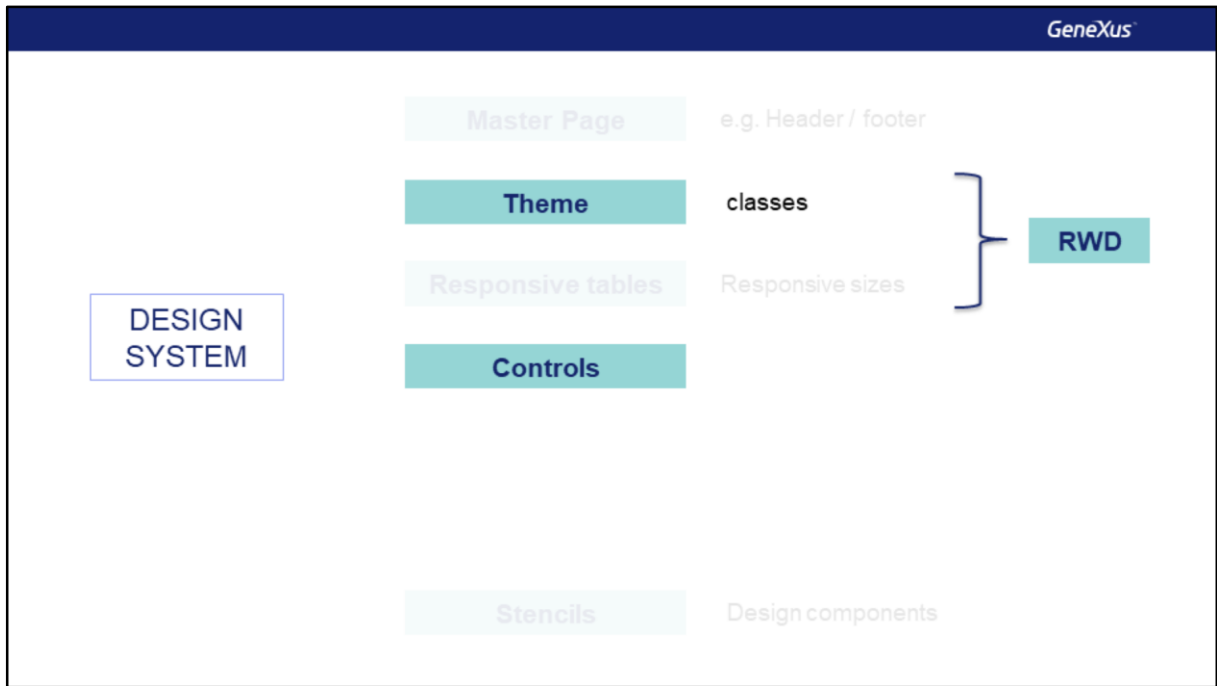


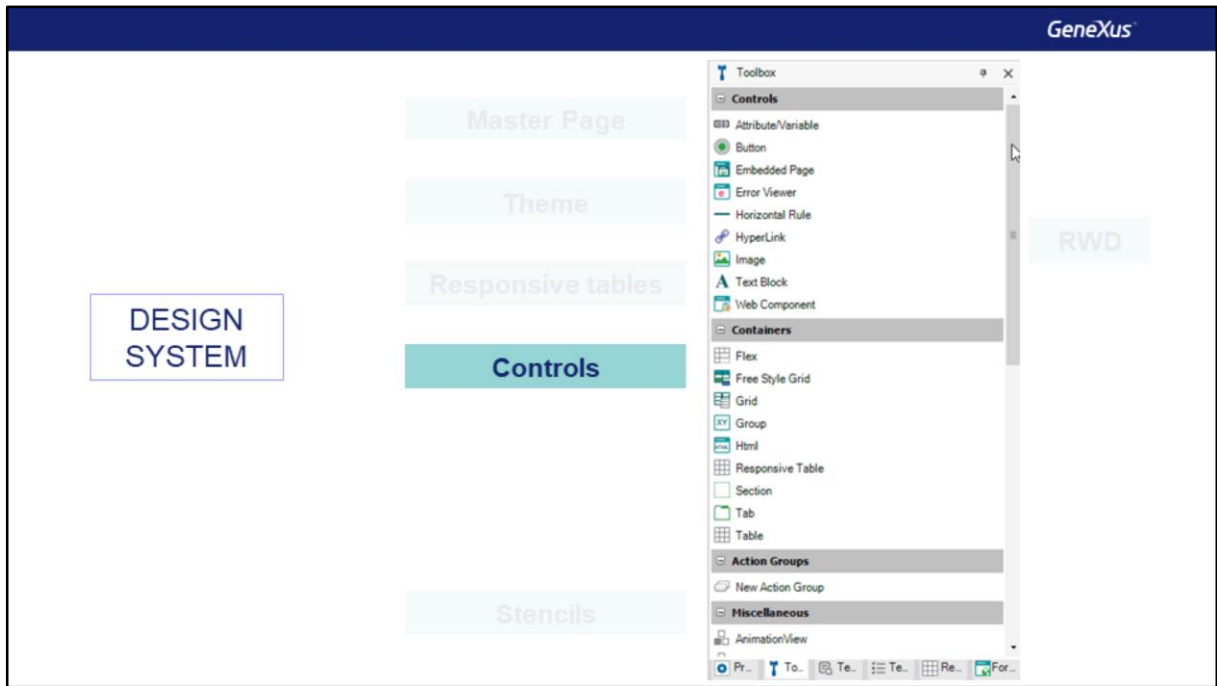
Design System en GeneXus

Frontend: GeneXus and User Controls

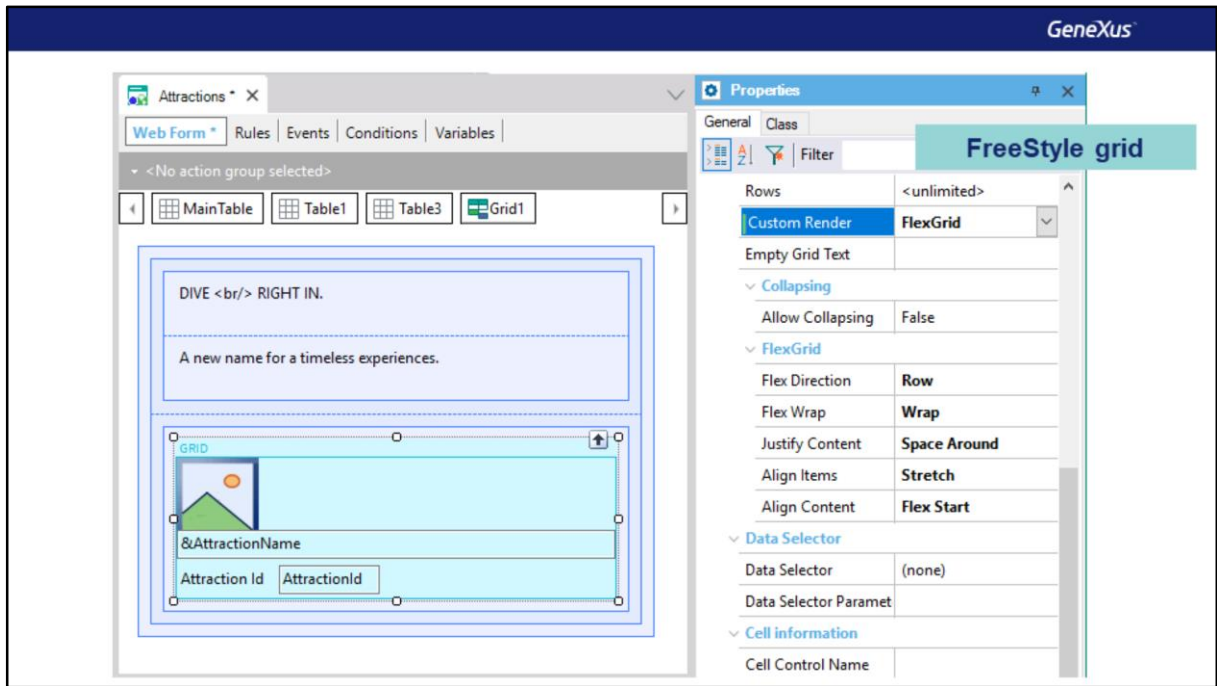
GeneXus 16



Hemos visto la importancia de las clases para los controles, en la implementación del Design System.



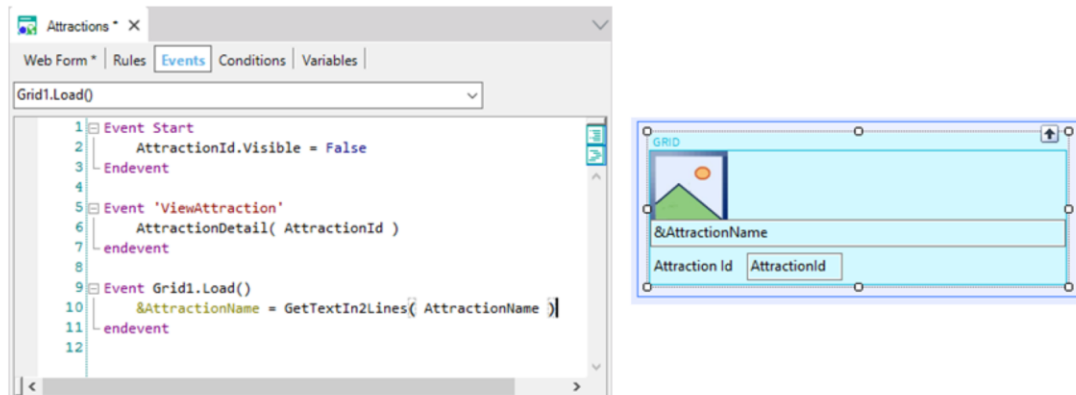
Hasta aquí hemos trabajado casi exclusivamente con los controles que nos ofrecía la toolbox de GeneXus. Tenemos controles simples, como el text block, o el atributo/variable, o más complejos: por ejemplo aquellos que contienen otros controles, como los Grids.



