blocket

FRONTEND ASSIGNMENT

We know, we know: code exercises can be dreadful at times, may feel like a school exam and are generally stressful.... But this is not how we want it to be for you! In fact the code you will produce for this exercise is meant as a chance for you to showcase your best self by presenting your code in person at the interview and sharing the rationale behind your choices.
For us, it is really a conversation starter. We will invest our time in reviewing your code prior to the interview, so we will be ready with interesting questions for what we hope will be an interesting technical conversation.

So what do you have to do?

There are two sections to this test:

- a JavaScript section, covering more programmatic questions
- an HTML/CSS section, covering standard markup problems

There are no off-the-shelf solutions; there is also no single solution to any given problem. Be creative. We don't expect you to spend hours of your time coming up with answers, and partial responses may be accepted for discussion.

The challenges get progressively harder, do as many of them as you are comfortable with. It is ok if you can't complete them all.

When you are finished with an assignment, you can either send the .zip folder (along with .git folder if any) or Send us a link to your repository.
We don't expect you to spend a lot of time on this. Not that you are not allowed to, but rather we don't want to steal your whole weekend. The goal is to have enough code to have an interesting conversation about it, and not to have it "feature complete".

JavaScript

Part 1:

Below contains a JavaScript implementation of a basic Component class, with an instantiation of it. Unfortunately, it doesn't work; list any errors or problems that are preventing it from successfully running.

```
var Component = function (config) {
                    for (property in config) {
                        this[propety] = config[property]
                    }
                }
var list = Array ( "Item 1", "Item 2", "Item 3" )
var instance = Component(id: "XF-254", list: list);
```

Part 2:

Change the code so that the Component class implements the Publish/Subscribe pattern. Clients should be able to register for notifications using simple keys, e.g. propertyChanged, user.loggedOut, or any other event that might make sense in an application.

Part 3:

Extending your answer to Part 2, suppose we now want to allow clients to receive notifications even if they registered themselves after such notification had been broadcast. Provide a simple implementation, noting any potential performance concerns.

Part 4:

Define a CustomComponent that extends Component. It should have a setValue method which broadcasts an event when it is called, specifying the old and new values.

Prepare to discuss:

1) How might your implementation change to allow for different base classes to implement the Publish/Subscribe pattern?
2) How might your implementation change to support wildcard tokens in the message key, e.g. app.*.log?

3) How might your implementation change to support a limited number of notifications for a given event?

HTML & CSS

Part1:

Create an HTML document containing a single element, that resizes with the browser window (the white element in Figure 1, below). This element is offset in the browser by 100px on the top and left. It is also offset by 100px on the right and bottom, if possible. That is, the white element resizes with the browser window, maintaining the 100px margin on all sides, until such time that the conditions in Figures 2 and 3 are met. It then maintains only the top/left margins.

This element should contain three side-by-side child elements, all of which will contain complex markup (the red, green and blue elements in Figure 1, below). These elements are vertically aligned in the middle of the variable-height white parent element.

- The red element on the left measures 100px x 200px, and is left-aligned within the parent.
- The blue element on the right measures 100px x 300px, and is right-aligned.
- The green element in the middle is 400px high, and of variable width. When the parent is resized, it should adapt, always keeping a 10px distance away from its siblings. However, it must maintain a minimum 100px width, and 10px margins, as illustrated in Figures 2 and 3

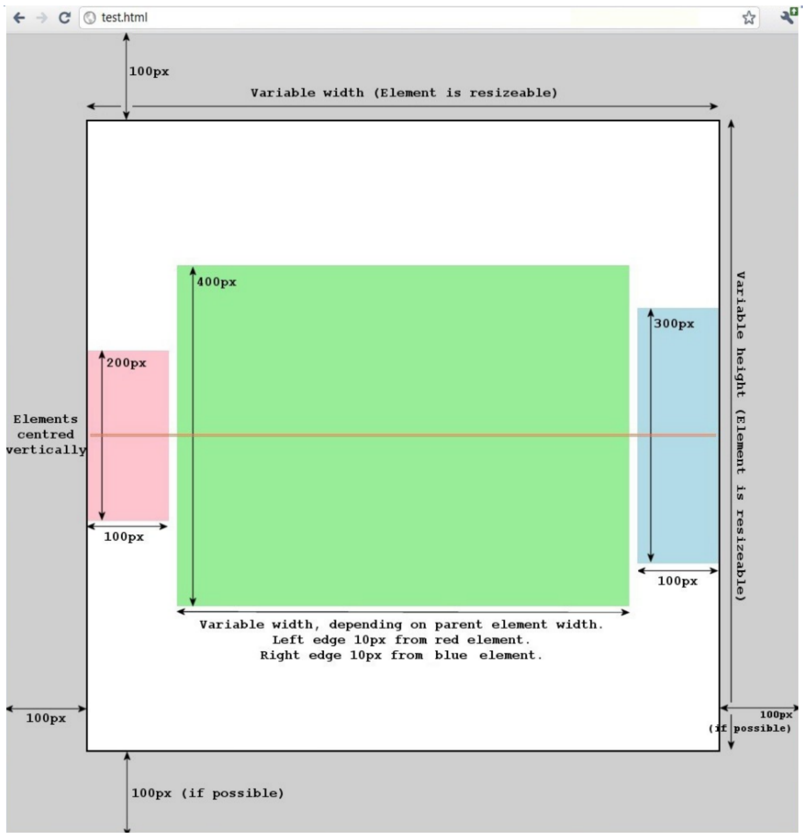
test.html
100px
Variable width (Element is resizeable)
400px
300px
200px
Elements centred vertically
100px
100px
Variable height (Element is resizeable)
Variable width, depending on parent element width.
Left edge 10px from red element.
Right edge 10px from blue element.
100px
100px (if possible)
100px (if possible)

