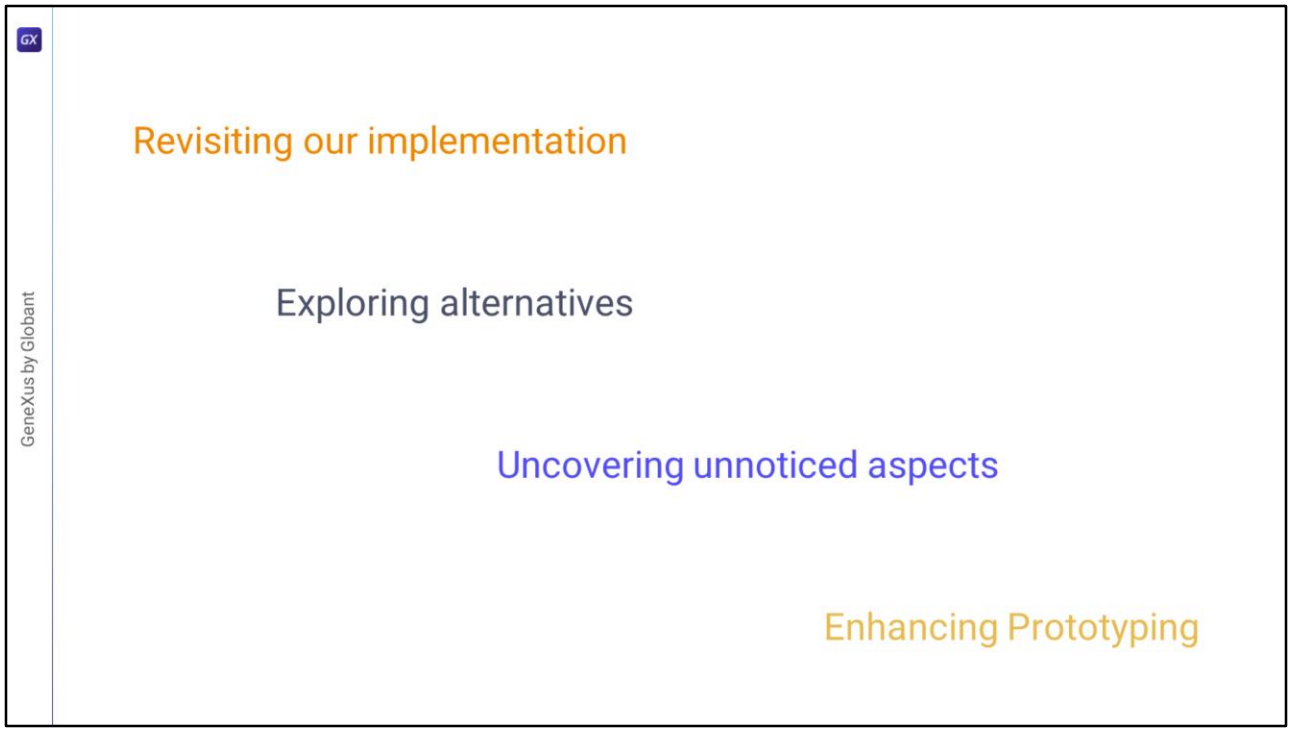
GeneXus by Globant

# First Layout in GeneXus

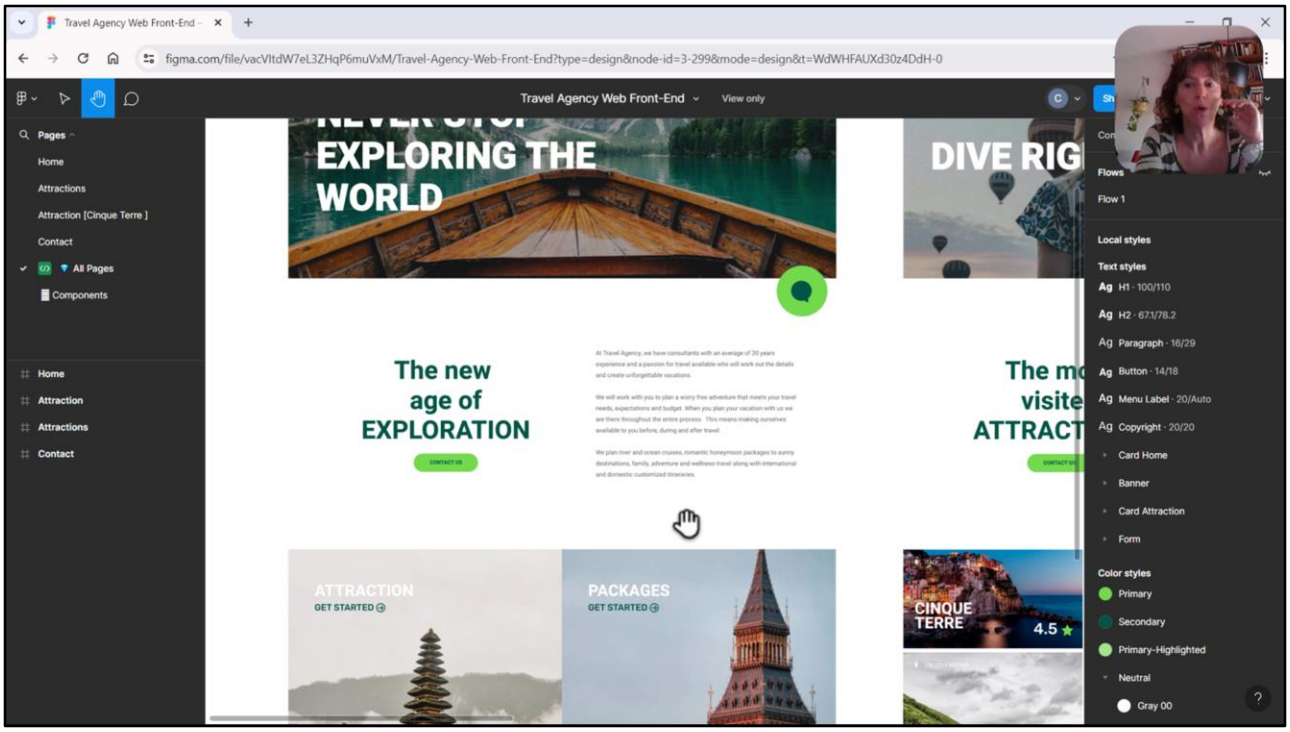
## Some additional considerations

Cecilia Fernández

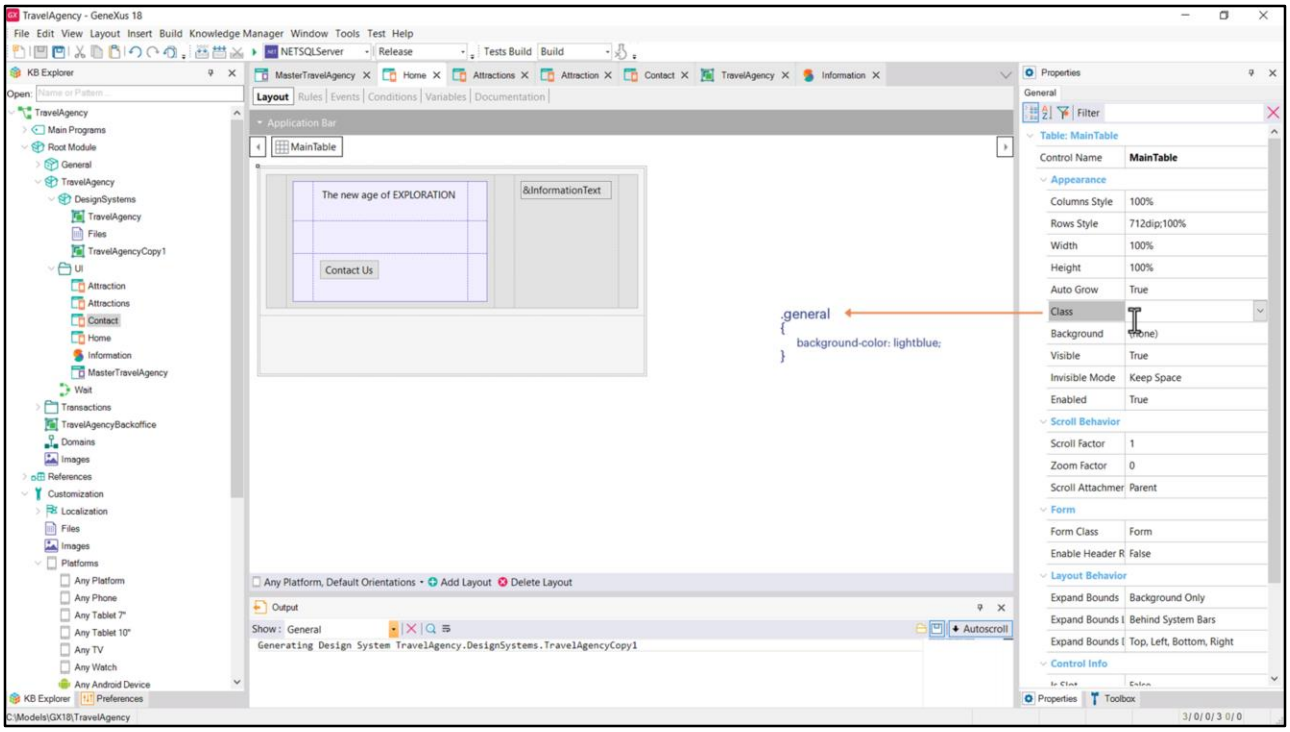



Antes de continuar con el desarrollo de la aplicación quisiera detenerme unos minutos, en este video y el que sigue, es decir en los videos de este módulo, antes de pasar al siguiente, para repensar algunas de las cuestiones que llevamos implementadas, para pensar otras alternativas, e incluso para ver cosas que antes no vimos en su momento, que pasaron desapercibidas.

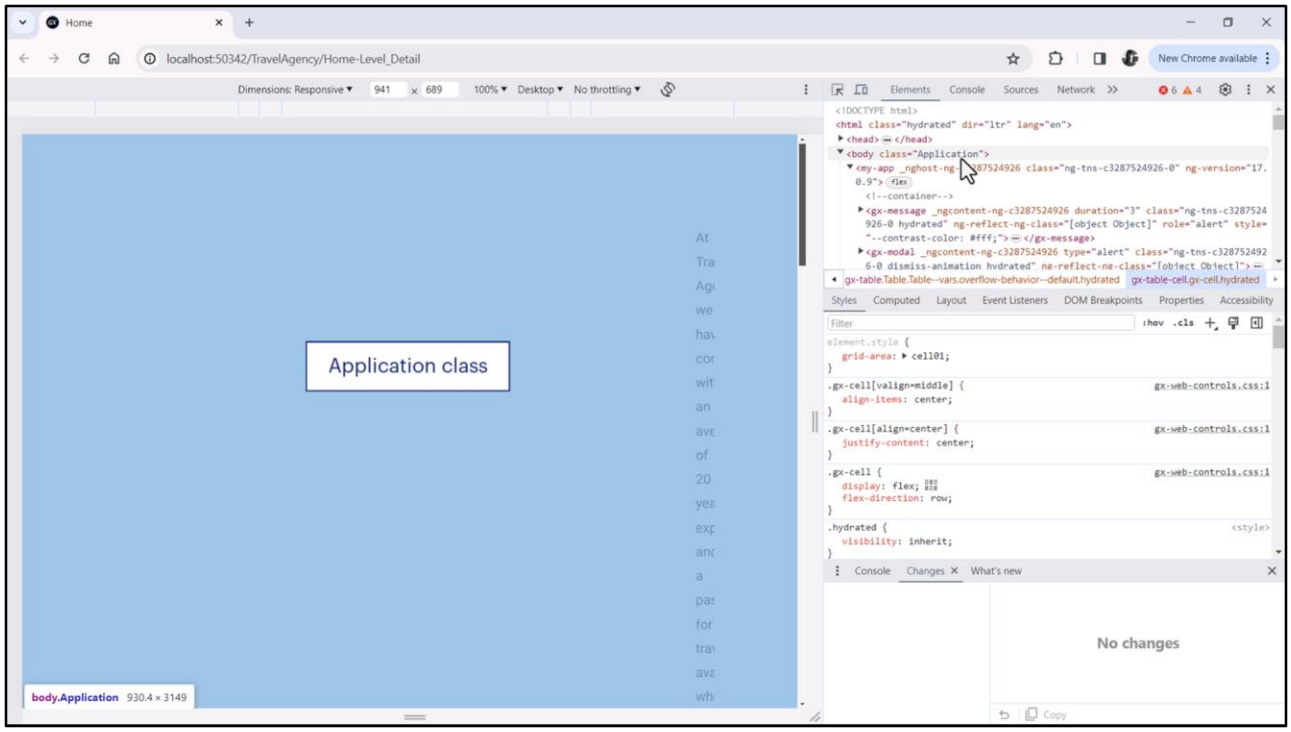
Y por último, sobre el final del video, vamos a ver cuestiones que hacen a la eficiencia en la etapa de desarrollo, en la prototipación. Cómo hacer que los tiempos de nuestro trabajo se reduzcan.




Entonces vamos a empezar por algo que nos pasó desapercibido y es que, por ejemplo, si quisiéramos que el color de fondo de nuestras pantallas no sea este blanco, sino sea por ejemplo lightblue, ¿cómo haríamos?



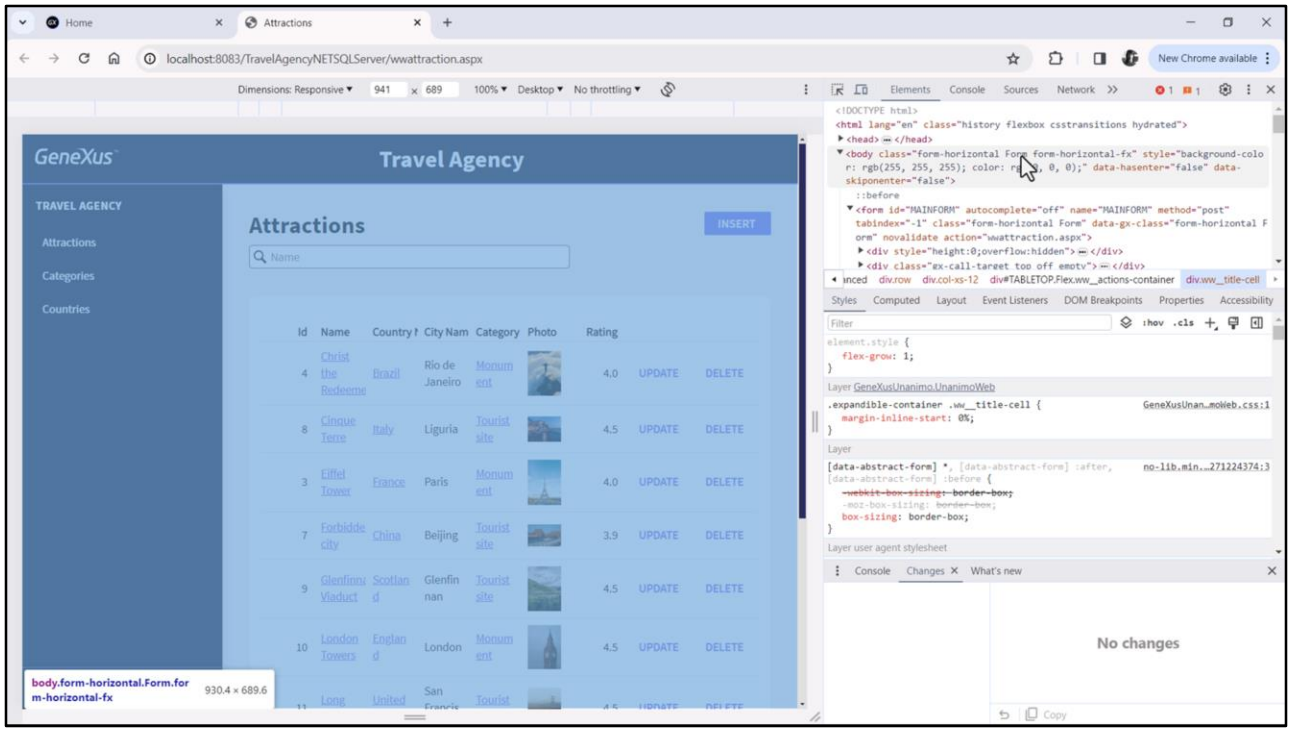

Podríamos crear una clase con la propiedad background-color lightblue y asociársela a la tabla main de cada panel. Pero cada vez tendremos que recordar hacer esta asociación, lo que no es tan práctico.



