Mobile First, Responsive Design and SMACCS

Building a maintainable website suitable to work on every device

Cesar Jimenez

Instructor / Developer

Responsive Web Design

Responsive web design is the practice of building a website suitable to work on every device and every screen size, no matter how large or small, mobile or desktop.

Mobile First

Mobile generally means to build a separate website commonly solely for mobile users.

And to do this mobile website first, even before creating the larger design for desktop.

Responsive Web Design vs. Mobile First

Mobile websites can be extremely light but they do come with the dependencies of a new code base and figuring out what browser is being used by the users and write code for it.

Best Practice

Why choose?

Just use a little bit of both.

Favor designs that dynamically adapts to different browser and device viewports, changing layout and content along the way.

But design the mobile look first.

Responsive Web Design

Responsive web design is broken down into three main components:

1. flexible layouts
2. media queries
3. flexible media.

Flexible Layout

Building the layout of a website with a flexible grid, capable of dynamically resizing to any width.

Flexible grids are built using relative length units, like percentages or em/rem units.

These relative lengths are then used to declare common grid property values such as width, margin, or padding.

Flexible Layout

We use relative lengths like percentages instead of fixed measurement units like pixels because the height and width continually change from device to device.

Website layouts need to adapt to this change and fixed values have too many constraints.

Flexible Layout Proportion Layout

Target width of an element and dividing it by the width of it's parent element.

target ÷ context = result

The result is the relative width of the target element.

FLEXIBLE GRID

DEMO

Flexible Layout

At times the width of a browser viewport may be so small that even scaling the the layout proportionally will create columns that are too small to effectively display content.

Specifically, when the layout gets too small, or too large, text may become illegible and the layout may begin to break.

Media Queries

Media queries provide the ability to specify different styles for individual browser and device circumstances, like the width of the viewport or device orientation

Apply uniquely targeted styles opens up a world of opportunity and leverage to responsive web design.

Media Queries

A media query may include a media type followed by one or more expressions.

Media types include all, screen, print, tv, and braille.

The default is screen.

Logical Operators in Media Queries

There are three different logical operators available for use within media queries:

1. and
2. not
3. only.

Logical Operators in Media Queries

Using and allows us to ensure multiple conditions are met before applying our styles.

```
@media all and (min-width: 800px) and (max-width: 1024px) {...}
```

Logical Operators in Media Queries

The not logical operator negates the query, specifying any query but the one identified.

```
@media not screen and (color) {...}
```

In the example above the expression applies to any device that does not have a color screen.

Black and white or monochrome screens would apply here for example.

Logical Operators in Media Queries

The only logical operator only

```
@media only screen and (orientation: portrait) {...}
```

the expression selects only screens in a portrait orientation.

Media Features in Media Queries

Media features identify what attributes or properties will be targeted within the media query expression.

Width Media Features in Media Queries

The height and width features are based off the height and width of the viewport rendering area, the browser window for example.

Within responsive design the most commonly used features include min-width and max-width.

```
@media all and (min-width: 320px) and (max-width: 780px) {...}
```

MEDIA QUERY

DEMO

Identifying Breakpoints

Your instinct might be to write media query breakpoints around common viewport sizes as determined by different device resolutions, such as 320px, 480px, 768px, 1024px, 1224px, and so forth.

This is a bad idea.

Identifying Breakpoints

When building a responsive website it should adjust to an array of different viewport sizes, regardless of the device.

Breakpoints should only be introduced when a website starts to break, look weird, or the experience is being hampered.

Mobile First

The mobile first approach includes using styles targeted at smaller viewports as the default styles for a website, then use media queries to add styles as the viewport grows.

Mobile First

The mobile first approach advocates designing with the constraints of a mobile user in mind. Before too long, the majority of Internet consumption will be done on a mobile device. Plan for them accordingly and develop intrinsic mobile experiences.

Mobile First Media Query

A breakout of mobile first media queries might look at bit like the following.

```
/* Default styles first then media queries */
@media screen and (min-width: 400px)  {...}
@media screen and (min-width: 600px)  {...}
@media screen and (min-width: 1000px) {...}
@media screen and (min-width: 1400px) {...}
```

Mobile First Media Query

Generally speaking, avoiding CSS3 shadows, gradients, transforms, and animations within mobile styles isn't a bad idea either.

MOBILE FIRST

DEMO

Flexible Media

Media like images, videos, and other types need to be scalable, changing their size as the size of the browser changes.

Flexible Media

One quick way to make media scalable is by using the max-width property with a value of 100%.

```
img, video, canvas {
  max-width: 100%;
}
```

SMACCS

Stands for
Scalable and Modular Architecture
for CSS.

Attempts to document a consistent approach to site development when using CSS.

