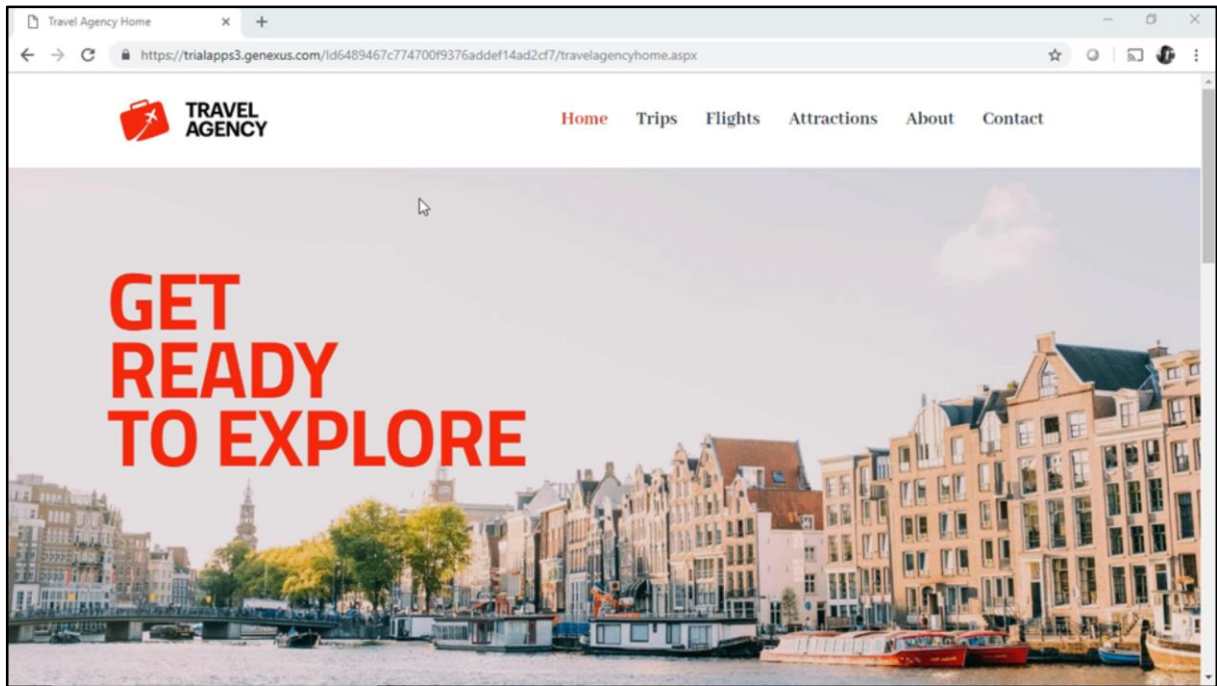


Design Systems in GeneXus

Frontend: OVERLAPPING OF CONTROLS THROUGH CLASSES

GeneXus 16

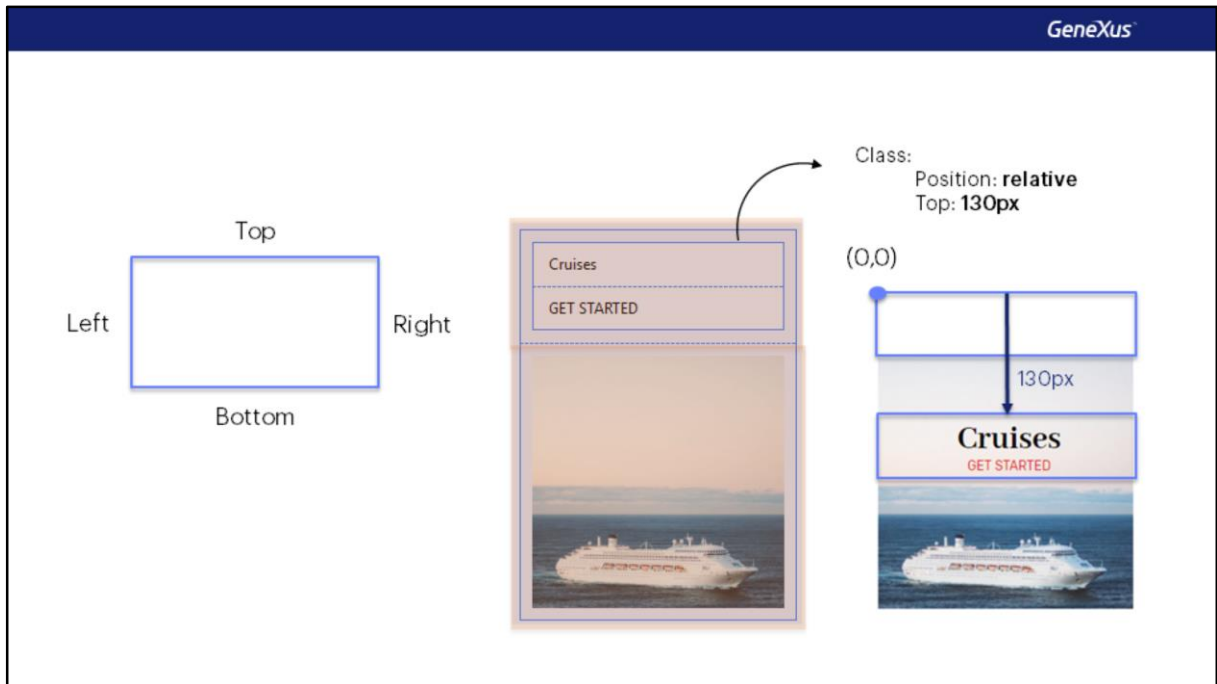


[DEMO: <https://youtu.be/hBcwzYVjtpk>]

Here we are, executing the home web panel. Below is an html file with associated styles, which would pretty much be our classes, interpreted by the browser.

Here, with F12, we may inspect the html element by element, to clearly see which classes and properties have been applied to it, and also vary them as a try. We may, for example, find a specific element in the screen by right clicking/Inspect. Or by selecting in this other way.

The interesting thing is that by standing on the element we want, we will see its size, internal spaces between the element and the border (padding), the border (not present in this case), the margin (not present either). And everything that relates to the styles (classes) and has been assigned to the element.



