Mobile Web Development for the Modern Coder
Written: 9/18/2013

By: Brian Dainis of Curotec.com

Table of contents

[1] Introduction

The Internet has really come a long way in the last 20 years from its early less mainstream form in the 90's. I remember the days of coding HTML sites in tables and using all those very ugly inline style attributes. As I type this I'm reminded of how thankful I am for CSS! Fast forward to 2013 and the Internet has taken on a whole new shape. We are more connected than ever before with the world's largest information bank in our pockets. It's no wonder why companies and organizations have taken full advantage of this global market place, which has changed the world in more ways than one. Enter the web developer.

As web developers, we are challenged with understanding our client's objectives, creating an easy to use platform to express those objectives and making sure this platform is cross compatible over various devices and browsers. In the past, cross compatibility was not as extraneous as it is now because there were only a handful of browser choices and mobile devices essentially didn't exist at the time. Today we are seeing 5 mainstream browsers with dozens of others including but not limited to Google Chrome, Firefox, Internet Explorer, Opera, Safari, Maxthon, RockMelt and the list continues. But that's not all; don't forget to take into consideration the mobile market. We see devices running OS's such as Android, iOS, Symbian, Blackberry OS and Windows Phone 8. Throughout this article I will be discussing the design, development and implementation of a user friendly website or web app built for mobile devices of the current generation. So bust out your text editor of choice (bonus points if it's vim) and let's dive into some development techniques.

[2] Designing a good user interface

Considering usability

Ok, maybe I lied a little bit. We are not ready to start looking at code yet. Every great website or web application starts with a great UI (User Interface). Every great UI starts with a great design. But it always boils down to one main thing... Usability! You can have the best back-end API and most fancy graphics, but if your UI is too hard to learn or understand, guess who's going to use it? No one!

Usability all starts with understanding your users and how they use your site or app. There are all kinds of research that has been conducted on this. Some of the more studied areas include eye tracking [http://en.wikipedia.org/wiki/Eye_tracking], multivariate testing [http://en.wikipedia.org/wiki/Multivariate_testing] and web navigation [http://en.wikipedia.org/wiki/Web_navigation]. Since this is a little off topic from the focus of this article, which is mobile web development, I will not get too deep into each of those areas I listed, but I encourage you to do some of your own research.

Keeping it simple

KISS -- and not the heavy metal band -- stands for “keep it simple, stupid”. Keeping your UI simple means your users can focus on their needs rather than how they use your interface. In fact, the success or failure of your project will likely be determined by the accessibility of your interface. If you’re stuck and not sure what to do it's always best to consult a friend or co-worker for their blunt opinion on your design. Ask them questions like, “How could I make this interface more user friendly?”

Consider the various screen sizes and resolutions

With so many different devices and screen sizes out there, it's important to test your code on all the popular resolutions. Below is a table showing all the screen resolutions you should be considering.

device screen width	**web site safe area**	
	width	**height**
iphone portrait (320 px)	310 px	352 px
iphone landscape (480 px)	468 px	202 px
ipad portrait (768 px)	750 px	920 px
ipad landscape (1024 px)	1010 px	660 px
screen : 1024 px	989 px	548 px
screen : 1152 px	1117 px	644 px
screen : 1280 px	1245 px	580 px
screen : 1360 px	1325 px	548 px
screen : 1366 px	1331 px	548 px
screen : 1440 px	1405 px	680 px
screen : 1600 px	1565 px	680 px
screen: 1680 px	1645 px	825 px
screen : 1920 px	1885 px	855 px

*Table data from [http://www.websitedimensions.com/]

Simple navigation

When restructuring a website for a smart phone or tablet the first thing you should do is simplify the navigation. Most desktop websites have buttons or links that are too small to click on a smart phone. Also, in many cases there are drop down menus that only appear on hover. There is of course no way to hover on a touch screen device, so we have to restructure the navigation to use a more user-friendly design.

