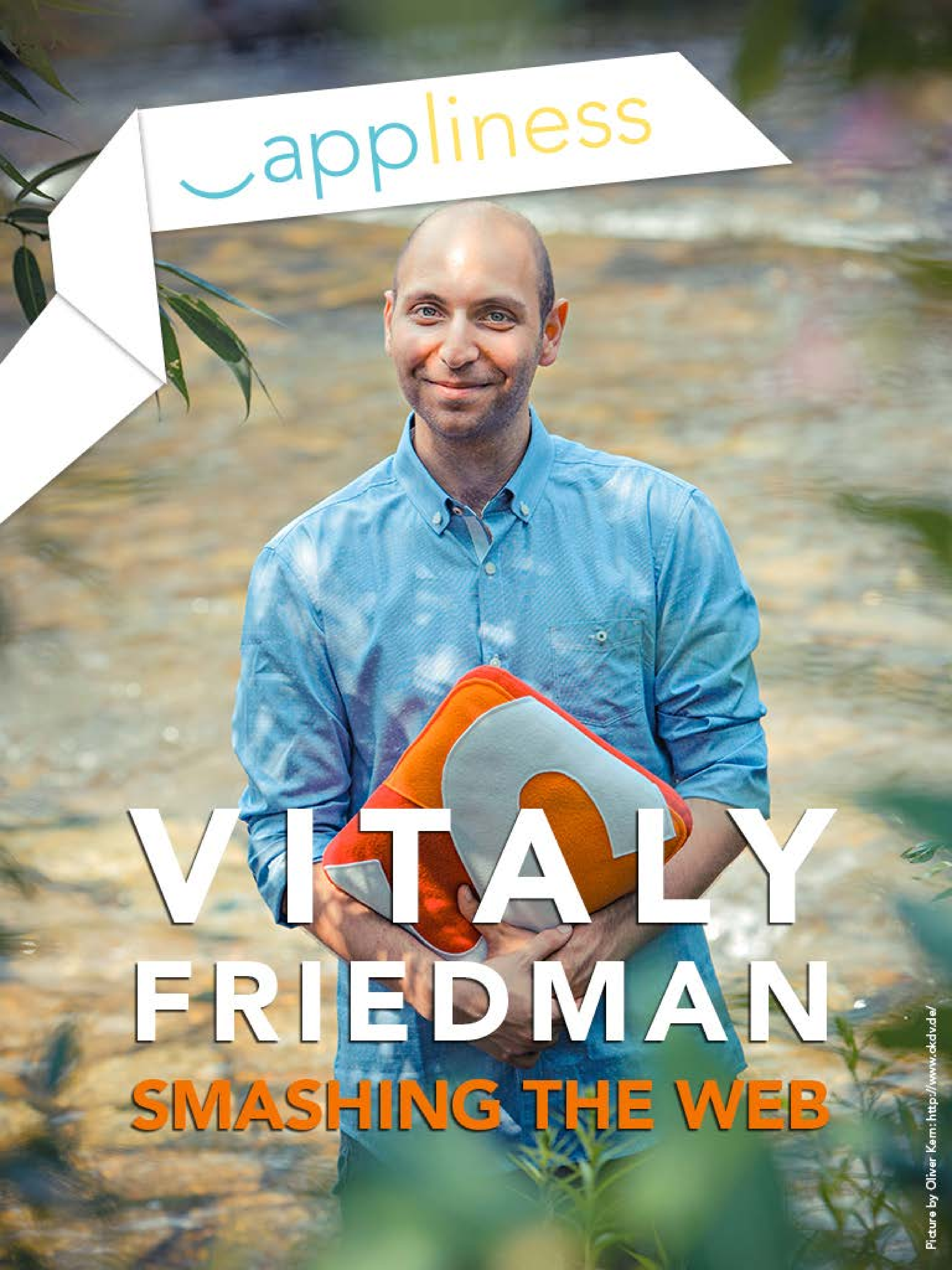
appliness


VITALY
FRIEDMAN
SMASHING THE WEB
Picture by Oliver Kern: http://www.ckdv.de/

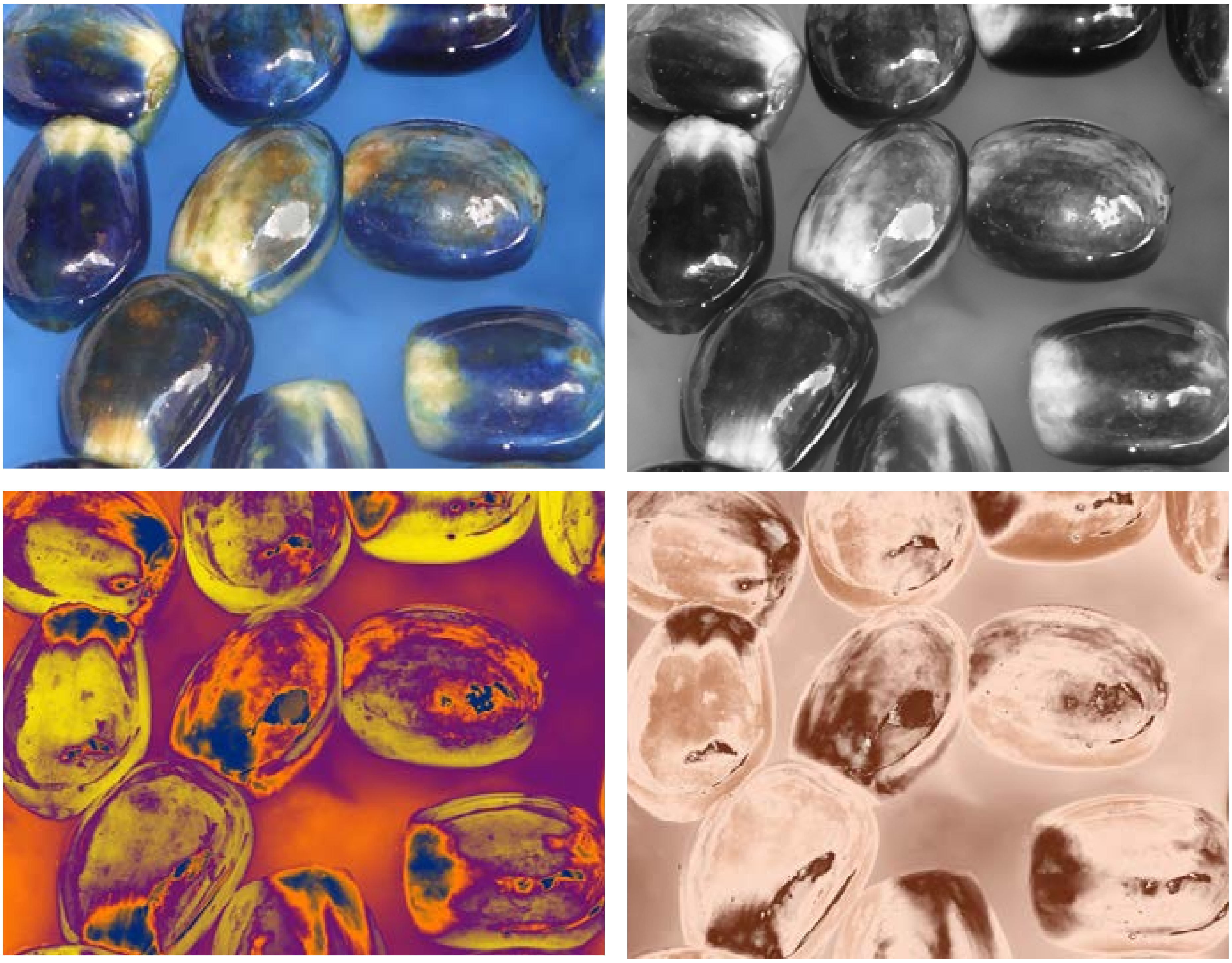
Gradient Maps for the Web

by Alan Greenblatt

by Alan Greenblatt

"What if you could manipulate HTML images using gradient map adjustment layers."

In Photoshop, you can manipulate images in all kinds of ways using gradient map adjustment layers. You can tint or tone images, easily convert them to black and white, adjust the midtones, or remap all the colors of an image into your own custom colormap for some great creative effects.



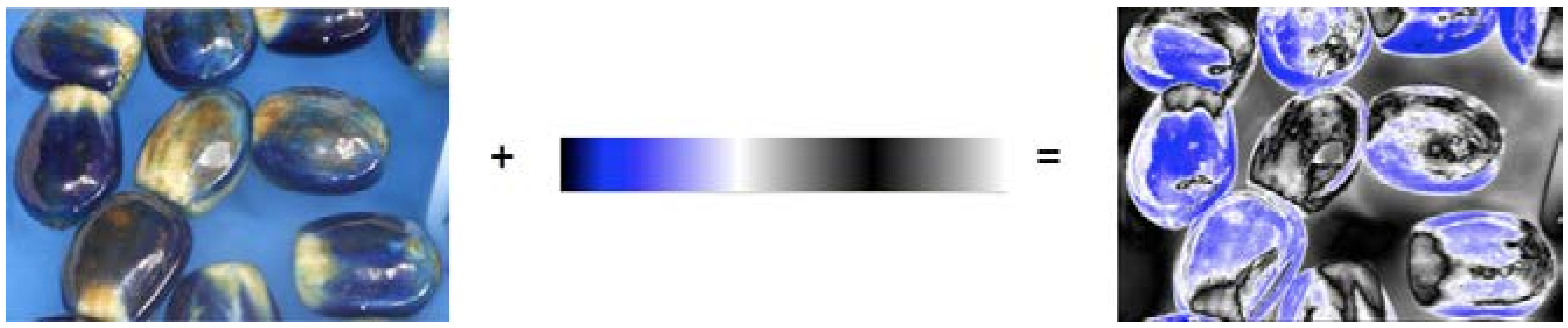
Now what if you could do that to HTML content as well?

Not simply images. I mean ***any*** HTML content: text, video, images, you name it.

gradientmaps.js, a new open source library recently released on GitHub lets you do just that, very simply. The library depends on the ability to apply SVG filters to HTML content. That support is currently present only in Chrome and Firefox, but hopefully the other browsers should be following suit with support in the near term. I wrote a support matrix that details which browsers support SVG filters on HTML content, which will help you know which browsers this library can be used on.

WHAT ARE GRADIENT MAPS?

The best way to understand gradient maps is to think in terms of images and linear gradients. The darker parts of the image get mapped to the left side of the gradient, while the brighter parts get mapped to the right side of the gradient.



But remember, we're not just talking about images, we're talking about HTML content. The image in this case is what actually gets rendered on the screen, be it text, images, video, etc.

GRADIENTMAPS.JS EXAMPLE

Let's look at an example so you'll see what I'm talking about. Make sure you are using Chrome or Firefox to view this, so you can see the gradient maps in action. Be sure to check this out on your desktop: http://blogs.adobe.com/webplatform/2013/08/06/gradientmaps-js-gradient-maps-for-html/.

On the right you should see an embedded iFrame. You can change the page you want to display by typing in your own URL and clicking the "Reload IFrame" button.

On the left you'll see a list of gradient map presets, and below that you can enter your own custom gradient maps and apply them to the iFrame on the right.

If you click on some of the presets, you'll see that the gradients look an awful lot like CSS linear gradients, and that is intentional. You can copy/paste directly from a CSS linear gradient. Everything is the same except there is no initial angle, sides or corners.

Due to security constraints with iFrames, the Codepen above will not show embedded Flash video. Instead, if you want to try applying gradient maps to sites like Adobe TV or YouTube, then you can run the same demo here: http://blattchat.com/gradient-map-on-an-iframe/. Make sure, if you want to view a YouTube video, that you use the embed URL. Or, if you want to see a wacky example of animated gradient maps, try this out.

USING GRADIENTMAPS.JS

The API is really easy to use. First, include the gradientmaps library on your page:

```
<script src="gradientmaps.min.js"></script>
```

Now, you can apply a gradient map to any element using the following Javascript:

```
GradientMaps.applyGradientMap(targetElement, gradientString);
```

gradientString is specified as a comma separated list of color stops. Each color stop is a color (specified in RGB, RGBA, HSL, HSLA, a named color or #hex format) followed by an optional position (specified as either a fraction from 0.0 to 1.0 or a percentage).

For example, here's a gradientString that would convert your content to black and white:

```
“black, white”
```

If you want the gradient to start at black, gradually turning to blue at 10%, and then to full white at 100%, you could do the following:

```
“black, rgb(0, 0, 255) 10%, white”
```

There are a few assumptions made when no position is specified. These are the same assumptions as for a CSS linear gradient:

- If the initial color stop has no position, it is assumed to be 0%
- If the final color stop has no position, it is assumed to be 100%
- If a color stop has no position, and it is not the first or last color stop, it is positioned half way between the previous and next color stop
- If a color stop's position is less than the previous color stop, it is repositioned to that of the previously positioned color stop

If you want to remove a gradient map from an element, the following method is available:

```
GradientMaps.removeGradientMap(targetElement);
```

That's it. Pretty simple and straightforward.

HOW IT WORKS

Behind the scenes, gradientmaps.js is making use of SVG filters, specifically the color matrix and component transfer filter primitives. When applying a gradient map, first an SVG element is added to the DOM, and a filter with a dynamically generated ID is added to the SVG:

```
<svg version=”1.1” width=”0” height=”0”>
   <filter id=”filter-1375725609202”/>
</svg>
```

A color matrix filter primitive is then used to convert the HTML rendered image to grayscale, with the darkest colors mapped to black, and the brightest colors mapped to white.

```
<svg version=”1.1” width=”0” height=”0”>
   <filter id=”filter-1375725609202”>
      <feColorMatrix type=”matrix”
         values=”0.2126 0.7152 0.0722 0 0 0.2126 0.7152 0.0722
                 0 0 0.2126 0.7152 0.0722 0 0 0 0 0 1 0”/>
   </filter>
</svg>
```

A component transfer filter primitive then takes the output from the previous filter primitive to convert the grayscale intensities to the requested gradient map.

```
<svg version="1.1" width="0" height="0">
   <filter id="filter-1375725609202">
      <feColorMatrix type="matrix"
         values="0.2126 0.7152 0.0722 0 0 0.2126 0.7152 0.0722
                 0 0 0.2126 0.7152 0.0722 0 0 0 0 0 1 0"/>
      <feComponentTransfer color-interpolation-filters="sRGB">
         <feFuncR type="table" tableValues="1 0 0"/>
         <feFuncG type="table" tableValues="0 1 0"/>
         <feFuncB type="table" tableValues="0 0 1"/>
         <feFuncA type="table" tableValues="1 1 1"/>
      </feComponentTransfer>
   </filter>
</svg>
```

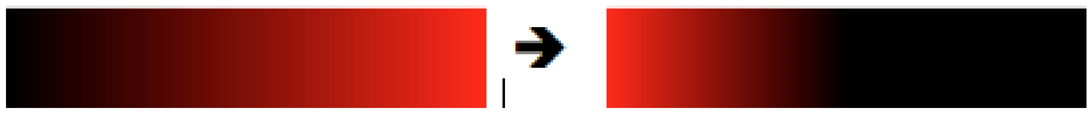
Calculating the actual values for the component transfer primitive's individual channel transfer functions is where the meat of the gradientmap library lies.

The component transfer filter primitive allows you to remap the individual color channels using transfer functions (feFuncR, feFuncG, feFuncB and feFuncA for RGBA respectively). In our case we are using ***table*** type transfer functions. There are different types of transfer functions that you can read about in the filter effects specification. With table type transfer functions, the function is defined through linear interpolation of the values specified in the ***tableValues*** attribute.

In the example above,the red transfer function has its tableValues set to:

```
1 0 0
```

The table values will be evenly distributed, meaning the first value will map reds of value 0.0, the last value will map reds of 1.0 and any stops in between will be evenly distributed. In the case above, the middle 0 will thus map reds of value 0.5. In this example, the darkest reds will get mapped to full red, and any reds with a value of 0.5 or more will get mapped so they have no red.



The problem of course is that we want to be able to add values at arbitrary positions in the gradient, not just at evenly distributed positions. The library takes care of that for you, figuring out what set of evenly distributed table values can be used with the individual channel transfer functions to achieve the desired effect.

For example, if you wanted to start at black, go to blue at 10% and then end on white:

the library would calculate table values for the transfer function table values at 0%, 10%, 20%, 30%, 40%, 50%, 60%, 70%, 80%, 90% and 100%:

```
<svg version="1.1" width="0" height="0">
   <filter id="filter-1375725609202">
      <feColorMatrix type="matrix"
         values="0.2126 0.7152 0.0722 0 0 0.2126 0.7152 0.0722
                 0 0 0.2126 0.7152 0.0722 0 0 0 0 0 1 0"/>
      <feComponentTransfer color-interpolation-filters="sRGB">
         <feFuncR type="table"
            tableValues="0 0 0.1111111111111111 0.2222222222222222
0.3333333333333333
                         0.4444444444444444 0.5555555555555555
0.6666666666666666
                         0.7777777777777778 0.888888888888889 1"/>
         <feFuncG type="table"
            tableValues="0 0 0.1111111111111111 0.2222222222222222
0.3333333333333333
                         0.4444444444444444 0.5555555555555555
0.6666666666666666
                         0.7777777777777778 0.888888888888889 1"/>
         <feFuncB type="table" tableValues="0 1 1 1 1 1 1 1 1 1 1"/>
         <feFuncA type="table" tableValues="1 1 1 1 1 1 1 1 1 1 1"/>
      </feComponentTransfer>
   </filter>
</svg>
```

Finally, the target element has its CSS modified, setting -webkit-filter and filter to point to the dynamically created SVG filter:

```
-webkit-filter: url(#1375725609202);
        filter: url(#1375725609202);
```

When modifying an existing gradient map, the library will look for the existing filter that is applied to that element, and modify it in place so you don't end up with zombie filters in your DOM. When removing a filter, the CSS filter attributes will be reset and the filter will be removed from the SVG element. If the SVG has no other filter children, it will be removed as well, ideally keeping everything nice and tidy.

SUMMARY

I'd like to the thank the Adobe Web engine team for originally pointing me down this path and for their support, both technically and legally.

You can find the all of the code and examples on my GitHub repository:

https://github.com/awgreenblatt/gradientmaps

By all means, let me know if you find problems or think there are some features to add. Even better, make the changes yourself and do a pull request. I'd love to get other's input on making this better.

I can best be reached on Twitter at @agreenblatt. That's also a great way to keep up with any major updates or improvements to the library. You can also read about some of my other technical ramblings at http://blattchat.com. Please do let me know if you do anything interesting with the library.

Enjoy!

Alan Greenblatt

Creative Cloud Evangelist

HIS BLOG SHARE TWITTER GITHUB

Writing Better CSS

by Mathew Carella

by Mathew Carella

"Essentially, the fewer the rules the engine has to evaluate the better."

This article first appeared on the Flippin' Awesome website on August 12, 2013. You can read it here.