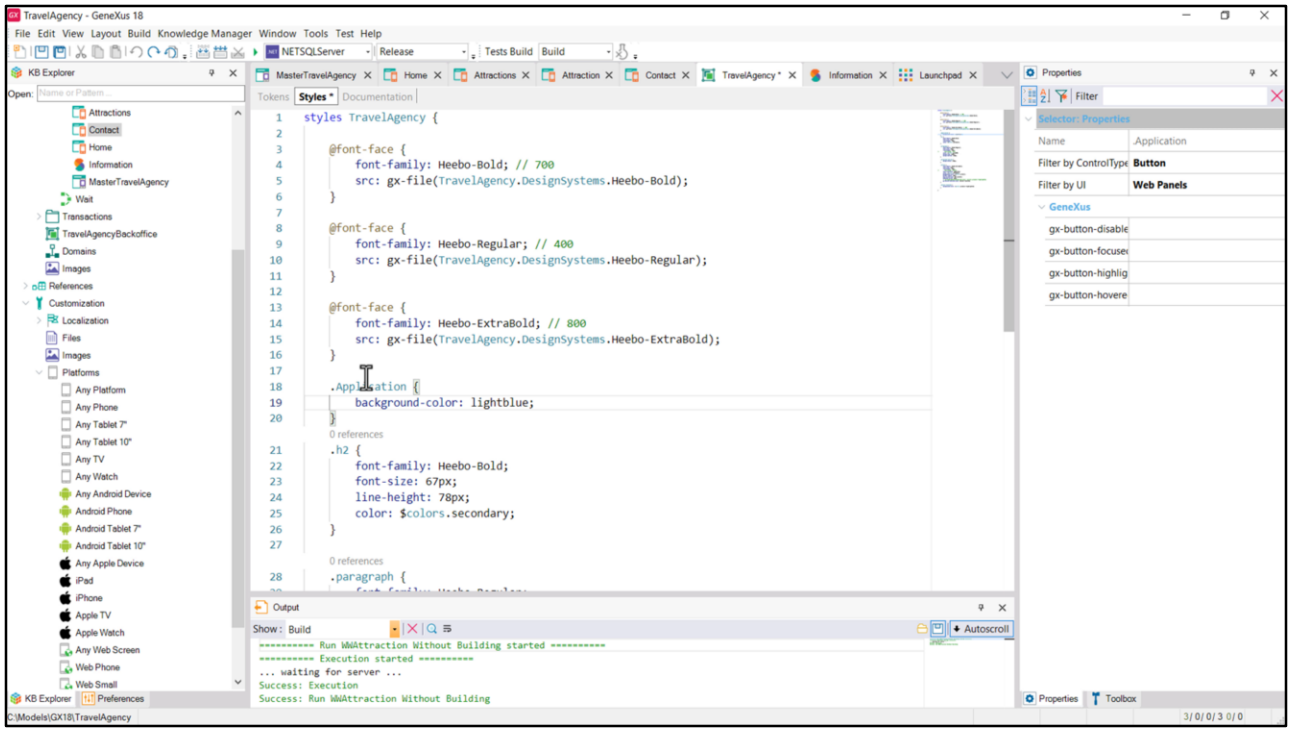

Si inspeccionamos el HTML de nuestra página Home, vemos que el tag del body tiene asociada la clase Application. Esto sucede en general con los frameworks como Angular o React. Todo tag html body tendrá esta clase asociada, que, por tanto, aplicará a todo lo que se ubique dentro del layout.

Y lo interesante es que esta clase **también** aplicará a los layouts de las aplicaciones nativas: Android, iOS. No importa que allí no haya html y la implementación sea completamente distinta. La clase Application tendrá el mismo sentido.

Por lo tanto, para los DSOs que aplicarán al mundo Panels, utilizar esa clase para configurar lo que queremos que valga para todas las pantallas es una buena idea, y económica.

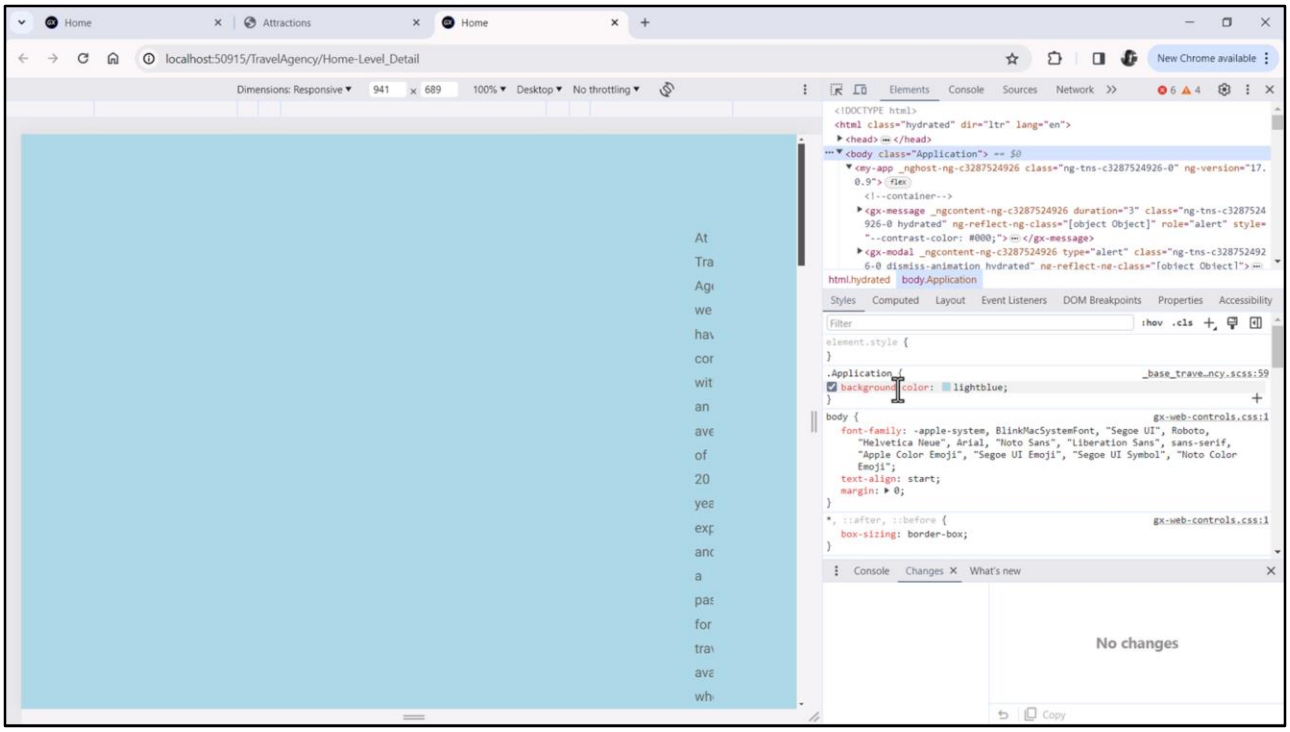



Si estuviéramos en el mundo Web Panels, como por ejemplo en el del backoffice, el tag body del html no tendrá la clase Application sino la clase Form, por lo que tendríamos allí que utilizar esta clase, en lugar de la Application.



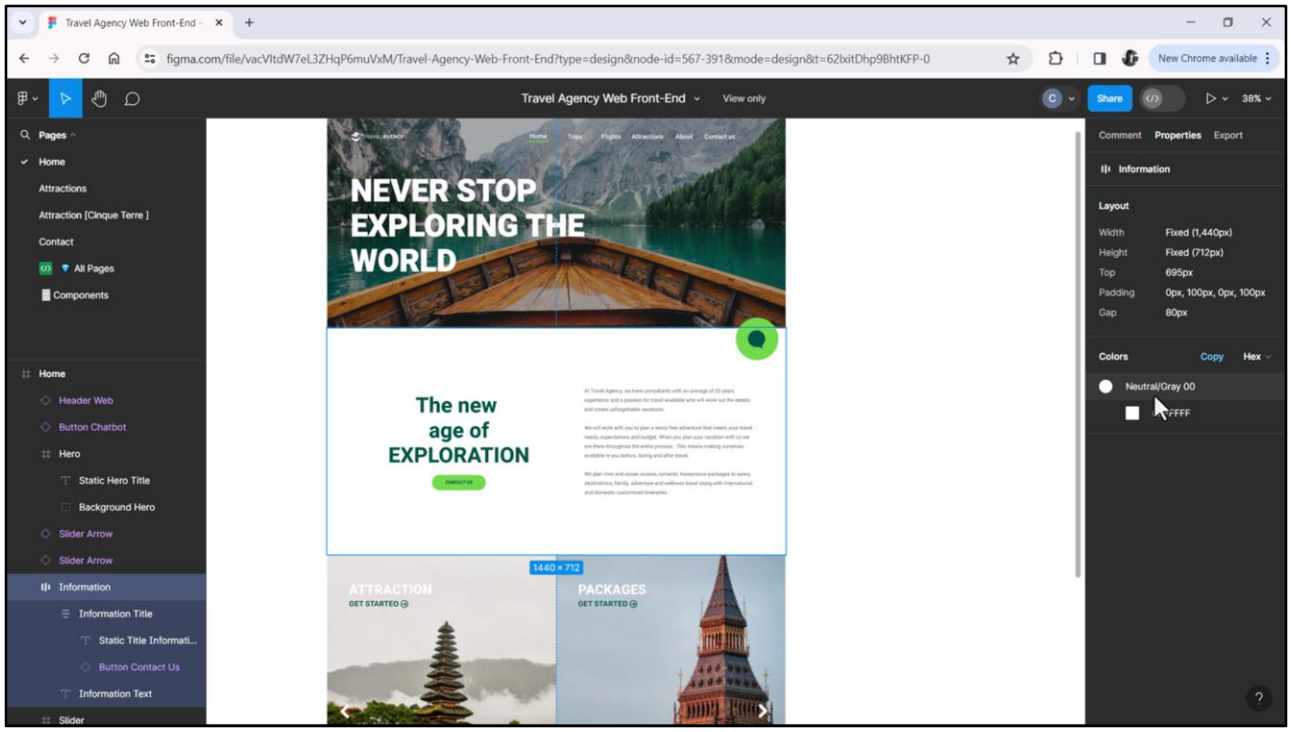

Bueno, pero como estamos en el mundo Panels, porque estamos implementando para Angular, probemos agregar la clase Application a nuestro DSO, con la propiedad background-color lightblue.

Y vuelvo a ejecutar.

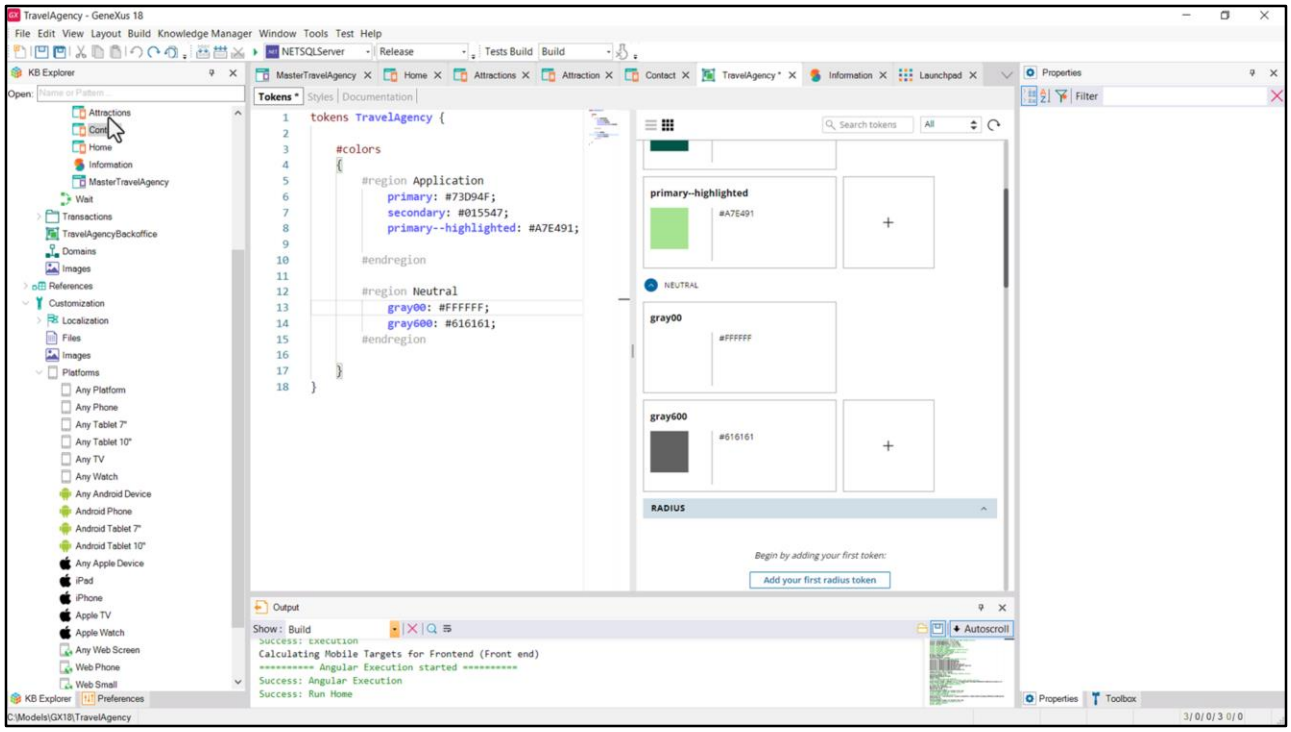



Bien, y si ahora inspeccionamos… aquí vemos la propiedad.

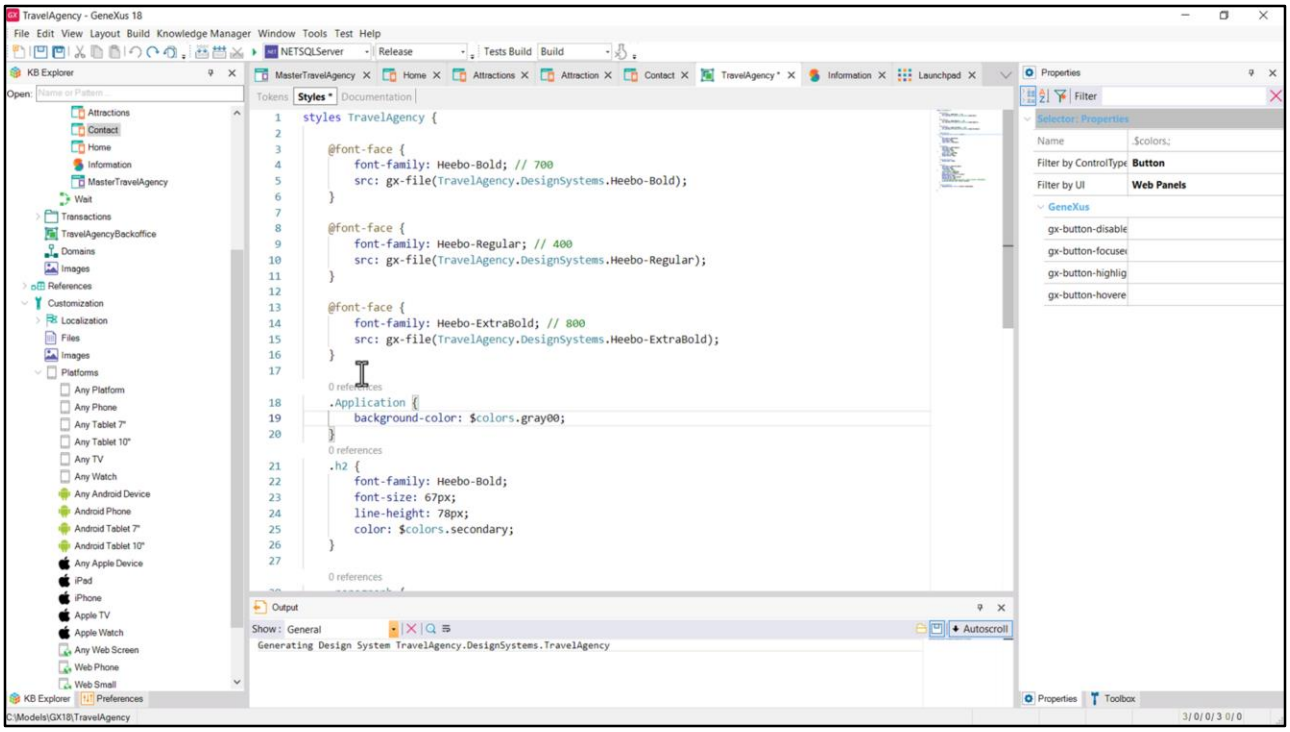
Una aclaración importante: Nos estamos valiendo de algo de bajo nivel que no es parte de GeneXus para modelar algo de nuestro Design System. Es algo básico de la construcción de un archivo HTML que luego del tag <head> venga el tag <body> que aparece una sola vez por HTML. Entonces nos estamos aprovechando de que para Frameworks como Angular a ese tag body se le asocia una clase Application (o Form si no estamos usando frameworks), para dar estilo general a las pantallas de nuestra aplicación, utilizando alguna de esas dos clases, pero si en el futuro al tag body no se le asociara más la clase Application todo esto nos dejaría de funcionar. Lo mismo que si para Web Panels no se le asociara más al tag body la clase Form. Habrá que estar atento porque de repente en algún upgrade futuro de GeneXus algo de esto se modifica, se brinda alguna propiedad especial para esto, o quién sabe. Volviendo…