One method includes converting your HTML <ul> and <li> elements to a <select> menu with <option>’s. On a smart phone the user can tap the select box and scroll through the menu options with ease. Another method is to increase the size of the buttons and arrange them vertically. For any drop downs you can have them appear with an accordion effect when the parent button is clicked. This will however restrict you from linking the parent items to another page. You will have to make the parent items dead links.

No Flash

In April 2010 Steve Jobs posted a rant on why flash does not belong in the mobile web space [http://www.apple.com/hotnews/thoughts-on-flash/]. Apple refuses to add support to their devices to run Flash. Some other mobile OS's have moderate or limited support for Flash. However, even devices that do support mobile Flash have issues rendering it. Among other things, Flash consumes excessive power from your device. It's truly built for a PC mouse environment and it falls short with the touch interface. Flash websites are not a good idea for many reasons, but this one is one of the top reasons when cross compatibility is concerned. So skip the Flash and check out some of the awesome things JavaScript, HTML5 and CSS3 can do.

[3] Content and images

Content and images are one of the main things your users come to your site for. As web designers/developers we like to think they are coming for our cool graphics or slick jQuery animations, but the truth is those things are not as important as your actual content. When the space is limited it's important to optimize your UI for content and images. That means remove all those fancy animations and graphics so that the user can focus on consuming your content.

For many websites, mobile users may have a different objective than your desktop users. Let's use the example of ABC Construction, Inc. The desktop users will likely be visiting the site to read about the company, see pictures of what the company has done and fill out a request for quote form. However, the mobile users will likely be looking for a phone number or address because they are on the go. Consider how the different users of your site may consume your content differently depending on the types of devices they are accessing it from.

One thing you may find with images is that on the desktop site they fit fine, but on the mobile site they are a little too big. There is a simple fix for that. You can apply the following CSS to all images or specific images you want to adjust to fit mobile.

```
img{
  max-width:100%;
}
```

[4] Mobile eCommerce

Recent studies have found that mobile ecommerce is growing at twice the rate than traditional ecommerce once grew. Smart phones and tablets accounted for 12% of ecommerce sales in the last quarter of 2012. People used to say that security would be the major consumer concern when deciding if they should make the purchase from their mobile device. However, SSL is just as strong on a mobile connection as it is on an ISP connection.

Amazon and eBay both have great examples of a user-friendly mobile ecommerce experience. The navigation structure is very simple to operate and breaks down the site by category. The home page has a few user recommended products that are related to the user's recent purchases if signed in. The product pages are also very clean with a prominent image gallery and buy now button. All of these factors add up to a good user experience and make the likelihood of them buying from you greatly improved.

[5] Types of mobile sites

Parallel mobile sites

When smart phones and tablets first hit the scene most companies where not too concerned about building for the mobile web. But in the mid 2000's when apple released their iPhone, the more future oriented companies began to realize they had to pay more attention to this whole mobile thing. Many of the mobile site early adopters used the parallel site method to implement their new mobile friendly environment. With the technology available at the time this was the clear best choice.

Essentially, a parallel mobile site uses the User-agent header from the HTTP request to detect at the server-side level what type of device is making the request for a page. If the server sees it's a mobile device it serves up a special version of the site built for mobile devices. Otherwise it will serve the standard desktop version of the site.

To view an example of a parallel mobile site visit [http://ebay.com].

Responsive mobile sites

Responsive websites are the latest craze. They caught on in mid 2012 when CSS3 started gaining traction and since then many people have showcased impressive designs with how they implement their code. The basic idea revolves around making the website layout fluid. Rather than having a static front-end layout that does not change size or arrangement, a responsive website can re-size and shape-shift based on the size of the screen viewing it. It will even do it in live time. You can open up a responsive website and slowly decrease the width of the site to see it re-size before your very eyes.

To view an example of a responsive mobile website, view [http://www.curotec.com].

