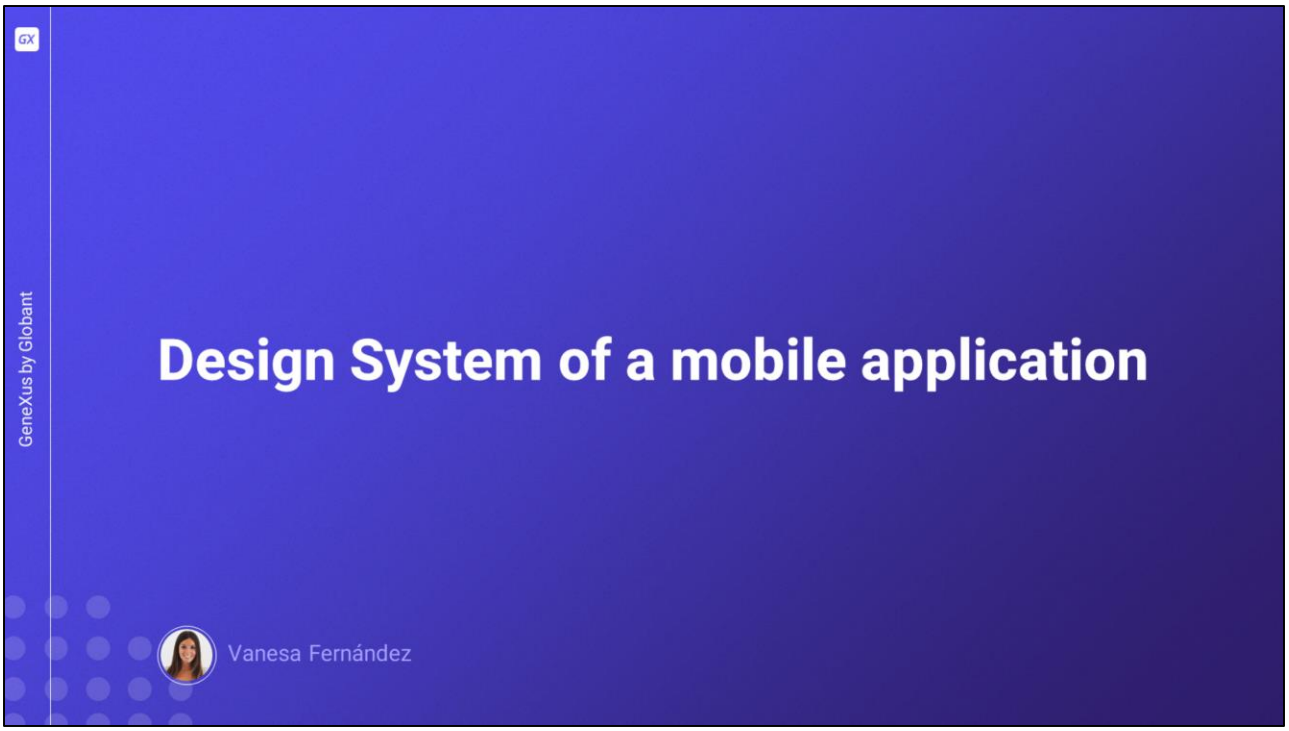
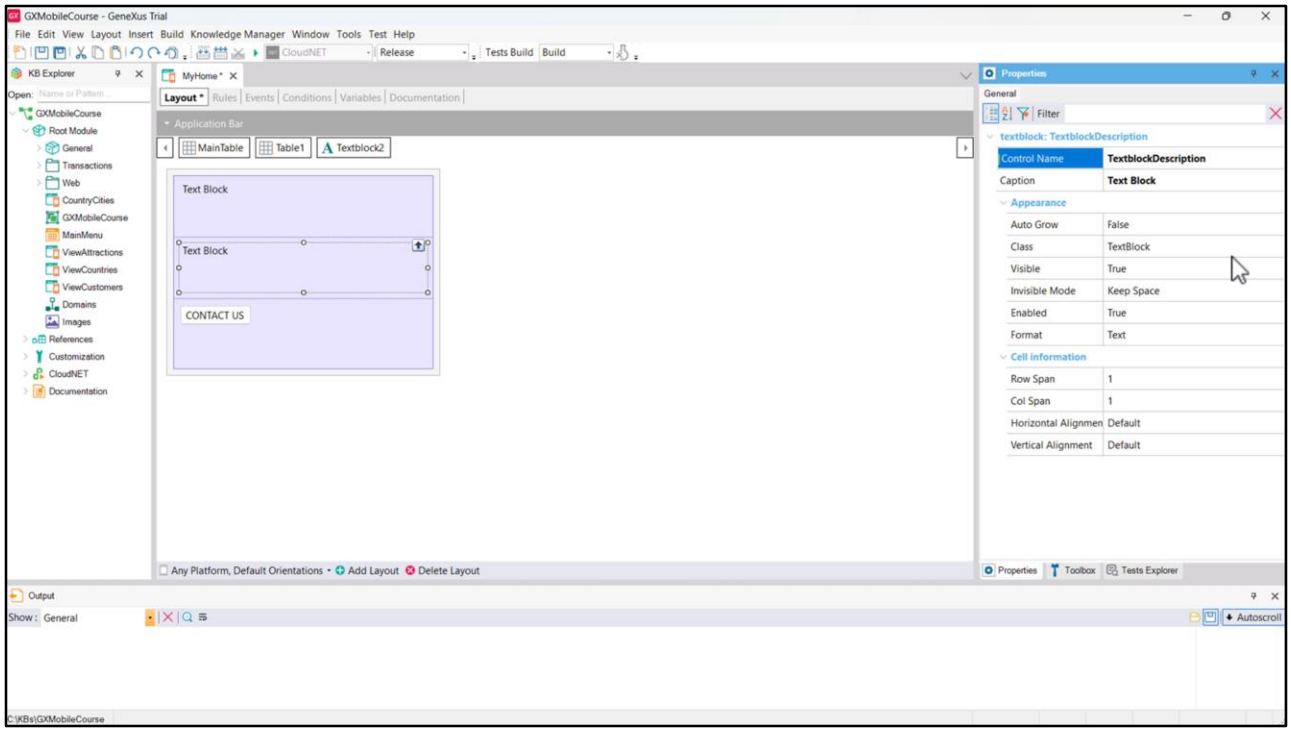
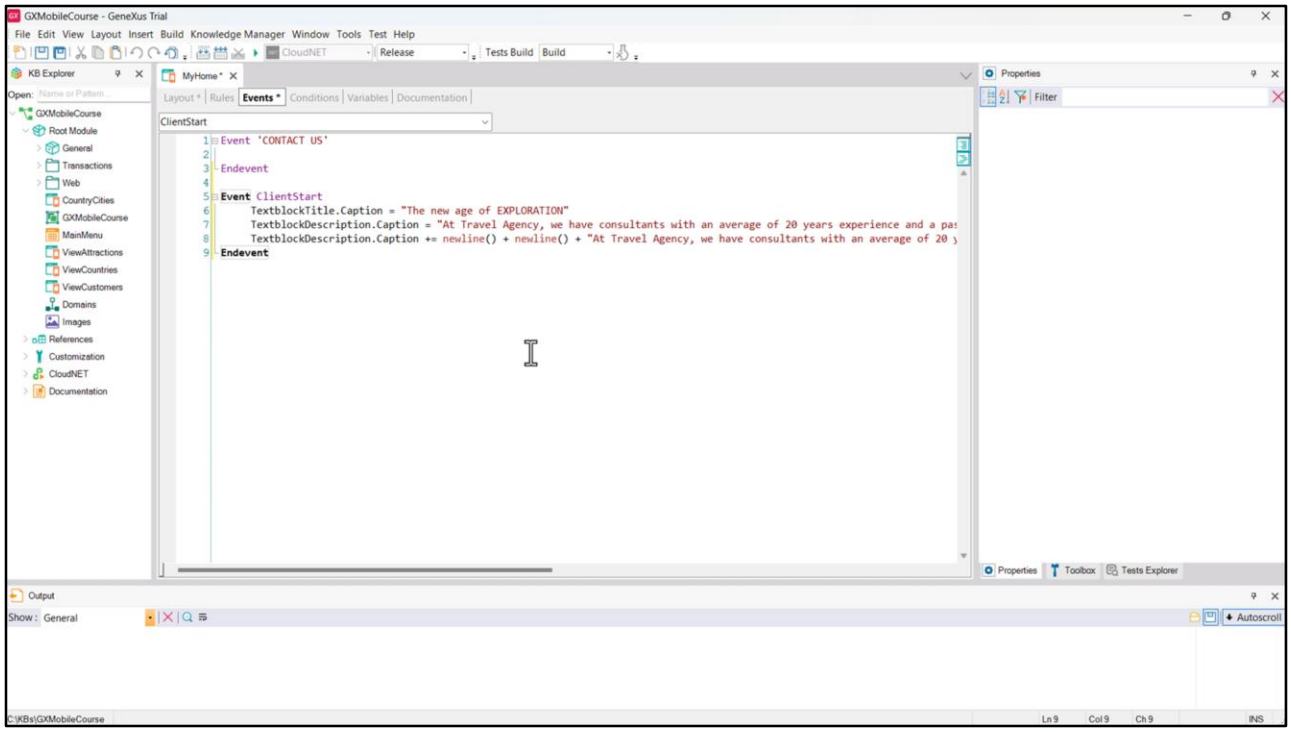


Neste vídeo estudaremos como trabalhar com o objeto Design System para dar o design à nossa aplicação. Veremos, entre outros, como utilizar tokens, estilos, propriedades, incorporar fontes e trabalhar com o modo claro e escuro.

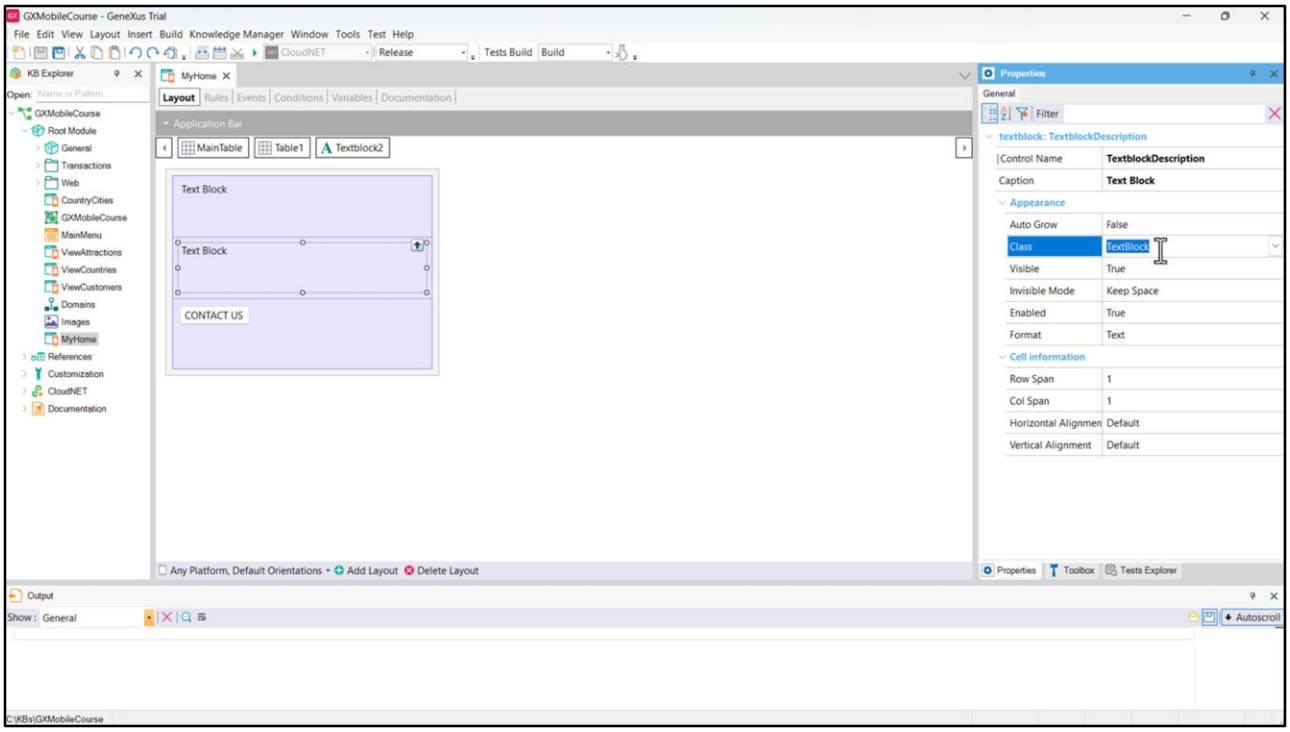


Com o único objetivo de fazer alguns testes, vamos criar um Panel chamado MyHome, para o qual iremos configurar a propriedade Main program como True para que possamos executá-lo facilmente, pois não terá dependências.

Vamos adicionar uma tabela, e dentro dela 3 elementos: 2 textblocks, e um botão, ao qual colocaremos “CONTACT US”. Vamos mudar o nome do controle do primeiro textblock para TextblockTitle... e o do segundo para TextblockDescription.

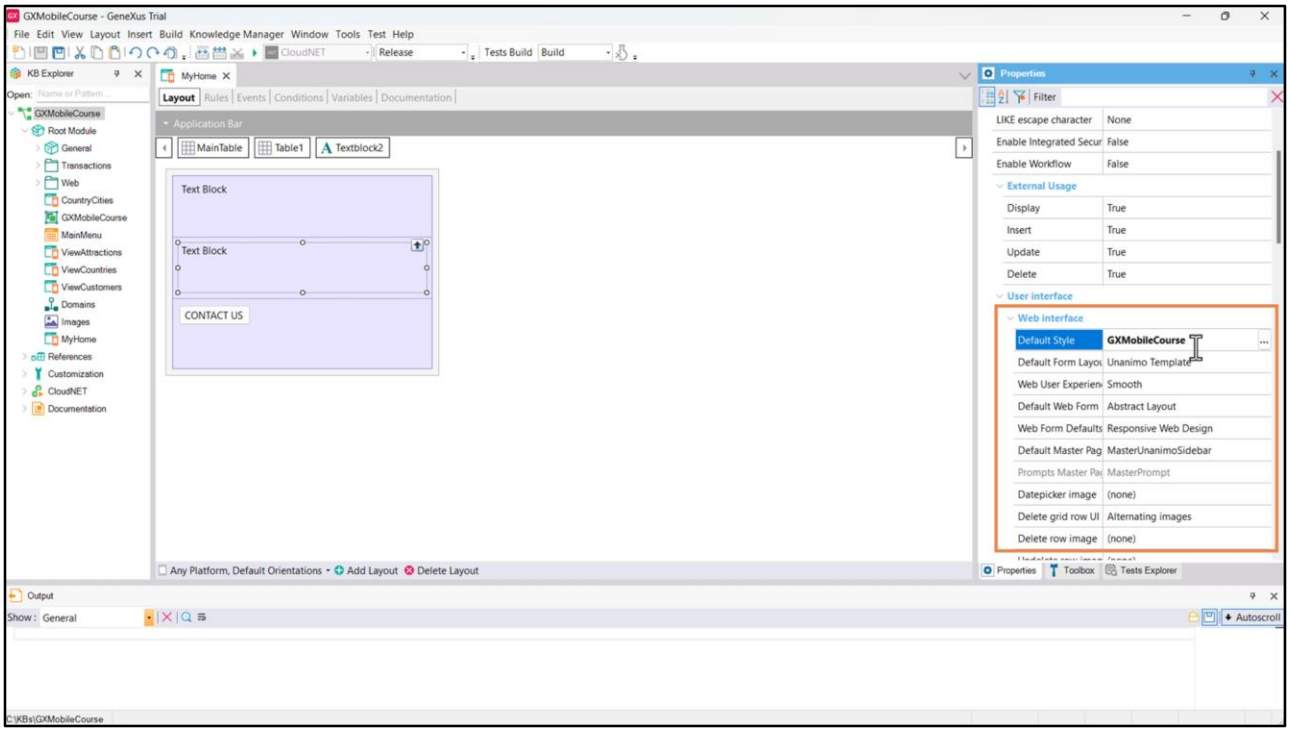


Vamos aos eventos, e adicionamos o ClientStart, que será executado quando for aberto o Panel. Vamos carregar os parágrafos separadamente para podermos adicionar a nova linha, que nos permite estabelecer a separação entre eles.

