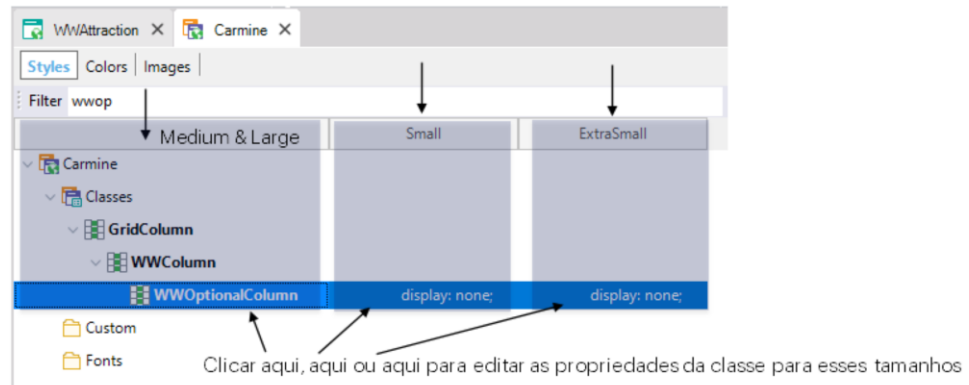
Por padrão, apenas dois são definidos: Small e ExtraSmall. Mas na verdade temos três, já que os valores que vemos na coluna default correspondem a Medium e Large.

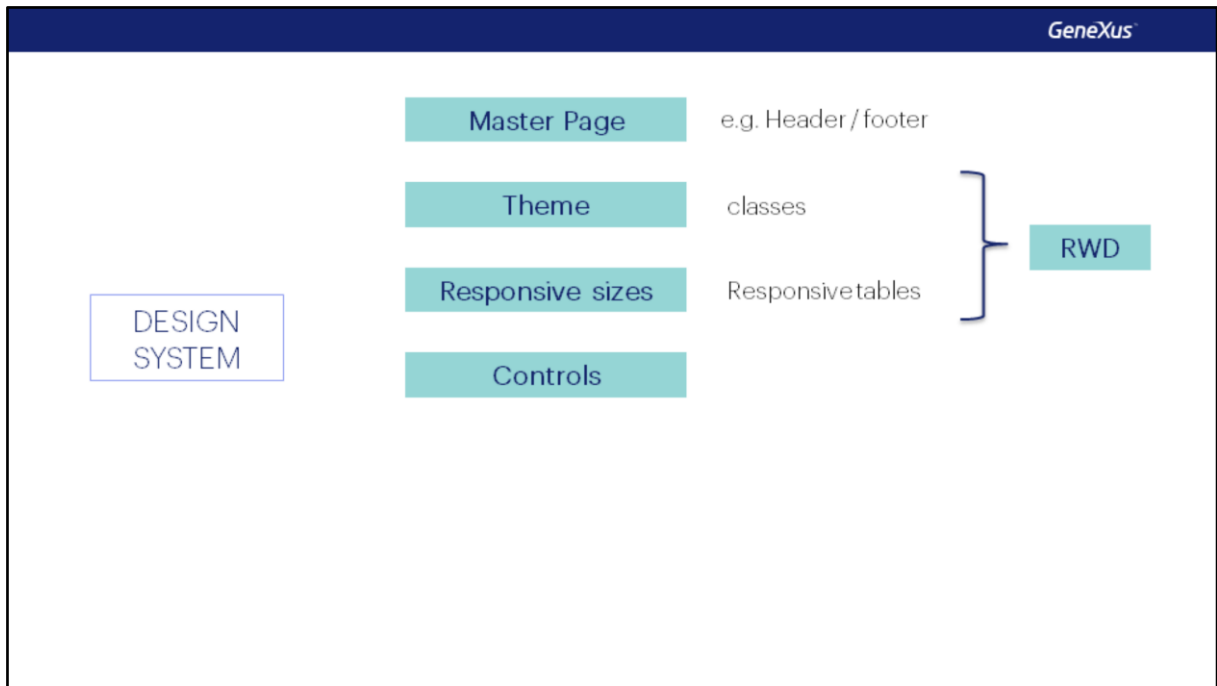
Então, aqui podemos ver que, enquanto a propriedade Display para esta classe não diz nada, o que significa que será exibida em execução para tamanhos de tela Medium e Large, para tamanho Small e ExtraSmall, tem o valor "none". E é por isso que não são exibidas.