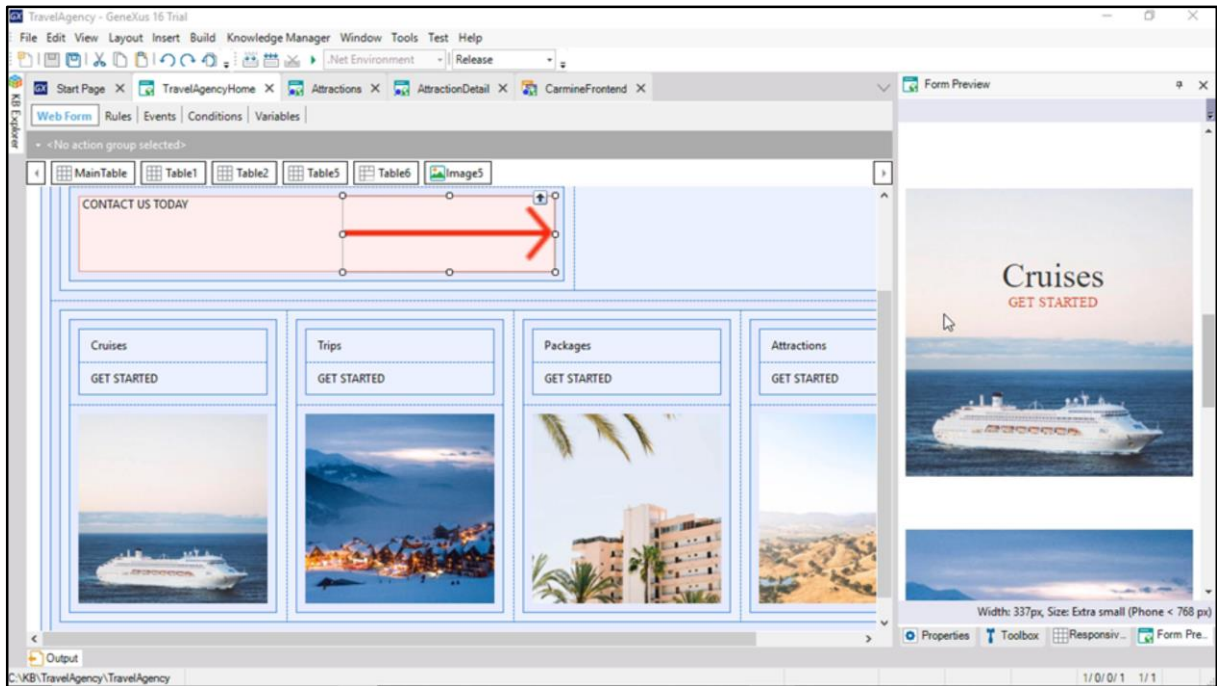
But, as we have said, in addition to establishing margins, paddings, borders, alignments for the control, and other aspects, classes have properties for the positioning of the control: Top, Bottom, Left, and Right, which will enable us to, for example, overlap it with another one.

As in this case. We have a table with two rows. In one there the table with the texts and in the second one the image.

What we need is for the table with the text to have a different position from the one it would naturally take, so that it may overlap the image.

In this case, to the control that we want to move we associated a class for which we indicated a position relative to the position that it would naturally occupy, which is this one, on top of the image, where the origin of its coordinates is the upper left corner. This would be the position (0,0). We will make it move 130 pixels from the top position, where the control would be found naturally.

From the left, and from the right, we leave it in the same location. And from the bottom we do not specify because we are moving it from the top and we do not want to vary the height of the control.



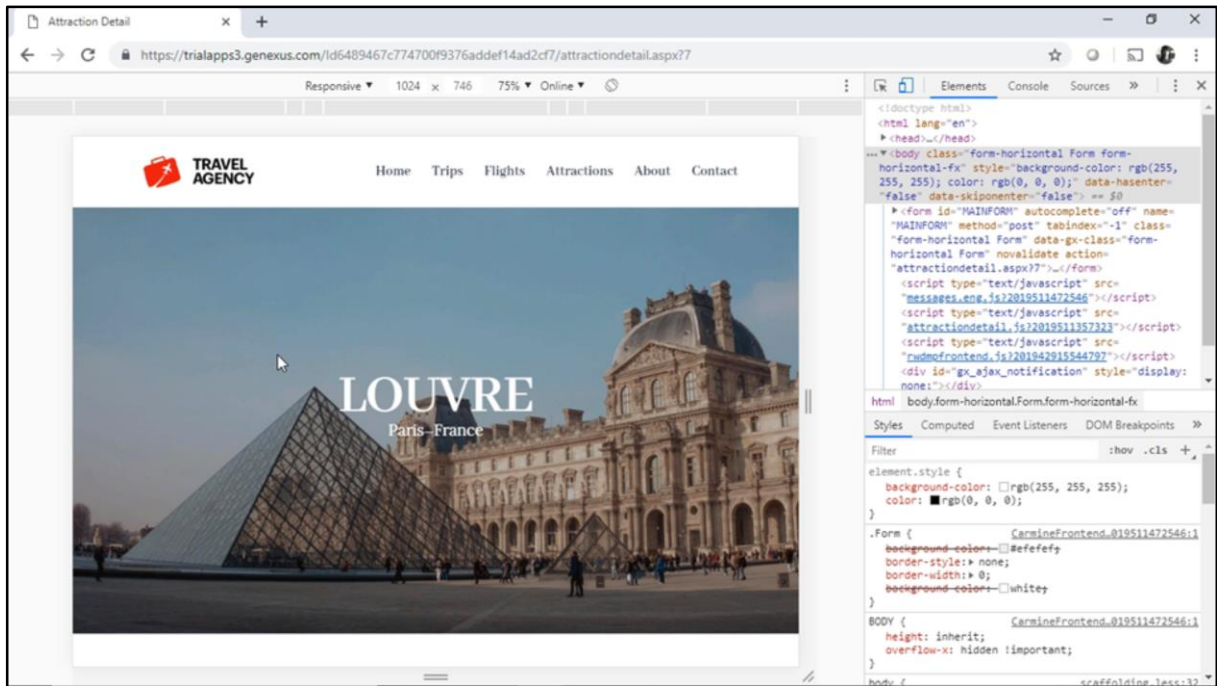
[DEMO: https://youtu.be/mZyERzrR1_0]

The bad thing about using this solution is that the space that the control would occupy if it did not move, is preserved, and that's why we see this blank space here.