Así, en el web panel que lista las atracciones turísticas, vemos que hemos implementado al grid de atracciones como un grid Freestyle, para que cada “línea” cargada pueda mostrar su información de forma más libre, sin tener que estar una al lado de la otra en columnas, como es el caso del grid estándar.

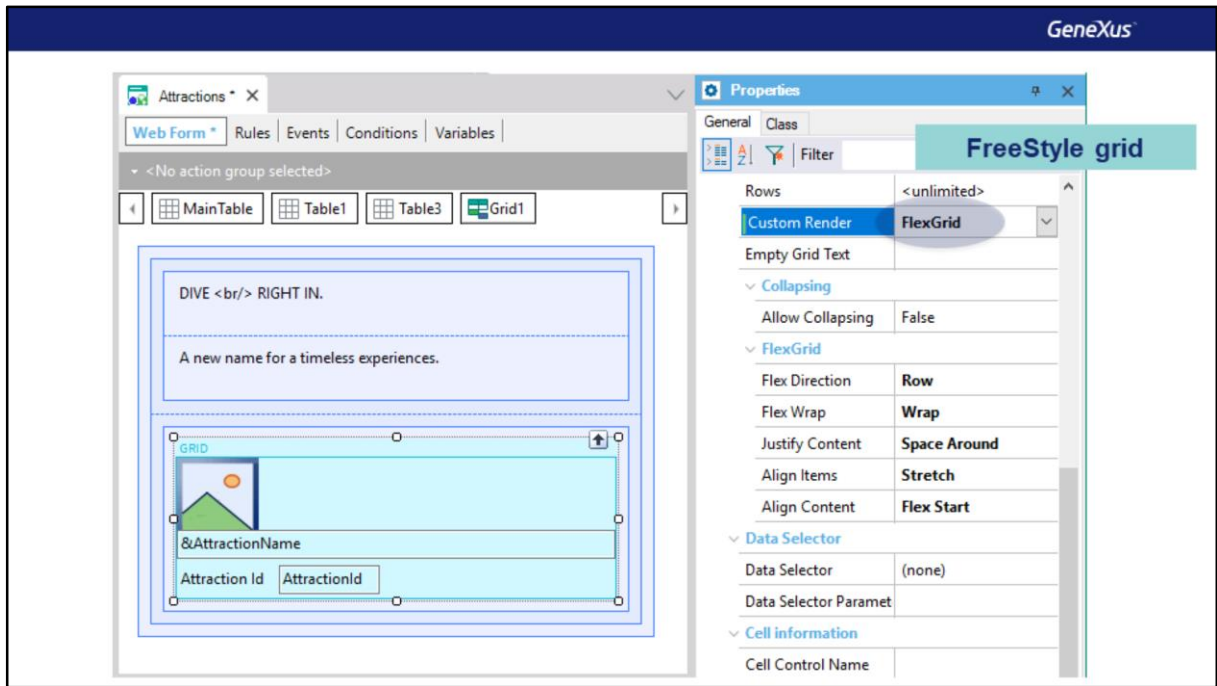
Carga y determinación de tabla base idéntica a Grid estándar

FreeStyle grid



La carga del grid es igual. Así como la determinación de su tabla base. Todo eso es idéntico.