Responsive Web Design

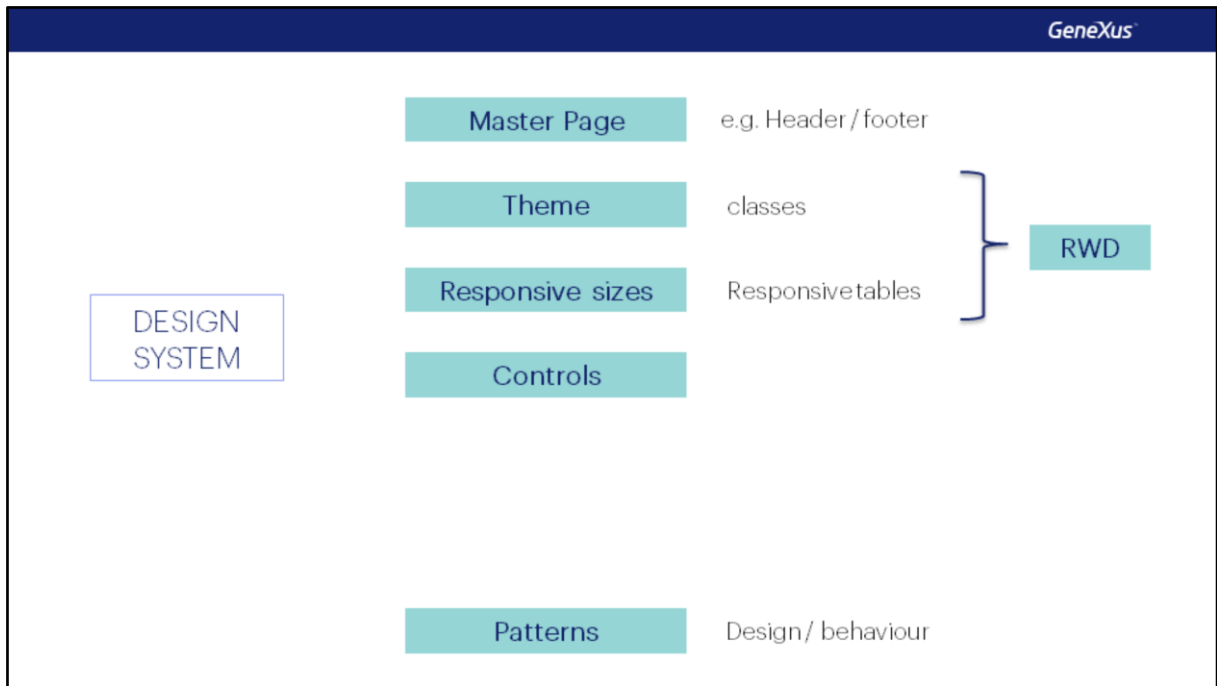
Backend

- Rules do Theme: Small, Extra-small e Default





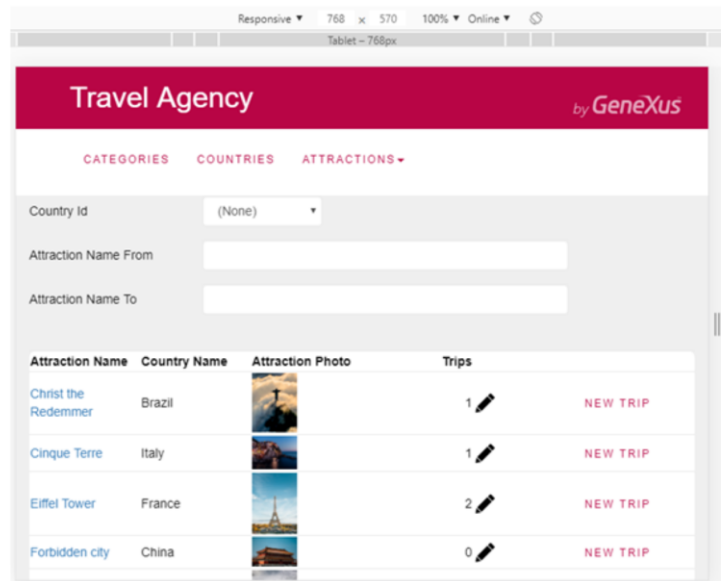
Em resumo: outra parte importante do Design System tem a ver com a responsividade, que é obtida tanto variando as posições e a visibilidade dos controlos das responsive tables, como variando as propriedades das classes de acordo também com o tamanho (através dessas colunas do Theme que vimos).