To avoid having that space preserved, we should use absolute positioning, as we did in the other two cases of overlapping: the one of the attractions panel, and the one of an attraction's detail –we will see those later.

Or we could use another more simpler option, which does not involve positioning but rather margins, but we will see that at the end.

The disadvantage in this solution is that when we use a fixed movement downwards of 130 pixels, it will not adjust well to responsiveness. As the image below changes its size, the texts remain the same, at the same distance from the upper position.



[DEMO: <https://youtu.be/8Yn8Rb7VO-8>]

On the other hand, let's see the case of the detail of the tourist attraction, which will have an adaptable positioning and it will not leave that empty space we did not want to have.

Here are the texts that have been moved, and here in the natural location (this time we put them underneath instead of on top of the image). We can see the cell with height zero that does not occupy any space.

Here is the web panel in GeneXus. We now temporarily delete this grid (which shows the attraction's extended data) to have a good view of the controls we want.

Here we have the table... where we put the name of the attraction and its country and city. As opposed to the previous case, here, the text data is not fixed, and it is taken from the database. For the city and country we decide to place them inside a **flex** control because we want to always have the city and the country in one row, next to each other, without unnecessary blank spaces between them, regardless of the length of their content. As it is taken from attributes, the content may be somewhat long, and we don't want it to leave a fixed space for each one, as it would happen if we put them on a table. This is the main advantage of a flex control.

Note that the direction selected for the elements is the default value, Row, so that they are shown in a row, and not in a column.

So, we will want this control with the texts to be located not where it would be naturally, that is, below the photo, so we will have to modify its position.

Note that, in this case, instead of place the photo and the texts on a table, we place them inside another flex, this time with the Column direction. We could have chosen a table, but that implied changing one more property (we will mention this later).

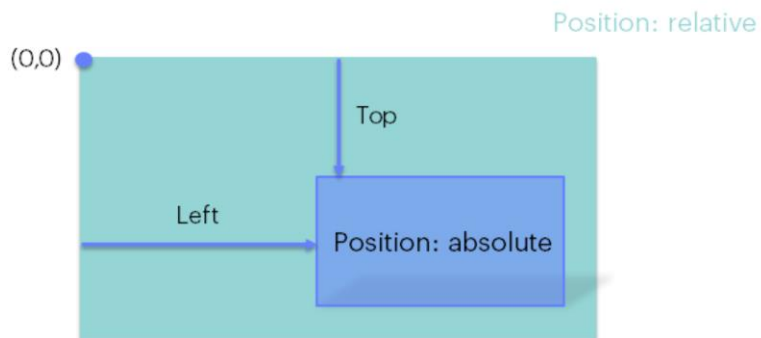
We assign 600 pixels in height to the image, which was what was indicated in the design received, for all screen sizes.

If we don't modify the positioning of the table with the texts, it will be below the photo. What we did was instruct, by means of the `TableHeroInsideText` class that we assigned to it, for it to take an absolute position in relation to its container, that is, the flex where the photo is located.

We are saying that, from left to right, it must stick to the borders of the container (thus making its content centered), and to move, at the top, 40% of the container's height. When the size is given as a percentage, it will be proportional to the height of the container.

A burdensome detail that we must mention: for your reference point to be the parent (this flex), we had to assign to it a relative position (even when we don't change its position in relation to itself, it does not matter). Otherwise, the referent would be the first ancestor found with relative positioning. By default, the flex has "static" positioning, meaning "the place where it is naturally located".

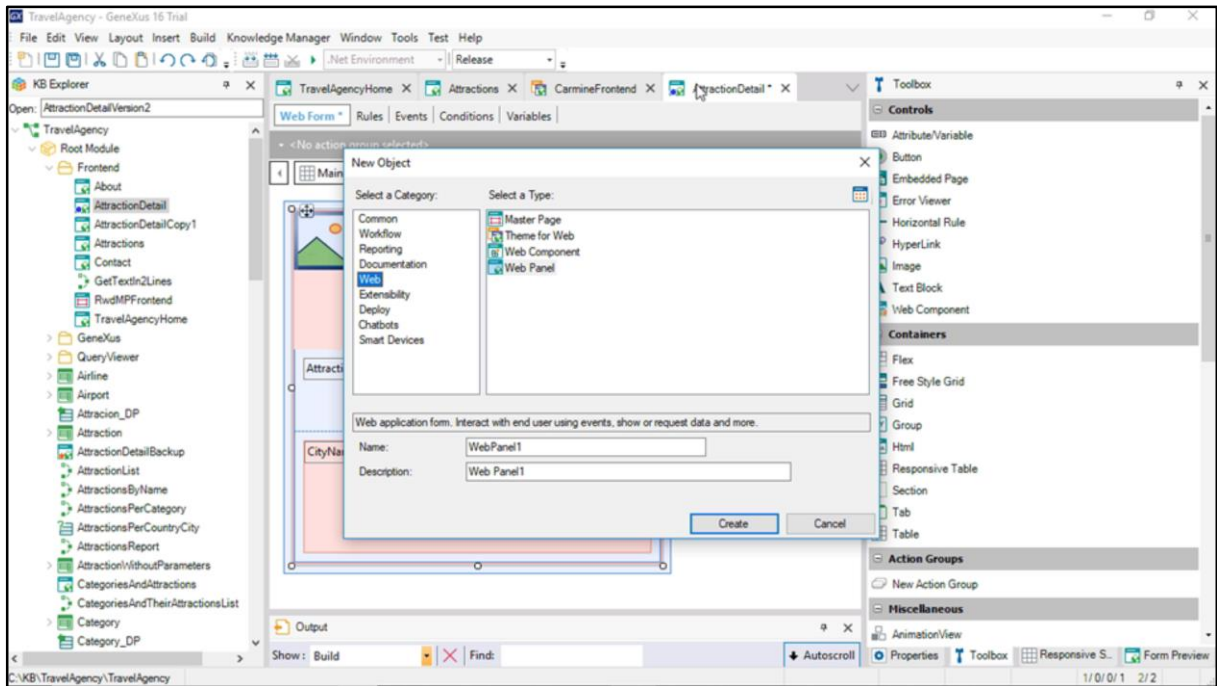
We will now see this so that it is useful when you decide to work and to explain a better solution that solves the defects we will point out in this one we see now. You may skip it now if you're not interested. To have an idea of the importance of classes and controls, what we have seen so far will suffice.