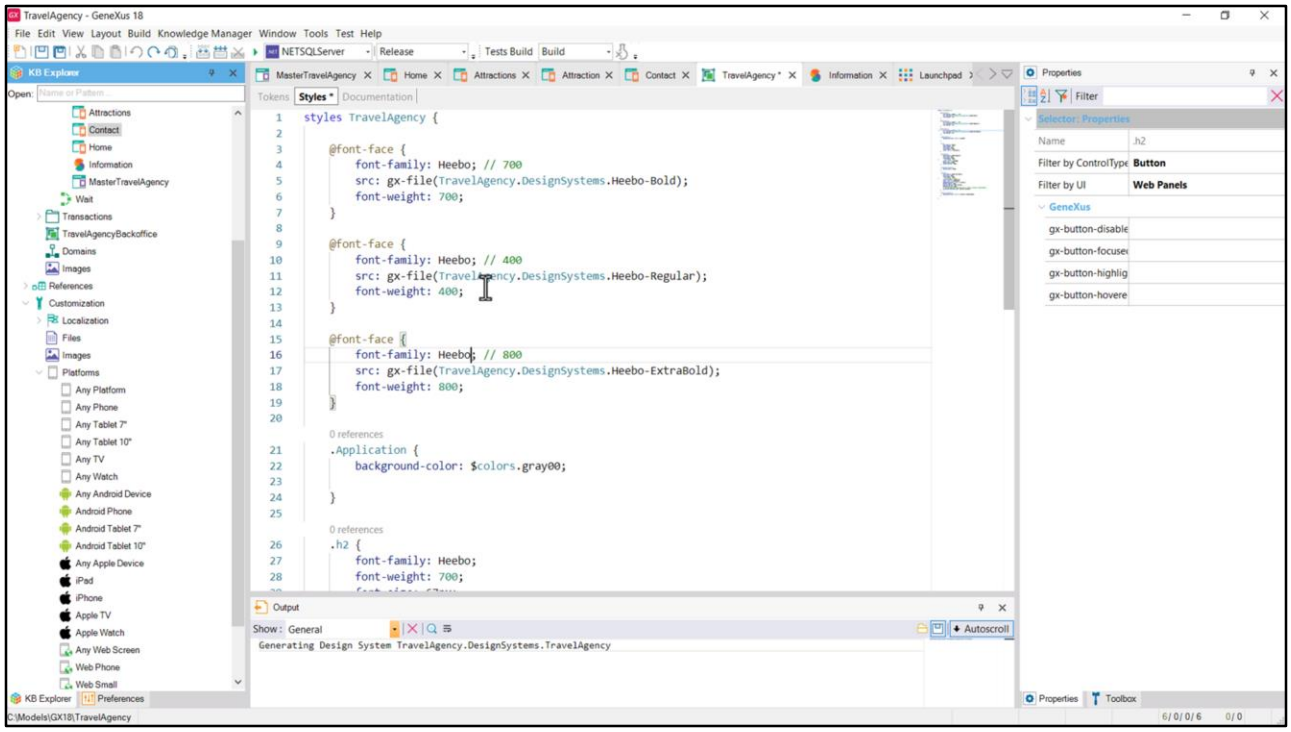

Ni nos habíamos ocupado del background-color porque no nos habíamos dado cuenta, debido a que el color de fondo es blanco, y ese es el default, pero para poder variarlo más adelante necesitaremos dejar explicitado valor en la propiedad, para poder cambiarla fácil en el futuro. Entonces primero agreguemos un token de color gray00…


TravelAgency - GeneXus 18
File Edit View Layout Build Knowledge Manager Window Tools Test Help
KB Explorer
Tokens *
Styles
Documentation
tokens TravelAgency {
#colors
#region Application
primary: #73D94F;
secondary: #015547;
primary--highlighted: #A7E491;
#endregion
#region Neutral
gray00: #FFFFFF;
gray600: #616161;
#endregion
}
primary--highlighted
#A7E491
NEUTRAL
gray00
#FFFFFF
gray600
#616161
RADIUS
Begin by adding your first token:
Add your first radius token
Output
Calculating Mobile Targets for Frontend (Front end)
========== Angular Execution started ==========
Success: Angular Execution
Success: Run Home
Properties
Filter

Listo.




Y ahora cambiemos el valor de la propiedad por el token.




Podríamos aprovechar esta clase que aplicará como raíz de todos los objetos con interfaz del mundo Panels para definir allí la familia de fuentes Heebo y no tener que repetirla en todos lados.

Ustedes me podrán decir, ¿pero cómo, si nunca repetimos la familia de fuentes hasta el momento? ¡Si para cada una de las 3 clases que tenemos, hay una Heebo, sí, pero es distinta! Es que, si recuerdan, lo que las diferenciaba, que era el peso, habíamos decidido expresarlo en conjunto con la familia.

Pero podríamos haber mantenido ambas cosas por separado. Así…

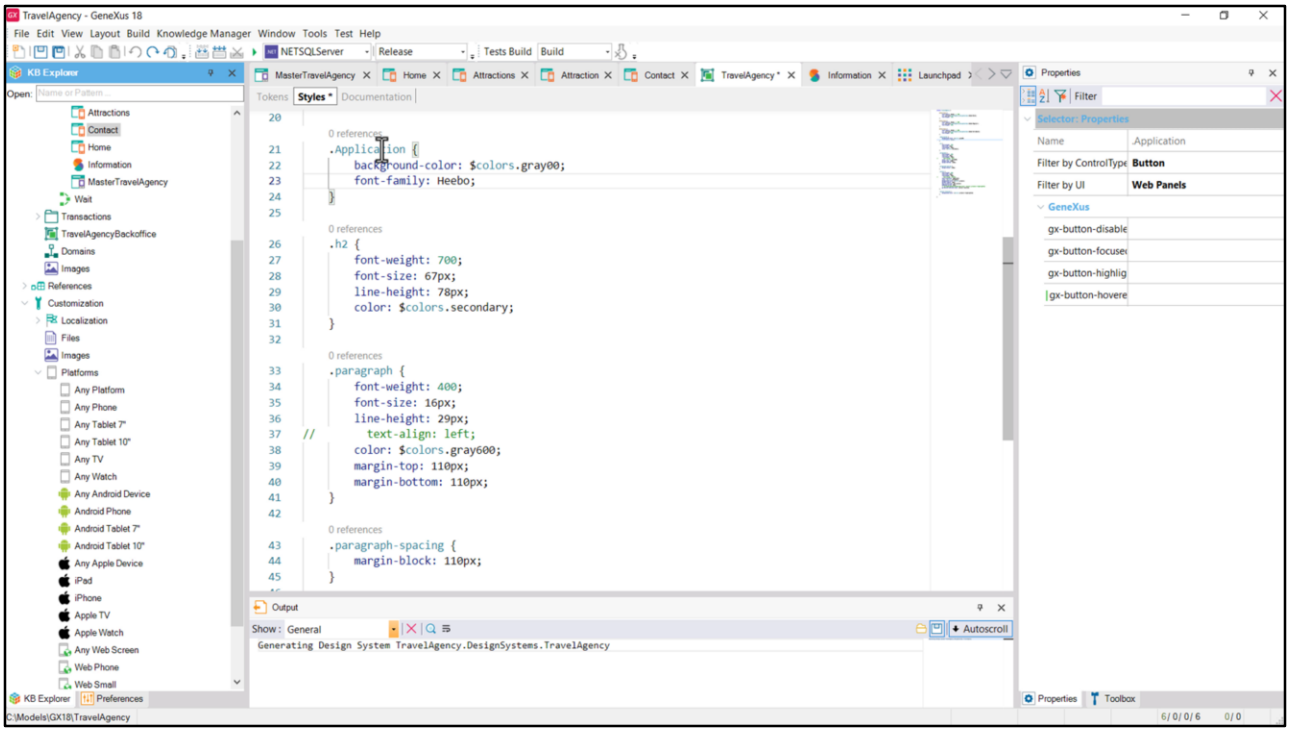
Le quito aquí el nombre que la diferenciaba, le dejo el genérico Heebo… que es el que coloco también aquí…

A todas les voy a quitar lo que las diferenciaba por nombre. Ahora las tres se van a llamar Heebo, y lo que las va a diferenciar va a ser el peso.

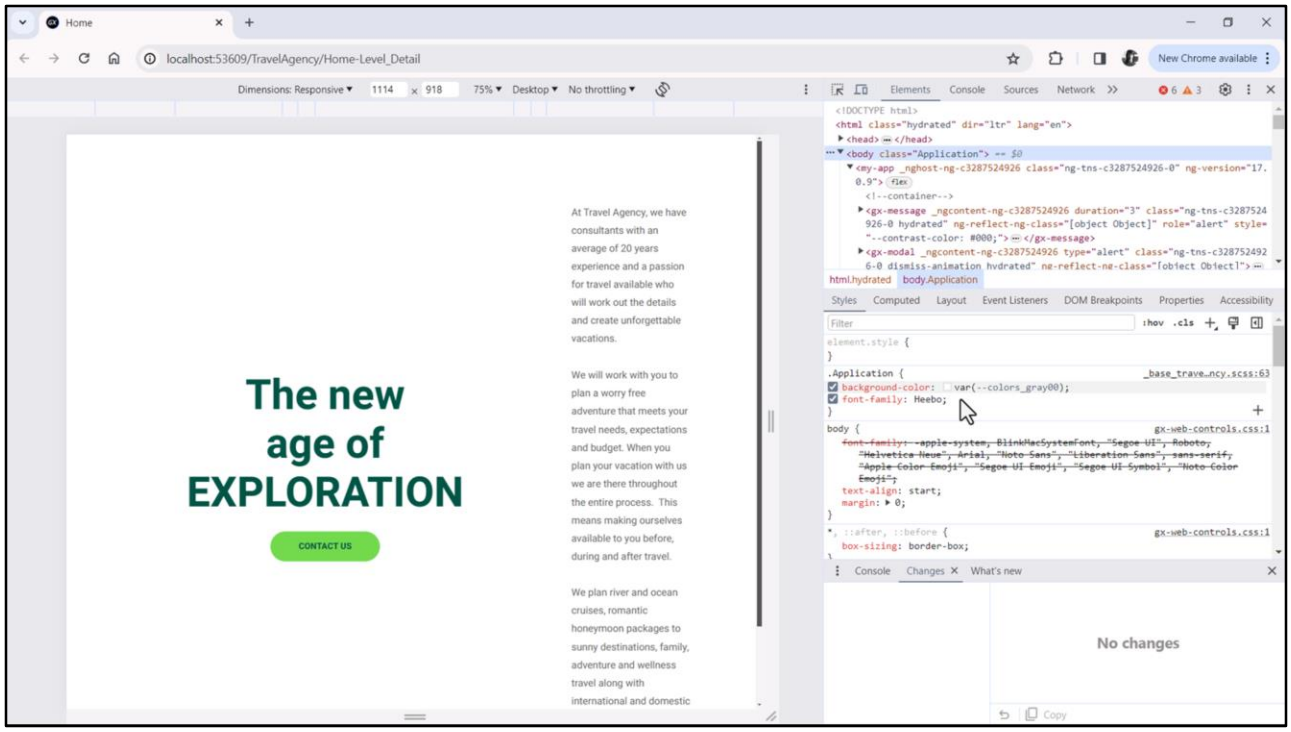
Entonces en la clase h2 digo que es la Heebo de peso 700; en la paragraph voy a decir que es la Heebo también, pero la de peso 400; y en la button diremos que es la de peso 800, Heebo también.

Entonces para identificarla ahora daremos estas dos dimensiones, porque la diferencia entre las Heebo vendrá dada por el peso.

Con estas dos dimensiones es que sabemos cuál de los archivos de fuentes va a tener que utilizar.

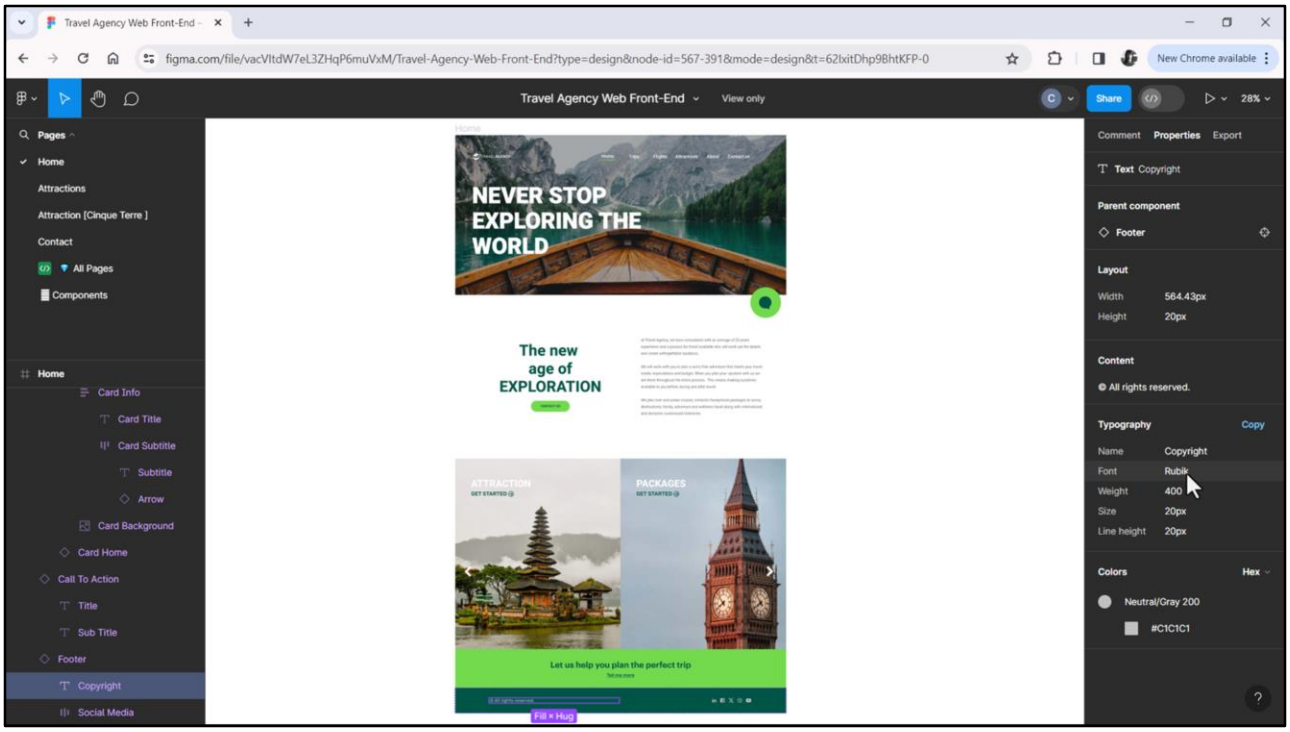



Entonces dado que la familia de fuentes se repite en todas las clases que tenemos hasta el momento podría aprovechar para especificar esa familia de fuentes como el default, y entonces quitarla de aquí, de aquí y de aquí… y colocarla dentro de la clase Application. De esta manera, como esta clase es la de la raíz de cada objeto con interfaz, esta va a ser la familia de fuentes default, es decir, la que va aplicar si no se especifica otra cosa en las clases.

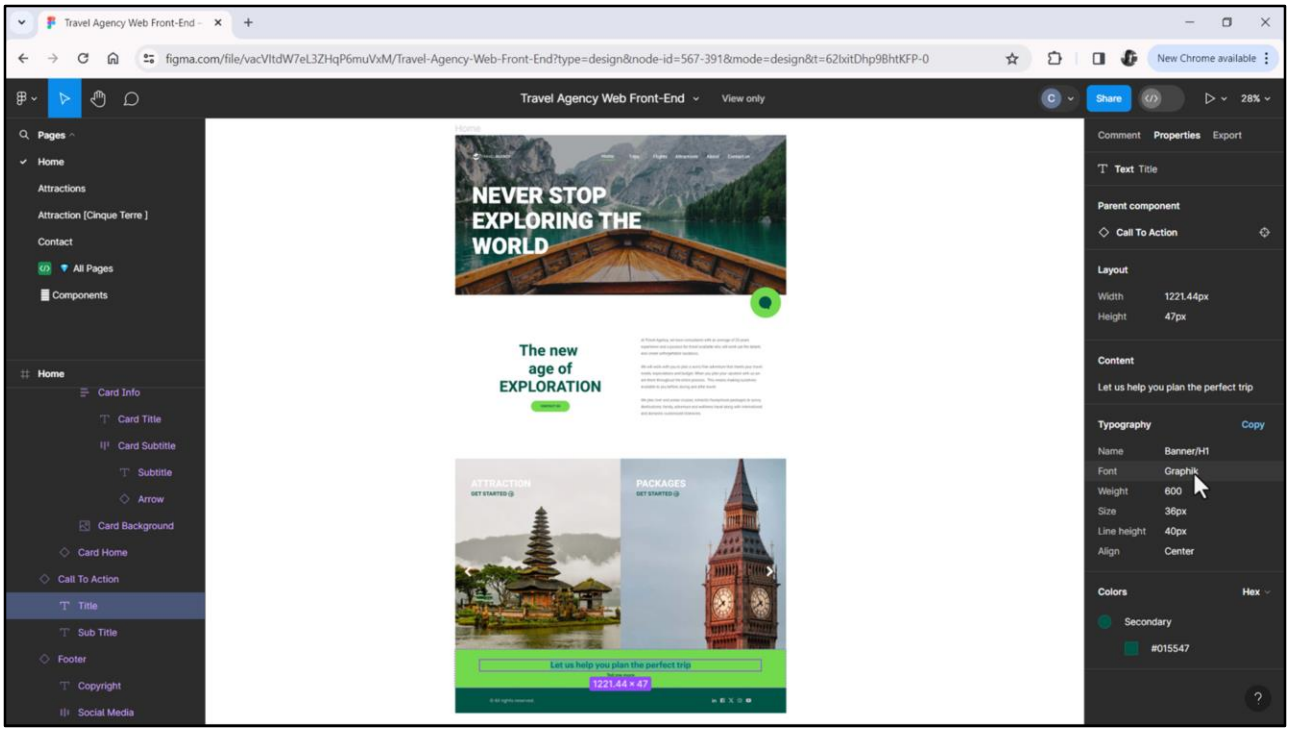



Si ahora probamos esto… vamos a volver a ver el fondo blanco, de acuerdo a este token, pero en lo demás, no nos vamos a dar cuenta visualmente de la diferencia, obviamente.

