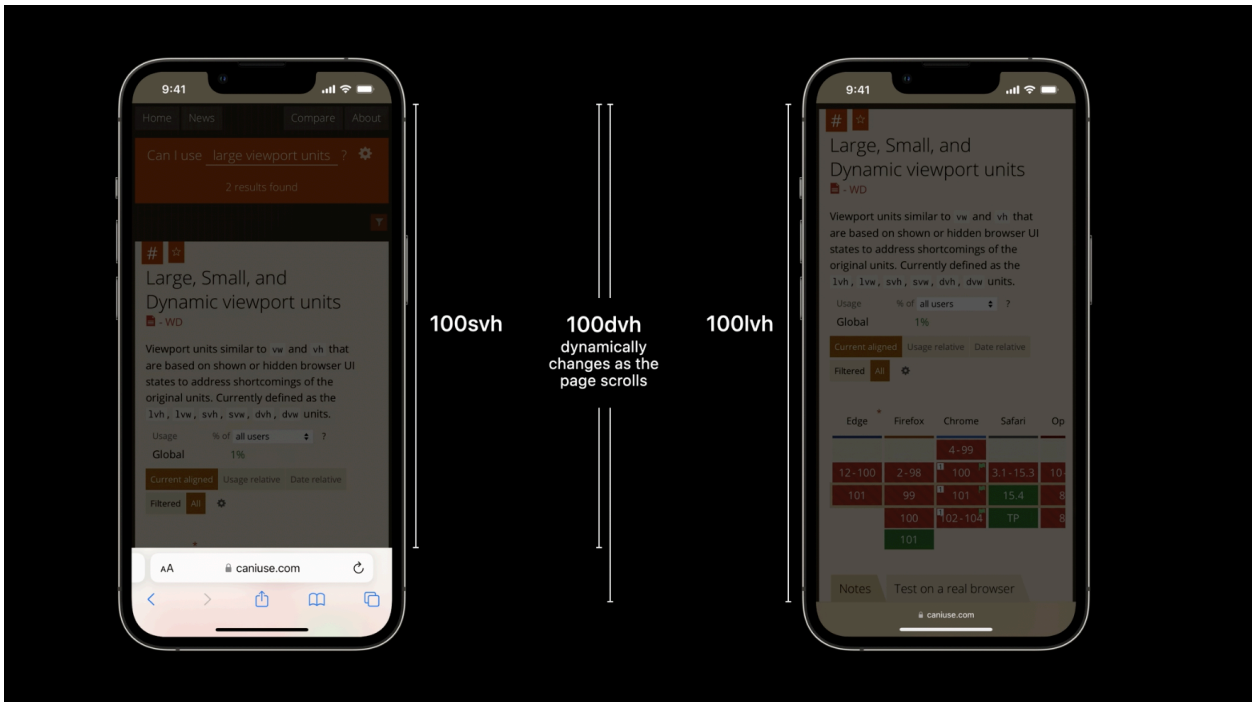
I’ve been looking into a solution where Firefox can support new viewport classes such as / svh / lvh/ dvh / svw / lvw / dvw vi / svi / lvi / dvi / vb / svb / lvb / dvb / svmin / lvmin / dvmin / svmax / lvmax / dvmax.

In iOS 15.5 WebKit was updated with the method **setMinimumViewportInset:maximumViewportInset:** which we supposedly can use to tell websites which small and large viewport is.

