

Behind the Scenes of Real-Life Projects

A large, bold, yellow capital letter 'R' is positioned in the bottom right corner of the image. The background features a network of white lines connecting circular nodes, with several overlapping semi-circular shapes in shades of yellow and white.

Imprint

© 2013 Smashing Magazine GmbH, Freiburg, Germany
ISBN: 978-3-94454027-6
Cover Design: Ricardo Gimenes
eBook Strategy and Editing: Vitaly Friedman
Technical Editing: Cosima Mielke
Planning and Quality Control: Vitaly Friedman, Iris Lješnjanić
Tools: Elja Friedman. Syntax Highlighting: Prism by Lea Verou.
Idea & Concept: Smashing Magazine GmbH

Preface

As Jeremy Olson states in one of the chapters of this eBook, “I believe we learn much more from success than from failure. It took Edison thousands of failed attempts to invent the electric light bulb, and it would be foolish to reinvent it based on trial and error, now that we have a working model.” The same holds true for Web design. After all, is there anything more insightful than learning about the workflows from fellow designers and developers, and what techniques they use? What made their projects a stunning success, and how do they overcome missteps? With this eBook, we’ll take a closer look at the techniques and stories of some folks behind real-life Web projects.

Among others, you will discover how renowned projects such as the Financial Times Web app or the Nike Better World website were built, and learn from the success story of a translation app that made it into Apple’s top ten charts. You’ll also get an insight into Google’s User Experience Lab, and (illustrated by the example of Pinterest) explore the importance of paint performance. Furthermore, our authors share valuable lessons learned in the course of their careers — from both successes and failures. This eBook is full of handy tips, ideas, and personal experiences that are beneficial to any Web professional.

— Cosima Mielke, *Smashing eBook Producer*

TABLE OF CONTENTS

Building The New Financial Times Web App (A Case Study).....	4
Bringing Angry Birds To Facebook	18
Behind The Scenes Of Nike Better World.....	25
Behind The Scenes Of Tourism New Zealand (Case Study).....	46
Tale Of A Top-10 App, Part 1: Idea And Design.....	60
Tale Of A Top-10 App, Part 2: Marketing And Launch.....	79
Gone In 60 Frames Per Second: A Pinterest Paint Performance Case Study	98
Inside Google's User Experience Lab: An Interview With Google's Marcin Wichary	118
Mistakes I've Made (And Lessons Learned Along The Way)	130
About The Authors	139

Building The New Financial Times Web App (A Case Study)

BY WILSON PAGE 🍷

When the mockups for the new Financial Times application¹ hit our desks in mid-2012, we knew we had a real challenge on our hands. Many of us on the team (including me) swore that parts of interface would not be possible in HTML5. Given the product team's passion for the new UI, we rolled up our sleeves and gave it our best shot.

We were tasked with implementing a far more challenging product, without compromising the reliable, performant experience that made the first app so successful.



We didn't just want to build a product that fulfilled its current requirements; we wanted to build a foundation that we could innovate on in the future. This meant building with a maintenance-first mentality, writing clean, well-commented code and, at the same time, ensuring that our code could accommodate the demands of an ever-changing feature set.

In this chapter, I'll discuss some of the changes we made in the latest release and the decision-making behind them. I hope you will come

1. <http://apps.ft.com/ftwebapp/postlaunch.html>

away with some ideas and learn from our solutions as well as our mistakes.

Supported Devices

The first Financial Times Web app ran on iPad and iPhone in the browser, and it shipped in a native ([PhoneGap](http://phonegap.com/)²-esque) application wrapper for Android and Windows 8 Metro devices. The latest Web app is currently being served to iPad devices only; but as support is built in and tested, it will be rolled out to all existing supported platforms. HTML5 gives developers the advantage of occupying almost any platform. With 2013 promising the launch of several new Web application marketplaces (eg. [Chrome Web Store](https://chrome.google.com/webstore)³ and [Mozilla Marketplace](https://marketplace.firefox.com/)⁴), we are excited by the possibilities that lie ahead for the Web.

Fixed-Height Layouts

The first shock that came from the new mockups was that they were all fixed height. By “fixed height,” I mean that, unlike a conventional website, the height of the page is restricted to the height of the device’s viewport. If there is more content than there is screen space, overflow must be dealt with at a component level, as opposed to the page level. We wanted to use JavaScript only as a last resort, so the first tool that sprang to mind was flexbox. Flexbox gives developers the ability to declare flexible elements that can fill the available horizontal or vertical space, something that has been very tricky to do with CSS. Chris Coyier has a [great introduction to flexbox](http://css-tricks.com/old-flexbox-and-new-flexbox/)⁵.

USING FLEXBOX IN PRODUCTION

Flexbox has been around since 2009 and has [great support](http://caniuse.com/flexbox)⁶ on all the popular smartphones and tablets. We jumped at the chance to use flexbox when we found out how easily it could solve some of our complex layouts, and we started throwing it at every layout problem we faced. As the app began to grow, we found performance was getting worse and worse.

We spent a good few hours in Chrome Developers Tools’ timeline and found the culprit: Shock, horror! — it was our new best friend,

2. <http://phonegap.com/>

3. <https://chrome.google.com/webstore>

4. <https://marketplace.firefox.com/>

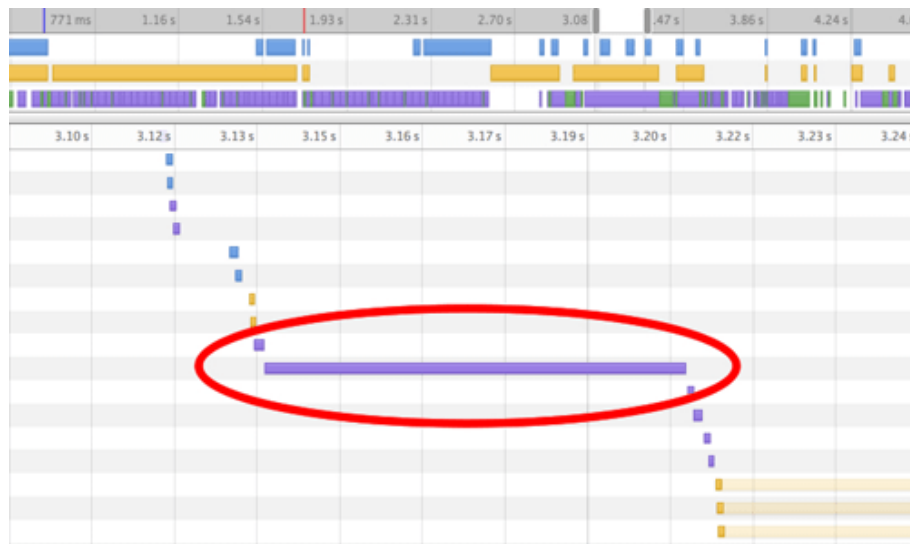
5. <http://css-tricks.com/old-flexbox-and-new-flexbox/>

6. <http://caniuse.com/flexbox>

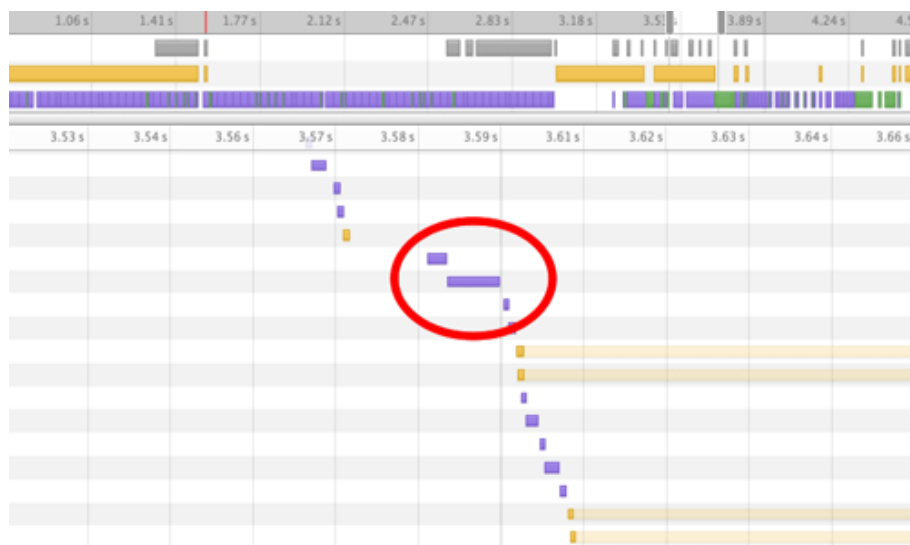
flexbox. The timeline showed that some layouts were taking close to 100 milliseconds; reworking our layouts without flexbox reduced this to 10 milliseconds! This may not seem like a lot, but when swiping between sections, 90 milliseconds of unresponsiveness is very noticeable.

BACK TO THE OLD SCHOOL

We had no other choice but to tear out flexbox wherever we could. We used 100% height, floats, negative margins, border-box sizing and padding to achieve the same layouts with much greater performance (albeit with more complex CSS). Flexbox is still used in some parts of the app. We found that its impact on performance was less expensive when used for small UI components.



Page layout time with flexbox



Page layout time without flexbox

Truncation

The content of a fixed-height layout will rarely fit its container; eventually it has to overflow. Traditionally in print, designers have used ellipses (three dots) to solve this problem; however, on the Web, this isn't the simplest technique to implement.

ELLIPSIS

You might be familiar with the `text-overflow: ellipsis` declaration in CSS. It works great, has [awesome browser support](http://caniuse.com/text-overflow)⁷, but has one shortfall: it can't be used for text that spans multiple lines. We needed a solution that would insert an ellipsis at the point where the paragraph overflows its container. JavaScript had to step in.



Ellipsis truncation is used throughout.

⁷. <http://caniuse.com/text-overflow>

After an in-depth research and exploration of several different approaches, we created our [FTellipsis](http://github.com/ftlabs/ftellipsis)⁸ library. In essence, it measures the available height of the container, then measures the height of each child element. When it finds the child element that overflows the container, it caps its height to a sensible number of lines. For WebKit-based browsers, we use the little-known `-webkit-line-clamp`⁹ property to truncate an element's text by a set number of lines. For non-WebKit browsers, the library allows the developer to style the overflowing container however they wish using regular CSS.

Modularization

Having tackled some of the low-level visual challenges, we needed to step back and decide on the best way to manage our application's views. We wanted to be able to reuse small parts of our views in different contexts and find a way to architect rock-solid styling that wouldn't leak between components.

One of the best decisions we made in implementing the new application was to modularize the views. This started when we were first looking over the designs. We scribbled over printouts, breaking the page down into chunks (or modules). Our plan was to identify all of the possible layouts and modules, and define each view (or page) as a combination of modules sitting inside the slots of a single layout.

Each module needed to be named, but we found it very hard to describe a module, especially when some modules could have multiple appearances depending on screen size or context. As a result, we abandoned semantic naming and decided to name each component after a type of fruit – no more time wasted thinking up sensible, unambiguous names!

An example of a module's markup:

```
<div class="apple">
  <h2 class="apple_headline">{{headline}}</h2>
  <h3 class="apple_sub-head">{{subhead}}</h3>
  <div class="apple_body">{{body}}</div>
</div>
```

An example of a module's styling:

8. <http://github.com/ftlabs/ftellipsis>

9. <http://dropshado.ws/post/1015351370/webkit-line-clamp>


```
.apple {}

.apple_headline {
  font-size: 40px;
}

.apple_sub-head {
  font-size: 20px;
}

.apple_body {
  font-size: 14px;
  column-count: 2;
  color: #333;
}
```

Notice how each class is prefixed with the module's name. This ensures that the styling for one component will never affect another; every module's styling is encapsulated. Also, notice how we use just one class in our CSS selectors; this makes our component transportable. Ridding selectors of any ancestral context means that modules may be dropped anywhere in our application and will look the same. This is all imperative if we want to be able to reuse components throughout the application (and even across applications).

WHAT IF A MODULE NEEDS INTERACTIONS?

Each module (or fruit) has its own markup and style, which we wrote in such a way that it can be reused. But what if we need a module to respond to interactions or events? We need a way to bring the component to life, but still ensure that it is unbound from context so that it can be reused in different places. This is a little trickier than just writing smart markup and styling. To solve this problem, we wrote FruitMachine.

Reusable Components

FruitMachine¹⁰ is a lightweight library that assembles our layout's components and enables us to declare interactions on a per-module basis. It was inspired by the simplicity of Backbone¹¹ views, but with a little more structure to keep “boilerplate” code to a minimum. FruitMa-

10. <http://github.com/ftlabs/fruitmachine>

11. <http://backbonejs.org/>

chine gives our team a consistent way to work with views, while at the same time remaining relatively unopinionated so that it can be used in almost any view.

THE COMPONENT MENTALITY

Thinking about your application as a collection of standalone components changes the way you approach problems. Components need to be dumb; they can't know anything of their context or of the consequences of any interactions that may occur within them. They can have a public API and should emit events when they are interacted with. An application-specific controller assembles each layout and is the brain behind everything. Its job is to create, control and listen to each component in the view.

For example, to show a popover when a component named “button” is clicked, we would not hardcode this logic into the button component. Instead “button” would emit a `buttonClicked` event on itself every time its button is clicked; the view controller would listen for this event and then show the popover. By working like this, we can create a large collection of components that can be reused in many different contexts. A view component may not have any application-specific dependencies if it is to be used across projects.

Working like this has simplified our architecture considerably. Breaking down our views into components and decoupling them from our application focuses our decision-making and moves us away from baking complex, heavily dependent modules into our application.

THE FUTURE OF FRUITMACHINE

FruitMachine was our solution to achieve fully transportable view components. It enables us to quickly define and assemble views with minimal effort. We are currently using FruitMachine only on the client, but server-side (NodeJS) usage has been considered throughout development. In the coming months, we hope to move towards producing server-side-rendered websites that progressively enhance into a rich app experience.

You can find out more about FruitMachine and check out some more examples in the [public GitHub repository](http://github.com/ftlabs/fruitmachine)¹².

¹². <http://github.com/ftlabs/fruitmachine>

Retina Support

The Financial Times' first Web app was released before the age of "Retina" screens. We retrofitted some high-resolution solutions, but never went the whole hog. For our designers, 100% Retina support was a must-have in the new application. We developers were sick of maintaining multiple sizes and resolutions of each tiny image within the UI, so a single vector-based solution seemed like the best approach. We ended up choosing [icon fonts](#)¹³ to replace our old PNGs, and because they are implemented just like any other custom font, they are really well supported. SVG graphics were considered, but after finding a [lack of support](#)¹⁴ in Android 2.3 and below, this option was ruled out. Plus, there is something nice about having all of your icons bundled up in a single file, whilst not sacrificing the individuality of each graphic (like sprites).