Pero si inspeccionamos… vemos allí las dos propiedades dentro de la clase Application, y, no apreciamos ninguna diferencia respecto a cómo habíamos implementado el asunto de las fuentes antes. Así que es otra manera de lograr lo mismo.

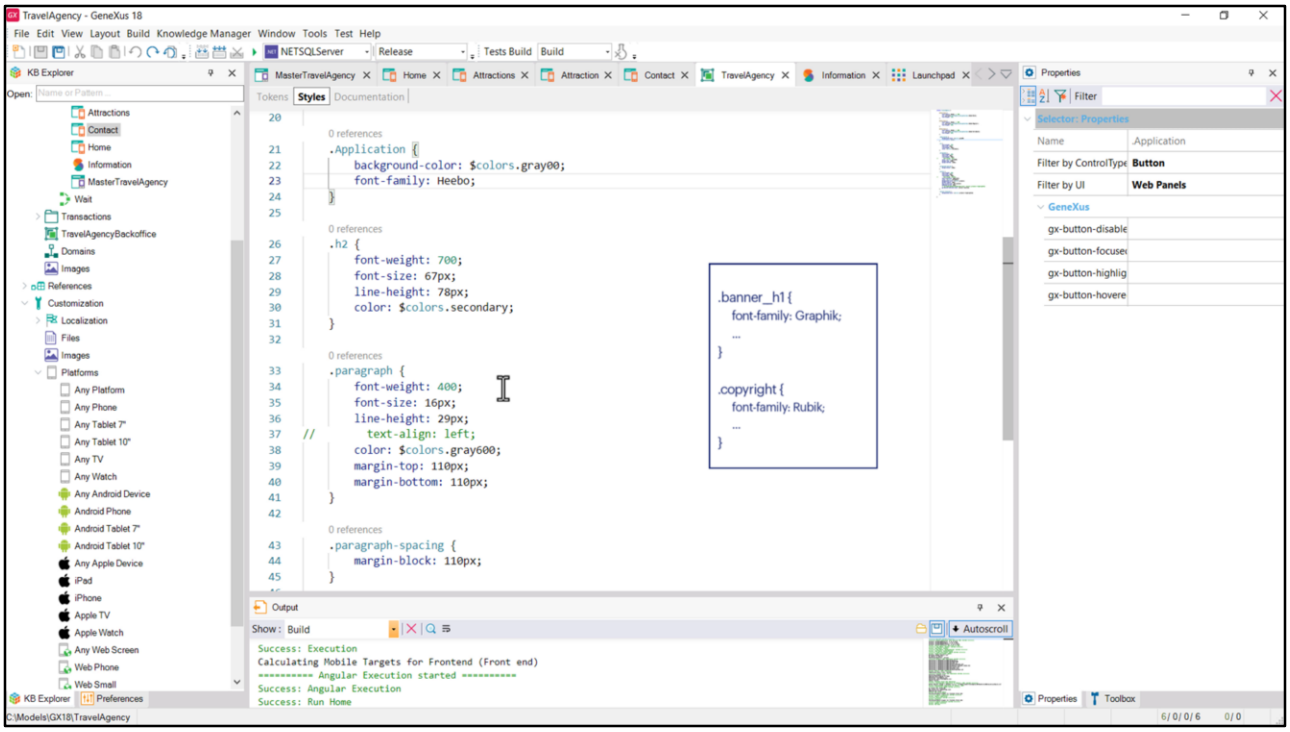



Por otro lado, si venimos a Figma a investigar los estilos tipográficos que creó nuestra diseñadora para las pantallas, y analizamos, por ejemplo, el de copyright, vemos que no está utilizando el font Heebo sino que está utilizando este Rubik.




Y para este otro texto, el principal del Banner, vemos que utiliza el font Graphik.

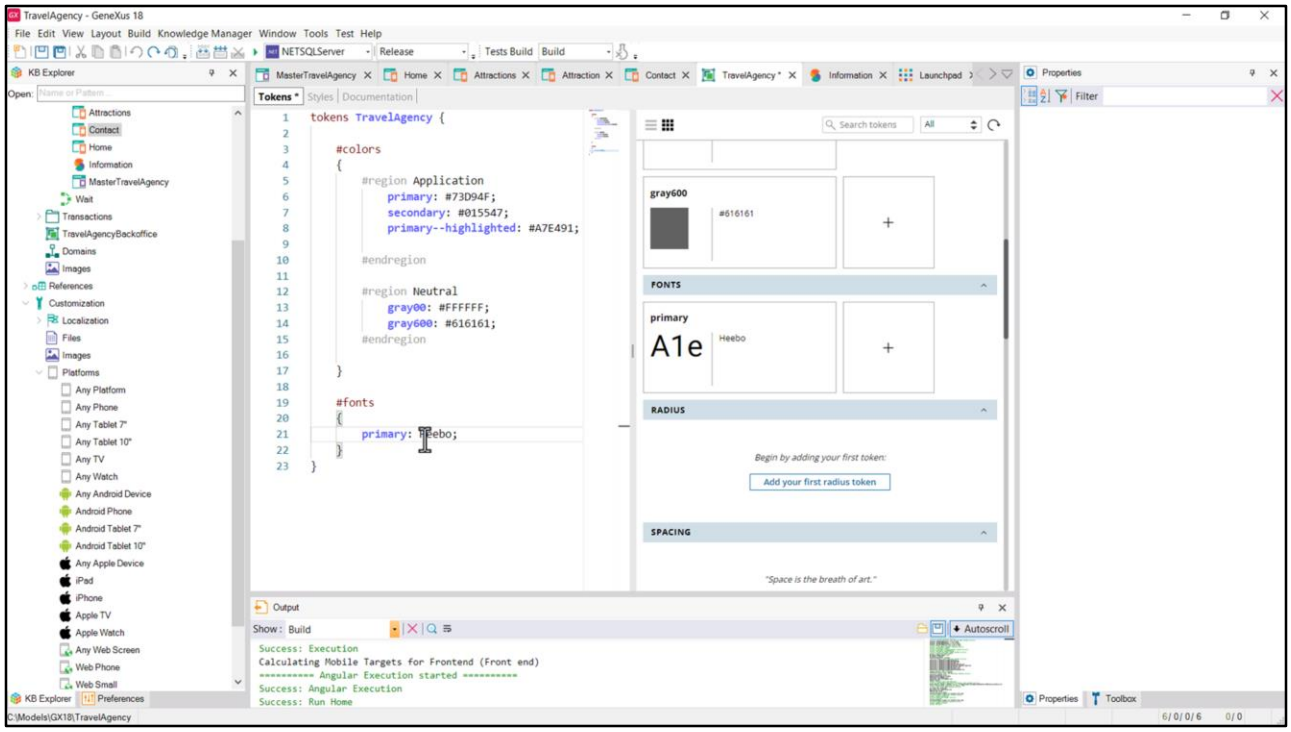
Podemos concluir que no será la única familia de fuentes la Heebo. Habrá otras dos.




Lo que sabemos es que la fuente Heebo va a ser al fuente principal de la aplicación, la primaria, y que solo en un par de excepciones se utilizarán otras familias de fuentes.

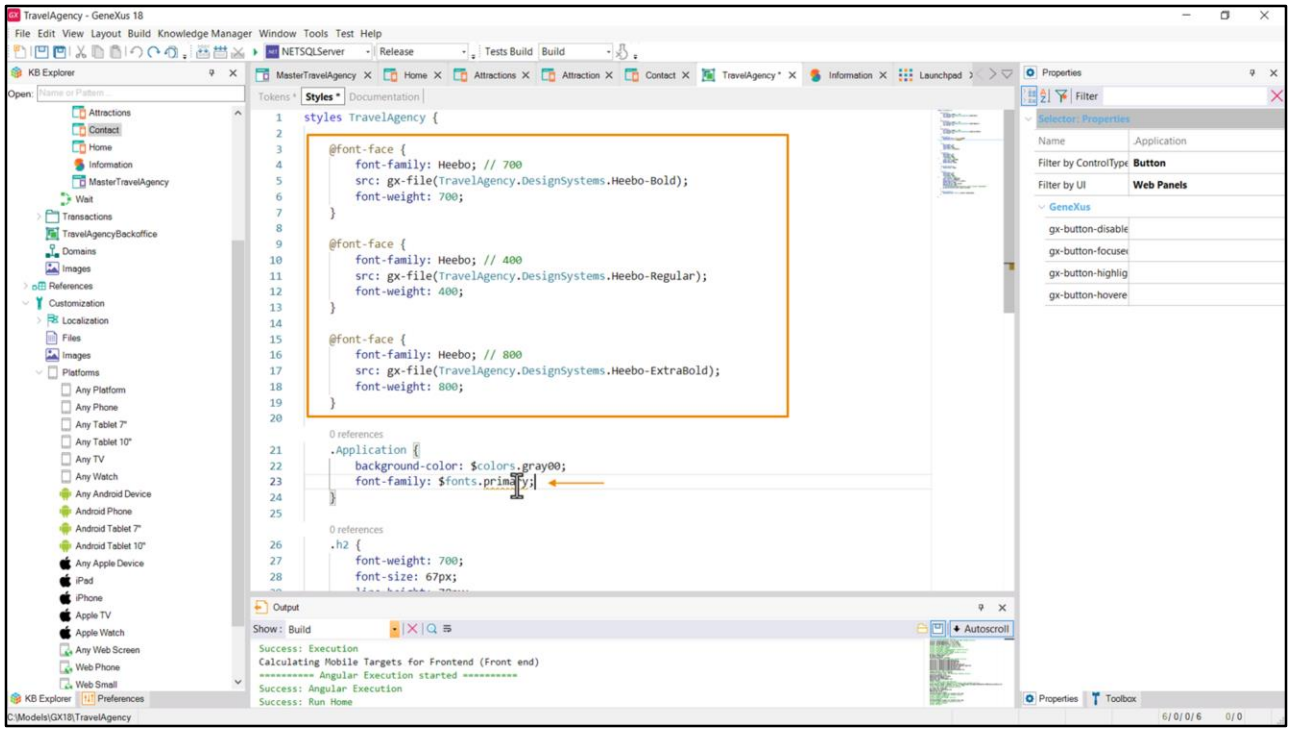
Para las clases que utilicen esas otras familias, las de los textos que veíamos recién en figma, alcanzará con explicitar la familia, para un caso será la Graphik, para el otro la Rubik, y con eso dejarán de utilizar esta default que es la Heebo, default a partir de que la colocamos dentro de la clase Application.

Lo que podemos hacer para conceptualizar mejor todo, dado que vamos a tener 3 familias de fuentes distintas, y una, esta, es la principal o primaria, es crear un token por cada una, para abstraerlas de acuerdo a su función.



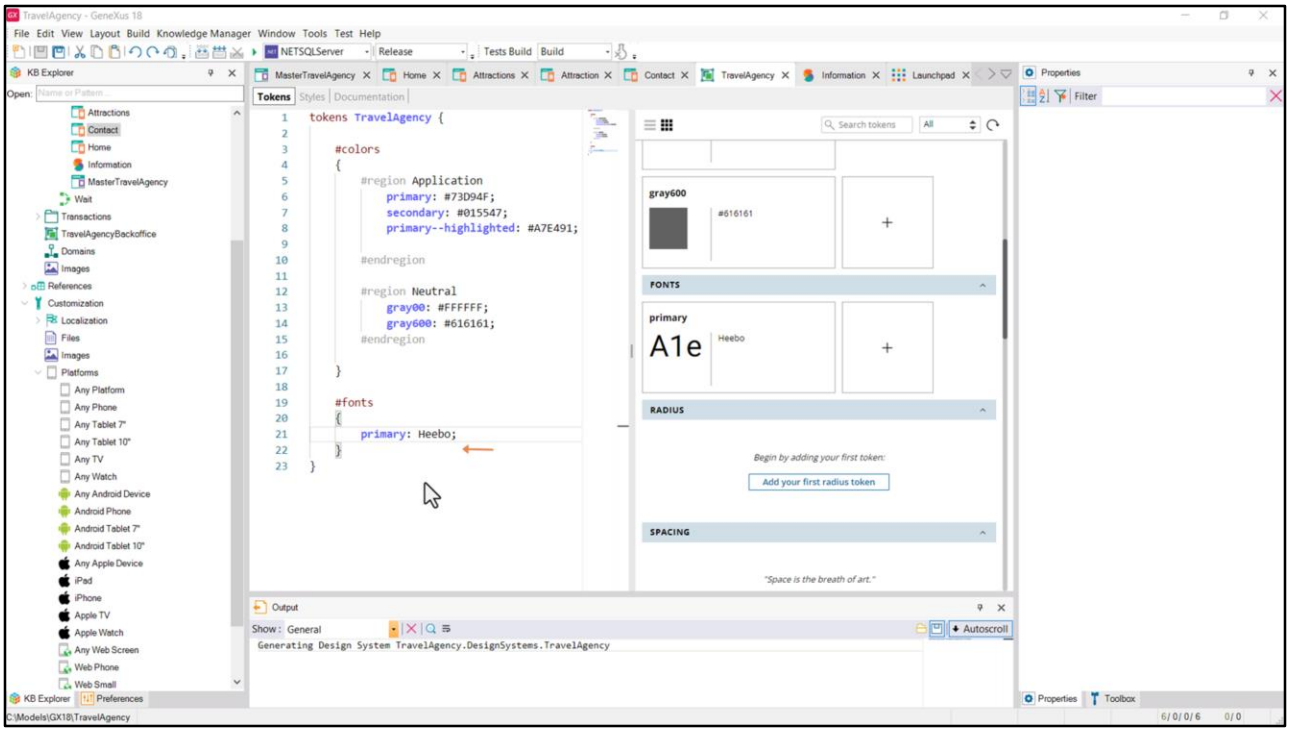

Los tokens no son solo de color. Podemos definir tokens de otros tipos. Por ejemplo, para fuentes.

Desde este editor nos permite agregar un nuevo token de este tipo, fonts. Voy a llamarle primary a mi fuente default, que es la Heebo. Este nombre de aquí va a corresponder…

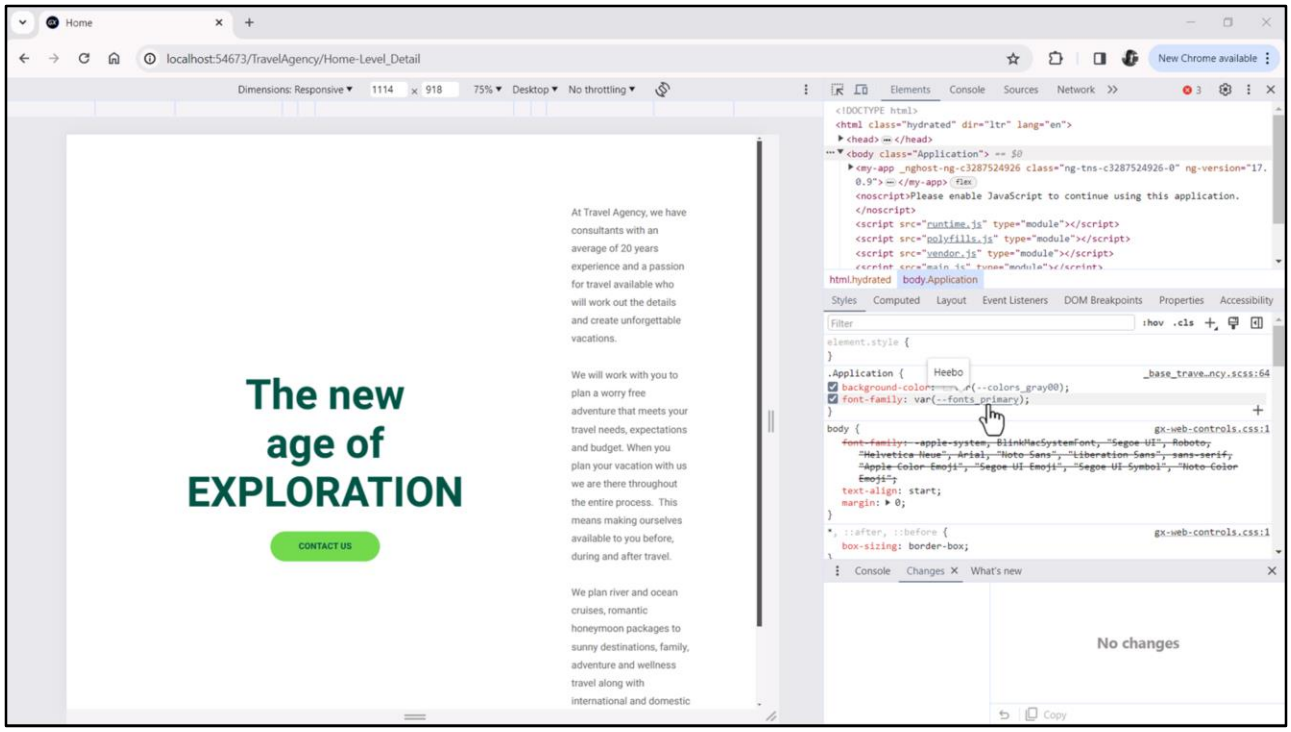



…a estas tres entradas.