Figure 1: An example highlighting the specification for a simple three-element layout.

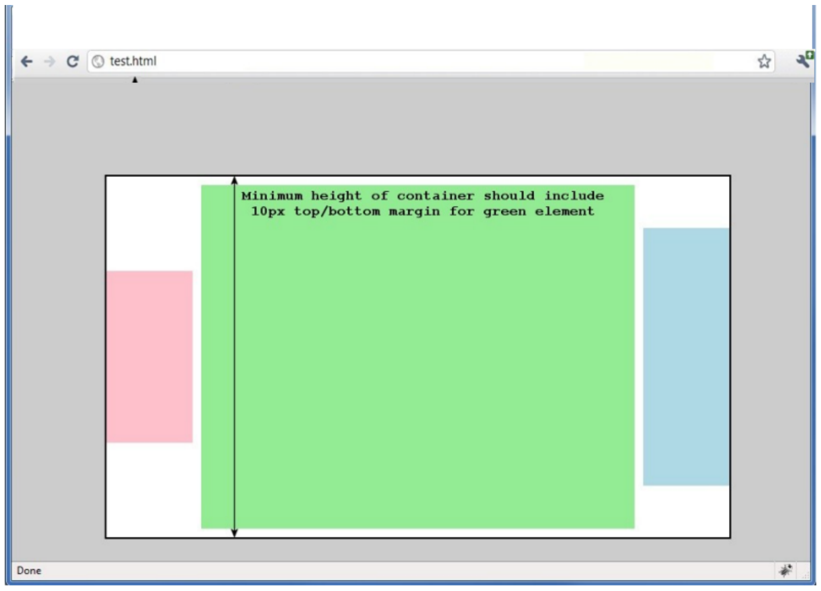
test.html
Minimum height of container should include 10px top/bottom margin for green element
Done

Figure 2: An example showing how the minimum height of the layout should behave

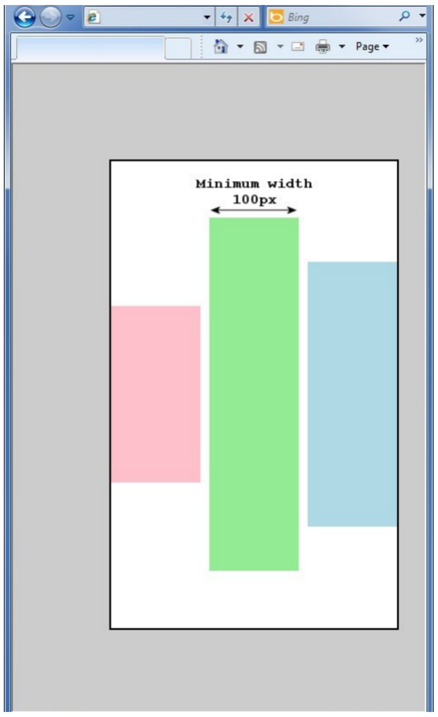

Figure 3: An example showing how the minimum width of the layout should behave.

Use only standard HTML and CSS – JavaScript, similarly. Your solution must work in all major modern browsers, Chrome, Edge and Safari.

Part 2:

Using your layout from Part 1, implement either of your choice.
Option: 1
— using HTML, CSS and/or JavaScript (as you see fit)
— the design provided in Figures 4-8, below.
Option: 2
Anything of your choice to design/build inside the greenbox, use option 1 if you can't decide for yourself.