Aquí vemos que en el Load estamos cargando el nombre de la atracción turística (porque tenemos un procedimiento que la parte en dos líneas). El grid tendrá tabla base Attraction. Aquí vemos el atributo AttractionPhoto. Y aquí el AttractionId, oculto.



Pero además, indicamos que ese grid freestyle es un Flex Grid. Con esto conseguimos el efecto de autoajuste de las imágenes dependiendo del tamaño de pantalla.

GeneXus

Attractions * X

Web Form * Rules Events Conditions Variables


<No action group selected>

MainTable Table1 Table3 Grid1

DIVE
 RIGHT IN.

A new name for a timeless experiences.

GRD

 &AttractionName

Attraction Id AttractionId

Properties

General Class Filter

FreeStyle grid

Rows <unlimited>

Custom Render HorizontalGrid

Columns 1 2 3 3

Extra Small 1

Small 2

Medium 3

Large 3

Empty Grid Text

Collapsing

Allow Collapsing False

HorizontalGrid

Paged True

Show Page Controller True

Page Controller Class GridPageController

Show Arrows True

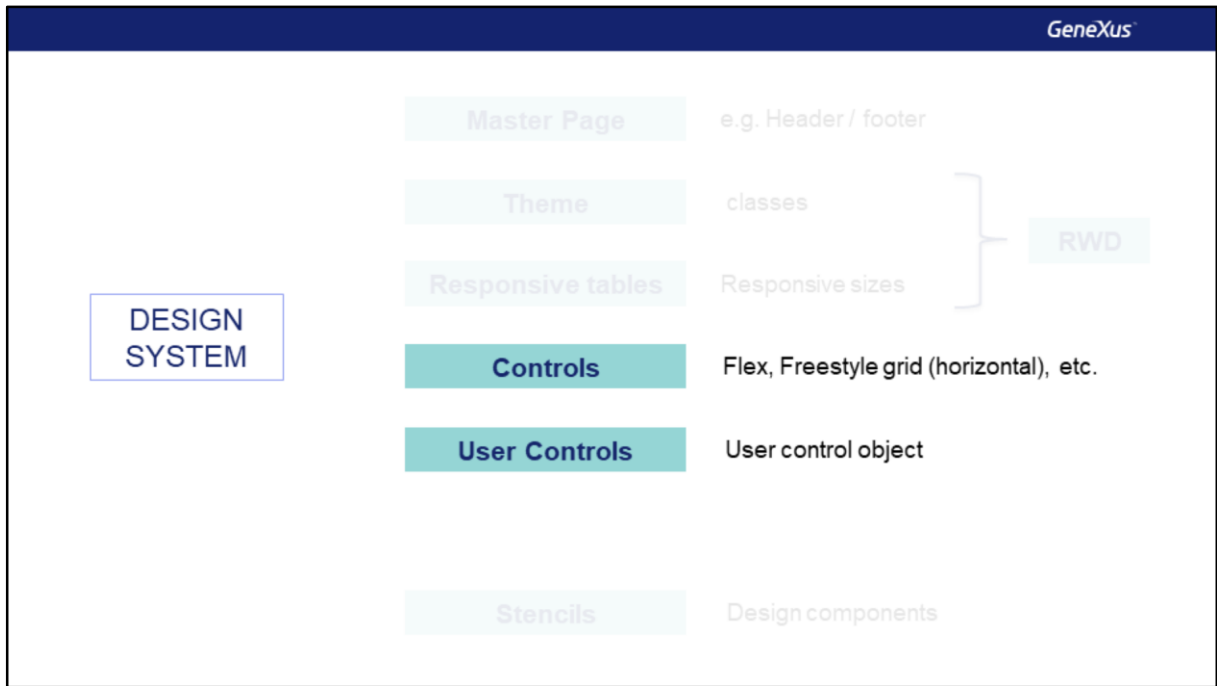
Infinite False

Auto Play False

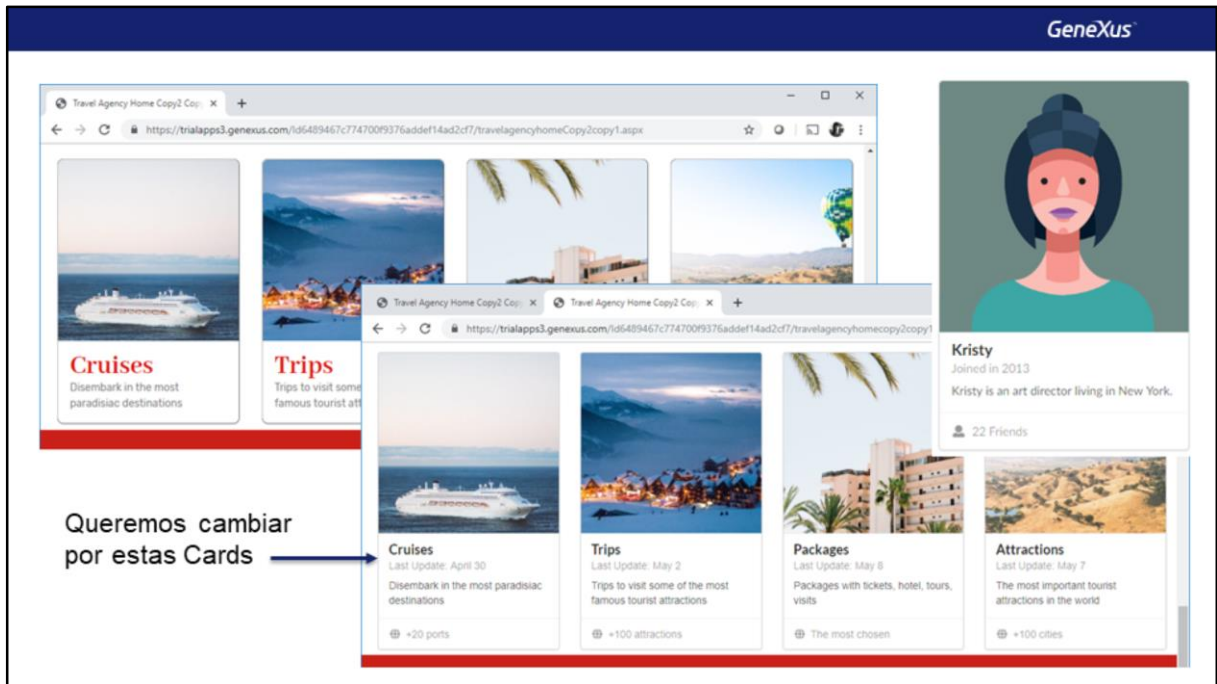
