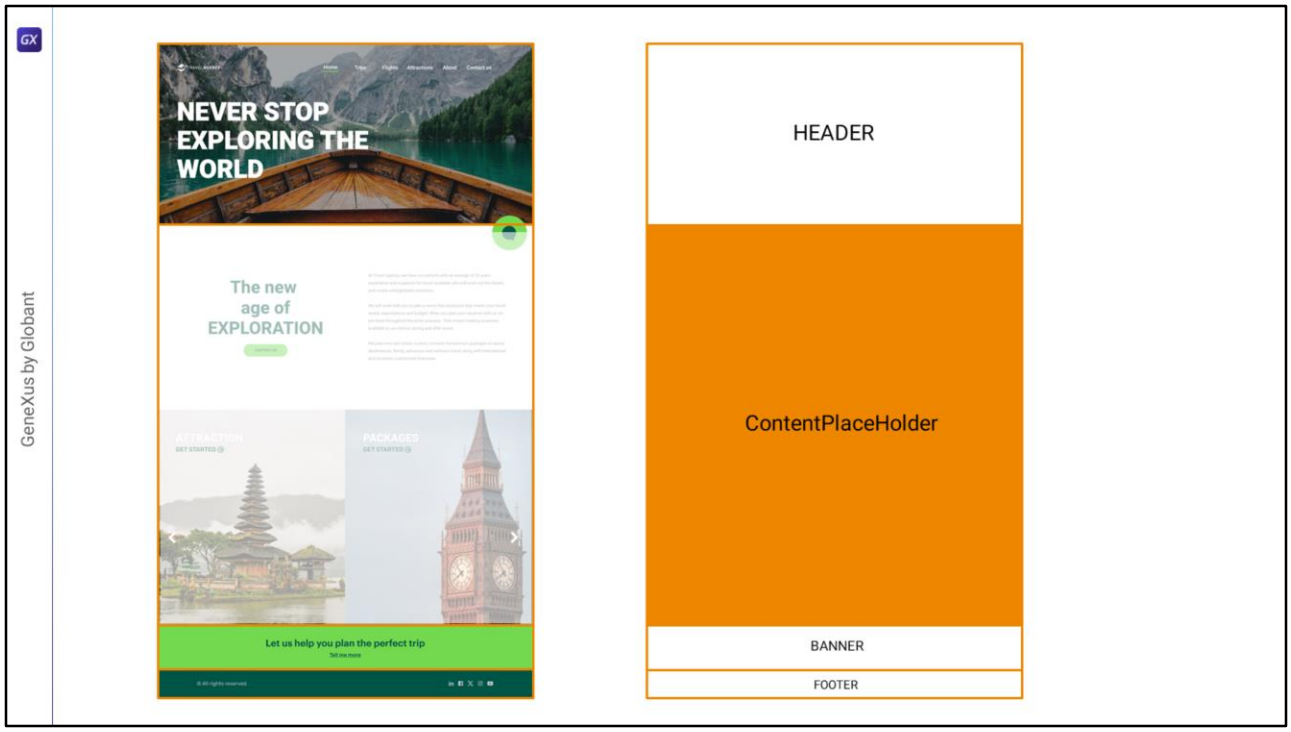
GeneXus by Globant

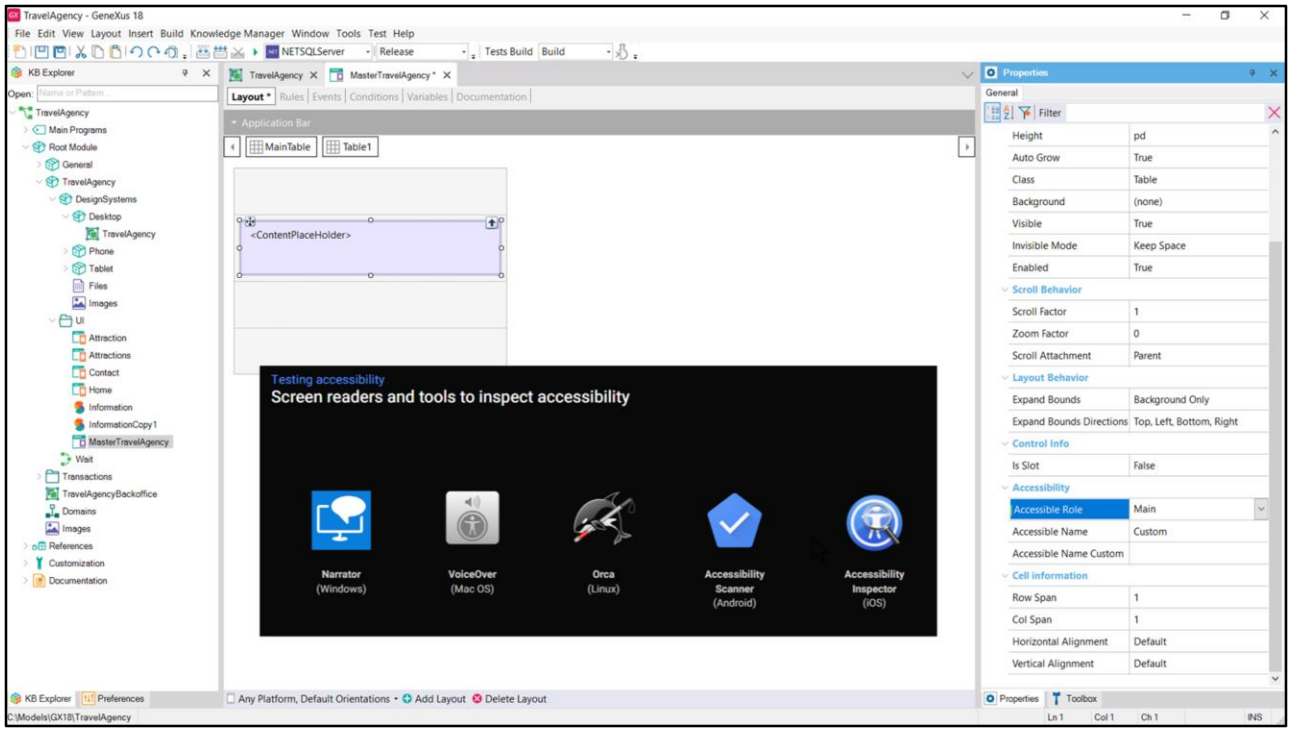
Accessibility and first steps for Master Panel

Cecilia Fernández



Aquí tenemos la primera pantalla de la aplicación, de la que extraeremos lo que corresponde al Master Panel. El área central es la que corresponderá al agujero dentro del cual se cargarán, entonces, los layouts de cada uno de nuestros paneles individuales: el panel Home, el de Attractions, el de Attraction, el de contacto.

Por tanto la estructura del Master Panel nos podría quedar de esta manera: como una tabla con 4 filas; en la primera de las cuales implementaremos el Header; la segunda contendrá el control ContentPlaceHolder, que es justamente el control que hace que las páginas individuales que tengan a éste como su Master Panel se carguen allí dentro; para la tercera fila tendremos la implementación del banner; y en la cuarta implementaremos el footer.

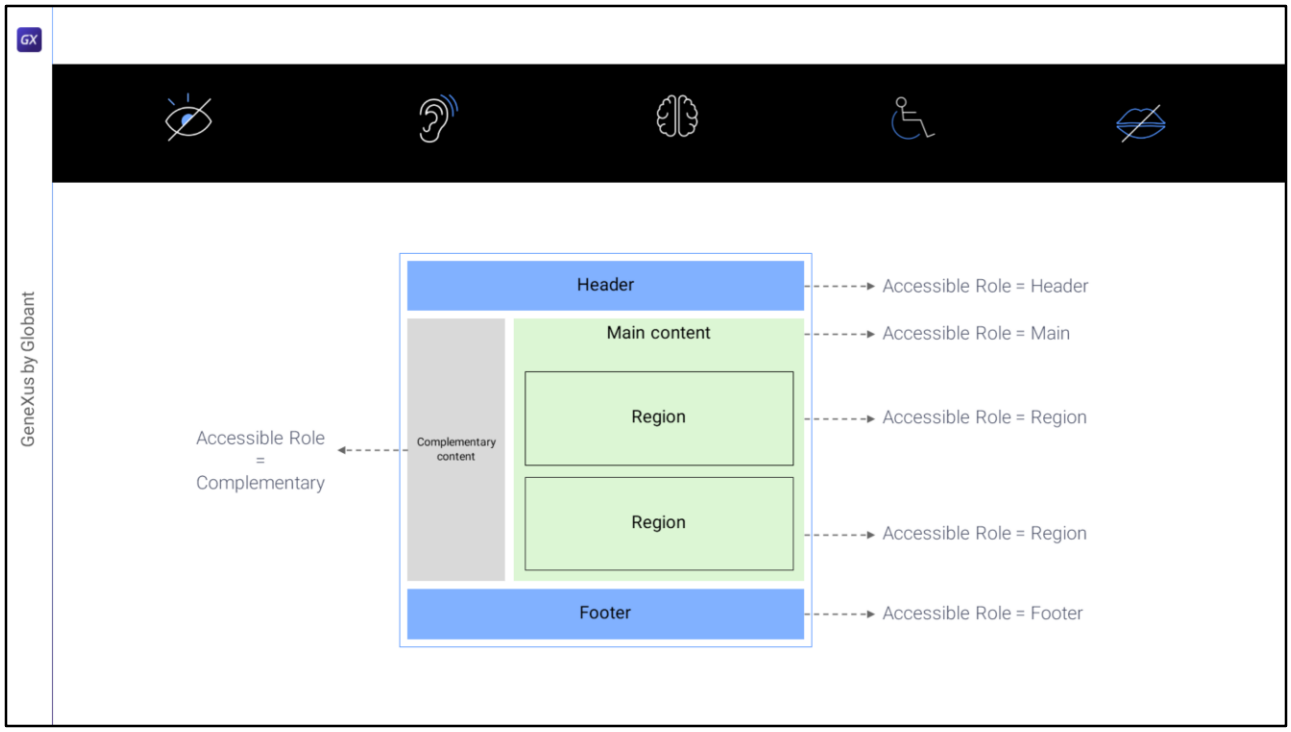


Entonces, comencemos por esta estructura en nuestro objeto MasterPanel.

Teníamos solamente el ContentPlaceHolder, pero ahora lo colocaremos dentro de una tabla, y a ambos dentro de otra tabla que tendrá 4 filas.

¿Por qué colocamos al ContentPlacerHolder dentro de una tabla y no simplemente dentro de la celda de la fila 2? Porque la tabla nos va a permitir semantizar su contenido: podremos decir que aquí estará el contenido principal de la página. ¿Para qué? Para hacer a la aplicación más accesible. Por ejemplo, si un usuario tiene dificultades visuales podrá utilizar uno de los tantos programas lectores de pantalla que le podrá informar vía audio que esta parte corresponde al contenido principal.