It also attempts to describe best practices for CSS.

SMACCS

At it's very core its just categorization.

There are five types of categories:

1. Base
2. Layout
3. Module
4. State
5. Theme

Reason for categorization

Much of the purpose of categorizing things is to codify patterns—things that repeat themselves within our design.

Reason for categorization

Repetition results in less code, easier maintenance, and greater consistency in the user experience.

These are all wins.

Base Rules

Base rules are the defaults

Essentially, a base style says that wherever this element is on the page, it should look like this.

Base Rules

Defines the default styling for how that element should look in all occurrences on the page.

Example Base Styles

```
body, form {
    margin: 0;
    padding: 0;
}

a {
    color: #039;
}

a:hover {
    color: #03F;
}
```

CSS Resets

A CSS Reset is a set of Base styles designed to strip out—or reset—the default margin, padding, and other properties.

Its purpose is to define a consistent foundation across browsers to build the site on.

Layout Rules

By its very nature, CSS is used to lay elements out on the page.

We will create a distinction however between layouts components.

So we will create this distinction by calling out major and minor components.

Layout Rules

By its very nature, CSS is used to lay elements out on the page.

We will create a distinction however between layouts components.

So we will create this distinction by calling out major and minor components.

Layout Rules

Major components are the header, footer, large section etc.

Minor components would be callout, or login form, or a navigation item

Layout Rules

We will call Major Components the Layout Styles.

And we will call the Minor Components Modules.

Layout Rules

We will call Major Components the Layout Styles.

And we will call the Minor Components Modules.

Naming Convention for Layout Rules

SMACCS suggest to use a prefix to differentiate between Layout, State, and Module rules. For Layout, use l- or layout-, both would work just as well.

Layout Rules

Generally, a Layout style only has a single selector: a single ID or class name.

Use of a higher level Layout style affecting other Layout styles.

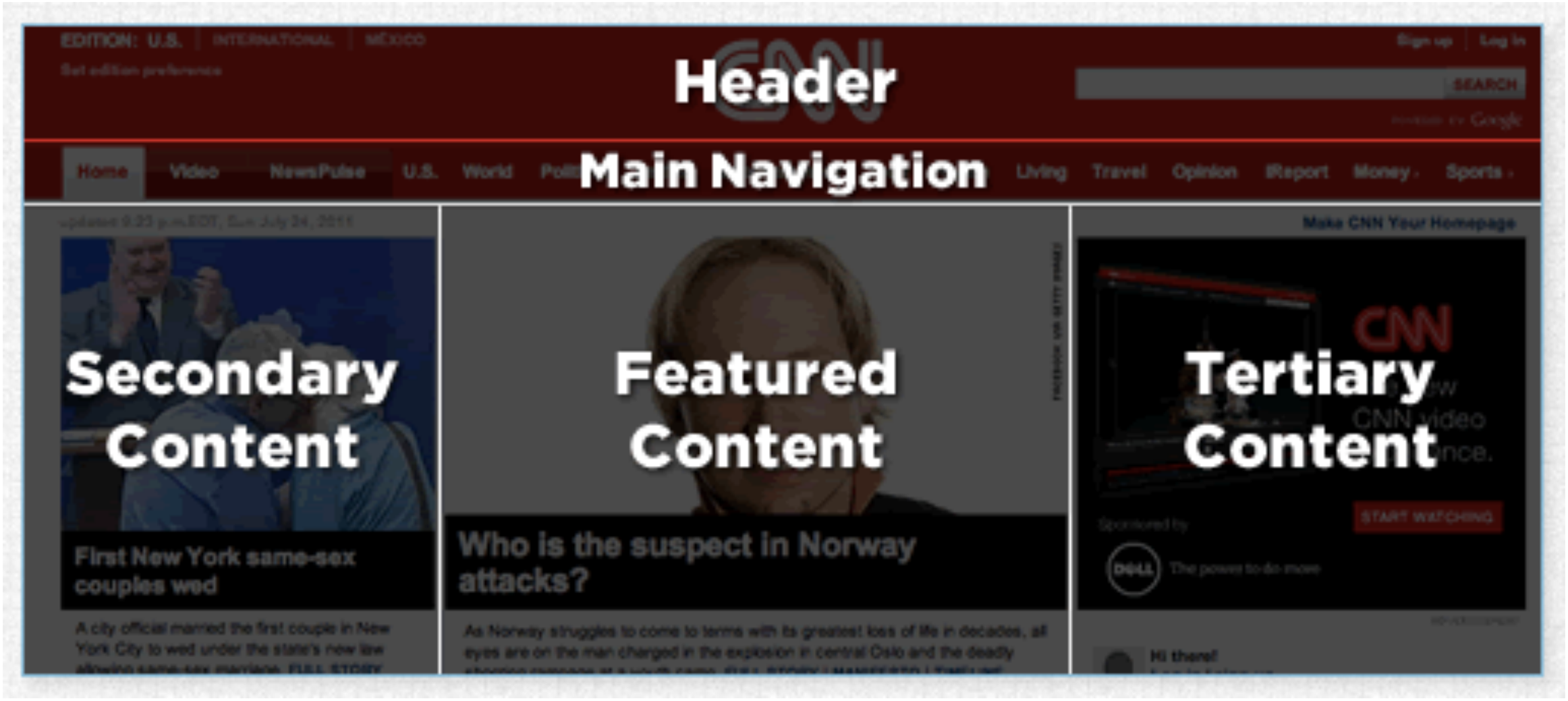
```
#article {
    float: left;
}

#sidebar {
    float: right;
}

.l-flipped #article {
    float: right;
}

.l-flipped #sidebar {
    float: left;
}
```