Writing good CSS code can speed up page rendering. Essentially, the fewer rules the engine has to evaluate the better. MDN groups CSS selectors in four primary categories, and these actually follow the order of how efficient they are:

1. ID Rules
2. Class Rules
3. Tag Rules
4. Universal Rules

The efficiency is generally quote from Even Faster Websites by Steve Souders which was published in 2009. Souders list is more detailed, though, and you can find the full list referenced here. You can find more details in Google's best practices for efficient CSS selectors.

In this article I wanted to share some simple examples and guidelines that I use for writing efficient and performant CSS. This is inspired by, and follows a similar format to, MDN's guide to writing efficient CSS.

DON’T OVERQUALIFY

As a general rule, don’t supply more information than is necessary.

```
// bad
ul#someid {..}
.menu#otherid{..}

// good
#someid {..}
#otherid {..}
```

DESCENDANT SELECTORS ARE THE WORST

Not only is this not performant but it is fragile, as changes to the HTML can easily break your CSS.

```
// very bad
html div tr td {..}
```

AVOID CHAINING

This is similar to overqualifying and it is preferable to simply create a new CSS class selector.

```
// bad
.menu.left.icon {..}

// good
.menu-left-icon {..}
```

STAY KISS

Let’s imagine we have a DOM like this:

```
<ul id=”navigator”>
    <li><a href=”#” class=”twitter”>Twitter</a></li>
    <li><a href=”#” class=”facebook”>Facebook</a></li>
    <li><a href=”#” class=”dribble”>Dribbble</a></li>
</ul>
```

Following upon the prior rules…

```
// bad
#navigator li a {..}

// good
#navigator {..}
```

USE A COMPACT SYNTAX

Whenever possible, use the shorthand syntax.

```
// bad
.someclass {
 padding-top: 20px;
 padding-bottom: 20px;
 padding-left: 10px;
 padding-right: 10px;
 background: #000;
 background-image: url(../imgs/carrot.png);
 background-position: bottom;
 background-repeat: repeat-x;
}

// good
.someclass {
 padding: 20px 10px 20px 10px;
 background: #000 url(../imgs/carrot.png) repeat-x bottom;
}
```

AVOID NEEDLESS NAMESPACING

```
// bad
.someclass table tr.otherclass td.somerule {..}

//good
.someclass .otherclass td.somerule {..}
```

AVOID NEEDLESS DUPLICATION

Whenever you can, combine duplicate rules.

```
// bad

.someclass {
 color: red;
 background: blue;
 font-size: 15px;
}

.otherclass {
 color: red;
 background: blue;
 font-size: 15px;
}

// good

.someclass, .otherclass {
 color: red;
 background: blue;
 font-size: 15px;
}
```

CONDENSE RULES WHEN YOU CAN

Following on the prior rule, you can combine duplicate rules but still differentiate classes.

```
// bad
.someclass {
 color: red;
 background: blue;
 height: 150px;
 width: 150px;
 font-size: 16px;
}

.otherclass {
 color: red;
 background: blue;
 height: 150px;
 width: 150px;
 font-size: 8px;
}

// good
.someclass, .otherclass {
 color: red;
 background: blue;
 height: 150px;
 width: 150px;
}

.someclass {
 font-size: 16px;
}

.otherclass {
 font-size: 8px;
}
```

AVOID UNCLEAR NAMING CONVENTIONS

It is preferable to use semantic names. A good CSS class name should describe what it is about.

AVOID !IMPORTANTS

When possible, you should instead use good qualified selectors.

FOLLOW A STANDARD DECLARATION ORDER

While there are a number of common ways to order CSS properties, this is a commonly used one that I follow.

```
.someclass {
 /* Positioning */
 /* Display & Box Model */
 /* Background and typography styles */
 /* Transitions */
 /* Other */
}
```

FORMAT YOUR CODE PROPERLY

Code that is easier to read is easier to maintain. Here's the format I follow:

```
// bad
.someclass-a, .someclass-b, .someclass-c, .someclass-d {
 ...
}

// good
.someclass-a,
.someclass-b,
.someclass-c,
.someclass-d {
 ...
}

// good practice
.someclass {
    background-image:
        linear-gradient(#000, #ccc),
        linear-gradient(#ccc, #ddd);
    box-shadow:
        2px 2px 2px #000,
        1px 4px 1px 1px #ddd inset;
}
```

WHERE TO GO FROM HERE

Obviously these are just a handful of rules that I try to follow in my own CSS to make it both more efficient and easier to maintain. If you want to read more on the topic, I suggest reading Writing Efficient CSS on MDN and Google's guide to optimize browser rendering.

This article was originally published at http://blog.mathewdesign.com/2013/07/04/writing-performant-and-quality-css/

Mathew Carella

Web Developer

HIS BLOG SHARE TWITTER GITHUB

SVG Filters on Text

by Chris Scott

by Chris Scott

"SVG with Filter Effects have a lot of potential for complex text styling."

This article appeared as a guest post on the CSS-Tricks website on August 9, 2013. You can read it here.

CSS-TRICKS

There has been a general trend in Web development, for some years now, away from using images in designs. Only a few years ago software companies would favour using an image of a rounded corner as the best "cross-browser" solution; the CSS attribute `border-radius` has made that technique seem very antiquated today. Titles are another example of this trend, where in the past one may have generated a fancy banner title in Photoshop and used an image to show it on the page. These days we have web fonts at our disposal and CSS3 to help us achieve shadows and other effects. These solutions load much faster, scale better and are more accessible and semantically correct. But there is even more we can do!

SVG FILTERS

SVG with Filter Effects have a lot of potential for complex text styling. Take a look at this example:

That line is created using SVG Filter Effects. It's just text. You can select it with your cursor. Search engines can index it. It will scale in size without losing quality. To boot, it takes very little time to download. You can achieve a whole lot more, too, the scope for creativity with Filter Effects is huge. The example was created with a library called Raphael.js and an extension I wrote for it. This article talks about the rationale for developing the extension and shows - in brief - how it can be used.

Apparently, only 0.1% of Web pages use SVG graphics. If that statistic is anything to go by, there's a good chance that you are probably not using SVG on a regular basis. Why is SVG used so unpopular? It's only a guess, but the reason I didn't get into SVG (until I absolutely had to) was its learning curve: SVG is an XML vocabulary and is, I think, extremely technical (matrix multiplication for colour shifts, anyone?). The way I got into SVG was through Raphael.js, a JavaScript Library for creating vector drawings. Because it's a JavaScript library it felt fairly familiar, all of the complexity was abstracted away. Before long I was creating complex graphics like a pro.

FILTER EFFECTS FOR RAPHAEL

Raphael has a shortcoming though: no support for Filter Effects. I remember one of my customers specifically requesting a drop shadow for bubbles in a data visualization which was interactive and animated. The request stumped me for a while as I bumped into this limit of Raphael. For that project I wrote a very specific extension to Raphael for handling drop shadows. But the same complexity that had initially put me off SVG was back and worse than ever. Make no mistake, Filter Effects are very, very technical.

So, after that project, I set about building a more full-featured library to make Filter Effects as easy to apply as Raphael makes drawing shapes.

Introducing Filter Effects for Raphael!

Here's how to use it:

First, the HTML. This bit is dead simple, just a `div` with an `id`:

```
<div id="title-container"></div>
```

Everything else is done in JavaScript.

To start an SVG drawing with Raphael you create a "paper" by referencing the id of the container element:

```
var paper = Raphael("title-container");
```

Now to do some drawing. This example creates some text and sets some of the style attributes:

```
// The first two arguments are the origin of the text
var title = paper.text(0, 30, “Filters are ice cold”);
title.attr({
“fill”: “MintCream”,
“font-family”: “Butcherman”,
“font-size”: “54px”,
“text-anchor”: “start”
});
```

Now for some effects! The simplest things you can do are shadows, blurring, lighting, embossing and colour shifting. Those require very little coding. Let’s try the emboss effect. Add to the JavaScript:

```
title.emboss();
```

You can chain effects, so applying a shadow afterwards is straightforward:

```
title.emboss().shadow();
```

Pretty cool, huh? You can take this much further if you want, by creating your own filters. The SVG spec lists (lower level) filter effects which can be combined to create all kinds of filters (convulsion matrices, in particular, can be used for a vast number of operations).

This demo has the full code and some other examples of different effects that can be achieved:

SHORTCOMINGS

What are the downsides to SVG? Well - aside from the pros and cons of vector over raster (like `canvas`) - there are a couple I can think of:

- ***Browser support*** - You may not be surprised to learn that the graphical trickery discussed here is not supported in older versions of IE, 10 only. SVG itself will render in IE9, just without the effects. Firefox, Chrome and Opera have supported Filter Effects for ages
- **Semantics** - Some may have reservations about the semantic validity of using SVG in documents, certainly the `svg` tag doesn't give any clue as to it's content; you can use a sensible parent, though

WRAPPING UP

Hopefully this piece has given you a good idea of what filter effects can do and how Filter Effects for Raphael can do them. Check out the project page on Github for more details. Thanks for reading!

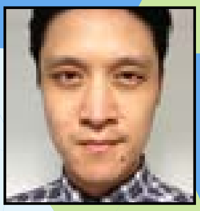
Chris Scott

Developer & Entrepeneur

HIS BLOG SHARE TWITTER GITHUB

Commercial PhoneGap in the Wild

by Sel-Vin Kuik



by Sel-Vin Kuik

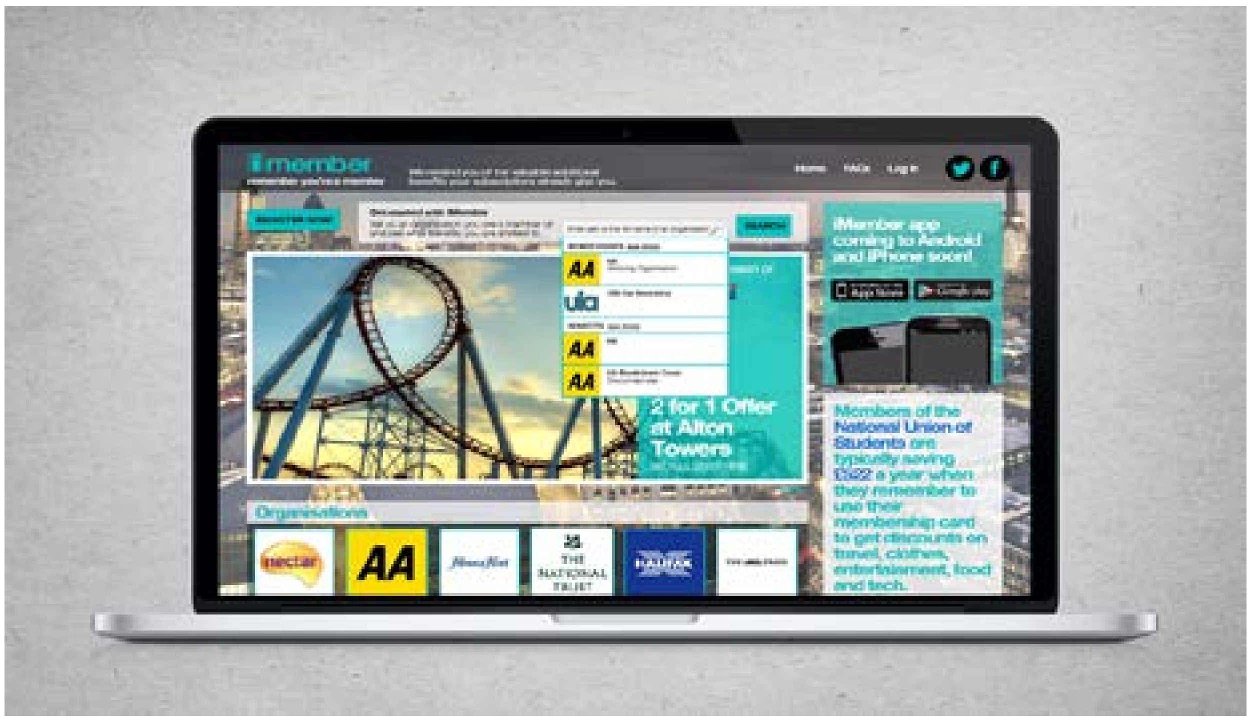
"Choosing unwisely, especially as a startup digital agency, could cost you dearly."

Working in development, you relish problem solving and finding the optimal platform on which to deploy your project. Choosing unwisely, especially as a startup digital agency, could cost you dearly. In September 2012, when SMACK (http://smackagency.com) won its first mobile app project, it was crucial that we got the platform right. The right technology, to be able to deliver the right functionality efficiently and, of course, at the right price to meet budgets.

Traditionally, although theoretically the better solution, hybrid apps in practice were the poor relation to their native counterparts. However, things were starting to change - the question was, was it enough to choose hybrid, and would the decision pay off.

THE PROJECT

Our client first came to us with the ground-breaking idea for a free, money-saving website through which users would be able to access offers and membership benefits for organisations, venues and retailers across the UK. With thousands of deals already logged in the database, a number which rapidly grew throughout development and past launch, it was imperative that the system was scalable to process large quantities of data within minimal time.

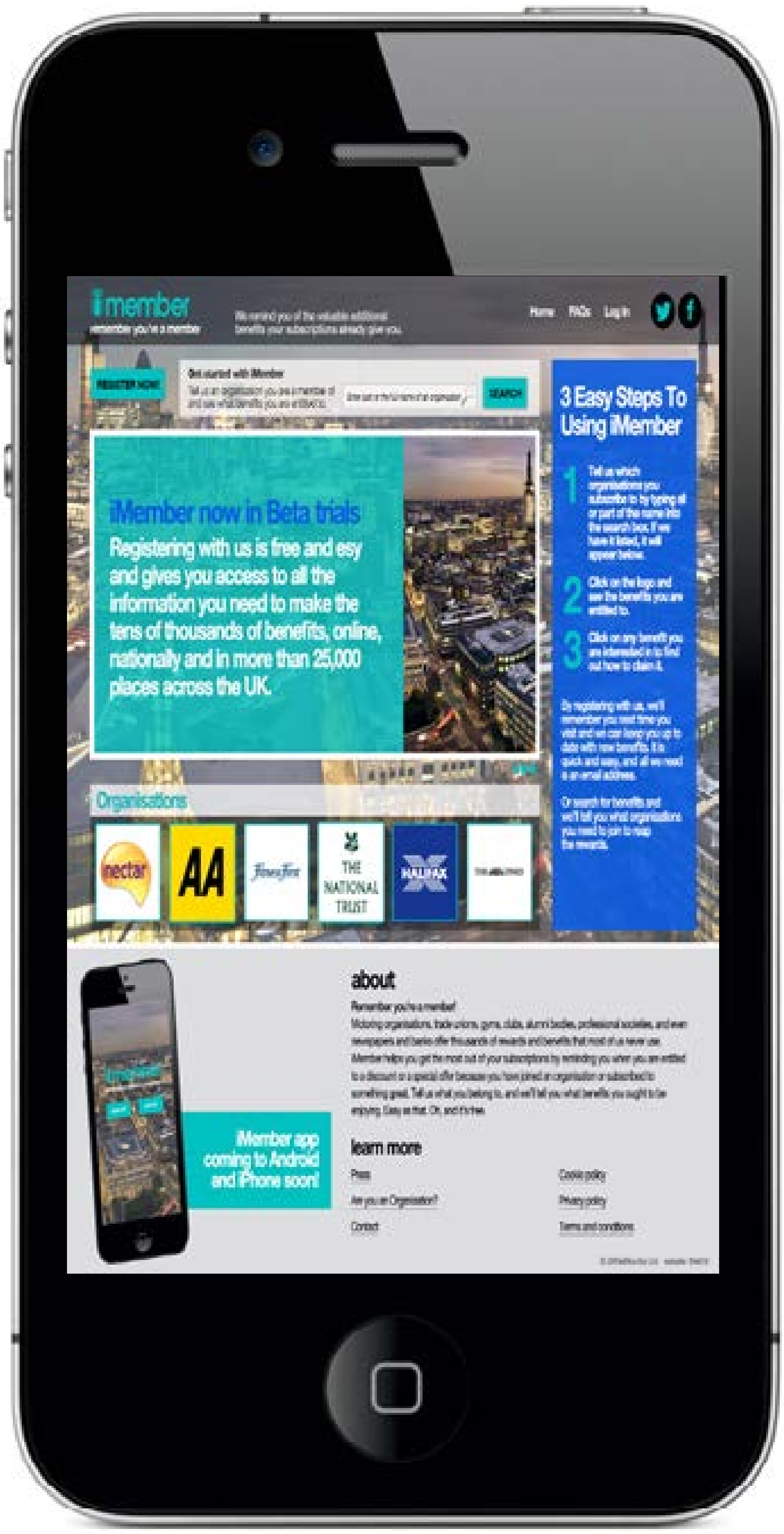


The mobile app was an essential addition to the website allowing users to access their profiles on the go. Using live geolocation based functionality, push notifications could be configured to alert of any benefits coming in to vicinity as the user moves around.

HYBRID VS. NATIVE

When it boiled down to it, choosing PhoneGap wasn't an instinctive decision. PhoneGap was the exciting upstart I had had my eye on since I first came across it in 2010. However, the jury was still out as far as speed and economies were concerned. Coming to it from a web background, I strongly supported the single-source code base offered. However, on the hybrid vs. native argument, I remained firmly on the native side of the fence. Why? Because to deliver commercially viable apps it has to be considered that there is a huge variety of devices on the market. Until very recently, all but the top-end devices out there suffered from poor HTML5 support and sluggish JavaScript engines. This gave native the upper hand for both performance and audience reach, preventing me from even considering deploying a hybrid solution to a full-scale project.

Swipe through the screens



We remind you of the valuable additional benefits your subscriptions already give you.

Get started with iMember

Tell us an organisation you are a member of and see what benefits you are entitled to.

iMember now in Beta trials

Registering with us is free and esy and gives you access to all the information you need to make the tens of thousands of benefits, online, nationally and in more than 25,000 places across the UK.

3 Easy Steps To Using iMember

1 Tell us which organisations you subscribe to by typing all or part of the name into the search box. If we have it listed, it will appear below.

2 Click on the logo and see the benefits you are entitled to.

3 Click on any benefit you are interested in to find out how to claim it.

By registering with us, we'll remember you next time you visit and we can keep you up to date with new benefits. It is quick and easy, and all we need is an email address.

Or search for benefits and we'll tell you what organisations you need to join to reap the rewards.

about

Remember you're a member!