From a graphical viewpoint, it would be like this:

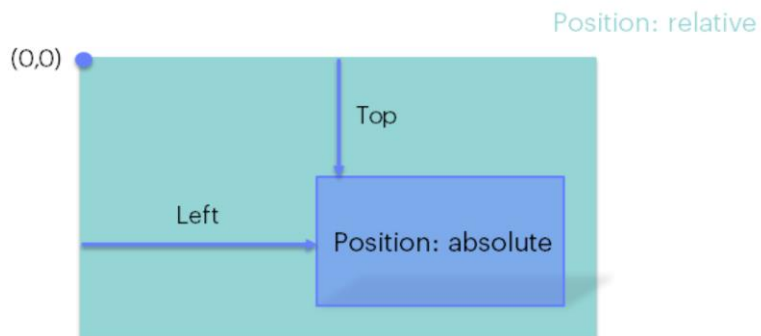
When we establish an absolute positioning for a control, it is withdrawn from the screen layer where it was “naturally” located and it is placed in a higher layer. As it is withdrawn, its space is not reserved. As opposed to what happens when the positioning is relative.

What is the control in relation to which we position it with top, left, right and bottom?



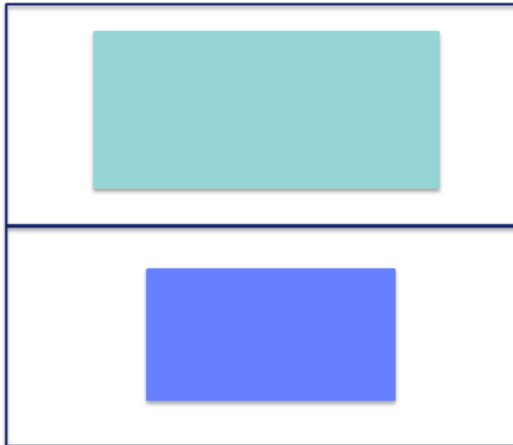
[DEMO: <https://youtu.be/AOn0Ygf0KqQ>]

Controls are all sorted by hierarchy. When we create a web panel, it is initialized with a control: Main table. The first control that we insert will be in a cell in that table. And that's how we will continue to insert controls in sibling cells like that one...
...or inside controls already inserted, so, they will be children, grandchildren, greatgrandchildren. We will have a full family, with grandparents, parents and siblings.



When the control's position is absolute, the referent in regards to which it will be positioned will be the first ancestor with a relative position. Though this ancestor may not change its natural position, it will not matter. (When we assign "relative" position to a control but without setting up top, bottom, left and right, it will actually remain where it was).

Responsive table



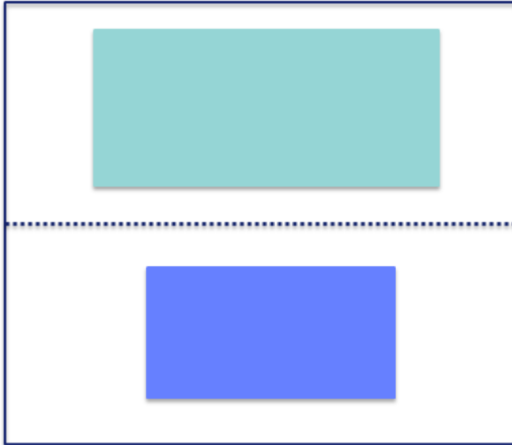
Cell:
Position: **relative** (default)

Cell:
Position: **relative** (default)

By default, the cells of the Responsive Table controls have relative positioning.

Flex

Position: static (default)

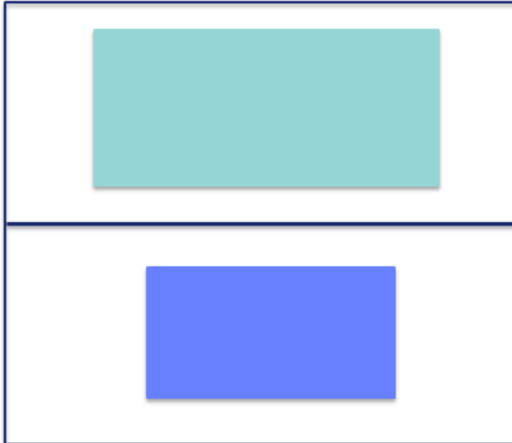
**Cell:**Position: **static** (default)**Cell:**Position: **static** (default)

The others, such as the cells of the flex will have “Static” positions.

Just like the flex itself...

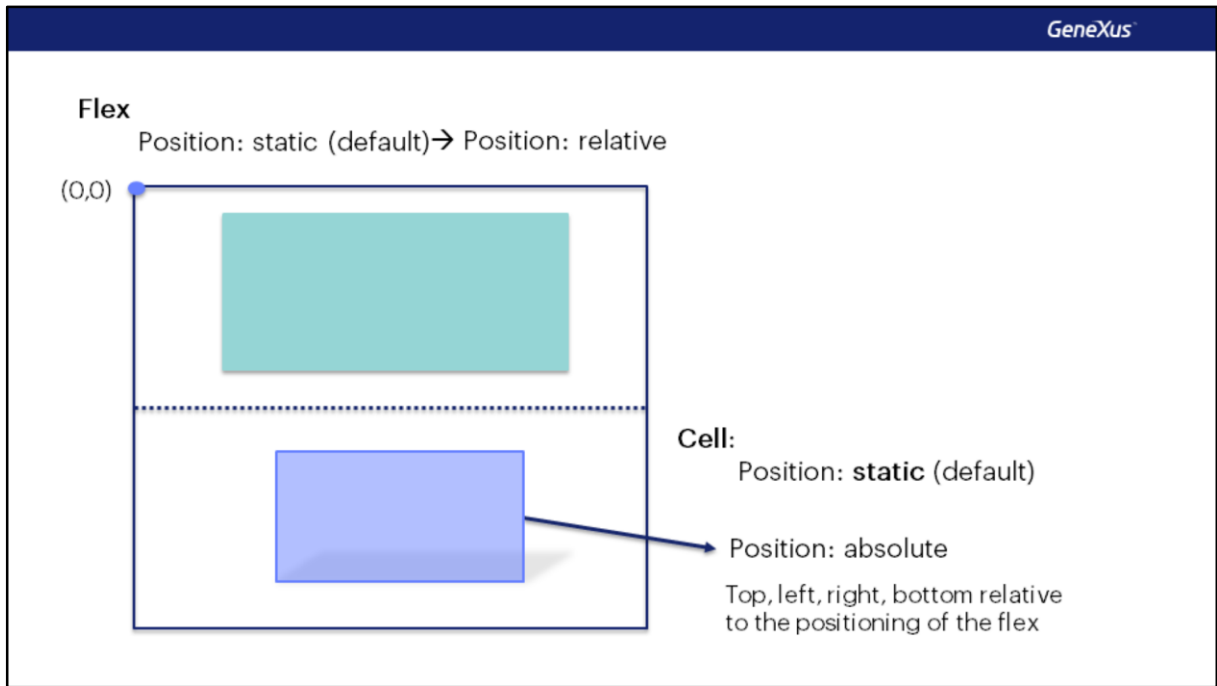
Responsive table

Position: static (default)