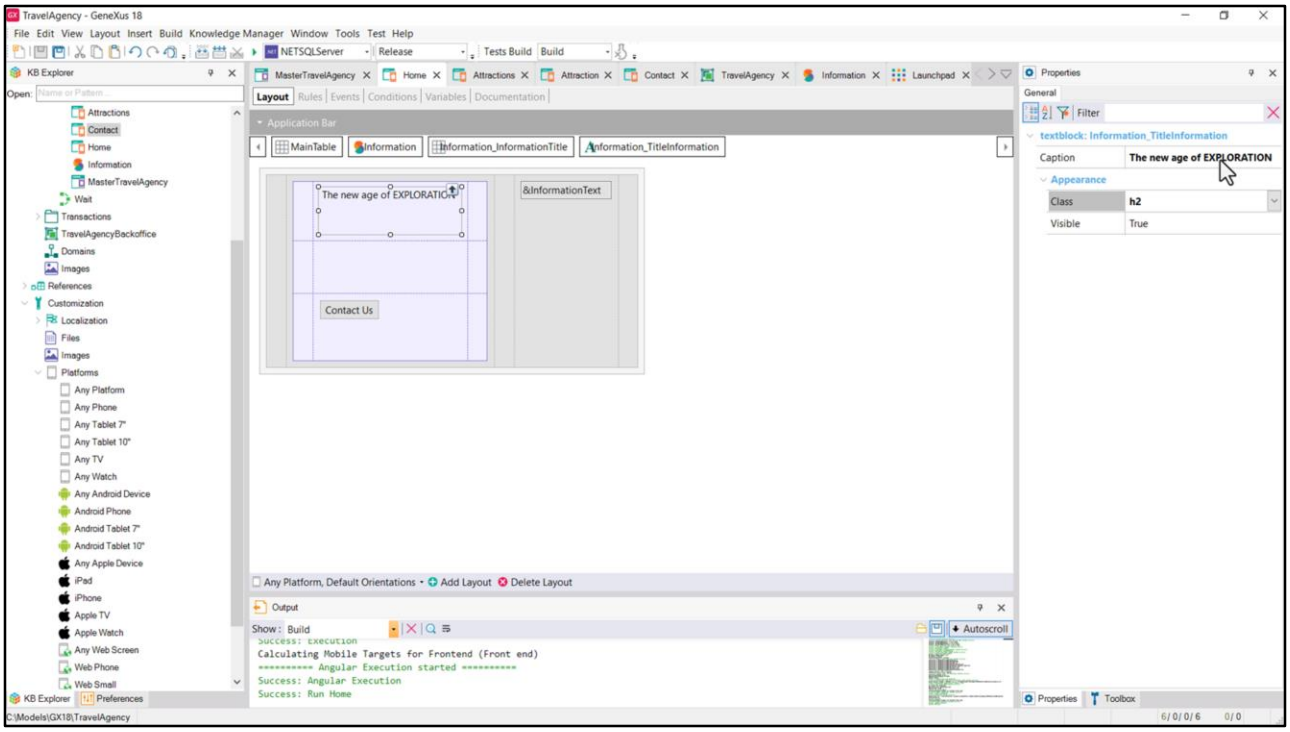
A partir de ahora en las clases puedo utilizar el token en lugar del valor concreto. Es decir, la familia de fuentes default será la indicada en el token primary. Primary de fuentes y no de colores. No se confunden porque están dentro de categorías distintas.




Vamos a grabar. Por supuesto, cuando necesitemos utilizar las otras familias de fuentes, será conveniente crear tokens aquí para ellas. De esta manera resultará bastante sencillo cambiar las familias de fuentes de la aplicación sin tener que recorrer la pestaña de estilos para modificar en cada clase la fuente que se está utilizando.  Y esta es la gran ventaja de tokenizar.

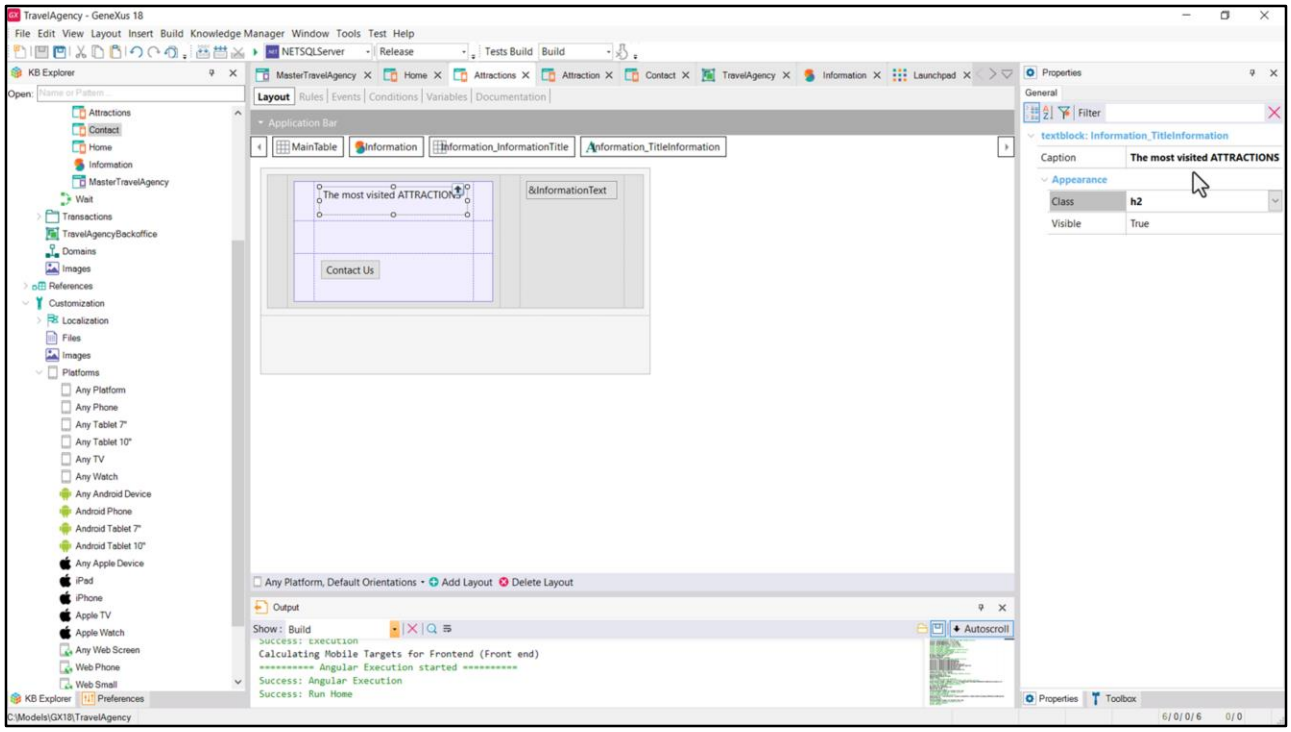



Si queremos probar para cerciorarnos de que todo va bien…. Parece que sí, que todo va bien. Observemos aquí en la clase Application cómo luce el token de la familia de fuentes.




Bueno, ahora quisiera aprovechar para discutir otra cosa que pasó desapercibida. Y tiene que ver con el Caption que le asignamos a este control text block, pero en el panel Home.

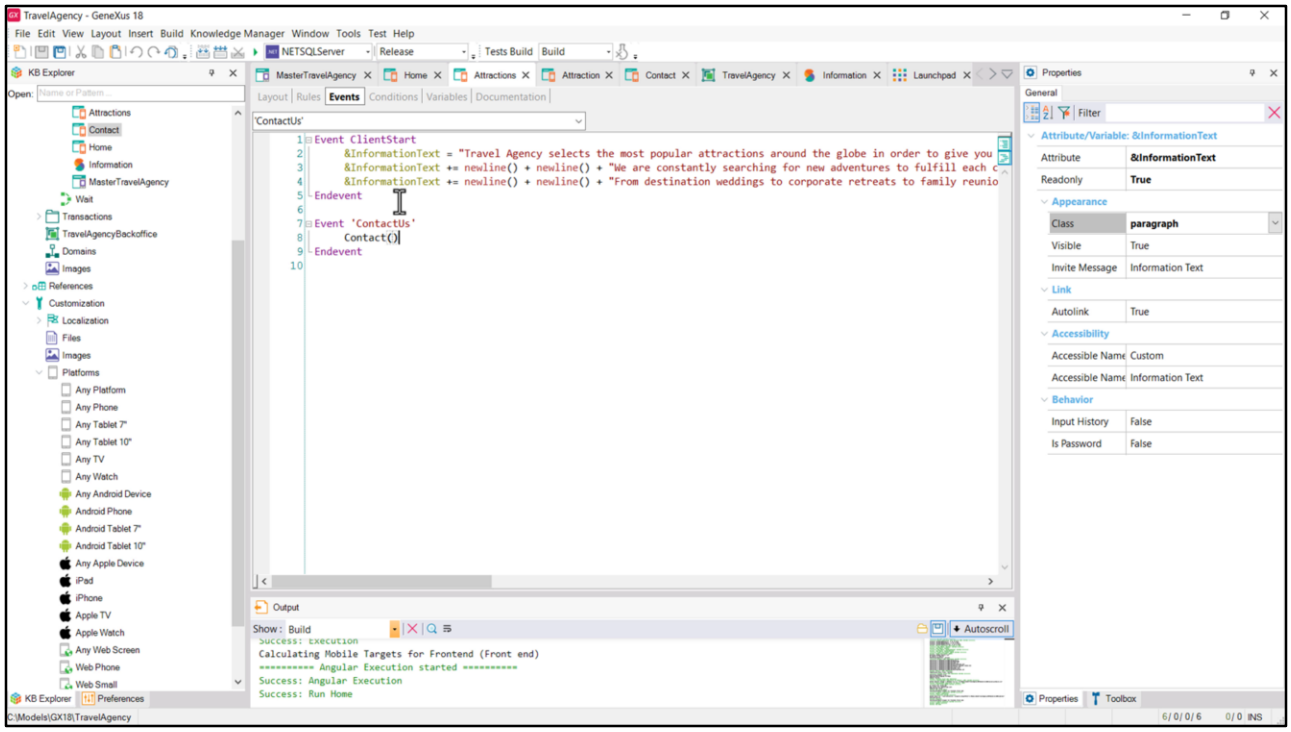
Habíamos empezado por asignarle en forma estática al textblock instanciado en el panel Home este caption…




…y a la instancia del textblock en Attractions también le habíamos asignado en forma estática a nivel de la propiedad del control este caption.

Como esta es información estática configurada en las propiedades del propio control, cuando en el browser se carga el panel por primera vez ya se cuenta con esta información y se puede, entonces, renderizar inmediatamente en la pantalla del navegador.

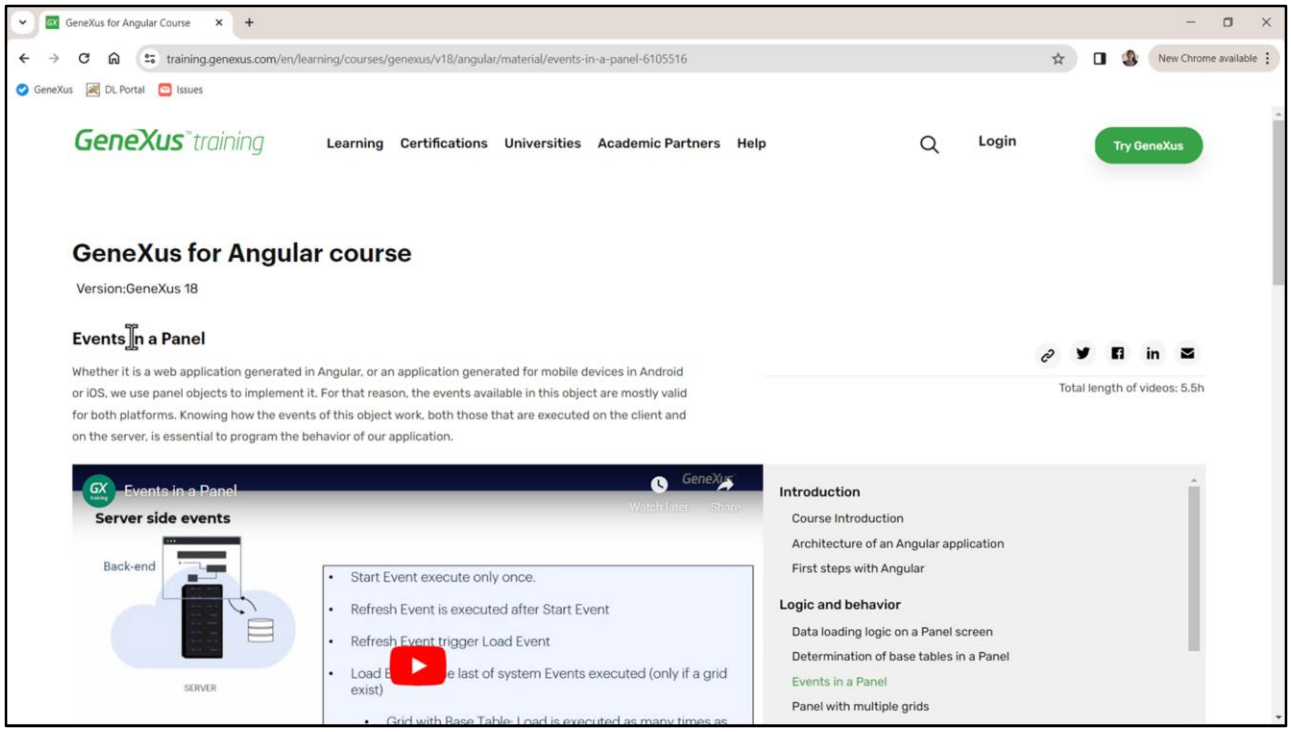
Luego se producirá, también en el cliente, el evento ClientStart, que si no tiene que invocar a ningún servicio en el servidor se ejecuta completamente allí mismo.



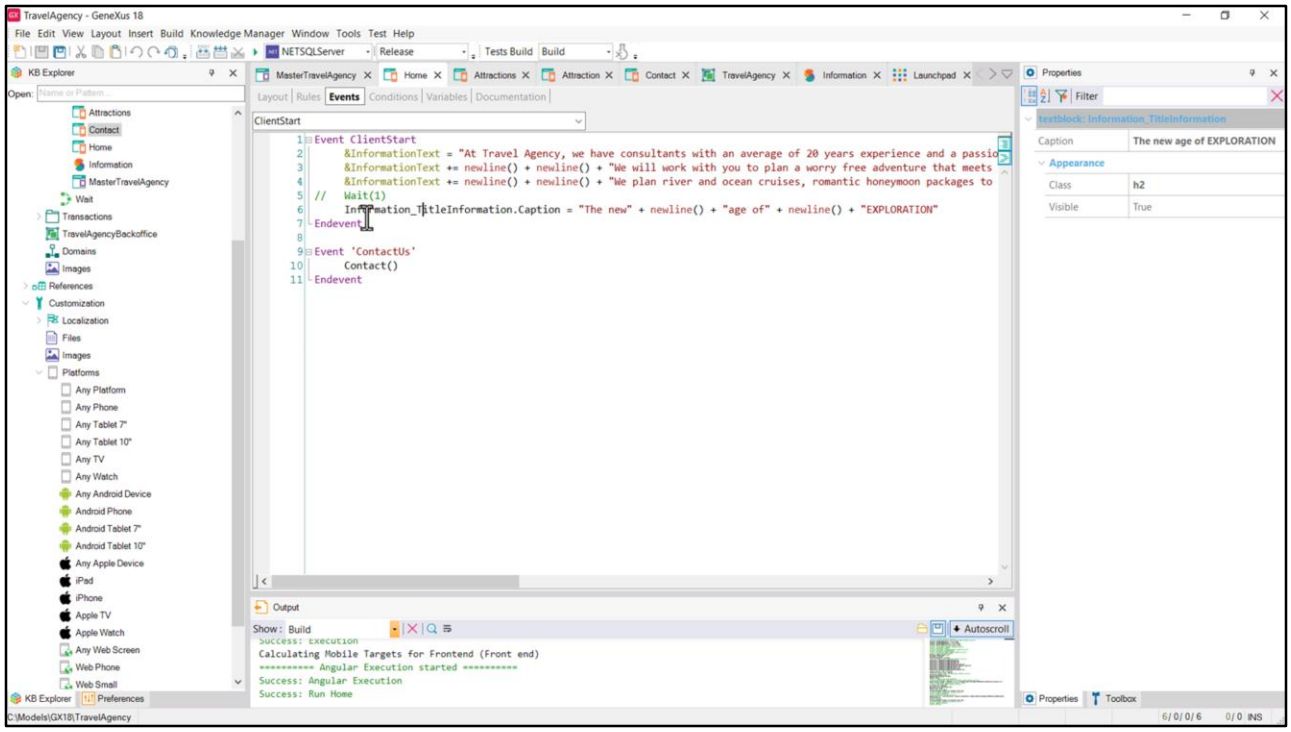

En el evento ClientStart era donde cargábamos el contenido de esta variable, ¿se acuerdan?. Observemos allí cómo en el panel Attractions le asignábamos valor.

Para la variable no teníamos otra opción que cargarla por código, porque no tenemos para las variables una propiedad que permita ingresarle contenido.