Do not take this necessarily as a challenge; prioritize your time and focus on a suitable solution. We are not expecting pixel perfection, but instead a realistic answer to a real-world scenario.

Each row should have a hover state (see Figure 5). A custom checkbox can be 'clicked' (Figure 6), and acts independently from the rest of its row, which is also selectable (Figure 7)—that is, clicking on the row anywhere other than on the checkbox should 'select' the row. Multiple rows may be selected, regardless of the checkbox state (Figure 8).

Prepare to discuss:

- Which design details are difficult to implement
- Which design details are impossible to implement
- What compromises or changes would you make to facilitate cross-browser implementation?
- What browser-specific issues arise from the designs?
- As regards the designs themselves:
    - Do they provide you with enough information to facilitate implementation? How would you change them? How might you seek confirmation or feedback?
    - Are they good designs? That is, in your opinion, do they contain design errors? If they were to be implemented as is, would they enhance or hinder the user experience?

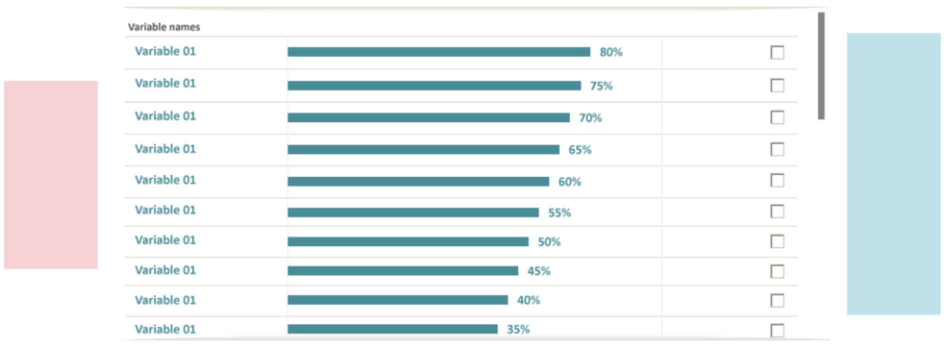

Figure 4: An overview of the final design requirements.

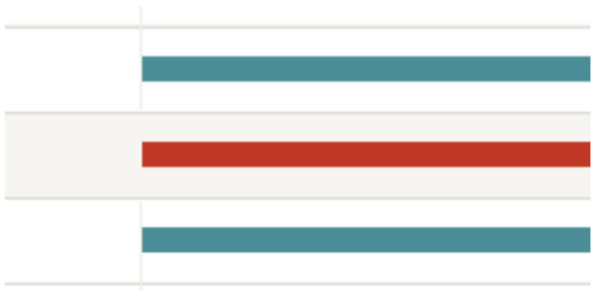

Figure 5: Rows should have a hover state

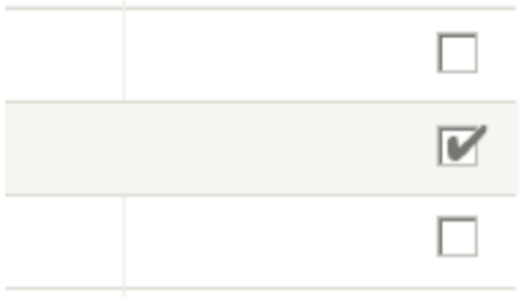

Figure 6: Custom checkboxes can be selected

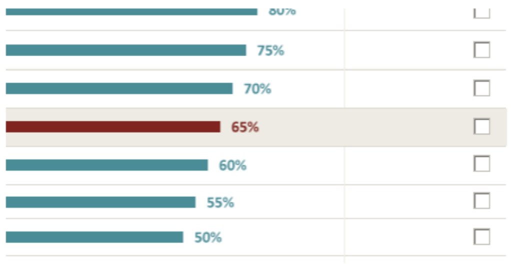

Figure 7: Rows can be selected without the checkbox.

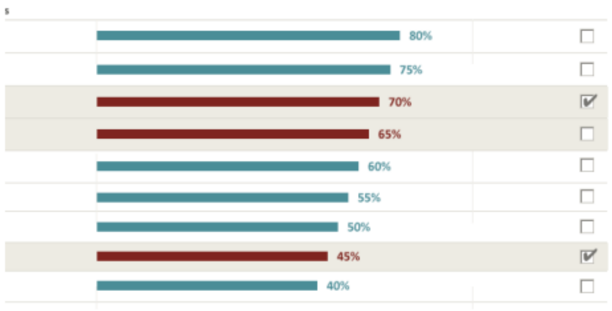

Figure 8: Multiple rows may be selected.