Dynamic Serving mobile sites

Some developers have found a way to serve up different content and layouts all within the same site. It's called dynamic serving. It works a bit like parallel mobile sites, but instead of sending the user to either a desktop site or a mobile site, a dynamic serving site will actually just send different content and markup based on the user's device. This is great if you only want to worry about managing 1 code base when you're tasked with rolling out new changes for the site.

To view an example of a dynamic serving site, pull up the following URL on a desktop and smart phone side by side [http://gregoryfca.com].

Native mobile applications

There is a very clear distinction between building a mobile website and building a native mobile application. To clear up any confusion I thought it made sense to at least summarize what separates the two. Since this article is not meant to discuss the process of building mobile apps I will not get too far into detail on how they work.

The easiest way to tell the difference between a mobile application and a mobile website is by the way it's accessed. Mobile applications need to be downloaded (typically from a mobile market place on your phones operating system) and installed on your phones SD card or internal storage. Most mobile applications can be accessed even when there is no Internet access unless they require an Internet connection to communicate with an API. Mobile websites, however, are accessed by visiting a web URL in your phones browser. The browser communicates over HTTP to a web server and sends the markup of a website back to your phone for you to view. This cannot be done without an Internet connection.

To view an example of a native mobile application, view [https://itunes.apple.com/us/app/heart-walk/id451276834?mt=8]

Using all the technology together

One thing I cannot stress enough is to think outside of the box when building your mobile website. Some of the most brilliant mobile sites out there combine 2 or 3 of the techniques discussed above to achieve their desired result. The real goal is to always make your interface as simple and user friendly as possible so your users come back later.

[6] Implementation methods

Now that you know the different types of mobile sites lets go over how to actually implement some modern mobile web solutions. In this next section we are going to start diving into some live coding examples.

Parallel Mobile Site

When building a mobile website one of the more traditional methods is to have one site for your desktop users and another site for your mobile users. Whether the two sites are static and independent from one another, or using a server side language to pull the same content from a

shared database, it will not affect our example. Although in this day and age I doubt it makes any sense at all to have a separate desktop and mobile sites that do not share the same data set to some degree. If you are still building this way you may want to look into migrating to a popular content management system framework like WordPress or Joomla to save your site a lot of time in updates. That brings us to server-side device detection.

The way server-side device detection works all starts with the browser on the device that is making the HTTP request. With every HTTP request there is HTTP headers. The HTTP headers carry information about the device and its browser as well as important information needed to process some requests.

Here is an example of how a standard User-Agent string may look (This example is the User-Agent of an iPad):

```
Mozilla/5.0 (iPad; U; CPU OS 3_2 like Mac OS X; en-us) AppleWebKit/531.21.10 (KHTML, like Gecko) Version/4.0.4 Mobile/7B334b Safari/531.21.10
```

Using the information contained in the User-Agent we can do a little magic (well, not quite magic) to redirect the user to a mobile friendly version of the site. Check out this example done in PHP.

```
<?php
  if(strstr($_SERVER['HTTP_USER_AGENT'],'iPhone') || strstr($_SERVER['HTTP_USER_AGENT'],'iPod')) {
    header('Location: http://m.example.com');
    exit();
  }
?>
```

If you don't understand what is happening in this above example, I'll explain. Essentially you would want to put that code on your front controller or main index file. When that file is called by a user request, either the User-Agent of the request matches the 'iPhone' or 'iPod' criteria and is redirected to http://m.example.com or the users request does not match either of those User-Agent strings and it's business as usual. The request then continues routing down the file and loads whatever other standard desktop site resources you have it set to load. You can get a little bit more complicated with your code by setting up an array of acceptable User-Agent strings that should redirect or you could even set up a couple different conditions if you have a desktop site, mobile site and tablet site.