Layout Examples



Module Rules

Module is a more discrete component of the page.

It is your navigation bars and your carousels and your dialogs and your widgets and so on.

Modules sit inside Layout components

Each Module should be designed to exist as a standalone component.

Module Rules

When defining the rule set for a module, avoid using IDs and element selectors, sticking only to class names.

Module Rules

A module will likely contain a number of elements and there is likely to be a desire to use descendent or child selectors to target those elements.

Module example

```
.module > h2 {
    padding: 5px;
}

.module span {
    padding: 5px;
}
```

Module Rules - Avoid element selectors

Using .module span is great if a span will predictably be used and styled the same way every time while within that module.

Styling with generic element

```
<div class="fld">
    <span>Folder Name</span>
</div>

/* The Folder Module */
.fld > span {
    padding-left: 20px;
    background: url(icon.png);
}
```

Module Rules - Avoid element selectors

The problem is that as a project grows in complexity, the more likely that you will need to expand a component's functionality

Styling with generic element

```
<div class="fld">
    <span>Folder Name</span>
    <span>(32 items)</span>
</div>
```

Module Rules - Avoid element selectors

Only include a selector that includes semantics. A span or div holds none. A heading has some. A class defined on an element has plenty.

Styling with generic element

```
<div class="fld">
    <span class="fld-name">Folder Name</span>
    <span class="fld-items">(32 items)</span>
</div>
```

State Rules

State rules are ways to describe how our modules or layouts will look when in a particular state.

Is it hidden or expanded?

Is it active or inactive?

They are about describing how a module or layout looks on screens that are smaller or bigger.

State Rules Example

Example:

an accordion section may be in a collapsed or expanded state.

A message may be in a success or error state.

State Rules

States are generally applied to the same element as a layout rule or applied to the same element as a base module class.

State applied to an element

```
<div id="header" class="is-collapsed">
    <form>
        <div class="msg is-error">
            There is an error!
        </div>
        <label for="searchbox" class="is-hidden">Search</label>
        <input type="search" id="searchbox">
    </form>
</div>
```

Difference between Module && State

State styles indicate a JavaScript dependency.

Combining State Rules with Modules

Sometimes a state is very specific to a particular module where styling is very unique.

In a case where a state rule is made for a specific module, the state class name should include the module name in it.

State rules for modules

```
.tab {
    background-color: purple;
    color: white;
}

.is-tab-active {
    background-color: white;
    color: black;
}
```

Theme Rules

Theme rules are similar to state rules in that they describe how modules or layouts might look.

Most sites don’t require a layer of theming but it is good to be aware of it.

Theme Rules

Theme defines colors and images that give your application or site its look and feel.

Separating the theme out into its own set of styles allows for those styles to be easily redefined for alternate themes.

Theme Rules

Themes can affect any of the primary types. It could override base styles like default link colors.

It could change module elements such as chrome colors and borders.

It could affect layout with different arrangements. It could also alter how states look.

Theme Rules

Let's say you have a dialog module that needs to have a border color of blue, the border itself would be initially defined in the module and then the theme defines the color:

Module Theming

```
// in module-name.css
.mod {
    border: 1px solid;
}

// in theme.css
.mod {
    border-color: blue;
}
```

Typography

Similar to themes, there are times when you need to redefine the fonts that are being used

As a result, we create separate font files for each locale that redefine the font size for those components.

Typography

Font rules will normally affect base, module and state styles.

Font styles won't normally be specified at the layout level as layouts are intended for positioning and placement, not for stylistic changes like fonts and colors.

Your site should only have 3 to 6 different font-sizes. If you have more than 6 font sizes declared in your project, your users will not likely notice and you are making the site harder to maintain.

Thank you!

Cesar Jimenez
Instructor/Developer
Code Fellows
@cesar_r_jimenez