Variable Width False

Rows per page 1

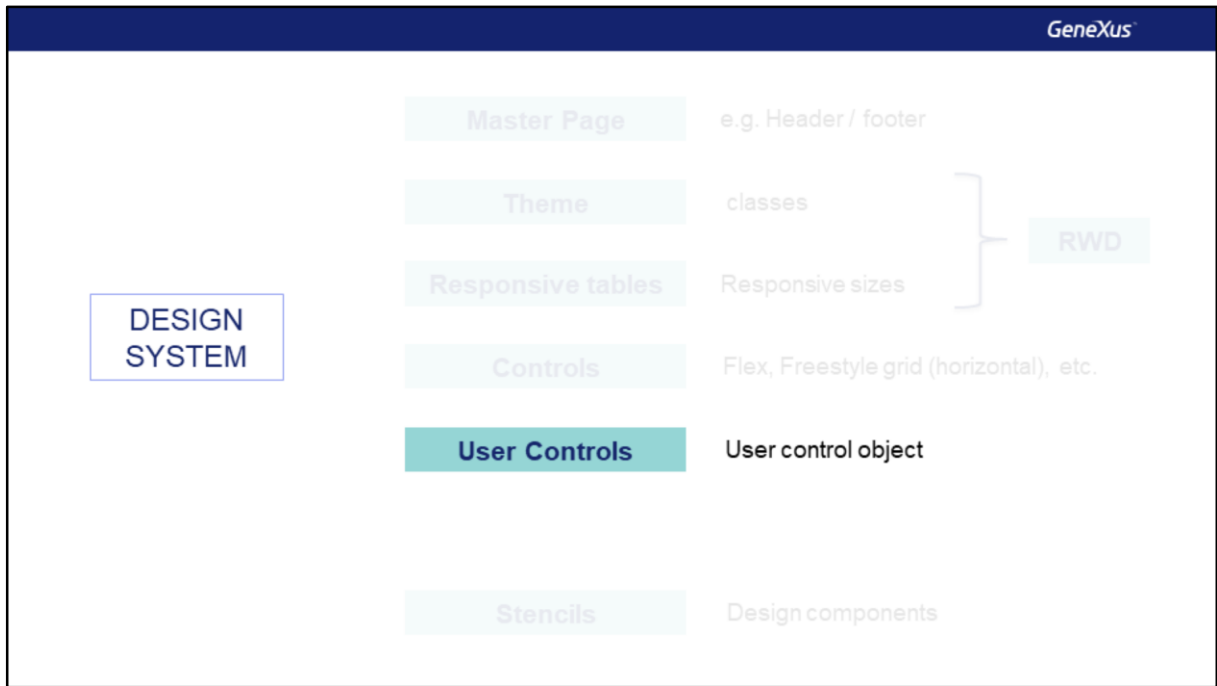
También podríamos haberlo implementado como un grid horizontal, con tres columnas por página para tamaño Medium y Large, 2 para Small y 1 para Extra Small... Ejecutemos...



Pero podemos utilizar en nuestros forms otros controles, aparte de los que vienen en la toolbox de GeneXus en forma predeterminada...



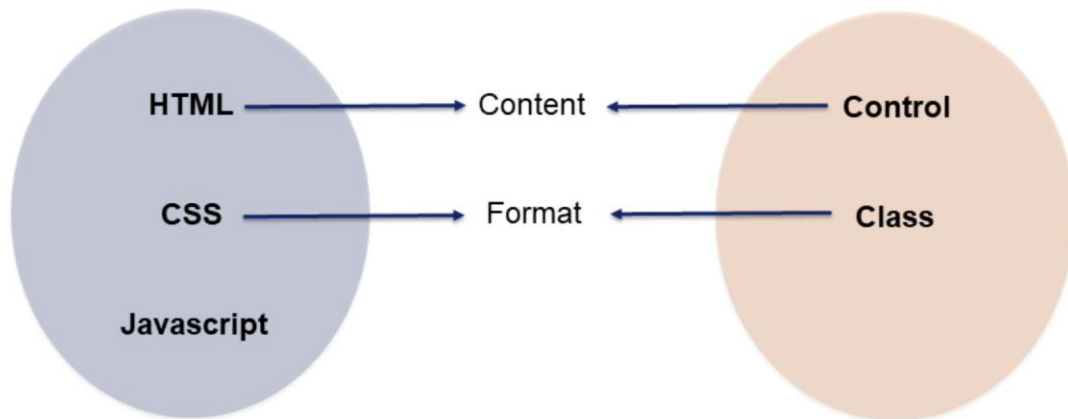
Por ejemplo, vamos a querer modificar nuestro web panel home, para que ahora, en vez de mostrar esta suerte de tarjetas que habíamos diseñado trabajosamente a mano en el stencil, definiendo una por una las clases que permitían mostrar los bordes redondeados y demás... utilice estas otras, que ya vienen predefinidas en la plataforma SemanticUI.



Para ello lo que haremos será definir un control de usuario en nuestra KB, que copiaremos de esa plataforma, para utilizarlo en nuestro form.

La plataforma nos ofrece un conjunto de controles junto con bibliotecas CSS (que son, como nuestros themes, las que especifican su estilo –el estilo de esos controles-).