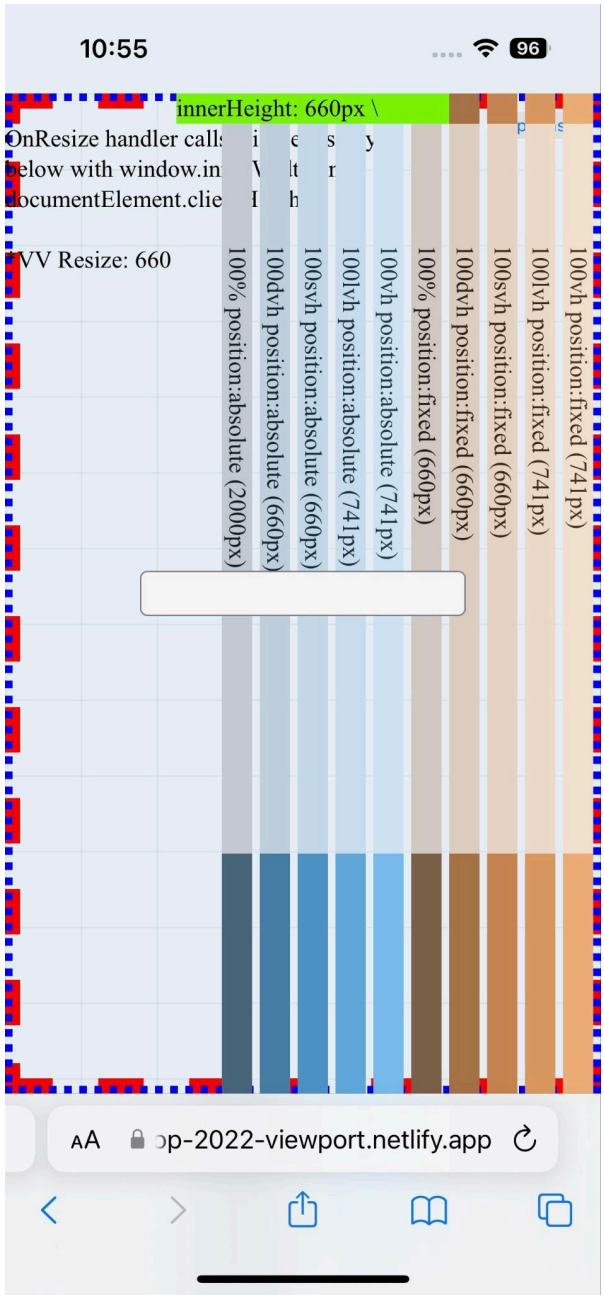
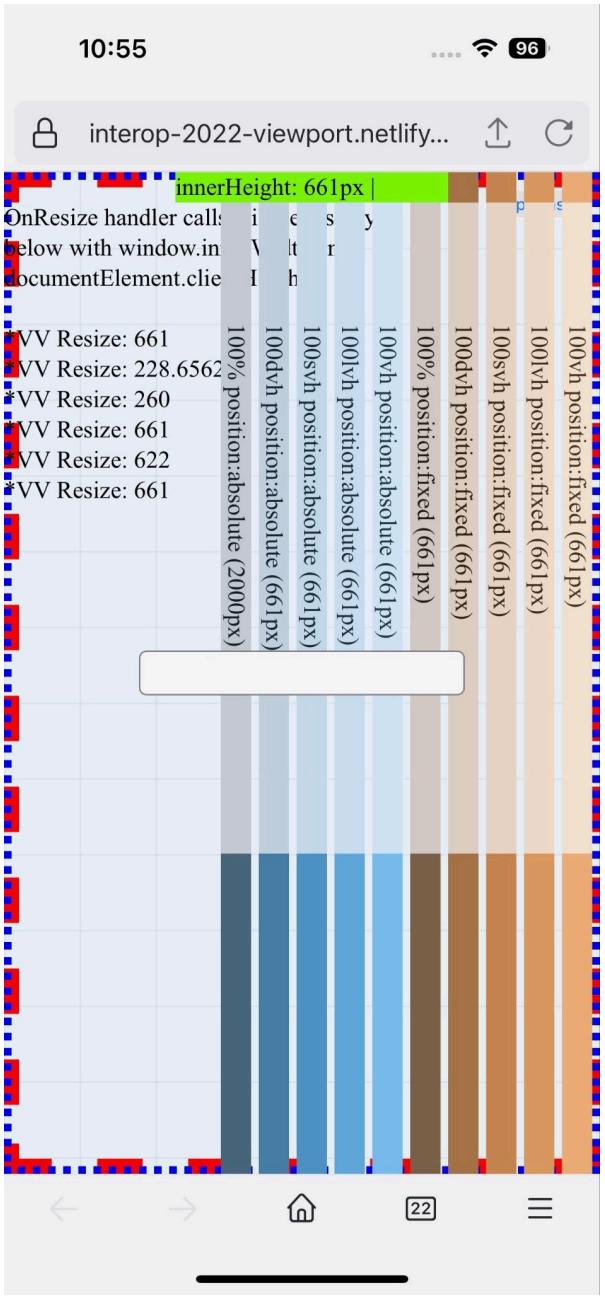
What are viewports? It’s the visible area a website can draw it’s content on.