Motoring organisations, trade unions, gyms, clubs, alumni bodies, professional societies, and even newspapers and banks offer thousands of rewards and benefits that most of us never use. iMember helps you get the most out of your subscriptions by reminding you when you are entitled to a discount or a special offer because you have joined an organisation or subscribed to something great. Tell us what you belong to, and we'll tell you what benefits you ought to be enjoying. Easy as that. Oh, and it's free.

learn more

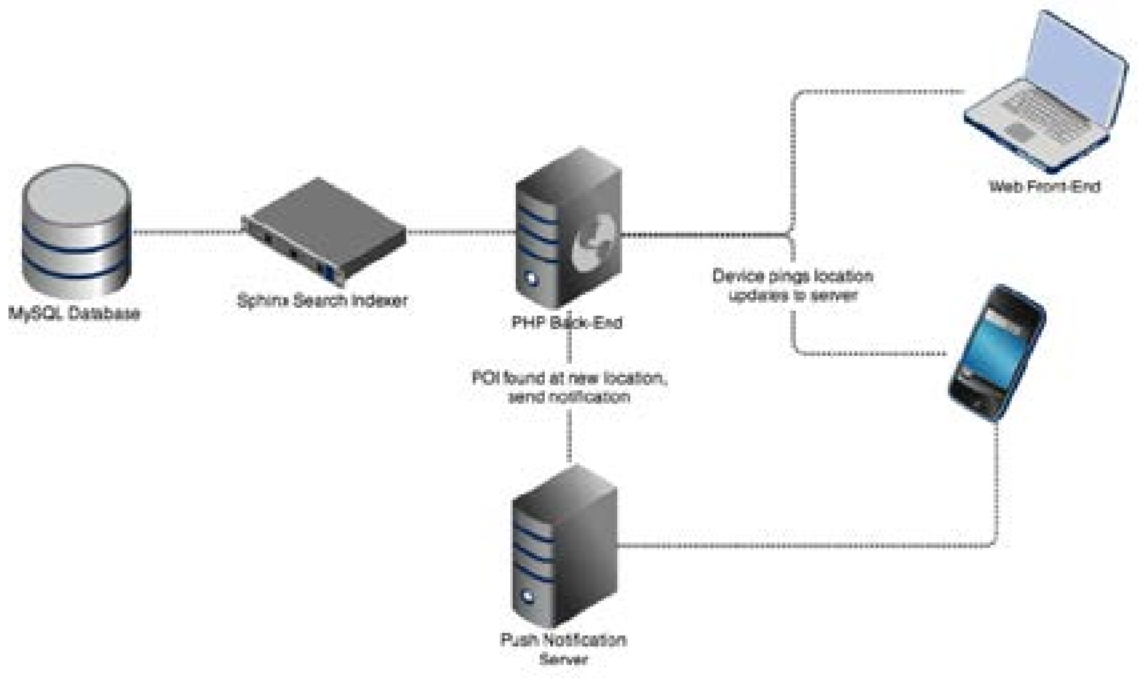
Press	Cookie policy
Are you an Organisation?	Privacy policy
Contact	Terms and conditions

THE TURNING POINT

When my client first approached me, devices such as the Nexus 4 and iPhone 5 were being unveiled. For Android, the Nexus 4 set a new milestone for mobile processing power, and the improvement of JavaScripting on the iPhone 5 created a benchmark above and beyond that seen before. For developers, PhoneGap was leading the way for hybrid platforms with a strong, growing community and a wide variety of plugin support to access native functionality. For me, that provided the turning point - hybrid applications had finally become a serious contender to native apps, and my decision to develop with PhoneGap was made.

LET'S GET TECHNICAL

When it comes to the app itself, it is worth taking a step back to outline the app back-end. The structure is fairly traditional, the data is stored server-side in a MySQL database, with a Sphinx (http://sphinxsearch.com/) search engine layer on top for optimised indexing. Practically all of the data stored can be represented with latitude and longitude coordinates for it to be mapped. Yii framework (http://www.yiiframework.com/) powered PHP controls the logic.



On the front-end, at the surface jQuery (http://jquery.com/) was used to do a lot of the heavy-lifting in terms of DOM manipulation and Ajax calls. I used parsley.js (http://parsleyjs.org/) to build a modular validation system, and mustache.js (http://mustache.github.io/) for simple repetitive templating within each page of the app. To maintain as much compatibility towards older devices, the use of transitions and animations was kept to an absolute minimum. Although it is possible, I would advise that the rule of thumb for any developer would be to always use CSS3 over JavaScript in this case.

You can see a sample of the JavaScript structure, which outlines handling the page transitions seamlessly with PhoneGap and jQuery.

```
/**
 * Page constructor
 */
$(document).bind('pageshow', function(event, ui) {

   var $content = $('.ui-page-active');

// Check if page content has already been loaded previously
   if(!$content.hasClass('initialised')) {

        // Show a loading spinner, initially hidden
        $.mobile.loading('show', {
            text: 'Your connection is wonky.',
            textVisible: false
        });

        // Dynamic page initialisation, based on content ID
        eval('app.' + $content.attr('id') + '.init();');

   }

});

/**
 * Page destructor
 */
$(document).bind('pagehide', function(event, ui) {

app.cleanUp();

});

/**
```

```
 * Page ready
 */
$(document).on(‘pageready’, function() {

  // Hide loader
  $.mobile.loading(‘hide’);

  // Flag page as initialised
  $(‘.ui-page-active’).addClass(‘initialised’);

});


/**
 * Global Ajax error handler
 * Hooks in to jQuery, if connection is ever lost during data population
 * the loading spinner is updated to notify the user
 */
$(document).ajaxError(function(event, jqXHR, ajaxSettings, thrownError) {

  $(‘.ui-loader’).removeClass(‘ui-loader-default’).addClass(‘ui-loader-
verbose’);

});

/**
 * Example page
 */
app.examplePage = {};
app.examplePage.init = function() {

  // Initialise page, setup interactions, populate data over Ajax

  // Trigger page ready when done
  $(document).trigger(‘pageready’);

};
```

THE NITTY GRITTY

Digging deeper, it gets a lot more interesting. When the app is booted it taps in to PhoneGap's location listener to track the device position. As PhoneGap uses the HTML5 geolocation specification this is exceptionally easy. Even location tracking when the app is in the background is simply achieved by switching a flag in either ADT or Xcode, functionality often considered impossible across the web. PhoneGap actually makes this remarkably simple, and this was a key requirement for the iMember app.

Once the location has been obtained, the app pings the server for relevant points nearby. I found this traditional approach to be much more effective than using geofencing, in our case only due to the sheer volume of positional data being stored in the database. The implementation of Sphinx makes light work of searching through this large data collection to return instant results.

JAVASCRIPT

```
/**
 * Device Ready
 */
document.addEventListener('deviceready', function() {

   // Listen for location changes
   navigator.geolocation.watchPosition(
        function(position) {

             // Success function, send the user location update to the server

        }
   );

}, true);
```

PHP

```
function updateLocation($user, $latitude, $longitude)
{

   // Find the user’s current location
   $oldLocation = $user->getCurrentLocation();

// Update the user location
   $user->updateLocation($latitude, $longitude);

   // Use the Haversine formula to calculate if the user has moved a significant
distance
   if(CMath::haversine(
$oldLocation->latitude, $oldLocation->longitude,
$latitude, $longitude
) < 0.1)
   return false;

   // Use Haversine/Sphinx to search the local area for relevant benefits
   if($benefits = $user->nearbyBenefits($latitude, $longitude))
   {
       // Send push notifications
   }
}
```

Through the use of PhoneGap plugins, iMember further taps into functionality on device. For when the user is on the move, the capability to register and receive push notifications was developed using PushPlugin. As location data is being sent to the server, the changes in position are monitored. If the user has moved to a new location where new points of interest are available, a notification is sent. There is a good chunk of back-end functionality here to register the device, store the location and send the notification. I opted for a bespoke build of our notification server, although it is outside the scope of this article, it is much simpler than it sounds and there are some great references online. There are also a good number of paid services if you don’t want the bother of managing a notification server.

For iOS, PushPlugin and plugins in general require quite a bit of work to configure within Xcode and the iOS Developer Centre. This can be quite daunting to a hybrid developer with little native experience, but plugin documentation is continuously updated, and it often comes down to a case of meticulously following instructions. For those of you working with Android however, you’ll be pleased to know that Google makes the whole process considerably simpler.

Probably one of the more deceiving complex pieces of functionality that had to be included was tapping in to the social functionality on device. For a web developer, it's not simply the case of copying in your favourite social widget and hey presto as you might expect. However, again a PhoneGap plugin was available to assist, and for this we used the ShareKit.

Again, the setup for iOS can be slightly more complicated than Android, especially if you are looking to make use of the recent updates to social in iOS 6 yet still maintain iOS 5 compatibility. It is on this note that although PhoneGap can provide a single-source principle, it is not the case that writing one piece of code will allow deployment across all devices. Each OS has it nuances, and this should be handled by splitting out plugin functionality in to modules for devices and even device versions. Good code organisation and namespacing cannot be emphasised enough, I included a global switch for devices, so although common functionality can be grouped together, device-specific alterations can easily be separated out.

```
var IS_IOS = true;

/**
 * Device Ready
 */
document.addEventListener(‘deviceready’, function() {

   // Set global variable to target platforms independently
   var platform = device.platform.toLowerCase();

if(platform.indexOf(‘android’) > -1)
   IS_IOS = false;

   // Register for push notifications
app.pushNotifications.register();

}, true);

/**
 * Push Notifications
 */
app.pushNotifications = {};
app.pushNotifications.register = function() {

   // Android
   if(IS_IOS)
        window.plugins.pushNotification.register({
             ecb: ‘app.pushNotifications.ios’
        });
   // iOS
   else
       window.plugins.pushNotification.register({
           ecb: ‘app.pushNotifications.ios’
       });
};

app.pushNotifications.android = function(event) {

   // Handle push notifications on Android

};

app.pushNotifications.ios = function(event) {

   // Handle push notifications on iOS

};
```

One other observation when developing with PhoneGap plugins, or indeed any third party plugins for a rapidly evolving platform, is versioning and compatibility. Especially when you begin to mix multiple plugins, unless you have the resource to contribute to the open-source development, it is often the case that an older version of PhoneGap needs to be used to cater across the board.

IN CONCLUSION

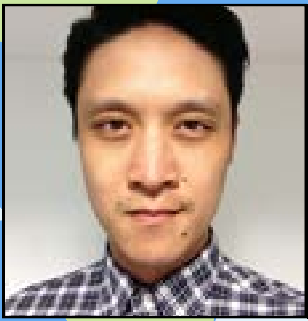
In order to deliver successful apps, it is crucial that the foundations can perform to 100%. Hybrid apps remain the most hotly discussed subjects when it comes to mobile development, however the decision to deploy with hybrid hasn’t always come easily in the past.

Hopefully, reading about my experience will help shed some light on the possibilities that PhoneGap can offer. Maybe it’ll even help convince those in “camp native” that there is another way in which to build full-featured, scalable commercial apps. I’ve found that PhoneGap can simplify complex functionality, thus optimising the build process.

As the number of hybrid apps grows, so will the community and support, and in turn the requirement for device capability. At SMACK, development is already fully underway to deliver a commercial hybrid HTML5 game, and my next personal challenge is to follow this with a commercial scale augmented reality hybrid app. The golden time for hybrid is dawning, and as PhoneGap improves and ideas and notions develop, there's no better time to start developing a hybrid app.

BIO

Sel-Vin Kuik is the technical director at SMACK, a digital creative agency developing web apps, mobile apps and digital magazines.



Sel-Vin Kuik

Technical Director

HIS BLOG

SHARE

TWITTER

GITHUB

INTERVIEW

VITALY FRIEDMAN

by Maile Valentine
photos Oliver Kern
OliverKern Fotografie

APPLINESS: Hello Vitaly, we're big fans of your work! Thank you for spending time with Appliness. Can you tell us a bit about yourself?

Thank you so much for inviting me. It's my honor and pleasure to be featured in Appliness — I've been reading it for a while now. Well, I am one of the guys who has been doing lots of different things on the Web throughout all these years. I jumped into Web design when I studied computer science and mathematics at the university and then, at some point, I realized I am a pretty mediocre developer which I was why I started doing something different. I was always fascinated by typography and design, so I explored this direction instead. At that time I didn't have any money and when I was asked about a couple of Web design projects at the university, I jumped into this area with both feet.

I used to design table-based websites when I was a kid, back in 1999, but then I was disillusioned by what I saw as an incredibly time-consuming and impractical craft. I did create a couple of websites back then, and I also experimented with Shockwave and VRML at some point. This is also the point when I started writing about the Web and my experiences with it. It was the time when everybody tried to figure out what to do with the Web, but the most important thing for me was the fact that I could publishing anything online and people across the globe could access it. It was a fascinating thought. Actually, it still is. I often think that this is one of the reasons why I still do my work today.

Smashing Magazine features articles on the latest techniques and tools for modern web design. Based on your experience as a web designer and your exposure to the latest technologies, what are some of the tools you are most excited about?

To be honest, I rarely get excited about tools; nor am I interested in trends or exciting visual treatments. I love clever solutions to hard design problems. I love seeing CSS trickery and innovative, smart designs. This is why I am particularly interested in responsive Web design patterns as well as performance optimization techniques which are derived from suboptimal real-life projects. Usually our solutions have to be pragmatic and work well within given project constraints. That's a gateway to smart, unusual design choices, in my opinion.

When I think about new techniques, I tend to think about Atomic Design as proposed by Brad Frost as well as general design of websites as systems rather than pages. I am also curious of how we can further improve the overall user experience of websites by using offline storage technologies and prefetching content in a smart way. I am very curious to see how our deliverables and our project workflow are changing (and will change in the future) as responsive Web design will become a default for every project. Overall, processes and workflow fascinate me. The insights you can learn from them are much more profound and empowering than any standalone tool could provide.

VITALY FRIEDMAN

SMASHING MAGAZINE

What were some of the tools or frameworks you and your team used to redesign Smashing Magazine for a responsive design?

Well, to be honest, we didn't use any framework at all — the whole site was built from scratch. I've never been a big fan of frameworks actually, and it wasn't necessary in our case because the actual design was quite… well, straightforward. We worked in the Chrome Devtools all the time and we played around with CSS values and properties. At that point we didn't really have any tools that would help us with debugging and maintenance or actual testing of media queries, so we kept resizing the window all the time.

We also used the asterisk technique as proposed by Trent Walton (http://trentwalton.com/2012/06/19/fluid-type/) but that was it, actually.
We didn't use Photoshop at all, and most experiments were done directly in the browser.

Smashing Magazine started as a side project and grew to be a very successful, full-time endeavor. Are there any plans for expanding your scope of products or services? More book publishing in the future?

Well, personally I love challenges. I tend to explore directions I've never thought of or never had experience in. This is how eBooks and printed books and conferences and workshops came to be. Indeed we are working on a variety of books at the moment, but I think I've already found a new challenge that I'd love us to take on in the nearest future. And no, that's not what you are expecting. :-)

Smashing Magazine gained popularity very quickly. What did it offer that your readers had been missing?

I think that our readers discovered that we emphasized the importance of practical, useful articles that always delivered on the promise of providing value to the reader. That's what has become our signature, so to say. It hasn't changed over all these years at all, only the form in which content is presented and how it double checked before it goes live, has changed significantly. I think that Smashing Magazine appeared at the right time in the right niche and so popular articles helped us grow over time and gain more and more readers over the years.

Can you tell us about the Smashing Conference you organize? What is the content of the conference? It's goal? Are you considering any pure online events?

When we started planning the Smashing Conference (www.smashingconf.com), we had a clear idea of what the conference should feel like and what its purpose should be. It can't be a generic, quick-paced, ordinary event. It has to be special, different, valuable and meaningful. It has to be… smashing in every possible way. At its essence, the SmashingConf is a very hands-on event, with thorough discussions about how we work, what strategies and techniques we use and how we can use them intelligently. Every talk should deliver value and should serve a purpose; it should be helpful and remarkable, it should challenge, share and inspire. If all pieces of the puzzle come together nicely, the conference should become a memorable community event, with practical insights and takeaways for speakers and guests alike. It should be an event that everyone remembers and looks forward to again next year.
We wanted to apply the very principles, values and philosophy that lie in the very heart of Smashing Magazine, to the conference and this is exactly how it came to be.

VITALY FRIEDMAN

TECHNOLOGY

With the rate and frequency at which new frameworks and toolsets are developed, what are the important factors you take into account when deciding on the appropriate technology to focus on?

Performance and speed. That's the key for every single design decision that we should make when deciding on an appropriate technology. Of course, every project will have different requirements and different scopes as well as different budgets, but I strongly believe that performance should become the established principle that should guide and inform design decisions.

The technology is appropriate for a project if it results in a clean, maintainable code that runs fast, is built within the progressive enhancement methodology (of which I am a strong proponent).

What is still not available that you would like to see for web designers and developers?

Good question. I see a couple of

interesting tools appearing now — http://macaw.co, for example. These tools try to solve the problem that designers currently have: Photoshop is very difficult to use for responsive design. Stephen Hay has been advocating using an entirely different responsive workflow which uses lots of different tools and Terminal for HTML prototypes, for examples. It would be nice to have a couple of bulletproof approaches that would ease the process of design and code of responsive websites and applications. The tools we have are either too general or too specific, so perhaps that's the gap we have to fill.

Also, I feel that many of us are still struggling with the actual responsive design process — what tools to use, how the deliverables should look like, what the workflow is, how to integrate everything into a working environment, where content strategy and performance fit within the project. So it's not only a matter of tools but rather a matter of workflow.

What are some of the fundamental changes in the process of web design you have noticed over the years?

There were lots of them. I saw a major shift from table-based layouts to CSS layouts, then a slow shift from fixed-width layouts to fluid layouts, later sprinkled with shiny AJAXness, and now we see another major shift towards responsive layouts which pushes away fixed-width layout even further away. But also the way we think and approach Web design has significantly changed. Our processes have matured and our coding practices have improved. There are still some things that keep repeating themselves as we move along in our discipline, and there are some mistakes that are done over and over again, but we should be proud of what we have achieved, and the tools we have developed.

I vividly remember that day when Firebug came out. Now, that was a big, big deal. Now we have lots of useful, smart tools that significantly empower us in our workflow and it has been an amazing fundamental shift in the process as well. I can't wait to see what other tools we'll come up with over time as Web design becomes even more mature.

How do you feel about promoting web standards vs. proprietary features? Any exceptions to this?

Perhaps I am a bit too pragmatic, but I would always choose the most practical solution over any theoretical discussion. I am a big believer of Web standards but I can see the advantage of native applications as well. It really depends on the budget and the audience and the requirements of a given project and return on investment. Sometimes a native application makes more sense (e.g., if you want to develop a graphics-heavy, rich game experience). I am not quite sure that it can be done within a short timeframe and with a limited budget using CSS/JavaScript alone.

VITALY FRIEDMAN
& FUN

What do you do for fun?

Music plays a huge part in my life. I spend a lot of time listening to new music, but you can also find me exploring old bookstores where I take pictures of old manuscripts. I love travelling, especially to unfamiliar and obscure places. It's always great to have a real change in perspective and explore how other people think, work and build stuff. I love observing people from other industries to better understand what goes into their process and how they do what they do. It's always fascinating.

You're a good writer, any projects (past or future) that you can speak of that are not related to technology?

Oh thank you for your kind words! Well, I always want to write more. But I am not talking about books, or magazines or articles. I am talking about telling stories. I have some of ideas of how to combine both meaning and new personality in everything I do. Let's wait and see how it plays out.

How does the philosophy of carpe diem affect your life?

I used to be a perfectionist, but moving away from this was definitely one of the most significant changes in my life which helped me become better at what I do. Pragmatism is good, and being human, and hence not without mistakes, is what makes you personal, authentic, honest and real. Everyday I make sure that whatever I do, and however I feel, that, at the end of the day I feel that it couldn't have been done otherwise because I want to make things that matter, and because what I do matters to me and makes me who I am.