Para conseguir tudo isso, não tivemos que fazer nada. O pattern Work With fez tudo automaticamente, em conjunto com o theme. Por isso que dizemos que GeneXus já nos fornece um Design System básico, predefinido, que podemos utilizar para nossos painéis.

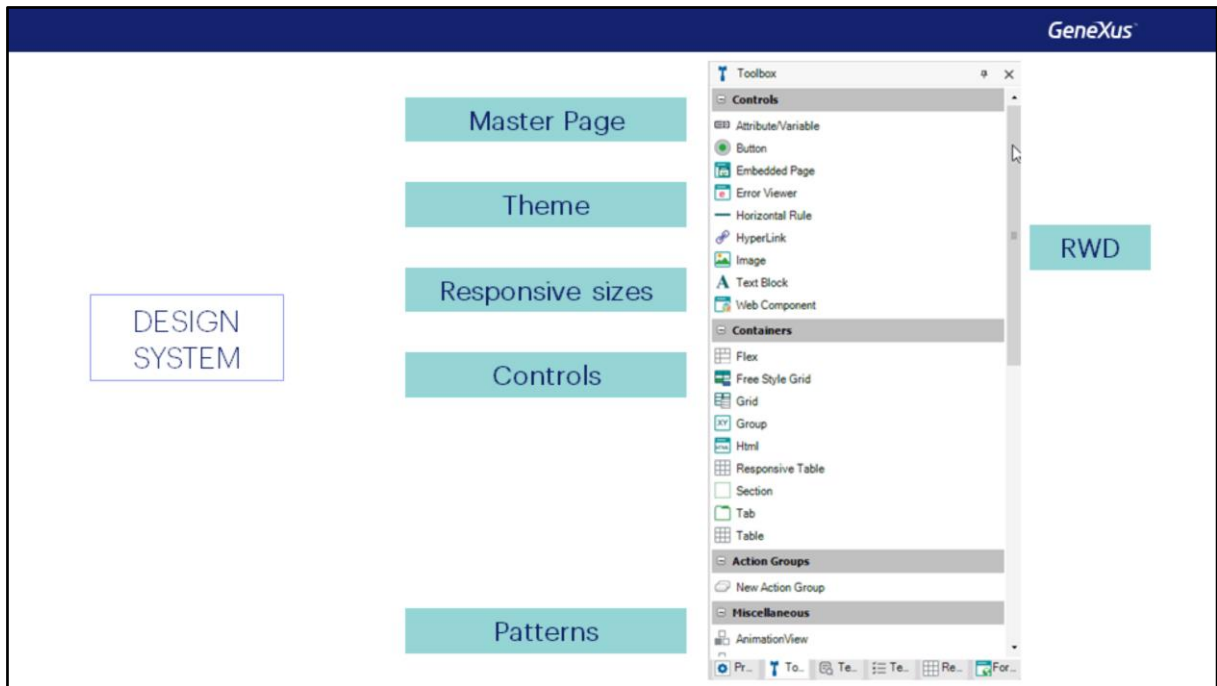
Tarefa