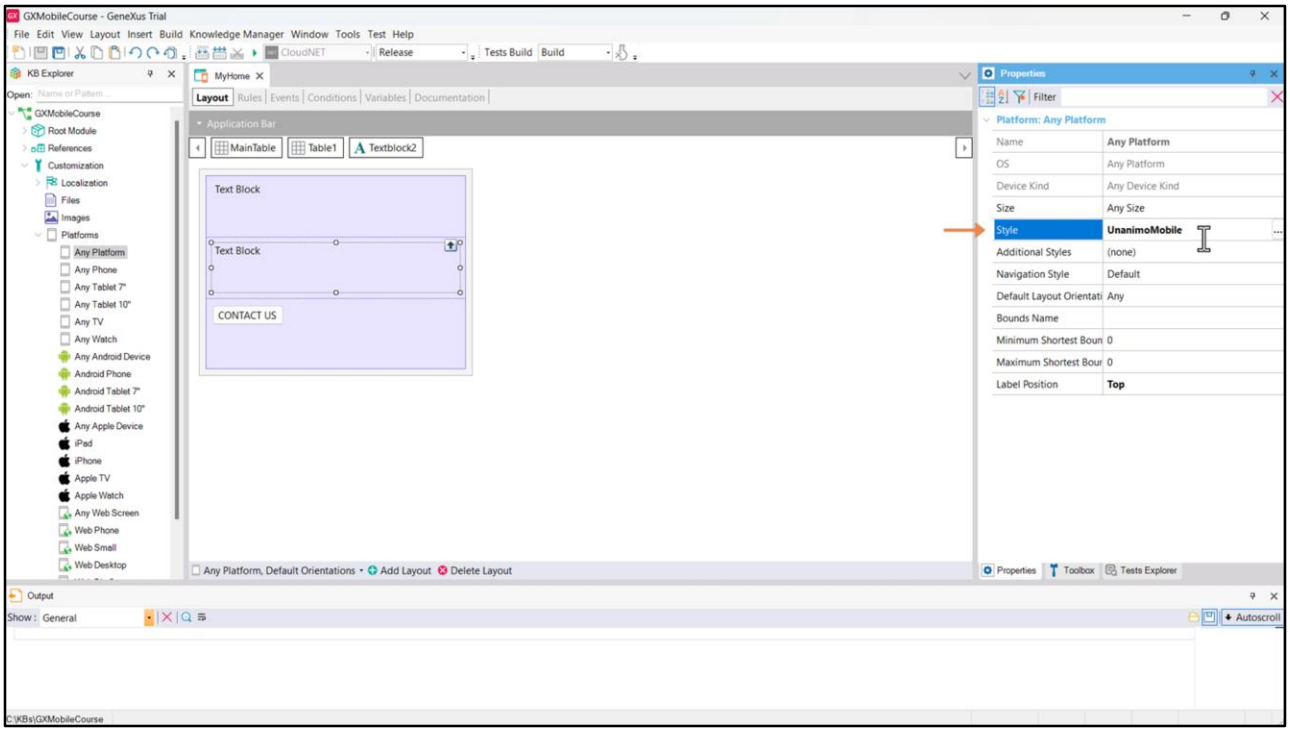


Se editarmos um controle TextBlock daqueles que adicionamos, vemos que ele possui a propriedade Class, que por padrão tem o valor TextBlock.

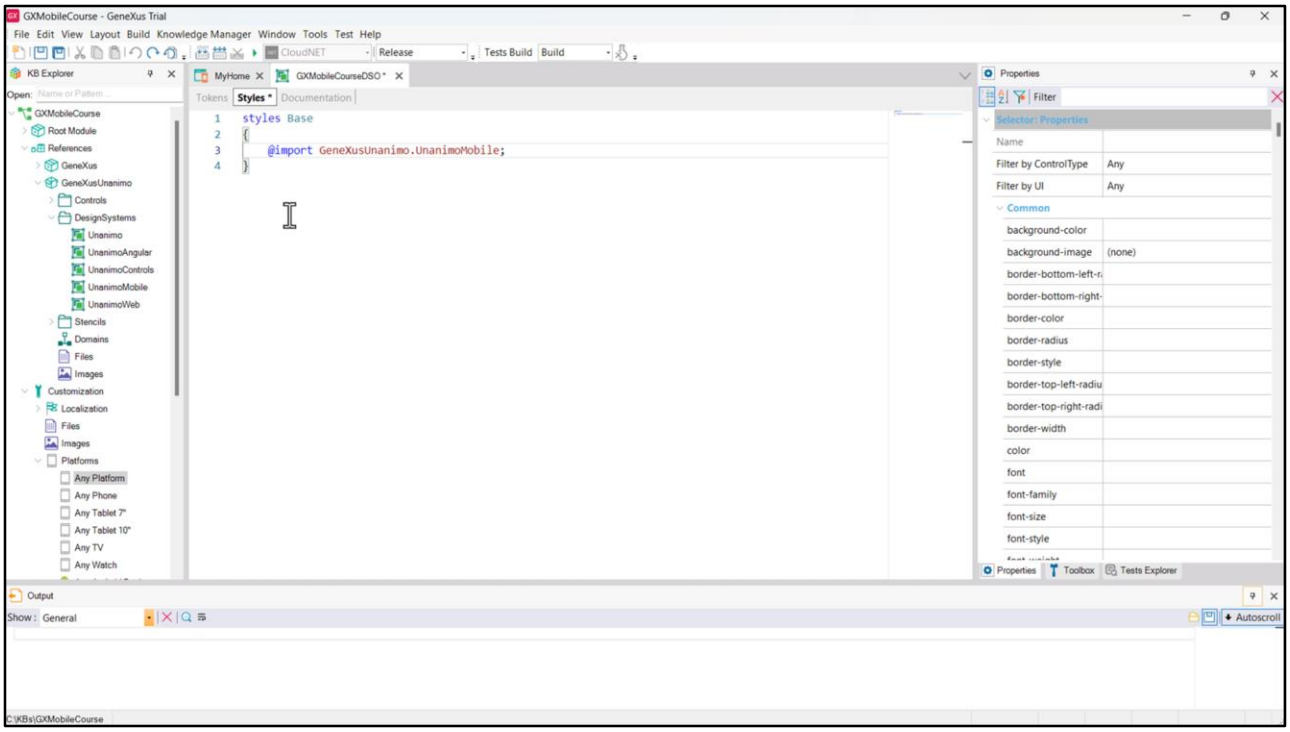
Essas definições das classes, que são as que conterão as características de cada elemento do layout, estarão definidas e centralizadas em um objeto: o Design System.



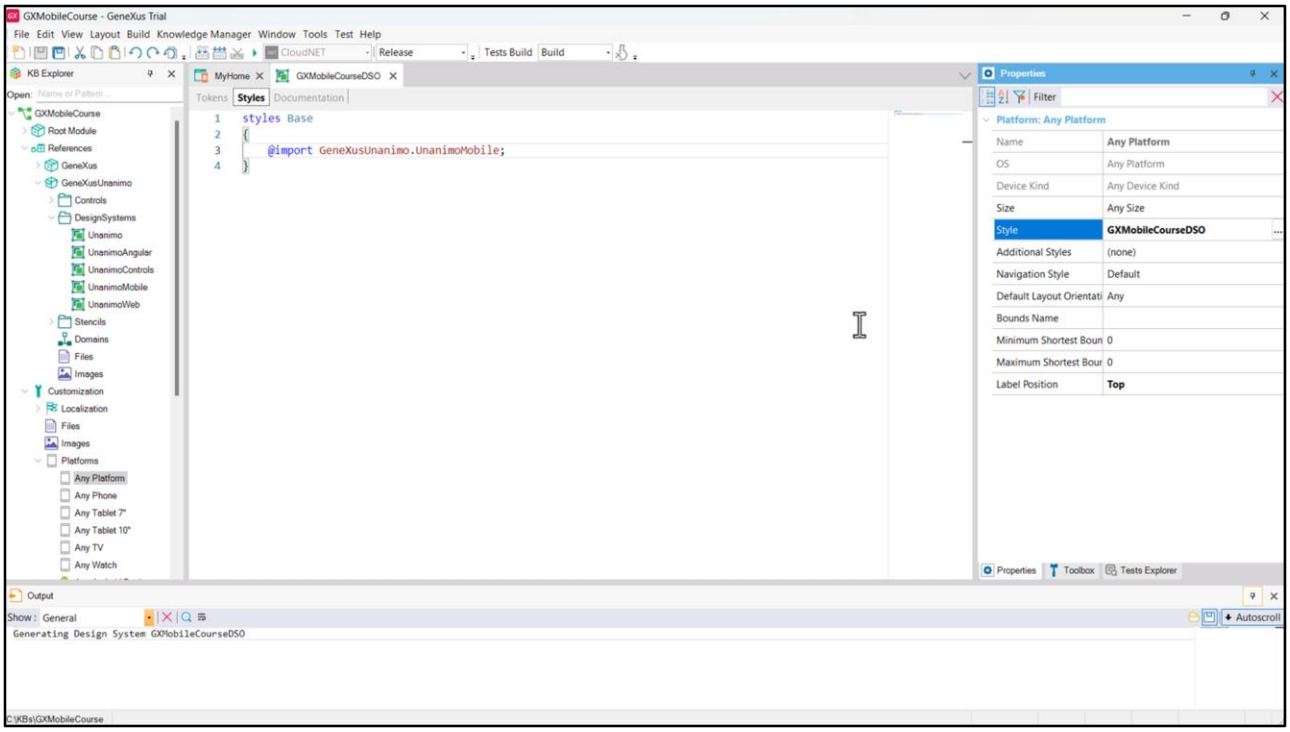
Toda KB é criada por padrão com um design system predefinido que vemos se editarmos as propriedades da versão, na chamada Default Style. O estilo corresponde ao objeto Design System que contém todas essas definições. Por padrão, é criado na KB este Design System object, que assume o mesmo nome da KB. Este DSO está configurado no grupo de propriedades Web interface: isto significa que o que comanda é o desenvolvimento da aplicação quando é com Web Panels. Como estamos trabalhando em uma aplicação móvel nativa, não é o nosso caso, pois estamos utilizando Panels em vez de Web Panels.



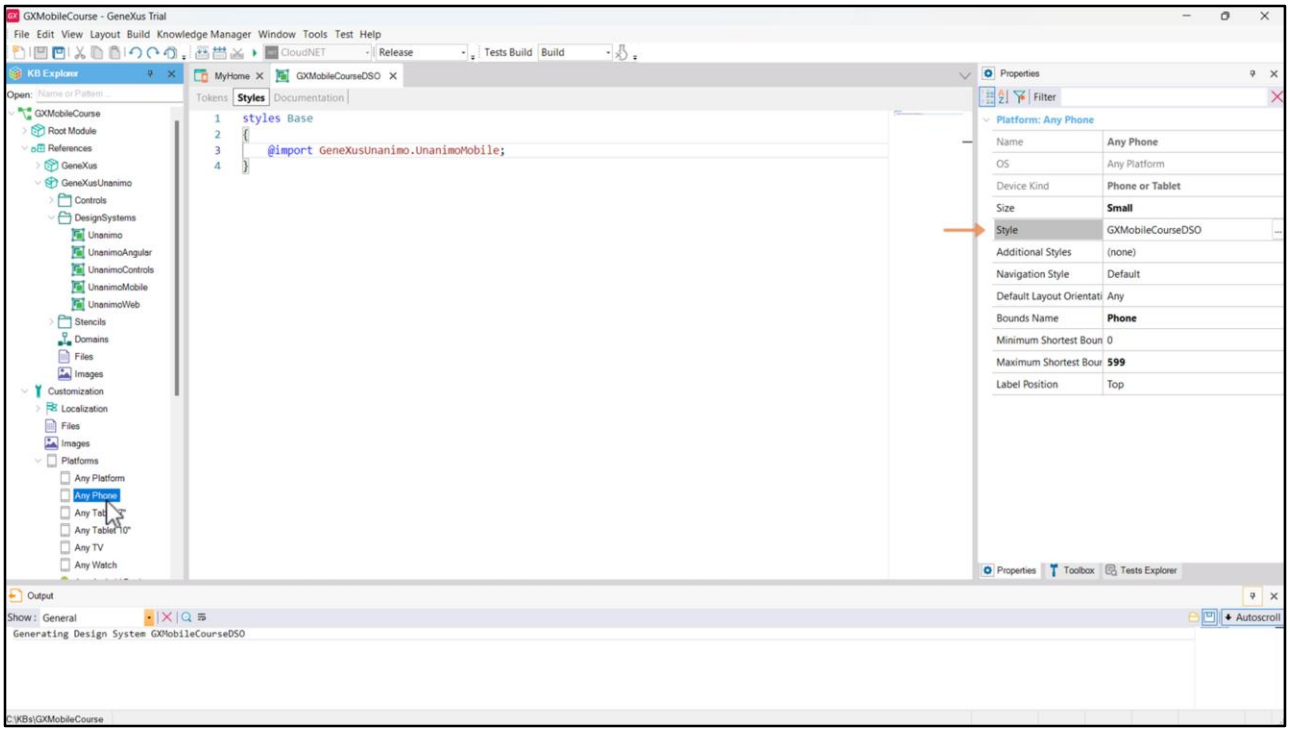
Para comandar o design quando estamos em um ambiente nativo, devemos alterar onde estão as definições das plataformas, sob o nó Customization. No nó Any Platform, vemos que na propriedade Style está pegando o objeto Design System denominado UnanimoMobile. Unanimo já é um Design System, aquele predefinido por GeneXus, criado no nó GeneXusUnanimo e é aquele que por padrão assumirá o controle do design de toda a aplicação se não definirmos outra coisa.