En este caso, como su contenido era también estático, conocido desde el vamos, estas asignaciones se ejecutarán inmediatamente, en el cliente, y por ello cuando carguemos la página en el browser…. nuestro ojo percibirá que se carga el caption del texto y el texto de la variable a la vez, sin ningún parpadeo ni delay ni nada por el estilo.

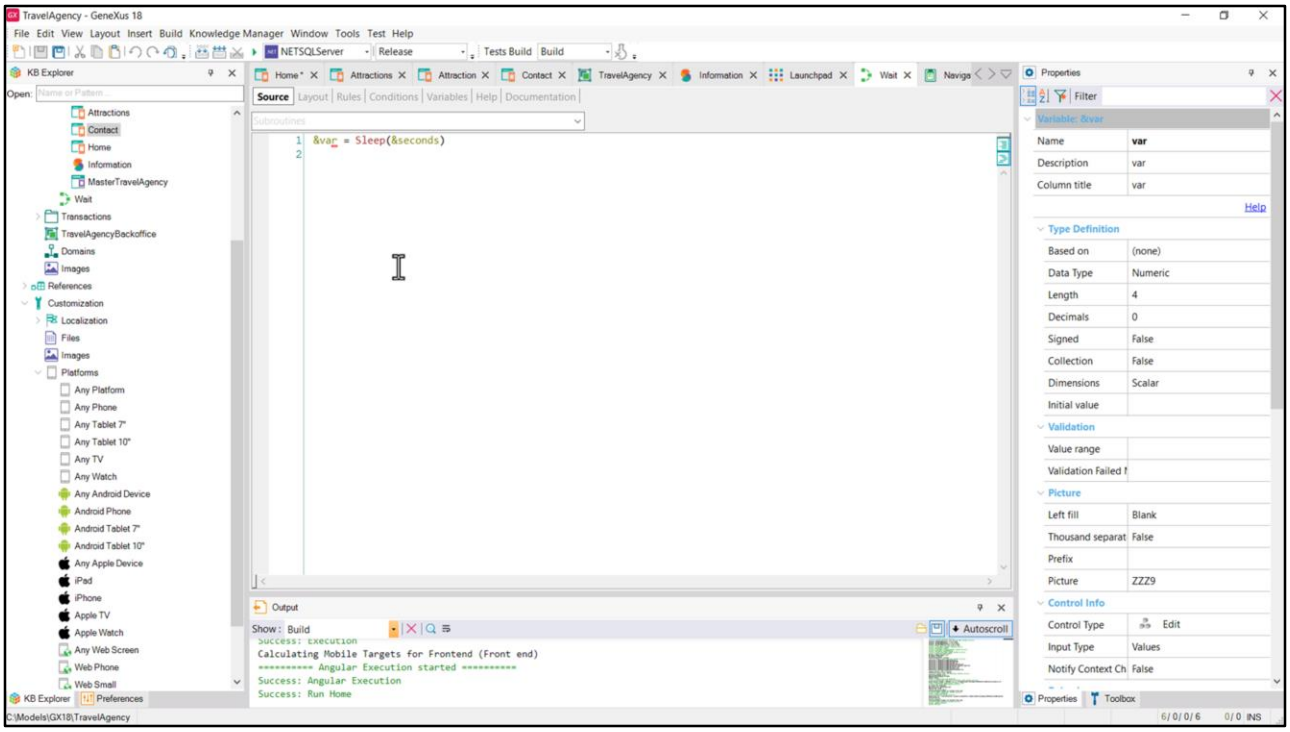



Antes de continuar les recomiendo repasar o ver por primera vez si no lo hicieron antes, este video en el sitio de training de GeneXus del curso de Angular (que también explica los eventos para las aplicaciones nativas).
Allí se estudian los eventos que se disparan en el cliente, los que se disparan en el servidor y su orden.




Volviendo a lo nuestro, para el text block en el panel Home habíamos tenido que asignar el caption por código, para poder separarlo en 3 líneas, ¿se acuerdan?
Aquí solamente cambié el orden de las asignaciones (primero venía la de del textblock y luego las de la variable pero ahora las invertí)… y coloqué esto, que como por ahora está comentado no importa.

Cuando implementamos esto no tuvimos el cuidado de, una vez agregada la asignación aquí, quitar, entonces, el caption fijado a nivel del control en la propiedad. Así que tenemos dos asignaciones del caption.




Para que nos quede más claro el efecto que puede llegar a tener esto, es que agregué (la descomento) esta invocación a un procedimiento después de cargada la info de la variable pero antes de cargar el caption del textblock.

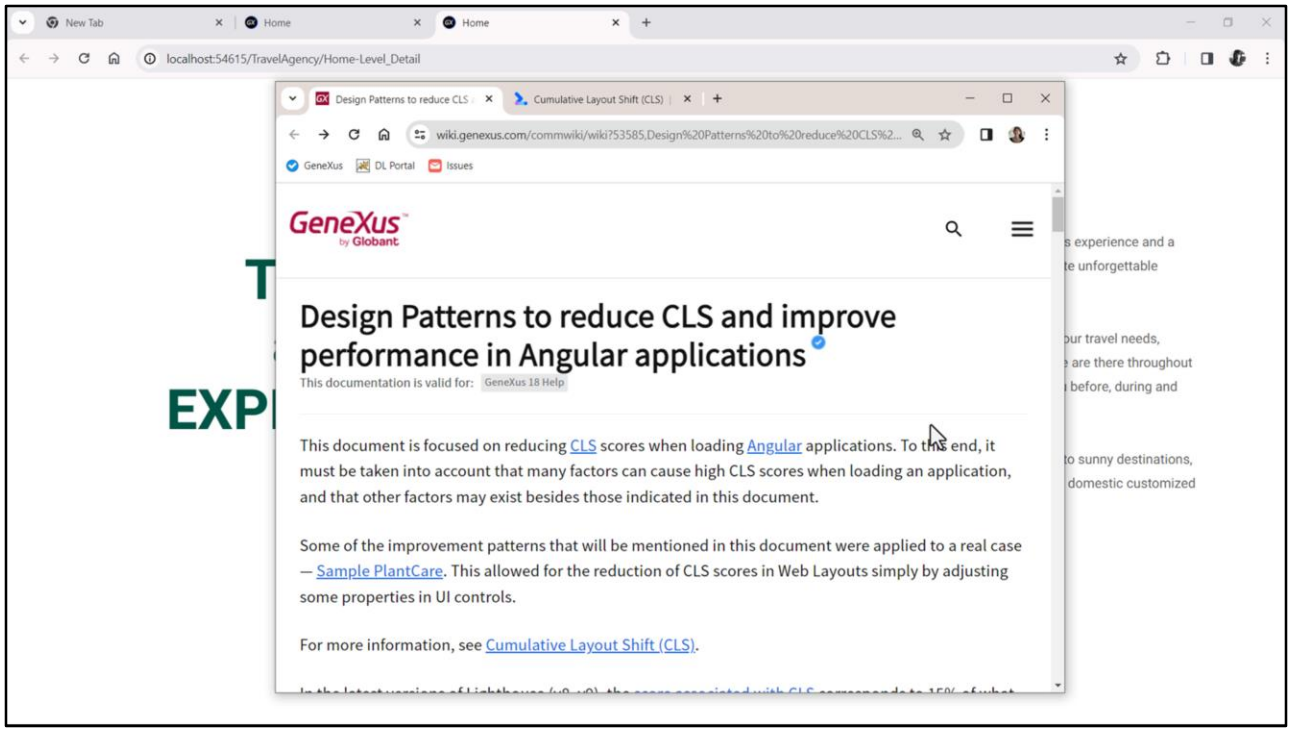
Esta invocación será a un procedimiento que creé previamente, que por ser un procedimiento se ejecutará en el servidor.

Así que del cliente tendrá que ejecutarse **como servicio** este procedimiento del server. Y lo único que hace este procedimiento es llamar a la función sleep, que implementa una pausa en este caso de la cantidad de segundos que el enviemos por parámetro, que va a ser de 1. Terminado ese tiempo el procedimiento culmina y la ejecución continúa en el cliente, en la sentencia siguiente, ejecutando, entonces, la asignación del caption.

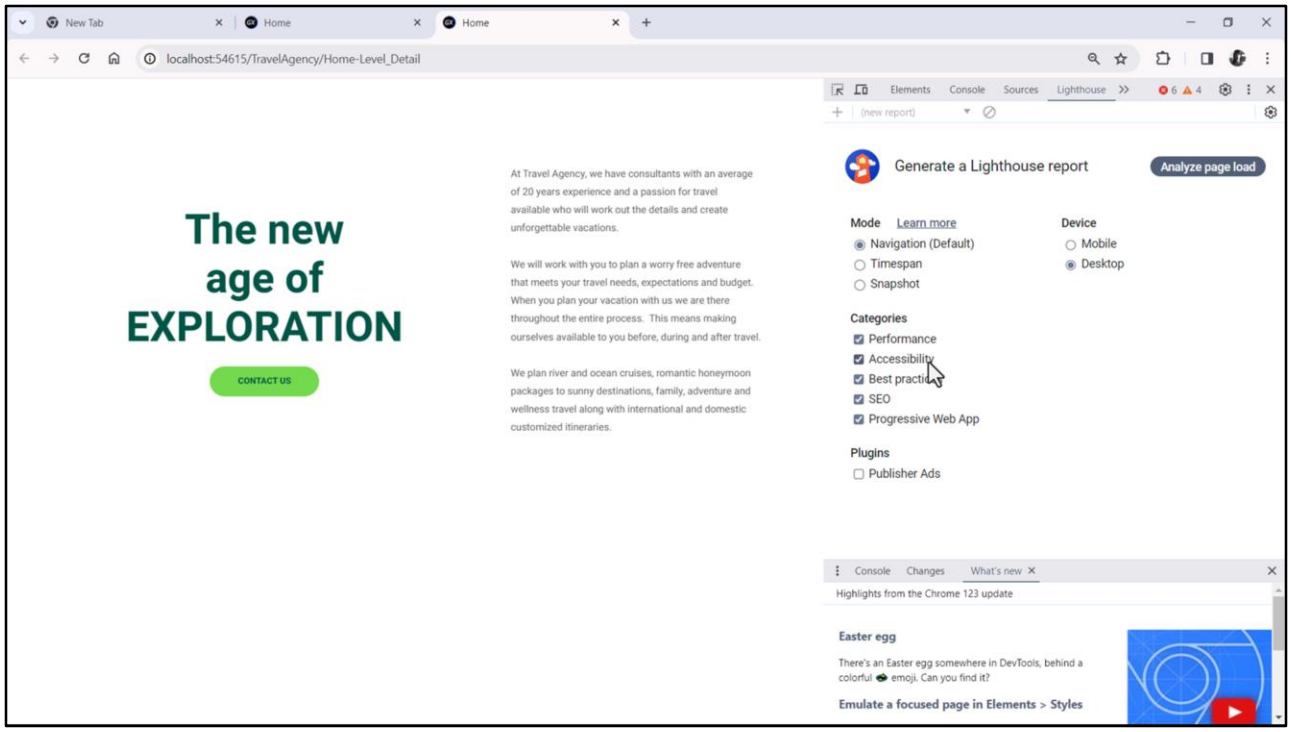
La mejor manera de entender lo que quiero mostrarles será ejecutando.


¿Se vio bien, no? Si la aplicación se comportara así, con ese cambio inesperado en el layout, el usuario seguramente se sentirá incómodo y nuestra aplicación bajará varios escalones en su valoración.

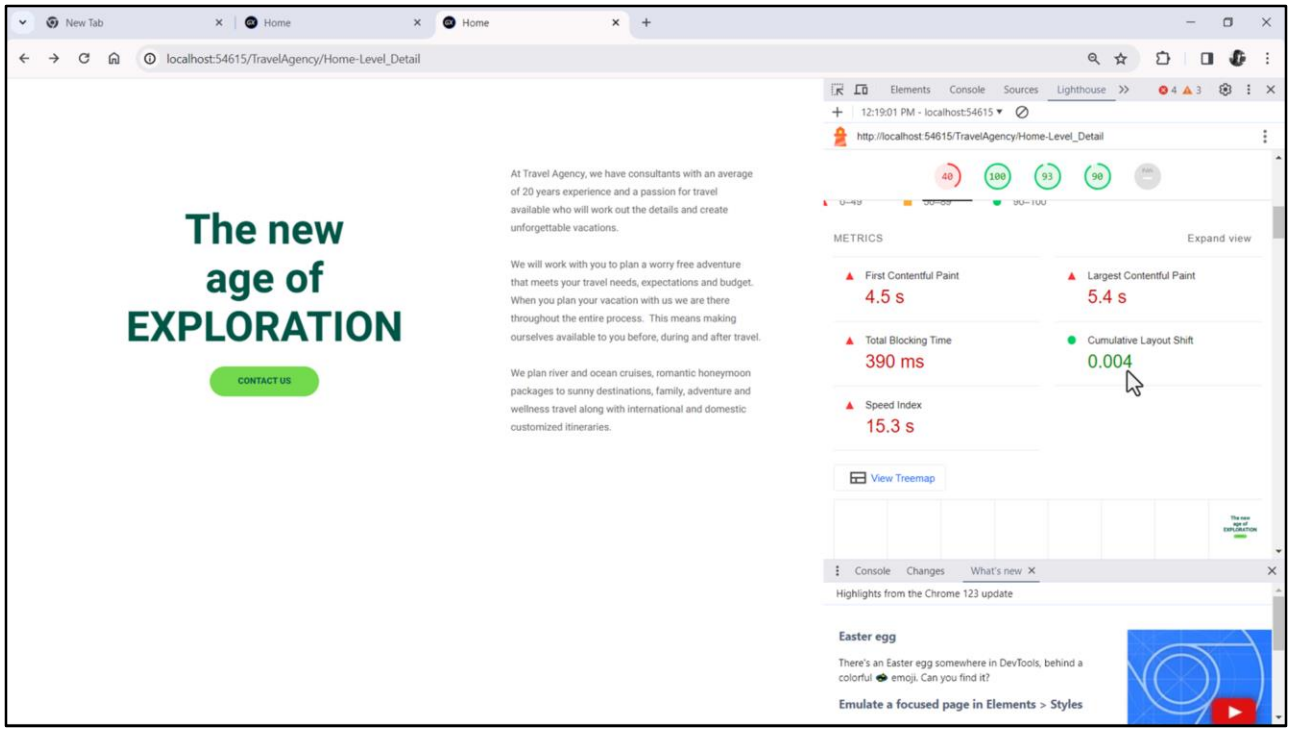
No es lo mismo que demore en aparecer el caption a que cambie. Lo segundo es lo que produce una impresión muy desfavorable. Yo coloqué ese procedimiento para que nos fuera evidente a la vista el cambio.  Lo voy a quitar… y vuelvo a ejecutar… ahora nos fue imperceptible en esta ejecución, pero no podemos asegurar que siempre lo sea. Tampoco podemos estar seguros de que en el futuro no coloquemos código en el evento ClientStart que demore su ejecución y que por tanto produzca este efecto indeseado.




Tan importante se ha vuelto medir este tipo de experiencias que aparece la métrica CLS (Cumulative Layout Shift) para dar una medida de qué tanto se producen estos cambios inesperados en una página. En nuestro wiki encuentran información de cómo reducir el CLS. Cuanto más próxima a 0 sea esta medida, mejor puntuada estará nuestra aplicación.




De hecho entre las DevTools (voy a pedir solo que muestre web, no dispositivos, y reducir el zoom, tenemos a disposición la herramienta de Google, Lighthouse, herramienta de auditoría y rendimiento que se utiliza para mejorar la calidad general de las páginas web mediante evaluación de aspectos como el rendimiento, la accesibilidad, el uso de las mejores prácticas de desarrollo web, la optimización para los motores de búsqueda, el PWA. Entre la evaluación el rendimiento estará la métrica CLS.