- Tente ocultar colunas do grid do web panel que criamos do zero



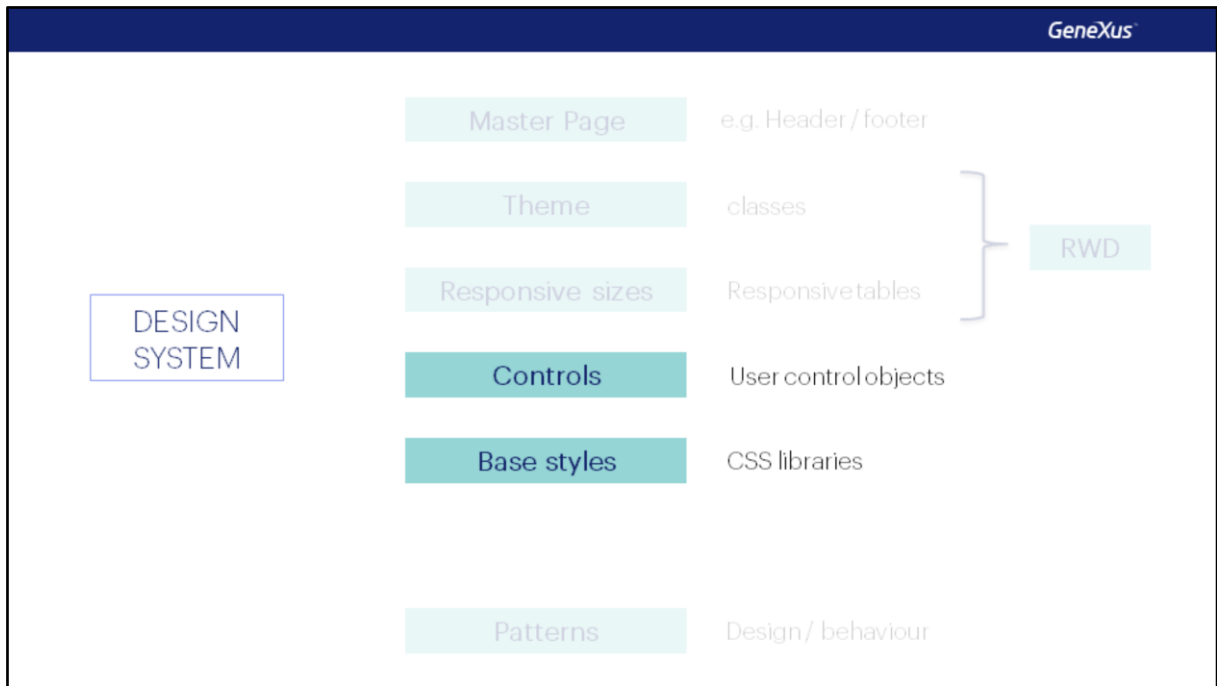
Por exemplo, como conseguimos que o painel que implementamos do zero tenha um comportamento responsivo similar ao do pattern? Nós o convidamos a alcançá-lo.

Há muito a dizer sobre este assunto. Aqui vamos nos contentar com esta introdução.