Language-wide Features in CSS

by Louis Lazaris

by Luis Lazazris

"There are some language-wide features that come in handy, some which are brand new in the spec."

This article first appeared on [Impressive Webs](#) on August 6, 2013. You can [read the original post here](#).

In addition to the unique property/value pairs that CSS offers, there are also a small handful of language-wide features that can come in handy, including a few that are brand new in the spec.

These often go unnoticed, especially by beginners, because when CSS properties and their possible values are discussed (in tutorials, books, or even in the spec), these features are usually omitted for brevity.

Here's a summary of four language-wide CSS features.

KEYWORD: 'INHERIT'

[Every CSS property](#) can accept, as its value, the keyword [inherit](#). Like this:

```
span {
    font-size: inherit;
}
```

What this does is cause the targeted element to inherit the specified property value from its parent element. If the parent element does not have that specific property defined in the CSS, it will inherit the parent's computed value (which could be the initial value or whatever is inherited from that element's parent).

The `inherit` keyword can come in handy when you want to assign the same value to a bunch of child elements for properties that don't normally inherit. For example:

```
.module {
  box-shadow:  0px 0px 11px rgba(0, 0, 0, 0.4);
}

  .module div {
    box-shadow: inherit;
  }
```

Now all `<div>` elements inside the main `.module` element will inherit its box shadow value.

Browser support for `inherit` is excellent. For a long time, nobody used `inherit` because IE6/7 didn't support it except on the `direction` property. Now that those browsers are mostly out of the picture, most developers should feel comfortable using it.

KEYWORD: 'INITIAL'

This one is easy to understand and is newly added in the CSS3 spec (although, technically, as with many CSS3 features, it has been in the works for a while).

Every CSS property has an initial, or default value. By defining the value using the keyword `initial`, you are telling the browser to render the property using the default for that property.

So basically this is the same as not defining the property at all, so you might think it's mostly useless. That's partly true, because you probably won't use this much. But when you do, it can be handy.

For example, you might want to use initial on a property that gets inherited from its parent by default, like `color`:

```
body {
  color: aquamarine;
}

.example {
  color: initial;
}
```

The `.example` element will normally have the same color set as that set on the body. But in this case, we're overriding this by letting the color be reset to its initial state (which is probably black).

Also, this can come in handy when changing styles dynamically with JavaScript. Through a user interaction or other change, a property can be set to its initial state, even though it may be defined specifically be default.

As for browser support, I'm really not sure. It seems to work in the latest Chrome and Firefox, but not in the latest Opera or IE10.

KEYWORD: 'UNSET'

This keyword value used to be called 'default' and has now been changed to 'unset'. This is very similar to the `initial` keyword. Basically, using `unset` as the value will erase any explicitly defined value that may have been passed to the element for that property, or previously defined elsewhere.

```
body {
  font-size: 1.5em;
}

.module {
  font-size: unset;
}
```

The difference between `unset` and `initial` is the fact that the unset value could be an inherited value. So in a way, this is kind of a combination of the previous two keywords. When 'unsetting' a value, first the browser looks for an inherited value. If none is found (for example, for a property that doesn't inherit, like `border-color`), then it sets it to the initial, or default value.

Since this is new, I don't think it has any browser support yet.

PROPERTY: 'ALL'

Finally, this is another new one: the all property. Naturally, this one would not be able to take custom values, so it has three possible keyword values: `inherit`, `initial`, and `unset` — the three keywords just discussed.

The all property resets all properties on the targeted element, with the exception of direction and unicode-bidi.

```
.example {
  all: initial;
}
```

The spec points out an interesting use case for `all` when it says:

> *This can be useful for the root element of a "widget" included in a page, which does not wish to inherit the styles of the outer page.*

As for browser support, due to the fact that the three keyword it accepts are not supported cross-browser yet, this property is probably not going to be usable for at least a little while.

You Might Also Like:

- Don't Forget About "transition: all"

Louis Lazaris
Web Developer

HIS BLOG
SHARE
TWITTER
GITHUB

Animating with AngularJS

by Holly Schinsky

by Holly Schinsky

"HTML5 makes life easier for us by defining the right element."

This article first appeared on the Flippin' Awesome website on August 5, 2013. You can read it here.

flippin' awesome!

AngularJS recently came out with support for CSS3 transitions and animations, as well as JavaScript Animations. The support is part of version 1.1.4 (unstable build), but was changed and refined a bit in version 1.1.5 so you should start with that version when you check it out. I definitely think it's worth trying because it allows you to add some fun interactions to your application quickly.

In this article I'll explain a bit about how it all works and include links to a demo application I created to try things our for yourself. The source to the demo application is located on my GitHub account here as well. I also included some great resources in the form of links at the end of the post. There's currently not a lot of documentation on this subject since it is so new, so I encourage you to check those out as well.

HOW IT WORKS...

`ngAnimate` is the name of the new directive for AngularJS animation support. The way it's applied is by adding the `ng-animate` attribute to any element containing one of the following directives in the list below:

- ngIf
- ngInclude
- ngRepeat
- ngShow / ngHide
- ngSwitch
- ngView

The only exception to the above is when you create your own custom directive with animation support using the `$animator` service. This is discussed later in the article.

Each of these directives causes a change to the DOM, which is how the transition or animation is triggered. For instance, on an `ngRepeat` they will occur when the items are repeating, for `ngShow`/`ngHide`, when the element is being shown or hidden. You simply specify `ng-animate` on your element with the name of the classes you've defined in your CSS to perform the transitions or animations and they will be applied automatically. Of course, you need to ensure you specify the `ng-animate` on the same element where you have one of the directives mentioned above (`ngRepeat`, `ngSwitch` etc.) defined or nothing will happen.

Here's a quick example of applying a scale type of transition the shorthand way (read on for notation details):

```
<ng-include ng-animate=”’scale’” src=”’partials/quote.html’”></ng-include>
```

Then in your CSS, you define a corresponding set of classes prefixed with `scale` that are used to trigger the transition or animation based on the type of event you want to animate for that directive. More info to come on this…

SUPPORTED EVENTS

Certain events are supported for each of the directives you can animate. They vary per directive and it's important to know which apply for a given directive when you're defining your CSS classes to perform the animation.

Here's the list:

Directive	Events
ngIf	enter/leave
ngInclude	enter/leave
ngRepeat	enter/leave/move
ngShow/ngHide	show/hide

Directive	Events
ngSwitch	enter/leave
ngView	enter/leave

The AngularJS docs describe exactly when those events occur for each of the directives supporting animations. So for instance the `ngIf` docs provide this description for the supported enter and leave events:

enter – happens just after the `ngIf` contents change and a new DOM element is created and injected into the `ngIf` container

leave – happens just before the `ngIf` contents are removed from the DOM

You can use this information to determine when your transitions and animations will actually be triggered.

CSS3 TRANSITIONS VERSUS CSS3 ANIMATIONS

`ngAnimate` can be used for both CSS3 animations and transitions, as well as JavaScript animations but that is beyond the scope of this article. I wanted to take a moment to briefly discuss the difference between CSS3 transitions and animations.

CSS3 Transitions
apply an effect to a style property for a certain duration. You can do things like fade, scale, move, slide, rotate, 3D effects etc.

CSS3 Animations are more complex than transitions and use keyframes to define different points to do things within the animation. These can be looped and auto-started. In addition, they don't have to depend on a DOM change to trigger them.

The difference in using CSS3 transitions versus CSS3 animations with `ngAnimate` is all in the way the CSS classes are defined. A great example can be found in the AngularJS 1.1.5 docs for ngAnimate. I'm including it here too for easy reference:

CSS3 TRANSITION SAMPLE

```
.animate-enter {
 -webkit-transition: 1s linear all; /* Safari/Chrome */
 -moz-transition: 1s linear all; /* Firefox */
 -o-transition: 1s linear all; /* Opera */
 transition: 1s linear all; /* IE10+ and Future Browsers */

 /* The animation preparation code */
 opacity: 0;
}

/*
 Keep in mind that you want to combine both CSS
 classes together to avoid any CSS-specificity
 conflicts
*/
.animate-enter.animate-enter-active {
 /* The animation code itself */
 opacity: 1;
}
```

CSS3 ANIMATION SAMPLE

```
.animate-enter {
  -webkit-animation: enter_sequence 1s linear; /* Safari/Chrome */
  -moz-animation: enter_sequence 1s linear; /* Firefox */
  -o-animation: enter_sequence 1s linear; /* Opera */
  animation: enter_sequence 1s linear; /* IE10+ and Future Browsers */
}
@-webkit-keyframes enter_sequence {
  from { opacity:0; }
  to { opacity:1; }
}
@-moz-keyframes enter_sequence {
  from { opacity:0; }
  to { opacity:1; }
}
@-o-keyframes enter_sequence {
  from { opacity:0; }
  to { opacity:1; }
}
@keyframes enter_sequence {
  from { opacity:0; }
  to { opacity:1; }
}
```

When using ngAnimate for either of the above, the same syntax would be used to apply the directive, such as:

```
<div ng-view ng-animate=”{enter: ‘animate-enter’}”></div>
```

I highly recommend reading through the documentation for further details on how this is all handled.

NAMING CONVENTIONS

When writing your CSS classes to apply your transitions or animations, you define two CSS classes for each supported event (enter, leave, show, hide, etc). One is used as a baseline class and the other is defined as an active class where the animation actually happens. There is a certain notation you need to use to define these classes. The base class is named as `name-event` (ie: `slide-enter`), whereas the active class is named the same with the keyword “active” appended to it (ie: `slide-enter-active`).

For example, using a fade type of effect on an `ngShow` or `ngHide` would require these 4 classes to be defined (since “show” and “hide” are the supported events for that directive):

```
.fade-show {
    -webkit-transition: 400ms cubic-bezier(0.250, 0.250, 0.750, 0.750) all;
    -moz-transition: 400ms cubic-bezier(0.250, 0.250, 0.750, 0.750) all;
    -ms-transition: 400ms cubic-bezier(0.250, 0.250, 0.750, 0.750) all;
    -o-transition: 400ms cubic-bezier(0.250, 0.250, 0.750, 0.750) all;
    transition: 400ms cubic-bezier(0.250, 0.250, 0.750, 0.750) all;
    white-space:nowrap;
    position:relative;
    overflow: hidden;
    text-overflow: clip;
}
.fade-show-active {
    opacity:1;
}
.fade-hide {
    -webkit-transition: 400ms cubic-bezier(0.250, 0.250, 0.750, 0.750) all;
    -moz-transition: 400ms cubic-bezier(0.250, 0.250, 0.750, 0.750) all;
    -ms-transition: 400ms cubic-bezier(0.250, 0.250, 0.750, 0.750) all;
    -o-transition: 400ms cubic-bezier(0.250, 0.250, 0.750, 0.750) all;
    transition: 400ms cubic-bezier(0.250, 0.250, 0.750, 0.750) all;
    white-space:nowrap;
    position:relative;
    overflow: hidden;
    text-overflow: clip;
}
.fade-hide-active {
    opacity:0;
}
```

APPLYING THE CSS NG-ANIMATE

Once you define all of your animations and transitions, you apply them via the `ngAnimate` directive. There are two ways you can describe which CSS classes to use, explicitly or shorthand. Both are shown below:

Explicit:
Here you explicitly name the class to apply for each supported event (enter, leave etc).

```
<div ng-view ng-animate="{enter: 'rotate-enter', leave: 'rotate-leave'}"></div>
```

Shorthand:
The shorthand way of applying them is through a name only, and then that name with the event appended is implied as the class name. So for instance, in the following, the resulting class names implied will be `name-event` or `rotate-enter` and `rotate-leave`, just as above:

```
<div ng-view ng-animate="'rotate'">
```

Check out the demo source or links provided for specific examples to understand the syntax further.

CUSTOMIZING ANIMATION

You can customize your AngularJS animation in different ways using the `$animator` service. You could create your own custom animation events or use the built-in ones like "enter" and "leave" with `ng-animate` in your own custom directive by accessing them off the `$animator` service.

You can also define custom events using the $animator.animate(myEvent,element) function, where `myEvent` is your own String and `element` is what to apply the transition or animation to.

I included an example of defining a custom directive and custom event in the demo application if you are interested in seeing how it works.

RELATED RESOURCES

- Demo Application
- AngularJS 1.1.5 Docs for `ngAnimate`
- nganimate.org – Great samples here
- Really nice detailed article about the new animations support in AngularJS
- AngularJS and Animation Slides with demos by Gias Kay Lee
- Animate your AngularJS apps with CSS3 and jQuery article
- Misko Hevery video talking about the new animation features
- Swipe Demo and code using `ngAnimate`
- Video showing swipe demo and code
- All About CSS3 Transitions
- All About CSS3 Animations

Holly Schinsky

Developer Evangelist

HER BLOG SHARE TWITTER GITHUB

Introduction to Topcoat

by Chris Griffith

by Chris Griffith

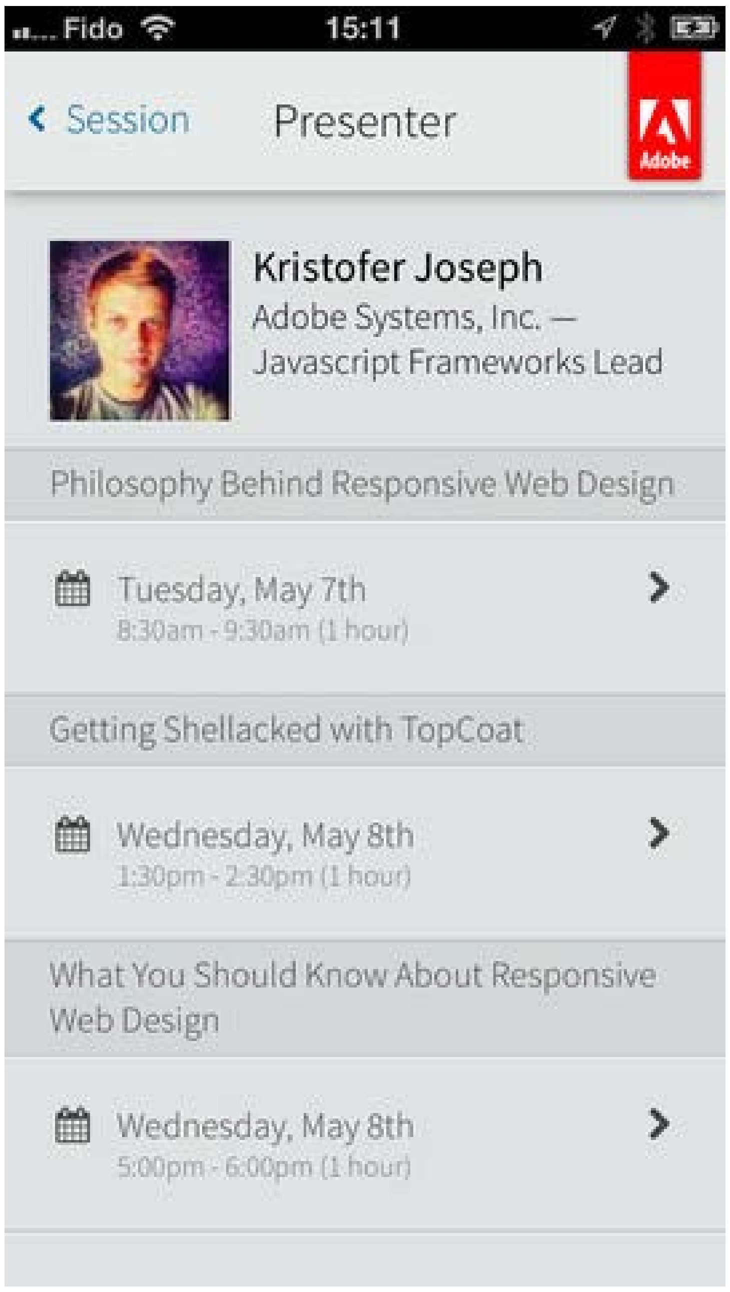
"HTML5 makes life easier for us by defining the right element."

This article first appeared on the Flippin' Awesome website on August 5, 2013. You can read it here.

flippin' awesome!

Topcoat is a brand new open source component library built entirely on web standards (CSS and HTML) and is available at topcoat.io. It was designed to assist developers and designers in creating high-performance web and PhoneGap applications. This project was announced shortly before the 2013 Adobe MAX conference and, in fact, the official 2013 Adobe MAX conference app was built using Topcoat for its user interface.

The Topcoat project originally grew out a need from several teams within Adobe (most notably the Edge Reflow and Brackets teams), as well as feedback from the PhoneGap developer community for component set that was light-weight, fast, and easily themeable. Lead by Kristofer Joseph, along with Garth Braithwaite and Brian LeRoux, this project is being driven by a great group of developers and designers.



GETTING STARTED WITH TOPCOAT

The out of the box installation of Topcoat is incredibly simple:

- Download Topcoat
- Open `index.html` to view the usage guides.
- Copy your desired theme CSS from the css/ folder into your project
- Copy the img/ and font/ folders into your project (eel free to only copy the images and font weights you intend to use)
- Link the CSS into your page. For instance, if you want to use the mobile light theme use:

```
<link  rel="stylesheet" type="text/css"  href="css/topcoat-mobile-light.min.
css">
```

- Done.

However, you probably will want to create a custom build of only the components that your app is using.

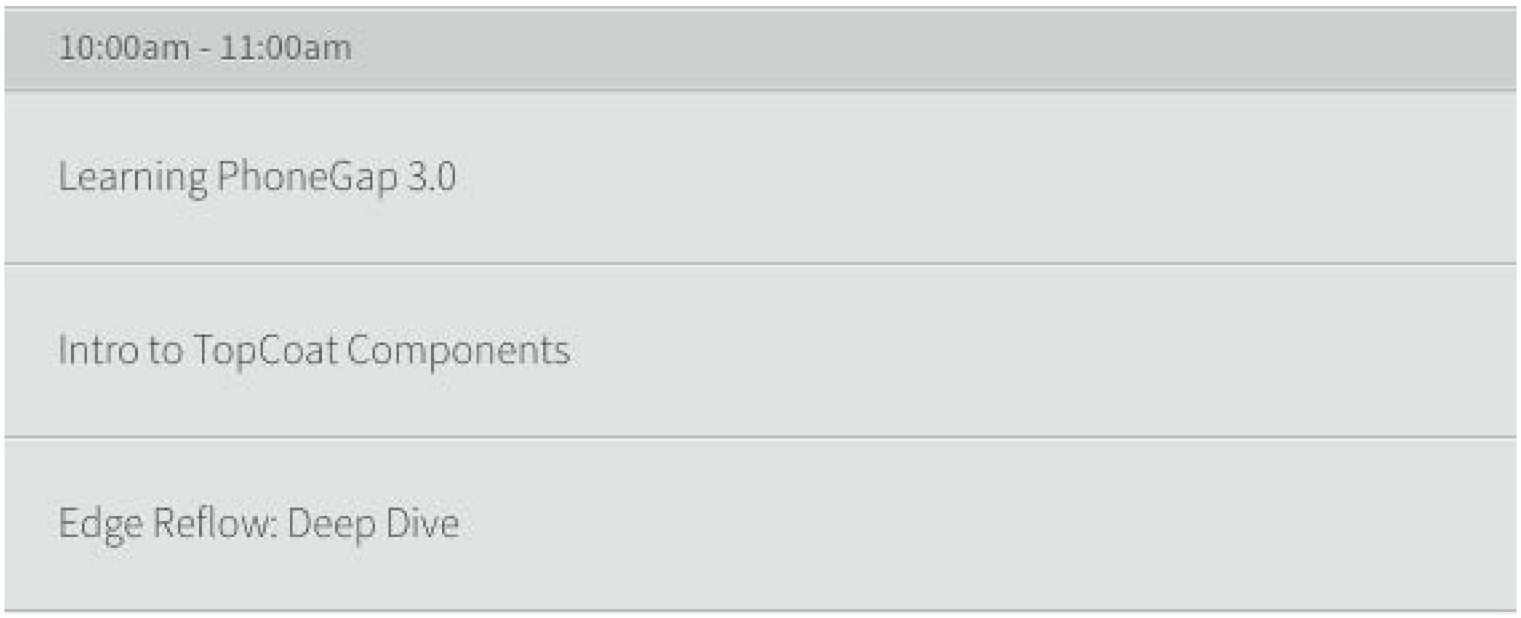
One important thing to remember about Topcoat is that it is not a framework. Topcoat's library can be used along with any JavaScript framework (Backbone, Angular, etc). It just a collection of CSS and HTML.