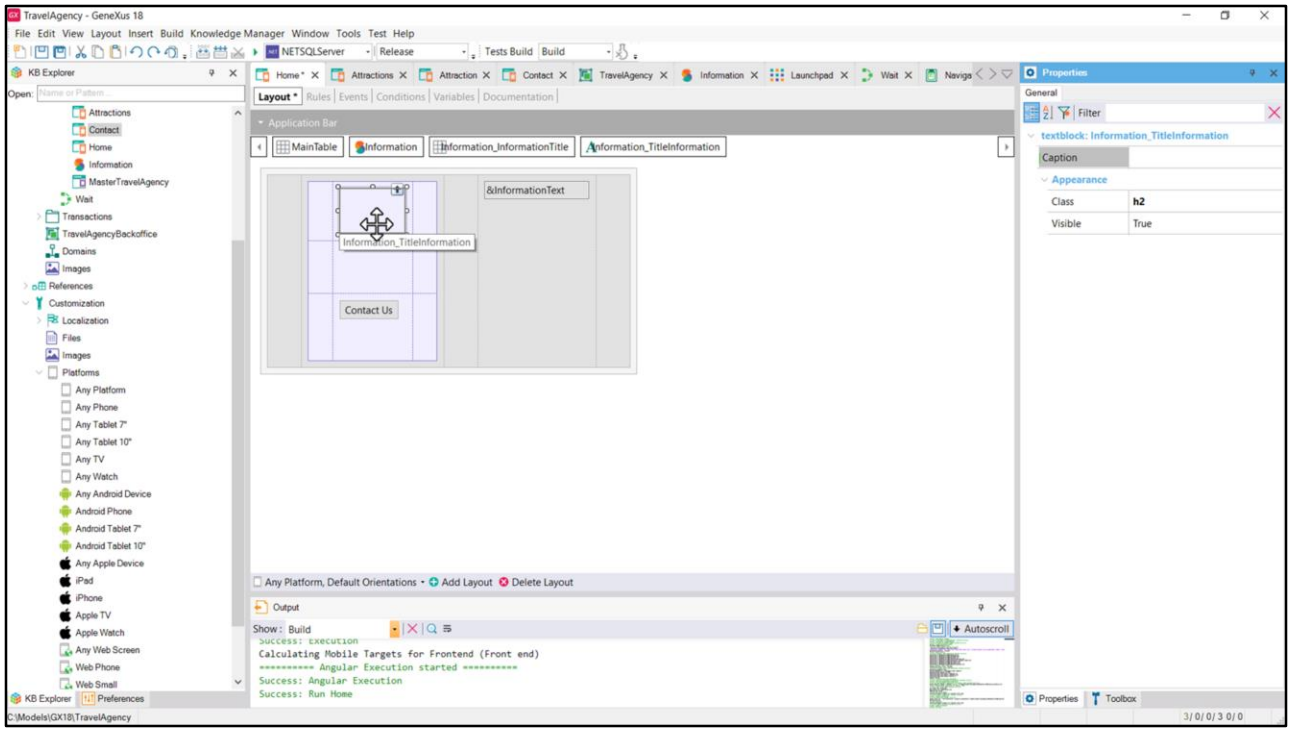



Vamos a pedirle, entonces, que analice la carga de nuestra página para tamaño Desktop… acá lo aceleré en el video, porque demoró 10 veces más. Vemos que nos sugiere ejecutar esto mismo en una ventana de incógnito.

Ustedes después pruébenlo.

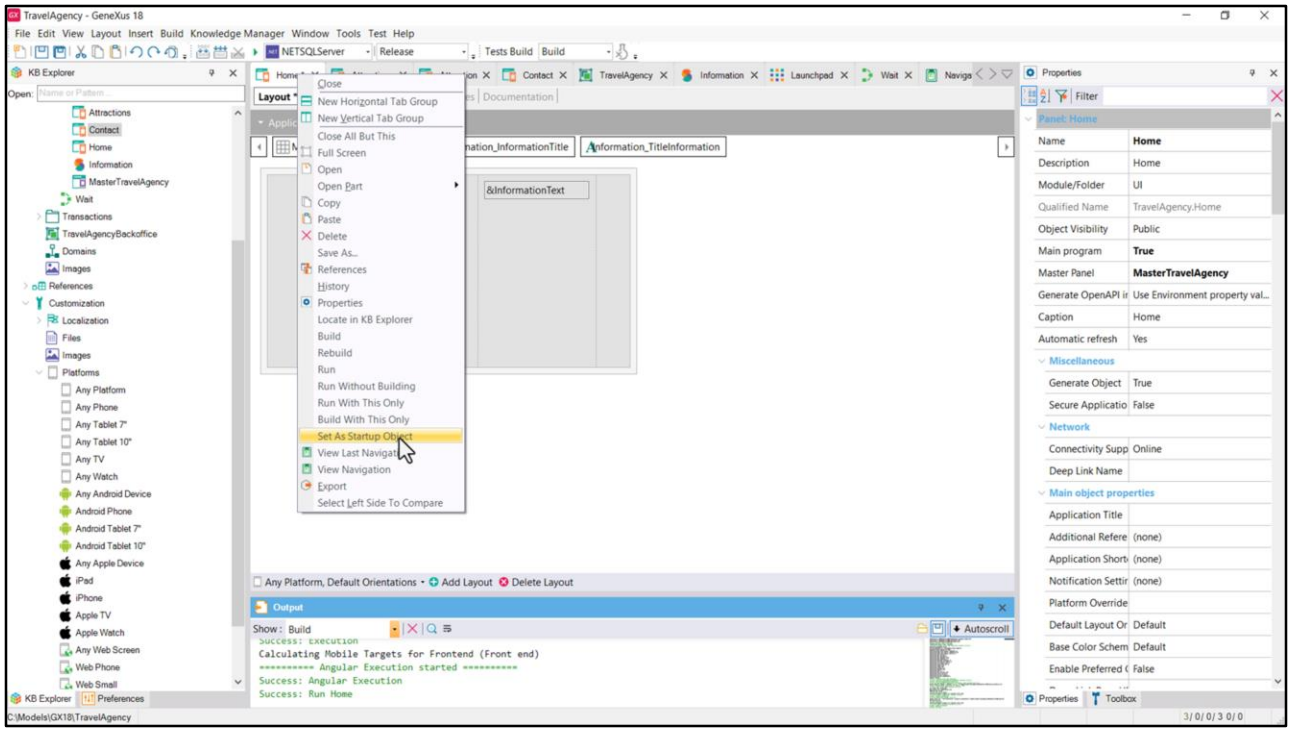
Ahora me interesa mostrarles que nos arroja medidas en las 5 dimensiones, de las cuales la de Performance nos es super relevante. Si vamos a mirar las métricas disgregadas, vemos la CLS, que no nos está dando 0, y eso seguramente tenga alguna relación con este cambio de layout indeseable del que veníamos hablando.

Puede parecer que es la menos importante de atender, dado que nos aparecen en rojo todas estas métricas relativas a la performance. Sin embargo es normal que ahora estas nos den así. Cuando vayamos a poner en producción deberemos cambiar una propiedad en las preferences de la KB, la Build Mode, pasándola a Distribution, y solo con eso ya mejorará esta métrica de performance en al menos 30 puntos. No lo hacemos en la etapa de desarrollo porque incrementa mucho los tiempos. En cambio, sí debemos atender la métrica CLS desde el vamos.




¿Qué tendríamos que hacer para reducir el CLS? Quitar la asignación del caption de aquí. Dejarlo vacío. ¿Cuál es el problema de dejarlo vacío? Este. Que nos es invisible la presencia del control en el layout, a menos que hagamos clic allí.

Conviene tener todas estas cuestiones presentes desde el principio, para evitarnos este tipo de dolores de cabeza más adelante y es por eso que se los quería contar en este video, antes de seguir.

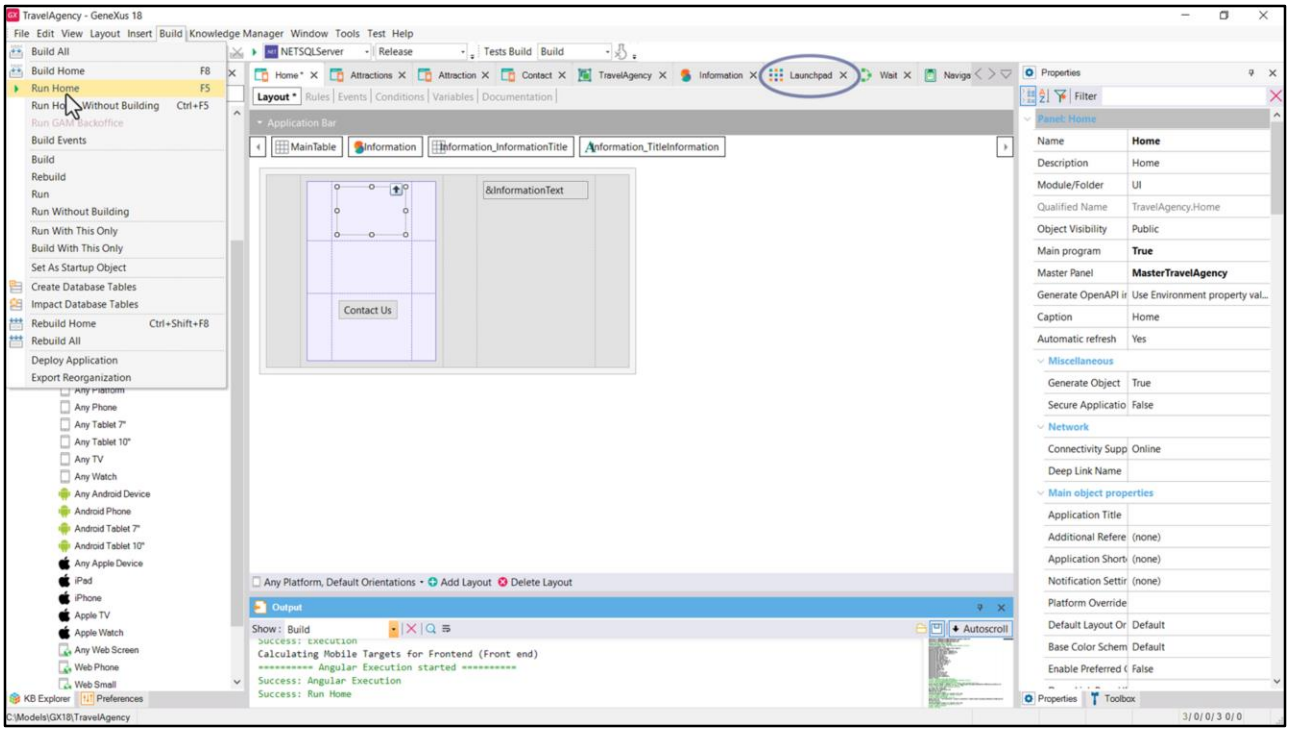



Por último, ya que estoy en plena sección recomendaciones: habíamos mencionado varios videos atrás que nos es muy importante optimizar los tiempos de desarrollo.

Por ello estamos prototipando local. Voy a cerrar, antes de seguir, las dos ventanas de los servidores: del servidor del frontend y del servidor del backend.

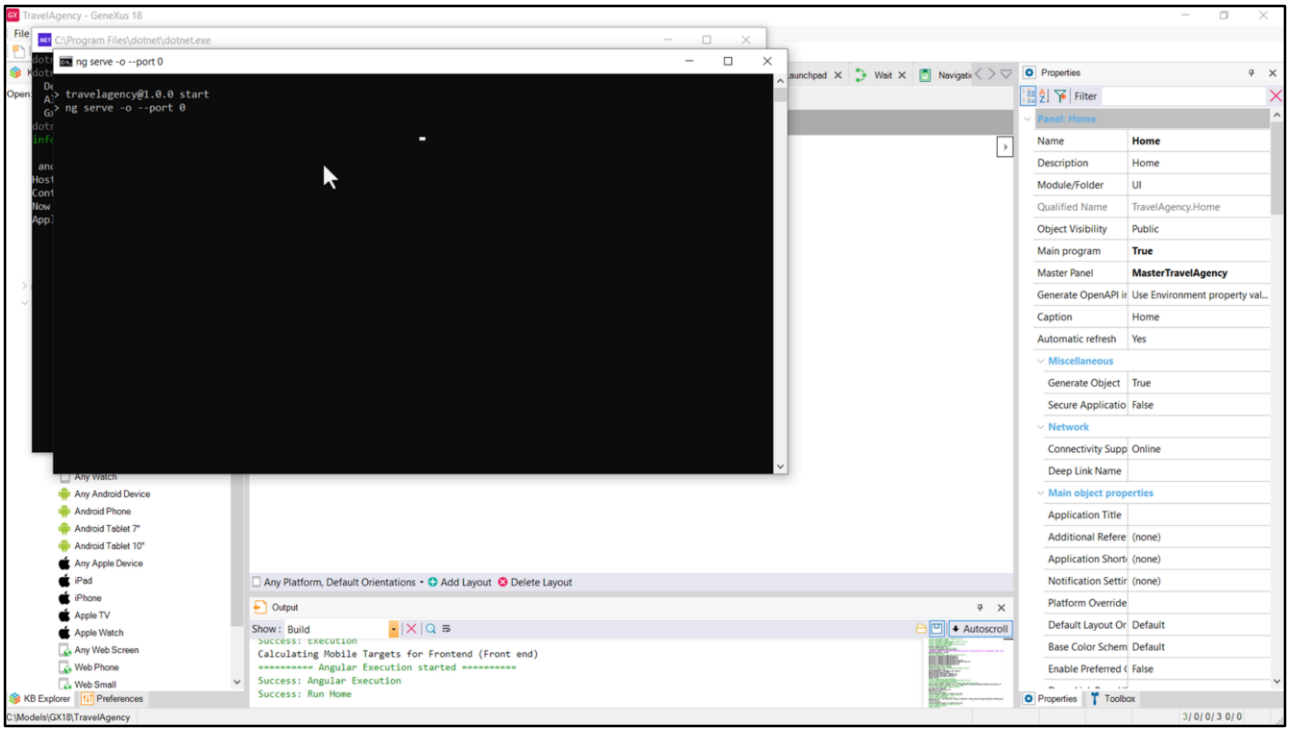
El punto de entrada de nuestra aplicación final va a ser el objeto Home, y por eso lo habíamos configurado como main program y por eso podemos correrlo con la opción Run.

De hecho hubiera sido razonable configurarlo como el startup object, que es lo que haré, de modo de no necesitar posicionarnos sobre él y seleccionar Run, sino simplemente presionar F5 para correrlo.

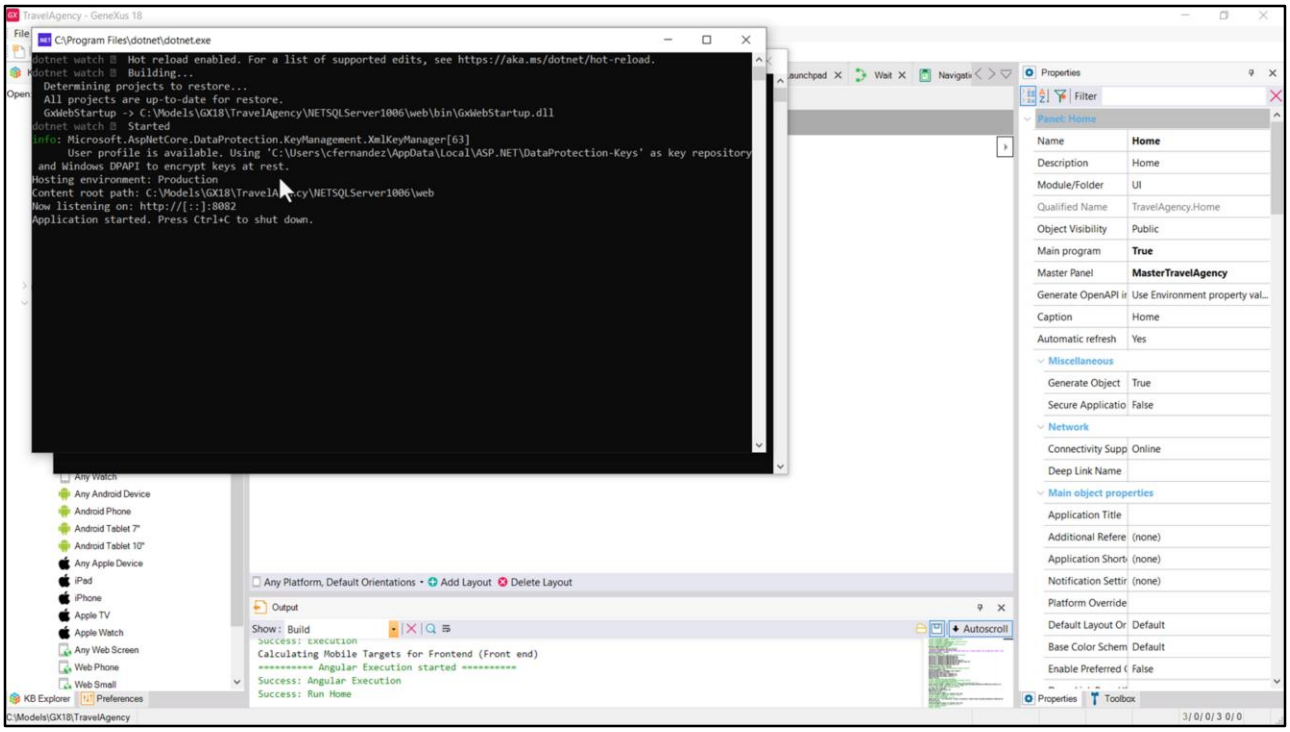



Observemos que ahora para el F5 en lugar de ofrecer el Run del Developer Menu que abre el launchpad, aparece el Run del Home. Ejecutémoslo.