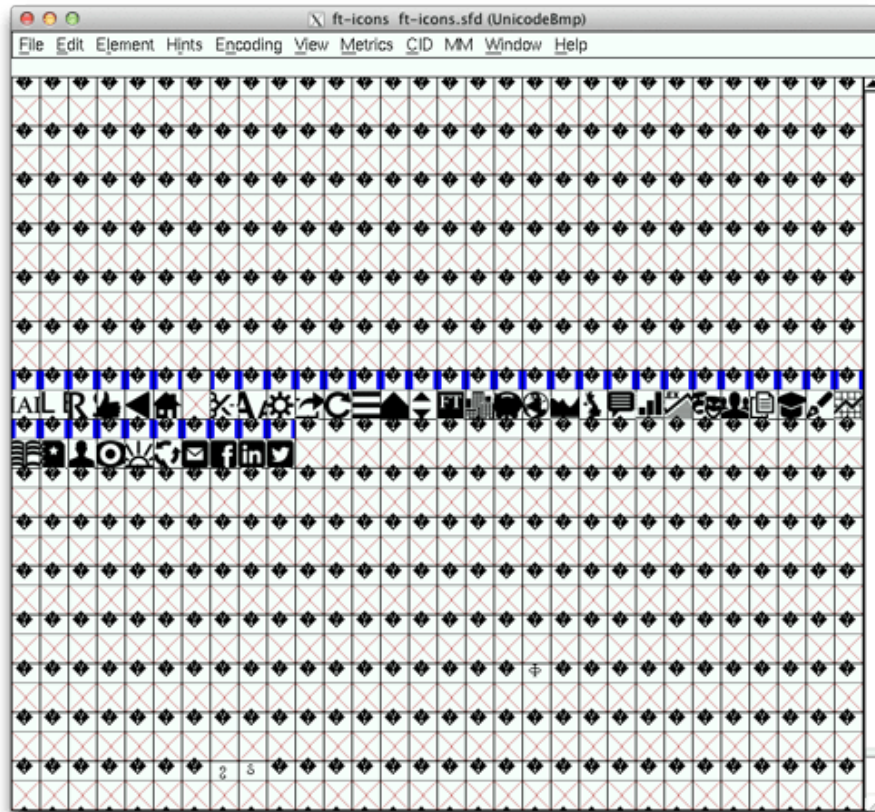
Our first move was to replace the Financial Times' logo image with a single glyph in our own custom icon font. A font glyph may be any color and size, and it always looks super-sharp and is usually lighter in weight than the original image. Once we had proved it could work, we began replacing every UI image and icon with an icon font alternative. Now, the only pixel-based image in our CSS is the full-color logo on the splash screen. We used the powerful but rather archaic-looking [Font-Forge](#)¹⁵ to achieve this.

Once past the installation phase, you can open any font file in Font-Forge and individually change the vector shape of any character. We imported SVG vector shapes (created in Adobe Illustrator) into suitable character slots of our font and exported as WOFF and TTF font types. A combination of WOFF and TTF file formats are required to support iOS, Android and Windows devices, although we hope to rely only on WOFFs once Android gains support (plus, WOFFs are around 25% smaller in file size than TTFs).

¹³. <http://css-tricks.com/examples/IconFont/>

¹⁴. <http://caniuse.com/svg>

¹⁵. <http://fontforge.org/>



The Financial Times' icon font in Font Forge

IMAGES

Article images are crucial for user engagement. Our images are delivered as double-resolution JPEGs so that they look sharp on Retina screens. Our image service (running ImageMagick¹⁶) outputs JPEGs at the lowest possible quality level without causing noticeable degradation (we use 35 for Retina devices and 70 for non-Retina). Scaling down retina size images in the browser enables us to reduce JPEG quality to a lower level than would otherwise be possible without compression artifacts becoming noticeable. [This article](#)¹⁷ explains this technique in more detail.

It's worth noting that this technique does require the browser to work a little harder. In old browsers, the work of scaling down many large images could have a noticeable impact on performance, but we haven't encountered any serious problems.

¹⁶. <http://www.imagemagick.org/>

¹⁷. http://filamentgroup.com/lab/rwd_img_compression/

Native-Like Scrolling

Like almost any application, we require full-page and subcomponent scrolling in order to manage all of the content we want to show our users. On desktop, we can make use of the well-established `overflow` CSS property. When dealing with the mobile Web, this isn't so straightforward. We require a single solution that provides a "momentum" scrolling experience across all of the devices we support.

OVERFLOW: SCROLL

The `overflow: scroll` declaration is becoming usable on the mobile Web. Android and iOS now support it, but only since Android 3.0 and iOS 5. iOS 5 came with the exciting new `-webkit-overflow-scrolling: touch` property, which allows for native momentum-like scrolling in the browser. Both of these options have their limitations.

Standard `overflow: scroll` and `overflow: auto` don't display scroll bars as users might expect, and they don't have the momentum touch-scrolling feel that users have become accustomed to from their native apps. The `-webkit-overflow-scrolling: touch` declaration does add momentum scrolling and scroll bars, but it doesn't allow developers to style the scroll bars in any way, and has limited support (iOS 5+ and Chrome on Android).

A CONSISTENT EXPERIENCE

Fragmented support and an inconsistent feel forced us to turn to JavaScript. Our first implementation used the TouchScroll¹⁸ library. This solution met our needs, but as our list of supported devices grew and as more complex scrolling interactions were required, working with it became trickier. TouchScroll lacks IE 10 support, and its API interface is difficult to work with. We also tried Scrollability¹⁹ and Zynga Scroller²⁰, neither of which have the features, performance or cross-browser capability we were looking for. Out of this problem, FTScroller was developed: a high-performance, momentum-scrolling library with support for iOS, Android, Playbook and IE 10.

¹⁸. <https://github.com/davidaurelio/TouchScroll>

¹⁹. <https://github.com/joehewitt/scrollability>

²⁰. <https://github.com/zynga/scroller>

FTSCROLLER

FTScroller²¹'s scrolling implementation is similar to TouchScroll's, with a flexible API much like Zynga Scroller. We added some enhancements, such as CSS bezier curves for bouncing, `requestAnimationFrame` for smoother frame rates, and support for IE 10. The advantage of writing our own solution is that we could develop a product that exactly meets our requirements. When you know the code base inside out, fixing bugs and adding features is a lot simpler.

FTScroller is dead simple to use. Just pass in the element that will wrap the overflowing content, and FTScroller will implement horizontal or vertical scrolling as and when needed. Many other `options`²² may be declared in an object as the second argument, for more custom requirements. We use FTScroller throughout the Financial Times' Web app for a consistent cross-platform scrolling experience.

A simple example:

```
var container = document.getElementById('scrollcontainer');
var scroller = new FTScroller(container);
```

THE GALLERY

The part of our application that holds and animates the page views is known as the “gallery.” It consists of three divisions: `left`, `center` and `right`. The page that is currently in view is located in the center pane. The previous page is positioned off screen in the left-hand pane, and the next page is positioned off screen in the right-hand pane. When the user swipes to the next page, we use CSS transitions to animate the three panes to the left, revealing the hidden right pane. When the transition has finished, the right pane becomes the center pane, and the far-left pane skips over to become the right pane. By using only three page containers, we keep the DOM light, while still creating the illusion of infinite pages.

²¹. <https://github.com/ftlabs/ftscroller>

²². <https://github.com/ftlabs/ftscroller#options>



Infinite scrolling made possible with a three-pane gallery

Making It All Work Offline

Not many Web apps currently offer an offline experience, and there's a good reason for that: implementing it is a bloody pain! The application cache (AppCache) at first glance appears to be the answer to all offline problems, but dig a little deeper and stuff gets nasty. Talks by [Andrew Betts](http://bdconf.com/ft)²³ and [Jake Archibald](http://www.youtube.com/watch?v=cR-TP6jOSQM)²⁴ explain really well the problems you will encounter. Unfortunately, AppCache is currently the only way to achieve offline support, so we have to work around its many deficiencies.

Our approach to offline is to store as little in the AppCache as possible. We use it for fonts, the favicon and one or two UI images – things that we know will rarely or never need updating. Our JavaScript, CSS and templates live in [LocalStorage](#)²⁵. This approach gives us complete

²³. <http://bdconf.com/ft>

²⁴. <http://www.youtube.com/watch?v=cR-TP6jOSQM>

control over serving and updating the most crucial parts of our application. When the application starts, the bare minimum required to get the app up and running is sent down the wire, embedded in a single HTML page; we call this the preload.

We show a splash screen, and behind the scenes we make a request for the application's full resources. This request returns a big JSON object containing our JavaScript, CSS and [Mustache](#)²⁶ templates. We [eval](#)²⁷ the JavaScript and inject the CSS into the DOM, and then the application launches. This "bootstrap" JSON is then stored in LocalStorage, ready to be used when the app is next started up.

On subsequent startups, we always use the JSON from LocalStorage and then check for resource updates in the background. If an update is found, we download the latest JSON object and replace the existing one in LocalStorage. Then, the next time the app starts, it launches with the new assets. If the app is launched offline, the startup process is the same, except that we cannot make the request for resource updates.

IMAGES

Managing offline images is currently not as easy as it should be. Our image requests are run through a custom image loader and cached in the local database ([IndexedDB](#)²⁸ or [WebSQL](#)²⁹) so that the images can be loaded when a network connection is not present. We never load images in the conventional way, otherwise they would break when users are offline.

Our image-loading process:

1. The loader scans the page for image placeholders declared by a particular class.
2. It takes the `src` attribute of each image placeholder found and requests the source from our JavaScript image-loader library.
3. The local database is checked for each image. Failing that, a single HTTP request is made listing all missing images.
4. A JSON array of Base64-encoded images is returned from the HTTP response and stored separately in the local database.

²⁵. <https://developer.mozilla.org/en-US/docs/DOM/Storage#localStorage>

²⁶. <http://mustache.github.io/>

²⁷. https://developer.mozilla.org/en-US/docs/JavaScript/Reference/Global_Objects/eval

²⁸. <https://developer.mozilla.org/en-US/docs/IndexedDB>

²⁹. <http://www.html5rocks.com/en/tutorials/webdatabase/websql-indexeddb/>

5. A callback is fired for each image request, passing the Base64 string as an argument.
6. An `` element is created, and its `src` attribute is set to the Base64 data-URI string.
7. The image is faded in.

I should also mention that we compress our Base64-encoded image strings in order to fit as many images in the database as possible. My colleague [Andrew Betts](#) goes into detail³⁰ on how this can be achieved.

In some cases, we use this cool trick to handle images that fail to load:

```

```

Ever-Evolving Applications

In order to stay competitive, a digital product needs to evolve, and as developers, we need to be prepared for this. When the request for a re-design landed at the Financial Times, we already had a fast, popular, feature-rich application, but it wasn't built for change. At the time, we were able to implement small changes to features, but implementing anything big became a slow process and often introduced a lot of unrelated regressions.

Our application was drastically reworked to make the new requirements possible, and this took a lot of time. Having made this investment, we hope the new application not only meets (and even exceeds) the standard of the first product, but gives us a platform on which we can develop faster and more flexibly in the future. 🍷

³⁰. <http://labs.ft.com/2012/06/text-re-encoding-for-optimising-storage-capacity-in-the-browser/>

Bringing Angry Birds To Facebook

BY RICHARD SHEPHERD 🐦

There's no avoiding those Angry Birds. They are, quite literally, everywhere: toys, snacks, cartoons, plush toys and that wildly addictive game that seemingly everyone has downloaded at some point — 1 billion of us last year alone.

2012 was another landmark year at the Angry Birds aviary, otherwise known as Rovio³¹. The Finnish-based developer not only released a slew of tie-ins — from Green Day³² to Star Wars³³ — but also went social.

Not only did releasing Angry Birds on Facebook require the game's engine to be rewritten in the all-new Flash 11, but the power of Facebook's platform meant that the game's mechanics could be tweaked further to take advantage of the social graph.



Ville sported a playoff beard during the last few weeks of the project.

³¹. <http://www.rovio.com/>

³². <http://www.billboard.com/articles/news/480358/green-day-partners-with-angry-birds>

³³. <http://www.rovio.com/en/our-work/games/view/50/angry-birds-star-wars>

At the forefront of this project was Ville Koskela³⁴, a lead game programmer at Rovio. Ville has been a developer for his entire adult life, with a background in C++ and ActionScript. And when he's not working on one of the biggest games in the world for the biggest social network in the world, he loves running in and around his home town of Espoo, Finland (he ran a 100-km race in under 12 hours).

I chatted with Ville about bringing Angry Birds to Facebook, and started by asking how he got the job.

I worked as a software architect at Sulake, the company behind the teenage online chat world Habbo Hotel. At Sulake, I had been one of the two guys leading the conversion of Habbo Hotel from Shockwave to the Flash version. I had naturally read about the success of Angry Birds and checked Rovio's website at some point, but back then there was no Web team yet, and even though I had done mobile programming myself, I wasn't too interested in that anymore.

Then, during the summer of 2011, I was contacted through LinkedIn and asked if I was interested in coming in for an interview at Rovio. After hearing what the new job would be about, the decision to join Rovio was a really easy one.

What exactly does his role entail?

As a lead game programmer, I am a supervisor for a team of ten other Flash and C++ programmers, varying from trainees to seniors. I go through job applications, do interviews, and try to make sure all the projects have right set of programmer resources available to launch on schedule – and that they meet our quality expectations. I also still do some (mostly engine) ActionScript programming myself, but mainly try to concentrate on helping others do their programming job the best way possible.

I work closely with the graphic designers, server programmers and producers – basically most of the different people involved in the development of every Angry Birds game.

So, what's it like working at Rovio?

The working environment is really nice, and the atmosphere is relaxed. We try to combine the good parts of traditional planning in advance with agile³⁵ methods to achieve the best possible results. Last

³⁴. <http://villekoskela.org/>

³⁵. http://en.wikipedia.org/wiki/Agile_software_development

but not least, working on something as hot as the Angry Birds brand ensures that everyone is giving their best in the different projects.

Angry Birds started life on the iPhone in December 2009 and has arguably become the biggest mobile franchise on the planet. With success came expansions, spin-offs and ports, and in 2010 Rovio started working on a Facebook version.

In March 2011, InsideMobileApps interviewed Peter Vesterbacka³⁶ (@pvesterbacka³⁷), Chief Marketing Officer and “Mighty Eagle” at Rovio, and asked him why development was taking so long.

You can’t take an experience that works in one environment and one ecosystem and force-feed it onto another. It’s like Zynga. They can’t just take FarmVille and throw it on mobile and see what sticks. The titles that have been successful for them on mobile are the ones they’ve built from the ground up for the platform.



The original release date for Angry Birds on Facebook, now known as Angry Birds Friends, was May 2011. It wasn’t until later that year that Ville joined Rovio, and the port still wasn’t finished.

When I joined the company at the beginning of September 2011, there was a Flash 10 prototype of the original Angry Birds game available that had been done earlier in 2011.

The prototype’s performance was not too good, which was the reason to wait for Flash 11, with the support of GPU-accelerated Stage 3D graphics. Before continuing the development, I was hired both to lead

³⁶. <http://www.insidemobileapps.com/2011/03/13/rovio-angry-birds-facebook/>

³⁷. <https://twitter.com/pvesterbacka>

the new team of Flash developers joining the company and to make sure a fully optimized Flash 11 version of the game was ready for release as soon as possible.

Flash? Yes, Flash. For those of us who have avoided Adobe's waning flagship for the last few months or years, it turns out that it has gotten considerably better. Indeed, Flash 11 is Adobe's attempt to reinvigorate the platform with cutting-edge 2-D and 3-D rendering.

The Stage 3D that Ville mentions is Adobe's very low-level GPU-accelerated APIs. You can hack them directly using the AGAL language or, as Rovio has done, use a framework and ActionScript. For 2-D applications with 3-D acceleration, the open-source Starling Framework³⁸ is the choice of many. But as Rovio was developing Angry Birds for Facebook, the framework still hadn't been completed.

We started the development with an unreleased version of the Starling Framework. Back then, Starling's performance was not too good, so I did lots of optimizations there myself, about which I also wrote on my blog³⁹. Now these optimizations (and also many others) have found their way into the current version of the Starling Framework, so other developers don't have to go through this process themselves anymore.

It soon became clear that we could achieve the desired performance with GPU rendering, but machines with only CPU rendering were not doing too well. To get the game running smoothly on those machines meant we had to cut down some graphical detail – lowering rendering quality and dropping some layers and animations from the background graphics.

Flash 11 and Stage 3D have excited many people, and they represent a lifeline for a product that seemed to be drowning, not waving⁴⁰. I asked Ville what he thought this meant for the future of the platform.