EXPLORING THE COMPONENTS

As of version 0.5.1, the library consists of:

- Button
- Quiet Button
- Large Button
- Large Quiet Button
- Call To Action Button
- Large Call To Action Button
- Icon Button
- Quiet Icon Button
- Large Icon Button
- Large Quiet Icon Button

- List
- Navigation Bar
- Search Input
- Large Search Input
- Text Input
- Large Text Input
- TopCoat Textarea
- Topcoat Large Textarea



The library ships with both a mobile-friendly set of components, as well as a desktop-friendly version. Each of these versions is also available in both a light or dark style. Each control lives in its own GitHub repository and has no dependencies (not even jQuery), so you can easily assemble just what your app will need (more on that later in this article). When creating mobile apps (or PhoneGap apps) in particular, having lightweight components can greatly improve the performance (and thus the user experience).

As an example, let's look at how to implement a list component:

```
<div class="topcoat-list__container">
    <h3 class="topcoat-list__header">Category</h3>
    <ul class="topcoat-list">
        <li class="topcoat-list__item">
        Item 1
        </li>
        <li class="topcoat-list__item">
        Item 2
        </li>
        <li class="topcoat-list__item">
        Item 3
        </li>
    </ul>
</div>
```

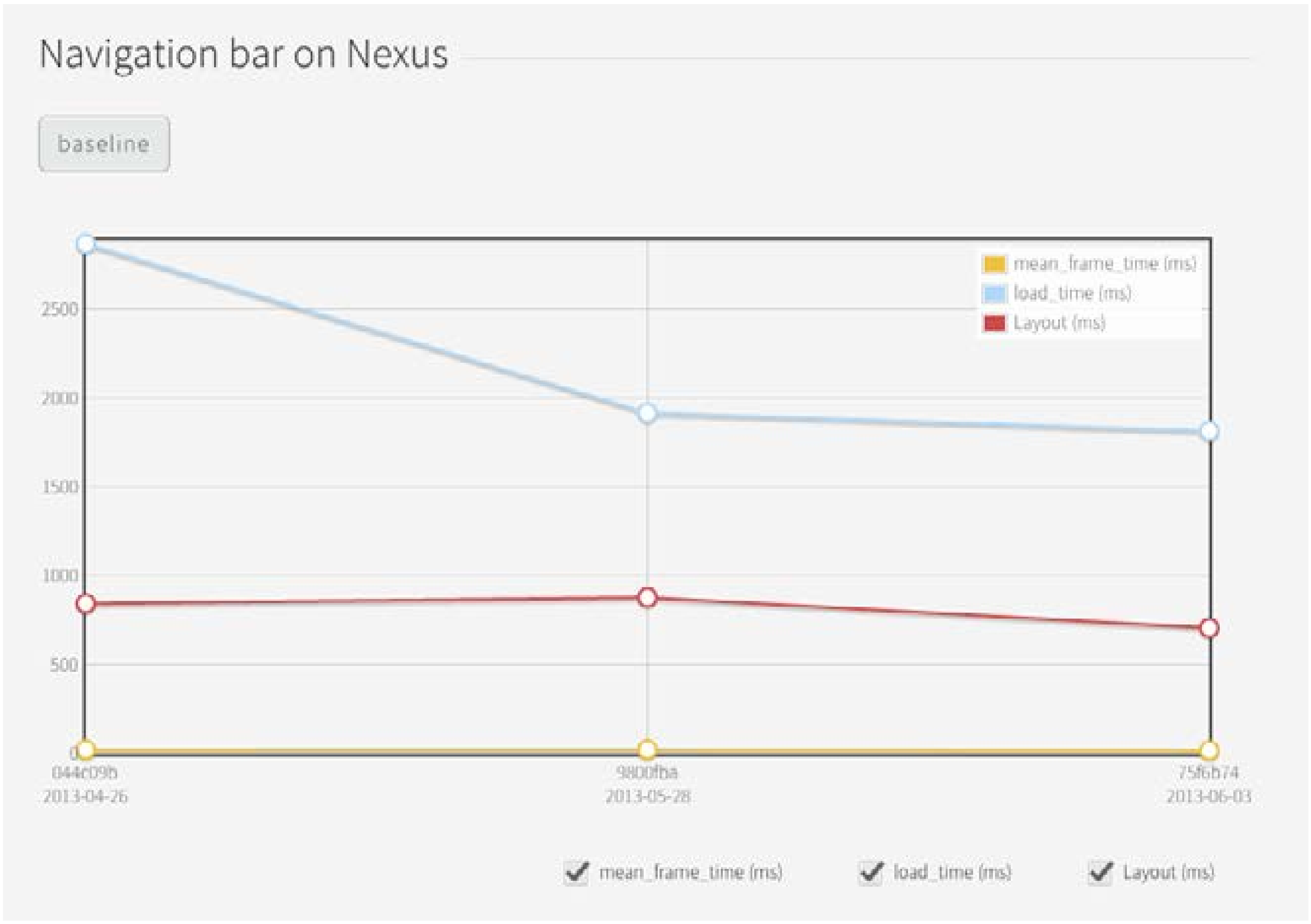
As you can see there is not a lot of additional markup to create our Topcoat list component. A containing `div` and few additional CSS classes on the `ul` and `li` elements. The accompanying CSS is also fairly straightforward. Here is generated CSS for the dark theme:

```
.topcoat-list__container {
  padding: 0;
  margin: 0;
  font: inherit;
  color: inherit;
  background: transparent;
  border: none;
  cursor: default;
  -webkit-user-select: none;
  -moz-user-select: none;
  -ms-user-select: none;
  user-select: none;
  overflow: auto;
  -webkit-overflow-scrolling: touch;
  border-top: 1px solid #2f3234;
  border-bottom: 1px solid #eff1f1;
  background-color: #444849;
}
.topcoat-list__header {
  margin: 0;
  padding: 0.3rem 1.6rem;
  font-size: 0.9em;
  font-weight: 400;
  background-color: #3b3e40;
  color: #868888;
  text-shadow: 0 -1px 0 rgba(0,0,0,0.3);
  border-top: solid 1px rgba(255,255,255,0.1);
  border-bottom: solid 1px rgba(255,255,255,0.05);
}
.topcoat-list {
  padding: 0;
  margin: 0;
  list-style-type: none;
  border-top: 1px solid #2f3234;
  color: #c6c8c8;
}
.topcoat-list__item {
  margin: 0;
  padding: 0;
  padding: 1.16rem;
  border-top: 1px solid #5e6061;
  border-bottom: 1px solid #2f3234;
}
.topcoat-list__item:first-child {
  border-top: 1px solid rgba(0,0,0,0.05);
}
```

The structure of the CSS follows the BEM (Block, Element, Modifier) model. If you are unfamiliar with BEM, take a look at this article. This structure greatly helps in understanding where and how the CSS attributes are being applied to the HTML. The components in Topcoat CSS are authored in Stylus, and utilizes many of its features to allow for a clean separation of resets, from layout, from aesthetic, and between platforms. I personally had not worked with Stylus before, but was able to quickly pick up their stylesheet language.

PERFORMANCE FIRST!

Although, the team has taken great efforts enable easy customization of each component, the heart of the project is the actual performance of each component. The team has set up a complete benchmarking suite to measure the performance.

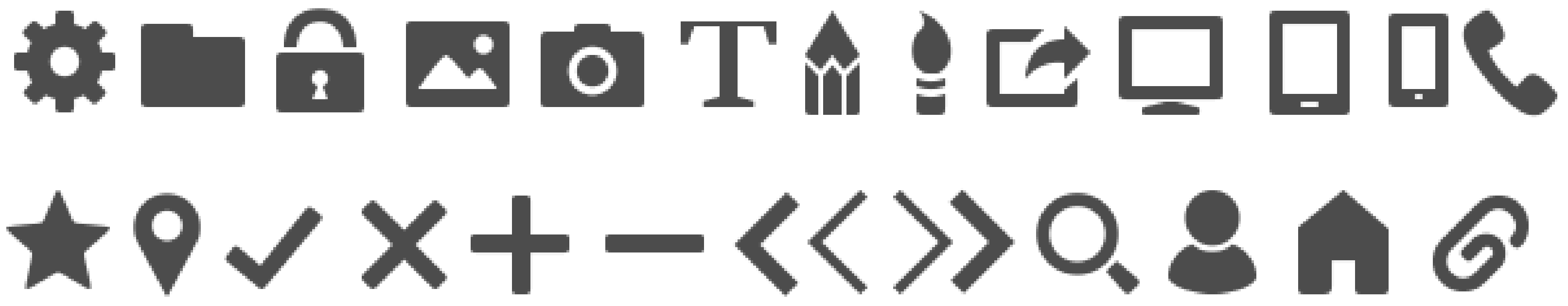


Source: http://bench.topcoat.io/

If fact, this system is also totally open source and can be used as a stand alone tool for teams looking to find ways to measure their own application's performance.

ICONS

What would a component library be complete without an icon set? Topcoat has included them as SVG, PNG or as a semantic icon font. And, yes, they are also open source.



TRANSITIONS AND ANIMATIONS

Recently, the team behind Effeckt.css, a new mobile friendly library for performant transitions and animations, began exploring possible collaboration with the Topcoat team. Components and transitions are tightly coupled from an actual user interface perspective. Often, it is the transitional portion of the component, for example how an overlay appears, that creates the value of the component to the application. This effort just got underway, so keep an eye out for improvements down the road.

CUSTOM BUILDS

Although the minified version of currently clocks in about 13K, you might wish to trim the library down to just the components that your application uses. Since Topcoat is built using Grunt, it is relatively easy to edit the build script to generate your custom library. Here are the steps:

- Fork Topcoat from http://github.com/Topcoat/Topcoat
- Install Node and run `npm install -g grunt-cli` and npm install in the Topcoat directory.
- Modify the package.json to point to only the controls you need
- Run `grunt` to generate your custom build

CUSTOM THEMES

Generating custom themes is also done through the use of Grunt. Here are the steps:

Fork http://github.com/Topcoat/theme
Modify various variables files to make your changes
Modify `./topcoat-X.X.X/package.json` to point to your theme and run `grunt`

WHAT'S NEXT?

Remember, Topcoat is totally open source software. There are bugs, and the team is still solidifying their architecture, so there are many ways to contribute! Here are some additional links to get you started:

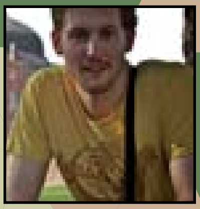
- The main home is http://topcoat.io
- Learn more about Topcoat on the wiki: https://github.com/topcoat/topcoat/wiki
- Get answers via the mailing list: http://groups.google.com/group/topcoat
- Request features and file bugs via the issue tracker. Note each control in Topcoat has its own git repo, thus versions, and therefore issue trackers too. http://github.com/topcoat
- Tweet the project on Twitter: http://twitter.com/topcoat

Chris Griffith
Engineer

HIS BLOG
SHARE
TWITTER
GITHUB

Ractive.js Expressions

by Rich Harris



by Rich Harris

"Dedicated followers of JavaScript fashion will by now have noticed this season's hot new trend."

This article first appeared on the Flippin' Awesome website on August 19, 2013. You can read it here.

flippin' awesome!

Dedicated followers of JavaScript fashion will by now have noticed this season's hot new trend. If you haven't spotted it yet, here are a few projects sporting this style on the GitHub catwalk – React, Reactive.js, component/reactive and reactive.coffee.

That's right: reactive programming is the new black.

At a high level, the idea behind reactive programming is that changes in state propagate throughout a system. Put crudely, this means that in a reactive system where `a = b * 2`, whenever the value of `b` changes, the value of a will also change, rather than forever being equal to whatever `b * 2` was at the time of the statement.

When we take this idea and apply it to user interfaces, we eliminate the DOM manipulation drudgery that dominates web developers' lives.

Ractive.js is a new library initially developed to create interactive (hence the name – not to be confused with Reactive.js!) news applications at theguardian.com. It is designed to dramatically reduce the effort involved in creating web apps by embracing these principles.

Let's look at a simple example:

```
// We create a new ractive, which renders the following to a container element:
// <p>Hello, Dave! You have 4 tasks remaining.</p>

var ractive = new Ractive({
  el: container,
  template: '<p>Hello, {{name}}! You have {{tasks.incomplete}} tasks
remaining.</p>',
  data: {
    user: { name: 'Dave', tasks: { incomplete: 4, total: 11 } }
  }
});

// Later we get some new data:
ractive.set( 'tasks', { incomplete: 5, total: 12 });

// The ractive reacts accordingly, surgically updating the part of the DOM that
is now out of date:
// <p>Hello, Dave! You have 5 tasks remaining.</p>
```

Rather than doing any kind of polling or brute-force 'dirty checking,' this uses an elegant dependency tracking system: the text node containing the number of incomplete tasks depends on the `tasks.incomplete` keypath, which is a child of the `tasks` keypath. So when we update tasks, we know that we need to check to see if `tasks.incomplete` has changed – but we don't need to bother checking `tasks.total`, because nothing depends on that keypath.

As applications grow in complexity, this means much less work for the developer. You might think it sounds like more work for the browser, but it's not. The non-reactive way to do interactive UI typically involves re-rendering views regardless of whether they've changed, and replacing chunks of perfectly good DOM (why hello, garbage collector), which is typically much less efficient.

In other words, reactive UI is a win-win – better for performance, and better for your sanity.

THE SECRET SAUCE: EXPRESSIONS

This article won't go any further into the basics of what Ractive does or why we built it – if you're interested, you can follow the interactive tutorials or read the introductory blog post. Instead, we're going to focus on one of the features that helps Ractive stand out from its peers, namely expressions.

Expressions allow you to take the logic that only your interface cares about, and put it in your template where it belongs. Yes, I just said that! If you've ever had to debug badly written PHP (for example), you may well shudder at the suggestion that logic belongs in templates. But while it's true that business logic doesn't belong in your templates, it's equally true that a lot of presentation logic – aka 'data massaging' – doesn't really belong in your code.

(If you still need convincing, here's a couple of good articles on the subject: The Case Against Logic-less Templates and Cult of Logic-Less Templates.)

Let's take our initial example and turn it into a basic todo app along the lines of TodoMVC. Our template looks like this – ignore the question marks for now:

```
<p>Hello, {{name}}! You have ??? tasks remaining.</p>

<ul>
{{#tasks :i}}
  <li class='task'>{{i}}: {{description}}</li>
{{/tasks}}
</ul>
```

Meanwhile our model, if you want to use MVC terminology, is a simple array of objects representing tasks:

```
tasks = [
  { completed: true,  description: 'Add a task' },
  { completed: false, description: 'Add some more tasks' }.
  { completed: false, description: 'Solve P = NP' }
];

ractive = new Ractive({
  el: container,
  template: template,
  data: { name: 'Dave', tasks: tasks }
});
```

This renders the following:

```
<p>Hello, Dave! You have ??? tasks remaining.</p>

<ul>
  <li class='task'>0: Add a task</li>
  <li class='task'>1: Add some more tasks</li>
  <li class='task'>2: Solve P = NP</li>
</ul>
```

This time, there's no `tasks.incomplete` property, because `tasks` is an array. We'll come back to that. The first job is to rejig the numbers so that it starts at 1, because lists starting with 0 only make sense to programmers. Doing so is trivial:

```
<li class='task'>{{i+1}}: {{description}}</li>
```

Next, let's add a `complete` class to any completed task:

```
<li class='task {{ completed ? "complete" : "pending" }}'>{{i+1}}:
{{description}}</li>
```

Now, our rendered task list looks like this:

```
<p>Hello, Dave! You have ??? tasks remaining.</p>

<ul>
  <li class='task complete'>1: Add a task</li>
  <li class='task pending'>2: Add some more tasks</li>
  <li class='task pending'>3: Solve P = NP</li>
</ul>
```

Now, let's deal with those question marks. One way – the traditional way – would be to keep track of the incomplete count as a separate value in our model (or viewmodel, depending on which tribe you belong to), and update it every time the task list changed. The trouble with that is you have to add the necessary logic to every bit of your app that can change the model in some way – a toggle on each task, a 'mark all as complete' button, the code that reads from the server (or local storage), or whatever else gets added in future. It doesn't scale.

A better way is to have the template react to changes by calling any necessary logic when it needs to:

```
<p>Hello, Dave! You have {{ tasks.filter( incomplete ).length }} tasks
remaining.</p>
```

Then, we just need to add an `incomplete` filter to our model:

```
ractive = new Ractive({
  el: container,
  template: template,
  data: {
    name: 'Dave',
    tasks: tasks,
    incomplete: function ( item ) {
      return !item.completed;
    }
  }
});
```

Now, whenever `tasks` changes – whether because we've added a new one, or changed the status of one or more tasks, or whatever – the expression will be re-evaluated. If the number of incomplete tasks has changed, the DOM will be updated.

As our app becomes more complex, this approach scales beautifully, saving us from a convoluted observing/massaging/updating of our data. You can see a fully fleshed out TodoMVC implementation here – (the source code is possibly the shortest of any implementation, and arguably some of the most readable).

It also allows us to do things like sophisticated animations, without reams of complex render logic.

HOW DOES IT WORK?

