CSS for Modern Apps

Part 2 - Holy Grail Layout

Johan Steenkamp / @johanstn

Part 2: Holy Grail Layout

Part I we discussed CSS and using less specific selectors, Grid systems, and using SCSS and more specific selectors to isolate modules.

Given the break between Part 1 and today I'll recap and the tackle the Holy Grail layout.

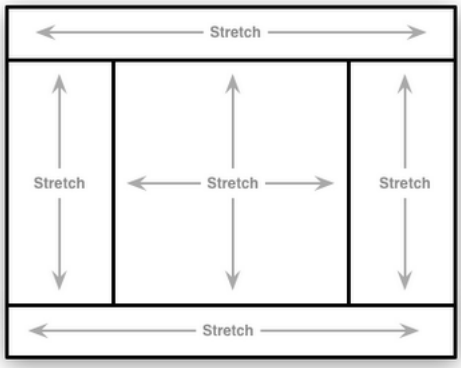
Feedback from Part I was that more detail and examples of Holy Grail layouts and SCSS would be helpful.

In Part 3 we'll pick-up on the future of CSS.

Holy Grail Layout

5 panel layout (header + 3 columns + footer)

- CSS Layout
- Solution using SCSS
- Solution using grid
- Solution using Flexbox



We are going to implement the Holy Grail layout.

The Holy Grail layout is a classic website layout problem: http://en.wikipedia.org/wiki/Holy_Grail_%28web_design%29

The requirement is a fluid layout that has a width header, and footer and a main content area comprising three columns (navigation, content, aside). The header and footer should move with the content when scrolled vertically - in other words they are not fixed in position.

We'll tackle the problem by first attempting to solve it using traditional ("old school") CSS. Then we'll update our CSS using SCSS to learn some of the simple and useful features. We also solve the problem without using a sticky footer. Finally we'll solve the problem using modern CSS and HTML by using flexbox.

The Holy Grail layout is still useful today even though designs have progressed and need to adapt to varied devices.

I recommend using Chrome and enabling the DevTools: https://developer.chrome.com/devtools

Install Pesticide to toggle outlining of elements: http://pesticide.io/

CSS

- position (static, relative, fixed, absolute)
- float (left, right)
- clear (left, right, both)
- display (block, inline, inline-block, none)
- margin (auto), padding
- box-model (box-sizing)
- .clearfix hack ({clear:both; overflow: auto;})

A quick look at one of the original Holy Grail layout articles gives you an idea of how elusive the optimum solution is: http://alistapart.com/article/holygrail

Keep in mind that web browsers and CSS has improved a lot since the article was written in 2006. Although browsers are better requirements are more complex with mobile devices, screen sizes and orientations to think about. So you still need a solid foundation by really understanding of basics of CSS layout.

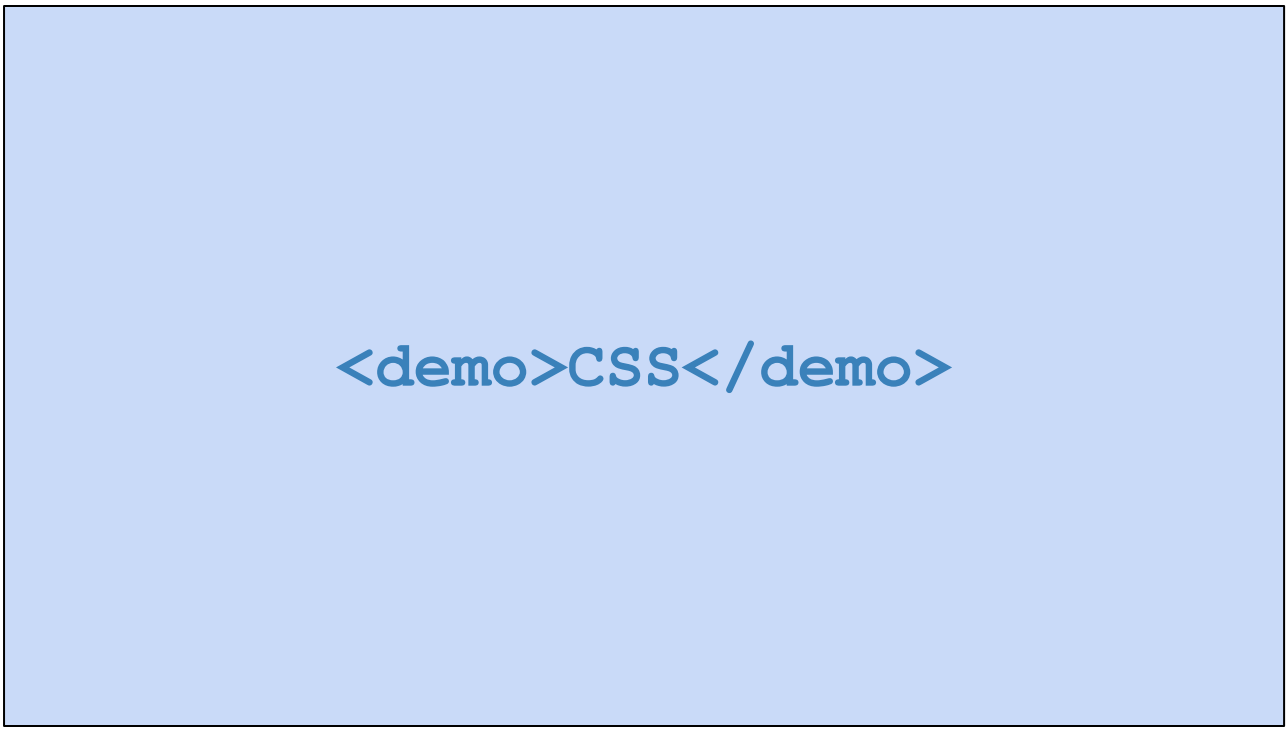
First read these articles and tutorial:

1. CSS Positioning 101: http://alistapart.com/article/css-positioning-101
   Take note of how relative positioning works.
2. CSS Floats 101: http://alistapart.com/article/css-floats-101
   This article explains why parent elements containing floated elements do not expand to completely surround them.
3. Review the excellent Learn Layout step-by-step tutorial: http://learnlayout.com/
   This tutorial gives you everything you need to tackle layouts and also has a section on Flexbox.