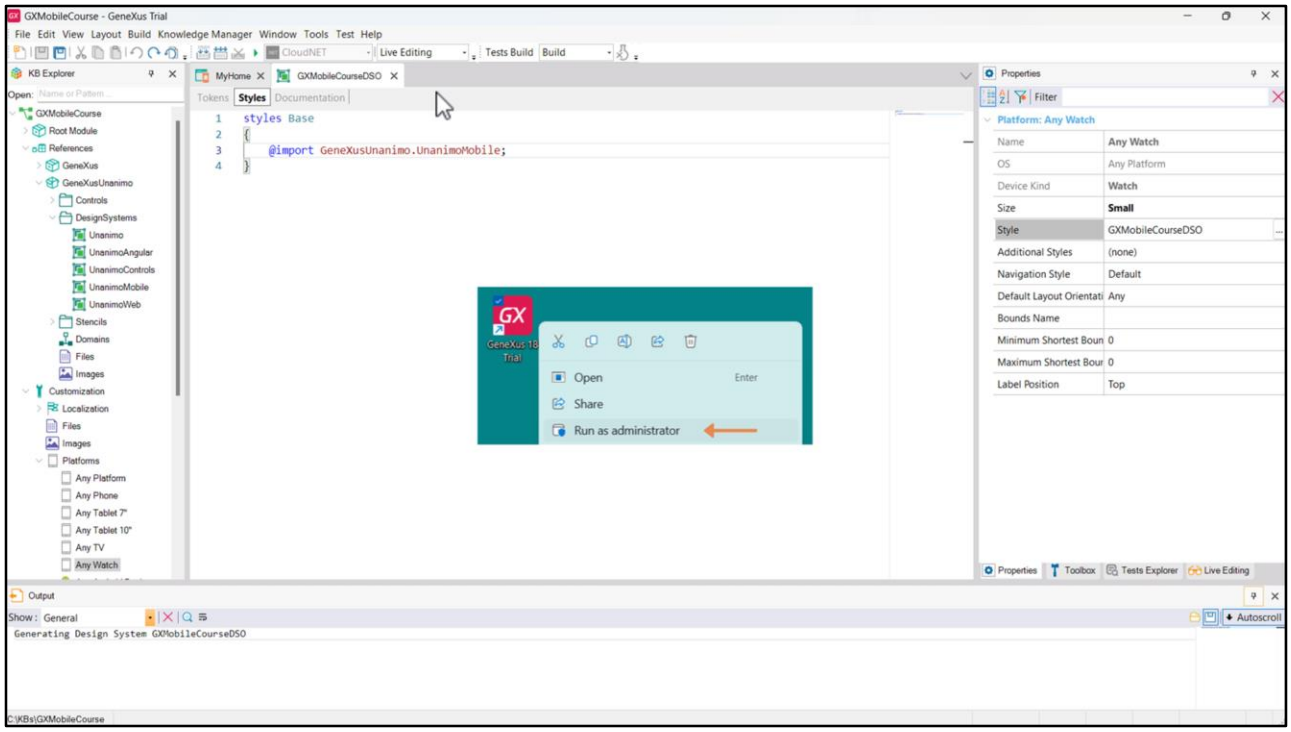
Como vamos fazer um desenvolvimento do zero, criaremos um novo objeto do tipo Design System que será aquele que aplicará os estilos à nossa aplicação nativa. Vamos chamá-lo de GXMobileCourseDSO, e indicaremos aqui que este Design System deve herdar das definições que tenha o Design System UnanimoMobile. Para fazer isso, escrevemos na aba Styles o seguinte:



Utilizaremos as abas Tokens e Styles para fazer nossas definições, mas antes vamos indicar ao GeneXus que queremos que nossa KB tome como Design System aquele que acabamos de criar. Vamos para Any Platform e alteramos a propriedade Style.



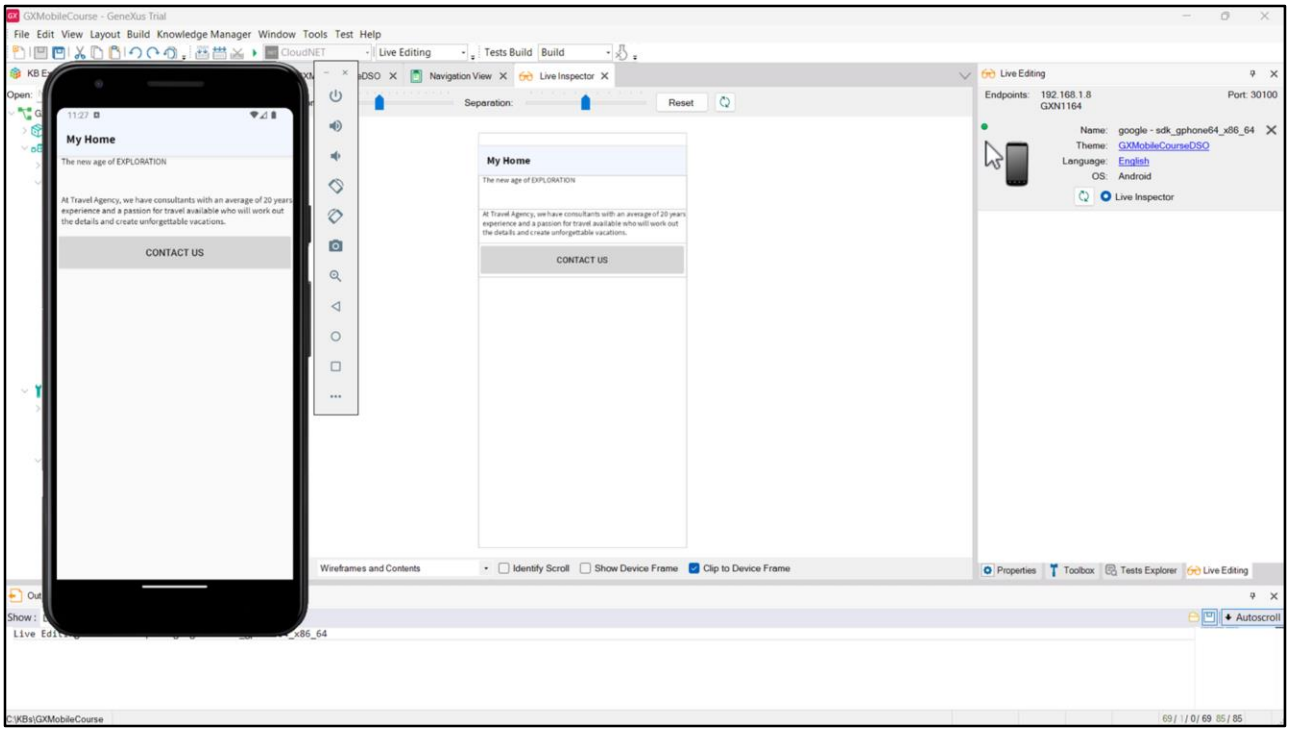
Se observarmos os diferentes tamanhos e plataformas, vemos que herdam daquele que acabamos de configurar, mas se quisermos podemos mudar isso, utilizando um Design System object para cada tamanho de tela e plataforma.



Antes de executar nosso Panel para ver como ele fica antes de começarmos a mudar seu estilo, vamos configurar o uso da ferramenta Live Editing. Mas… o que é Live Editing? Ao prototipar uma aplicação, uma das tarefas que consome mais tempo é aperfeiçoar o Look&Feel (ou User Interface -UI-) e a User Experience -UX-. Para simplificar esta tarefa, GeneXus conta com uma funcionalidade que permite alterar o estado da aplicação em tempo real a partir do IDE sem salvar os objetos modificados.

Para prototipar em modo Live Editing, simplesmente alteramos neste ComboBox o valor Release para Live Editing e executamos novamente a aplicação. Se você estiver utilizando a versão Trial de GeneXus, deverá executar com permissões de Administrador.

Neste caso, por ser a primeira vez que o modo Live Editing é ativado, será necessário compilar novamente a aplicação, mas para as próximas alterações que fizermos, isso não será mais necessário, e simplesmente serão refletidas as alterações automaticamente no emulador e no IDE de GeneXus.

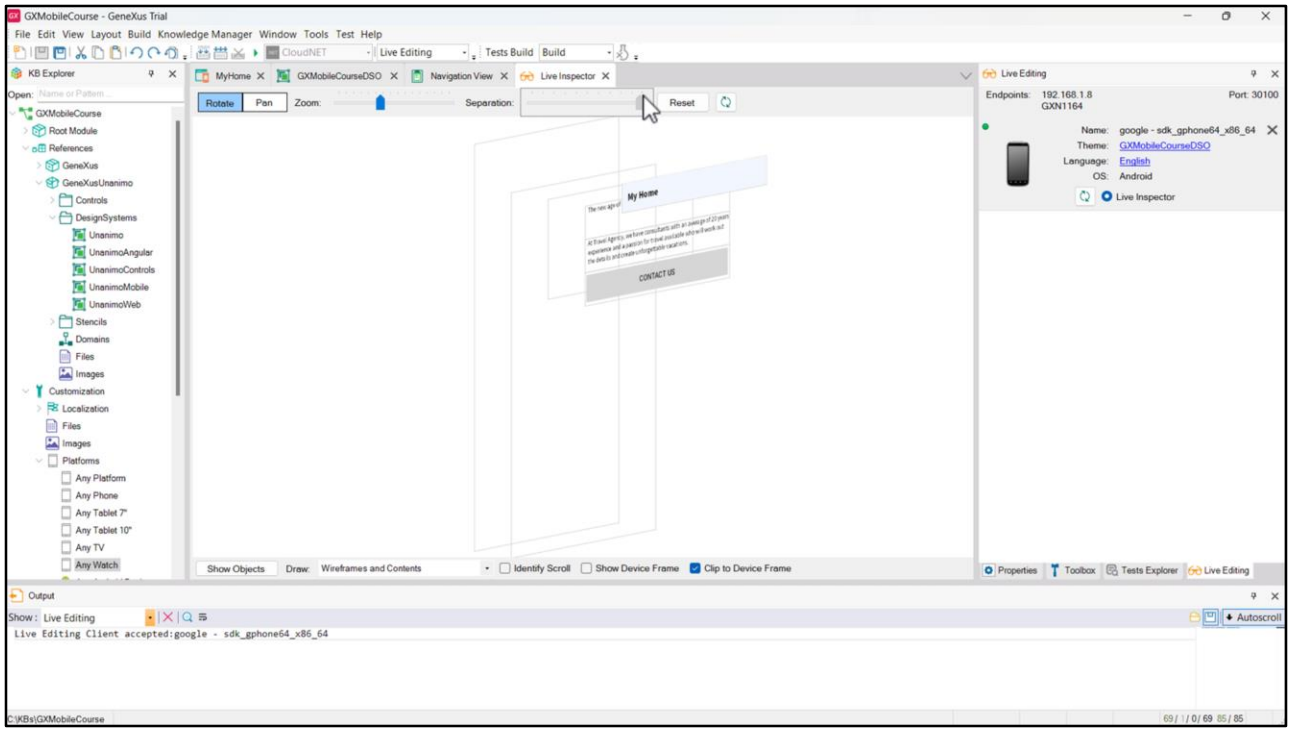


Vamos executar.

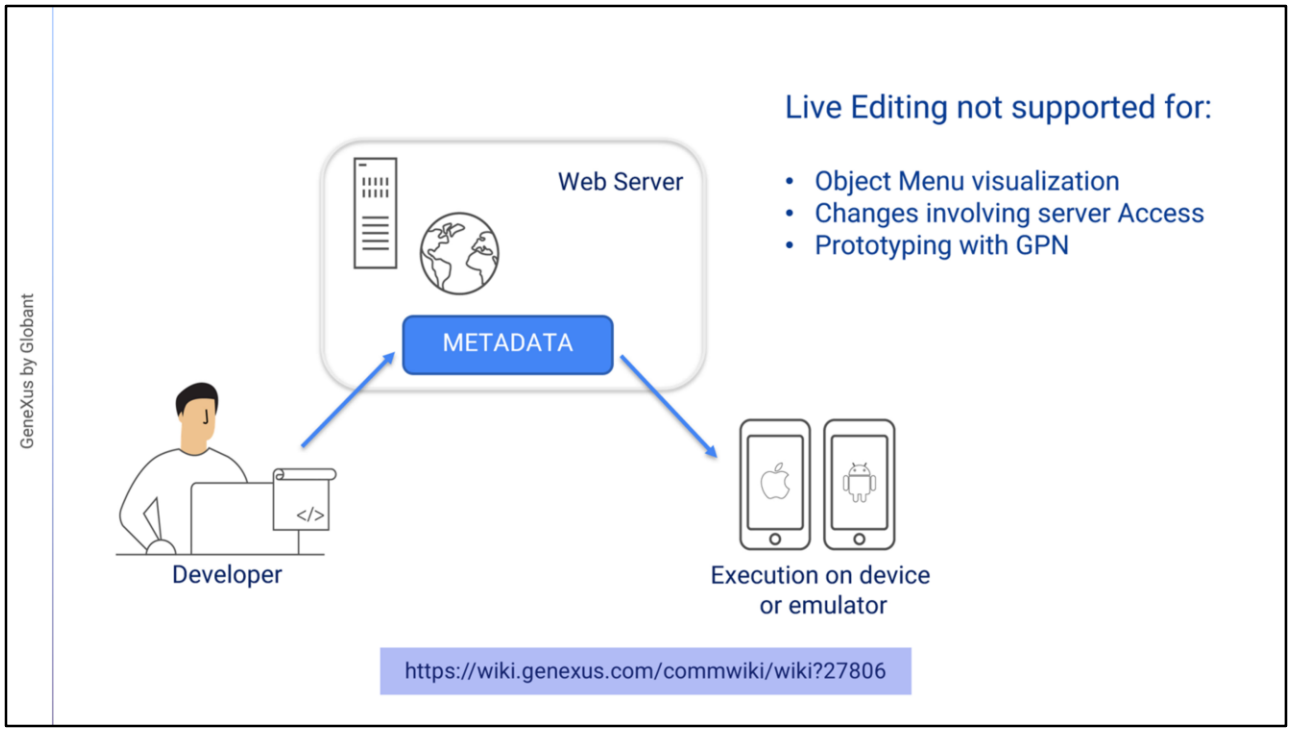
Já temos a aplicação pronta novamente no emulador, só que agora habilitamos Live Editing.

No IDE podemos ver que foi habilitada uma aba Live Editing, onde temos a informação dos dispositivos que estão conectados, no nosso caso o emulador que tínhamos em execução. Podemos ver dados do dispositivo, como o nome, o Design System que está sendo utilizado na aplicação e o idioma. O ponto verde indica que o dispositivo está conectado ao Live Editing.

Também podemos ver que foi aberta uma nova janela, chamada Live Inspector. Esta é uma das principais vantagens do Live Editing. Embora haja uma restrição e não possamos ver quando está sendo executado um objeto menu, podemos ver o restante dos painéis. Basta passar o mouse sobre os controles em tela para ser possível ver seu nome e se clicarmos podemos ver seus dados principais.



Se movermos o mouse podemos girar a imagem em qualquer direção, vemos algumas camadas que nos permitem ver dentro de qual controle está localizado cada controle. Estas camadas permitem selecioná-los de uma forma muito mais fácil, pois em geral as seções ou tabelas estão sobrepostas. Com o slider de Separação podemos aumentar ou diminuir o espaço entre as camadas, com Zoom podemos aumentar ou diminuir o Zoom sobre o que estamos vendo, e se selecionarmos Pan em vez de Rotate, podemos mover o conteúdo com o mouse, sem Girá-lo. Com Reset voltamos à visualização padrão.