Traditional templating engines work by interpolating strings, the result of which is typically rendered using `innerHTML`. Ractive is different – it parses templates into a tree-like JSON structure which contains the DNA of the app, using a PEG-style parser. When it encounters an expression, the parser first creates an abstract syntax tree representation of it, then extracts the references from the AST, then collapses it down to a string representation.

```
Ractive.parse( '{{i+1}}' );

// results in the following - it's deliberately terse, so that
// you can parse on the server and send the result to browsers
// without wasting bytes:
// {
//   t: 2,          // the type of mustache
//   x: {           // the expression
//     r: [ "i" ],  // the references used in the expression
//     s: "${0}+1"  // a reference-agnostic string representation
//   }
// }
```

Later, when Ractive renders this mustache, it will try to resolve the references used in the expression. In this example, there's only one reference – `i` – which resolves to `0` for the first task, then `1`, then `2` and so on. Ractive creates a function – using the Function constructor – that takes the value of `i` as an argument and returns `i+1`.

This might seem like a roundabout way to add 1 to something. But because we only need to create the function once, rather than repeatedly `eval`ing code (which is slow), it's a memory-efficient and performant way to solve the problem even when dealing with hundreds of tasks.

And because we know which references are involved, we know when to call the function again. Consider the next expression:

```
Ractive.parse( ‘{{ completed ? “complete” : “pending” }}’ );

// {
//   t: 2,
//   x: {
//     r: [ “completed” ],
//     s: “${0}?’complete’:’pending’”
//   }
// }
```

For the first task, the `completed` reference resolves to the keypath `tasks.0.completed` – for the second, tasks.1.completed and so on. In each case, Ractive registers the mustache as a dependant of the keypath, so that when `tasks.0.completed` changes (again, it doesn’t matter what happened to cause it to change – that’s the beauty of reactive programming), the expression is re-evaluated and the DOM is updated.

SO IS IT ‘REAL JAVASCRIPT’?

Up to a point. Since in a reactive system we don’t have control over when the expression is evaluated, it’s important that expressions don’t have side effects – so if we try to use assignment operators such as `foo = bar` or `foo += 1`, the parser will fail. The same goes for certain keywords, such as `new`, `delete` and `function`.

Of course, it's still possible to create side-effects by referencing a function which itself has side-effects. Remember our `incomplete` filter? There's nothing to stop you doing this:

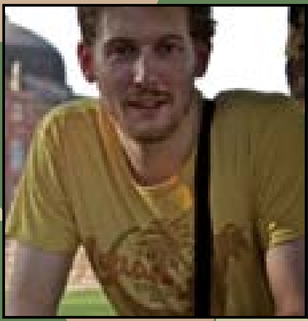
```
ractive = new Ractive({
  el: container,
  template: template,
  data: {
    name: 'Dave',
    tasks: tasks,
    incomplete: function ( item ) {
      doSomeExpensiveComputation();
      counter += 1;
      return !item.completed;
    }
  }
});
```

But by parsing expressions and blocking 'accidental' side-effects, we can encourage best practices without preventing power users from manipulating the system to their own ends.

THE FUTURE

Expect to see more examples of the reactive programming trend coming to a repository near you. As with all programming paradigms, it's not a silver bullet, but by integrating reactive thinking into our work we can start to structure our applications in a way that is more readable and (often) more efficient.

Ractive.js is under active development – it's production-ready (to get started, try the 60 second setup or follow the interactive tutorials), but will continue to evolve as we collectively figure out the secrets of reactive programming. If you're interested in being part of that conversation, come on over to GitHub and help shape the future of web development.



Rich Harris

Web Developer

HIS BLOG
SHARE
TWITTER
GITHUB

5 New Useful Services to Create Web Services

by Anastasia Antonova

by Anastasia Antonova

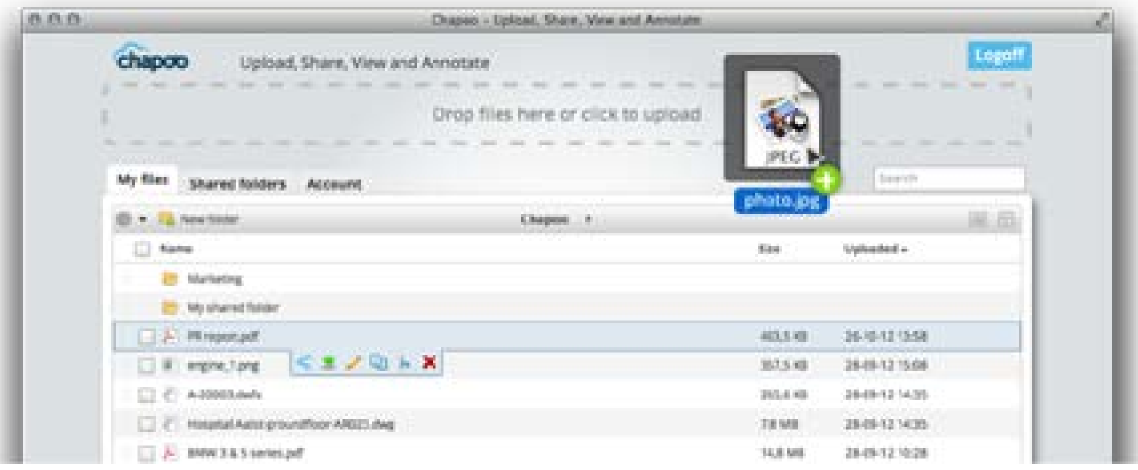
"We have prepared a list of useful, but maybe unknown services."

Web projects are something that many people dream about. There are a lot of guides devoted to the strategy or team building, idea generation, etc. We have prepared a list of useful but maybe unknown services that would help building your own web project.

CHAPOO

From casual file sharing to complete project collaboration

Chapoo - is service that let you upload and share all kinds of documents with friends and colleagues. You can share via Facebook, Twitter or Email. And with other Chapoo users you can also share complete folders and fine-tune access rights. Chapoo supports even CAD files! Service is perfect for collaborative work over the project.


Chapoo - Upload, Share, View and Annotate
chapoo
Upload, Share, View and Annotate
Logoff
Drop files here or click to upload
JPEG
photo.jpg
My files
Shared folders
Account
Search
New folder
Chapoo

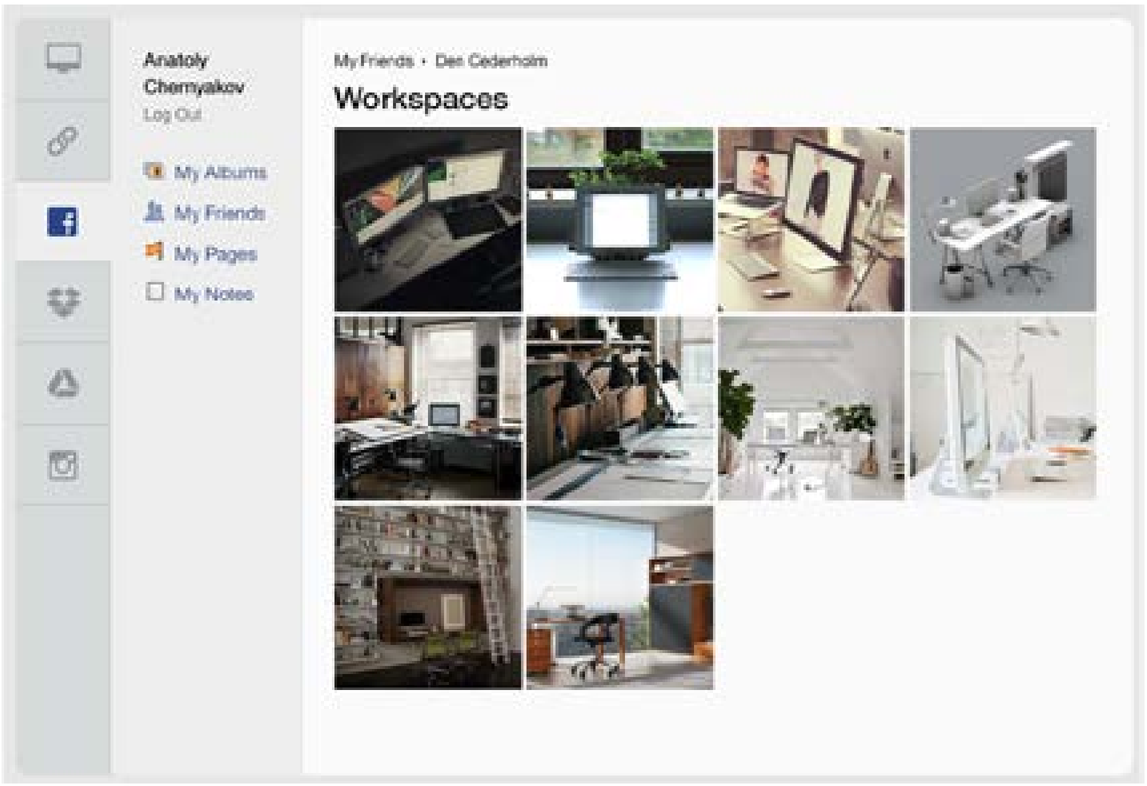
Name	Size	Uploaded
Marketing		
My shared folder		
PR report.pdf	403,5 KB	26-10-12 13:58
engine_1.png	357,5 KB	28-09-12 15:08
A-20003.dwfx	355,4 KB	28-09-12 14:35
Hospital Aalst groundfloor AR025.dwg	7,8 MB	28-09-12 14:35
BMW 3 & 5 series.pdf	14,8 MB	28-09-12 10:28

UPLOADCARE

We handle files, so you don’t have to

Uploadcare helps media creators, businesses and developers store, process and deliver visual content to their clients. Uploadcare seamlessly integrates to allow uploading, editing and posting images to a blog, website or app straight from a computer, a mobile device, Facebook or Instagram account, or a popular cloud solution - like Dropbox or Google Drive. Uploadcare utilizes Amazon S3 file hosting & backup, and Cloudflare Content Delivery Network, with the option to connect any S3/CDN account. Uploadcare takes just 10 minutes to integrate on a variety of platforms.

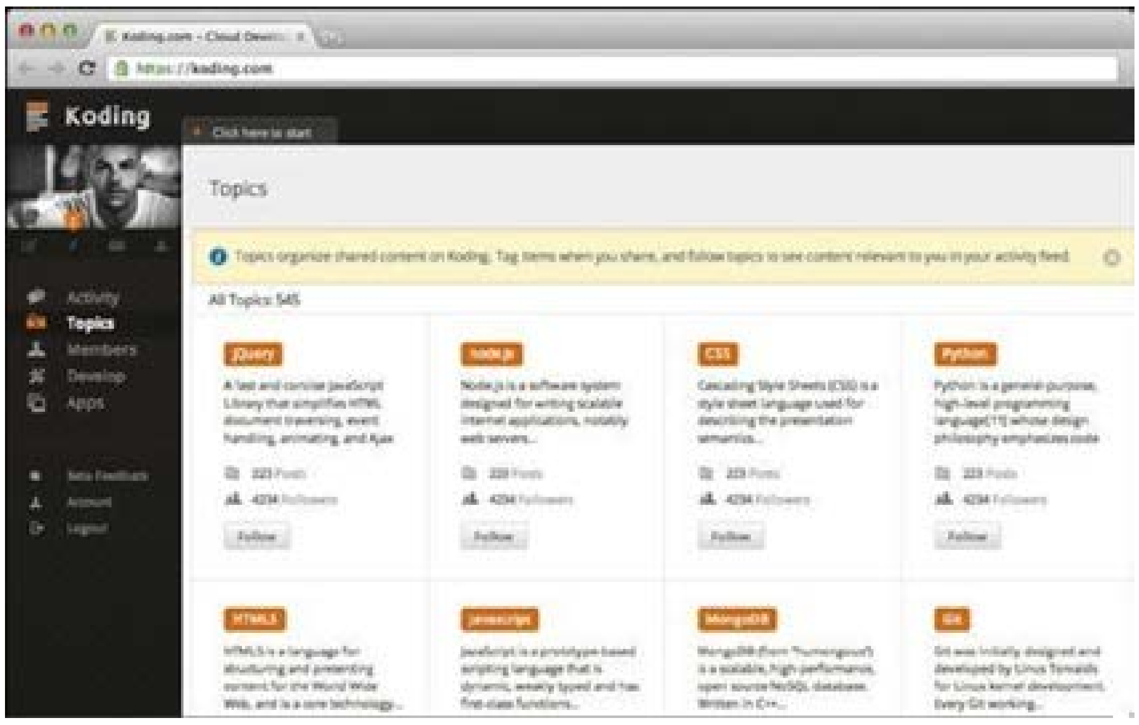
There’s no miles of code, painful integration or costly maintenance. All of Uploadcare’s components - widgets to CDN - work seamlessly together, require almost no configuration and are highly customizable to meet the client’s demands.



KODING

Real software development in the browser

Koding is like new development computer in your browser. which allows software developers to program and collaborate online in the browser without the needs of downloading the software development kits. The platform supports multiple programming languages, including Python, Java, Perl, Node.js, Ruby, C, C++ and Go.



Topics organize shared content on Koding. Tag items when you share, and follow topics to see content relevant to you in your activity feed.

All Topics: 545

A fast and concise JavaScript Library that simplifies HTML document traversing, event handling, animating, and Ajax

323 Posts
4234 Followers

Node.js is a software system designed for writing scalable internet applications, notably web servers...

223 Posts
4234 Followers

Cascading Style Sheets (CSS) is a style sheet language used for describing the presentation semantics...

223 Posts
4234 Followers

Python is a general-purpose, high-level programming language[11] whose design philosophy emphasizes code

223 Posts
4234 Followers

HTML5 is a language for structuring and presenting content for the World Wide Web, and is a core technology...

JavaScript is a prototype-based scripting language that is dynamic, weakly typed and has first-class functions...

MongoDB (from "humongous") is a scalable, high-performance, open source NoSQL database. Written in C++...

Git was initially designed and developed by Linus Torvalds for Linux kernel development. Every Git working...

PROTO.IO

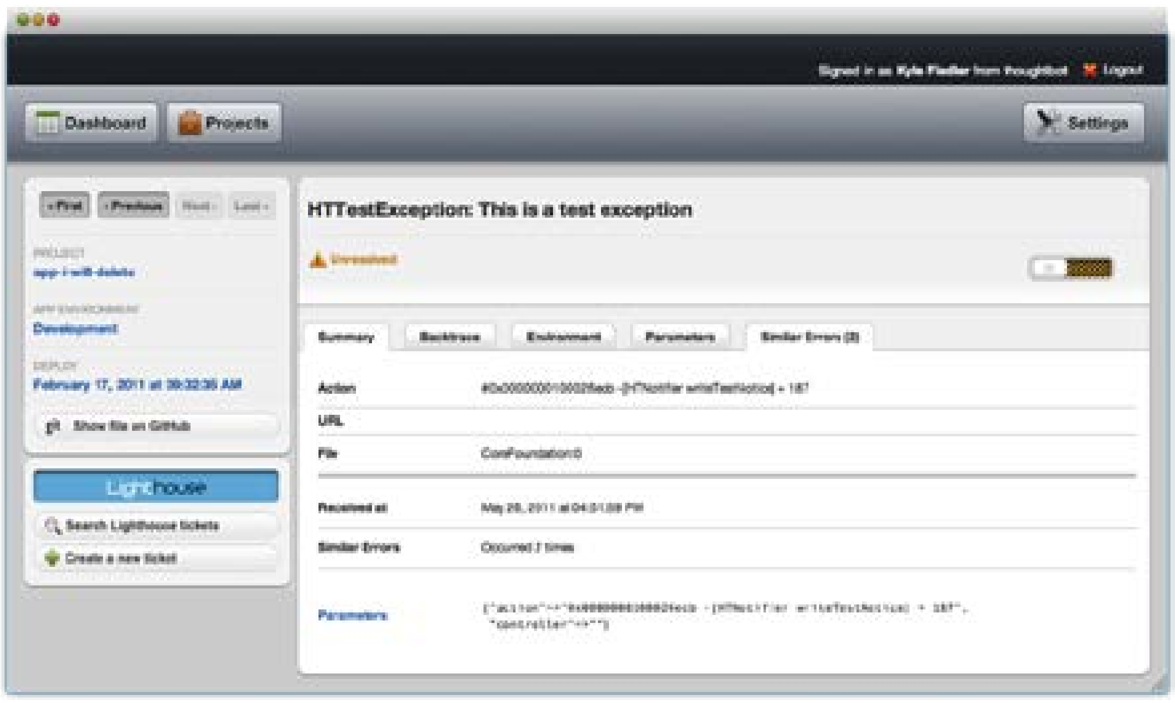
Cool stuff to make your life much easier

Proto - is a mobile prototyping platform to easily build and deploy fully interactive mobile app prototypes and simulations that look and behave exactly like a finished product. It allows users to create realistic, sharable prototypes that work as a real app should and experience their prototype on the actual device. You can actually test any project from UI service to the new car model. Proto.io uses drag and drop interface, hence no coding knowledge are needed.

ROLLBAR

Take control of your errors

Rollbar collects and analyzes errors in web and server applications, so you can find and fix them faster. Rollbar is platform-agnostic and can accept data from anything that can speak HTTP and JSON. You can use official libraries for Ruby,Python, PHP, Node.js, Javascript, or Flash, or roll your own with our API.



HTTestException: This is a test exception

Unresolved

PROJECT
app-i-will-delete

APP ENVIRONMENT
Development

DEPLOY
February 17, 2011 at 39:32:36 AM

Action	#0x00000001000 [illegible] -[HTNotifier writeTestNotice] + 187
URL	
File	CoreFoundation:0
Received at	May 28, 2011 at 04:01:08 PM
Similar Errors	Occurred 2 times
Parameters	{"action"=>"0x0000000100002 [illegible] -[HTNotifier writeTestNotice] + 187", "controller"=>"*"}

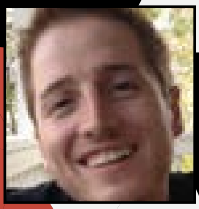
Anastasia Antonova, startuper, PR and marketing specialist. Anastasia is running marketing activities at Whitescape, web-development company, and business development at Feedback Media Agency, social media advertising company.

Anastasia Antonova
Startuper and PR Specialist

HER SITE
SHARE
TWITTER
GITHUB

AngularJS Best Practices

by Jaco Pretorius



by Jaco Pretorius

"Angular suggests that you use a certain directory structure."

Miško Hevery – the father of AngularJS – gave a very interesting presentation on AngularJS best practices in December of last year. This presentation is available on YouTube and the actual slides are on Google Docs. Watch the Youtube video here:

I'm going to cover a few of the points he covered in his presentation and give my take on them.

DIRECTORY STRUCTURE

Angular suggests that you use a certain directory structure, which you can create either using the Angular-seed project or the new Yeoman tool. (Angular-seed was actually created before Yeoman was around so you're probably better off using Yeoman now)

This is called 'Convention over Configuration' – a concept you will hear a lot if you delve into Rails – although this doesn't really apply in Angular as far as I can tell. For example, you still need to manually include all your controllers in an index file (or use something like RequireJS) – if we were in a 'Convention over Configuration' world I would expect a controller to be picked up automatically if I put it in the correct folder.