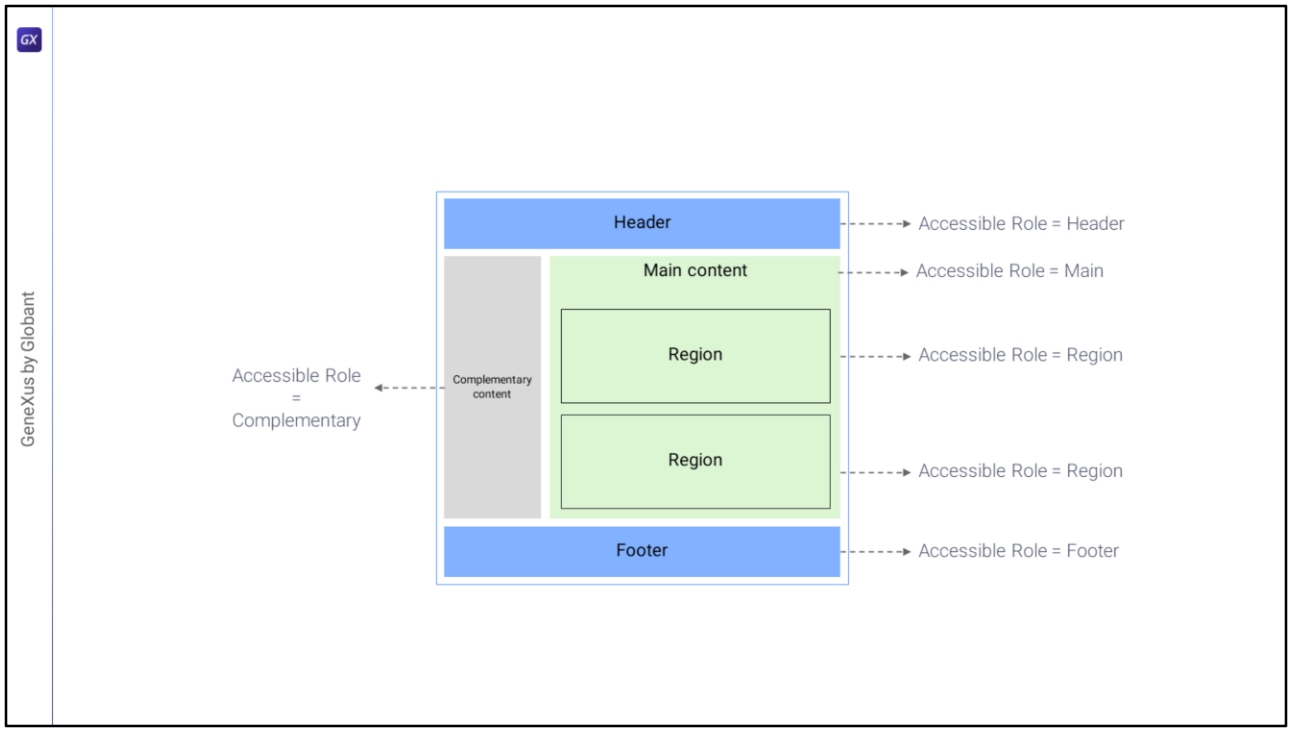
Observemos las opciones de accesibilidad que aparecen a nivel de la tabla. Son estas 3: la 2 y la 3 van juntas.
La primera, la del rol, puede tener alguno de estos valores.



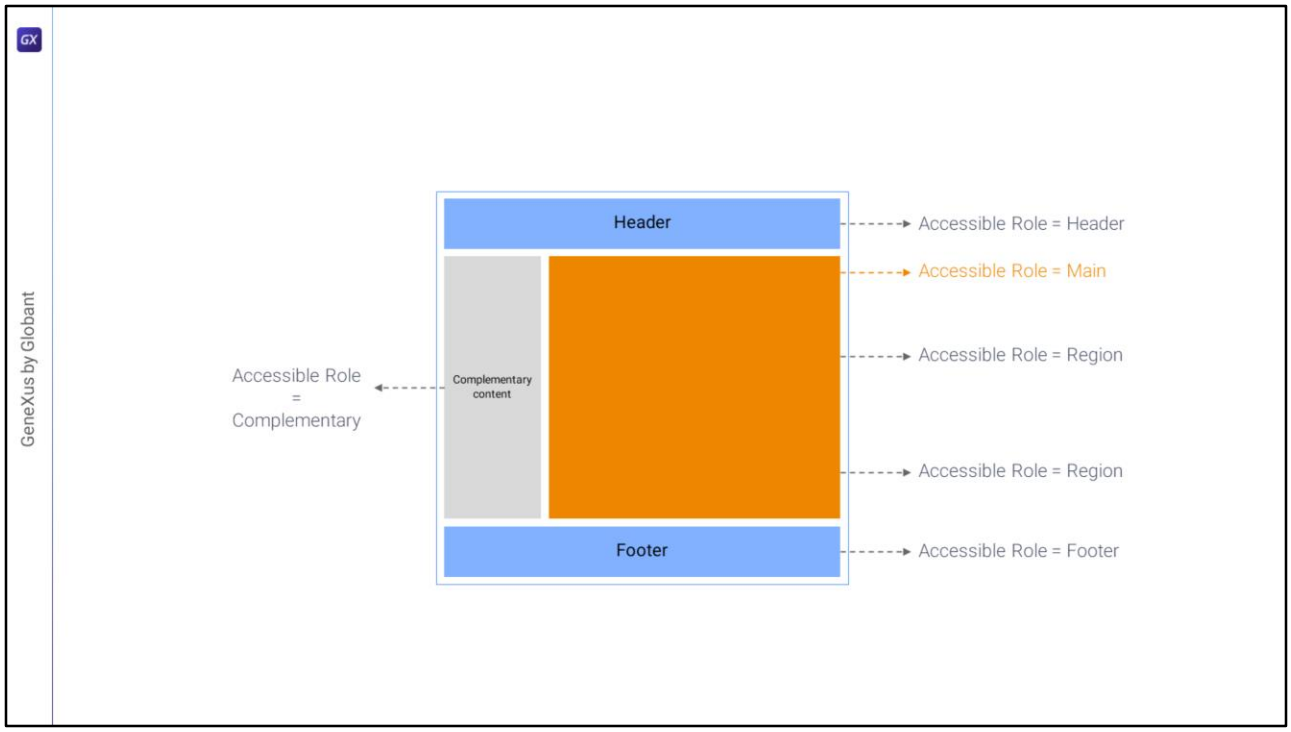
Aquí vemos la semántica de la mayoría de ellos.

La accesibilidad de nuestras aplicaciones es cada vez más importante, incluso es exigida por ley en muchos casos, en muchos países, y no solamente beneficia a personas con limitaciones físicas o cognitivas, sino también cuando hay limitaciones con la velocidad de internet, por ejemplo, porque el usuario podrá saber qué contenido va allí aunque deba esperar para visualizarlo. Además de que mejora la optimización en motores de búsqueda (lo que se conoce como SEO), y entonces la gente encuentra nuestra aplicación web con mayor facilidad.

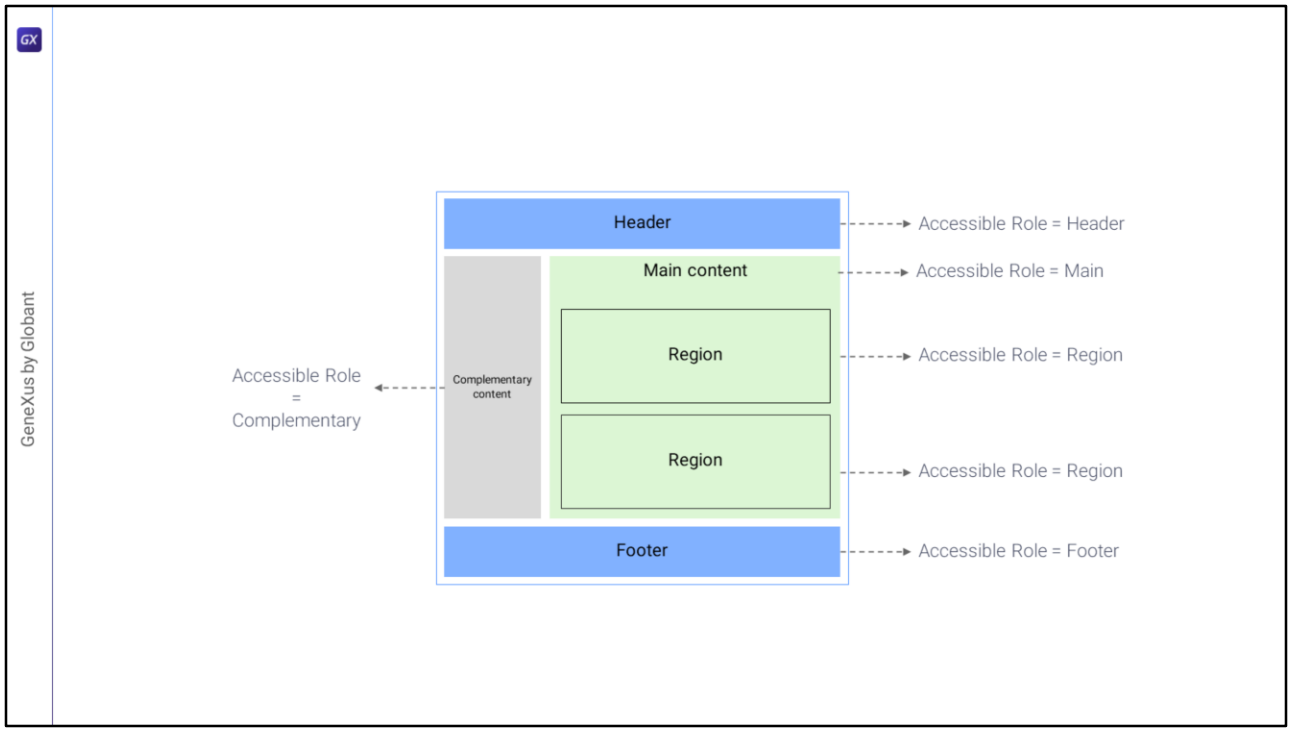
Si desde el principio del desarrollo estamos atentos a hacer la aplicación lo más accesible posible el costo es mínimo y el beneficio es elevado. Tener que rehacer luego el desarrollo para atender a la accesibilidad tendrá un costo muchísimo mayor. ¡Por lo que atendamos esto desde el principio!



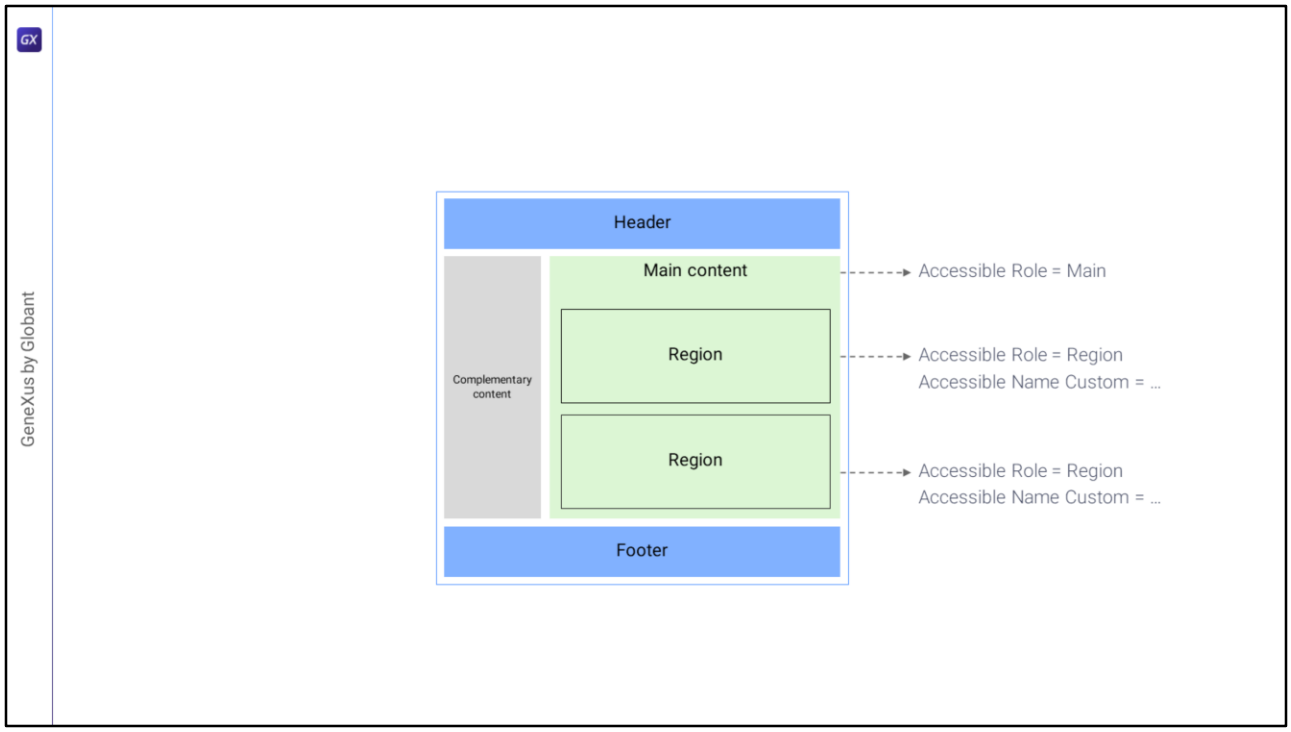
Es evidente, entonces, que tendremos que armar el layout de modo que nos permita indicar los **roles de los contenedores…**