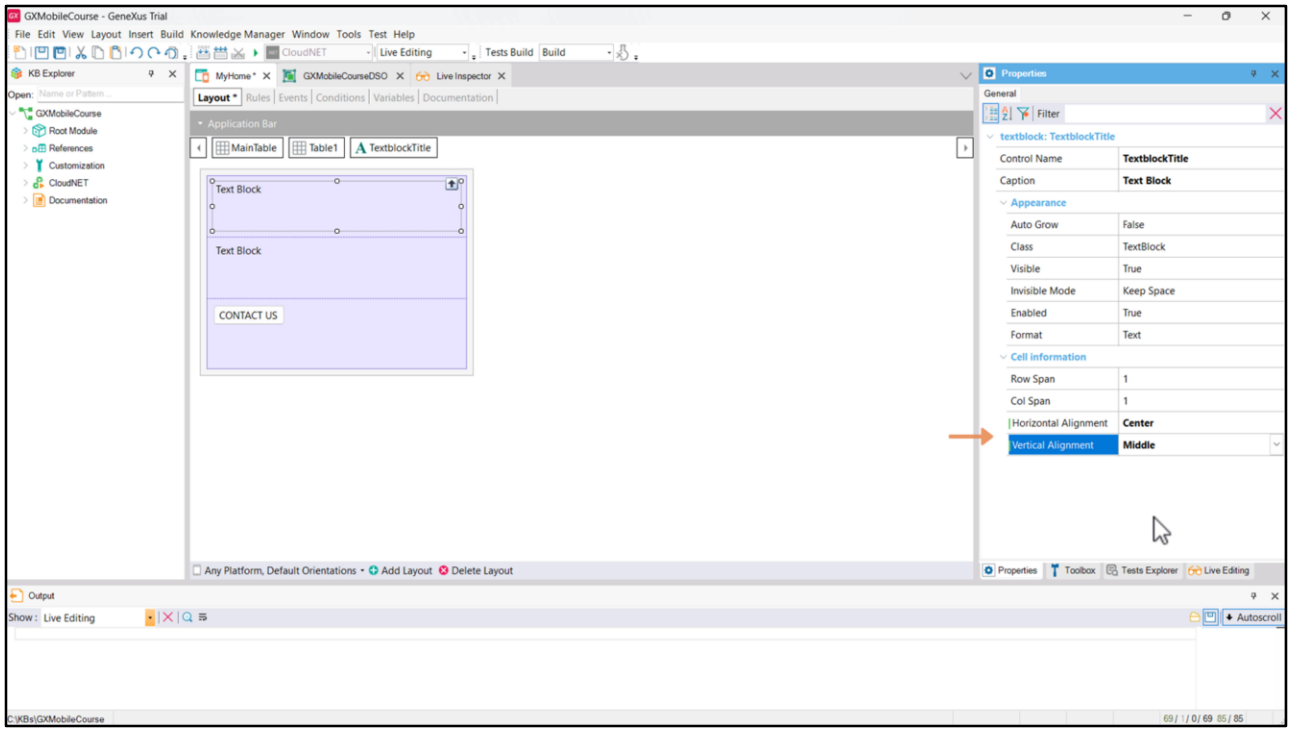


Mas como funciona o Live Editing? Quando usamos Live Editing, o Servidor ficará "monitorando" as alterações que fizermos e irá replicar essas alterações nos Metadados acessados pelo dispositivo móvel (no nosso caso o emulador) e que são utilizados para desenhar a tela no dispositivo e definir seu comportamento.

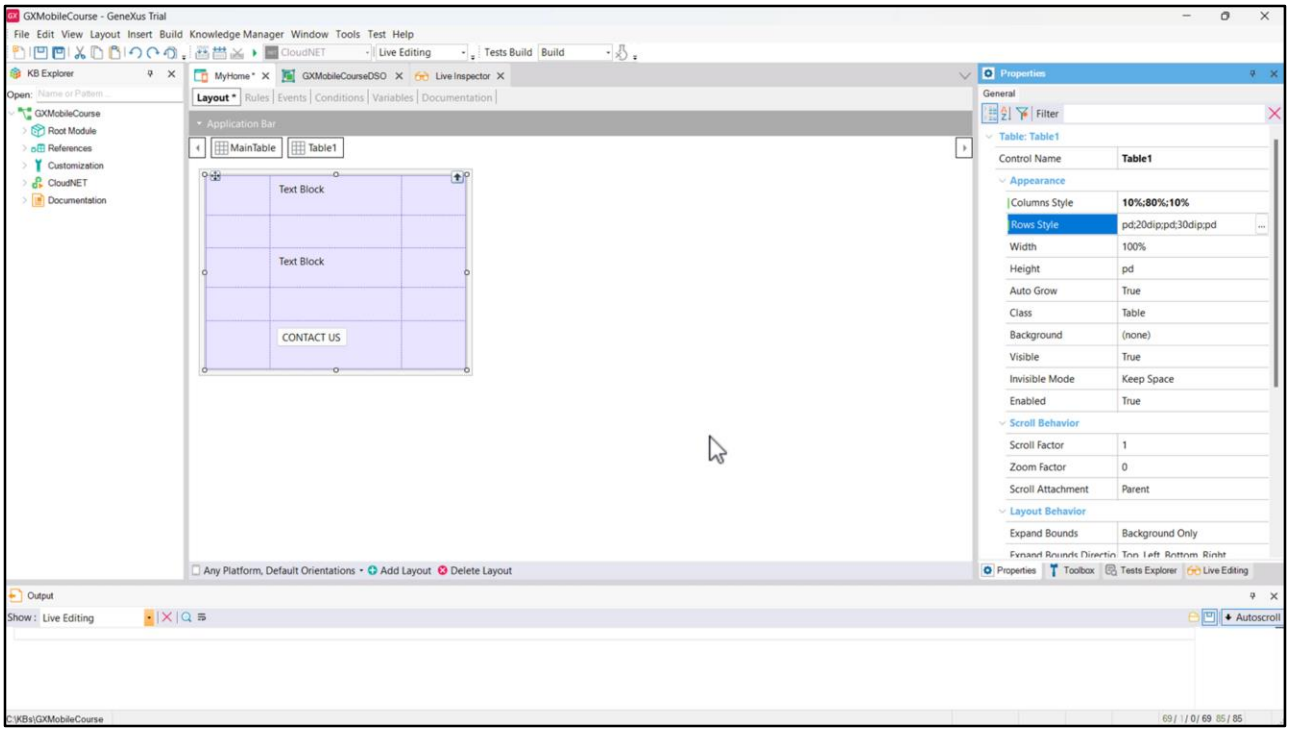
O Live Editing tem algumas restrições de uso, por exemplo:

- Não funciona para objetos menu.
- Não podem ser visualizadas as alterações que impliquem acessos ao servidor. Por exemplo, se adicionarmos um atributo ao form, o valor precisará ser recuperado da base de dados em um evento do servidor (Start, Refresh ou Load), portanto será necessário compilar a aplicação novamente.
- Como o live editing funciona apenas com a aplicação compilada, não funciona se estiver prototipando com o GPN (GeneXus Project Navigator).

Para conhecer mais detalhes sobre o uso do Live Editing e do Live Inspector, convido você a visitar o conteúdo da wiki de GeneXus: https://wiki.genexus.com/commwiki/wiki?27806

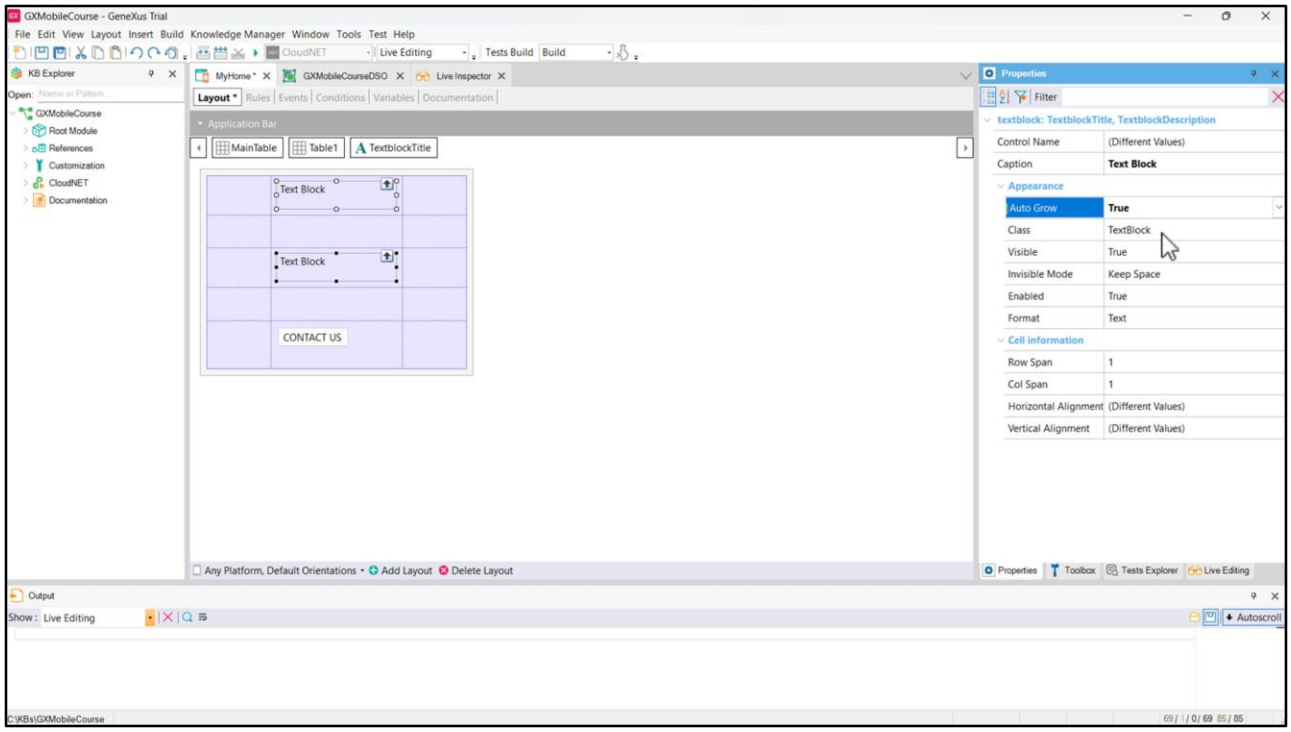


Voltemos ao nosso exemplo. Como queremos o título centralizado na célula que o contém, podemos utilizar as propriedades Horizontal Alignment e Vertical Alignment com os valores Center e Middle, respectivamente.

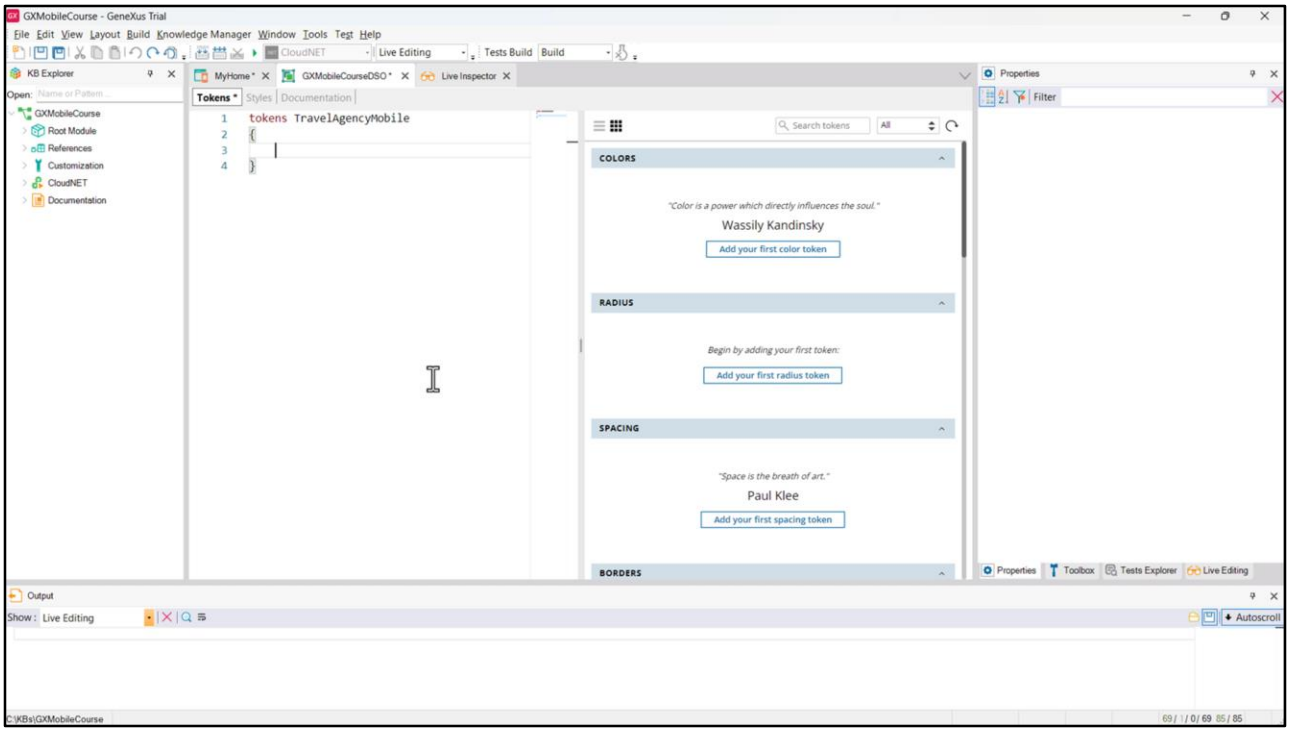


Vamos alterar também a propriedade Horizontal Alignment do TextblockDescription, para que o texto fique justificado.

Vamos adicionar mais duas colunas à tabela, uma à esquerda e outra à direita do conteúdo, e mais duas linhas, uma abaixo do título e outra acima do botão, com a finalidade de gerar espaços que tornem a informação mais visível. Se observarmos a propriedade Columns Style, vemos que cada coluna ficou ocupando 33% da largura total da tabela. Vamos alterar os valores para que a primeira coluna ocupe 10% da largura total, a segunda (que é a que comporta o conteúdo) 80% e a última os 10% restantes. Agora, vamos alterar os valores da propriedade Rows Style, para indicar que a segunda e a quarta linhas, que são as separadoras, ocupem 20 e 30 dips respectivamente.