Additionally you could also use your sites .htaccess file for the redirection as well.

```
RewriteCond %{HTTP_USER_AGENT} ^.*iPad.*$
RewriteRule ^(.*)$ http://m.example.com [R=301]
```

Responsive Design

Responsive web development has really been the trend in the last couple years. Many companies don't have the time or resources to build several different websites for the different types of devices that are calling upon them. It may help to relate if you think of responsive design as a Transformer (Yeah, I made a cartoon/movie reference). The concept behind responsive design is to make the website layout fluid depending on the size of the browser viewing it. This way you can setup break points for where you want your website to shift sizes or have the content re-size based on a percentage width value of its environment.

In the past, in order to accomplish this result, you would have to write some pretty complex JavaScript and there was no guarantee it would even work consistently all the time on every device. However in June 2012 media queries

became the W3C recommended standard and have been widely adopted for building mobile websites all around the world. The browser support on media queries is now outstanding and it's safe to use in production!

Here is an example of a media query at work:

```
.box{
  width:960px;
}
@media screen and (max-width:959px) {
  .box{
    width:720px;
  }
}
@media screen and (max-width:719px) {
  .box{
    width:300px;
  }
}
```

Let's recap what is going on here. First, you'll see we set the width of all HTML elements using the class name "box" to 960px wide. Then, you will notice some additional code that you may not recognize in CSS if you are not familiar with media queries. Its starts with @media and then has some parameters after that. Essentially what we are saying in the first media query is if your screen size is 959px wide or less, then load the CSS inside that media query. Then again we override the CSS below that with a second media query. CSS interprets the code from top to bottom and conflicts result in the declaration lowest on the page being the one that gets set to the element. This allows us to construct and style our site first for the standard desktop view, then go back and add break points for each of the different screen sizes we want to target. Simple enough!

Here is another way to implement responsive design in a website without using media queries (although this method is not as powerful as using the media queries method):

```
<!DOCTYPE html>
<html>
<head>
  <title>Test</title>
  <style type="text/css">
    #mainWrapper{
      max-width:1000px;
      min-width: 300px;
      margin:0 auto;
    }
    .insideBox{
      width:100%;
    }
  </style>
</head>
<body>
  <div id="mainWrapper">
    <div class="insideBox">
      Content Here
    </div>
  </div>
</body>
</html>
```

In this above example you see we set the "mainWrapper" div to max-width:1000px and min-width: 300px. Then we set the div inside that wrapper to width: 100%. This allows the content to re-size to the exact size of the screen without getting too large when the screen size gets big and also not too small when the screen size gets really small.

Dynamic Serving

I like to think of dynamic serving as a bit of a hybrid between having a parallel site and having a responsive site. The main difference is that instead of detecting the User-Agent and redirecting to another URL if there is a match, a dynamic serving site will have standard content that loads to both desktop and mobile sites then specific content that only loads to one or the other. It allows you to utilize a little bit of all the technology we have discussed so far. You still have to make use of the media queries for adapting the size and layout of your HTML unless you go as far as serving up different HTML templates.

Here is a basic example of how to implement this strategy:

```
<?php
    $mobile_device = 0;
    if(strstr($_SERVER['HTTP_USER_AGENT'],'iPhone') || strstr($_SERVER['HTTP_USER_AGENT'],'iPod')) {
      $mobile_device = 1;
    }
  ?>
<!DOCTYPE html>
<html>
<head>
  <title>Test</title>
  <style type="text/css">
    #mainWrapper{
      max-width:1000px;
      min-width: 300px;
      margin:0 auto;
    }
    .insideBox{
      width:100%;
    }
  </style>
</head>
<body>
  <div id="mainWrapper">
    <div class="insideBox">
      Content Here
    </div>
    <?php if($mobile_device == 1) : ?>
    <div class="insideBox">
      Mobile Only Content Here
    </div>
    <?php endif; ?>
  </div>
</body>
</html>
```