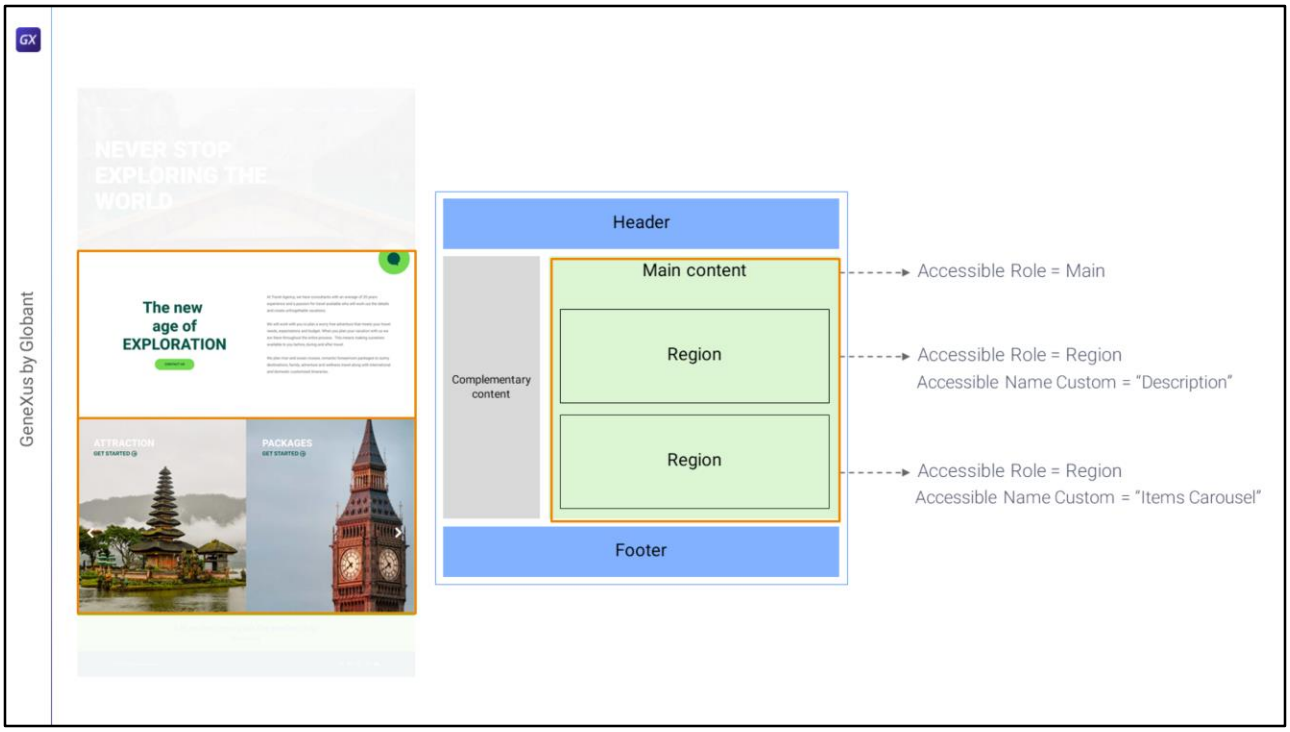
(por eso necesitamos colocar dentro de una tabla al ContentPlaceHolder, porque solo los controles tabla (en todas sus variedades, es decir, también flex y canvas) tienen la propiedad Role).



Y en verdad solo tendrá efecto esta propiedad para Angular. Por el momento la aplicación nativa no hace nada con esta información.

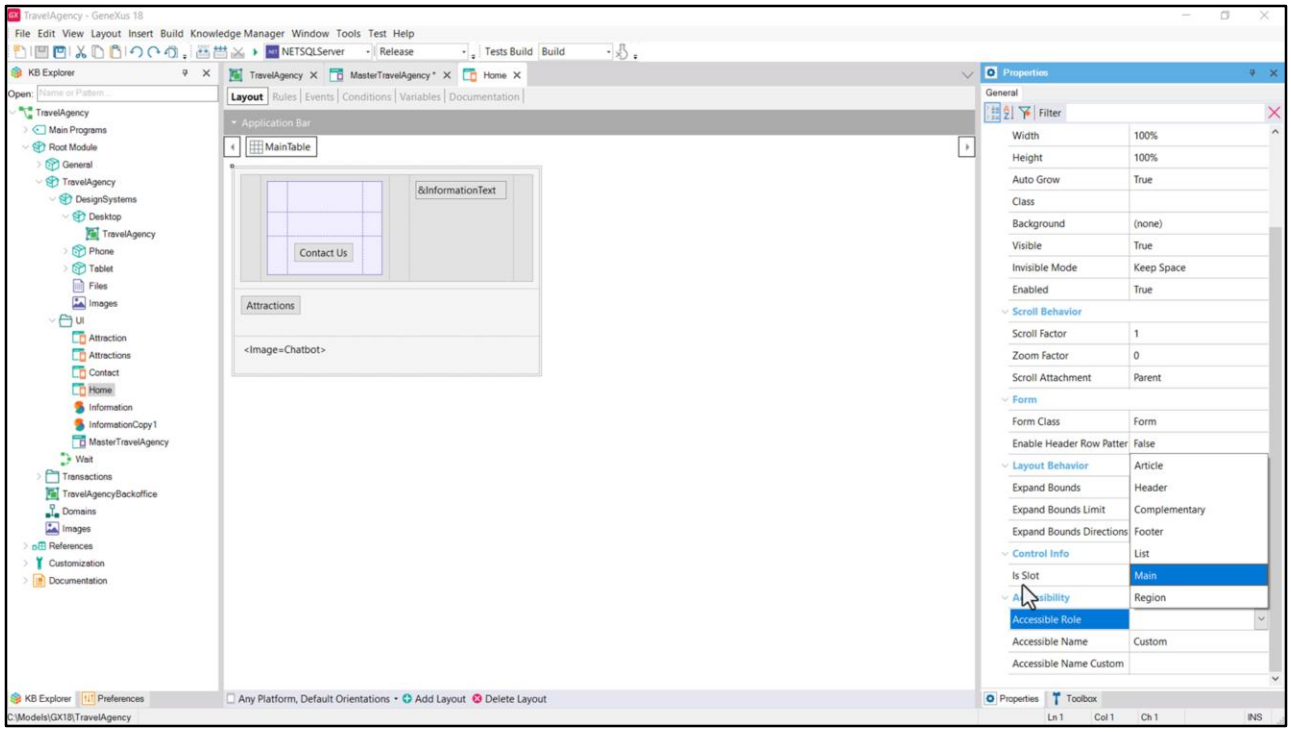


Por otro lado, mientras que por página solamente podrá haber un contenedor Main, regiones podrá haber muchas, por lo que para describirlas semánticamente necesitaremos otra propiedad, que será la Accessible Name Custom.

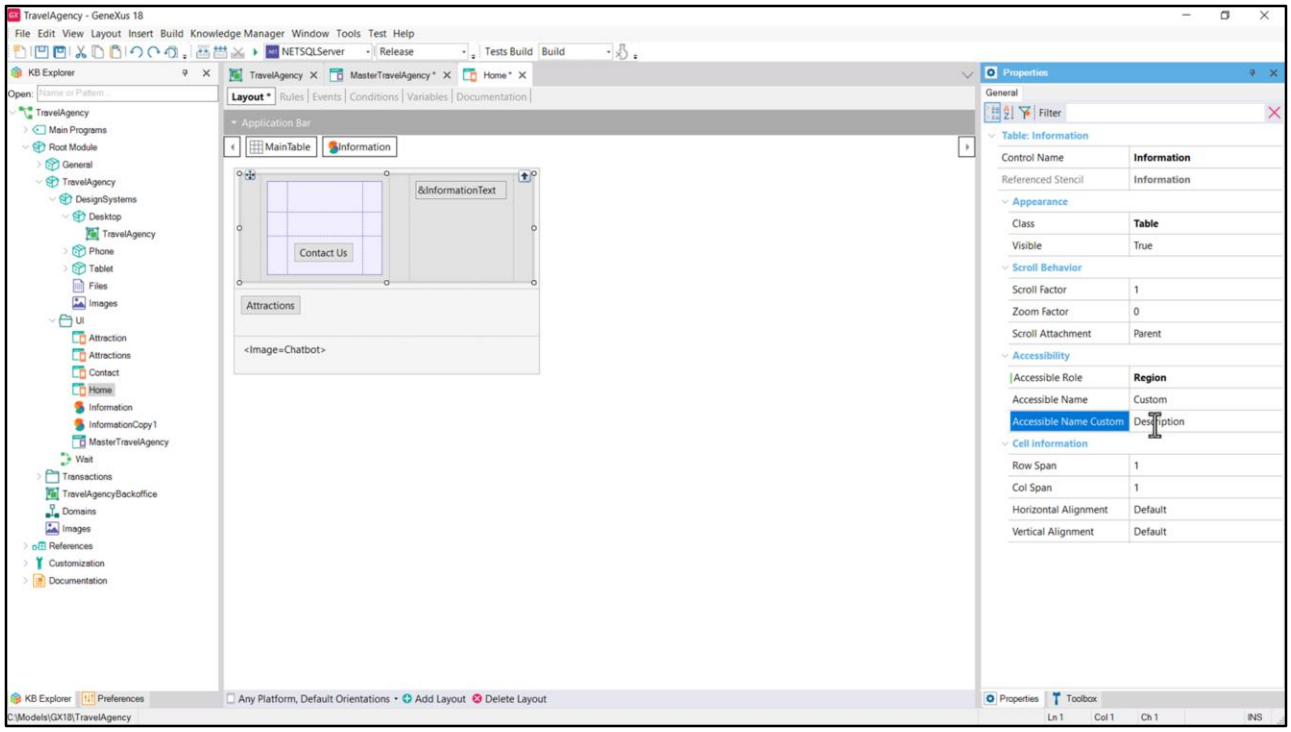


Si pensamos en el panel Home (y no en el Master Panel): tendremos una tabla principal, Main, con dos filas, y dentro de cada una una tabla con Accesible Role Region y Accessible Name Custom. Y el nombre descriptivo del contenido de la primera región podría ser “Description”, y el de la segunda “Items Carousel” para indicar que allí habrá un carrusel de ítems.