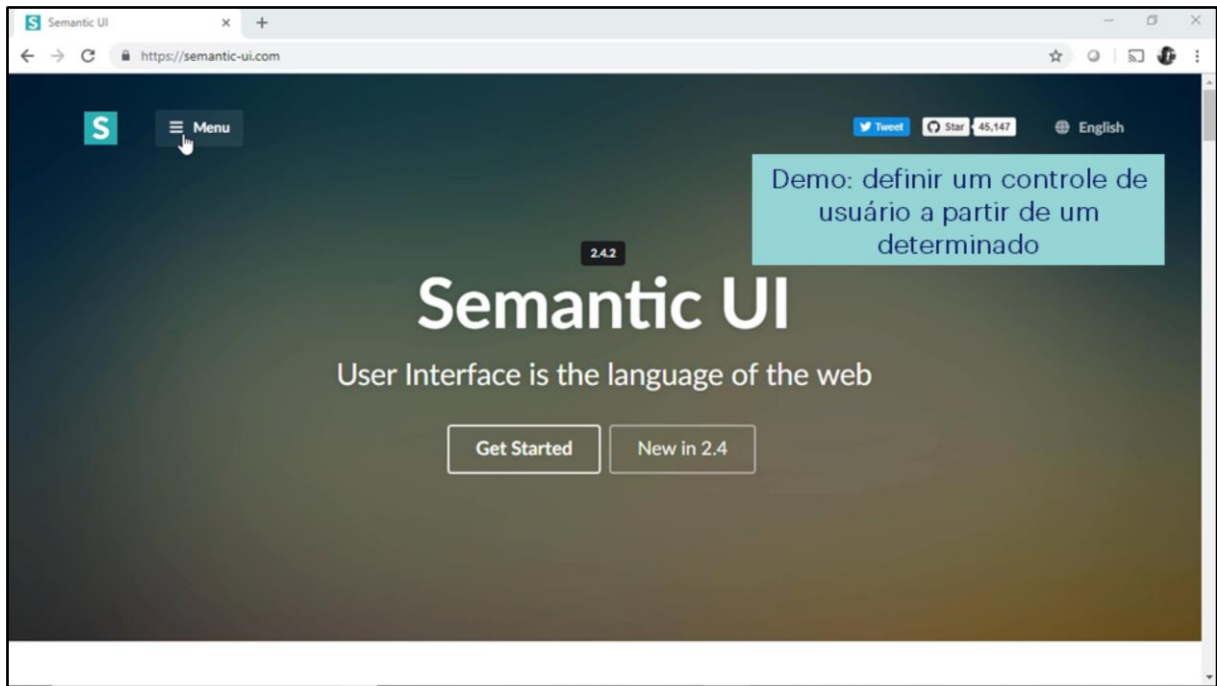


Existem mais alguns atores que, juntamente com todos os vistos, permitem utilizar no GeneXus um Design System potente. Não vamos estudá-los neste nível do curso, mas os deixamos apresentados.

Podemos não apenas utilizar em nossos forms os controles que vêm na toolbox do GeneXus de forma predeterminada...



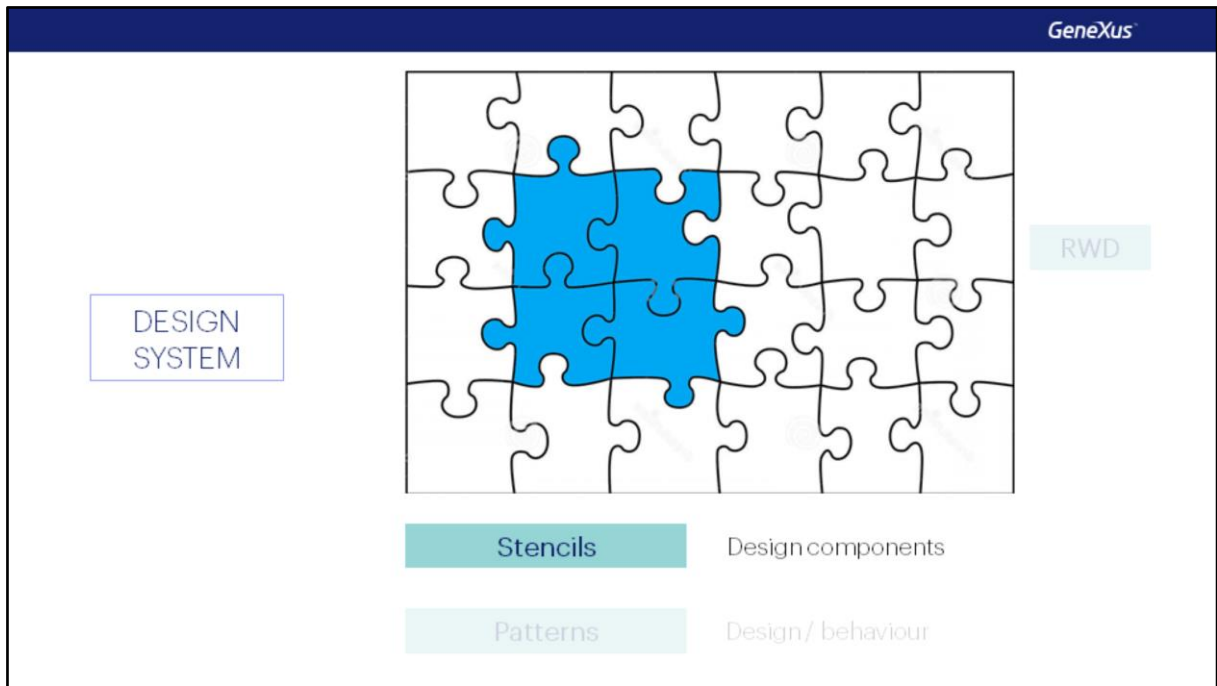
...mas também podemos definir controles de usuário, que podem ser copiados de plataformas que oferecem esses controles junto com bibliotecas CSS (que são, como nossos themes, aquelas que especificam seu estilo –o estilo desses controles).