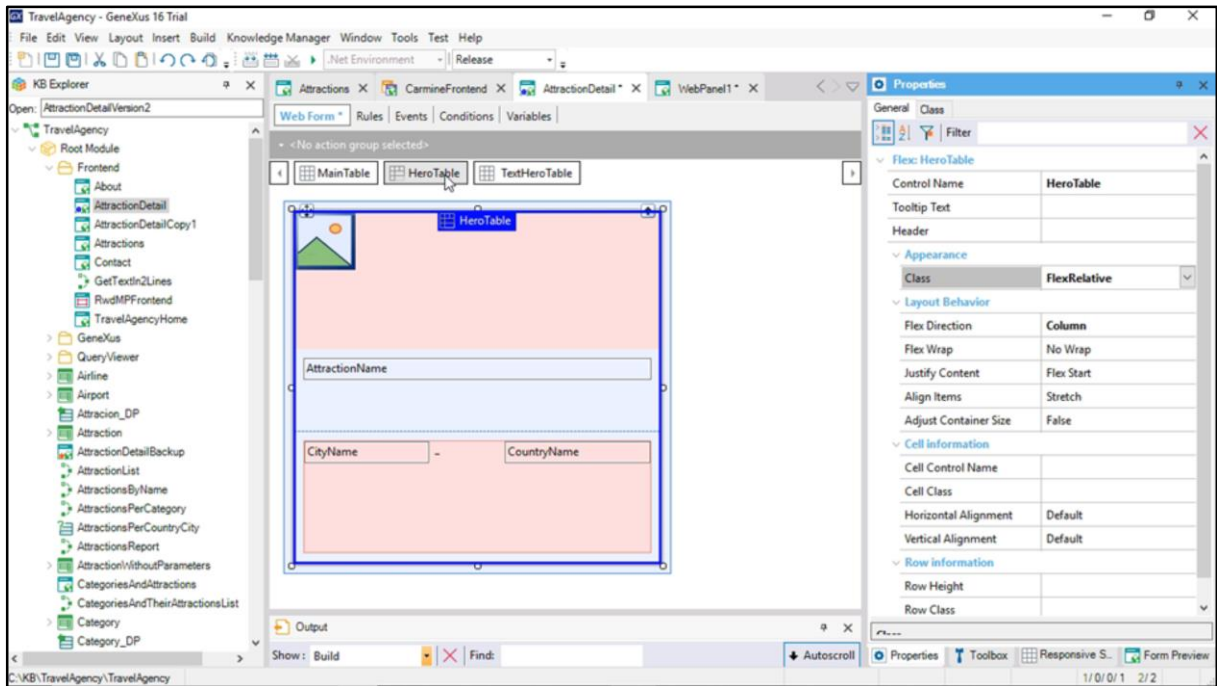


Cell:Position: **relative** (default)**Cell:**Position: **relative** (default)

... and all responsive tables. Only their cells have relative position by default.

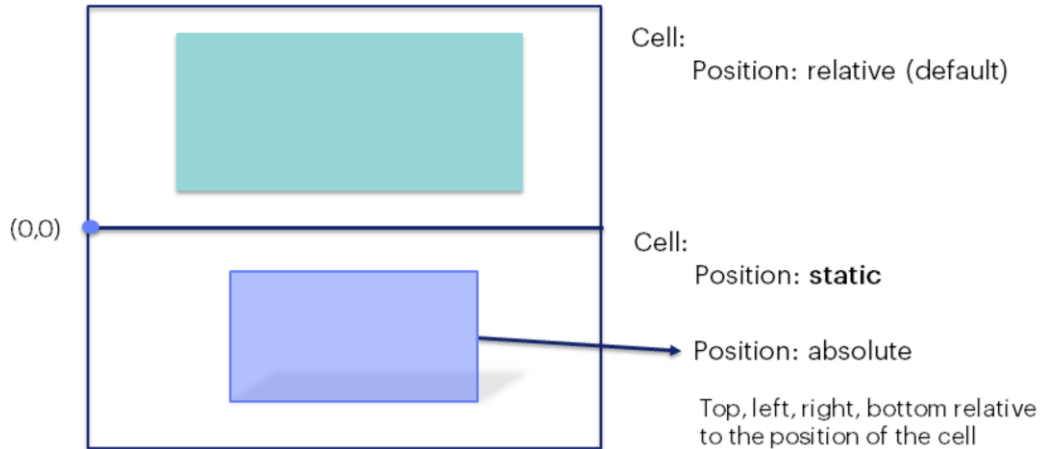


That's the reason why our table with the texts (to which we assign an absolute position) will not be positioned in relation to its container cell (which we did not modify at all, so it has a static position), but in relation to the flex, its parent, because we have changed its positioning to "relative".