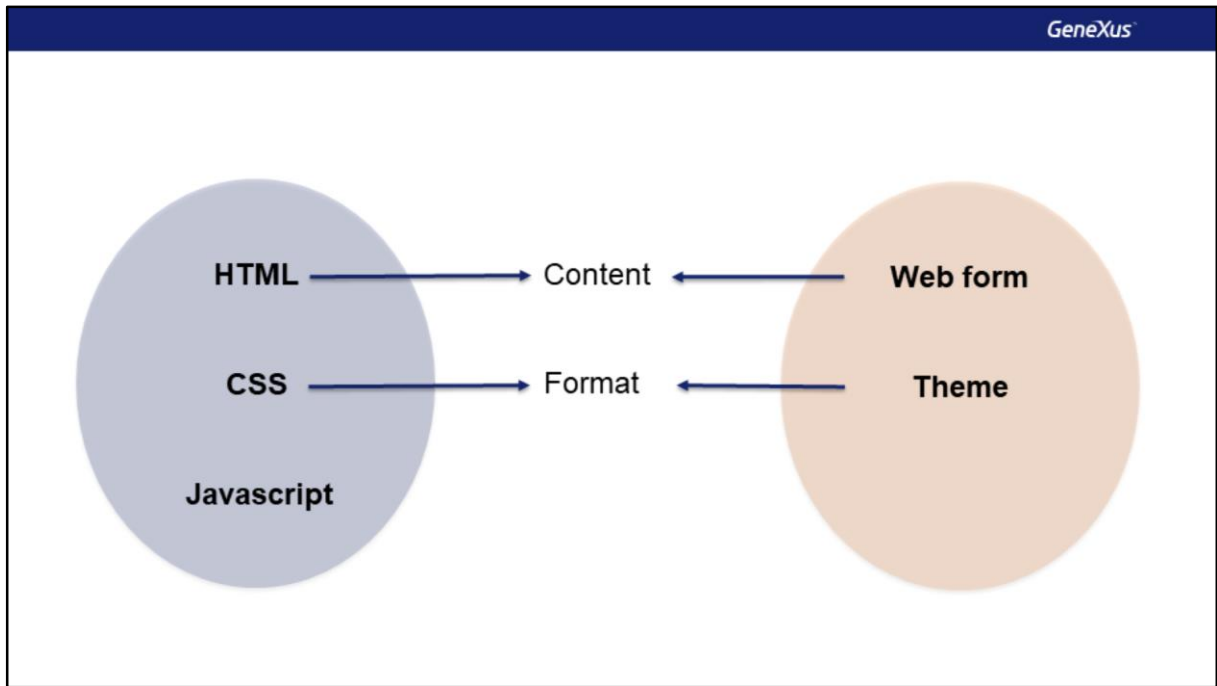
Aclaremos un poco antes de continuar. Vayamos a las bases...



Las tecnologías utilizadas por la gran mayoría de los sitios web son un lenguaje marcado (como el html), un lenguaje de estilos: Cascade Style Sheets y Javascript.

Hace ya mucho tiempo que en el mundo web se ha separado el contenido de la página de lo que es su formato de presentación. Por eso tenemos el archivo en formato HTML que será interpretado por el navegador para mostrar la información en la página pedida por el usuario, y aparte, el o los archivos CSS, Cascade Style Sheets (en español, hojas de estilo), que determinan mediante reglas cómo deberá mostrarse esa información en la pantalla (colores, fuentes, alineación, tamaño, etc).

Claramente es la misma separación que en GeneXus se establece entre el control y su clase.



Es decir, entre el form del objeto web y el Theme que contiene las clases que permiten mostrarlo de una manera determinada.

De hecho, el theme se convertirá en CSS a la hora de generar la aplicación.

CSS Frameworks

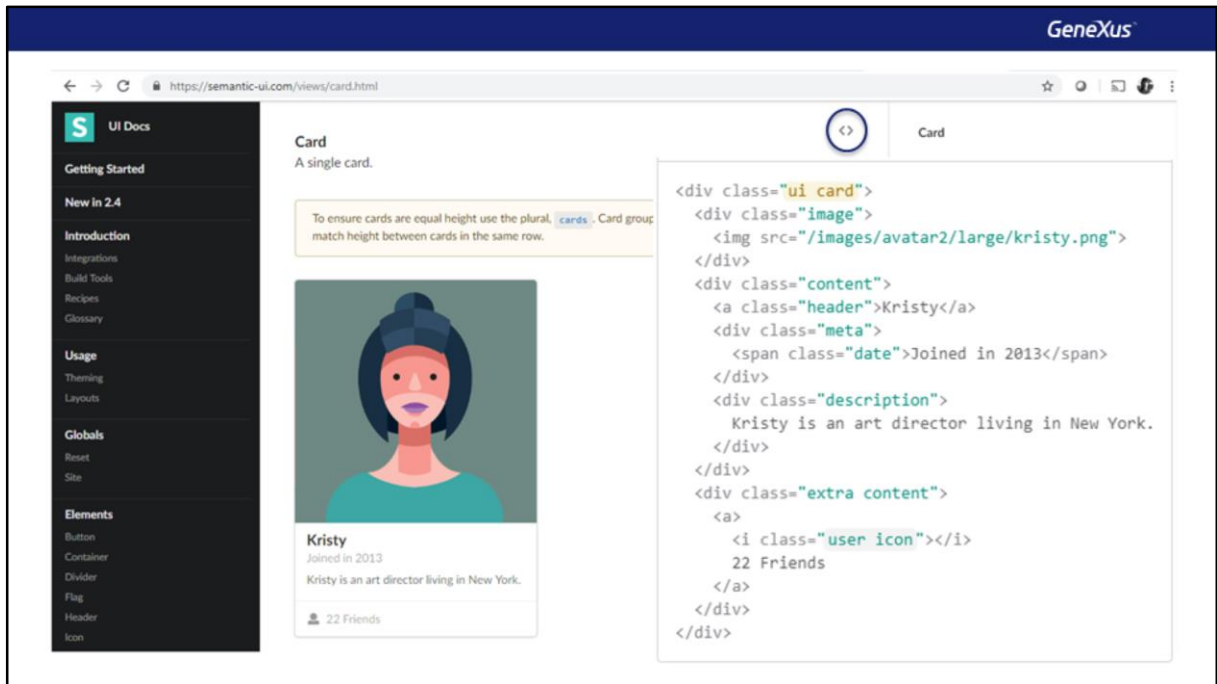


etc...

¿Qué son los frameworks CSS?

Son bibliotecas de CSS preparadas para permitir la simplificación y el mejor cumplimiento de los estándares.

Aquí listamos solo algunas...



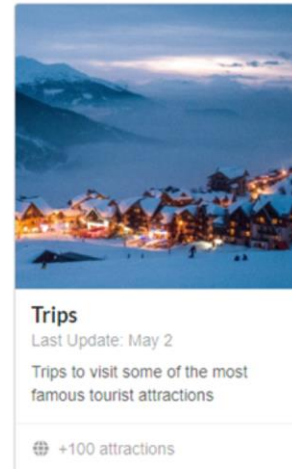
Veamos el ejemplo de Semantic UI. Como ya vimos, ofrece una serie de controles, junto con las bibliotecas CSS que manejan su diseño.