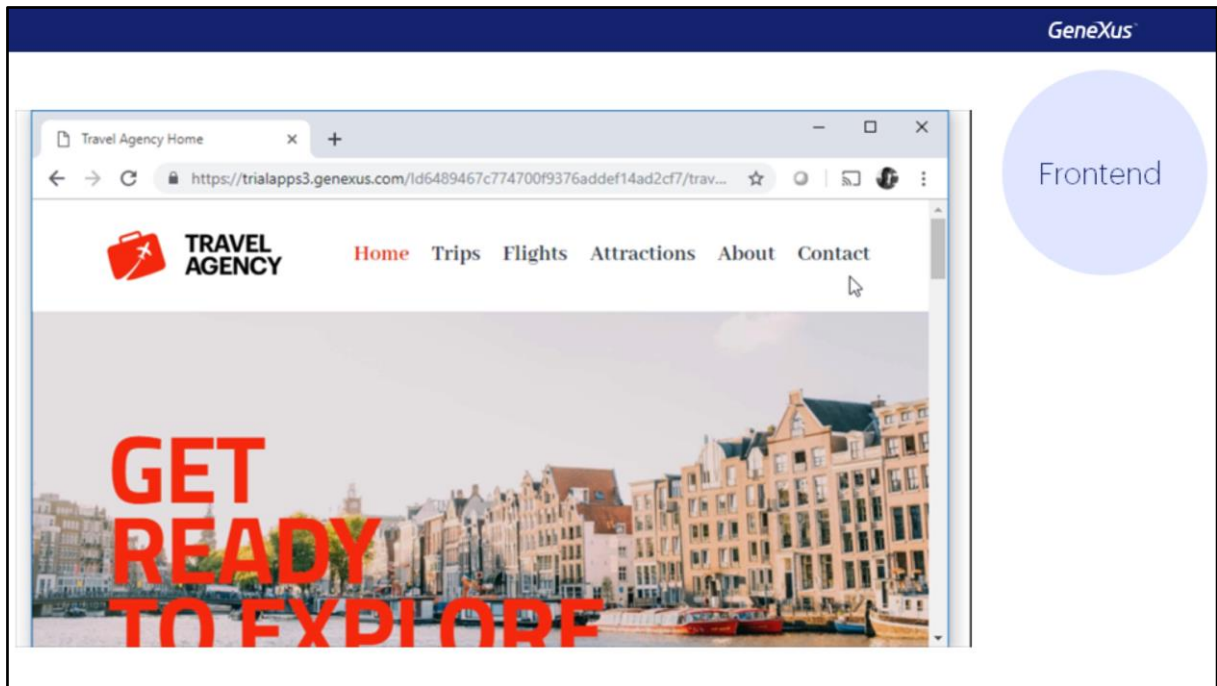
[DEMO: DesignSystemsInGX_demo10.mp4]

Por exemplo, esta, Semantic UI. Podemos querer usar um controle Card como este.

Para isso, conseguirá criando um objeto User control (podemos chamá-lo igual), copiar e colar seu html, alterar os dados fixos por algo como nomes de elementos de um SDT (para torná-lo dinâmico, ou seja, ser capaz de carregar dinamicamente esse controle, com dados que poderemos especificar, por exemplo, da base de dados), especificar de onde tomará o CSS, ou seja, o design das classes, fazer alguma coisa mais e já o teremos disponível na Toolbox para utilizá-lo.



Por outro lado, GeneXus oferece Stencils, que servem para repetir o design de uma mesma parte de tela (um conjunto de controles), em muitas telas. É uma maneira de abstrair o design em um nível superior. Se pudermos pensar nas telas de uma aplicação web como quebra-cabeças compostos de peças atômicas (que seriam os controles), os stencils são uma forma de agrupar um conjunto de controles cujo design se repetirá em várias telas. Seria uma peça feita de peças menores.



[DEMO: <https://youtu.be/X4fb58cxmrA>]

Mencionaremos exemplos no próximo vídeo, onde colocaremos estes conceitos vendo o frontend da Travel Agency.

GeneXus™

The power of doing.

More videos
Documentation
Certifications

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