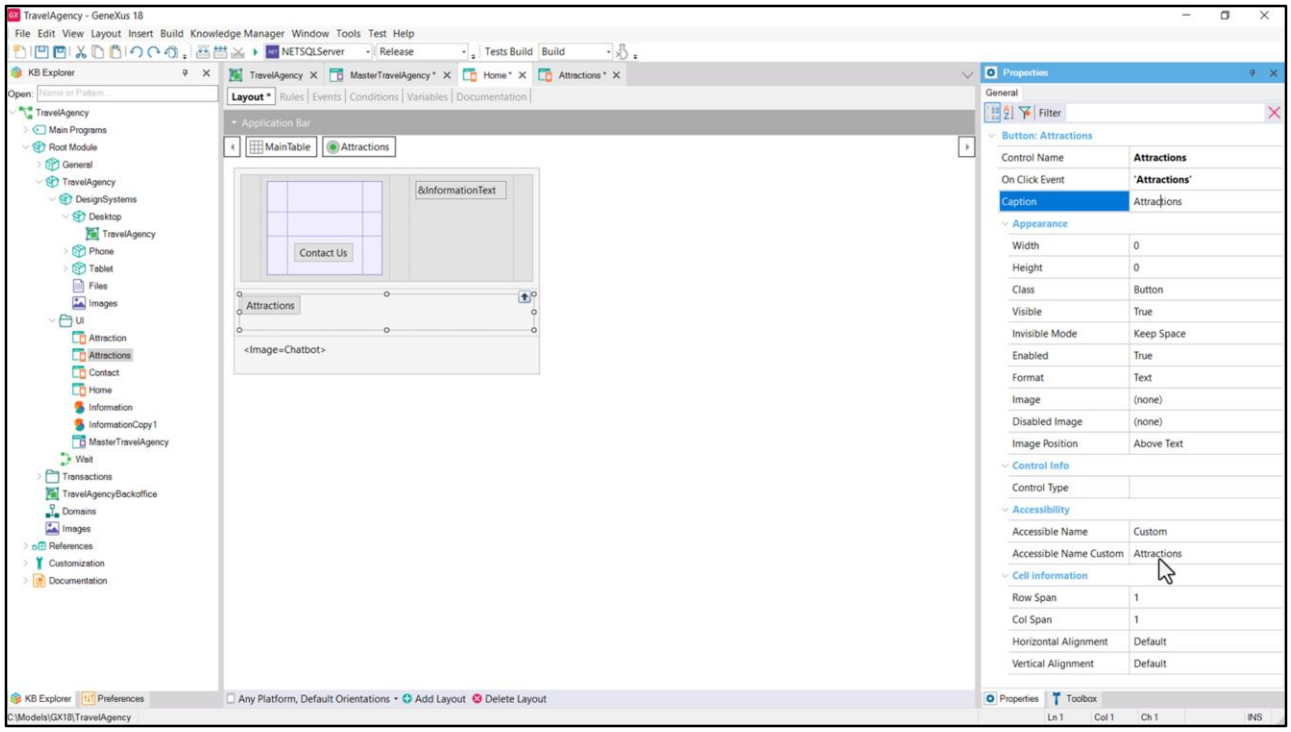
Entonces vayamos a la página Home…



… a la tabla main le colocaríamos como Role el Main…



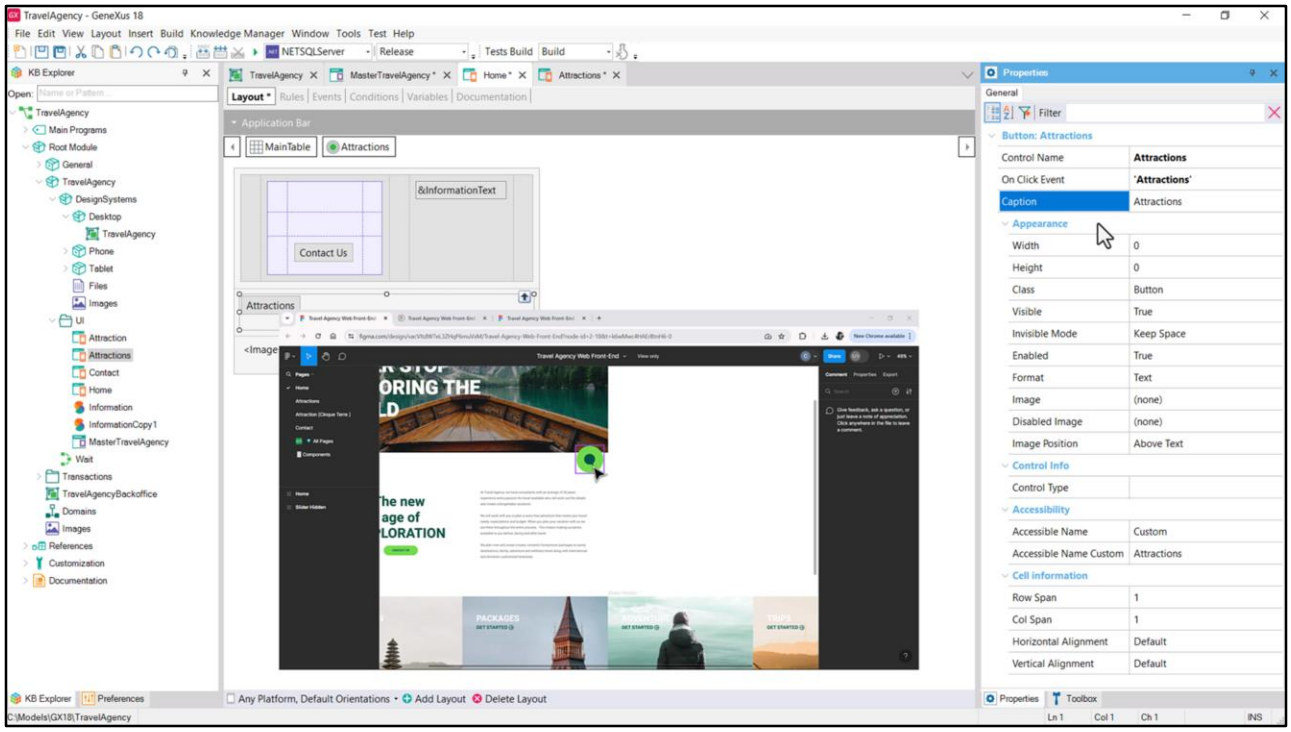
…y a la tabla de la fila 2 le colocaríamos Region, al Accesible Name le dejamos el valor default Custom (porque el otro valor posible es para que tome la descripción a partir del caption de un text block que bien podría ser en este caso pero que veremos luego) y le colocaríamos como descripción entonces, en esta propiedad el valor Description. Lo mismo haremos en el panel Attractions.



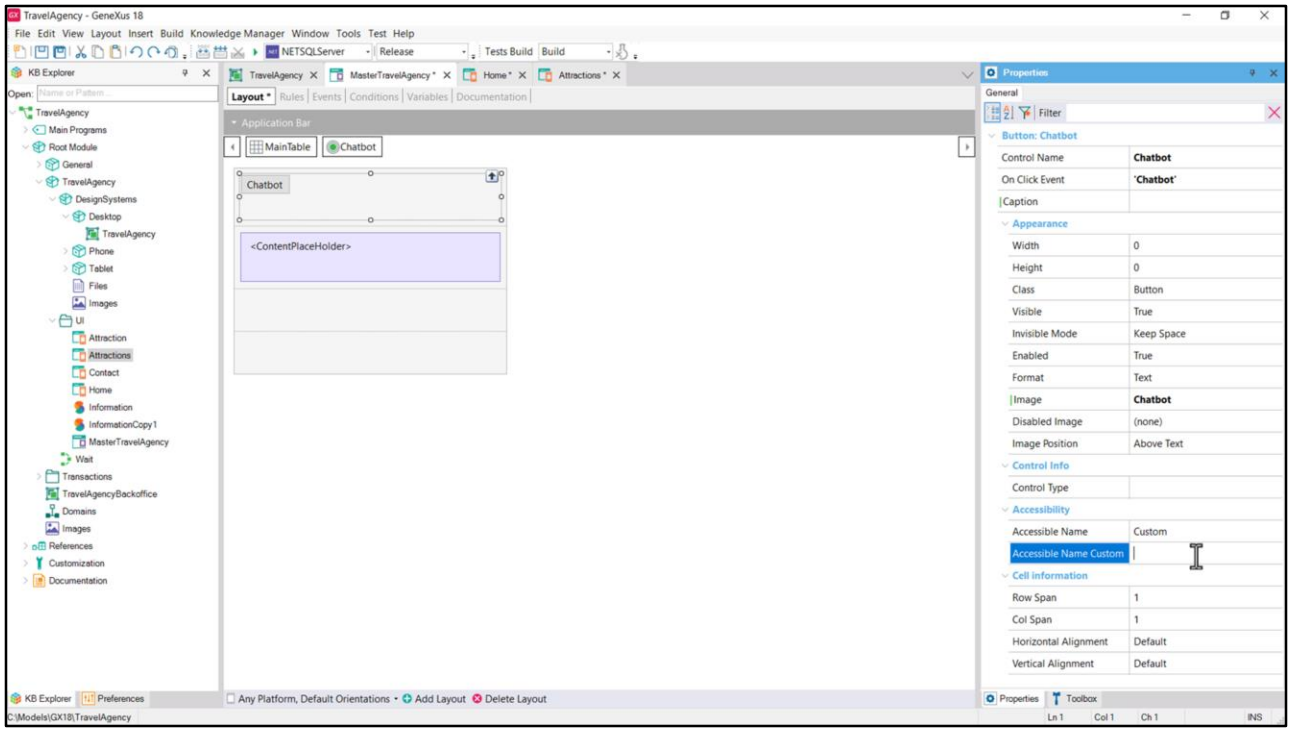
Todavía no implementamos el carrusel para el Home, y ya sabemos que este botón está aquí temporalmente, lo quitaremos cuando implementemos el menú. Y que esta imagen la habíamos dejado solo para mostrar algo sobre las imágenes en un video anterior, tampoco quedará.

Pero aprovechemos a observar que ambos controles tienen propiedades de accesibilidad. Para el botón que tiene Caption Attractions observemos que por defecto GeneXus colocó ese mismo valor para la descripción semántica del botón.

Que sea un **botón** (y no un textblock con evento asociado, por ejemplo) y que su descripción semántica sea Attractions tendrá un par de consecuencias importantes en la aplicación generada para Angular: por un lado, por ser botón ya tiene la semántica universal del botón: entonces podrá manipularse por teclado, enfocándolo con el tabulador y accionando con Enter, por ejemplo. Y por ser botón y tener descripción, un lector de pantalla le informará al usuario que se trata de un botón con descripción Attractions.
Acá no tuvimos que hacer nada porque el botón tomó como Accessible Name Custom el Caption del botón.

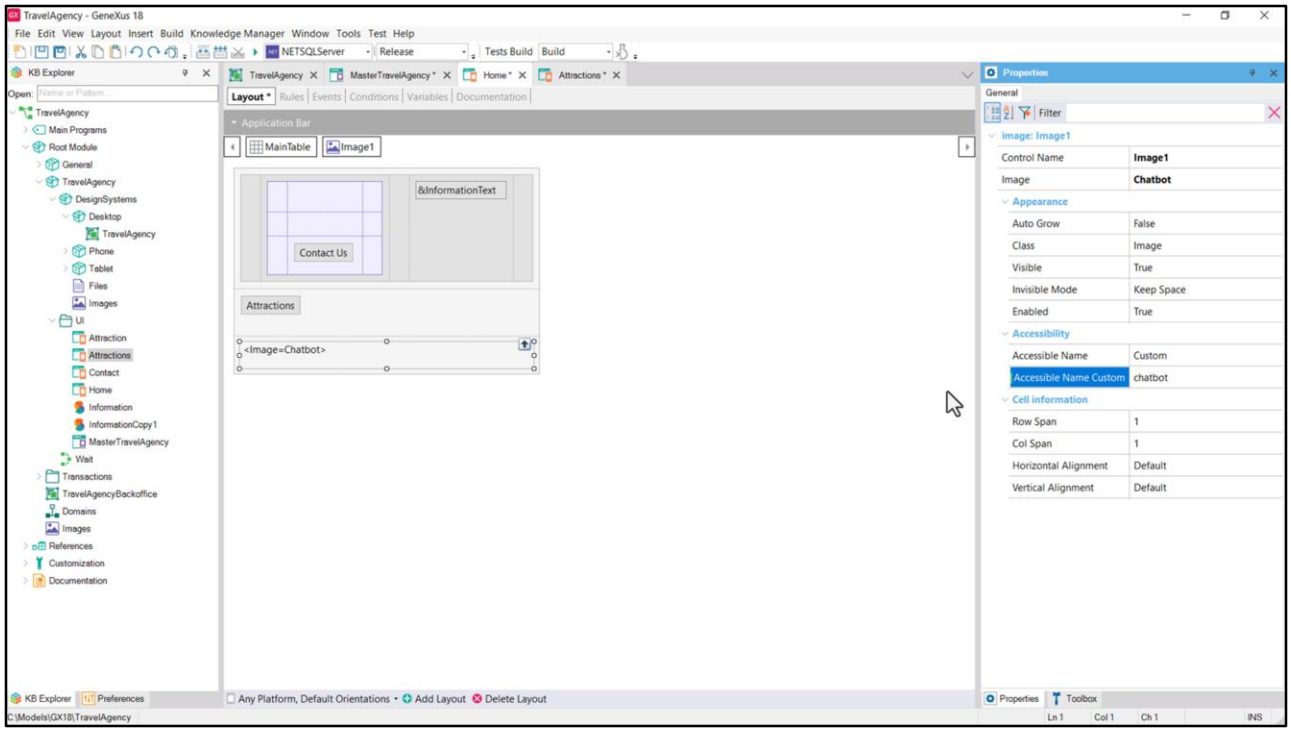


Pero si no tuviera Caption, por ejemplo porque va a ser mostrado con una imagen únicamente, como será el caso de la imagen del chatbot…