Are there multiple viewports? When scrolling, most of the mobile browsers enlarge their widow by hiding the toolbar, or the urlbar and that offers users a larger area see more content. (check the following screenshot from the WWDC talk)

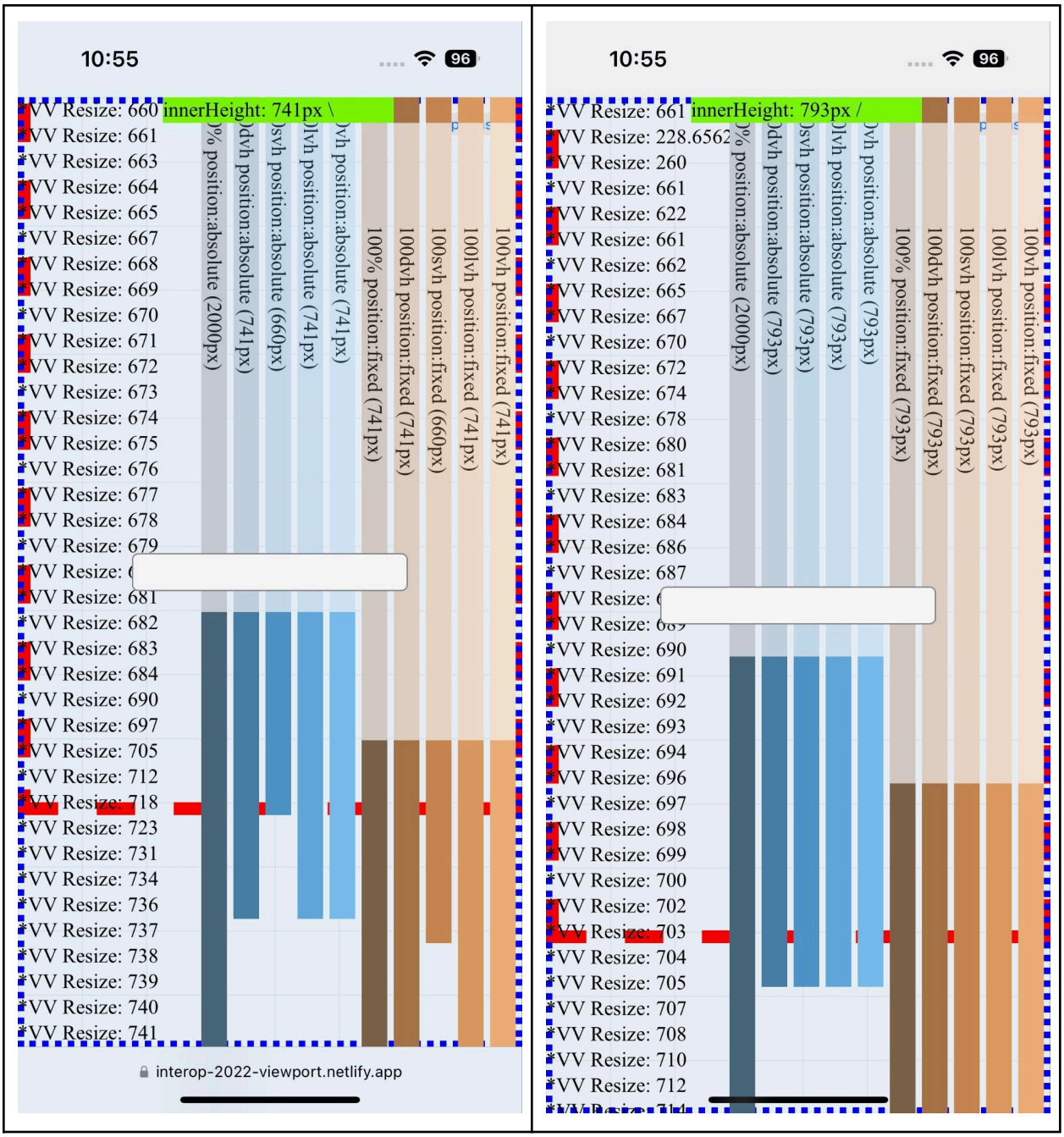


How does Firefox work now?

I’ve been using this website for testing.

Safari	Firefox
	

We can see everything looks similar when we first load the website. The difference appear when we start scrolling.

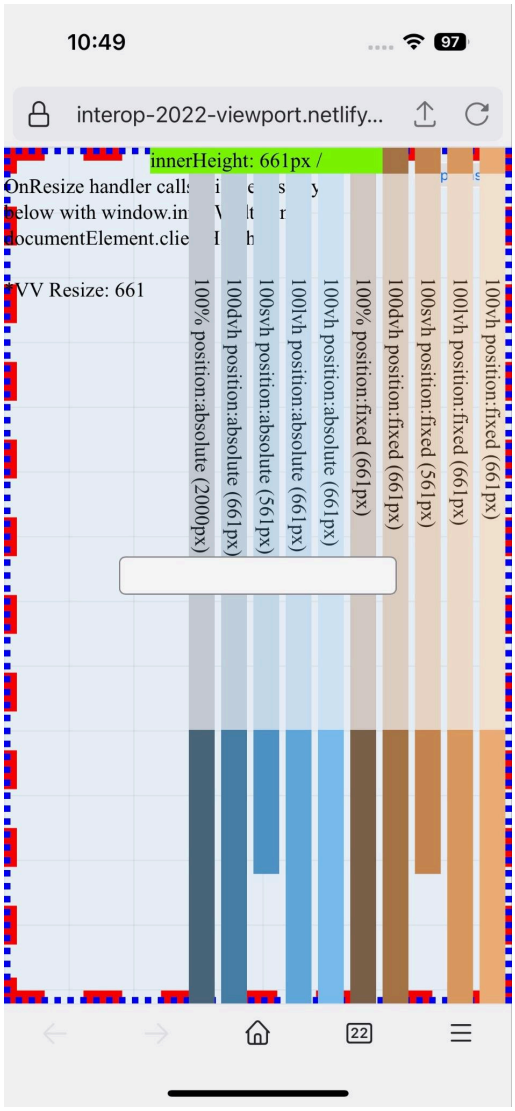


We can see on Safari, svh keeps the height of the smallest height of the viewport, while on Firefox there's no concept of a small viewport and the bar grows with the screen.

So I started with a naive way of configuring the viewport at webkit creation. So I added this piece of code.

```
let minimum = UIEdgeInsets(top: 0, left: 0, bottom: 0, right: 0)
let maximum = UIEdgeInsets(top: 52, left: 0, bottom: 80, right: 0)
if #available(iOS 15.5, *) {
    webView.setMinimumViewportInset(minimum, maximumViewportInset: maximum)
}
```

52 for the header, and 80 for the toolbar.



The current configuration appears to be implemented eagerly, leading to the need for a potential solution. One option to consider is adjusting the constraints for the WebKit container. Currently, the container's top is anchored to the bottom

of the URL bar, and the container's bottom is connected to the top toolbar. As scrolling occurs, the height of auxiliary containers decreases, while the WebKit height expands.
Alternatively, I attempted to address the issue by incorporating insets into the TabScrollingController. However, this approach proves challenging to manage due to the presence of gestures that incrementally minimize the toolbars, as well as flicks that dynamically show or hide them based on scroll velocity.