Por ejemplo, veamos que entre los controles que ofrecen vistas de información, están los de tipo Card. Y aquí vemos una serie de variaciones.

Observemos la primera opción: una tarjeta simple. Este control se compone de una imagen, y luego información textual, un ícono, y links.

Si cliqueamos aquí... nos muestra el código html que, como vemos, es el que define la información a ser mostrada, que está estructurada a partir de clases. Estas clases son las que definen, justamente, el diseño de la tarjeta. Están definidas dentro de hojas de estilo que SemanticUI distribuye.

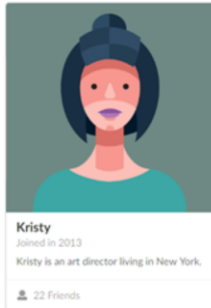
Control Card de SemanticUI en nuestra KB



En lugar de la que creamos a mano anteriormente, queremos utilizar esta Card.

Control Card de SemanticUI en nuestra KB

- Necesitamos dos cosas: el html y la biblioteca CSS de Semantic UI



User control object

+

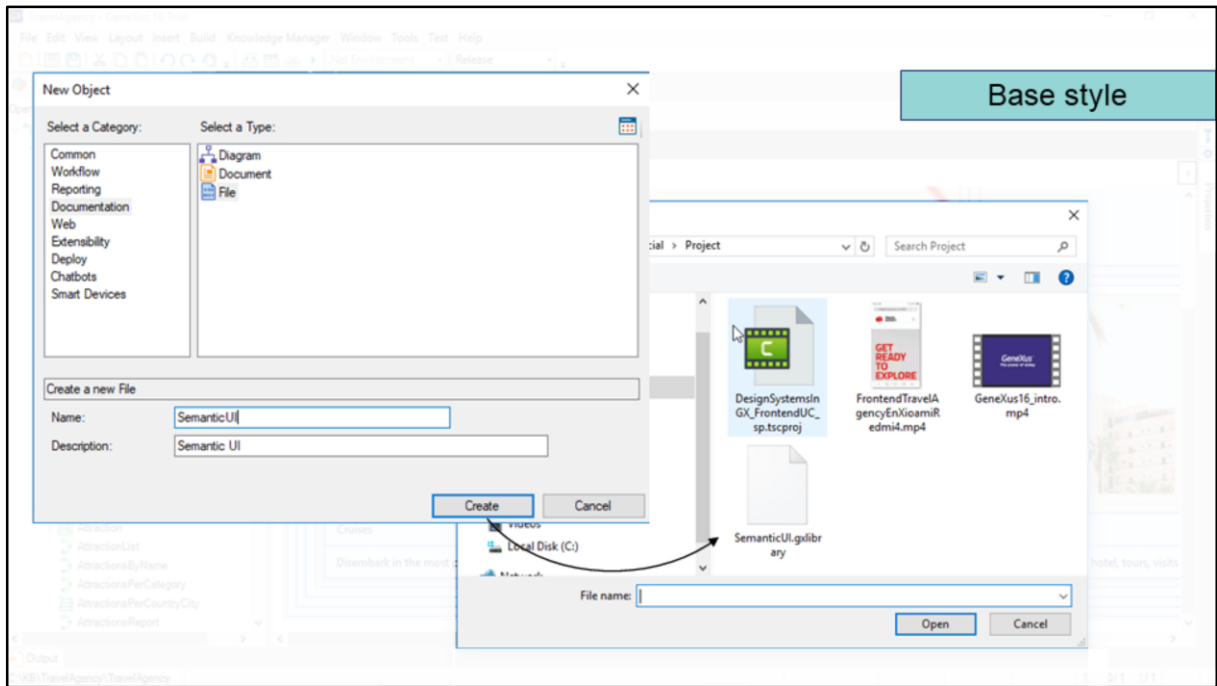
Base style

(File –con CSS- agregado a la KB)

```
<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>
```

Para ello, necesitaremos dos cosas: crear un objeto User control (en el que copiaremos el código html brindado por Semantic UI)

Y la biblioteca con el CSS que define el estilo de las clases que, entre otros, utiliza este control. Semantic UI distribuye esa biblioteca, así que tendremos que insertar ese archivo en la KB, para luego poder decirle a nuestro nuevo objeto que lo utilice.



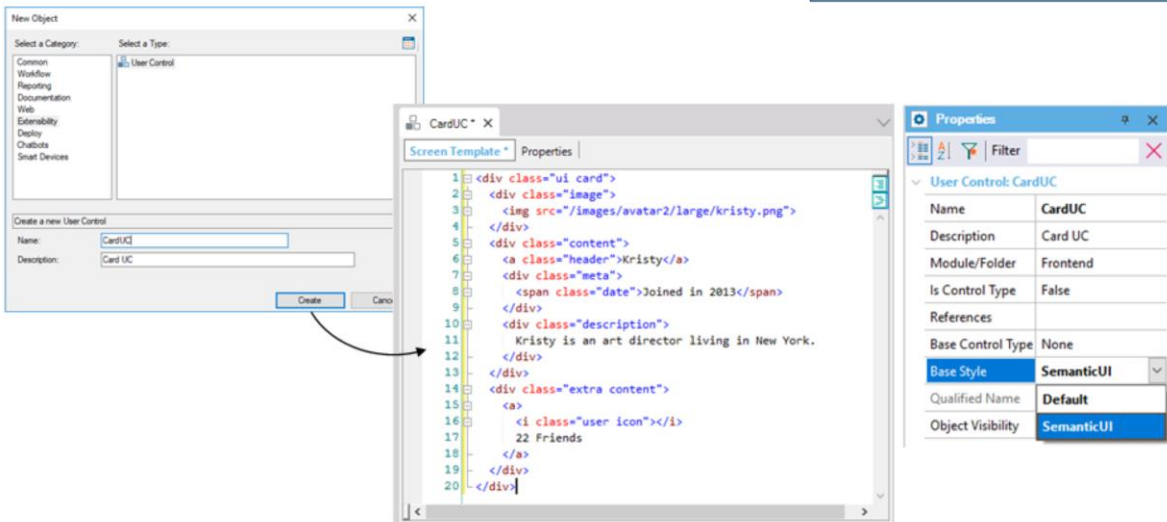
Entonces, empecemos por crear un objeto File en nuestra KB, al que le llamaremos SemanticUI.