In any case, it's still a good idea to stick to the conventions simply to make it easier for a developer to look at your codebase and know where files should live. The default conventions also dictate that:

- Everything that gets deployed is in the **app** directory
- Everything outside of the **app** directory is for development

DEPENDENCY INJECTION

Dependency Injection (DI) is built-in with Angular so not only is it best practice to use Dependency Injection it's absolutely necessary (except perhaps for the most rudimentary of applications). It becomes especially important when we delve into unit testing – you can use dependency injection to substitute actual dependencies for mock dependencies to isolate behavior.

I must confess I'm not 100% sure why we need Dependency Injection in Angular. Keep in mind that that Dependency Injection is not a requirement for testability – for example, Rails is perfectly testable and contains no Dependency Injection. I find it rather odd to use Dependency Injection in a dynamic language, but in any case – you definitely should use it in Angular.

HIDING FLASHES OF UNSTYLED CONTENT

When your page first loads you will often see the angular code in the view before Angular is loaded (especially on older browsers) – it will looks something like this (for a moment):

```
{{ interpolation }}
```

We have a few options in order to get around this. One is to use ng-cloak – simply create a style which says

```
[ng-cloak] { display: none; }
```

We usually put this on the body tag, which means the page will only be shown once Angular is loaded. This does means there is a slight delay before anything is rendered.

An alternative is to use ng-bind, so instead of:

```
<p>Hello, {{ user.name }}</p>
```

Use this:

```
<p>Hello, <span ng-bind=”user.name” /></p>
```

What’s rather neat about this approach is that we can set a default value to display, before the browser has loaded Angular.

```
<p>Hello, <span ng-bind=”user.name” >???<span /></p>
```

Keep in mind this is only necessary on your first page (typically index.html) if you’re building a single-page application (SPA).

SEPEARATE BUSINESS AND PRESENTATION LOGIC

One of the ideas behind MVC is the separation of business and presentation logic. Angular helps to accomplish this with the concepts of controllers and services.

Controllers should contain view logic, but shouldn’t actually reference the DOM (referencing the DOM should only be done through directives). The controller should answer questions such as:

- What should happen when the user clicks this button?
- Where do I get the data to display?

Services on the other hand should not contain view logic – services should contain logic independent of the view.

For example, if we are building an email management app, we might have the ability to delete an email. You might even be able to delete an email from multiple places – for example, in a list view as well as when viewing an individual email. So 2 separate controllers might handle the event of the user clicking the ‘delete email’ button, but the logic for deleting an email should actually reside in a service. Likewise, if we need to display a list of emails or an individual email it is the responsibility of the controller to ask a service for this data.

Angular allows the controller to interact with the view through the scope object. While this is a clean separation, it's also pretty easy to make the scope object very messy. There are two general guidelines to stick to with regards to the scope:

- Treat the scope as read-only in views
- Treat the scope as write-only in controllers

These guidelines are very similar to the way you tend to interact with the view in a Rails application – you send variables to the view by writing to local fields – which Rails then magically links into the view – but you would very rarely read from local fields in the controller. In the view you can only read from fields assigned by the controller – there is no way to communicate back to the controller except through an action.

I have found these 2 guidelines to be most useful in my current application. It's still difficult to stick with these guidelines 100% of time, but treating any exception as a code smell is a great way to go.

MORE

There's quite a few topics I haven't covered here – the actual presentation I mentioned covers deployment, minification, etc, but those issues aren't really unique to Angular so I haven't covered them. Do take a look at the original presentation – it's definitely worth your time. Happy coding.

Jaco Pretorius

Programmer

HIS BLOG SHARE TWITTER GITHUB

The flag / g of JavaScript’s regular expressions

by Dr. Axel Rauschmayer

by Axel Rauschmayer

"This describes when and how to use regular expressions whose flag / g is set and what can go wrong."

This article describes when and how to use regular expressions whose flag /g is set and what can go wrong.
(If you want to read a more general introduction to regular expressions, consult [1].)

THE FLAG /G OF REGULAR EXPRESSIONS

Sometimes, a regular expression should match the same string multiple times. Then the regular expression object needs to be created with the flag /g set (be it via a regular expression literal, be it via the constructor RegExp). That leads to the property global of the regular expression object being true and to several operations behaving differently.

```
> var regex = /x/g;
> regex.global
true
```

The property lastIndex is used to keep track where in the string matching should continue, as we shall see in a moment.

REGEXP.PROTOTYPE.TEST(): DETERMINING WHETHER THERE IS A MATCH

Regular expressions have the method

```
RegExp.prototype.test(str)
```

Without the flag /g, the method test() of regular expressions simply checks whether there is a match somewhere in str:

```
> var str = '_x_x';

> /x/.test(str)
true
```

With the flag `/g` set, `test()` returns `true` as many times as there are matches in the string. lastIndex contains the index after the last match.

```
> var regex = /x/g;
> regex.lastIndex
0
> regex.test(str)
true
> regex.lastIndex
2
> regex.test(str)
true
> regex.lastIndex
4
> regex.test(str)
false
```

STRING.PROTOTYPE.SEARCH(): FINDING THE INDEX OF A MATCH

Strings have the method

```
String.prototype.search(regex)
```

This method ignores the properties `global` and `lastIndex` of `regex`. It returns the index where `regex` matches (the first time).

```
> '_x_x'.search(/x/)
1
```

REGEXP.PROTOTYPE.EXEC(): CAPTURING GROUPS, OPTIONALLY REPEATEDLY

Regular expressions have the method

```
RegExp.prototype.exec(str)
```

If the flag /g is not set then this method always returns the match object [1] for the first match:

```
> var str = '_x_x';
> var regex1 = /x/;

> regex1.exec(str)
[ 'x', index: 1, input: '_x_x' ]
> regex1.exec(str)
[ 'x', index: 1, input: '_x_x' ]
```

If the flag /g is set, then all matches are returned – the first one on the first invocation, the second one on the second invocation, etc.

```
> var regex2 = /x/g;

> regex2.exec(str)
[ 'x', index: 1, input: '_x_x' ]
> regex2.exec(str)
[ 'x', index: 3, input: '_x_x' ]
> regex2.exec(str)
null
```

STRING.PROTOTYPE.MATCH():

Strings have the method

```
String.prototype.match(regex)
```

If the flag `/g` of `regex` is not set then this method behaves like `RegExp.prototype.exec()`. If the flag `/g` is set then this method returns all matching substrings of the string (every group 0). If there is no match then `null` is returned.

```
> var regex = /x/g;

    > '_x_x'.match(regex)
    [ 'x', 'x' ]
    > 'abc'.match(regex)
    null
```

REPLACE(): SEARCH AND REPLACE

Strings have the method

```
    String.prototype.replace(search, replacement)
```

If search is either a string or a regular expression whose flag /g is not set, then only the first match is replaced. If the flag /g is set, then all matches are replaced.

```
    > '_x_x'.replace(/x/, 'y')
    '_y_x'
    > '_x_x'.replace(/x/g, 'y')
    '_y_y'
```

THE PROBLEM WITH THE /G FLAG

Regular expressions whose /g flag is set are problematic if a method working with them must be invoked multiple times to return all results. That's the case for two methods:

- `RegExp.prototype.test()`
- `RegExp.prototype.exec()`

Then JavaScript abuses the regular expression as an iterator, as a pointer into the sequence of results. That causes problems:

- You can't inline the regular expression when you call those methods. For example:

```
// Don't do that:
var count = 0;
while (/a/g.test('babaa')) count++;
```

  The above loop is infinite, because a new regular expression is created for each loop iteration, which restarts the iteration over the results. Therefore, the above code must be rewritten:

```
var count = 0;
var regex = /a/g;
while (regex.test('babaa')) count++;
```

  Note: it's a best practice not to inline, anyway, but you have to be aware that you can't do it, not even in quick hacks.

- Code that wants to invoke `test()` and `exec()` multiple times must be careful with regular expressions handed to it as a parameter. Their flag `/g` must be set and it must reset their `lastIndex`.

The following example illustrates the latter problem.

EXAMPLE: COUNTING OCCURRENCES
The following is a naive implementation of a function that counts how many matches there are for the regular expression `regex` in the string `str`.

```
// Naive implementation
function countOccurrences(regex, str) {
    var count = 0;
    while (regex.test(str)) count++;
    return count;
}
```

An example of using this function:

```
> countOccurrences(/x/g, '_x_x')
  2
```

The first problem is that this function goes into an infinite loop if the regular expression's /g flag is not set, e.g.:

```
countOccurrences(/x/, '_x_x')
```

The second problem is that the function doesn't work correctly if regex.lastIndex isn't 0. For example:

```
> var regex = /x/g;
> regex.lastIndex = 2;
2
> countOccurrences(regex, '_x_x')
1
```

The following implementation fixes the two problems:

```
function countOccurrences(regex, str) {
    if (! regex.global) {
        throw new Error('Please set flag /g of regex');
    }
    var origLastIndex = regex.lastIndex;  // store
    regex.lastIndex = 0;

    var count = 0;
    while (regex.test(str)) count++;

    regex.lastIndex = origLastIndex;  // restore
    return count;
}
```

USING MATCH() TO COUNT OCCURRENCES

A simpler alternative is to use `match()`:

```
function countOccurrences(regex, str) {
    if (! regex.global) {
        throw new Error('Please set flag /g of regex');
    }
    return (str.match(regex) || []).length;
}
```

One possible pitfall: `str.match()` returns null if the /g flag is set and there are no matches (solved above by accessing length of `[]` if the result of `match()` isn't truthy).

PERFORMANCE CONSIDERATIONS

Juan Ignacio Dopazo compared the performance of the two implementations of counting occurrences and found out that using test() is faster, presumably because it doesn't collect the results in an array.

ACKNOWLEDGEMENTS

Mathias Bynens and Juan Ignacio Dopazo pointed me to `match()` and `test()`, Šime Vidas warned me about being careful with `match()` if there are no matches.

REFERENCE

[1]JavaScript: an overview of the regular expression API`

Dr. Axel Rauschmayer

JavaScript Trainer & Consultant

HIS BLOG SHARE TWITTER GITHUB

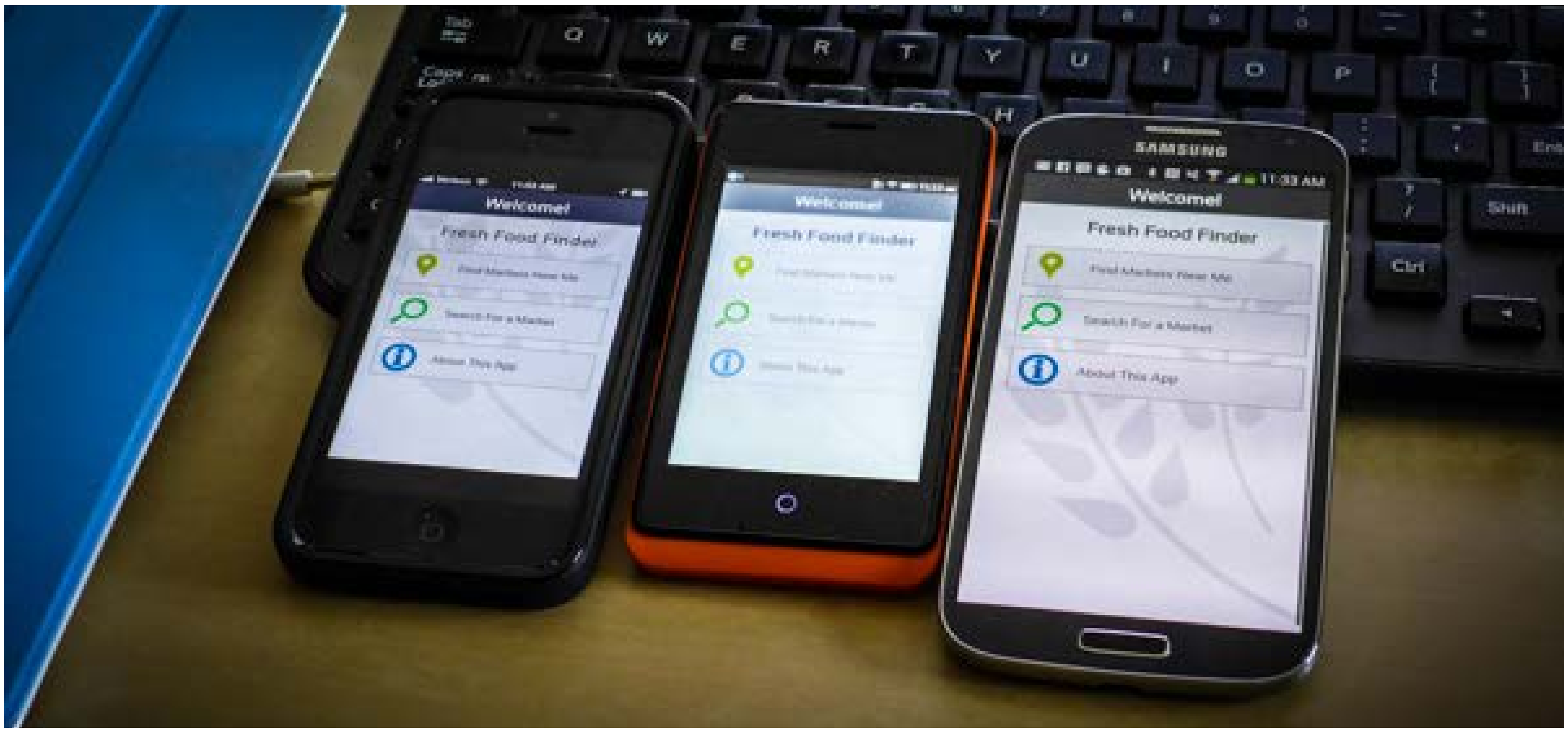
Porting a PhoneGap App to FireFox OS

by Andrew Trice

by Andrew Trice

"PhoneGap support is coming for FireFox OS."

About a year ago I released the Fresh Food Finder, a multi-platform mobile application built with PhoneGap. The Fresh Food Finder provides an easy way to search for farmer's markets near your current location, based on the farmer's markets listings from the USDA. This app has seen a lot of popularity lately, so I'm working on a new version for all platforms with a better data feed, better UI, and overall better UX – unfortunately, that version isn't ready yet. However, I have been able to bring it to an additional platform this week: Firefox OS!



PhoneGap support is coming for Firefox OS, and in preparation I wanted to become familiar with the Firefox OS development environment and platform ecosystem. So… I ported the Fresh Food Finder, minus the specific PhoneGap API calls. The best part *(and this really shows the power of web-standards based development…)* is that I was able to take the existing PhoneGap codebase, and turn it into a Firefox OS app AND submit it to the Firefox Marketplace in **under 24 hours!** If you're interested, you can check out progress on Firefox OS support in the Cordova project, and it will be available on PhoneGap.com once it's actually released.

So, on to the app (you can download the Fresh Food Finder in the Firefox Maketplace here)…

Basically, I commented out the PhoneGap-specific API calls, added a few minor bug fixes, and added a few Firefox-OS specific layout/styling changes (just a few minor things so that my app looked right on the device). Then you put in a mainfest.webapp configuration file, package it up, then submit it to the app store. Check it out in the video below to see it in action, running on a Firefox OS device…

Porting the Fresh Food Finder
PhoneGap app to Firefox OS
The power of w development.

The phone I am using is a Geeksphone Firefox OS developer device. It's not a production/consumer model, so there were a few hiccups using it, but overall it was a good experience. Also, many thanks to Jason Weathersby from Mozilla for helping me get the latest device image running on my phone.

You can learn more about getting started with Firefox OS development here:

- Getting Started
- Firefox OS Simulator
- Packaging Firefox OS Apps
- Submitting to the Firefox Marketplace

Also, be sure to check out the Fresh Food Finder:

- PhoneGap Source
- Firefox OS Source
- Original Blog Entry
- Download for Firefox Market (Open Web App)
- Download for iOS (PhoneGap)
- Download for Android (PhoneGap)

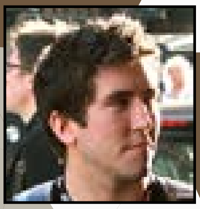
Andrew Trice

Technical Creative Cloud Evangelist

HIS BLOG
SHARE
TWITTER
GITHUB

PouchDB

by Dale Harvey



by Dale Harvey

"We will write a basic Todo web application based on TodoMVC."

GETTING STARTED GUIDE

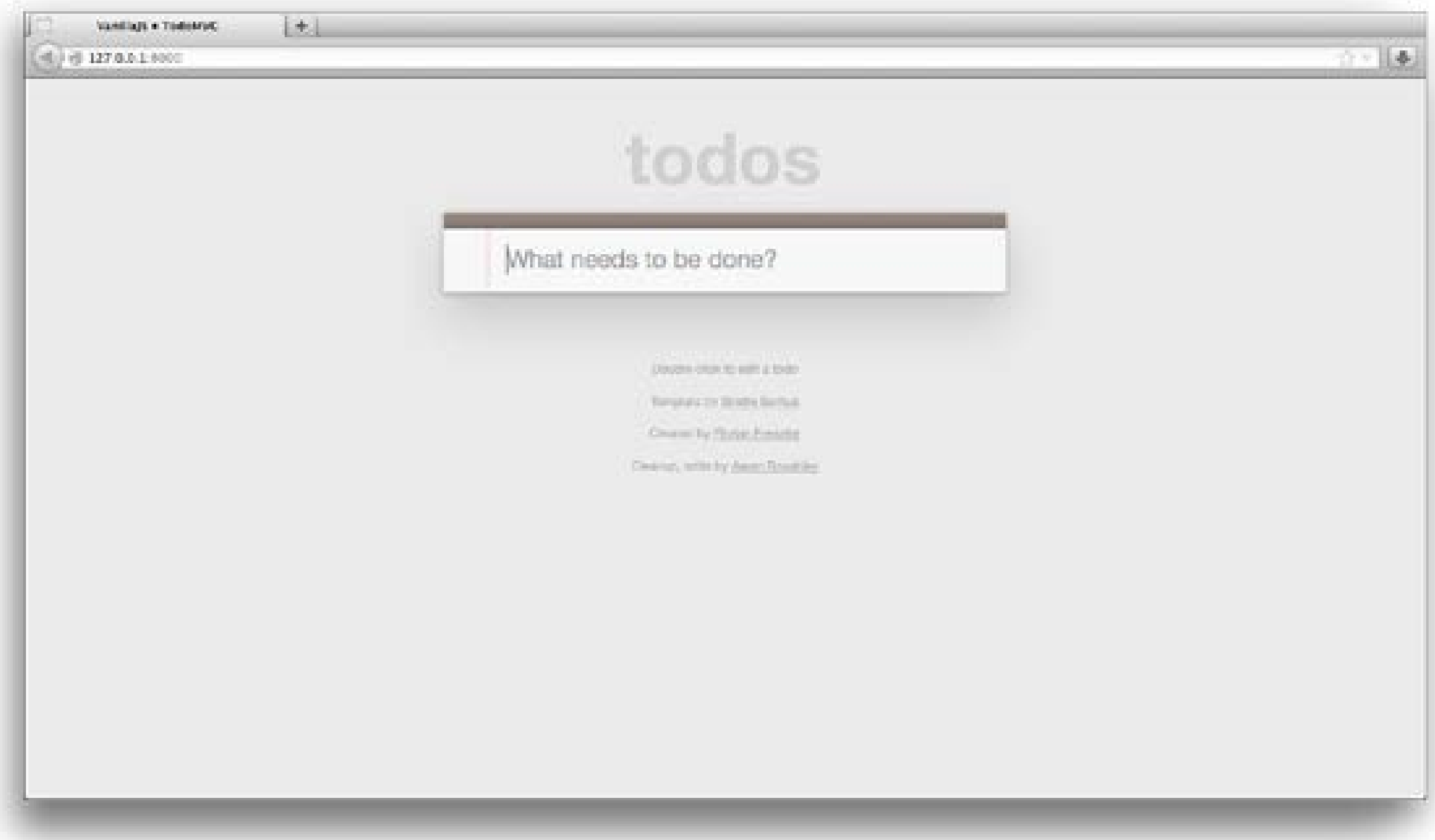
In this tutorial we will write a basic Todo web application based on TodoMVC that syncs to an online CouchDB server. It should take around 10 minutes.