Here is another method of detecting the device, but rather than looking on the server-side this function checks in the browser using JavaScript:

```
  function detectmob() {
    if( navigator.userAgent.match(/Android/i)
     || navigator.userAgent.match(/webOS/i)
     || navigator.userAgent.match(/iPhone/i)
     || navigator.userAgent.match(/iPad/i)
     || navigator.userAgent.match(/iPod/i)
     || navigator.userAgent.match(/BlackBerry/i)
     || navigator.userAgent.match(/Windows Phone/i)
    ){
    // Do something for mobile devices
```

```
    }else {
  // Do something for non mobile devices
    }
}
```

User initiated methods

This last method is the simplest method to implement. The user initiated method involves building two separate sites (as described in the parallel site example) and giving your user the choice of which version they want to load. Most of the time you could just default the site to loading the desktop version and give the user a link on the page that says “View the mobile friendly version”. When the user clicks that link they are redirected to a sub domain that houses your mobile site. The only way for them to get back from here is to click the link that now says “view the full version of this site”. It’s not the most elegant way to implement a mobile site, but it's sure better than no mobile site at all.

Pros and Cons of each method

Parallel Site Pros – The biggest argument for this method is that both your desktop site content and your mobile site content can be one hundred percent optimized for their respective environments. This is important on weak mobile phone connections where the user is trying to get a fast page load and your site has been optimized to load super fast. It does not restrict you from going all out on your desktop site with all kinds of cool images, jQuery animations and videos that would normally not be ideal for placing on your mobile site version.

Parallel Site Cons – The biggest disadvantage here is that it’s more complex to implement and the cost is generally higher. But if your company has a big budget and is choosing based on performance alone, this method is for you. Another disadvantage here is that in most cases both of your sites cannot share the same URL. Lastly, if you want a site for desktop, mobile, and tablet, you would have to build 3 separate sites. One way around this is to use a hybrid of parallel development and mobile development and make your desktop site responsive to fit better on tablets then have your mobile site for smart phones only.

Dynamic Serving Pros – A big advantage to this over having a parallel site is that both your desktop and mobile friendly site can share the same URL. The other major advantage is that you can display different content to you mobile users that your desktop users are not seeing or vice versa.

Dynamic Serving Cons – As with having a parallel mobile site, having a dynamic serving site is much more complicated to implement and maintain. But, if your company is after full control and performance this may be the way to go for you.

Responsive Design Pros – One advantage commonly overlooked with responsive design is that you have a single URL for your desktop and mobile sites, although that works with dynamic serving as well. Additionally, the entire overall development process is much simpler overall when you choose to build with the responsive design structure. It allows you to build your back end how you want while only focusing on writing some CSS and HTML code to handle the mobile friendly design aspect.

Responsive Design Cons – If you are implementing responsive design after building your desktop site, in some cases you may have to rebuild your existing site although this is not always the case. Another major downfall here is that you have to find a happy medium between your desktop site and mobile site because, no matter what, the same HTML and content is being sent to both, so you can only change its formatting and layout structure. Additionally, to add on to that last point, you may find that since so much HTTP overhead is being sent to mobile devices (objects that are only needed for the desktop site and disabled in the mobile version), it slows down load time and performance, which could be costing you money.

User Initiated Pros – Really the only pro worth mentioning here is that it’s very simple to integrate and does not take much time to implement.

User Initiated Cons – The biggest disadvantage is that this method is not the most user friendly way. Some users do not know what they want and presenting them too many options or hard to find links will confuse them and cause them to hit the back button. Another disadvantage may be that when the page first loads on a phone, it would be the desktop version and it could be hard for the user to find the small link for switching over to the mobile version.

[7] Mobile web frameworks

There are a seemingly endless amount of mobile frameworks out there available to use and many of them are completely

open source. I'll list a few I am familiar with and describe how they work.