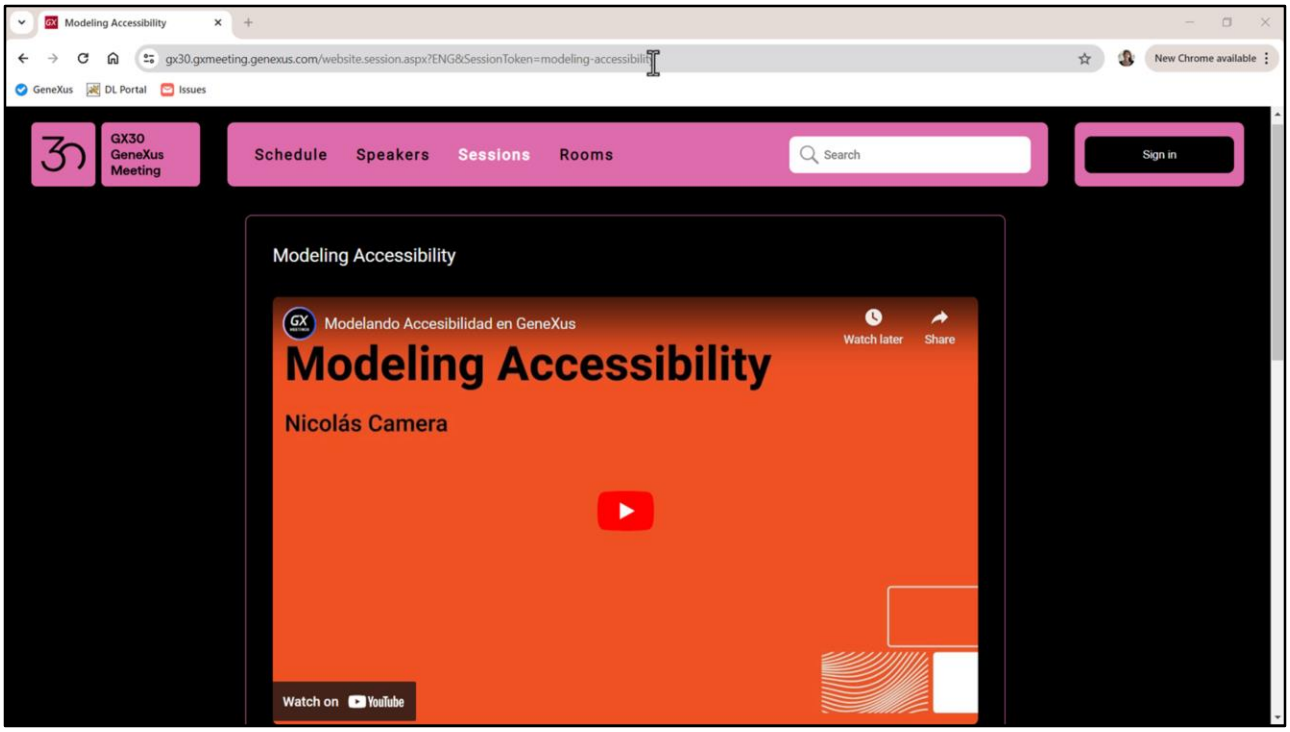


…y ya podemos irlo haciendo en el Master Panel para mostrar...

…arrastro el control botón… le asocio un nombre al evento que se va a disparar cuando se lo presione… ese nombre es el que por default se le asocia al Caption… pero yo no voy a querer que el usuario vea un texto sino lo imagen del chatbot… así que le asocio la imagen que ya tenía cargada en la KB y dejo vacío el Caption…
Pero al hacerlo, la propiedad Accesible Name Custom queda vacía también. Aquí es donde tenemos que intervenir explícitamente. Le colocaré “Chatbot”, como su semántica.

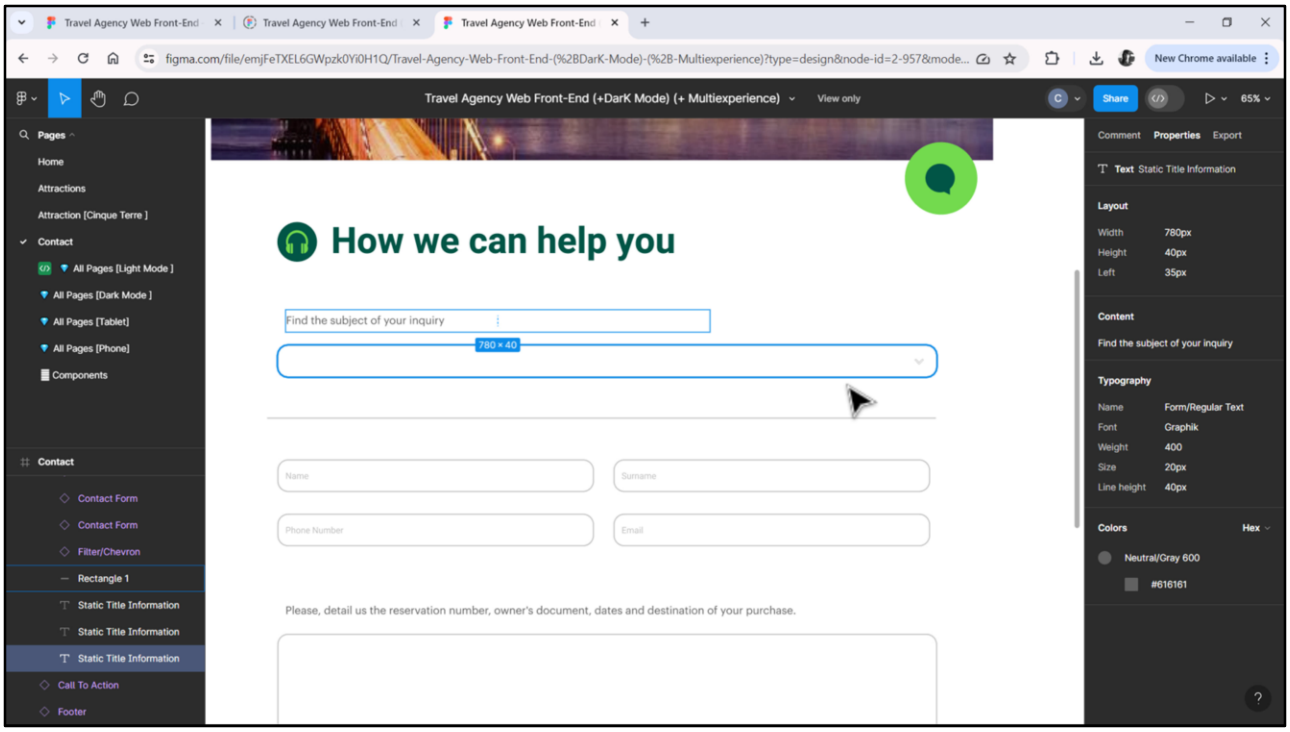


Lo mismo podemos ver con el control imagen insertado en un layout, como imagen y no como botón. Por defecto no tiene ninguna propiedad descriptiva un control de tipo imagen, por lo que no tiene ningún valor default para dar aquí. Tendremos que escribirlo explícitamente.



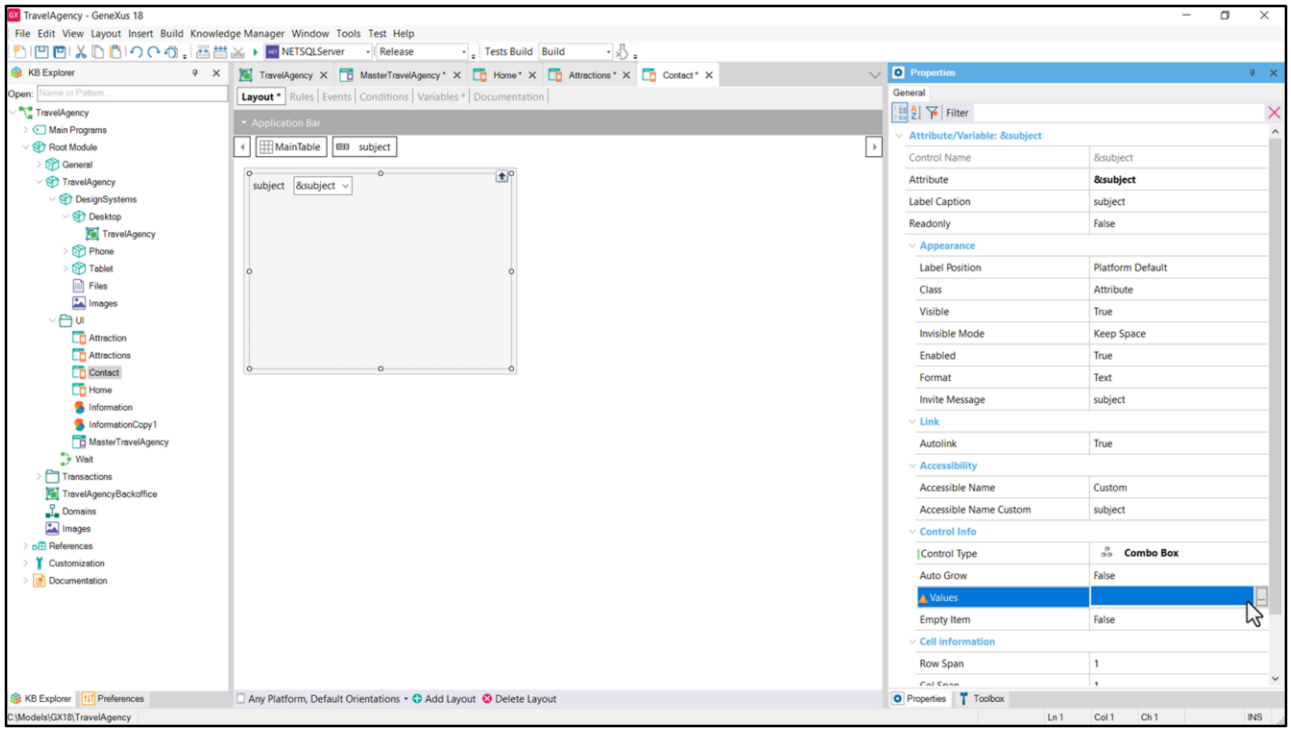
Aquí podrán ver una presentación de fines de 2023 donde Nico Cámera analizó todos los aspectos de accesibilidad a tener en cuenta y proporcionó tips para atenderla desde el principio del desarrollo con GeneXus.

Allí verán que GeneXus ya hace mucho por nosotros sin que debamos preocuparnos más que por elegir los controles apropiados y por proporcionar, vía sus propiedades, valores semánticos descriptivos.

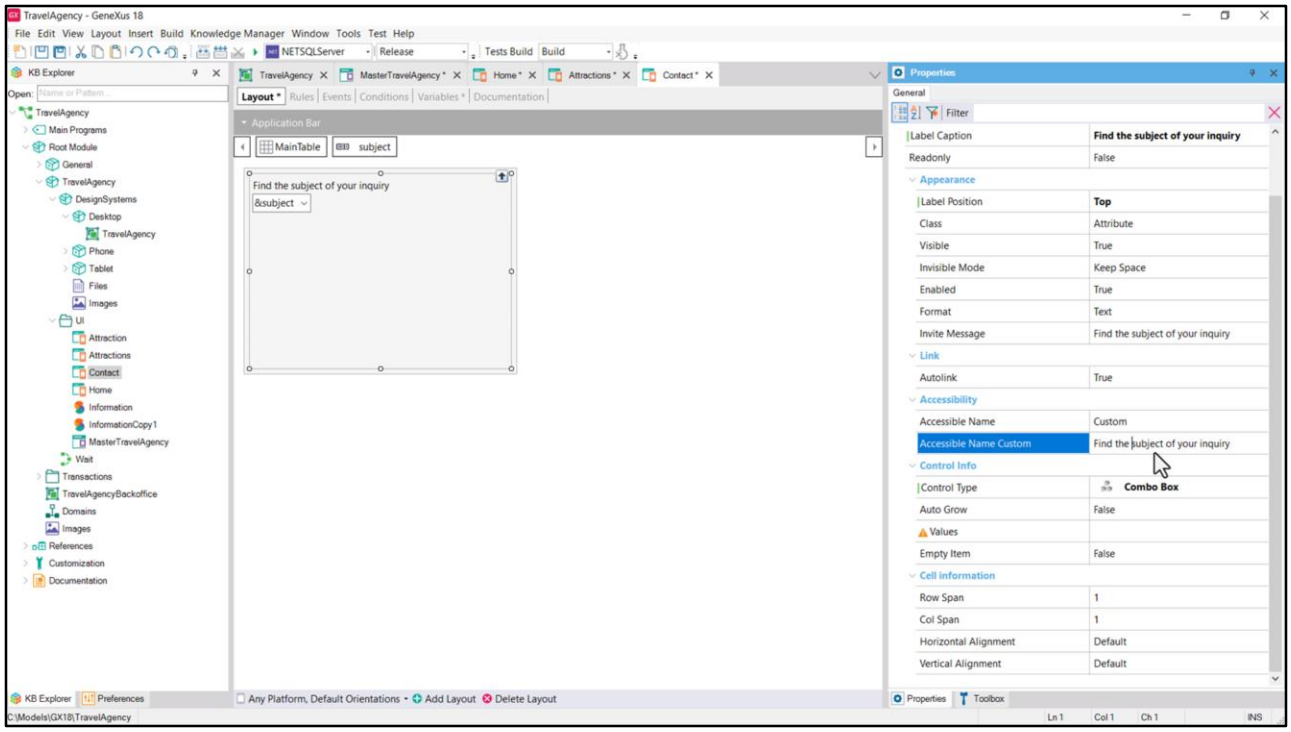


Así, por ejemplo, nuestro formulario de Contacto tendrá campos a ser ingresados por el usuario.