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

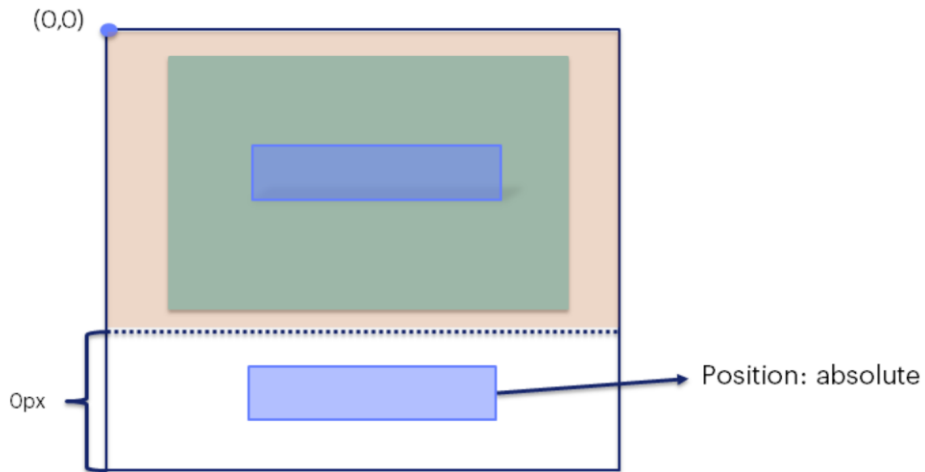
If it were not this way, it would have worked anyway, because that flex is inside a cell in a responsive table, the main table, which has a relative position by default. However, it is more orderly and safe to do it this way, which is the one we actually want because we could change the position of the flex to have it inside another control with a different positioning.

Responsive tablePosition: static (default) → Position: **relative**

If instead of using a flex control we had used a responsive table, in addition to modifying the position property of the table we would have needed to modify that for the cell because, otherwise, the absolute positioning of our control would be so in relation to the cell and not in relation to the parent.

Flex

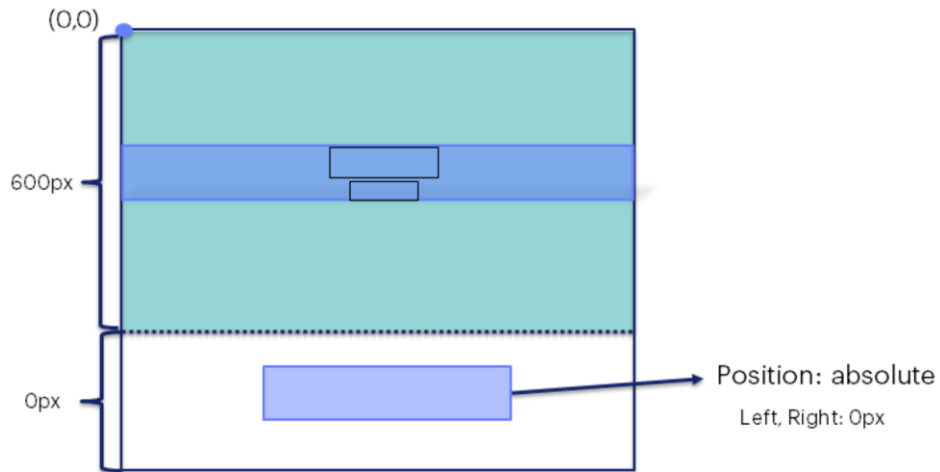
Position: static (default) → Position: relative



Getting back to our example: what we actually want is to overlap our control of cell 2 over the image, which is in cell 1, and not over the whole table. But note that when we have a table with just two cells, and with the second cell remaining empty and occupying no space because we selected an absolute positioning for its control, then the table size will be the same as the size of cell 1.

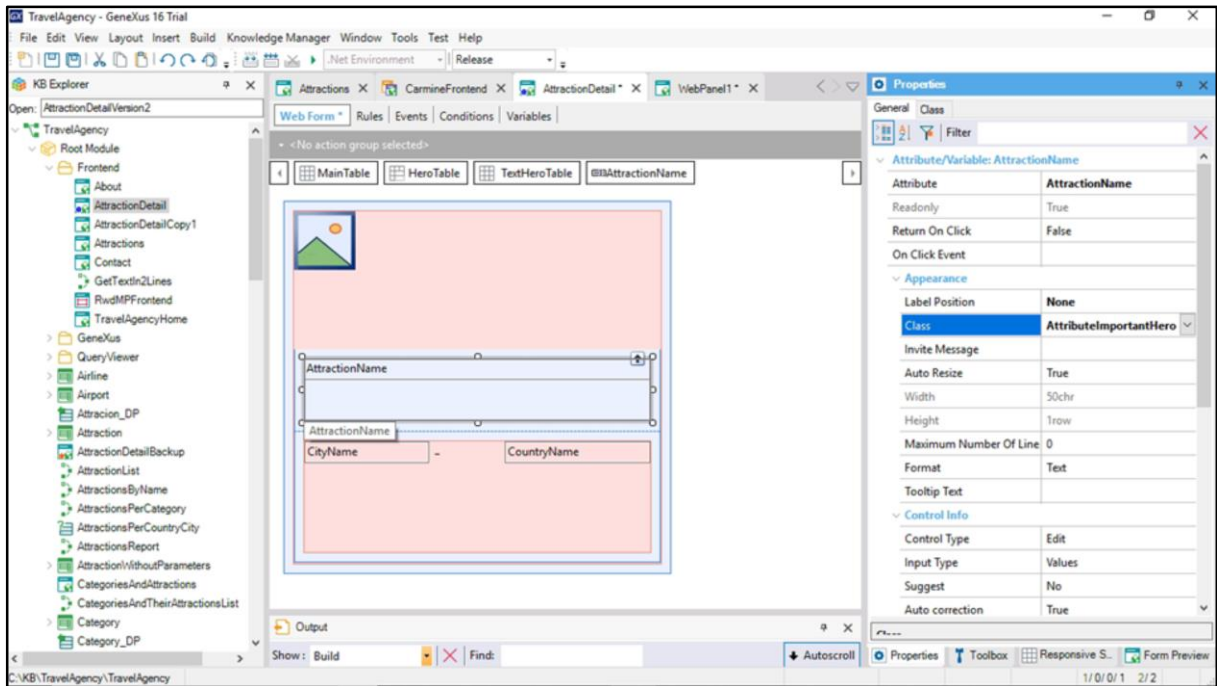
Flex

Position: static (default) → Position: relative



Cell 1 and the image match exactly in the space: we had assigned a height of 600px to the image, and the width is the same as the screen.

Therefore, when we define our control Left and Right as 0px, we are making it expand until it occupies the whole table width...