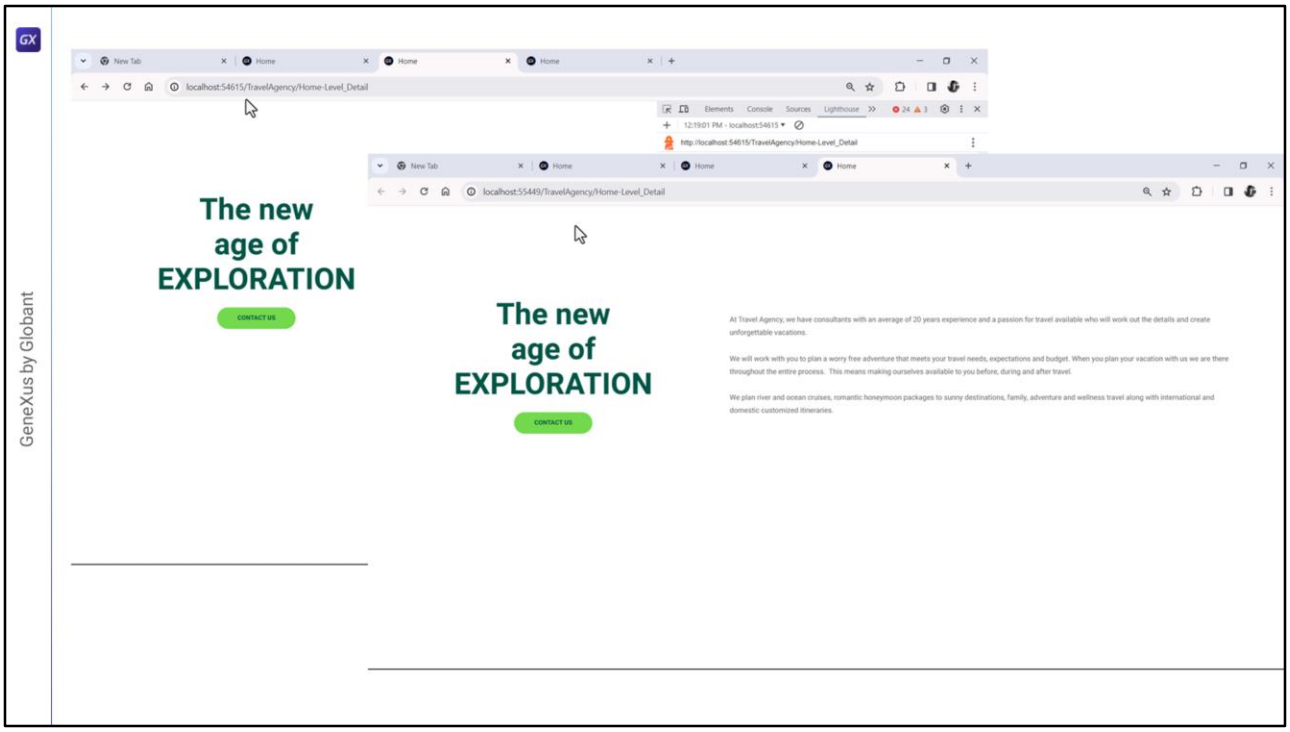
Va a especificar, generar, compilar todo el árbol de dependencias que parte del objeto Home… y luego levantar el servidor del backend, por los servicios, y el del frontend para la aplicación Angular.



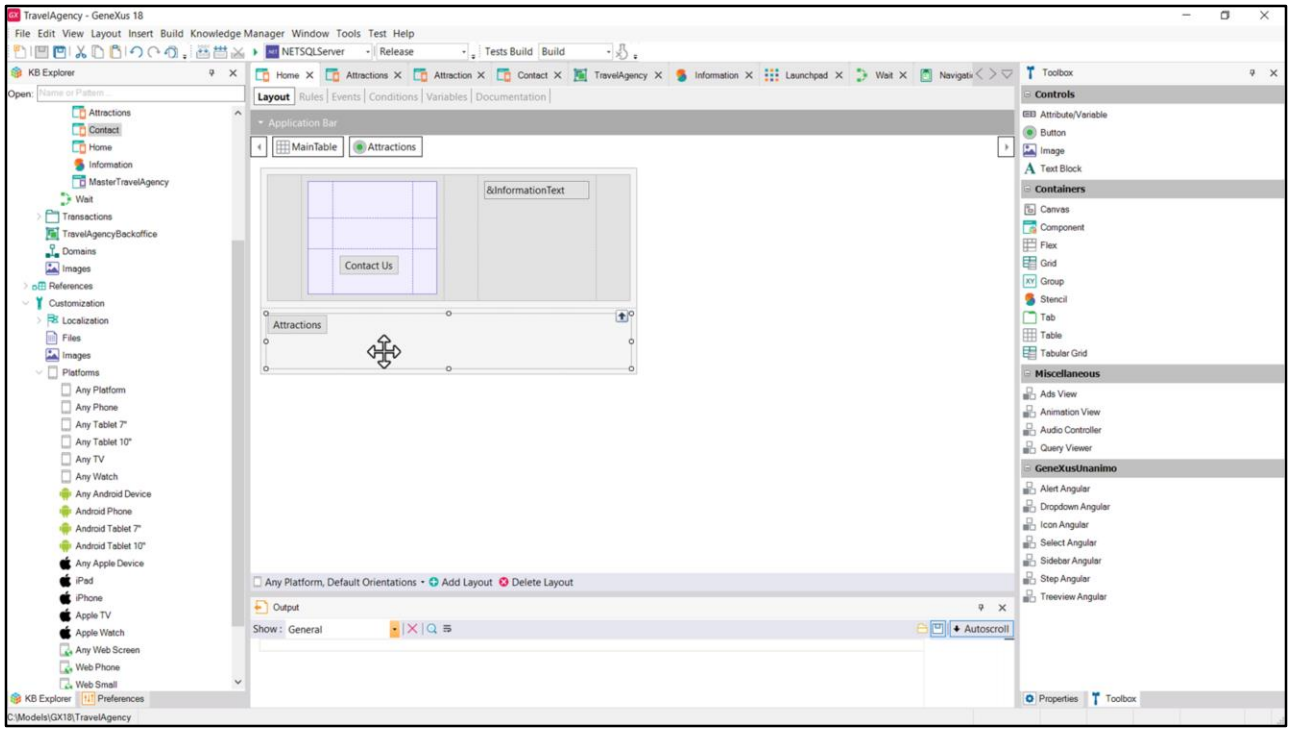

Este es el de la aplicación Angular. Cada vez que se cierra el servidor y se vuelve a levantar consume tiempo en esa inicialización, y además se le asigna a la aplicación un puerto diferente… en un instante nos quedará claro esto.




Y este de aquí es el servidor del backend, que es donde están siempre los servicios, por ejemplo, nuestro procedimiento Wait, que ahora ya no es invocado pero cuando lo era, era servido de aquí.



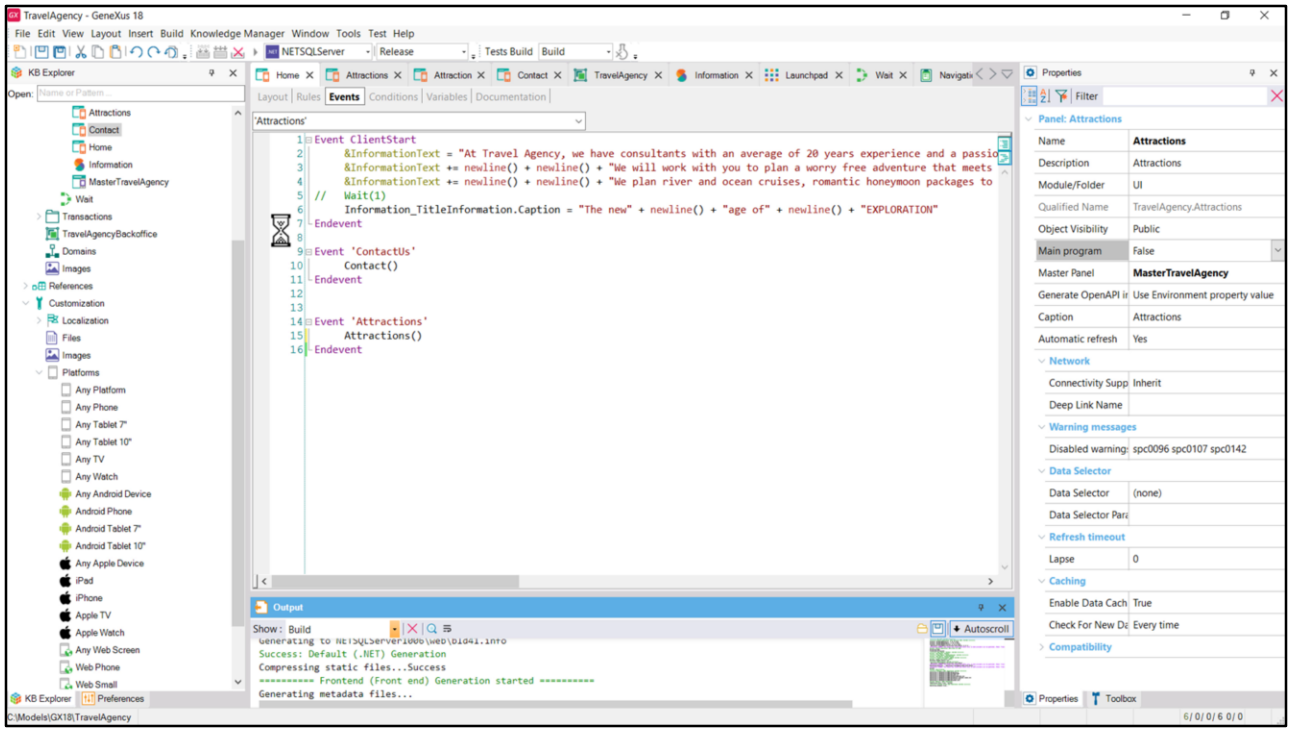

Y acá vemos esto que les decía, se abrió en otra pestaña porque el puerto es diferente respecto a la ejecución anterior y fue por eso que cerré antes las ventanas, para que pudieran apreciar bien esto.
Si estuviéramos prototipando en la nube nada de esto nos sería evidente, pero como contra, los tiempos serían más largos, dado que hay que transferir todo el paquete cada vez al servidor en la nube.



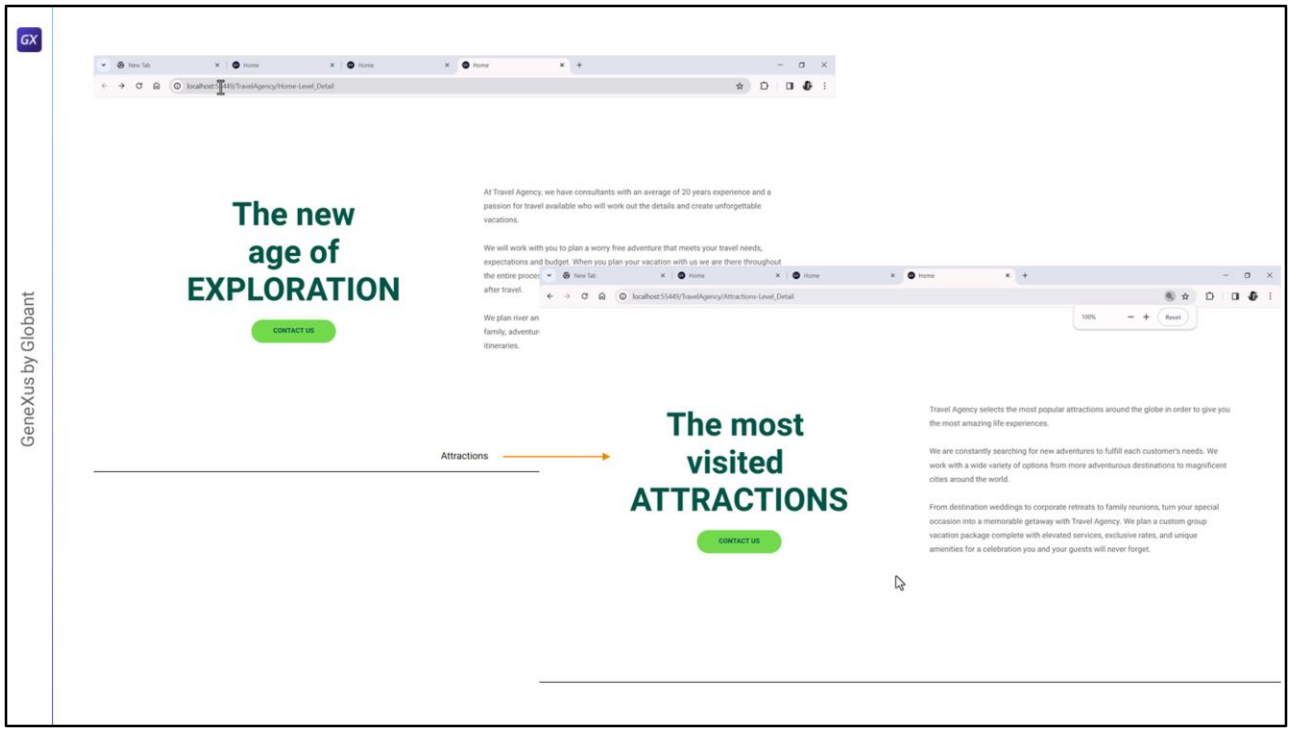

¿Bueno, y cuál es nuestro problema ahora? Que si queremos ejecutar el panel Attractions, como no lo tenemos accesible desde el panel Home la solución provisoria que habíamos encontrado antes era declararlo también como main, es decir, estamos generando una segunda aplicación. A partir de allí nos aparece también la posibilidad de Run del objeto, que compilará todo su árbol de dependencias y tendrá que abrir otro servidor en otro puerto, por lo que, antes de presionar el Run tendemos que cerrar los dos servidores, que tendrán el costo de levantarse nuevamente y de instalarse el paquete, ahora de Attractions, de cero. Todo eso es igual a tiempo perdido.

En cambio, ahora tenemos abiertos los servidores para la aplicación Home, por lo que si hacemos un cambio en cualquier objeto de su árbol, por ejemplo él mismo… agreguemos un botón con evento asociado Attractions que por ahora dejaré vacío. Luego llamaré desde aquí al panel Attractions, así nos queda enganchado el panel dentro de la misma aplicación, aunque sea de manera provisoria, para la prototipación, para optimizar nuestros tiempos de desarrollo. Cuando implementemos el menú ya podremos quitar este botón. Así que nos conviene para ser más eficientes en los tiempos de desarrollo.

Ahora ejecutemos el Home. Vemos que se abrió en la misma pestaña de antes, en el mismo puerto y con el botón.

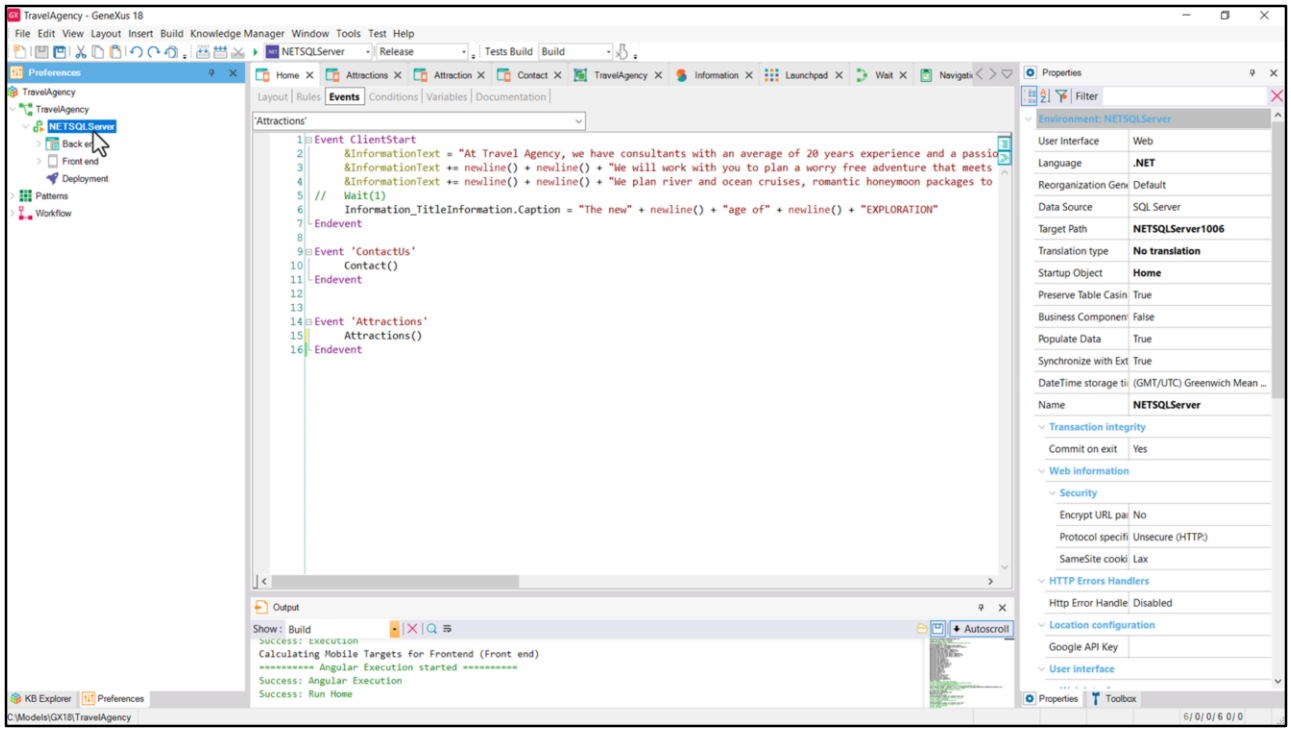



Entonces la recomendación es que incluyamos provisoriamente el botón con la invocación a Attractions y le reasignemos el valor default a la propiedad Main Program, que es False. Ejecutemos.




Bien, presionemos el botón. Y acá tenemos la panel Attractions. Vemos así de chicas las cosas porque habíamos achicado el zoom. Volvámoslo a 100%. Bien.

Y podemos apreciar que seguimos en el mismo puerto, con la misma aplicación.




Por último último, recuerden que el startup object se encuentra configurado a nivel del Environment activo, lo ven acá, por si quieren volver a la situación anterior que era “ningún startup object” y al hacerlo, en el Build vean que vuelve a estar asociado el F5 al Run del Developer Menu.

Bueno, y con esto termino lo que quería que viéramos juntos en este video. Los espero en el siguiente.


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