Aquí lo buscamos. Debemos haberle colocado previamente extensión glibrary para que GeneXus lo reconozca como biblioteca y pueda luego ofrecerlo.

En la versión trial no tenemos disponible la ventana que muestra todos los archivos ingresados de este modo a la KB. Confiamos en que quedó ingresado correctamente.

Ya tenemos la biblioteca en la KB.

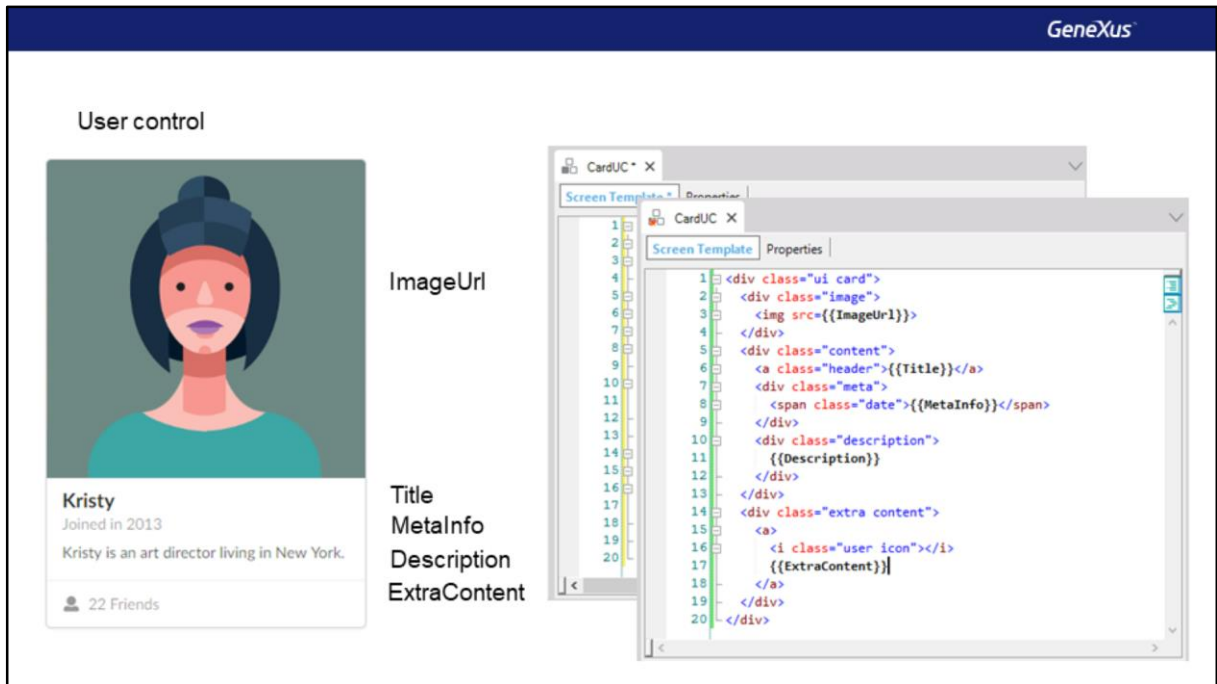
User control object



Ahora creamos el objeto User control. Lo encontramos bajo la categoría de extensibilidad.

Llamémosle CardUC.

Aquí pegaremos el html que copiamos de SemanticUI. Y ya le configuramos que tome el estilo base de la biblioteca que acabamos de insertar en la KB.

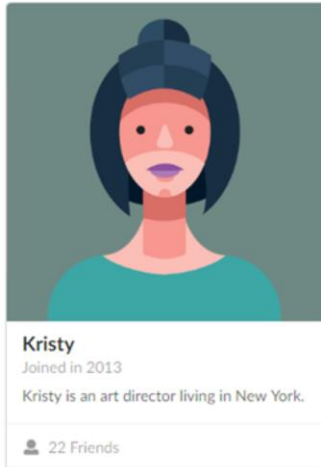


Lo siguiente que debemos hacer es sustituir la información del html, que es información hardcoded, es decir, fija, por información variable.

Aquí irá la url de la imagen, aquí un título, aquí la información que deseemos (por ejemplo, una fecha), una descripción, y contenido extra con un ícono.

Para poder convertir lo que en el html es contenido fijo en contenido variable, de modo que podamos cargarlo con lo que deseemos en cada oportunidad, necesitamos convertirlo en propiedad. Como si fuera un elemento de un SDT, por ejemplo. Eso lo hacemos utilizando esta sintaxis (se conoce como mostacho).

User control

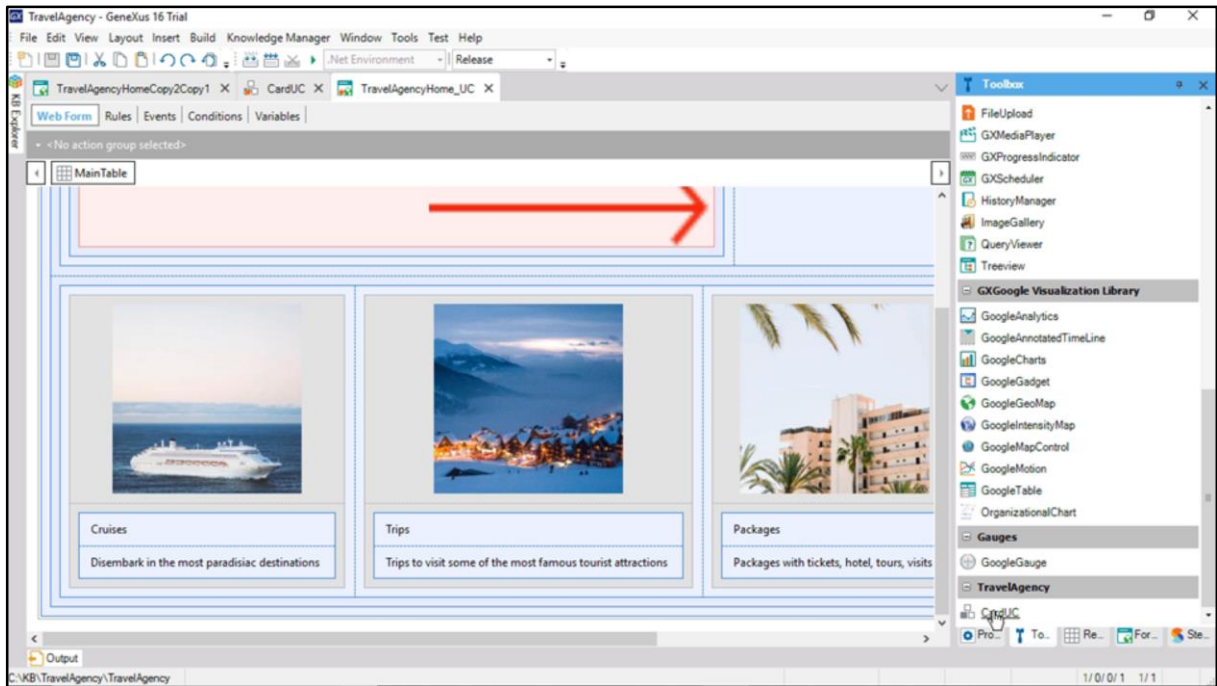


```
1 <div class="ui card">
2   <div class="image">
3     
4   </div>
5   <div class="content">
6     <a class="header">{{Title}}
7     <div class="meta">
8       <span class="date">{{
9       }}
10    </div>
11    <div class="description">
12      {{Description}}
13    </div>
14    <div class="extra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