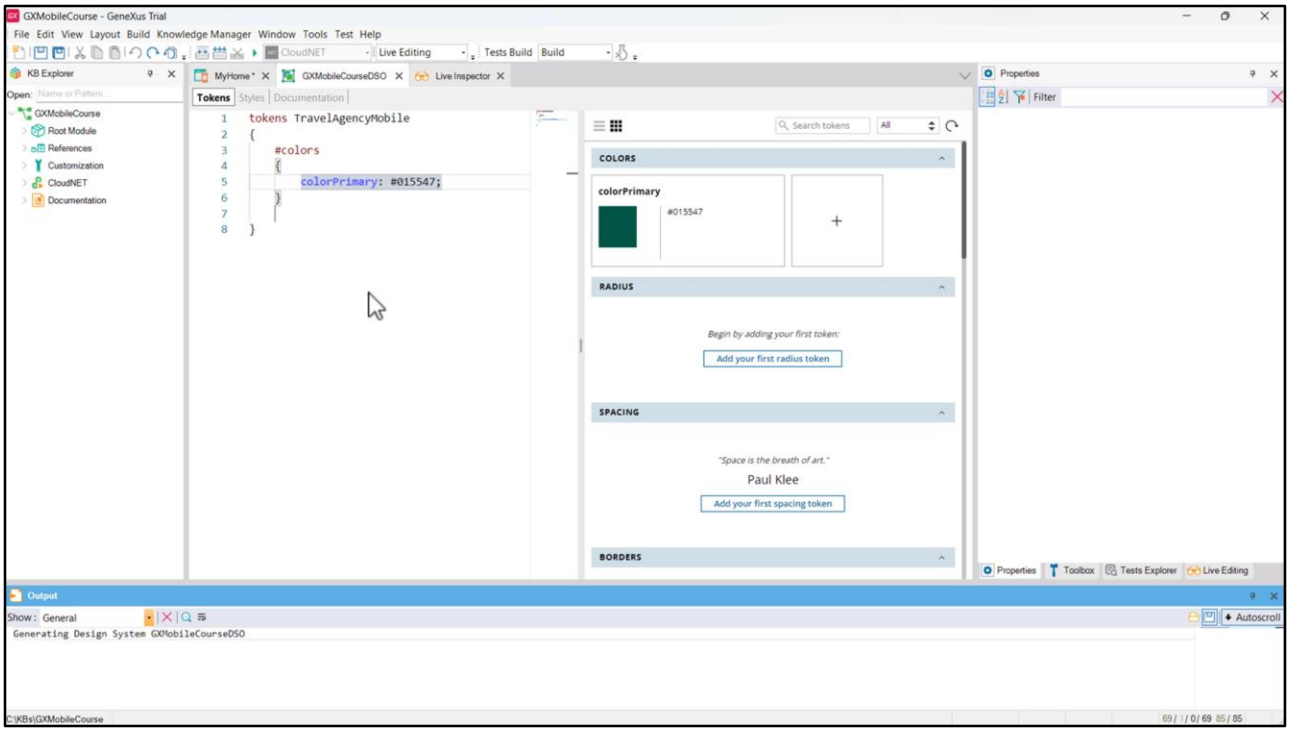


Vamos também definir a propriedade Auto Grow das células que contém os Textblocks como True, para que caso não atinjam o espaço predefinido que ocupa cada célula, se expandam de modo que todo o conteúdo fique visível.



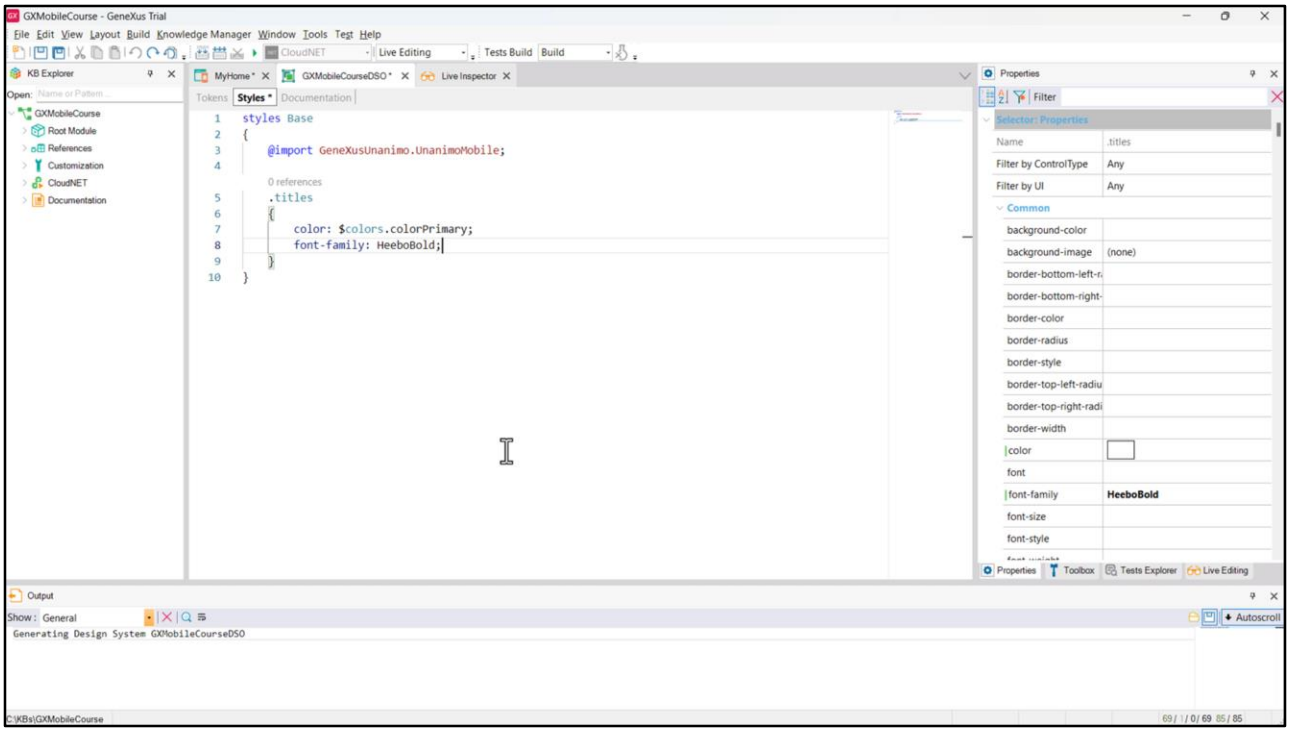
Vamos agora começar a mudar o estilo dos controles de nosso Panel. Comecemos pelo título: vamos alterar a cor e o tipo de fonte. Poderíamos também definir em nosso DSO o alinhamento centralizado, mas não será necessário já que o configuramos através das propriedades, e estas pertencem a uma ordem hierárquica superior ao DSO: ou seja, se estabelecermos o alinhamento centralizado em nosso DSO, os valores das propriedades passariam por cima deles.

Vamos dar um nome ao conjunto de tokens que iremos criar, com a finalidade de que nosso DSO fique organizado: colocamos TravelAgencyMobile.



Vamos começar definindo uma constante de cor: daremos a ela um nome que a identifique, por exemplo, Primary, mas para que todas as cores sejam facilmente identificáveis apenas olhando o código, manteremos a mesma nomenclatura para toda constante de cor que criarmos, e seu nome começará com a palavra *color*. Se já sabemos o valor hexadecimal da cor (por exemplo, porque a equipe de designers nos passou em um arquivo de Figma), o colocamos aqui. Usaremos para o título este verde escuro.

Vemos que à direita temos outro editor que está sincronizado com o da esquerda, ou seja, podemos trabalhar em um ou outro e o que fizermos em um será refletido no outro.



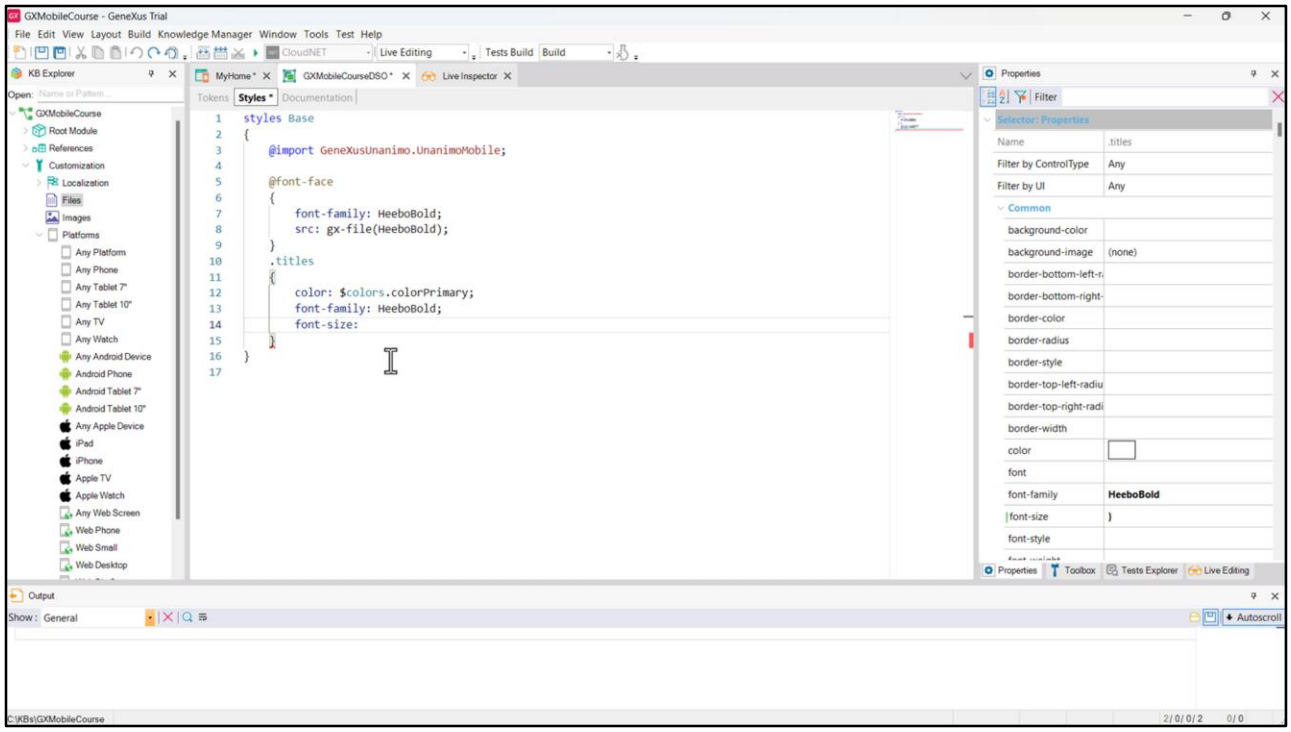
Na aba Styles, criaremos a classe que dará estilo ao conteúdo.

Daremos o nome de *titles* e adicionaremos as propriedades CSS necessárias para estabelecer as características desta classe.

A cor que vai aplicar é a do token que acabamos de definir, por isso escrevemos:
color: $(que nos permite referenciar um token) colors (porque será um token de cor).colorPrimary;
Com isso estamos indicando que todo controle que tenha esta classe associada, terá esta cor.

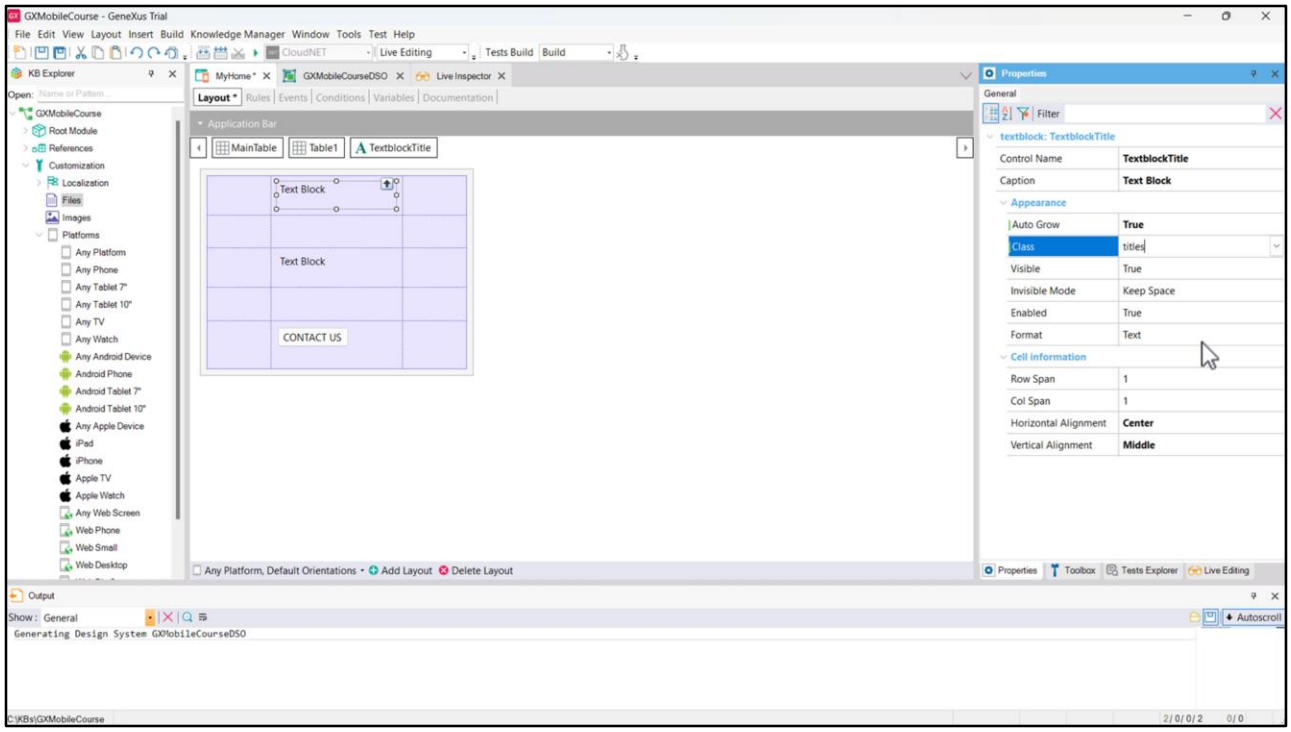
Vamos agora especificar a família de fonte a ser utilizada:
font-family: HeeboBold;

Esta família de fontes não faz parte das predefinidas, ou seja, aquelas que entendem por padrão os navegadores. Isso significa que devemos definir no DSO qual é essa fonte, de onde ela é obtida. Para incluí-la teremos que importar o arquivo da fonte para a KB para poder referenciá-lo na propriedade font-family do DSO.