Skeleton - [http://www.getskeleton.com/]

Skeleton is a really great framework for building responsive websites. It's based on the 960 grid system which many of you may already be familiar with. Skeleton gives you out of the box 4 break points and allows you to build markup in HTML using their predefined CSS classes. These predefined CSS classes are built to be already responsive so all you have to do is add a little of your own customization in their pre-written media queries to ensure that your sites content can reform to the various break points used in skeleton.

jQuery Mobile - [http://jquerymobile.com/]

jQuery mobile is built using the latest version of the jQuery library combined with jQuery UI. It's a powerful and fully cross compatible mobile web development framework that will work as desired in all major smart phone platforms, tablets and desktop environments.

iUI - [http://www.iui-js.org/]

iUI is a mobile website development framework. If you're looking to build a parallel mobile web structure this may be something worth checking out. This framework's main purpose is to provide rapid mobile site development that has the feel of your native device UI. Users will have the experience they are familiar with already on their native OS while browsing your mobile site.

Wordpress Mobile Theme Switcher - [http://wordpress.org/plugins/wp-mobile-theme-switcher/]

It's a well-known fact that WordPress is one of the top choices for web developers. A few months ago when I checked the latest statistics WordPress powered 17% of the websites on the web. So with these kinds of numbers I figured it would be good to pay some attention to WordPress users in this article. With the WordPress Mobile Theme Switcher, you can develop 2 or more separate themes for your website and use this plugin to set rules for what devices load which themes. It's a high level example of a dynamic site and parallel site hybrid.

PHP Mobile Detect Class - [http://code.google.com/p/php-mobile-detect/]

The PHP Mobile_Detect class is available on GitHub and is a very handy piece of code when you are building your mobile friendly website from scratch using the PHP language. I have used it a few times and it is truly a no guesswork class that is very easy to use. Just include the file and use the is_mobile function along with several other functions included in the class to detect mobile devices.

Ruby mobile-fu - [http://rubygems.org/gems/mobile-fu]

For the Ruby on Rails developers out there this is a gem for you. This gem will automatically detect mobile devices that are accessing your application for all major mobile platforms. It will then change the format of the request from :html to :mobile.

[8] Debugging issues

If you have ever built a mobile website you know that debugging is one of the most difficult tasks. Your site may look great on Android and iOS but looks horrible on Blackberry and Windows Phone. This is because there is no complete standard in the way that mobile browsers interpret HTML, CSS and JavaScript code. As coders, we have all had those nights where we have been coding for hours or days and all of a sudden find this pesky bug that is seemly impossible to shake. In the desktop world we can just use an element inspector or developer tools to help identify the problem and solve it, but it's not so easy on a mobile device. I'll try to make your life a little easier and discuss some popular tools and methods for debugging on the mobile web.

User Agent Switcher

First, one tool that I cannot live without is User Agent Switcher. There are separate extensions available for both Chrome and Firefox. Essentially what it does is spoof the user-agent that gets sent in your HTTP request so the server responds as if it's responding to a mobile device. This allows you to use your desktop browsers native developer tools to dig deeper into the issues. The downside is that you will not always see the same bugs on your desktop as you will on your mobile device even if you're using this nifty tool. Also, keep in mind this tool only works when your building a site that responds to the user-agent string. If your site is responsive you can achieve the same effect on your desktop by simply shrinking the width of your browser window.

IOS Safari debugging console

For you iOS users out there, Apple has inserted an iOS Safari debugging console into your phone. It works similar to a normal developer console and can be activated by going to: Settings => Safari => Developers => Turn On - Debug Console.

Remote Debugging in Chrome using ADB extension

Google is always coming up with sweet new tools for making developers more productive. They did it again with their remote debugging feature. You can now debug issues on your mobile device from your desktop developer tools using the Chrome ADB extension available in the Chrome Web Store. Simply install the extension, enable USB debugging on your device, connect your device to your desktop computer with USB, and your ready to start debugging using the ADB extension!

[9] Improving load times and lowing bandwidth

Page loading time is very important whether you're talking desktop or mobile. Many sites loose users due to slow page loads. It's even more important however to optimize your mobile site's load times because there is already an added lag from the cell tower connection. If you want Google's cold honest opinion on your page load speeds check out this awesome tool [https://developers.google.com/speed/pagespeed/insights/]. Let's talk about how to optimize your site's load times.