In my opinion, with the release of Flash 11, Adobe has proven that it really wants to provide millions of developers with tools with which you can bring A-quality games to Web. Stage 3D graphics might not be a thousand times faster like some advertisements read, but they are still a lot faster, enabling 60-FPS full-screen gameplay for our 2-D Angry Birds and really high frame rates for some 3-D games out there. The

³⁸. <http://gamua.com/starling/>

³⁹. <http://villekoskela.org/2011/12/12/starling-gets-wings/>

⁴⁰. http://en.wikipedia.org/wiki/Not_Waving_but_Drowning

release of Flash 11 has also made the “Flash is dead” chanters pretty quiet this year.

Or has it? Many proponents of HTML5-based apps are quick to dismiss Flash. And a certain Steve Jobs announced the end of support for it completely on iOS in a well-publicized open letter⁴¹ back in 2010.

Indeed, Google is dropping support for Flash⁴² on Android, and an HTML5 version of Angry Birds is available in the Chrome Web Store⁴³. Even Microsoft is getting in on the act, teaming up with ZeptoLab to create an HTML5 version of Cut the Rope⁴⁴ to show off the latest version of Internet Explorer.



Microsoft used an HTML5 version of Cut the Rope to demo its latest browser.

With all of the trash talk about HTML5 versus Flash, does Ville think comparing the two is even fair?

Here at Rovio, we do not have “this technology” versus “that technology” thinking. If a new platform has the capabilities and a big enough audience, there’s a good chance that you will see Angry Birds there. I believe that all the technologies have their strengths, and when there’s competition, each participant will gain from that and become better in the long run.

⁴¹. <http://www.apple.com/hotnews/thoughts-on-flash/>

⁴². <http://www.theverge.com/2012/6/29/3125219/flash-mobile-android-4-1-not-supported>

⁴³. <http://chrome.angrybirds.com/>

⁴⁴. <http://www.cuttherope.ie/>

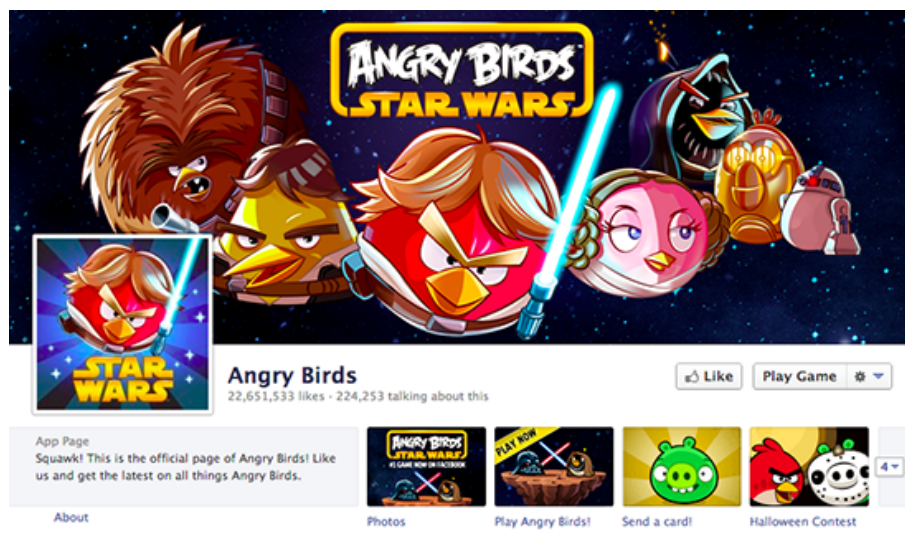
As Peter Vesterbacka mentions in his interview with InsideMobileApps, a successful game needs to be built from the ground up for each platform. This presented an opportunity for Rovio to tweak the tried-and-trusted Angry Birds formula for a social environment.

I asked Ville just how far they pushed the envelope.

Bringing the game to Facebook naturally added the social aspect. We tried to think of how players would like to either compete against or support their friends. Our game designer spent time playing other successful Facebook games and thinking about what ideas would work in Angry Birds and what wouldn't. We came up with the crown concept; the three best players in your friends network would have crowns – gold, silver and bronze – for each of the levels, so that there is continual competition to be the best player on every single level.

The biggest change we introduced to the actual game was the power-ups, with which the players can achieve even higher scores on the levels. The power-ups provide a way to monetize the game since they are available as an in-application purchase; but they also help with retention since players who keep returning to the game get free power-ups, and it's also possible to gift your friends with those.

Those two magic words, “social” and “monetize.” This is perhaps one of the main draws of Facebook for Rovio. And with over 23 million likes already, it is clearly delivering on the promise that the Angry Birds brand makes.



But with so many platforms, so many crossovers and licensing opportunities, what is now driving development for the team? Is it new game features? Or is it new content, like the recent tie-in with Green Day? Just how far can these birds fly?

The guys making decisions [about licensing opportunities] go through the pros and cons in each individual case both before and after the project. All of the Angry Birds fans out there should be pretty confident that the brand will stay strong, and new and interesting things will keep coming at a steady pace.

As the Angry Birds empire continues to grow — and it certainly shows no signs of stopping — people such as Ville are the ones keeping the brand and experience consistent across all of the platforms that we now use.

We can only hope that the continued development of new angles and spin-offs — like Bad Piggies — do not dilute the experience that many of us fell in love with. There can be too much of a good thing, after all.

For games developers and designers, Angry Birds Friends also reminds us that Flash still has a prominent place in the market. By finally harnessing the power of the GPU, Adobe has opened up the floodgates to 3-D games that look and play just like their PC and console equivalents. The big question is whether it's too little, too late.

Ultimately, the power now lies with Android and iOS, because the desktop market — and, thus, the market for Flash apps — is shrinking. But by covering all of its bases, Rovio has made sure that Angry Birds will be flying in and out of our lives for years to come. 🐷

Resources

- [Angry Birds on Facebook](#)⁴⁵
- [Ville Koskela](#)⁴⁶
- “Rovio Has Been Working on the Facebook Version of Angry Birds for a Year⁴⁷,” Kim-Mai Cutler, [InsideMobileApps](#)
- [The Starling Framework](#)⁴⁸
- [Stage 3D](#)⁴⁹ for Flash
- [Rovio](#)⁵⁰

⁴⁵. <https://www.facebook.com/angrybirds>

⁴⁶. <http://villekoskela.org/>

⁴⁷. <http://www.insidemobileapps.com/2011/03/13/rovio-angry-birds-facebook/>

⁴⁸. <http://gamua.com/starling/>

⁴⁹. <http://www.adobe.com/devnet/flashplayer/stage3d.html>

⁵⁰. <http://www.rovio.com>

Behind The Scenes Of Nike Better World

BY RICHARD SHEPHERD 🐶

Perhaps one of the most talked about websites has been Nike Better World⁵¹. It's been featured in countless Web design galleries, and it still stands as an example of what a great idea and some clever design and development techniques can produce.

In this chapter, we'll talk to the team behind Nike Better World to find out how the website was made. We'll look at exactly how it was put together, and then use similar techniques to create our own parallax scrolling website. Finally, we'll look at other websites that employ this technique to hopefully inspire you to build on these ideas and create your own variation.

Nike Better World



Nike Better World is a glimpse into how Nike's brand and products are helping to promote sports and its benefits around the world. It is a web-

⁵¹. <http://www.nikebetterworld.com>

site that has to be viewed in a browser (preferably a latest-generation browser, although it degrades well) rather than as a static image, because it uses JavaScript extensively to create a parallax scrolling effect.

A good deal of HTML5 is used to power this immersive brand experience and, whatever your views on Nike and its products, this website has clearly been a labor of love for the agency behind it. Although parallax scrolling effects are nothing new, few websites have been able to sew together so many different design elements so seamlessly. There is much to learn here.

AN “INTERACTIVE STORYTELLING EXPERIENCE”

In our opinion, technologies are independent of concept. Our primary focus was on creating a great interactive storytelling experience.

— Wieden+Kennedy

Nike turned to long-time collaborator Wieden+Kennedy (W+K), one of the largest independent advertising agencies in the world, to put together a team of four people who would create Nike Better World: Seth Weisfeld⁵² was the interactive creative director, Ryan Bolles produced, while Ian Coyle⁵³ and Duane King⁵⁴ worked on the design and interaction.



I started by asking the team whether the initial concept for the website pointed to the technologies they would use. As the quote above reveals,

⁵². http://twitter.com/#!/seth_weisfeld

⁵³. <http://www.iancoyle.com/>

⁵⁴. <http://twitter.com/#!/duaneking>

they in fact always start by focusing on the concept. This is a great point. Too many of us read about a wonderful new technique and then craft an idea around it. W+K walk in the opposite direction: they create the idea first, and sculpt the available technologies around it.

So, with the concept decided on, did they consciously do the first build as an “HTML5 website,” or did this decision come later?

There were some considerations that led us to HTML5. We knew we wanted to have a mobile- and tablet-friendly version. And we liked the idea of being able to design and build the creative only once to reach all the screens we needed to be on. HTML5 offered a great balance of creativity and technology for us to communicate the Nike Better World brand message in a fresh and compelling way.

— W+K

HTML5 is still not fully supported in all browsers (read “in IE”) without JavaScript polyfills, so just how cross-browser compatible did the website have to be?

The primary technical objectives were for the site to be lightweight, optimized for both Web and devices, as well as to be scalable for future ideas and platforms.

— W+K

To achieve these goals, the website leans on JavaScript for much of the interactivity and scrolling effects. Later, we’ll look at how to create our own parallax scrolling effect with CSS and jQuery. But first, we should start with the template and HTML.

THE STARTING BOILERPLATE

It’s worth pointing out the obvious first: Nike Better World is original work and should not be copied. However, we can look at how the website was put together and learn from those techniques. We can also look at other websites that employ parallax scrolling and then create our own page, with our own code and method, and build on these effects.

I asked W+K if it starts with a template.

We started without a framework, with only reset styles. In certain cases, particularly with experimental interfaces, it ensures that complete control of implementation lies in your hands.

— W+K

If you look through some of the code on Nike Better World, you'll see some fairly advanced JavaScript in a class-like structure. However, for our purposes, let's keep things fairly simple and rely on HTML5 Boilerplate as our starting point.

Download HTML5 Boilerplate⁵⁵. The "stripped" version will do. You may want to delete some files if you know you won't be using them (*crossdomain.xml*, the test folder, etc.).

As you'll see from the source code (see the final code below), our page is made up of four sections, all of which follow a similar pattern. Let's look at one individual section:

```
<section class="story" id="first" data-speed="8"
data-type="background">
  <div data-type="sprite" data-offsetY="950" data-Xposition="25%"
data-speed="2"></div>
  <article>
    <h2>Background Only</h2>
    <div>
      <p>Pellentesque habitant morbi tristique senectus et netus et
malesuada fames ac turpis egestas. Vestibulum tortor quam,
feugiat vitae, ultricies eget, tempor sit amet, ante. Donec eu
libero sit amet quam egestas semper. Aenean ultricies mi vitae
est. Mauris placerat eleifend leo.</p>
    </div>
  </article>
</section>
```

I'm not sure this is the best, most semantic use of those HTML5 tags, but it's what we need to make this effect work. Normally, a section has a heading, so, arguably, the section should be a div and the article should be a section. However, as W+K points out, the HTML5 spec is still young and open to interpretation:

HTML5 is still an emerging standard, particularly in implementation. A lot of thought was given to semantics. Some decisions follow the HTML5 spec literally, while others deviate. As with any new technology, the first to use it in real-world projects are the ones who really shape it for the future.

— W+K

⁵⁵. <http://html5boilerplate.com/>

This is a refreshing interpretation. Projects like Nike Better World are an opportunity to “reality check” an emerging standard and, for the conscientious among us, to provide feedback on the spec.

In short, is the theory of the spec *practical*? W-K elaborates:

We use the article tag for pieces of content that can (and should) be individually (or as a group) syndicated. Each “story” is an article. We chose divs to wrap main content. We took the most liberty with the section tag, as we feel its best definition in the spec is as chapters of content, be it globally.

— W+K

As an aside (no pun intended!), HTML5 Doctor has begun a series of mark-up debates called Simplequizes⁵⁶, which are always interesting and illustrate that there is rarely one mark-up solution for any problem. Make sure to check them out.

In *style.css*, we can add a background to our section with the following code:

```
section { background: url(../images/slide1a.jpg) 50% 0 no-repeat
fixed; }
```

We should also give our sections a height and width, so that the background images are visible:

```
.story { height: 1000px; padding: 0; margin: 0; width: 100%;
max-width: 1920px; position: relative; margin: 0 auto; }
```

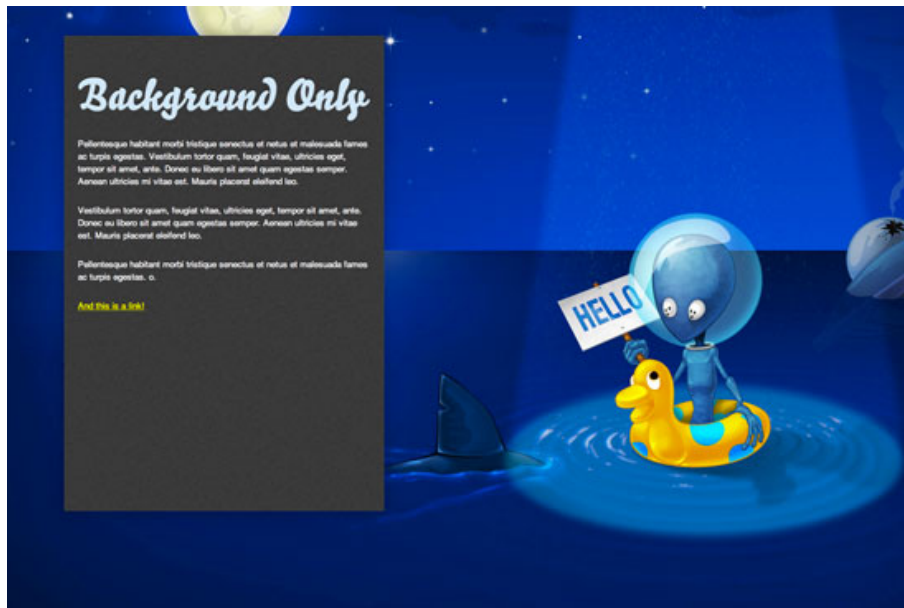
I’ve set the height of all our sections to 1000 pixels, but you can change this to suit your content. You can also change it on a per-section basis.

We have also made sure that the maximum width of the section is the maximum width of the background (1920 pixels), and we have specified a relative position so that we can absolutely position its children.

Here’s the page before adding JavaScript⁵⁷. It’s worth digging into the source code to see how we’ve duplicated the sections in the HTML and CSS.

⁵⁶. <http://html5doctor.com/html5-simplequiz-6-zeldmans-fat-footer/>

⁵⁷. <http://www.richardshepherd.com/smashing/parallax/background.html>



Even with this code alone, we already have a pleasing effect as we scroll down the page. We're on our way.

THE HTML5 DATA ATTRIBUTE

Before looking at parallax scrolling, we need to understand the new **data** attribute, which is used extensively throughout the HTML above.

Back in the good old days, we would shove any data that we wanted to be associated with an element into the **rel** attribute. If, for example, we needed to make the language of a story's content accessible to JavaScript, you might have seen mark-up like this:

```
<article class='story' id="introduction" rel="en-us"></article>
```

Sometimes complex DOM manipulation requires more information than a **rel** can contain, and in the past I've stuffed information into the **class** attribute so that I could access it. Not any more!

The team at W+K had the same issue, and it used the **data** attribute throughout Nike Better World:

*The **data** attribute is one of the most important attributes of HTML5. It allowed us to separate mark-up, CSS and JavaScript into a much cleaner workflow. Websites such as this, with high levels of interaction, are almost application-like behind the scenes, and the **data** attribute allows for a cleaner application framework.*

— W+K



So, what is the data attribute? You can read about it in the official spec, which defines it as follows:

Custom data attributes are intended to store custom data private to the page or application, for which there are no more appropriate attributes or elements.