DOWNLOAD ASSETS

We will start with a template of the project where all the data related functions have been replaced with empty stubs, download and unzip pouchdb-getting-started-todo.zip. When dealing with XHR and IndexedDB you are better running web pages from a server as opposed to a filesystem, to do this you can run:

```
$ cd pouchdb-getting-started-todo
$ python -m SimpleHTTPServer
```

Then visit http://127.0.0.1:8000/, if you see the following screenshot, you are good to go:



It wont do anything at this point, but it is a good idea to open your browsers console so you can see any errors or confirmation messages.

INSTALLING POUCHDB

Open `index.html` and include PouchDB in the app by adding a script tag:

```
<script src=”http://download.pouchdb.com/pouchdb-nightly.js”></script>
<script src=”js/base.js”></script>
<script src=”js/app.js”></script>
```

PouchDB is now installed in your app and ready to use (In production you should copy a version locally and use that).

CREATING A DATABASE

The rest of the work will be done inside `app.js`. We will start by creating a database to enter your todos, to create a database simply instantiate a new PouchDB object with the name of the database:

```
// EDITING STARTS HERE (you dont need to edit anything above this line)

var db = new PouchDB(‘todos’);
var remoteCouch = false;
```

You dont need to create a schema the database, you simply give it a name and you can start writing objects to it.

WRITE TODOS TO THE DATABASE

The first thing we shall do is start writing items to the database, the main input will call addTodo with the current text when the user presses Enter, we can complete this function will the following code:

```
function addTodo(text) {
  var todo = {
    _id: new Date().toISOString(),
    title: text,
    completed: false
  };
  db.put(todo, function callback(err, result) {
    if (!err) {
      console.log('Successfully posted a todo!');
    }
  });
}
```

In PouchDB each document is required to have a unique _id, any subsequent writes to a document with the same _id will be considered updates, here we are using a date string as for this use case will be unique and it can also be used to order items by date entered (you can use PouchDB.uuids() or db.post() if you want random ids). The _id is the only thing required when creating a new document, the rest of the object you can create as you like.

The callback function will be called once the document has been written (or failed to write). If the err argument is not null then it will have an object explaining the error, otherwise the result will hold the result.

SHOW ITEMS FROM THE DATABASE

We have included a helper function redrawTodosUI here that takes an array of todos to display so we just need to read them from the database, here we will simply read all the documents using db.allDocs, the include_docs option tells PouchDB to give us the data within each document and the descending option tells PouchDB how to order the results based on their _id field, giving us newest first.

```
function showTodos() {
  db.allDocs({include_docs: true, descending: true}, function(err, doc) {
    redrawTodosUI(doc.rows);
  });
}
```

Once you have included this code you should be able to refresh the page to see any todos you have entered.

UPDATE THE UI

We dont want to refresh the page to see new items, more typically you would update the UI manually when you write data to it, however in PouchDB you may be syncing data remotely and want to make sure you update when the remote data changes to do this we will call `db.changes` which subscribes to updates to the database wherever they come from. You can enter this code between the `remoteCouch` and `addTodo` declaration:

```
var remoteCouch = false;

db.info(function(err, info) {
  db.changes({
    since: info.update_seq,
    continuous: true,
    onChange: showTodos
  });
});

// Show the current list of todos by reading them from the database
function addTodo() {
```

So every time an update happens to the database we will redraw the UI showing the new data, the `continuous` flag means this function will continue to run indefinitely. Now try entering a new todo and it should appear immediately.

EDIT A TODO

When the user checks a checkbox the `checkboxChanged` function will be called so we shall fill in the code to edit the object and call `db.put`:

```
function checkboxChanged(todo, event) {
  todo.completed = event.target.checked;
  db.put(todo);
}
```

This is similiar to creating a document however the document must also contain a `_rev` field (in addition to `_id`) otherwise the write will be rejected, this ensures that you dont accidently overwrite changes to a document.

You can test this works by checking a todo item and refreshing the page, it should stay checked.

DELETE AN OBJECT

To delete an object you can call db.remove with the object.

```
function deleteButtonPressed(todo) {
  db.remove(todo);
}
```

Similiarly to editing a document, both the `_id` and `_rev` properties are required. You may notice we are passing around the full object that we previously read from the database, you can of course manually construct the object like: `{_id: todo._id, _rev: todo._rev}`, passing around the existing object is usually more convenient and less error prone.

COMPLETE REST OF THE TODO UI

`todoBlurred` is called when the user edits a document, here we shall delete the document if the user has entered a blank title or update it if not.

```
function todoBlurred(todo, event) {
  var trimmedText = event.target.value.trim();
  if (!trimmedText) {
    db.remove(todo);
  } else {
    todo.title = trimmedText;
    db.put(todo);
  }
}
```

INSTALLING COUCHDB

Now we will implement the syncing, you need to have an CouchDB instance, you can either install CouchDB(1.3+) locally or use an online provider like IrisCouch.

ENABLING CORS

To replicate directly with CouchDB you need to make sure CORS is enabled, only set the username and password if you have set them previously, by default CouchDB will be installed in "Admin Party" and they are not needed, you will need to replace the `myname.iriscouch.com` with your own host (`127.0.0.1:5984` if installed locally):

```
$ export HOST=http://username:password@myname.iriscouch.com
$ curl -X PUT $HOST/_config/httpd/enable_cors -d '"true"'
$ curl -X PUT $HOST/_config/cors/origins -d '"*"'
$ curl -X PUT $HOST/_config/cors/credentials -d '"true"'
$ curl -X PUT $HOST/_config/cors/methods -d '"GET, PUT, POST, HEAD, DELETE"'
$ curl -X PUT $HOST/_config/cors/headers -d \
  '"accept, authorization, content-type, origin"'
```

IMPLEMENT BASIC TWO WAY SYNC

Now we will have the todo list sync, back to `app.js` we need to specify the address of the remote database, remember to replace `user`, `pass` and `myname.iriscouch.com` with the credentials of your CouchDB instance:

```
// EDITING STARTS HERE (you dont need to edit anything above this line)

var db = new PouchDB('todos');
var remoteCouch = 'http://user:pass@mname.iriscouch.com/todos';
```

Then we can implement the sync function like so:

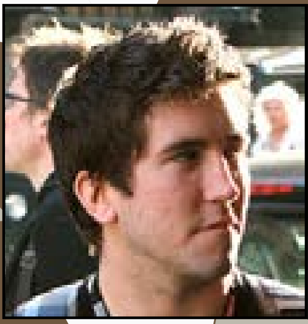
```
function sync() {
  syncDom.setAttribute('data-sync-state', 'syncing');
  var opts = {continuous: true, complete: syncError};
  db.replicate.to(remoteCouch, opts);
  db.replicate.from(remoteCouch, opts);
}
```

`db.replicate()` tells PouchDB to transfer all the documents `to` or `from` the `remoteCouch`, this can either be a string identifier or a PouchDB object. We call this twice, one to receive remote updates and one to push local changes, again the `continuous` flag is used to tell PouchDB to carry on doing this indefinitely. The `complete` callback will be called whenever this finishes, for continuous replication this will mean an error has occured, losing your connection for instance.

You should be able to open the todo app in another browser and see that the 2 lists stay in sync with any changes you make to them, You may also want to look at your CouchDBs Futon administration page and see the populated database.

CONGRATULATIONS!

You completed your first PouchDB application. This is a basic example and a real world application will need to integrate more error checking, user signup etc, however you should now understand the basics you need to start working on your own PouchDB project, if you have any more questions please get in touch on IRC or the mailing list.



Dale Harvey

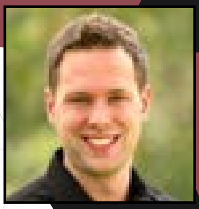
Web Developer

HIS BLOG SHARE

TWITTER GITHUB

Why I test private functions in JavaScript

by Philip Walton



by Philip Walton

"I'm not saying you should always test your private functions."

I published an article on this blog entitled How to Unit Test Private Functions in JavaScript. The article was well received and even mentioned in a few popular newsletters (most notably JavaScript Weekly). Still, a decent amount of the feedback I received in the comments and on Twitter strongly disagreed with my approach. The most common criticism was: unit testing private functions is unnecessary and a mark of bad design.

Admittedly, that article focused too much on the "how" of the technique and not enough on the "why".

In this article I'll try to respond to some of that criticism and go a little deeper into the rationale behind my decisions.

WHAT I'M NOT SAYING

When you don't explain your reasons, it's easy for readers to make incorrect assumptions. I definitely didn't explain my rationale nearly enough in the last article, so I suppose I'm mostly to blame.

To help clarify what I *am* saying, I think it would be helpful to start by pointing out what I *am not* saying:

- I'm not saying you should *always* test your private functions.
- I'm not saying you should test *all* of your private functions.
- I'm not saying people who don't test private functions are doing it wrong.

It's fine if people disagree with me. In fact, I welcome the debate and enjoy hearing counter arguments, especially from developers with a lot more experience. At the end of the day, it's up to each individual to weigh the arguments and make up their own mind.

I just want to make sure we're arguing about the same thing.

DEFINING THE TERMINOLOGY: WHAT DOES PRIVATE MEAN IN JAVASCRIPT?

The title of my previous article used the term "private functions", but if you look at the language used by most of the commenters who disagreed with my approach, they almost all said "private methods".

I'm not pointing this out to be nit-picky or overly technical. I honestly think this fact helps illuminate a disconnect. Perhaps a carryover of general computer science principles applied too liberally to JavaScript.

To avoid confusion, I probably should have titled the article: "How to Test JavaScript Code That's Inside a Closure". After all, that's really what I was talking about.

JAVASCRIPT DOES NOT HAVE PRIVATE METHODS

Wikipedia defines a method in computer science as:

> *Methods define the behavior to be exhibited by instances of the associated class at program run time. Methods have the special property that at runtime, they have access to data stored in an instance of the class (or class instance or class object or object) they are associated with and are thereby able to control the state of the instance.*

By this definition, a method in JavaScript is a function found on the object's prototype chain that can use the `this` context to change or report the state of the instance. But any code with access to an instance variable also has access to that instance's constructor and thus the constructor's prototype.

That means any "methods" on an object must be public to the current scope.

IN JAVASCRIPT, PRIVATE SIMPLY MEANS INACCESSIBLE TO THE CURRENT SCOPE

Private variables and functions in JavaScript aren't just used to modify or report the state of an instance. They do much more. They could be a helper function, a constructor function; even an entire class or module.

In other words, "private" in JavaScript doesn't necessarily mean "implementation detail". Private simply means the code is not accessible in the current scope.

WHY DO DEVELOPERS MAKE CODE PRIVATE IN JAVASCRIPT?

Since JavaScript usually runs in the browser and shares the same global scope as all other JavaScript code on the page, most developers wrap their entire library in a closure and only expose the parts they need to.

This is probably the most common reason JavaScript developers make their code private.

In addition, since browser libraries are under tremendous pressure to keep their file size small and their dependency count low, it's quite common for library authors to roll their own implementations of already solved problems.

Most back-end languages have a standard way of including modules, and there is usually little to no performance cost incurred by including a large module and only using a small portion of it. But this is definitely not the case in the browser. Until there is more standardization and consensus around client-side modules and module dependencies, and until more front-end libraries transition from large monolithic frameworks to smaller, single-purpose modules, most browser library authors will continue to operate in this manner.

For example, I'm not going to make Underscore.js a dependency of my library if all I need is `_.debounce()`. I'll just write my own debounce function. And if I write it myself, I'm probably going to want to test it. However, I'm probably not going to want to expose it globally. I mean, why would I expose it when it's just a simple implementation only meant for my particular use case?

Similarly, if my library requires doing only a few basic DOM operations, I'm not going to require all of jQuery, I'll probably just write the needed functions myself. I might even combine them into their own DOM module that perhaps has its own private functions inside the module closure. I may choose not to test the private functions in the module, but I'll definitely want to test the module itself, regardless of whether I choose to expose it globally.

Hopefully the day will come when we have an elegant solution to library dependency management in the browser, but until that day is here, I'm going to continue to write my own simple implementations rather than force my users to add hundreds of kilobytes to their JavaScript footprint.

In any case, my decision as to whether or not to test these implementations should have no bearing on my decision as to whether or not to expose them globally.

WHAT IS THE PURPOSE OF TESTING?

From one of the comments on my previous post:

> *Unit tests are supposed to test the interfaces to an object without any concern for their implementation.*

I fully agree with this statement. And if you're doing traditional TDD, you'll typically write your unit tests first, before you decide how you'll implement a feature. So for true TDD, it's really not possible to test your implementation details since they're undecided at the time you write your tests.

However, this isn't the only reason I test my code. In addition to testing the public API, unit tests are also very useful for catching regression caused by future code changes.

For example, if a test fails and you're only testing functions in the public API, it may be difficult to find out what's causing the failure. If you only have a few private functions it's probably not a big deal, but if you have several self-contained private modules and numerous helper functions, a failing test might not give you any indication as to where exactly the failure is occurring, and it might take a while to track down.

Again, keep in mind that I'm not just talking about private "methods" within a particular module. I'm talking about any code that you've chosen to keep hidden in a closure and you're testing implicitly through the public API, be it functions, modules, classes — whatever. The more code you keep hidden and don't test explicitly, the harder it is to track down errors when a test fails.

At some point it makes a lot more sense to just test that behavior explicitly.

A REAL-LIFE EXAMPLE

My HTML Inspector tool illustrates a lot of the points I've made in this post. In fact, it's what prompted me to try to figure out ways to test private code in the first place.

HTML Inspector exposes a single object called `HTMLInspector` in the global scope. The `HTMLInspector` object contains a few public functions, but internally there are a number of modules that I've chosen to keep hidden. Here are the three main hidden modules:

- **Listener:** which implements a basic observable pattern and contains the methods `on`, `off`, and `trigger`.
- **Reporter:** stores the errors reported by the rules and contains the methods `warn` and `getWarnings`.
- **Callbacks:** similar to jQuery's callbacks and contains the methods `add`, `remove`, and `fire`.

Each of these modules is my own implementation of objects I've seen in other libraries. I could have merely used the versions in those libraries, but it was quite easy to implement them myself, and doing so greatly reduces the barrier to entry for my users.

I could have also exposed these modules publicly by adding them to the global `HTMLInspector` object (and originally that's what I did), but in the end I concluded it made much more sense to keep them hidden. After all, there's no real use-case for users of the library to access them directly.

Still, I definitely wanted to test each of the public methods listed above, despite the fact that the modules themselves were private.

To help understand the structure of the library, here's a simplified representation. The "include module" comments represent individual JavaScript files that are inserted at that particular place in the code during a build step:

```
(function(root, document) {

  // include module Listener
  // include module Reporter
  // include module Callbacks

  root.HTMLInspector = {
    somePublicMethod: function() {
      // makes calls to the included modules
    },
    someOtherPublicMethod: function() {
      // makes calls to the included modules
    }
  }
}(this, document))
```

As you can see, the above library includes three self-contained modules, each with their own set of methods. But each of those modules is inside the closure and thus hidden from the global scope.

PRIVATE OR PUBLIC

If you were writing tests for HTML Inspector, you'd have to make a choice:

- Keep the modules private, don't test them at all, and hope the tests written for the public API provide enough coverage.
- Expose the modules publicly by storing them on the HTMLInspector object and test their individual functions that way.

Many people argue that if your code is private it doesn't need to be tested, and if it's so complex that it warrants testing, then it should be made public. But to be honest, that logic seems rather circular to me.

The modules either warrant testing or they don't. And they either should be private or they shouldn't. The two issues are orthogonal.

The decision to make functionality public or private should be purely about program design, encapsulation, and a separation of concerns.

Sometimes, due to language constraints, code must be public in order to be tested, but I consider that an entirely different question. Sacrificing API design for testing because of language constraints should be seen as a necessary evil, not as a best-practice.

And taking advantage of tools to mitigate language constraints should be seen as liberating, not as proof of a code smell.

NOT TESTING PRIVATE CODE DOESN'T NECESSARILY MEAN WRITING FEWER TESTS

The HTML Inspector example above has three modules, and each module contains a few functions — eight in total.

If you were unit testing the module functions directly, you'd likely only need eight tests.

However, if you were testing the module functions indirectly through the public API, you may very well need a lot more tests depending on the complexity of those modules and the sheer number of combinations with which their functions may call each other. In addition, those tests may require a lot more setup and tear down just to get to a state where you could even write your assertions.

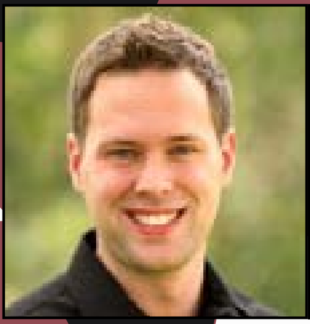
In short, if testing just the public API ends up making the tests longer, more complex, and harder to debug regressions, it's no longer a better option.

WRAPPING UP

In JavaScript, developers often hide a lot more than just "methods". Modules, classes, and even entire programs can all be private. In fact, it's not uncommon to see a library that doesn't expose a single global variable.

Due to the nature of the browser environment, code is kept private for a variety of reasons. It's not always simply because that code is an implementation detail.

As a final thought, I understand (and largely agree with) the aversion to testing private methods and the general desire to avoid over-testing. However, I think it's worth putting the whole issue into perspective. Obviously over-testing is something that is possible, but in the end it does very little harm. At worst, it may waste some time, but I'd far rather err on the side of over-testing than under-testing.



Philip Walton

Software Engineer

HIS BLOG SHARE

TWITTER GITHUB

Why all objects are truthy in JS
by Dr. Axel Rauschmayer

Functional CSS
by Matt Baker

Webkit implemented srcset
by
Mat Marquis

Exploding Blocks with CSS & JS
Johnny Simpson

Kickstart Angular dev with Yeoman, Grunt & Bower
by Brad Barrow

Traceur TodoMVC
by Addy Osmani

Generate CSS Grid with Stylus
by David Walsh

Protecting Objects in JS
by Dr. Axel Rauschmayer

Testing node apps with Jasmine
by Codeship

Detecting CSS Style Support
Ryan Morr