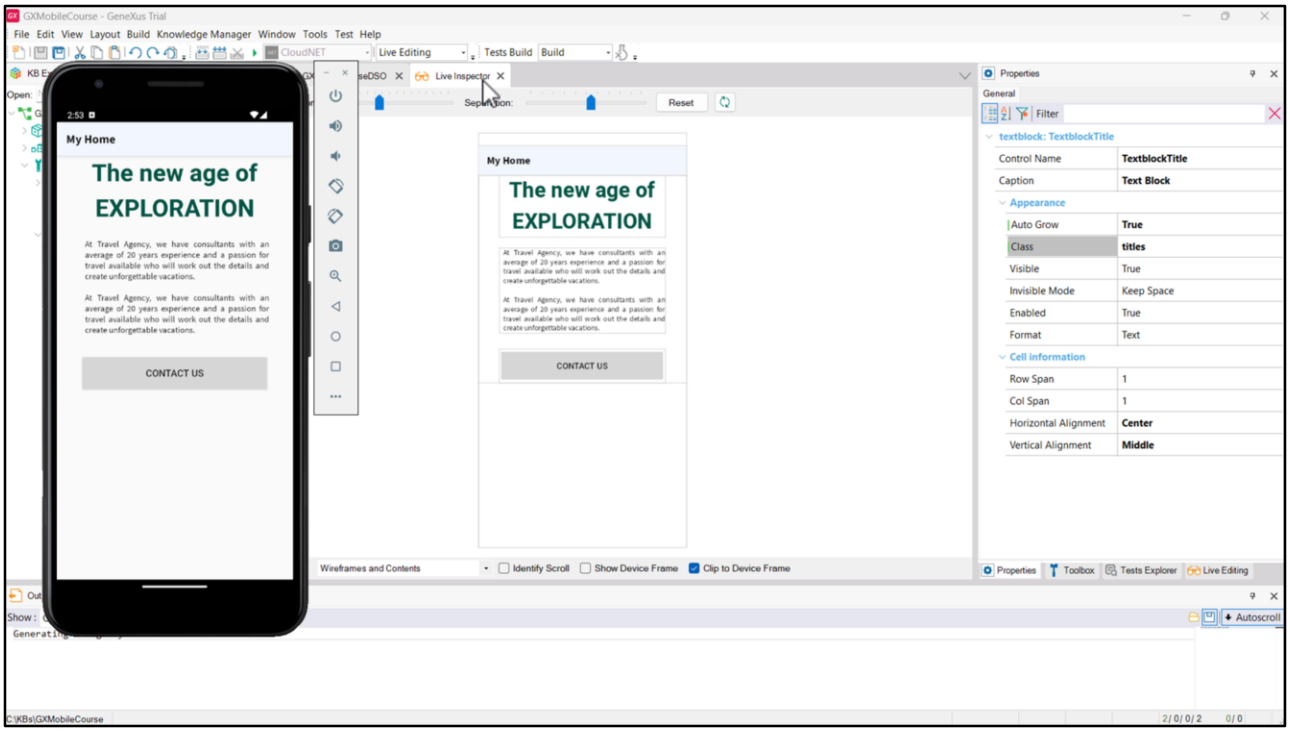


Vamos em Customization → Files → New file, criamos o arquivo chamado HeeboBold e o escolhemos a partir de sua localização. Aqui colocamos o nome que terá a família de fontes, e na propriedade src indicamos de onde é tomado esse arquivo. Com a função gx-file() recuperamos o arquivo acessível em nossa KB. Com isso simplesmente estamos dizendo ao DSO para incluir esta fonte, mas depois onde se usa, é na referência da classe através da propriedade font-family que definimos para nossa classe titles.
font-family: HeeboBold;

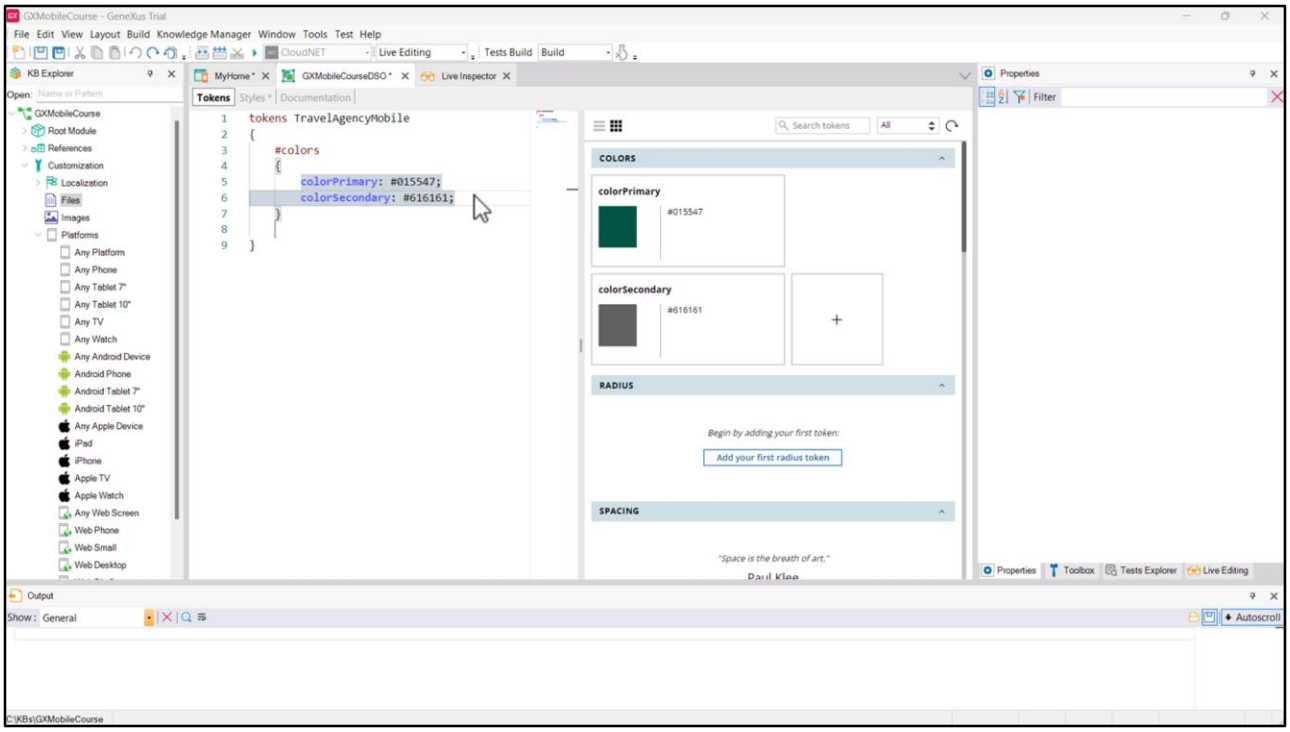
Com a propriedade font-size indicamos o tamanho que queremos para o título.



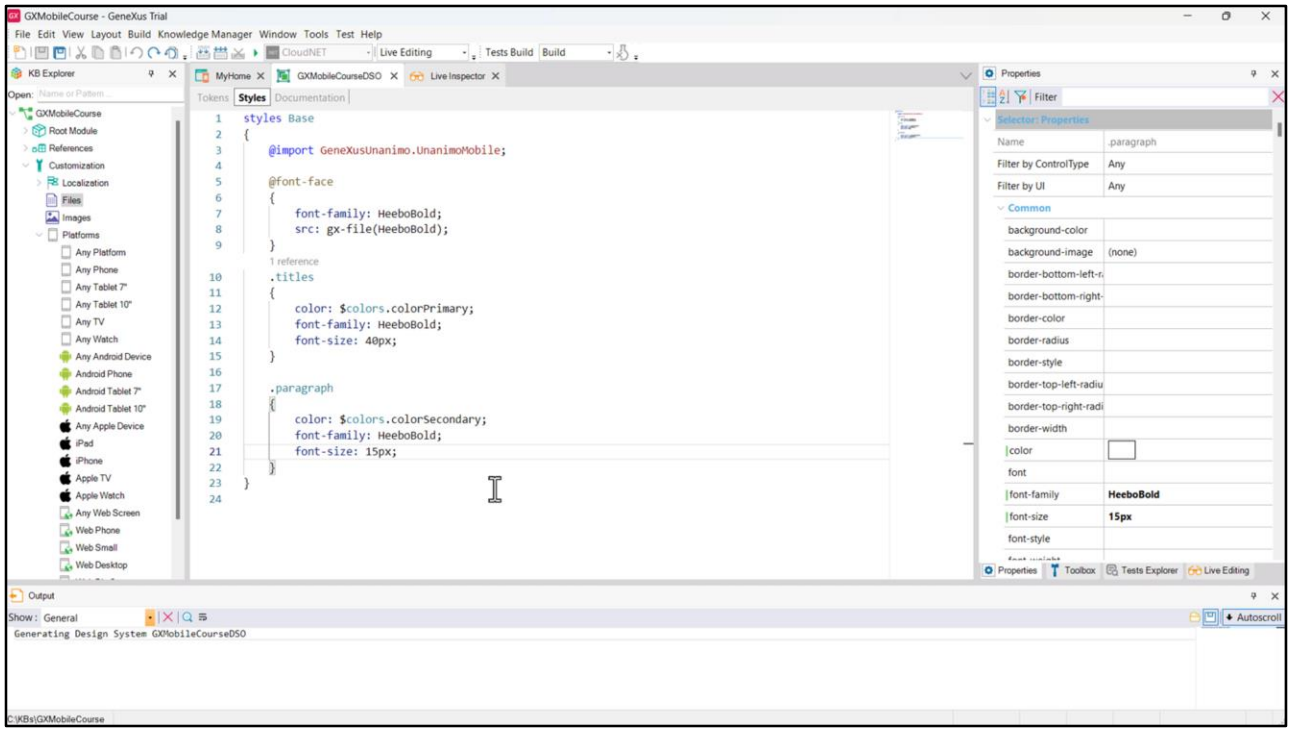
O próximo passo é indicar ao nosso controle Textblock que use a classe *titles* que acabamos de definir.



Já podemos ver no Live Editing os efeitos do que fizemos em nosso DSO.

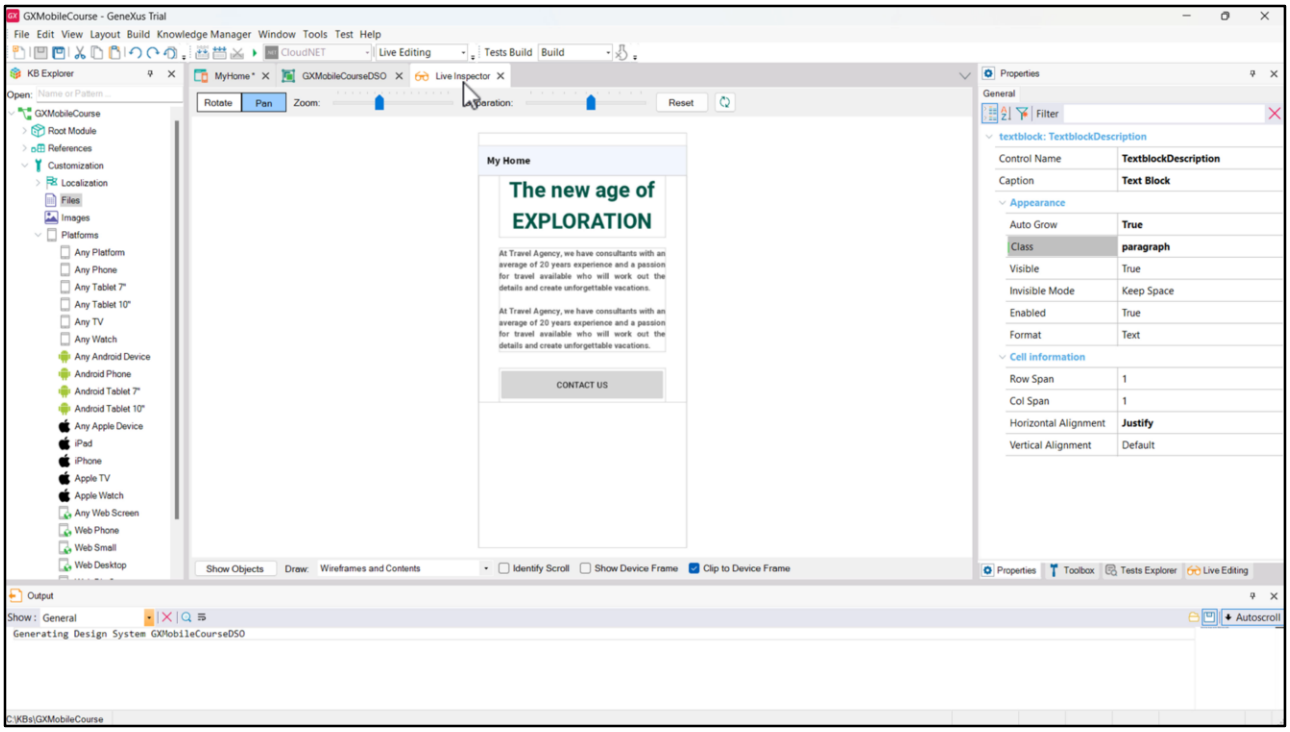


Vamos fazer o mesmo para o texto informativo. Vamos criar outro token de cor que chamaremos de colorSecondary, desta vez um cinza escuro...

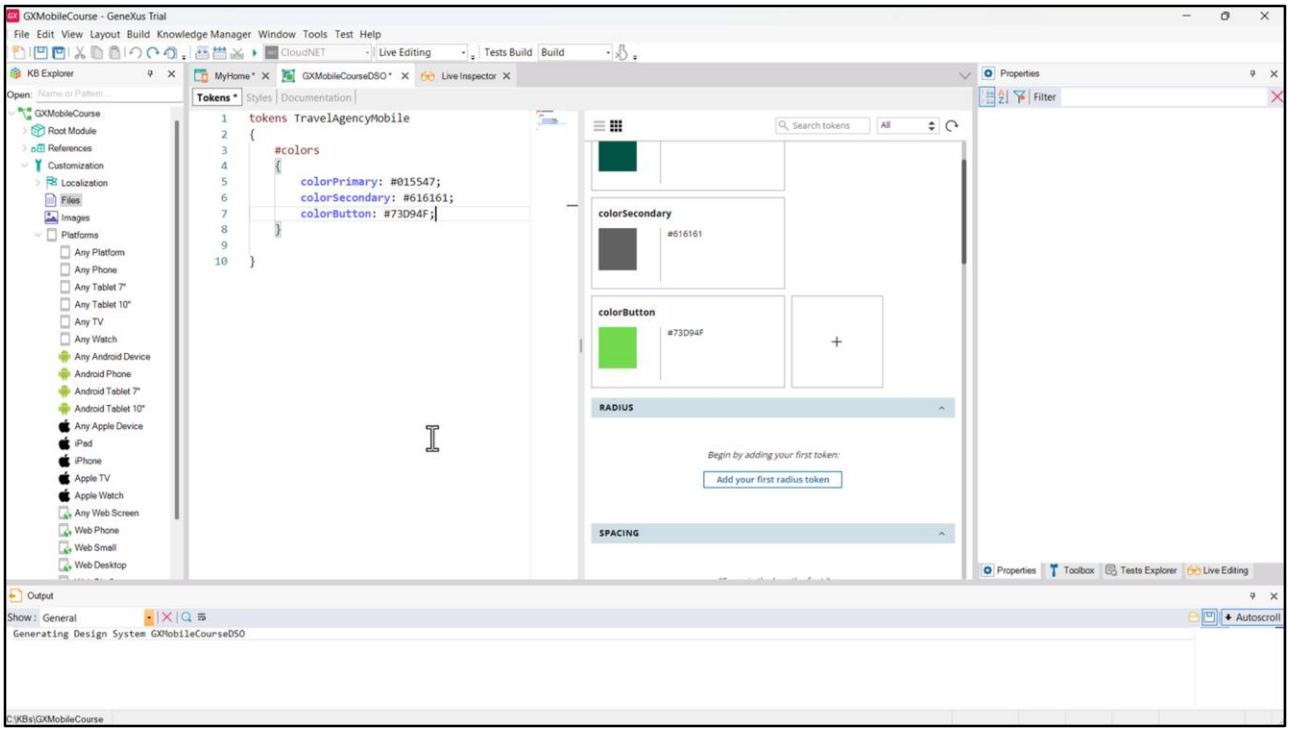


… e uma classe chamada *paragraph* para a qual estabelecemos as seguintes propriedades:

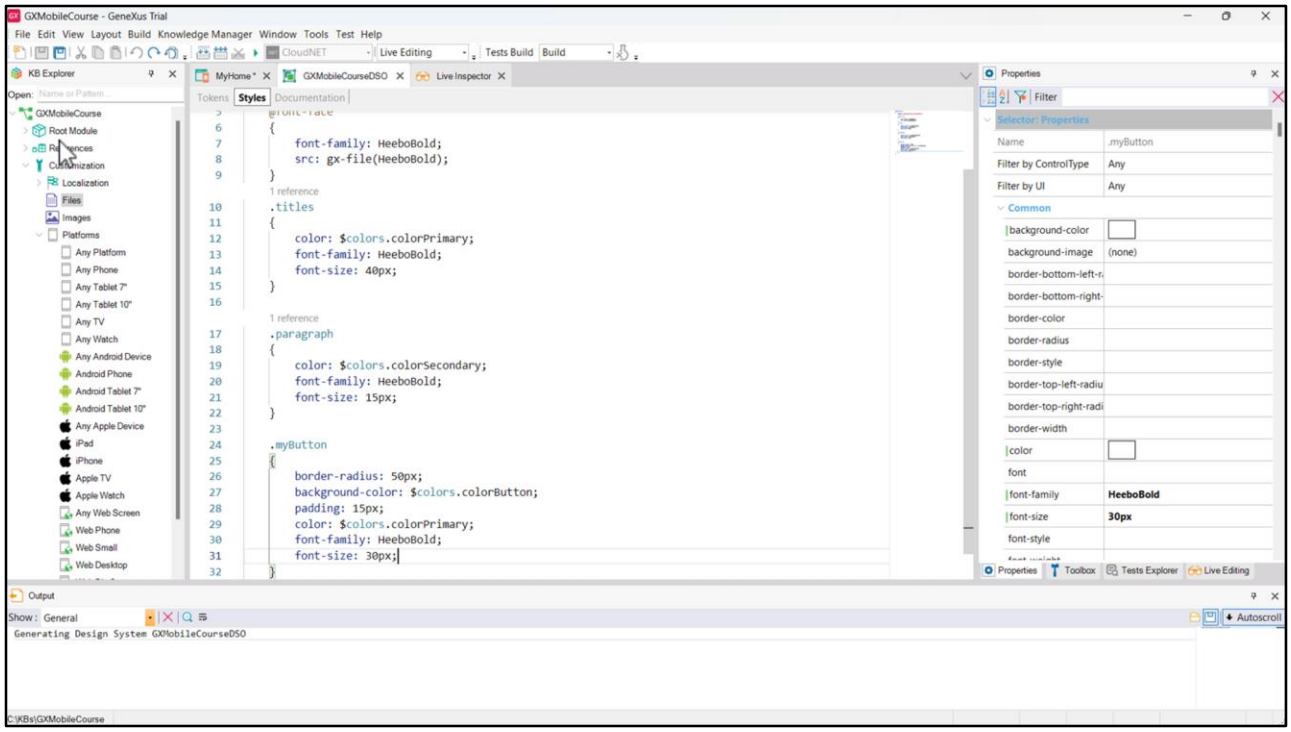
- color
- font-family
- font-size



Vamos atribuir ao controle TextblockDescription a classe *paragraph* e vejamos como ficaram as alterações.



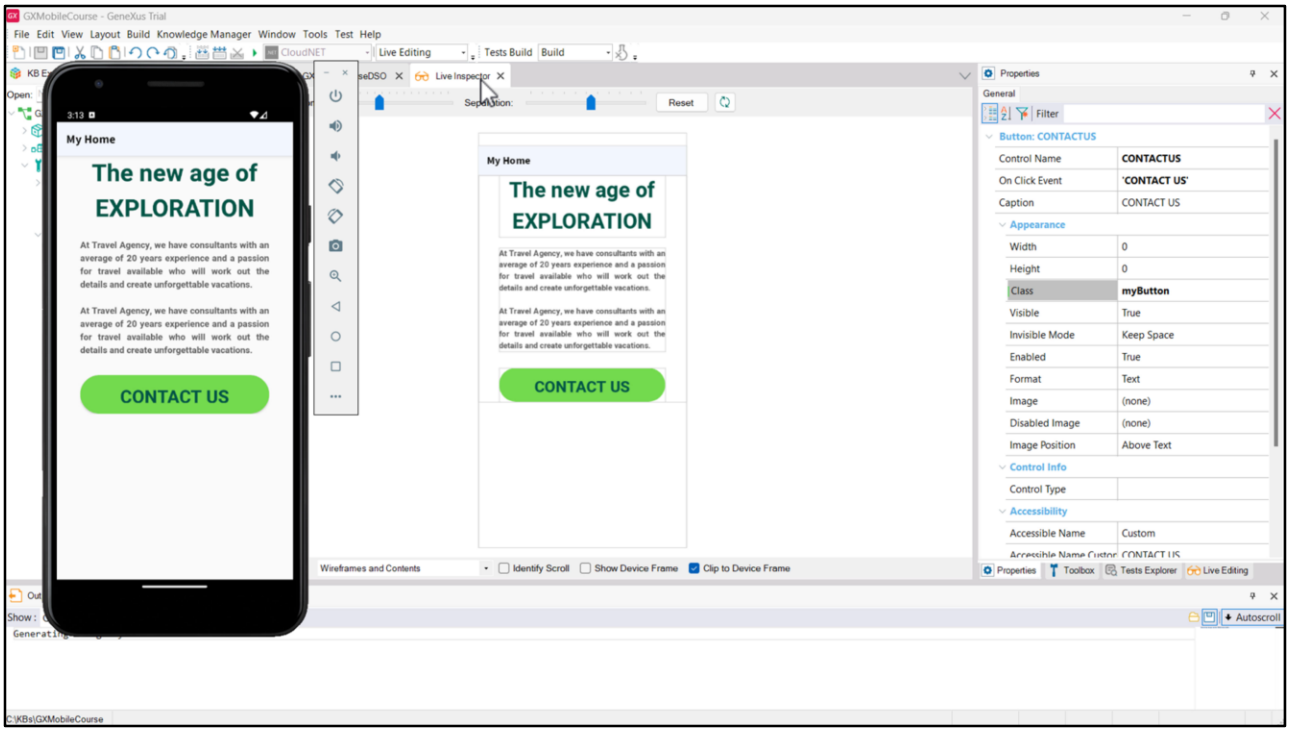
Agora, nos resta dar estilo ao botão. Para isso, vamos criar um novo token de cor chamado colorButton para associar este tom de verde,



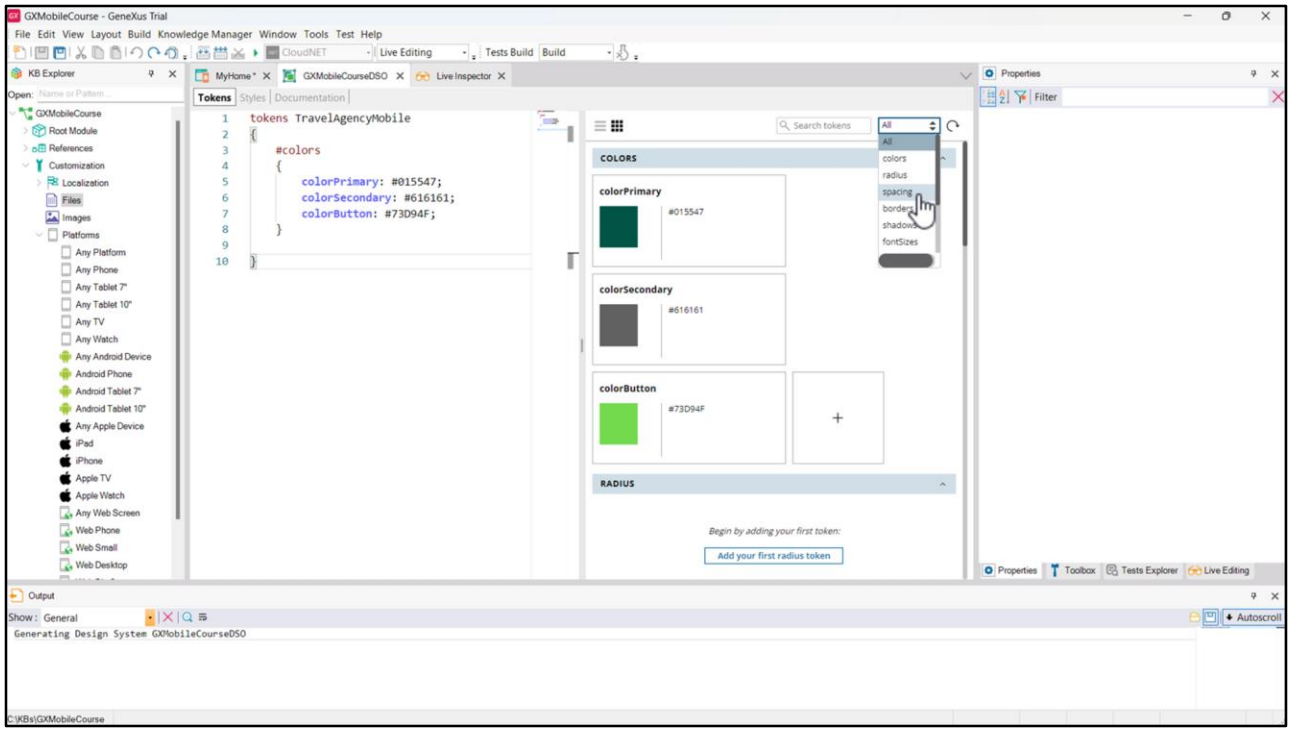
e vamos começar a definir as propriedades da classe *myButton*.

A propriedade border-radius serve para dar o arredondado aos cantos; a propriedade background-color permite estabelecer a cor de fundo; A propriedade padding permite gerar um espaçamento entre o texto do botão e suas bordas.

Agora completamos com o estilo do texto, que é o mesmo que já fizemos antes...

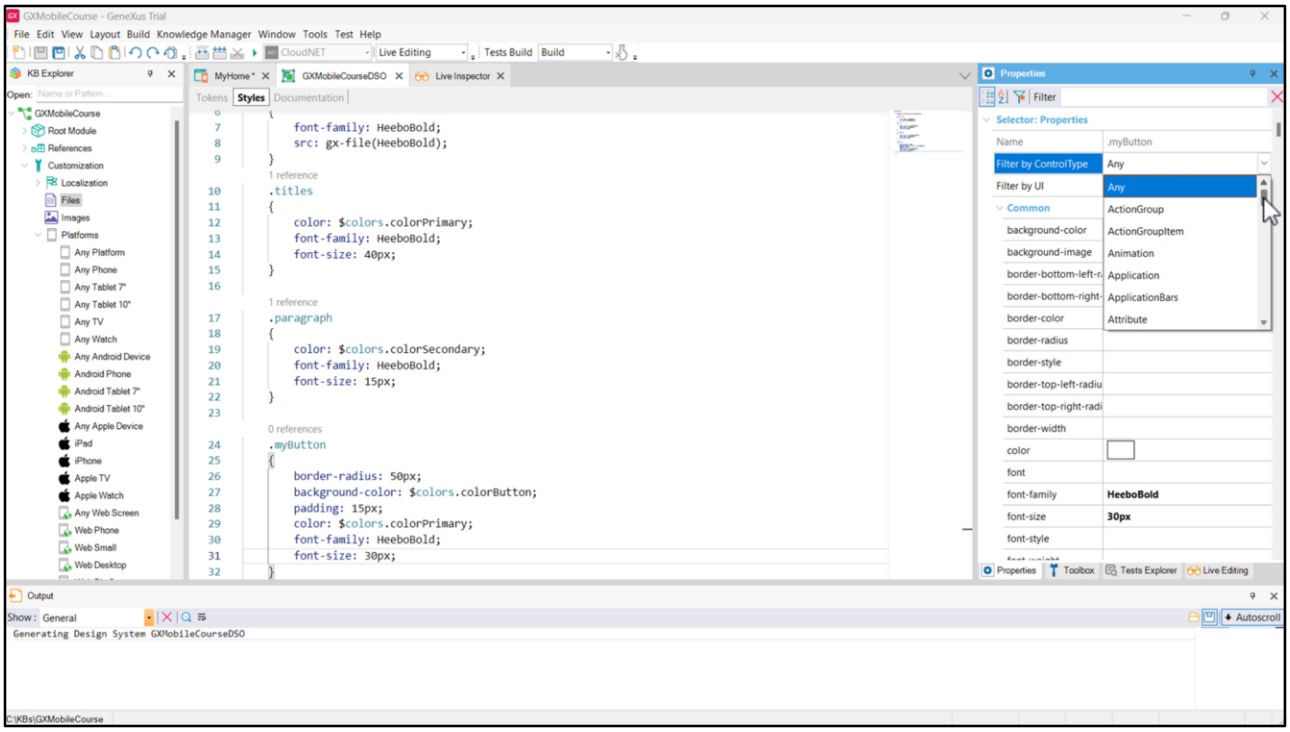


… e vamos associar nossa classe ao botão do Panel.

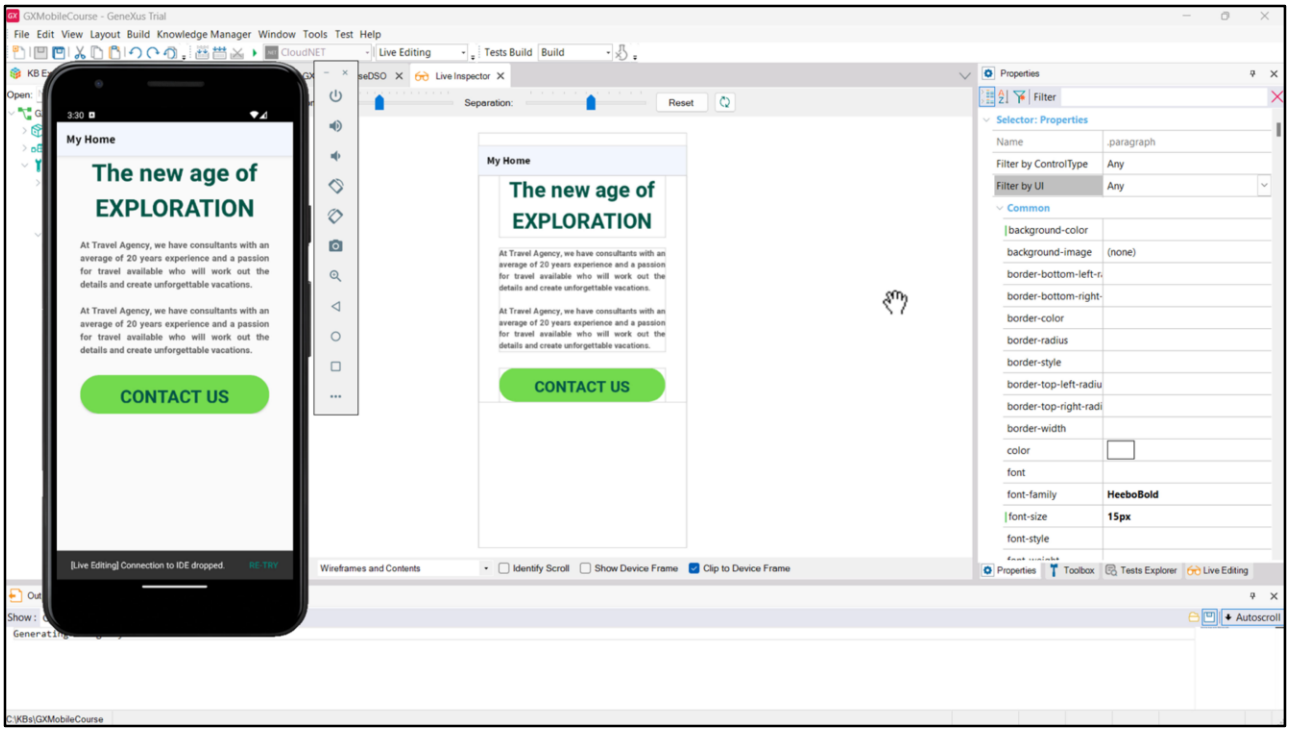


Não só podemos utilizar tokens de cor que são os mais óbvios, mas também podemos defini-los para outros tipos de coisas, por exemplo: para dar espaçamento, para definir os tamanhos de fontes, etc.