— W+K

In other words, any attribute prefixed with data- will be treated as storage for private data; it does not affect the mark-up, and the user cannot see it. Our previous example can now be rewritten as:

```
<article class='story' id="introduction" data-language="en-us">
</article>
```

The other good news is that you can use more than one **data** attribute per element, which is exactly what we're doing in our parallax example. You may have spotted the following:

```
<div data-type="sprite" data-offsetY="100" data-Xposition="50%"
data-speed="2"></div>
```

Here, we are storing four pieces of information: the x and y data offsets and the data speed, and we are also marking this element as a data type. By testing for the existence of **data-type** in the JavaScript, we can now manipulate these elements as we wish.

Parallax Scrolling

On our page, three things create the parallax scrolling illusion:

- The background scrolls at the slowest rate,

- Any sprites scroll slightly faster than the background,
- Any section content scrolls at the same speed as the window.

With three objects all scrolling at different speeds, we have created a beautiful parallax effect.



It goes without saying that we don't need to worry about the third because the browser will take care of that for us! So, let's start with the background scroll and some initial jQuery.

```
$(document).ready(function(){
  // Cache the Window object
  $window = $(window);

  // Cache the Y offset and the speed
  $('[data-type]').each(function() {
    $(this).data('offsetY', parseInt($(this).attr('data-offsetY')));
    $(this).data('speed', $(this).attr('data-speed'));
  });

  // For each element that has a data-type attribute
  $('section[data-type="background"]').each(function(){
    // Store some variables based on where we are
    $(this).data('speed', parseInt($(this).attr('data-speed')));
    var $self = $(this),
        offsetCoords = $self.offset(),
        topOffset = offsetCoords.top;
    $(window).scroll(function(){
```

```

    // The magic will happen in here!
  }); // window scroll
}); // each data-type
}); // document ready

```

First, we have our trusty jQuery `document ready` function, to ensure that the DOM is ready for processing. Next, we cache the browser window object, which we will refer to quite often, and call it `$window`. (I like to prefix jQuery objects with `$` so that I can easily see what is an object and what is a variable.)

We also use the jQuery `.data` method to attach the Y offset (explained later) and the scrolling speed of the background to each element. Again, this is a form of caching that will speed things up and make the code more readable.

We then iterate through each section that has a `data` attribute of `data-type="background"` with the following:

```

$('section[data-type="background"]').each(function(){}

```

Already we can see how useful data attributes are for storing multiple pieces of data about an object that we wish to use in JavaScript.

Inside the `.each` function, we can start to build up a picture of what needs to be done. For each element, we need to grab some variables:

```

// Store some variables based on where we are
var $self = $(this),
    offsetCoords = $self.offset(),
    topOffset = offsetCoords.top;

```

We cache the element as `$self` (again, using the `$` notation because it's a jQuery object). Next, we store the `offset()` of the element in `offsetCoords` and then grab the top offset using the `.top` property of `offset()`.

Finally, we set up the window `scroll` event, which fires whenever the user moves the scroll bar or hits an arrow key (or moves the trackpad or swipes their finger, etc.).

We need to do two more things: check that the element is in view and, if it is, scroll it. We test whether it's in view using the following code:

```

// If this section is in view
if ( ($window.scrollTop() + $window.height()) >
    ($offsetCoords.top) &&
    ( ($offsetCoords.top + $self.height()) > $window.scrollTop() ) )

```

```
{
}
```

The first condition checks whether the very top of the element has scrolled into view at the very bottom of the browser window.

The second condition checks whether the very bottom of the element has scrolled past the very top of the browser window.

You could use this method to check whether *any* element is in view. It's sometimes useful (and quicker) to process elements only when the user can see them, rather than when they're off screen.

So, we now know that some part of the section element with a **data-type** attribute is in view. We can now scroll the background. The trick here is to scroll the background slower or faster than the browser window is scrolling. This is what creates the parallax effect.

Here's the code:

```
// Scroll the background at var speed
// the yPos is a negative value because we're scrolling it UP!
var yPos = -($window.scrollTop() / $self.data('speed'));

// If this element has a Y offset then add it on
if ($self.data('offsetY')) {
  yPos += $self.data('offsetY');
}

// Put together our final background position
var coords = '50% ' + yPos + 'px';

// Move the background
$self.css({ backgroundColor: coords });
```

The **y** position is calculated by dividing the distance that the user has scrolled from the top of the window by the speed. The higher the speed, the slower the scroll.

Next, we check whether there is a **y** offset to apply to the background. Because the amount that the background scrolls is a function of how far the window has scrolled, the further down the page we are, the more the background has moved. This can lead to a situation in which the background starts to disappear up the page, leaving a white (or whatever color your background is) gap at the bottom of each section.

The way to combat this is to give those backgrounds an offset that pushes them down the page an extra few hundred pixels. The only way to find out this magic offset number is by experimenting in the brows-

er. I wish it was more scientific than this, but this offset really does depend on the height of the browser window, the distance scrolled, the height of your sections and the height of your background images. You could perhaps write some JavaScript to calculate this, but to me this seems like overkill. Two minutes experimenting in Firebug yields the same result.

The next line defines a variable `coords` to store the coordinates of the background. The `x` position is always the same: 50%. This was the value we set in the CSS, and we won't change it because we don't want the element to scroll sideways. Of course, you're welcome to change it if you want the background to scroll sideways as the user scrolls up, perhaps to reveal something.

(Making the speed a negative number for slower scrolling might make more sense, but then you'd have to divide by `-$speed`. Two negatives seems a little too abstract for this simple demonstration.)

Finally, we use the `.css` method to apply this new background position. Et voila: parallax scrolling!

Here's the code in full:

```
// Cache the Window object
$window = $(window);

// Cache the Y offset and the speed of each sprite
$('[data-type]').each(function() {
    $(this).data('offsetY', parseInt($(this).attr('data-offsetY')));
    $(this).data('speed', $(this).attr('data-speed'));
});

// For each element that has a data-type attribute
$('section[data-type="background"]').each(function(){

// Store some variables based on where we are
var $self = $(this),
    offsetCoords = $self.offset(),
    topOffset = offsetCoords.top;

$(window).scroll(function(){

// If this section is in view
if ( ($window.scrollTop() + $window.height()) > (topOffset) &&
    ( (topOffset + $self.height()) > $window.scrollTop() ) ) {

// Scroll the background at var speed
```

```

// the yPos is a negative value because we're scrolling it UP!
var yPos = -($window.scrollTop() / $self.data('speed'));

// If this element has a Y offset then add it on
if ($self.data('offsetY')) {
    yPos += $self.data('offsetY');
}

// Put together our final background position
var coords = '50% ' + yPos + 'px';

// Move the background
$self.css({ backgroundColor: coords });

}; // in view

}); // window scroll

}); // each data-type

```

Of course, what we've done so far is quite a bit simpler than what's on Nike Better World. W+K admits that the parallax scrolling threw it some challenges:

The parallax scrolling presented a few small challenges in cross-browser compatibility. It took a little experimenting to ensure the best scrolling experience. In the end, it was less about the actual parallax effect and more about synchronized masking of layers during transitions.

— W+K

W+K also reveals how it maintained a fast loading and paint speed by choosing its tools wisely:

The key to maintaining faster speeds is to use native CSS where possible, because DOM manipulation slows down rendering, particularly on older browsers and processors.

— W+K

For example, the “More” button below spins on hover, an effect achieved with CSS3. In browsers that don't support CSS3 transforms, the purpose of the graphic is still obvious.

WAGE SPORT ON WAR.

Need to stop a war? Look no further than soccer. A civil war in Ivory Coast came to a cease-fire during the 2006 Football World Championship when the national soccer team progressed to the finals. Sport 1. War 0.



ADDING MORE ELEMENTS

Of course, one of the other common features of parallax scrolling is that *multiple* items on the page scroll. So far, we have two elements that move independently of each other: the first is the page itself, which scrolls when the user moves the scroll bar, and the second is the background, which now scrolls at a slower rate thanks to the jQuery above and the `background-position` CSS attribute.

For many pages, this would be enough. It would be a lovely effect for the background of, say, a blog. However, Nike and others push it further by adding elements that move at a different speed than that of the page and background. To make things easy — well, easier — I'm going to call these new elements sprites.

Here's the HTML:

```
<div id="smashinglogo" data-type="sprite" data-offsetY="1200" data-Xposition="25%" data-speed="2"></div>
```

Put this just before the closing `</article>` tag, so that it appears *behind* the contents of `<article>`. First, we give the div an id so that we can refer to it specifically in the CSS. Then we use our HTML5 `data` attribute to store a few values:

- The status of a `sprite`,
- A `y` (vertical) offset of 1200 pixels,
- An `x` (horizontal) position as a percentage,
- A scrolling speed.

We give the `x` position of the sprite a percentage value because it is relative to the size of the viewport. If we gave it an absolute value, which you're welcome to try, there's a chance it could slide out of view on either the left or right side.

Now about that `y` offset...

INCEPTION

This is the bit that's going to mess with your noodle and is perhaps the hardest part of the process to grasp.

Thanks to the logic in the JavaScript, the sprite won't move until the parent section is in view. When it does move, it will move at (in this case) half the speed. You need the vertical position, then, to account for this slower movement; elements need to be placed higher up if they will be scrolling more slowly and, therefore, moving less on the `y` axis.

We don't know how far the user has to scroll before the section appears at the bottom of the page. We could use JavaScript to read the viewport size and then do some calculations based on how far down the page the section is positioned. But that is already sounding too complicated. There is an easier way.

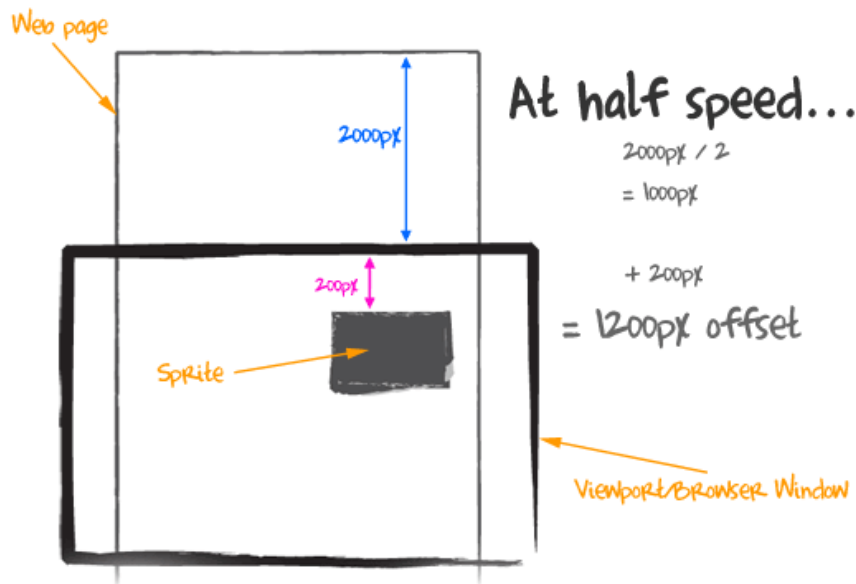
What we *do* know is how far the user has scrolled before the current section is flush with the *top* of the viewport: they have scrolled the `y` offset of that particular section. (Put another way, they have scrolled the height of all of the elements *above* the current one.)

So, if there are four sections, each 1000 pixels high, and the third section is at the top of the viewport, then the user *must* have scrolled 2000 pixels, because this is the total height of the preceding sections.

If we want our sprite to appear 200 pixels from the top of its parent section, you'd figure that the total vertical offset we give it should be 2200 pixels. But it's not, because this sprite has speed, and this speed (in our example) is a function of how far the page has been scrolled.

Let's assume that the speed is set as `2`, which is half the speed at which the page is scrolling. When the section is fully in view, then the window has scrolled 2000 pixels. But we divide this by the speed (`2`) to get 1000 pixels. If we want the sprite to appear 200 pixels from the top, then we need to *add* 200 to 1000, giving us 1200 pixels. Therefore, the offset is 1200. In the JavaScript, this number is inverted to -1200 because we are pushing the `background-position` down off the bottom of the page.

Here's a sketch to show this in action.



This is one of those concepts that is easier to understand when you view the page and source code and scroll around with the console open in Firebug or Developer Tools.

The JavaScript looks like this:

```
// Check for other sprites in this section
$('[data-type="sprite"]', $self).each(function() {

    // Cache the sprite
    $sprite = $(this);

    // Use the same calculation to work out how far to scroll
    // the sprite
    var yPos = -($.Window.scrollTop() / $sprite.data('speed'));
    var coords = $sprite.data('Xposition') + ' ' + (yPos +
    $sprite.data('offsetY')) + 'px';
    $sprite.css({ backgroundColor: coords });
}); // sprites
```

HTML5 VIDEO

One criticism levelled at Nike Better World is that it didn't use HTML5 video. HTML5 is still not fully supported across browsers (I'm looking at you, Internet Explorer), but for the purposes of this chapter, we'll embrace HTML5 video, thanks to the lovely folks at [Vimeo](http://vimeo.com/14592941)⁵⁸ and [Yum Yum London](http://vimeo.com/14592941)⁵⁹.

⁵⁸. <http://vimeo.com/14592941>

But we can't set a video as a background element, so we have a new challenge: how to position and scroll this new sprite?

Well, there are three ways:

1. We could change its **margin-top** property within its parent section;
2. We could make it a **position: absolute** element and change its top property when its parent section comes into view;
3. We could define it as **position: fixed** and set its **top** property relative to the viewport.

Because we already have code for the third, let's grab the low-hanging fruit and adapt it to our purposes.

Here's the HTML we'll use, which I've placed *after* the closing **</article>** tag:

```
<video controls width="480" height="320" data-type="video"
data-offsetY="2500" data-speed="1.5">
  <source src="video/parallelparking.theora.ogv" type="video/ogg"
/>
  <source src="video/parallelparking.mp4" type="video/mp4" />
  <source src="video/parallelparking.webm" type="video/webm" />
</video>
```

First, we've opened our HTML5 video element and defined its width and height. We then set a new **data-type** state, **video**, and defined our **y** offset and the speed at which the element scrolls. It's worth noting that some experimentation is needed here to make sure the video is positioned correctly. Because it's a **position: fixed** element, it will scroll on top of all other elements on the page. You can't cater to every viewport at every screen resolution, but you can play around to get the best compromise for all browser sizes (See "Bespoke to Broke" below).

The CSS for the video element looks like this:

```
video { position: fixed; left: 50%; z-index: 1;}
```

I've positioned the video **50%** from the left so that it moves when the browser's size is changed. I've also given it a **z-index: 1**. This z-index prevents the video element from causing rendering problems with neighbouring sections.

And now for the JavaScript! This code should be familiar to you:

59. <http://www.yumyumlondon.com/>

```
// Check for any Videos that need scrolling
$('[data-type="video"]', $self).each(function() {

    // Cache the sprite
    $video = $(this);

    // Use the same calculation to work out how far to scroll
    // the sprite
    var yPos = -($window.scrollTop() / $video.data('speed'));
    var coords = (yPos + $video.data('offsetY')) + 'px';

    $video.css({ top: coords });

}); // video
```

And there you have it! A parallax scrolling HTML5 video.

BESPOKE OR BROKE

Of course, every design is different, which means that *your* code for *your* design will be unique. The JavaScript above will plug and play, but you will need to experiment with values for the *y* offset to get the effect you want. Different viewport sizes means that users will scroll different amounts to get to each section, and this in turn affects how far your elements scroll. I can't control it any more than you can, so you have to pick a happy medium. Even Nike Better World suffers when the viewport's vertical axis stretches beyond the height of the background images.

I asked W+K how it decides which effects to power with JavaScript and which are better suited to modern CSS techniques:

Key points that required complex interaction relied on JavaScript, while visual-only interactivity relied on CSS3. Additionally, fewer browsers support native CSS3 techniques, so items that were more important to cross-browser compatibility were controlled via JavaScript as well.