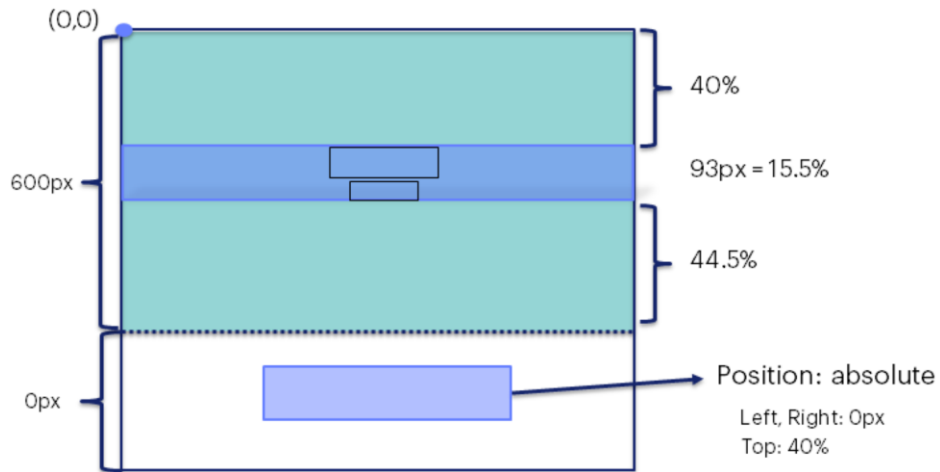


[DEMO: https://youtu.be/RleiTlSzt_k]

(note that we also centered the internal controls horizontally)...

Flex

Position: static (default) → Position: relative

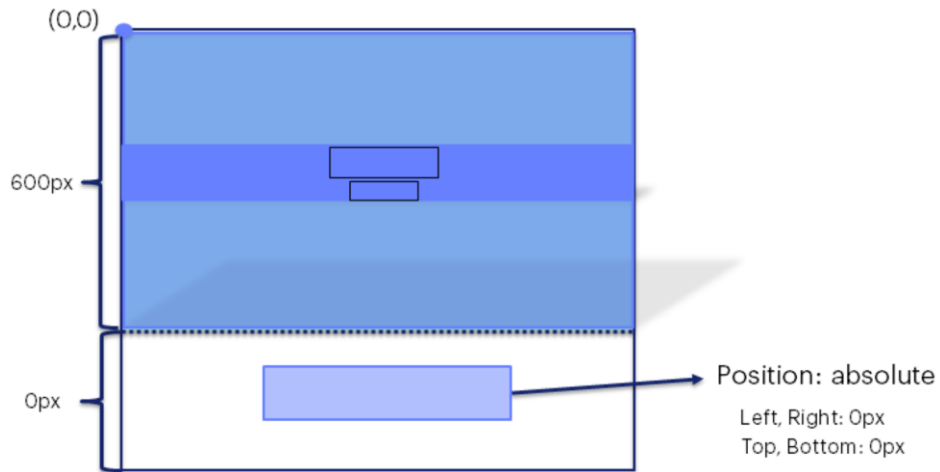


...and when we define top: 40%, we are expressing that the top border of the table will be moved 40% of the whole height, which we know is 600px, regardless of the screen size.

Why do we use 40%? Because we know that the width of the texts is approximately 93 pixels, which is 15.5% of 600. All this is too dependent on those sizes. If we change them, we will be losing the adjusted proportions and this solution will not be useful anymore.

Flex

Position: static (default) → Position: relative



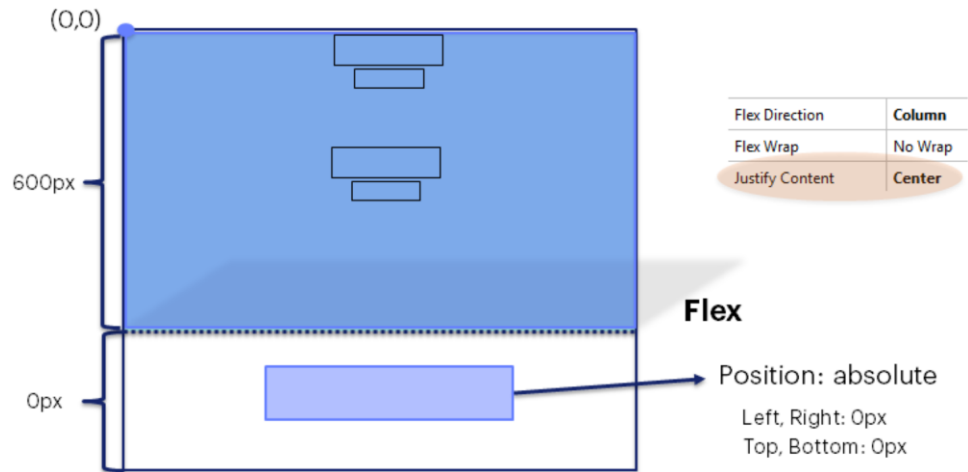
The best solution would be to have the control always centered, regardless of all sizes.

To do that, we just need to also specify the Top and Bottom positions at 0px, so that the control may expand and occupy the same space as the image, which is the space of cell 1, the space of the whole container, that is, the flex.

To also have the texts vertically centered as we see in the image...