Here are some things you can do to optimize your site's performance:

Minify your CSS and JavaScript files - Minifying your files simply means to remove extra white space and smash your code down to lower the size of your files. It also helps to combine all your CSS and JS files into a single file to lower the HTTP responses needed to load each page.

Server caching – There are several ways to cache things at your server level depending on which server you are using. Most websites run on Apache. Using Apache, you have server-side caching options such as mod_cache, mod_file_cache, mod_mem_cache and mod_disk_cache. We're not going to get into mentioning the specifics of each in this article, but I did want to mention that using these techniques could significantly improve your performance. Additionally, if you are using the popular WordPress framework, there is a number of server-side caching plugins available that require minimal configuration out of the box for less technical webmasters.

Browser caching – Browser caching happens by default in most browsers, but there is ways to tweak what gets cached and when to improve your sites performance for you reoccurring visitors. For static resources, I recommend using something like the following on your sites .htaccess file:

```
  ## EXPIRES CACHING ##
<IfModule mod_expires.c>
ExpiresActive On
ExpiresByType image/jpg "access plus 1 year"
ExpiresByType image/jpeg "access plus 1 year"
ExpiresByType image/gif "access plus 1 year"
ExpiresByType image/png "access plus 1 year"
ExpiresByType text/css "access plus 1 month"
ExpiresByType application/pdf "access plus 1 month"
ExpiresByType text/x-javascript "access plus 1 month"
```

```
ExpiresByType application/x-shockwave-flash "access plus 1 month"
ExpiresByType image/x-icon "access plus 1 year"
ExpiresDefault "access plus 2 days"
</IfModule>
## EXPIRES CACHING ##
```

You will have to tweak the above settings for your sites needs, but this should be a pretty good starting point for you. Additionally, the above .htaccess script depends on the module mod_expires to function properly.

Using Compression - If you're using Apache, there are a few efficient ways to compress things like images, code, and text that are being sent from the server to your browser. The two most common modules for this are mod_deflate and mod_gzip. Here is a basic example of what you could add to your .htaccess file to compress objects before they are sent to your browser:

```
# compress text, html, javascript, css, xml:
AddOutputFilterByType DEFLATE text/plain
AddOutputFilterByType DEFLATE text/html
AddOutputFilterByType DEFLATE text/xml
AddOutputFilterByType DEFLATE text/css
AddOutputFilterByType DEFLATE application/xml
AddOutputFilterByType DEFLATE application/xhtml+xml
AddOutputFilterByType DEFLATE application/rss+xml
AddOutputFilterByType DEFLATE application/javascript
AddOutputFilterByType DEFLATE application/x-javascript

# Or, compress certain file types by extension:
<files *.html>
SetOutputFilter DEFLATE
</files>
```

[10] Conclusion

The web is full of wonderful technologies but it is expanding at an ever increasing rate. It seems not long ago that we were building for the first generation of the desktop web. Now with mobile devices, tablets and high resolutions monitors, as a web developer you have so much more to consider.

Thank you for reading this article and I hope it was informative to you. I challenge you to take the things we discussed and apply that knowledge to developing the next generation of the mobile web. As web and application developers, we can truly shape the next generation of technology with our own ideas. To me, knowing that I can shape the future of the web technology industry is the reason I do what I do.

[11] Brian Dainis' bio

Brian Dainis is a web and application developer as well as web entrepreneur from Philadelphia. Acting as the president and founder of Curotec.com, Brian has experience working with web start-ups, companies and organizations of all sizes. Brian and the Curotec.com team focus on website and web application development, mobile application development and many other related web services.

Brian is an expert in front-end web development, server side programming, linux/unix administration and application development. Some of his favorite languages include JavaScript, HTML5, CSS, PHP, Ruby and Python. Frameworks he uses include CodeIgniter, Wordpress, jQuery, Skeleton and Ruby on Rails. If you have any questions for Brian feel free to email him directly: b@br1an.co.