— W+K

This is a wonderful example of “real-world design.” So often we are bamboozled with amazing new CSS effects, and we make websites that sneer at older browsers. The truth is, for most commercial websites and indeed for websites like Nike Better World that target the biggest audience possible, stepping back and considering how best to serve your visitors is important.

W+K explains further:

We started by creating the best possible version, but always kept the needs of all browsers in mind. Interactive storytelling must balance design and technology to be successful. A great website usable in one or two browsers ultimately fails if you want to engage a wide audience.

— W+K

And Internet Explorer?!

IE was launched in tandem with the primary site. Only IE6 experienced challenges, and as a deprecated browser, it gracefully degrades.

— W+K

The Final Code

The code snippets in this piece hopefully go some way to explaining the techniques required for a parallax scrolling effect. You can extend them further to scroll multiple elements in a section at different speeds, or even scroll elements sideways!

Feel free to grab the [full source code from GitHub](#)⁶⁰, and adapt it as you see fit.

Of course, remember that manipulating huge images and multiple sprites with JavaScript can have huge performance drawbacks. As [Keith Clark](#) tweeted⁶¹:



Lots of parallax scrolling websites around at the moment. If you build one, please check it on an average spec PC < most are sluggish!

Test, test and test again. Optimize your images, and be aware that you may have to compromise to support all browsers and operating systems.

⁶⁰. <https://github.com/richardshepherd/Parallax-Scrolling>

⁶¹. <http://twitter.com/#!/keithclarkcouk/status/86367151585378304>

Tell A Story

Above and beyond the technical wizardry of parallax websites — some of the best of which are listed below — the common thread that each seems to embody is story. That’s what makes them great.

I asked W+K what it learned from the project:

That a strong voice, simplicity and beauty are integral to a great interactive storytelling experience. We often hear things like “content is king, and technology is just a tool to deliver it,” but when you’re able to successfully combine a powerful message with a compelling execution, it creates an experience that people really respond to and want to spend time with.

— W+K

We really have just scratched the surface of the work that goes into a website like Nike Better World. The devil is in the details, and it doesn’t take long to see how much detail goes into both the design and development.

However, if you have a compelling story to tell and you’re not afraid of a little JavaScript and some mind-bending offset calculations, then a parallax website might just be the way to communicate your message to the world.

More Examples

Nike wasn’t the first and won’t be the last. Here are some other great examples of parallax scrolling:

Manufacture d’essai⁶²



⁶². <http://www.manufacturedessai.it/en/>

Yebo Creative⁶³



TEDx Portland⁶⁴



Ben the Bodyguard⁶⁵



^{63.} <http://yebocreative.com/>

^{64.} <http://tedxportland.com/>

^{65.} <http://benthebodyguard.com/>

Campaign Monitor Is Hiring⁶⁶



Nizo App⁶⁷



... If you need any more encouragement to create a website as compelling as these, here's what the team at W+K used to put together Nike Better World: MacBook Air 13", Canon 5D Mark II, Coda, Adobe Photoshop and Adobe Illustrator.

THANKS

Putting together this article took the cooperation of a number of people. I'd like to thank Seth, Ryan, Ian and Duane for answering my questions; Katie Abrahamson at W+K for her patience and for helping coordinate the interview; and Nike for allowing us to dissect its website so that others could learn. 🐼

⁶⁶. <http://www.campaignmonitor.com/hiring/>

⁶⁷. <http://nizoapp.com/>

Behind The Scenes Of Tourism New Zealand (Case Study)

BY RICHARD SHEPHERD 🐏

In 2011 we saw the rise in popularity of two relatively new trends: responsive Web design and the use of HTML's canvas. While some websites had experimented with both, we've seen these trends move from the fringes firmly into the mainstream.

Responsive Web design is more a concept than a technology – an ideal that many new websites aspire to. Canvas, on the other hand, is an HTML5-based technology that opens the door to a new wave of interactivity. In this chapter, we'll look at a website that embraces both of these elements, one that has been nominated for a Technical Achievement award at SxSW Interactive 2012: Tourism New Zealand⁶⁸.

I spoke with the creative and technical teams at the New Zealand-based firms Shift⁶⁹ and Assembly⁷⁰ about the technology behind Tourism New Zealand's home page, how it was made and how they balanced the breakthrough effect with bandwidth.



⁶⁸. <http://www.newzealand.com/int/>

⁶⁹. <http://www.shift.co.nz>

⁷⁰. <http://www.assemblyltd.com/>

After we learn how Tourism New Zealand was built, we'll look at the basics of putting together your own animated HTML5 canvas background.

Editor's note: At the time of production of this eBook (December 2013) parallax scrolling and the layering effect mentioned in this chapter were no longer in use on the Tourism New Zealand website.

The Inspiration

Tourism New Zealand has a history of strong media and advertising campaigns, particularly through TV commercials, and its destination marketing website has long featured these videos.

So, when a redesign of the website was planned without an accompanying TV commercial, the team at Shift had a new challenge: to showcase the landscape and breadth of activities in New Zealand to an audience that has come to interact with a website, not to watch a video.

Shift was inspired by, among other things, Nike's Better World website. But whereas that website used HTML5 animation to create an effect of parallax scrolling, Shift proposed a camera view that moves through the real world.

Shift's first idea was to use video as a background, with the camera descending through some of New Zealand's most exciting locations. The problem was bandwidth: at 25 frames per second, Shift would need to deliver thousands of full-screen frames. This was an impossible goal, so instead the team set a target of around 200 frames for the entire website. But 200 frames for the amount of footage they wanted worked out to about three frames per second, which doesn't look particularly good.

At this point, they nearly abandoned the idea.

By chance, though, a solution was found. Scrubbing through a QuickTime video, Shift's creative director, Mark Zeman, noticed that being able to control the frame rate as you move the play head gives the illusion of a smooth experience. He suspected that if the browser's scroll bar could be used to scrub the footage, it would feel similarly smooth. And if the camera shot photographs as it moved through each scene, it would create the illusion that the user was practically controlling the camera as it descended through New Zealand.

The first prototype was made with a few images and some JavaScript, just to prove that the browser could handle this type of manipulation. Freelance front-end developer [Jeff Nusz](http://www.custom-logic.com/)⁷¹ was then tasked

⁷¹. <http://www.custom-logic.com/>

with turning this proof of concept into a working website. The brief was simple: no loading bar, and the first frame must load almost instantly. Furthermore, the website had to be usable even when the user doesn't wait for all of the images to download.

On Location

With the prototype built and approved by Tourism New Zealand, it was time to load up the trucks with scaffolding and start shooting.

Shift worked with directors Damon Duncan and Matt von Trott of *Assembly*⁷² throughout development of the project, which enabled them to build the large motion-control rigs that were required for the shoot.

A 15-meter-high motion-control rig was built, with a vertical dolly track that housed a Canon 5D DSLR. The rig was set up at each location, and the camera travelled 12 meters down it, taking anywhere between 750 and 950 stills, a process that lasted at least two hours each time (and six hours at Milford Sound, the first location on the website), because the camera moved around 15 millimeters every 10 seconds.



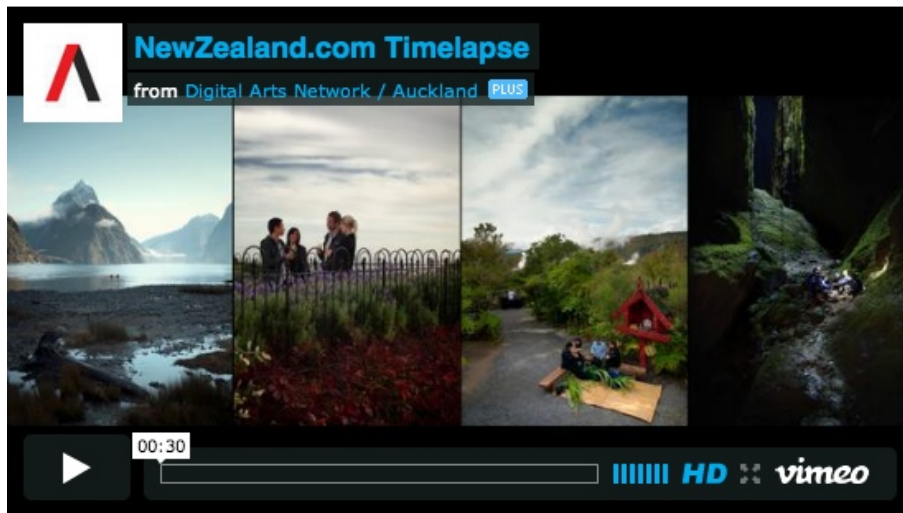
*Milford Sound*⁷³, New Zealand

Each location took three days to shoot. The first day was spent building the tower and putting together the motion-control rig. The second day was usually spent rehearsing the scene, and the third would be for the

⁷². <http://www.assemblyltd.com/>

⁷³. <http://www.newzealand.com/int/milford-sound/>

actual shoot, which included multiple passes of the scene. Indeed, the scene for Waiheke Island is a combination of two takes.



Time-lapse sequences from all four shoots ([watch on Vimeo](https://vimeo.com/36803041)⁷⁴)

Front-end developer Jeff Nusz had built a suite of tools⁷⁵ to help the crew plan the photography. The directors also showed the original prototype to the cast and crew, which helped to demonstrate what they were filming and how the time-lapse footage would work.



⁷⁴. [http://vimeo.com/36803041](https://vimeo.com/36803041)

⁷⁵. <http://www.custom-logic.net/test/tnz/tower/>

Post-Production

There were only five days between the final day of shooting and the launch of the website. Most of the code had been written and ported to JavaScript using prototype images, and so it was now a matter of editing the hundreds of photos from each location down to 200 hand-optimized 1280 × 720-pixel frames.

The team spent time testing the website in each browser, noticing different memory quirks in Chrome and Firefox, and it worked on optimizing the website's performance. Because many of New Zealand's tourists come from China, the time was also spent making sure the website worked even in IE 6. To better understand the toll of such a heavy home page, Shift used WebPagetest⁷⁶ to test it from various parts of the globe.

From 25 MB To 300 KB

Although the entire website weighs in at 25 MB, it enables a usable experience after the first 300 KB of data and images have loaded.

On the initial loading of the page, a 40 KB image is loaded that contains all 200 frames at a very low resolution. This means that as soon as the initial frame of Milford Sound has loaded, you can scrub through the entire website and get some approximation of the full experience. Then, when you stop scrubbing, the website loads the frame you're on and starts loading the frames before and after it. Moreover, as you scroll the page, all downloading activity stops to ensure the smoothest possible scrolling experience. When you stop scrolling, all downloading resumes.

Interestingly, Jeff's background is mainly in working with Flash, so the initial website was built in Flash and then ported to HTML5 at the last minute. Jeff says that the similarities between ActionScript and JavaScript made this a relatively pain-free process:

ActionScript 3 is a lot like JavaScript. I did a lot of the work in Flash, and I wanted to see how well it would translate, and it was pretty much cut-and-paste for the most part. I just had to structure things in a similar way and make a few semantic changes. Ninety percent of the code ported as it was, which was great.

One of the team's early breakthroughs was in figuring out how to split up the website's elements. You can see in the sketch below how a web-

⁷⁶. <http://www.webpagetest.org/>

A hand-drawn 3D perspective diagram of a web browser window layout. The window is divided into several sections: a top 'HEADER' bar, a 'FOOTER' bar at the bottom, and a main content area. The content area is further divided into a 'BACKGROUND' section on the right and a 'TAGS' section on the left. The 'TAGS' section contains three 'UI WIDGETS'. The entire structure is labeled 'BROWSER' at the top.

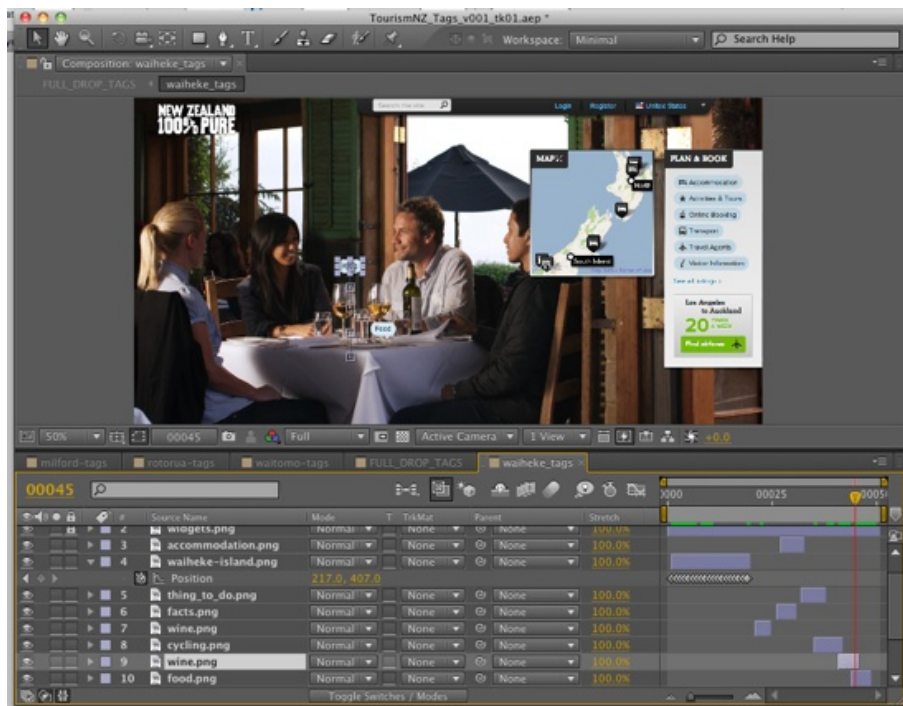
small, then the scroll bar would be tiny; Shift wanted to keep the size of the scroll bar consistent, regardless of the browser's size.

As Jeff notes, “Although it’s not exact, the goal was that one click of your mouse-wheel would be one frame”:

The tag layer and the UI layer is just HTML and CSS, but the background is either an HTML5 canvas that we draw to or it’s Flash. There’s about a 20-millisecond lag between the frames moving and the background changing in Flash, so the canvas version became the primary method used for browsers that support it.

The tags do not scroll in the conventional sense, but for each frame, they are assigned specific x and y coordinates based on the frame. This allows tags to be “glued” to background elements — a tree, mountain top, etc. — and these coordinates would be recalculated on the fly if the browser window was resized.

To create this effect, the team used Adobe After Effects to keyframe each tag and output the coordinates. Jeff built a PHP script into which the team could copy and paste the After Effects keyframe data, whereupon the script would generate JavaScript that the website could use⁷⁷. Jeff then tweaked the tags in JavaScript to make sure they scrolled as expected.



⁷⁷ http://www.custom-logic.net/test/tnz/prod/data/tag_positions/tagData_milford.js

Sprite Sheets

To create the illusion that the page loads quickly, Jeff came up with the idea of using sprite sheets. As he puts it, “The sprite sheets were the breakthrough that allowed us to make the website usable so quickly after the page loads”.



The first frame (of the background image) is 90 KB. So, together with the first low-resolution sprite sheet (which is 30 KB), these images initially weigh only 130 KB. Once the page has loaded all of the content, it loads a higher-resolution sprite sheet, and then another, and then it loads in the full-resolution images one by one. Rather than loading these large images in sequence, it loads them based on your scroll position within the document.

You can check that the sprite sheets load before the full-resolution images by turning the Webkit Inspector on and watching the “Network” activity after the page has started to render.

Using Inline JavaScript

I asked the team at Shift why so much inline JavaScript was in the head of the document. Best practice would suggest it should be moved to a separate file. However, something more is going on here.

Shift’s solution architect, Glenn Wright, has been working on the website for the last six years. He explains:

The inline JavaScript is there to define the configuration for each page. We can cache each page’s configuration in JavaScript, then all the external libraries that the page loads utilize that data. It gives us an easy way to configure the responsive grid and all of the interactions that the page will have to use.