El primero se tratará de un combo-box, y su función semántica se la da este texto… aquí vemos que se trata de un combo.

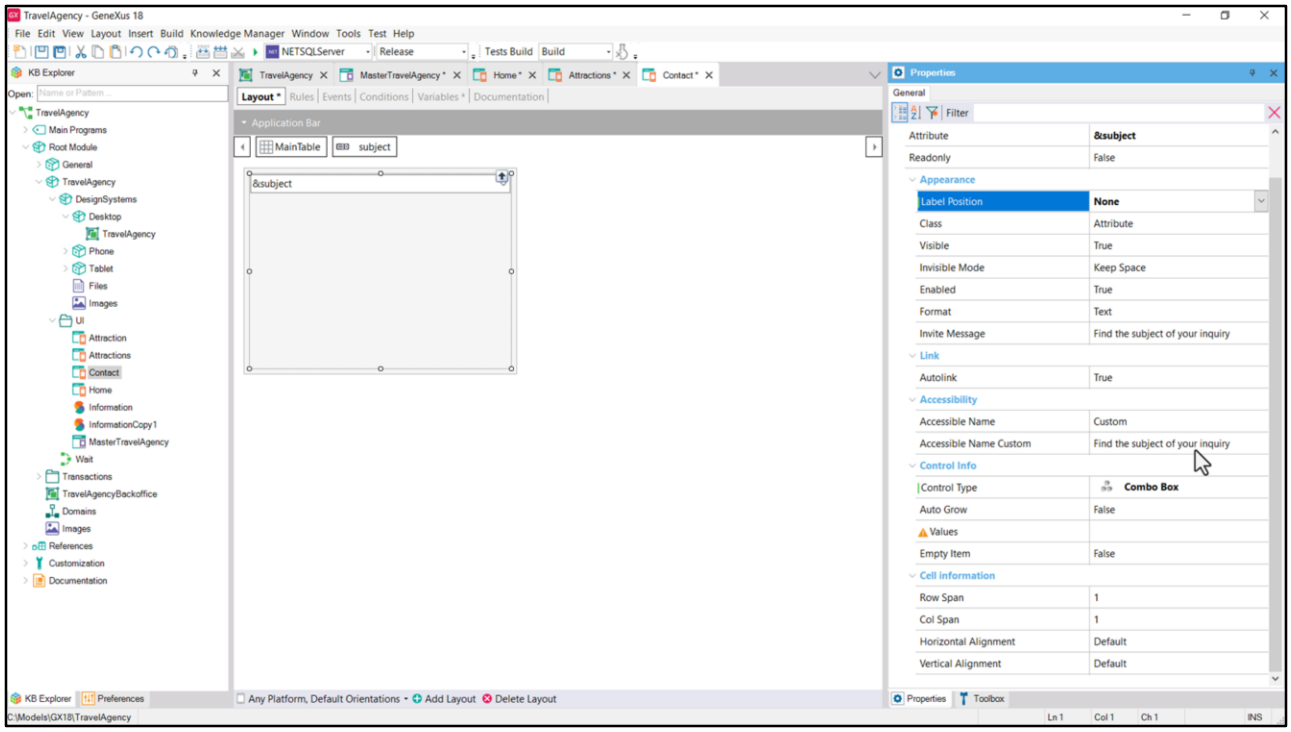


Por lo que lo implementaremos con una variable, a la que le llamé así, que insertaremos en el layout a través de un control attribute/variable al que le especificaremos como tipo de control el combo box. Allí deberemos ingresar sus opciones.

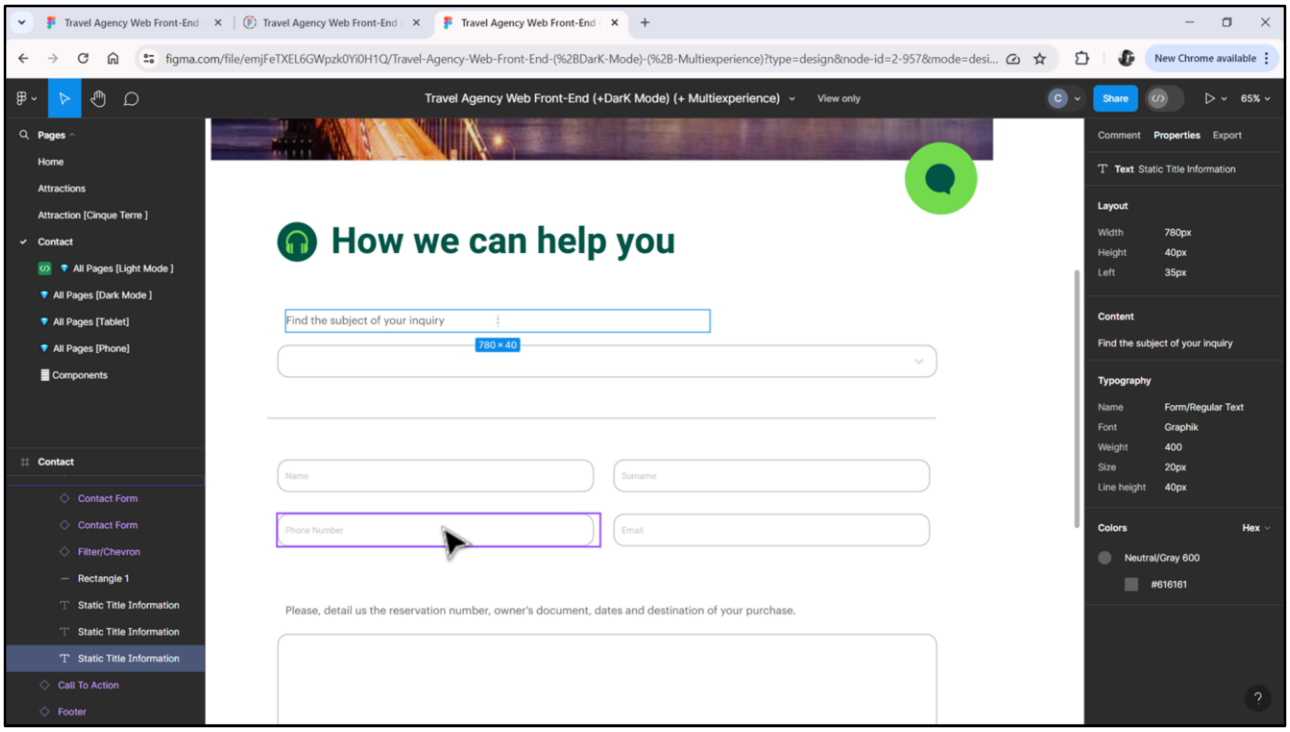


Pero ahora lo que me interesa es ¿cómo modelamos el texto que aparecerá arriba?

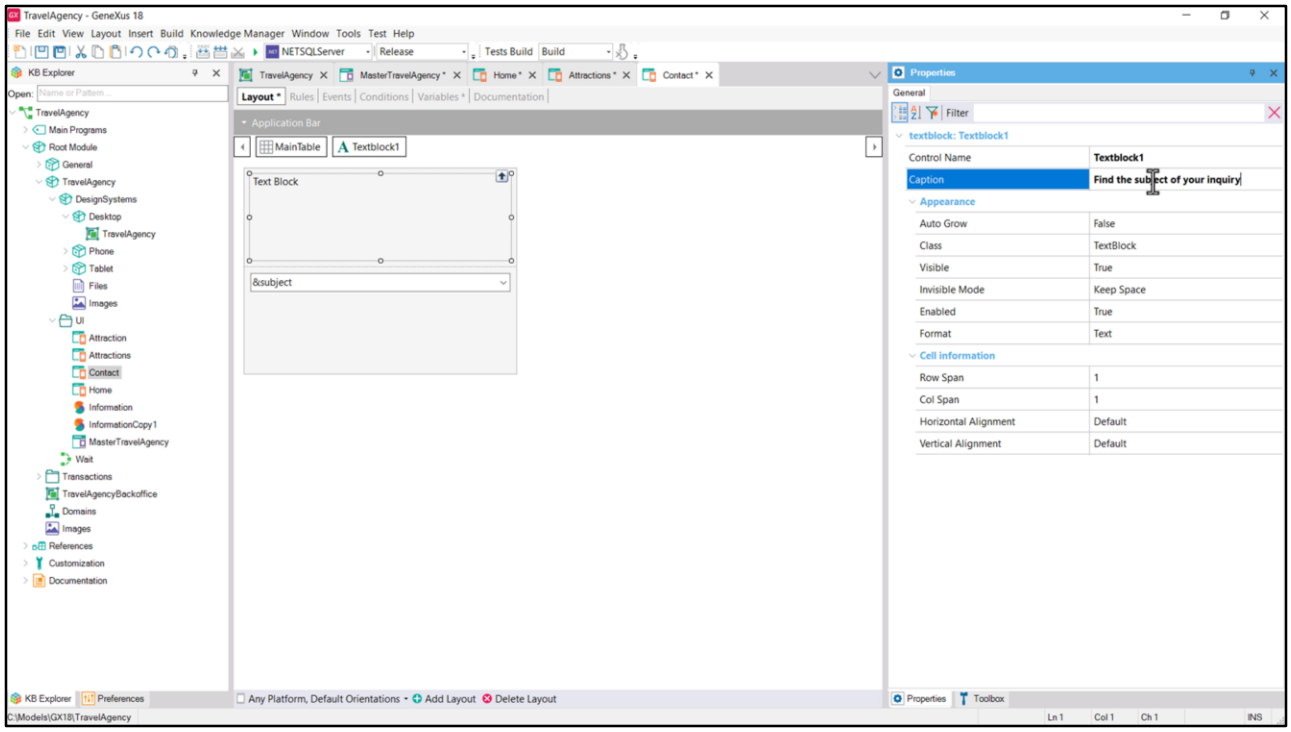
La forma nativa y más evidente es utilizar la propiedad Label Caption de la variable y luego colocar esa etiqueta arriba.
El valor que asumirá por defecto la propiedad Accessible Name Custom será esta misma…



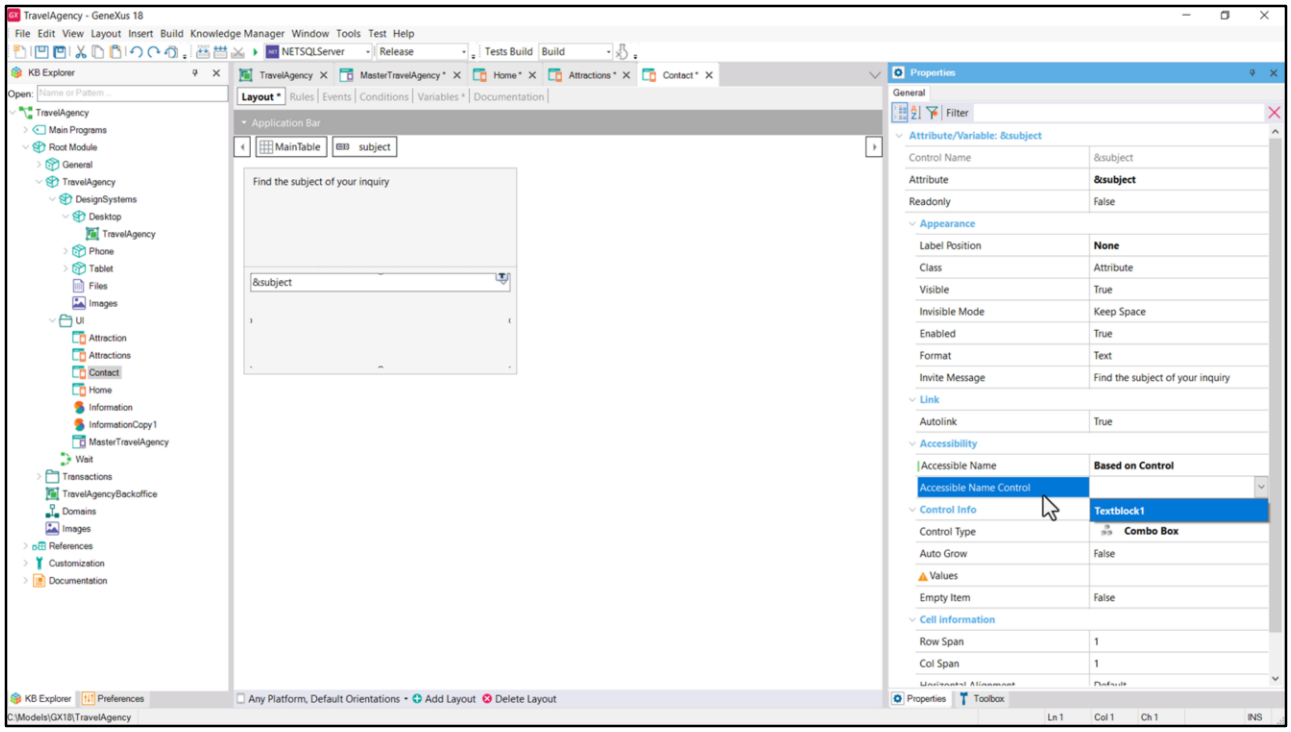
Observemos, por otra parte, que si no queremos tener visible la etiqueta, al haberle ingresado este valor será el que asumirá por defecto la propiedad Accessible Name Custom.