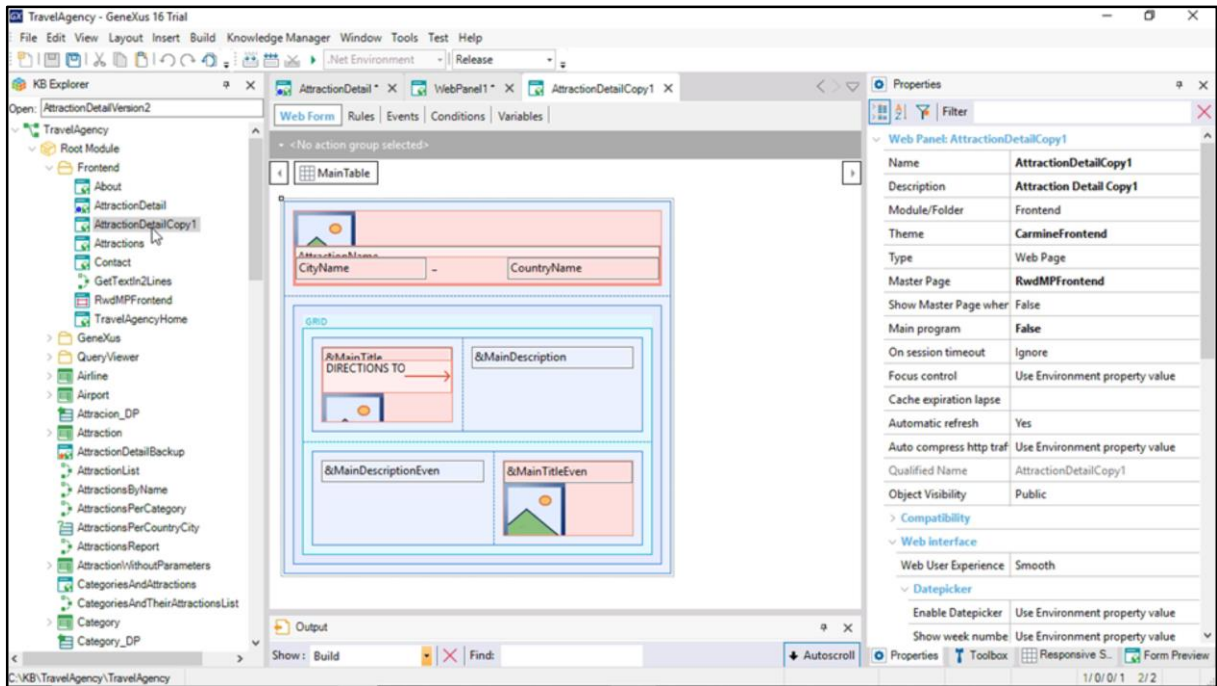
Flex

Position: static (default) → Position: relative



...instead of this way, we will also have to do something.

The best is to also use a flex for these texts instead of a table, with the direction Column, clear, and Justify content: center...

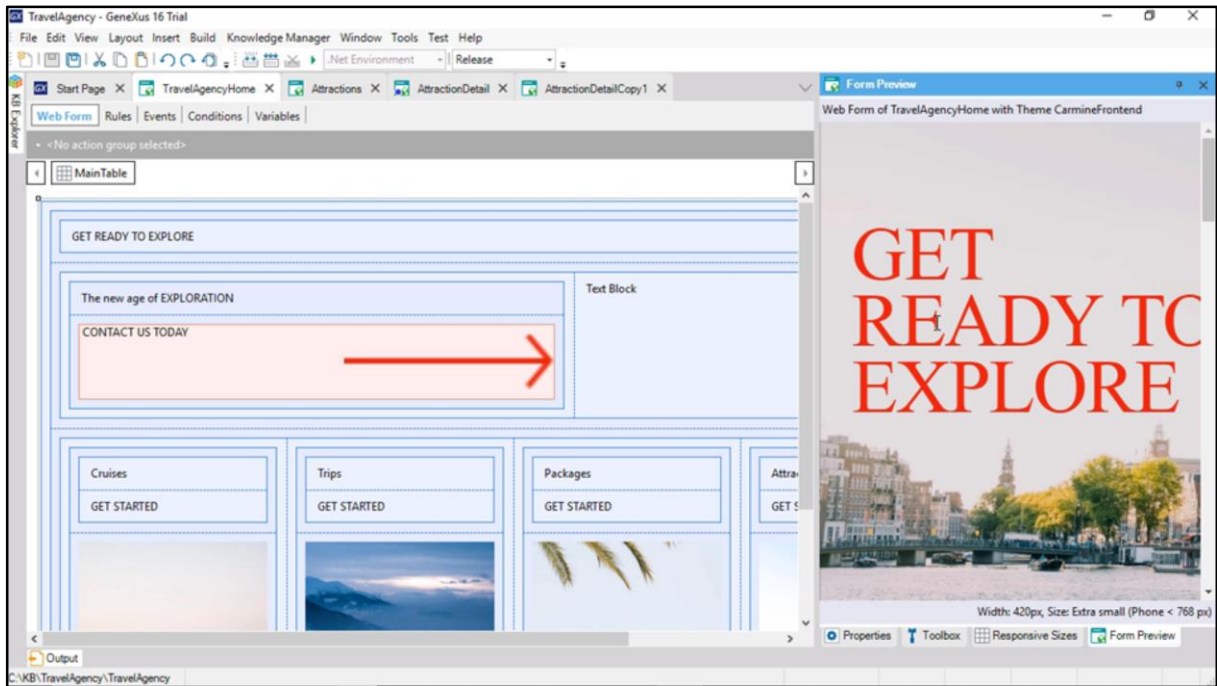


[DEMO: <https://youtu.be/urTsOwSudIQ>]

As we did in this copy of the web panel...

Here we have the best solution of all. Always, regardless of the screen size, of the image size, of the size of texts, these will be centered.

In sum



[DEMO: <https://youtu.be/tn8PlmrsMNQ>]

To sum up what we saw to this point:

To implement the overlapping of one or several texts over an image...

When we have a fixed image, one option is the one we used here, defining it as background image in the class of the table that contains the text block.

The second option we saw applies to both fixed data and to data taken from the database: it implied changing the position of the tables with the texts in a relative manner. It is relative to the position that it would occupy naturally. We saw two disadvantages, which could be solved with absolute positioning. One, which matters to us now, was the one that preserved the natural space of the control. There, we mentioned the other, simpler option for the same purpose, which we mentioned at the end, that implied margins and had no positioning.

It is the one we implemented in this copy of the home panel, where we inverted the order of appearance of the image and the texts. We associated another class to the texts, where instead of changing the position through the corresponding properties, we set up a negative top margin. A control's margin is the space surrounding it. If we assign a negative value to it, the control will move in the opposite direction as it would move naturally (with a positive margin).

If we want it to move upwards, we assign a negative top margin to it, and if we want it to move downwards, we assign a negative lower margin.

Here we had to invert the order of controls because we need the control that we are moving as we use the margin to remain on top of the other. The ones on top always remain below the ones that come after them.

We will not go into much detail, but here, the z-index property used to indicate the order of layers in the case of overlapping (which will overlap which) has no effect, because we have two sibling controls. To read more about this go to: https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Positioning/Understanding_z_index.

This solution, however, still has the problem that does not allow us to adapt to responsiveness. It is better to use positioning when we have to move a percentage in relation to a container. And that was the last solution we saw. There, we used absolute positioning, which allowed the control that would overlap another one to move according to the position of the container holding both (whose position must be relative. Note: when we assign relative position to a control, and we don't modify any of its Top, Bottom, Left or Right properties, it will logically remain exactly where it is).

This was just a small sample of the significance of classes (and also of controls, of course) in the design of our application.



Videos

training.genexus.com

Documentation

wiki.genexus.com

Certifications

training.genexus.com/certifications