Testing Mode

A testing mode on the website lets you see some of this loading process in action. Clear your browser's cache, load Tourism New Zealand's home page, and then hit the tilde (~) key five times to enable testing mode. You won't see any change immediately, but once in this mode you can use the following keyboard commands:

- **1**
Turn on the visualizer for “frames loaded”
- **3**
Toggle tags on and off
- **0**
Toggle the high-resolution image on and off
- **-**
Go to the previous low-resolution sprite
- **+**
Go to the next high-resolution sprite

Building Your Own Canvas Background Animation

While websites like Tourism New Zealand take months to design and build, creating your own animated canvas background is relatively straightforward. All it takes is a little JavaScript, a pinch of HTML5 and an enormous amount of imagination to dream up a worthwhile use.

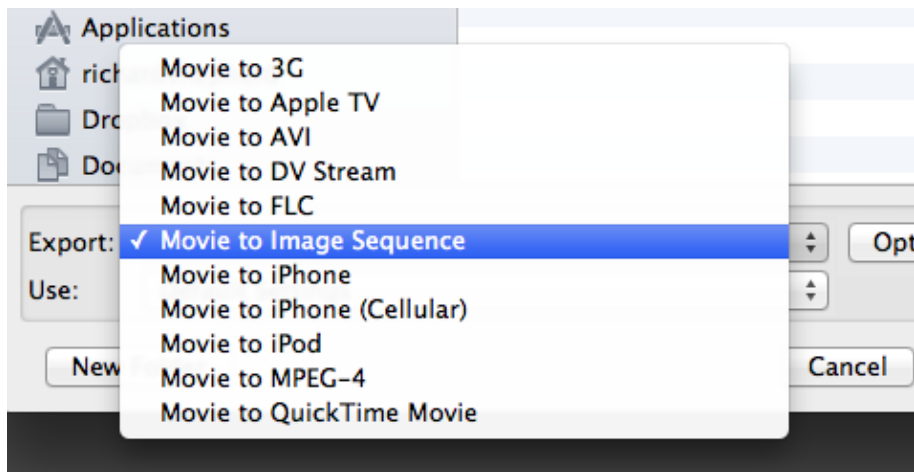
Indeed, a word of caution is in order. We all know that with great power comes great responsibility, and just because we can load 50 high-resolution frames and animate them as the user scrolls the window doesn't mean we should. As mentioned earlier, websites that use this technique are both lauded and criticized. Make sure that you are enriching the user experience rather than needlessly slowing it down.

First, we need to do a couple of things. Let's grab some video (a time-lapse video works best for this), and then create our 50 frames. You can create as many frames as you like, but we'll stick with 50 because this is how many Tourism New Zealand uses for each scene.

Next, let's grab some royalty-free time-lapse footage from [Artbeats](http://www.artbeats.com/)⁷⁸ and import it into iMovie, where you can export individual frames.

Choose **Share → Export Using QuickTime**, and in the pop-up dialog box select “Movie to Image Sequence.”

⁷⁸. <http://www.artbeats.com/>



We export the images as JPEGs, which leaves us with hundreds of images. So, we have to do two things:

1. Batch process the images in Photoshop to the required size (here, we went with 480×270 pixels to maintain the aspect ratio).
2. Edit the number of images down to 50. Shift was painstaking in doing this, retouching and editing thousands of images down to just 200. To keep things easy for this tutorial, let's just delete three photos out of every four.

You should now have 50 photos that, when played in sequence, make for a somewhat choppy time-lapse video.

On to the next couple of steps. First, we'll figure out how to load the images in the browser and control them with the scroll bar. Secondly, we'll create our own sprite sheet and use that instead. Open your code editor of choice, and create a new JavaScript file. Or why not head over to GitHub and download the project⁷⁹ to play around with?

You'll also need an HTML file with a canvas element in it, which is as simple as this:

```
<canvas id="canvas"></canvas>
```

Hopefully, the comments in the code sufficiently explain what's going on, but here's a breakdown anyway.

First, we set up the canvas element and use some basic maths to set start and end points:

```
// Create our timeLapseVideo object and cache the window object
timeLapseVideo = new Image();
$window = $(window);
```

⁷⁹. <https://github.com/richardshepherd/Canvas-Background>

```
// Define the timeLapseVideo element
timeLapseVideo.onload = function(){
  videoContainer.width = 960;
  videoContainer.height = 540;
  videoContext.drawImage(timeLapseVideo, 0, 0, 960, 540);
}

// Load the first frame
timeLapseVideo.src = 'images/TimeLapse001.jpg';

// Calculate how far to scroll before changing a frame
var scrollDistance = $window.height() - $window.scrollTop(),
    totalHeight = $(document).height() - scrollDistance;
    frameDiff = totalHeight / (numberOfFrames - 1);
```

Then, when the window scrolls, we work out which frame needs to appear on screen:

```
// Which frame do we need to show?
frameString = (Math.round($window.scrollTop()/frameDiff) +
1).toString();

// Change the image in the canvas
timeLapseVideo.src = 'images/TimeLapse' + frameString + '.jpg';
```

Make sure to go through the [full source code](https://github.com/richardshepherd/Canvas-Background)⁸⁰ to see it in action. Be careful, because there is no check to determine whether an image has already loaded; you could end up hitting your server with hundreds of requests per page. So, like Tourism New Zealand, you need to build in a check to see whether an image has been cached.

Creating And Using A Sprite Sheet

We can also create our own sprite sheet with our image to emulate more of Tourism New Zealand's behavior. Some apps and websites are available to help you do the job (see the "Related Links" section at the end), but it's probably easier to run a simple script that pastes all of the images onto a canvas and then take a plain screenshot of that. The code

80. <https://github.com/richardshepherd/Canvas-Background>

for this script is in the [GitHub repository](#)⁸¹ in case you want to take a look.

There are two main differences with this technique. First, we load only one image onto the canvas, which is the sprite sheet:

```
timeLapseVideo.src = 'images/spritesheet.jpg';
```

When the window scrolls, we then draw part of this sprite to the canvas:

```
videoContext.drawImage(timeLapseVideo, x, y, 80, 45, 0, 0, 960, 540);
```

In the line above, `x` and `y` are the starting coordinates of the sprite, which in turn are dictated by the frame. We're using a simple loop to iterate through the sprite to find the right coordinates, but you could set up an array of values to look up. These tiny frames are 80×45 pixels, and they are drawn from 0,0 to 960×540 pixels. CSS then stretches this canvas to fit the entire window.

You can improve this script a great deal by, for example, preloading images when the user isn't scrolling (some links are below to point you in the right direction).

Creativity In Design And Development

Distilling 18 months of hard work into one chapter like this is an impossible task. Shift had the time and budget to fly around New Zealand and take thousands of beautiful photographs. This is not an option for the average Web designer!

However, what is perhaps most striking about Tourism New Zealand is the imagination in its concept and execution. By taking the leap from virtual to physical world (and back again!), the scroll bar has become a conduit by which the user can control their experience of New Zealand – if only for 200 frames.

Jeff's ingenious loading techniques and the use of sprite sheets also kept away the loading bar that is so familiar with websites like this. On an average broadband connection, the experience is effortless and fun.

81. <https://github.com/richardshepherd/Canvas-Background>



Perhaps the biggest lesson to be learned here is this. Despite its many bells and whistles, Tourism New Zealand encourages the user to discover the website in much the same way they would discover the country.

Technology supports the great design, not the other way around. If we could all exercise this level of creativity, imagination and user engagement, then we, too, would create amazing websites that shape the future of Web design.

RELATED LINKS

- “[Blowing Up HTML5 Video and Mapping It Into 3D Space⁸²](#)” (demo), Sean Christmann, Craftymind
- “[How Browsers Work: Behind the Scenes of Modern Web Browsers⁸³](#)” (“The Canvas”), Tali Garsiel, HTML5 Rocks Tutorials
- [HTML5 Canvas Tutorials⁸⁴](#)
- “[Load a Photo in a Canvas, Then Flip⁸⁵](#),” Stoyan Stefanov, phpied.com
- “[How to Draw With HTML5 Canvas⁸⁶](#),” Jamie Newman, Think Vitamin
- “[Preloading Images in HTML5/JavaScript⁸⁷](#)” (using the canvas), Stack-Overflow
- “[Using Images⁸⁸](#)” (canvas tutorial), Mozilla
- Create sprite sheets easily in Mac and PC:
 - [CSS Sprite Generator⁸⁹](#), Project Fondue
 - [Instant Sprite⁹⁰](#), Brian Grinstead
 - [Texture Packer⁹¹](#), Andreas Löw, code’n’web
 - [Sprite Sheet Packer⁹²](#), CodePlex

I must extend my very special thanks to Jeff (@jeffnusz⁹³), Glenn, Che and, in particular, Mark (@markzeman⁹⁴), who were patient with my intermittent Skype connection and who were so transparent in offering their thoughts and many of the images and resources featured in this chapter. 🐼

⁸². <http://www.craftymind.com/factory/html5video/CanvasVideo3D.html>

⁸³. http://www.html5rocks.com/en/tutorials/internals/howbrowserswork/#The_canvas

⁸⁴. <http://www.html5canvastutorials.com/>

⁸⁵. <http://www.phpied.com/photo-canvas-tag-flip/>

⁸⁶. <http://thinkvitamin.com/code/how-to-draw-with-html-5-canvas/>

⁸⁷. <http://stackoverflow.com/questions/9334083/preloading-images-in-html5-javascript>

⁸⁸. https://developer.mozilla.org/en/Canvas_tutorial/Using_images

⁸⁹. <http://spritegen.website-performance.org/>

⁹⁰. <http://instantsprite.com/>

⁹¹. <http://www.texturepacker.com/>

⁹². <http://spritesheetpacker.codeplex.com/>

⁹³. <https://twitter.com/#!/jeffnusz>

⁹⁴. <https://twitter.com/#!/markzeman>

Tale Of A Top-10 App, Part 1: Idea And Design

JEREMY OLSON 🍷

My name is Jeremy Olson. I'm a senior in college, living in Charlotte, North Carolina, and this is the story of how my little app beat Angry Birds.

I'm writing this because I believe we learn much more from success than from failure. It took Edison thousands of failed attempts to invent the electric light bulb, and it would be foolish for us to reinvent it based on trial and error, now that we have a working model.

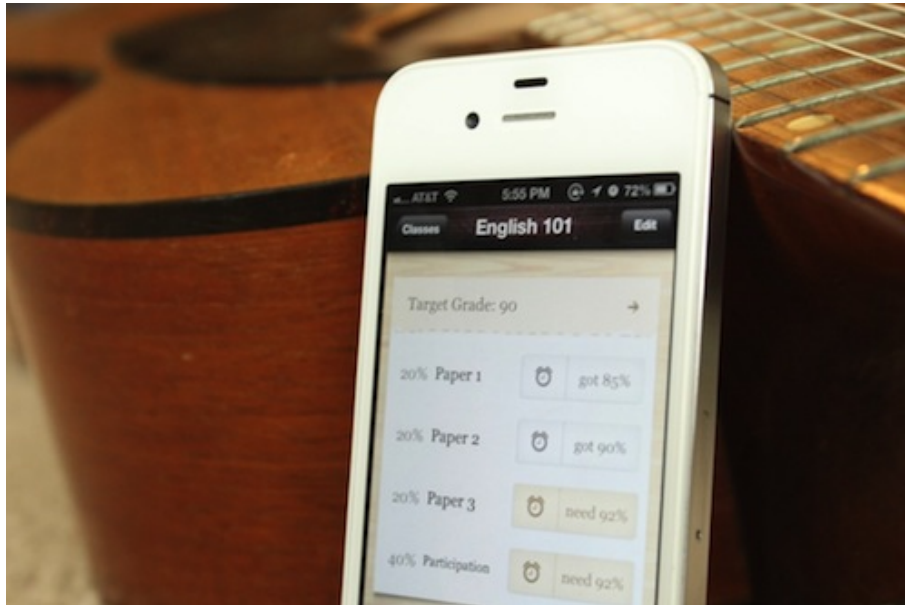
We have many shining lights in the app industry. While I would love to claim that my success stems from my own genius, nothing could be further from the truth. By studying independent developers who have succeeded in the App Store again and again, I was able to learn the basic principles that I needed to succeed, and I hope this story will help others do the same.



A Big Idea

My first app, Grades, had everything going for it. The press loved it, users loved it, and Apple loved it. There was only one problem: It didn't make any money. Sure, it generated a little cash, but despite all of the buzz, Grades was always limited by the tiny niche it served: college stu-

dents who cared enough about their grades to faithfully track them throughout the semester.



Our first app, Grades, was a success for our reputation, not for our bank account.

If we were to continue making cheap apps, our next one had to be big. It had to appeal to almost anyone.

The solution came when Alex Marktl, founder of Sonico Mobile, approached us about partnering on an offline translation app. It was a proven market. Sonico's app iTranslate had over 30 million users, and the market had an immense gap for an affordable translation app that worked without an Internet connection.

After seeing some user feedback for Sonico's popular iTranslate app and researching the competition, we were pretty sure the market opportunity was huge. In addition, my four-person team is really passionate about education and language. The market was there, the opportunity was there, and the passion was there — a perfect fit.




A few Skype calls later, we had hashed out agreements and were ready to roll.

(Spoiler: It turns out that ideas matter a lot. Languages attracted a similar amount of press and buzz as Grades, but it made more money in one day than Grades made in two years!)

Defining The Dictionary

Although I was tempted to jump right into wireframing, we did some research up front to help us define the problems we were trying to solve.

COMPETITIVE LANDSCAPE

	Description	Analogy (Good and Bad)	Competition
 iTranslate	Sonico's flagship free app that uses the Google translation API to translate phrases among multiple languages.	Allows translation of whole phrases or sentences.	Free and extremely popular, is somewhat a competitor to Languages. But languages will differentiate itself by working offline and being extremely fast for single words.
 LEO dictionary	Translation dictionary. Translates between German and English, French, Spanish, Russian, and Chinese. Connects to LEO's online dictionary, so requires Internet connection.	Guesses what language is being typed, so user does not need to select language direction. UI is not very good on iPhone.	Since it is free, it is a competitor for German-based language pairs, but can be easily beat on UI and by off-line capabilities.
 Nifty Words	Translation dictionary. German-English only.	Searches for results for both languages at the same time, so user does not need to select a language direction.	Price is fairly reasonable at \$1.99, but only German-English pair is available. Reviews are mixed. A limited competitor.
 Accio	Translation dictionary. Separate app for each language pair. Works offline.	Requires selection of language direction. Includes conjugations.	Price is fairly reasonable at \$1.99 per language pair for an off-line dictionary. Several language pairs available. Reviews are generally good. A competitor that can be beat on price and user interface.

*Large view.*⁹⁵

The App Store is great because it is one of the few markets in the world where you can so easily find valuable information about your potential competitors. They are just a search away. Looking at reviews, sales rankings and marketing materials of competing apps can give you great insight into the market. It is a great way to see the market for your app, how much people are willing to pay for it, what features to include, and a slew of other insights. Websites such as [App Annie](http://www.appannie.com)⁹⁶ even enable you to analyze your competitors' rankings over time.

We took about a dozen of the best competing apps and analyzed their strengths and weaknesses and how we could beat them. We found that, while a number of offline translation apps existed, they were poorly designed and cost a fortune. We knew we could do better.