Esto nos será útil para estas otras variables que no tienen texto descriptivo visible.

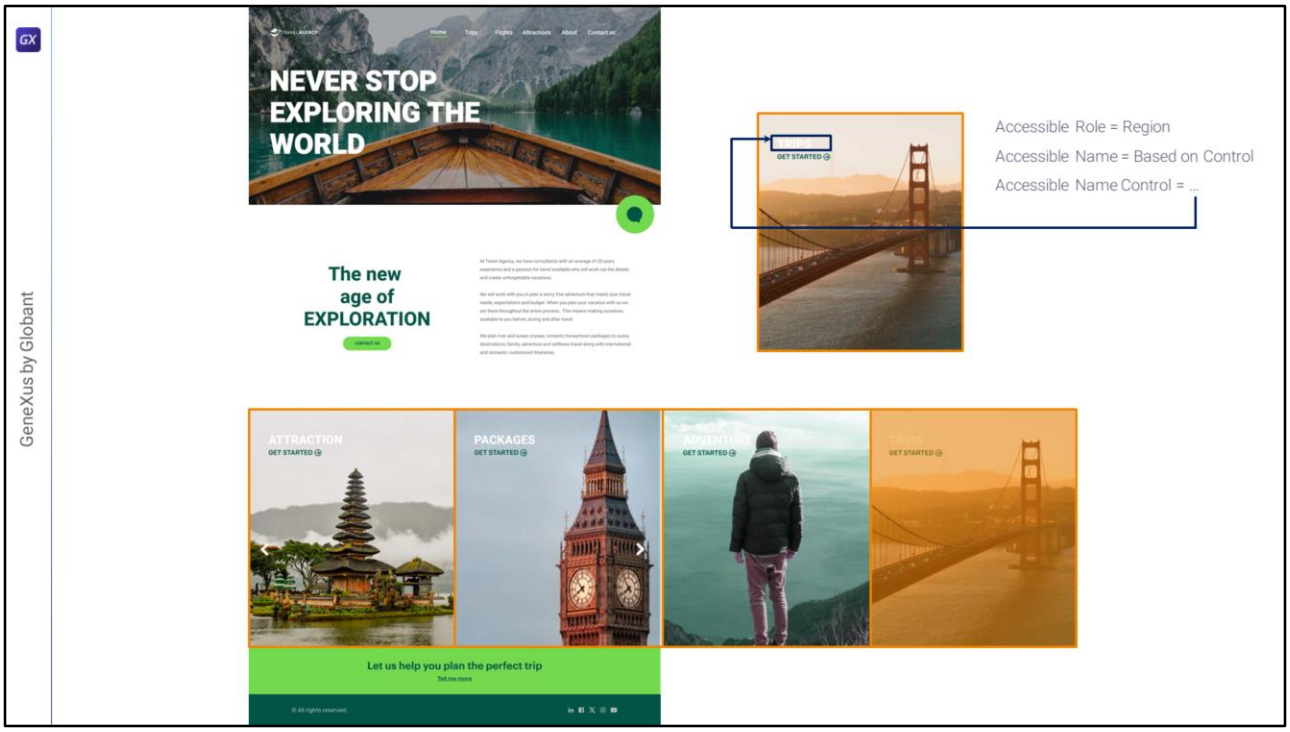


Pero también se nos podría haber ocurrido implementar la label de manera independiente, es decir, a través de un control Text block (opción que desaconsejamos). Entonces insertaríamos el control, de nombre Textblock1, le especificaríamos su caption…



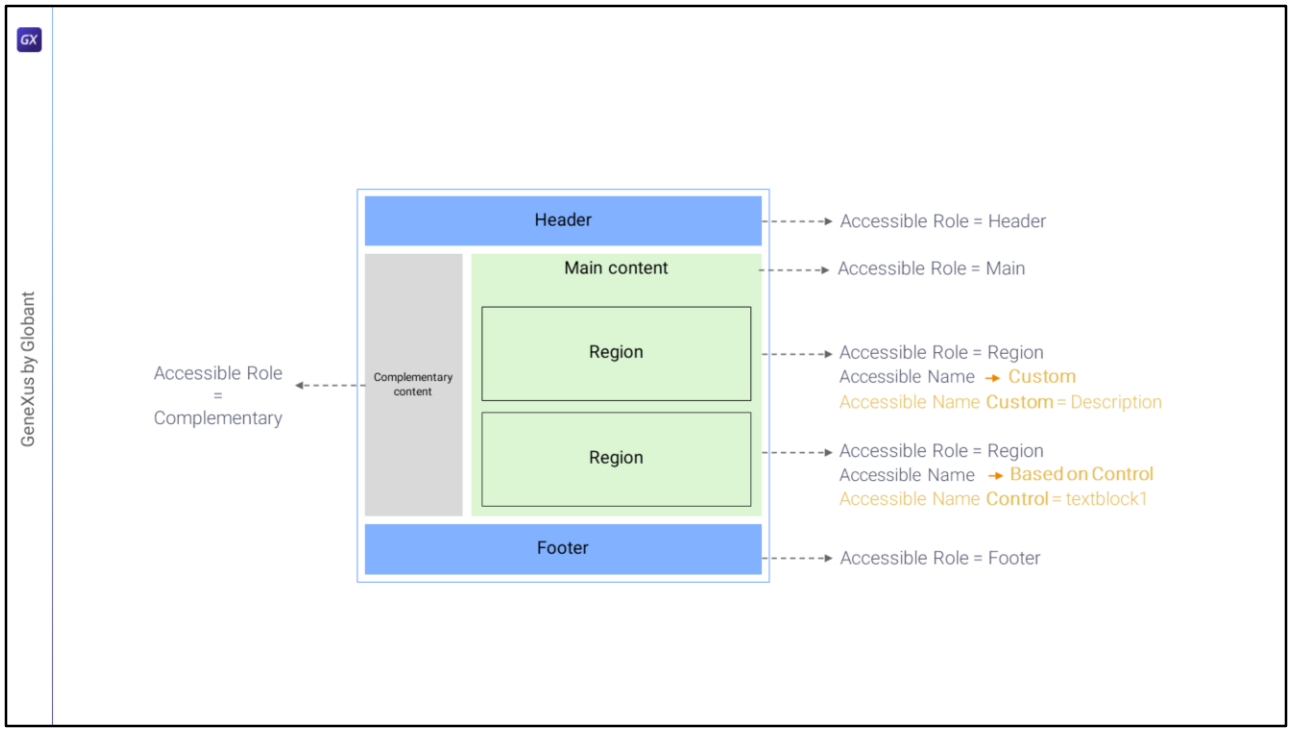
…y lo que tendríamos que hacer es decirle a nuestro combo box que tome su nombre para la accesibilidad de este control.

Pero al hacer esto cuando se haga clic en la etiqueta, la label, no se va a hacer foco en la variable, como sí ocurriría con la primera solución, porque en este caso se trata de dos controles independientes, mientras que en el primer caso la etiqueta era parte del campo.



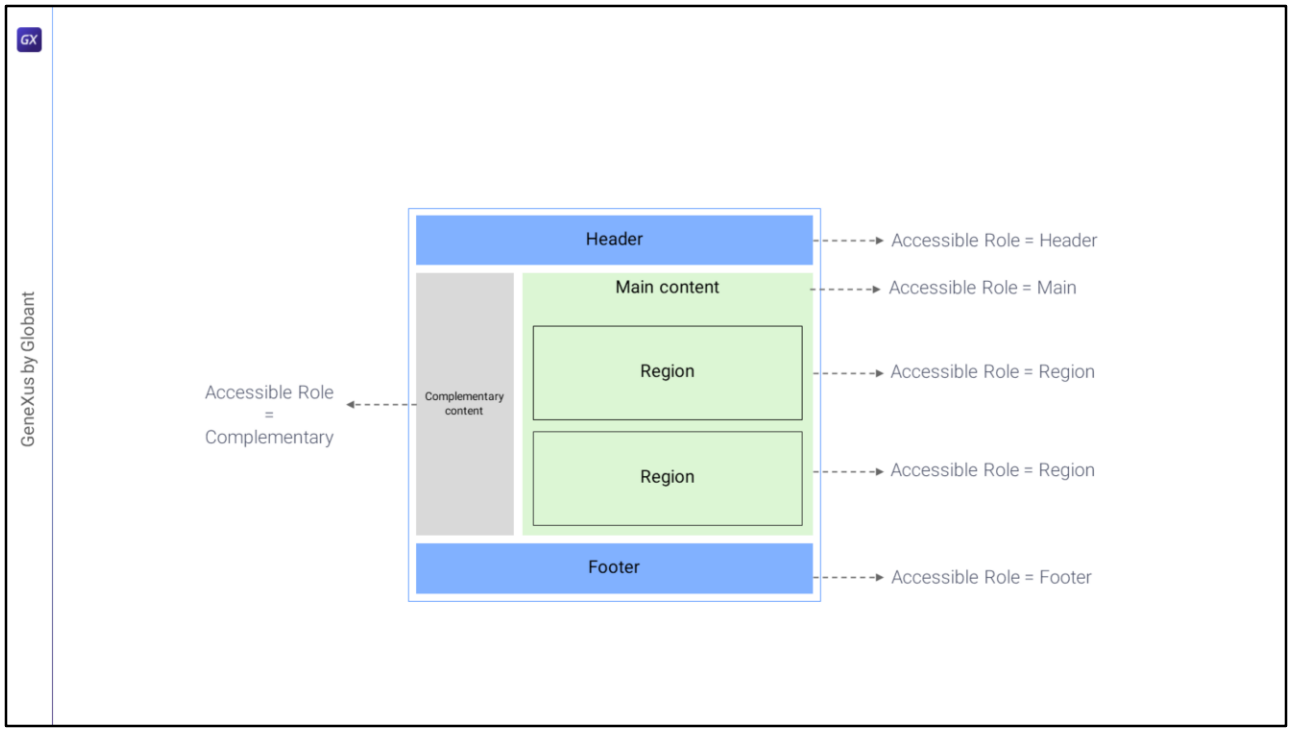
La propiedad Based on control aparece para aquellos casos en los que el control no tiene etiqueta en forma nativa entre sus propiedades y sin embargo existe un text block en el layout que puede describirlo.

Por ejemplo, el carrusel de Home contendrá 4 cards (2 visibles). Lo implementaremos como un grid horizontal donde cada ítem será una card. Y cada una de estas cards se implementará mediante un contenedor canvas (que es como una tabla que permite superponer controles, como ya veremos). Así que a cada uno de esos canvas le colocaremos la propiedad Accessible Role Region, pero el Accessible Name, observemos que ya lo tenemos en el propio layout. Es el que corresponde al título de cada Card, que se implementará como un text block. Por lo que en Accessible Name colocaremos Based on Control y en Accessible Name Control ubicaremos el nombre de este text block.