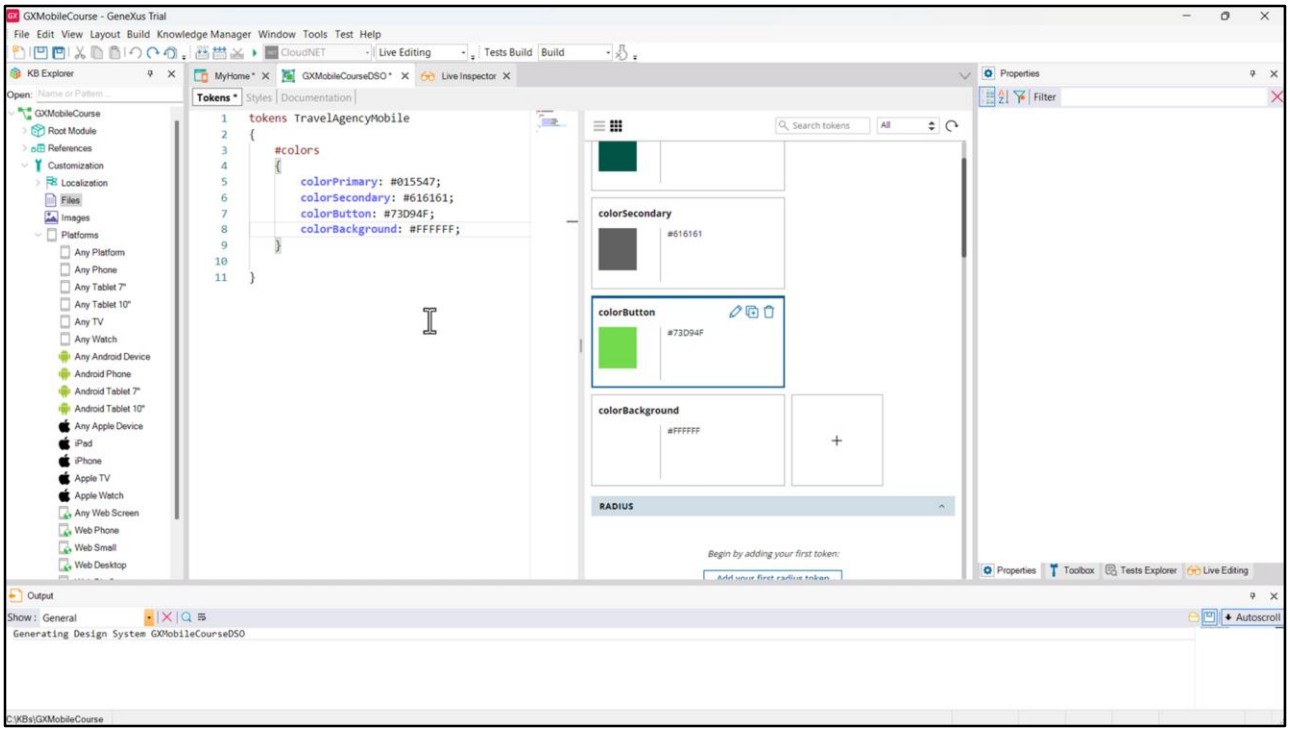
Existem propriedades que são comuns para vários controles (como a border-radius, background-color, color, font-family, font-size, entre outras).



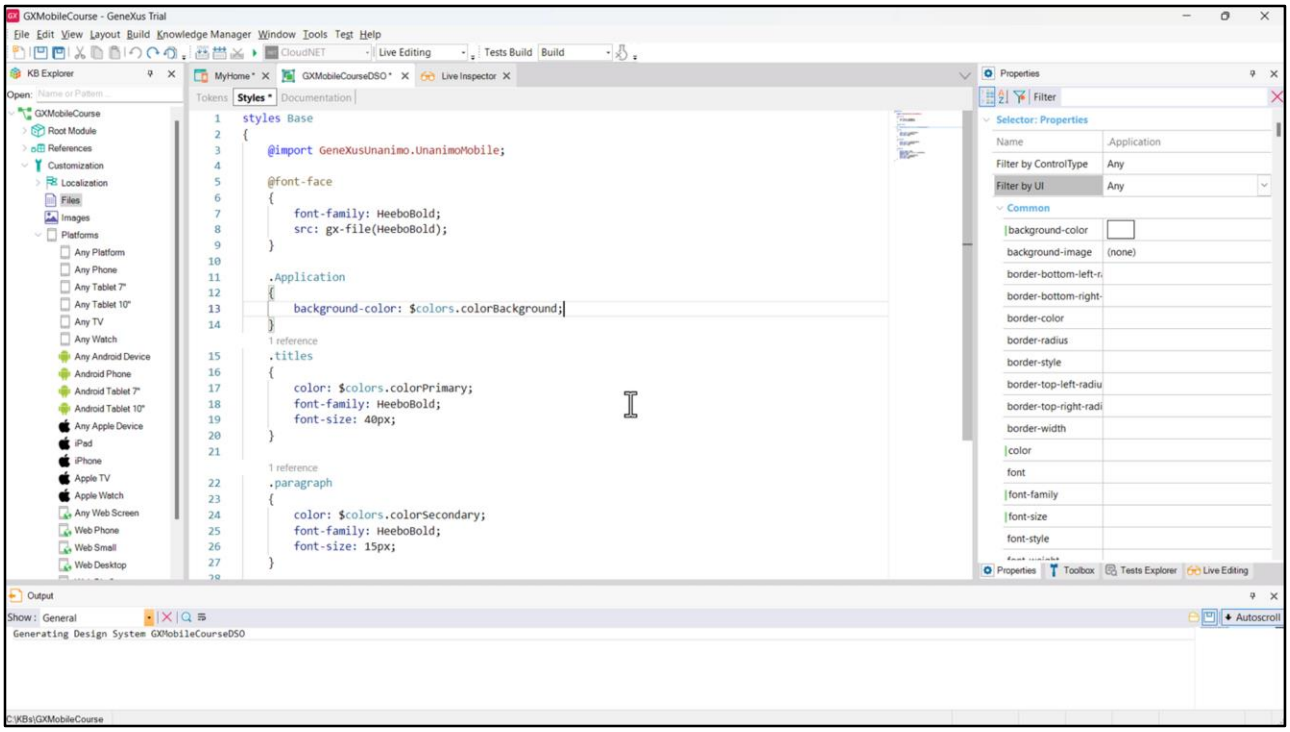
Quando estamos trabalhando na aba Styles, a janela de propriedades nos oferece a opção de filtrar por tipo de controle, onde veremos e poderemos configurar apenas aquelas propriedades que se aplicam ao controle selecionado, e filtrar por UI, onde poderemos escolher o tipo.



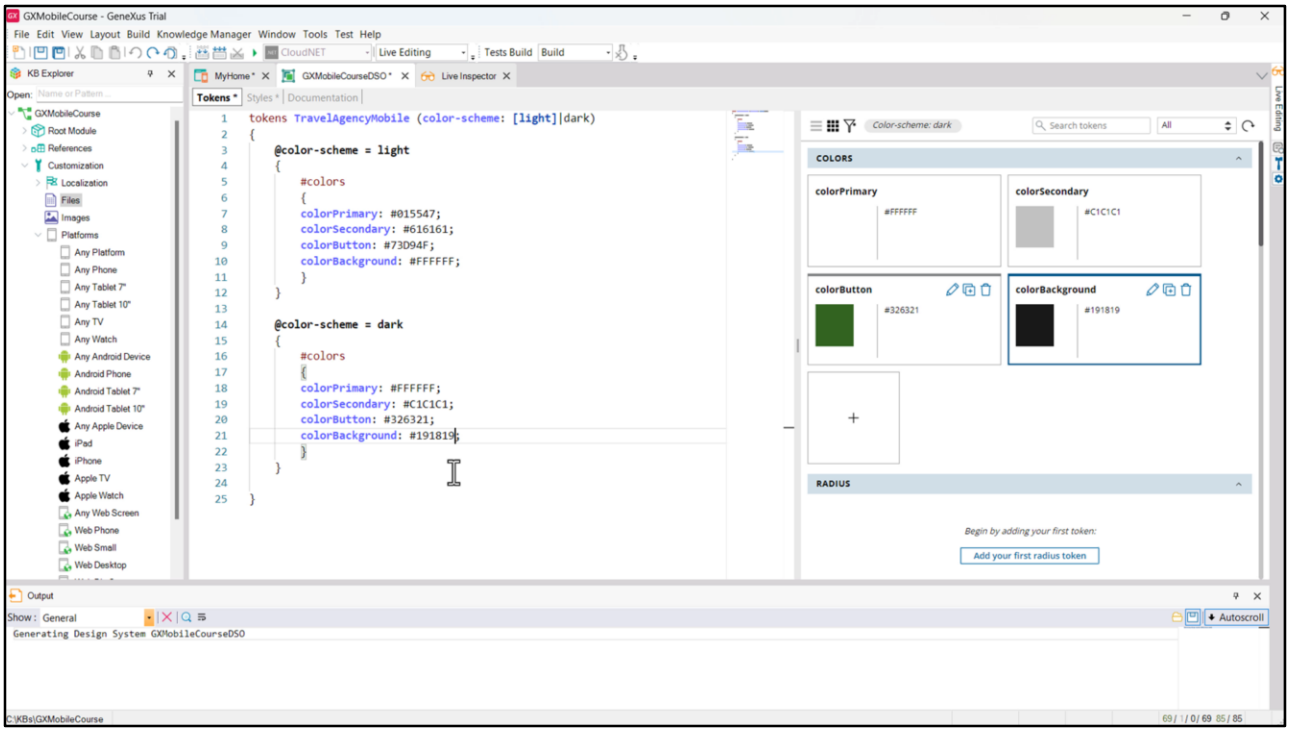
Até aqui, desenhamos o modo *light* de nossa aplicação, mas também podemos desenhar o modo *dark*. Para isso teremos que pensar em variar a cor de fundo, a cor dos textos e também do botão. No caso do fundo, no modo *light* assumirá a cor branca que já temos, e no caso do modo *dark* assumirá um cinza bem escuro. Para o título, no modo *light* será o verde escuro que definimos como cor primária, e no modo *dark* será o branco, para contrastar com o fundo escuro. De forma análoga definiremos as cores para o texto da descrição e para o botão no modo *dark*.



Vamos especificar a cor de fundo da aplicação: criemos a classe *Application* e definamos a cor, que recuperaremos de um token chamado colorBackground.

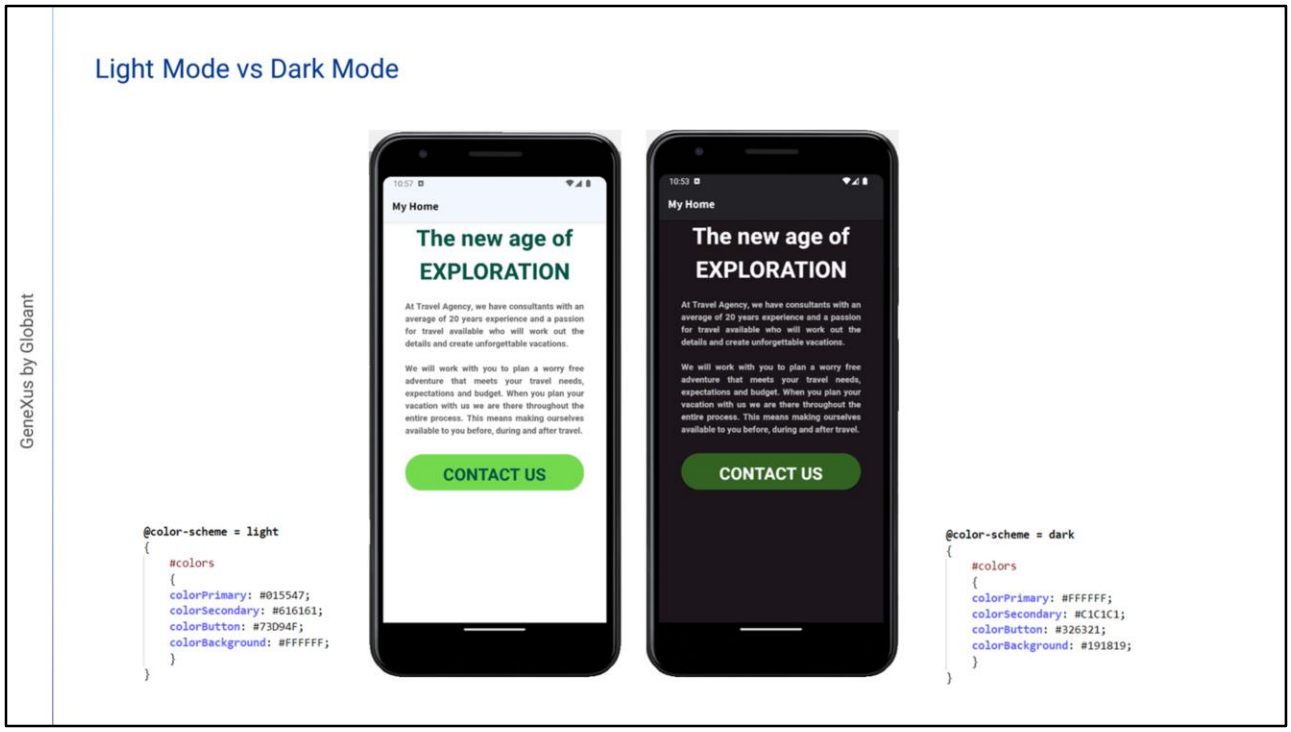


Ao chamar a classe assim, será a que se aplica à tag body de todo HTML de maneira automática.



Para especificar os modos, devemos utilizar as opções: color-scheme é o que nos permite indicar o esquema de cores, *light* ou *dark*. Ao escrever o *light* entre colchetes, estamos indicando que queremos que este seja o modo padrão. Em seguida, variamos os tokens de cores conforme seja o color-scheme. Para o modo *light* utilizaremos essas cores, e para o *dark* copiamos as da *light* e simplesmente vamos alterando as cores que devem ser aplicadas: o título será branco, a descrição será cinza claro, o botão será um verde diferente daquele que usamos no modo claro, e o fundo um cinza bem escuro.

Antes de executar para ver as alterações, defina a propriedade "Enable Preferred Color Scheme" do Panel MyHome como True e faça Build All.



Desta maneira, nossa aplicação estará adaptada às necessidades de todos os usuários.

Vimos uma introdução de tudo o que podemos fazer usando o Design System Object. Em um próximo vídeo estudaremos como importar para nossa KB um design do Figma criado pelos designers.


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