USER EXPERIENCE MAPPING

In defining the app, we focused on solving a few problems that people actually experience in their daily life, rather than just coming up with a list of cool features. To this end, we went through a little exercise that we call user experience mapping. This exercise generally takes a day or three. In it, we did the following:

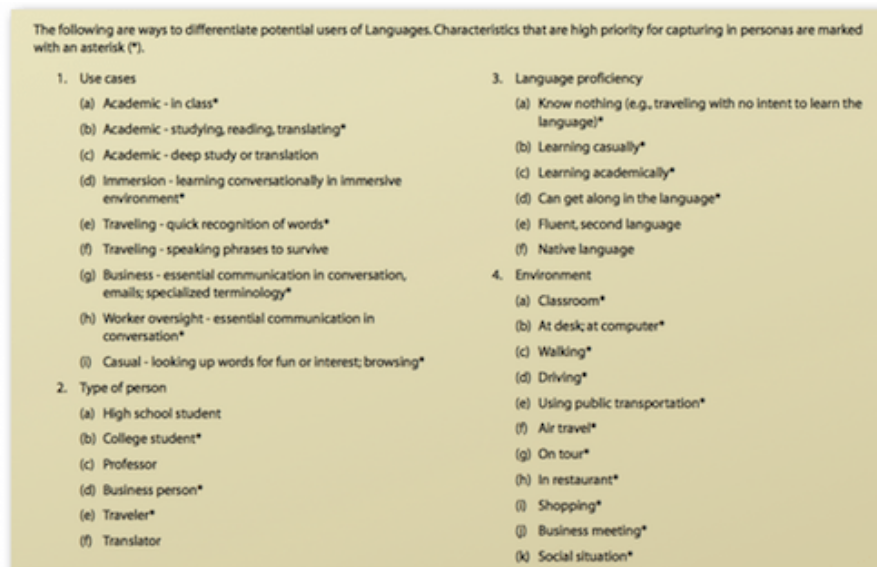
⁹⁵. <http://media.smashingmagazine.com/wp-content/uploads/2013/07/competitors-large.png>

⁹⁶. <http://www.appannie.com>

- Analyze users' daily experience without the app — i.e. identify the problems they currently face.
- Brainstorm ways that an ideal app could solve those problems.
- Choose which problems to focus on, and decide which features were feasible for the first release.

Step 1: Define Personas

As designers, we need to thoroughly empathize with our users and understand their current experiences and thought processes as much as possible. Going out and talking to people can yield a lot of great insight, but in this case we were pretty familiar with the translation experience, so we didn't feel the need to talk to potential users at this stage.






*Large view.*⁹⁷

Instead, we went ahead and started brainstorming potential characteristics of our users.

We then chose characteristics of users who we really wanted to focus on, and turned them into personas.

⁹⁷ <http://media.smashingmagazine.com/wp-content/uploads/2013/07/users-large.png>

Emily	Johann	Paul
 <p>Age: 21 College student, Emory Atlanta, GA Single Proficiency: Third year French in college; not fluent</p>	 <p>Age: 41 Corporate Sales for Siemens, Austria Married, with a son and daughter Proficiency: native German; decent Italian; nearly fluent in French; fluent in English</p>	 <p>Age: 26 IT Professional Newark, NJ Single Lives in apartment Proficiency: Native English; can get along in Spanish</p>
<p>Quote: "I love French, but it is a struggle at times. I wish my reading comprehension and vocabulary were better. It would be cool to study abroad in France."</p>	<p>Quote: "Traveling is my life. I go all over Europe on business."</p>	<p>Quote: "I love dabbling in different languages, and I love to see new places and cultures."</p>
<p>Key Attributes:</p> <ul style="list-style-type: none"> • Likes French; interested in French culture, movies • Likes trying to read French literature • Not naturally strong in language • Has visited Paris; would like to do study abroad program in France 	<p>Key Attributes:</p> <ul style="list-style-type: none"> • Travels all over Europe selling Siemens equipment to businesses • Has built up a strong ability to acquire and learn language • Likes to polish his fluency in the languages he uses for business (Italian, French, English) 	<p>Key Attributes:</p> <ul style="list-style-type: none"> • Loves to travel • Enjoys language and culture • Watches foreign films • Not a master of any second language • Likes to attempt conversation in various languages • Likes to travel with people or meet up with people when traveling • Purchases Rosetta Stone programs for fun • Enjoys language and culture • Watches foreign films

*Large view.*⁹⁸

A persona is a fictitious person who embodies the characteristics of the target demographic. While personas aren't real, they should be based on reality and should make the abstract idea of a "user" much more concrete. Without a human face, mapping out the user's experience is hard.

So, Emily is a 21-year-old college student studying French at Emory University. She is not naturally gifted in language, but really likes French and tries to read French literature. She is looking forward to doing a study-abroad program in France.

We created three personas that encapsulate most of the key characteristics of our target market: Emily, the student; Johann, the European business traveler (it turns out that we nailed this: 70% of our sales ended up being from outside the US); and Paul, the IT guy who learns new languages as a hobby.

Step 2: Map the Personas' Experience Without the App

To map out users' current predicament, we started by picking three key experiences related to translation — in this case, solo translation, social translation, and translation during travel.

We then brainstormed the activities and issues involved in these experiences that each persona might face. For example, in the solo category, Johann writes emails to clients in various languages and looks up the words he is not sure of.

⁹⁸ <http://media.smashingmagazine.com/wp-content/uploads/2013/07/personas-large.png>

Experience domain: need for language translation of words and short phrases			
	Solo	Social	Travel
 Emily	Looks up correct French words for workbook exercises Looks up words for writing essay in French Looks up unknown words while reading French literature (Uses physical French-English dictionary)	Quickly looks up French words for class participation Looks up words while studying French with friends Watches French movies with friends; looks up some words	Same as for Paul
 Johann	Writes emails to clients in various languages; looks up words he's not sure of Browses translation dictionaries to improve language skills Reads online in various languages; looks up unknown words	In business meetings in various countries, sometimes looks up words he's not sure of In one-on-one conversations, looks up words once in a while In business social situations will sometimes look up a word	Needs minimal language help just to get to his destination
 Paul	Uses Rosetta Stone to learn languages Browses language dictionaries out of curiosity Watches foreign films; looks up words he's not sure of Emails friends who speak other languages; looks up words to help him (Uses Google Translate)	Likes to talk with his friends in a language he is practicing; sometimes looks up words Practices his languages with strangers when he has a chance	Needs language help: <ol style="list-style-type: none"> 1. Changing money 2. Calling a cab 3. Reading road signs 4. Ordering in restaurants 5. Taking public transportation 6. etc.

*Large view.*⁹⁹

Perform this exercise with people in the room who are similar to your personas. They will validate your insights and add to the brainstorming. If you don't have that luxury, simply brainstorming and thinking through their possible experience is still a helpful exercise.

Step 3: Brainstorm the Ideal Assistance

After picturing our users' lives, we brainstormed how the ideal app could solve their problems. We didn't worry about viability, budget or timeline here; it's all about coming up with really creative ideas to solve our users' problems.

Step 4: Kill the Baby

This part is brutal. Having come up with a ton of cool ideas for features, we had to obliterate most of them. Good design is more about subtraction than addition. It's all about finding the essential problems you want to solve and removing the features that are unrelated, inessential or unrealistic for the first version.

⁹⁹. <http://media.smashingmagazine.com/wp-content/uploads/2013/07/personas2-large.png>



Polishing an app takes a ridiculous amount of time. So, if you start out with too broad a feature set, your app will lack focus and you will have no way to polish those features adequately.

Solo	Social	Travel
Fast word lookup Voice Auto suggest/define Correct part of speech Most common match on top Text-to-speech results Auto language direction Clear indication of language Easy and delightful browsing Word by word By theme By part of speech Depth Full definition Words are links Gender and plural forms of nouns Conjugation of verbs Theme word belongs to All definitions are 100% thorough and correct Community—chat to native speakers about words and phrases	Quick lookup Incognito lookup Key words and phrases listed based on context	Search-based on context No or little work [best we can do] Number translation Decimal Time Money Ordinal Sign dictionary

Large view.¹⁰⁰

Mission accomplished. Now we had a tentative 1.0 definition. Now we knew what this app would be about. You can read more about our user experience mapping exercise¹⁰¹ on our blog.

¹⁰⁰. <http://media.smashingmagazine.com/wp-content/uploads/2013/07/contexts-large.png>

¹⁰¹. <http://tapity.com/iphone-app-design/user-experience-mapping-strategic-design-part-3/>

THE DEFINITION

Based on the last exercise, we crafted a statement that defines the essence of the app:

An offline translation dictionary that gives instant access to words and definitions at 99¢ for multiple language pairs.

This statement helped to focus our development process. It became a litmus test for any cool feature idea we came up with during development. If the feature didn't support this statement, it didn't belong in 1.0.

Sketching The Interactions

It was time to get down and dirty and start to shape our abstract ideas into a blueprint.

We started by sketching general ideas on how the various screens could flow together. These days, I mostly stick to sketches, and I use tools such as [POP](http://popapp.in)¹⁰² to share ideas with remote team members and clients. At the time, however, we were using [OmniGraffle](http://www.omnigroup.com/products/omnigraffle/)¹⁰³ to create a rough prototype of the interactions.

DON'T MAKE ME THINK

Our goal at this stage was to solve our users' problems with an intuitive and easy to use interface. In essence, our job was to free users from having to think about the interface and instead to focus on the content.

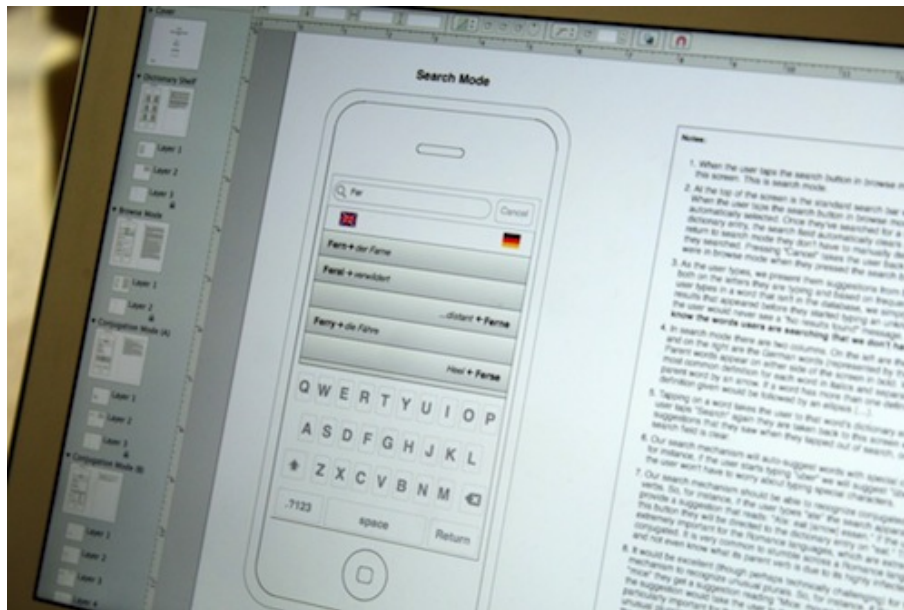
This is a huge topic and Steve Krug literally wrote the book on it, so if you haven't read [*Don't Make Me Think*](http://www.sensible.com/dmmt.html)¹⁰⁴, do so now. Seriously, it's a great book.

^{102.} <http://popapp.in>

^{103.} <http://www.omnigroup.com/products/omnigraffle/>

^{104.} <http://www.sensible.com/dmmt.html>

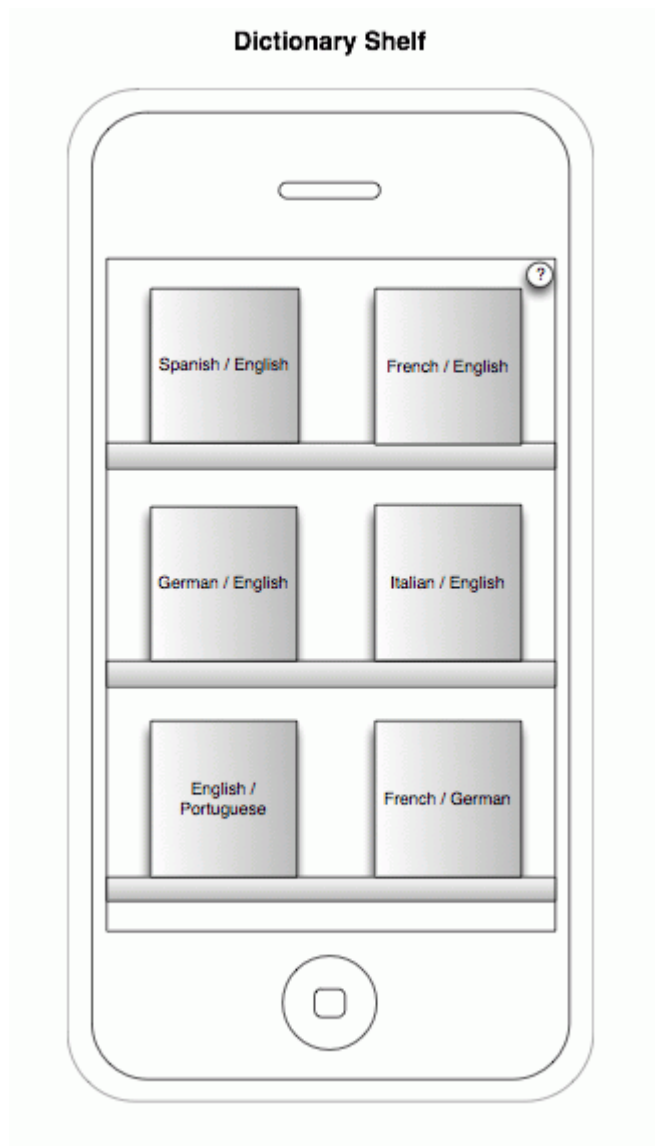
DON'T MAKE ME WORK



People don't like to work, so I am always looking for ways to save them from unnecessary keystrokes, taps and irrelevant information. The OmniGraffle wireframe above illustrates how we tried to serve that goal. We wanted our word lookup to be the fastest available; so, instead of providing on-the-fly search suggestions like most apps, we provide on-the-fly translations to those suggestions. We also found a way to solve the language-switching problem of most apps by enabling users to type in either language and displaying the results for one language on the left and the other language on the right.

THINK LIKE A HUMAN

Because this is an offline translation app, we wanted to give users the strong impression that the dictionaries are on their phones. We wanted the dictionaries to feel not like some abstract database in the cloud, but rather like physical dictionaries that they can access anytime, anywhere. We used the metaphor of a shelf with books to quickly communicate this to users.



We wanted our dictionaries to feel physical.

While touch interfaces have matured, and users no longer need interfaces to look like physical objects in order to relate to them, sometimes physical metaphors can set expectations and convey feelings that purely digital interfaces cannot.

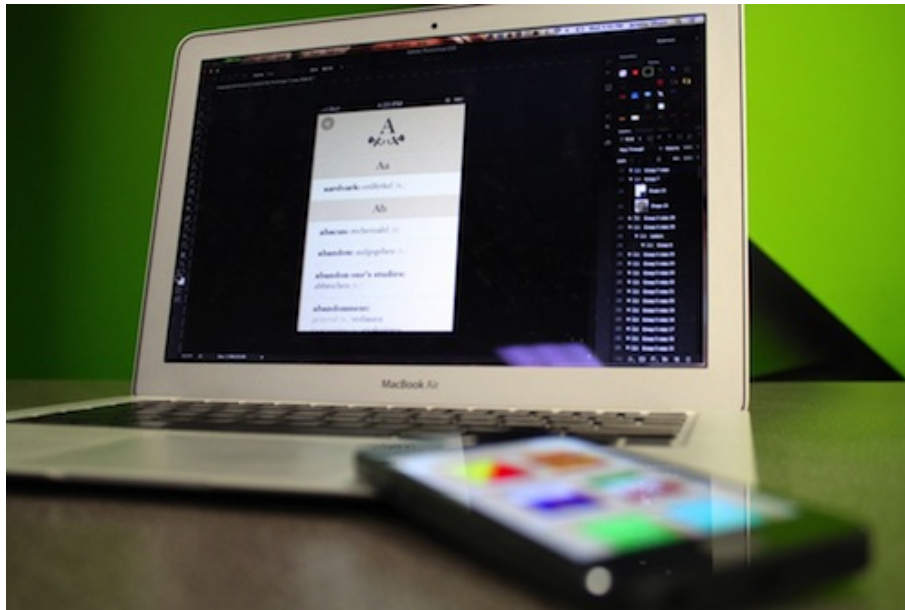
RELENTLESS EXPLORATION

Note that these wireframes are ugly on purpose. This stage has nothing to do with visual design. We don't jump right into Photoshop, because the ugly sketches help us to focus on the interaction problems and enable us to quickly explore hundreds of ideas.