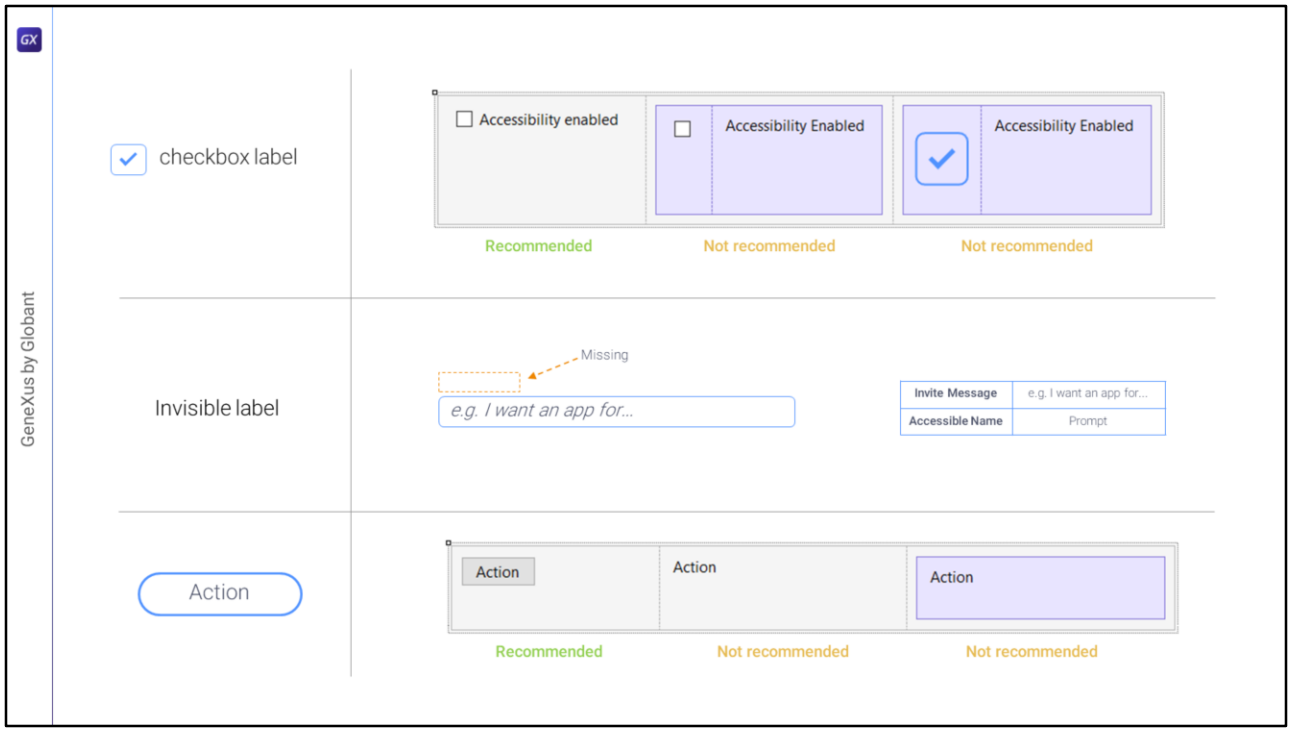
En resumen, para los contenedores (y sólo para ellos) tenemos que indicar el rol que cada uno cumple en la pantalla.

Cuando se trata de un rol que se puede repetir, como el de las regiones, entonces le damos la descripción semántica a la región a través de la propiedad: Accessible Name Custom si vamos a proporcionar directamente el valor allí mismo de modo customizado, escribiéndolo directamente… o a través de la propiedad Accessible Name Control si queremos que tome su valor de un control text block que esté por allí.



La propiedad Role por el momento solamente tomará efecto para la aplicación Angular y no para Android o Apple. Y vale sólo para los contenedores.

Pero las otras dos propiedades valen también para otros controles como los botones, las imágenes, los atributos/variables, como vimos. También para los grids. Estas sí valen también para Android y en este momento se estaban terminando de implementar para Apple (si es que ya no se terminaron).

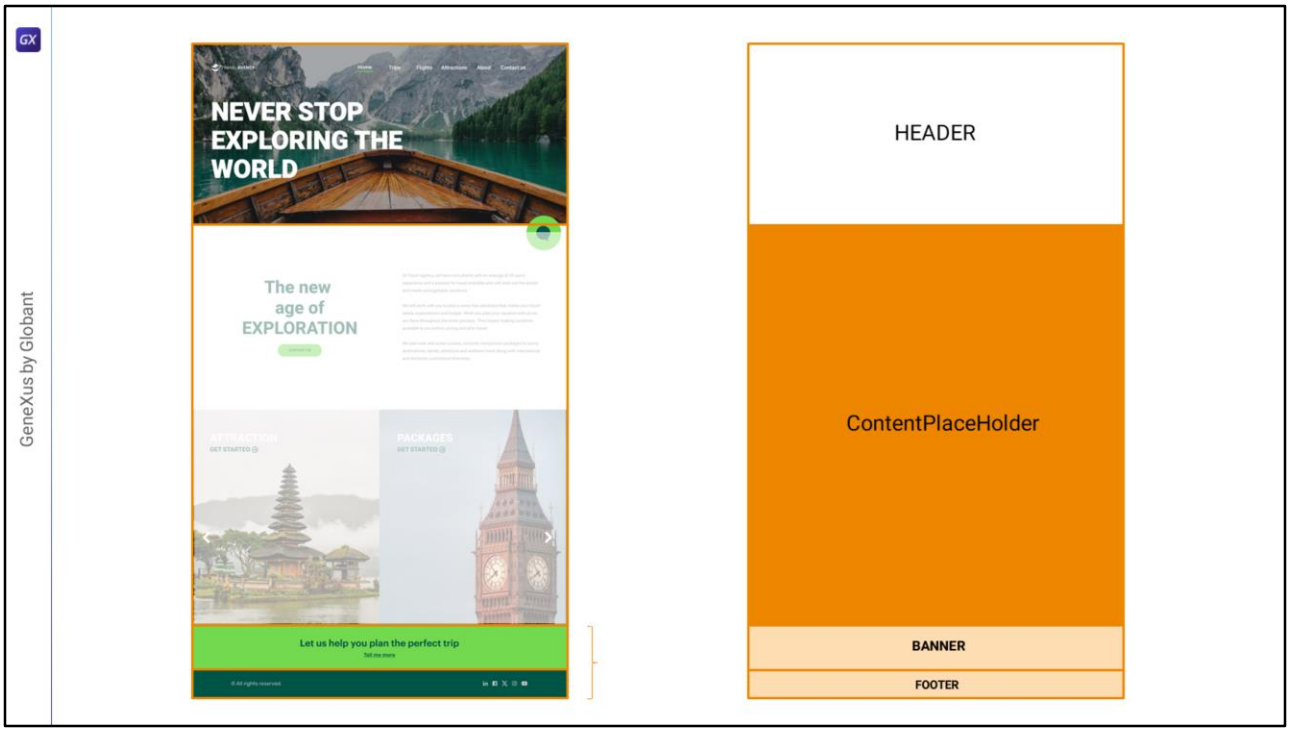


Bueno, entonces vimos como recomendaciones para la accesibilidad: dividir la pantalla en contenedores con roles especificados y luego, para los demás controles:

- Utilizar la etiqueta (label) del propio control y no un control text block aparte. Por ejemplo, la manera recomendada de implementar un checkbox es a través de un control attribute/variable con tipo de control checkbox y con la indicación semántica a través de su propio Label Caption, y no con un text block aparte para eso, o incluso utilizando un control imagen y un text block en lugar del control nativo, checkbox.

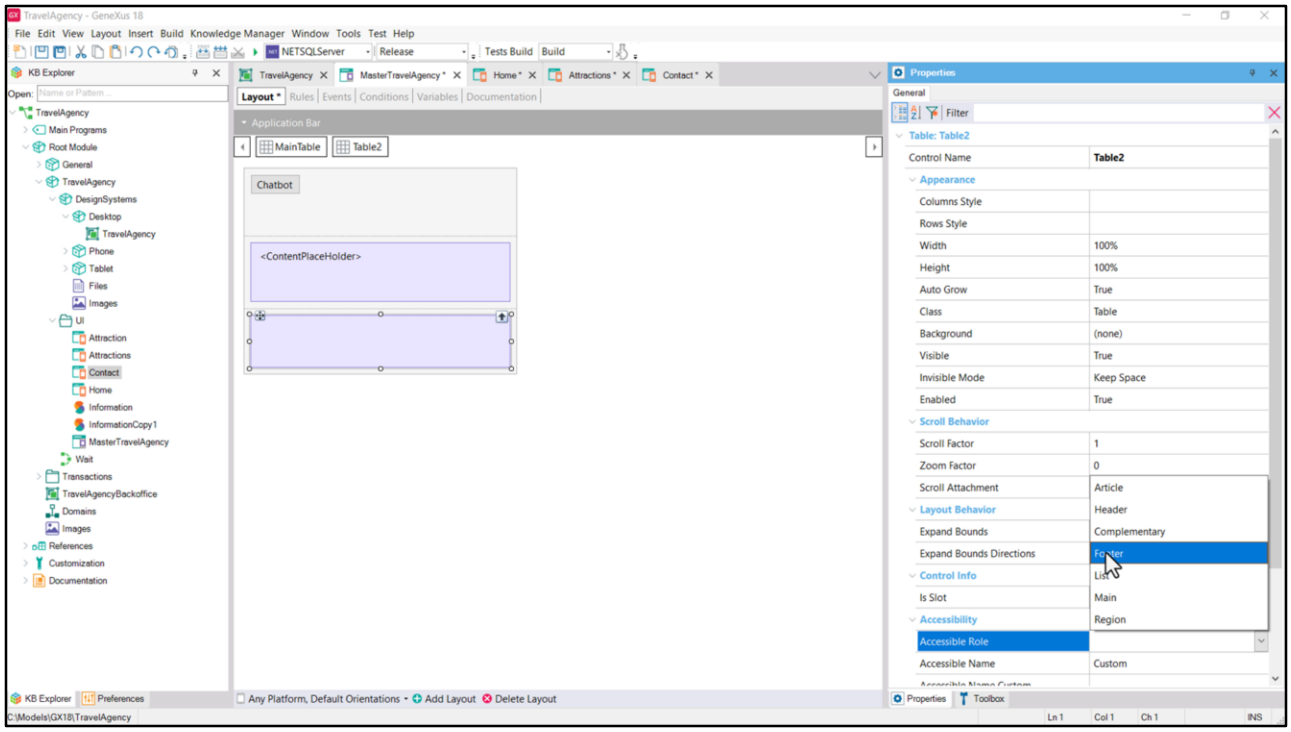
- Si el control no tiene etiqueta, o no será visible, no olvidar dar valor a la propiedad Accessible Name Custom. En este ejemplo tenemos una variable que utiliza un mensaje interno al campo que, cuando el usuario empieza escribir en él, desaparece. No es el texto descriptivo, es un mensaje de invitación a empezar a ingresar valor allí. Ese mensaje se da a través de la propiedad Invite Message del control, pero esta no tiene ninguna repercusión en la accesibilidad. Debemos preocuparnos de que el campo tenga asignado valor semántico en la propiedad Accessible Name.

- Y por último utilizar botones toda vez que se quiera implementar una acción y no otro tipo de control con evento asociado, como puede ser un text block, un text block dentro de una tabla, o incluso (aunque aquí no se muestre) una imagen con evento asociado. En cualquiera de esos casos se pierde la semántica del botón, que es justamente el implementar una acción. Para acciones siempre deberían utilizarse botones, a los que se les da luego la estética deseada, que puede ser sin bordes, sin recuadro, solo texto, o solo imagen, o ambos, pero donde su “ser botón” queda asegurado.



Bueno, todo esto lo iremos poniendo en práctica en los videos que siguen, a medida que vayamos implementando los layouts.

Volviendo entonces a la implementación del Master Panel. En la fila uno tendremos que implementar el Header, y en la 3 y 4 estas otras partes, que Chechu llamó en Figma Banner y Footer. Sin embargo Banner a nivel de la semántica web se entiende como Header, y en realidad estas dos secciones funcionan más bien como un Footer…



Así que dejaremos una sola fila en la que colocaremos una tabla con rol Footer.

La implementación de esta parte es más sencilla, así que vamos a meternos de lleno en la más compleja, que es la del Header. A ella nos vamos a dedicar en lo que sigue.

Continuamos en el próximo video.


GeneXus by Globant
GeneXus™
by Globant
training.genexus.com