```
1 <Definition auto="true">
2   <Property Name="ImageUrl" Type="string" Default="" />
3   <Property Name="Title" Type="string" Default="" />
4   <Property Name="MetaInfo" Type="string" Default="" />
5   <Property Name="Description" Type="string" Default="" />
6   <Property Name="ExtraContent" Type="string" Default="" />
7 </Definition>
```

Si ahora vamos a la solapa de propiedades, allí las vemos.

Observemos que este "user icon" responde a una clase del semanti ui para íconos. Vamos a cambiarla por esta otra: "globe icon" que va a responder mejor a nuestra realidad.



[DEMO: <https://youtu.be/rdfnhY377W4>]

A partir de ahora, en la toolbox aparecerá este nuevo control para ser elegido.

Grabemos con otro nombre esta última alternativa al web panel home que creamos con la última versión del stencil.

Y ahora sustituycamos cada uno de estos stencils por un control CardUC...

Arrastramos el control, tal como hacíamos con los predefinidos...

Veamos sus propiedades. Este corresponderá a la tarjeta de los cruceros. Le cambiamos el nombre. Observemos que están apareciendo estas propiedades. Que corresponden exactamente a los nombres que le dimos a las propiedades del User Control. Podemos, por tanto, especificarlas aquí, o por código.

Vayamos al evento Load (este es un web panel sin tabla base y sin grid). Allí habíamos cargado el caption del text block que era muy extenso como para definirlo estáticamente.

Veamos que si escribimos el nombre del user control y punto, nos ofrece las mismas propiedades...

Vamos a cargarlas aquí...

Ya las mostramos cargadas...

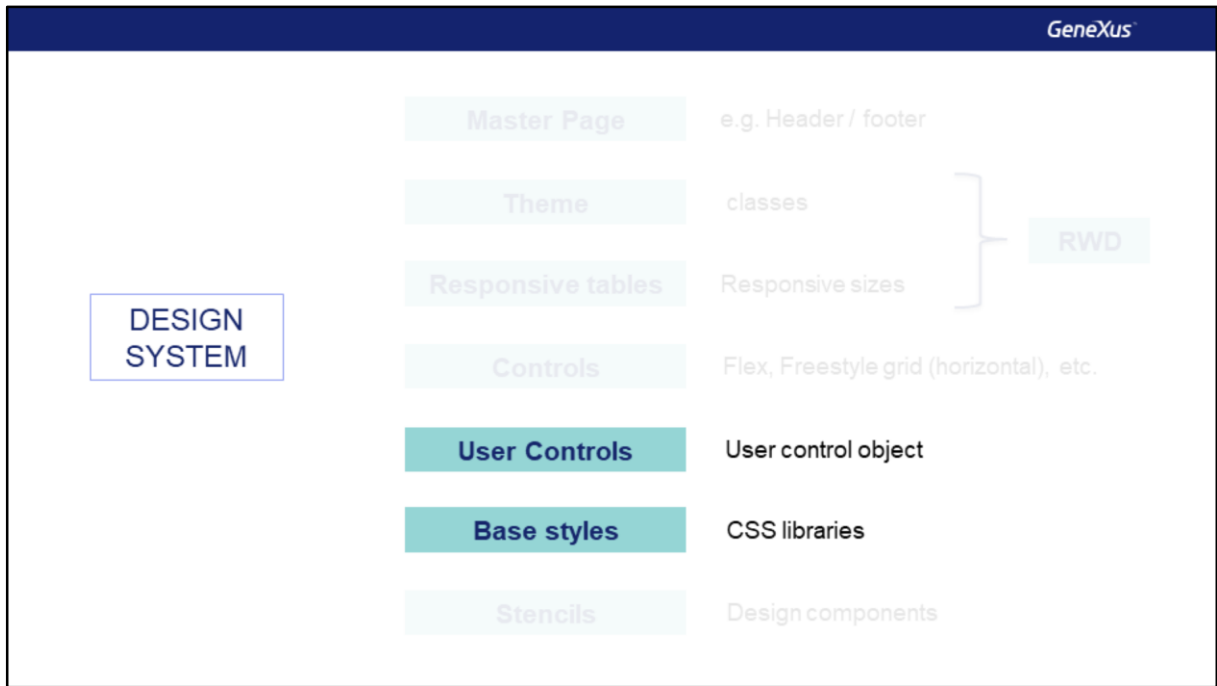
Ahora simplemente eliminemos el stencil que ya no necesitamos más. Y generemos y ejecutemos.

Aquí vemos el user control en acción.

Deberemos hacer lo mismo para las otras tres tarjetas...

Ejecutemos...

Pregunta: ¿por qué, en vez de sustituir las cuatro apariciones del stencil por cuatro instancias del user control, no abrimos el stencil y allí dentro sustituimos su contenido por el user control? Recuerde que el stencil solo permite definir diseño, no comportamiento.



Por supuesto, podemos crear nuestros propios User Controls, sin copiarlos de Frameworks. Deberemos brindar el html con las clases, y el Base Style.

Hay mucho más para decir sobre este tema.

Con la posibilidad de definir controles de usuario extendemos la capacidad de GeneXus para implementar el Design System.

Con esto, terminamos esta aproximación al Design System de nuestra aplicación final, y a los jugadores que participan para su implementación.



Videos

training.genexus.com

Documentation

wiki.genexus.com

Certifications

training.genexus.com/certifications