I recommend learning by experimenting with positioning and floating using the project files in the `/css-position-float` folder.

In case you were wondering, using HTML tables is not a valid solution. Using HTML

tables will not help you understand CSS layouts.



Files: /holy-grail-layout/css-basic

Things to try:

- Remove spacing between the layout containers
- Set widths for columns
- Try fixed width columns and calc(100% - px)
- Absolute position footer
- Position footer using the .clearfix hack

You should be able to get close. Make sure to test your solution with varying amounts of content in all the layout areas.

Resize your browser to see what happens when scrolling is activated.

The most common problem that remains after getting the layout working is that the columns do not extend to equal and full page height.

Tip: Always fix problems as early and do not move on until you understand why your solution works. If you don't your problems compound and combinations of errors are even harder to fix. Before long you have no idea what is causing an issue and start

“hacking” and this just makes things worse.

Tip: Keep in mind there are many ways to solve the problem - there is not a single “right” way. Experimenting and knowing how different solutions work will give you the understanding of the trade-offs of each and knowing which is best for a given application.

Solution:

- Float the containers to remove the spacing that is caused by the default margin of <p> tags. Floating ensures the <p> tags are enclosed (in the flow) of the containers. Alternatively you can apply the .clearfix hack to each container, however this will not work when you want to align the central columns horizontally (why?).
- Set the central columns to 20%, 50%, 30% width, they should now form a single row below the header and the footer aligned with the bottom of the tallest column.
- Try to position the footer at the bottom of the page. For example use absolute positioning. Resize your browser to force vertical scrolling to test your solution. Can you think of other ways to solve this problem? We’ll see how with the next by creating a “sticky” footer.

Bonus:

- Create a center, fixed-width layout

SCSS

- @import
- $variables
- @mixin
- @extend
- %placeholders
- Interpolation #{%placeholder}
- Source maps

Files /holy-grail-layout/css-tables

Solution: The best solution is to use tables! But NOT HTML tables - CSS tables: http://colintoh.com/blog/display-table-anti-hero

Using SCSS we can be DRY (Don't Repeat Yourself) and modularize our CSS

Read the SASS Basics section http://sass-lang.com/guide

Note: SCSS is the newer version of Sass - a SCSS file is a valid CSS file which is not the case with Sass.

Getting started with SCSS is easy IF you have the right tools installed. Popular Web IDEs support SCSS via plug-ins and this will allow you to write SCSS and have the IDE pre-process it and output CSS for the web browser.

variables: $my-color: red;
mixin: @mixin border-radius($radius) -> @include border-radius(10px)
extend: .column { … } -> @extend .column
placeholder: %column{ … } -> @extend %column

Source maps: Typically an SCSS pre-processor will also output a source map, if not there should be a configuration setting. This is used by your web browser dedugging

tools (Chrome: DevTools) to map the output CSS code to the corresponding SCSS code. This helps during debugging. Without a source map you would have to figure out where the CSS you are examining was generated in the SCSS. This is complicated if you are using imported SCSS files and other features like mixins, extends and computed styles (iterators, loops etc.).

You do not need to write you own mixins. There are solutions like this one for Flexbox:

https://github.com/mastastealth/sass-flex-mixin

You can also use a task runner like Gulp to process your SCSS and automatically add browser specific extensions using plug-ins like autoprefixer:

https://github.com/postcss/autoprefixer

<demo>SCSS</demo>

Files: /holy-grail-layout/css-scss

Try variables, mixins, extends and placeholders

Why is extending using a placeholder preferable to extending a class?

Flexbox

- parent (container) and children (items)
- container (row, column, wrap, spacing)
- items (order, grow, shrink, basis)
- flex shorthand (flex: fg, fs, fb)

Files: /holy-grail-layout/flex-box

Flexbox syntax changed significantly during development so watch out for tutorials and articles that are out of date.

Recent Flexbox guide: https://scotch.io/tutorials/a-visual-guide-to-css3-flexbox-properties

Flex guide: https://css-tricks.com/snippets/css/a-guide-to-flexbox/

Flex shorthand: https://css-tricks.com/almanac/properties/f/flex/

MDN reference: https://developer.mozilla.org/en-US/docs/Web/Guide/CSS/Flexible_boxes

Holy Grail using flexbox (columns only): http://appendto.com/2014/03/solving-the-holy-grail-layout-2/

Selection of layout solutions: http://www.cssplay.co.uk/layouts/

Files: /holy-grail-layout/flex-box

Did you notice that the markup order (article, nav, aside) is different to the rendered order (nav, article, aside). How does that work?

Try the flex short hand notation.

Notice how the footer content has lost it's padding at the bottom. Can you fix it?

Why do some styles have browser specific prefixes but not others? For example:

-webkit-flex: 1;
-moz-flex: 1;
-ms-flex: 1;
flex: 1;

Can you make the layout responsive? Tip: you need a media query, change the flex direction and maybe setting order. Some clues in the Flex guide from CSS Tricks. (Watch out the example layout they show does not work correctly if you add lots of content in the containers.)

Compare this solution with the CSS tables solution. Which is better? Why?

Next...

Part 3 - Modular CSS

Johan Steenkamp / @johanstn

Next... Part 3 Modular CSS