Because sketching a rough idea takes only a few seconds, we go really crazy at this stage. The more ideas, the better. Leave no stone unturned to find the ideas worth pursuing.

Sometimes your first idea turns out to be the best, but the only way to prove that is to test all of the other ways of looking at the problem. I've gotten to meet the designers of some of my favorite apps, and one of the main commonalities among them is this: The secret to their amazing designs is a lot less about genius than about relentless exploration. They don't stop once they've found a good solution. They keep going until they've exhausted the possibilities.

Of Photoshop And Xcode



This is when things get really exciting. It's when ideas start to become reality. This is the point when we design and code the actual elements that our users will touch.

Some people think this stage is just about making pretty graphics, but that mindset leads to mediocrity. This stage is all about polish on all levels: interaction, usability and visual. This is when a good app becomes great.

While we hoped that our sketches and wireframes would provide a good outline, upon seeing things visually and playing with an actual coded prototype, we realized that we sometimes got it all wrong. Also, when we are merely sketching, it is difficult to imagine the creative details that will take our app beyond being usable and into the realm of fun. Once we start working with visual metaphors, colors and textures, dreaming up fun details becomes much easier.

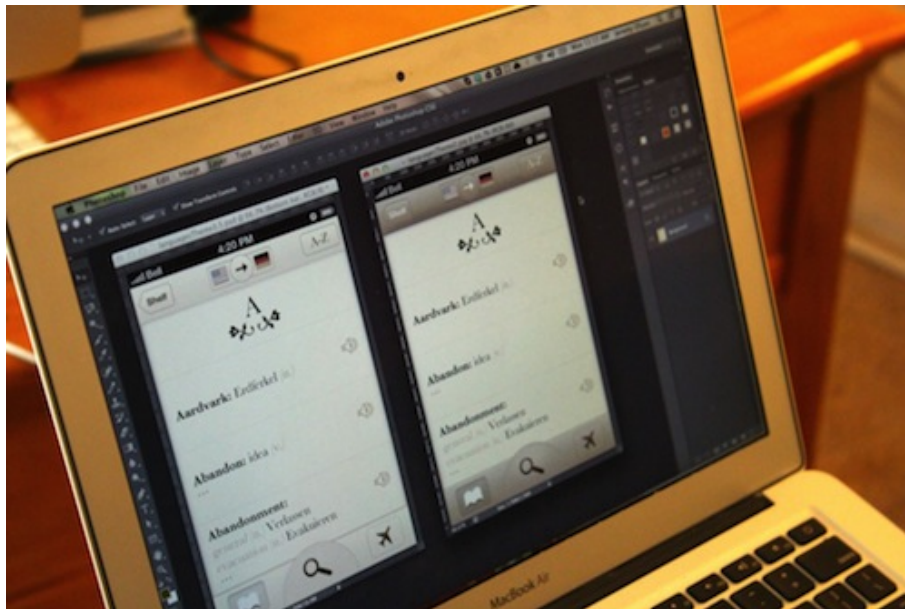
So, this is all about polish, polish, polish, and it is by far the most rewarding and time-consuming stage of app development.

ESTABLISHING THE THEME

Some people hate skeuomorphism¹⁰⁵, probably because mimicking elements from the real world in our digital interfaces can easily lead to overdesigned visuals and inconsistent interactions. However, skeuomorphism can be useful, fun and powerful if wielded carefully and deliberately. In fact, every app that contains buttons has skeuomorphism because buttons are borrowed straight from the real world. When used correctly, skeuomorphism provides much needed affordances that help users instantly understand how an app works.

With that in mind, we knew from the outset that we wanted to use the metaphor of physical books to reinforce the concept that these dictionaries are stored on the phone itself.

When working on the theme for the app, we generally iterated like crazy on two or three of the main screens until we were convinced that a certain look would work really well for the whole app.



Given that we were working towards a book-like theme, we explored an elegant earth-toned theme that let the content and typography shine. We continued to refine the theme along the way.

Side note: While iOS 7 rid itself of Corinthian leather and other such ornamental UI, and the industry is certainly shifting away from realistic interfaces at the moment, the best designers don't just follow trends. Trends are important, but we should consider all styles and techniques as tools in our toolbox and use them where they make sense. Granted, realistic UIs look quite dated now that iOS 7 arrived, but within a year

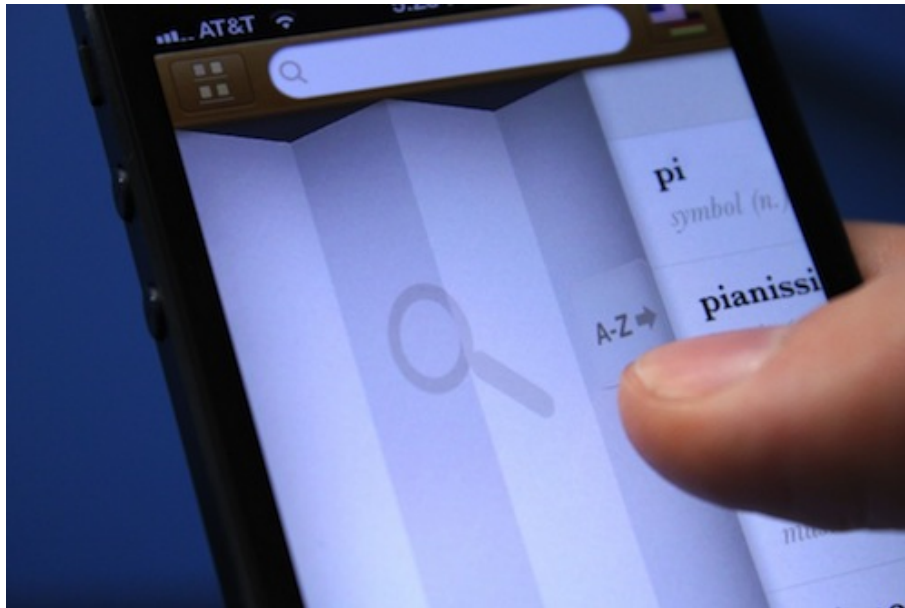
¹⁰⁵. <http://en.wikipedia.org/wiki/Skeuomorph>

or so, diversity in design styles will intensify as the novelty of iOS 7's minimal aesthetic begins to wear off.

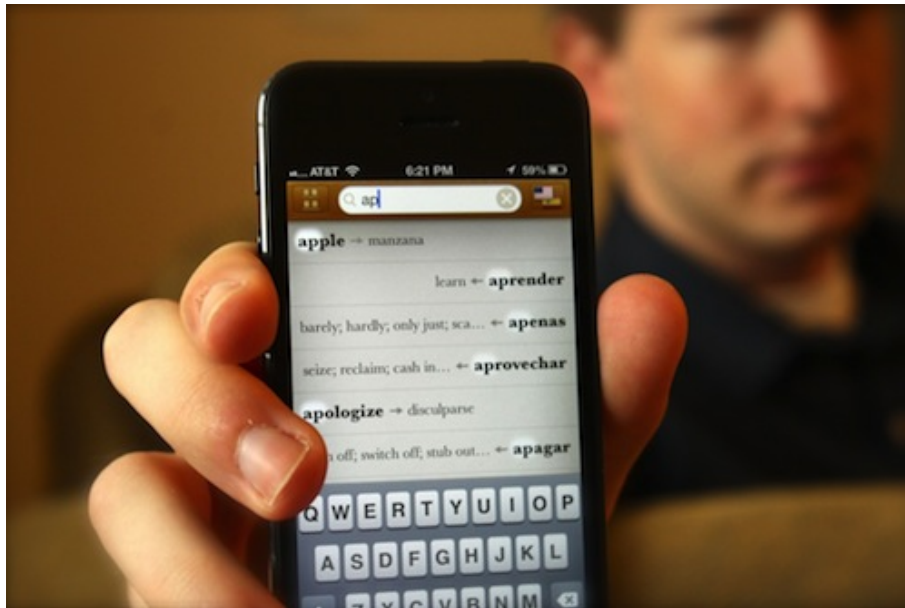
DELIGHT IS IN THE DETAILS

As we continued to flesh out all of the different screens, we looked for opportunities to delight our users with details that would make the app enjoyable to use. Part of this is just about making the app look nice, but you can also delight users by adding a fun transition, or make them laugh with some quirky copy, or save them work in surprising ways.

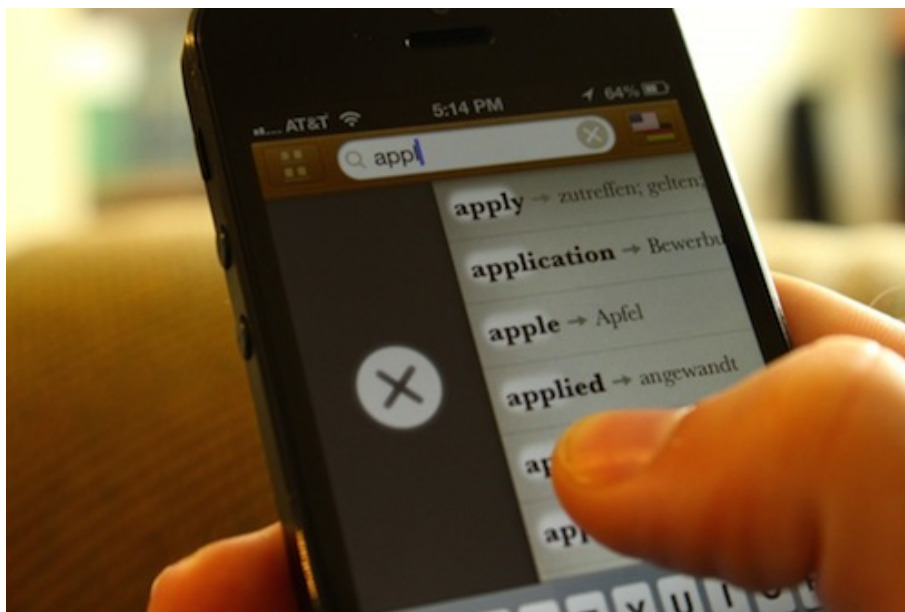
Take search:



The user can start searching by tapping the search bar. But reaching for that bar with a thumb can be a hassle, so we added the ability to swipe anywhere on the screen to unfold the search interface.



As the user types, the interface populates with suggested results and translations, highlighting the matching letters like a spotlight.



Folks who translate speech or passages of literature will look up a lot of words in rapid succession. We found that having to clear a search term by tapping the tiny “x” in the search field broke the flow and was physically strenuous, especially on the iPhone 5’s taller screen. So, we decided to let users swipe right to quickly clear a term – thus, allowing them to type a few letters, get the translation, and then swipe to begin typing another word all in a matter of seconds.

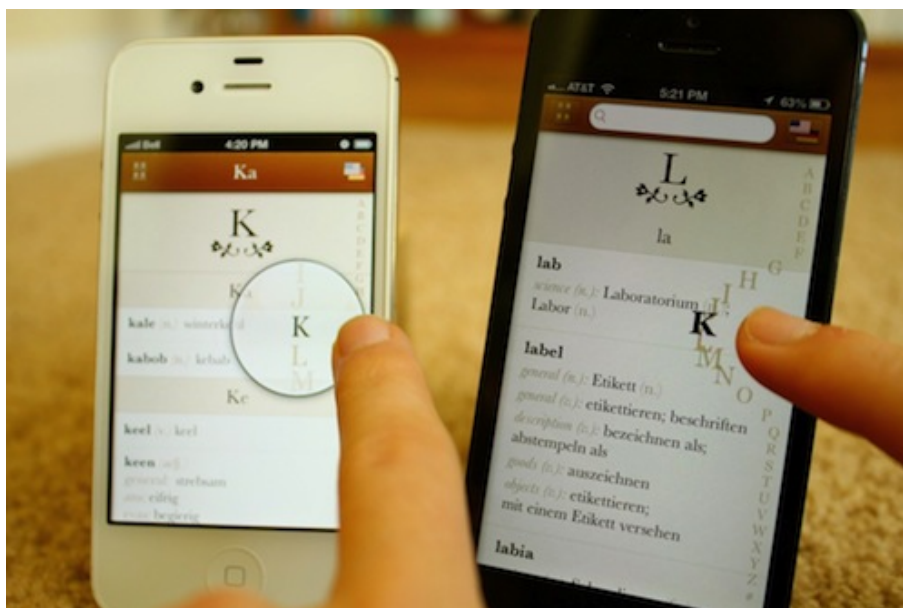
These kinds of details made all the difference when we were testing the app in the wild.

DESIGNER AND PROGRAMMER: CONSTANT COLLABORATION



Please, please, do not hand a programmer your design assets and expect a job well done. Not only do designers need to continually stay involved to ensure that their designs are implemented well, but using coded prototypes and testing them on users will inform the design in ways you can't imagine. I don't care what kind of genius designer you are: There is no substitute for testing a design and iterating on it.

Testing coded software exposes blatant weaknesses in the design that you may have never considered and shines light on areas where details could be added to make the experience more enjoyable. Because of this, I ended up doing even more design iterations after we had a coded prototype than I did before.



Changes at this stage are costly but extremely necessary.

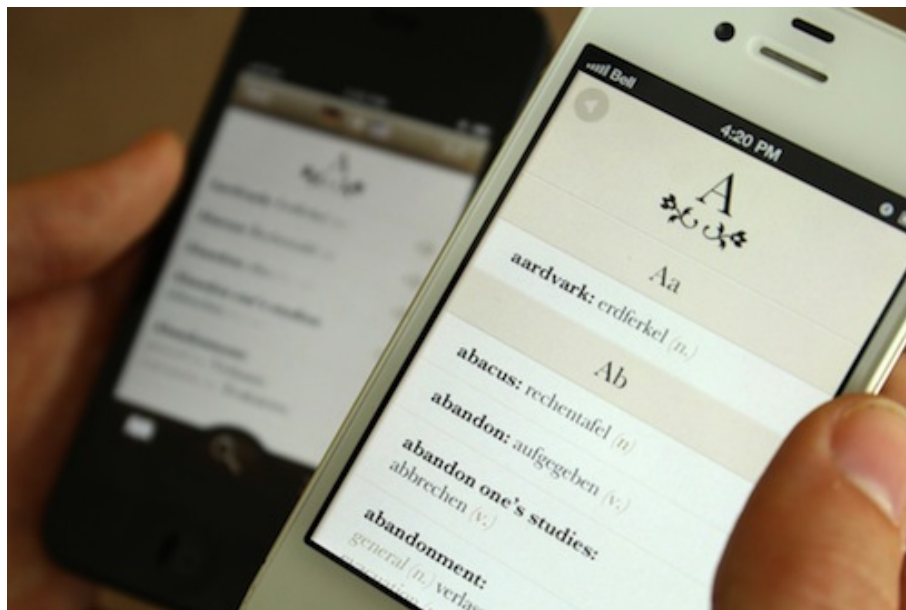
Programmers with a good design sense can also have great ideas. We wanted to make a better index. My mental model was some kind of magnifying glass. But Richard, Sonico's programmer, had a better idea: Magnify the letters themselves around your finger as you move your finger.

GESTURE EXPERIMENTS

As we started to code our first prototype, Impending and Realmac Software launched an app named Clear¹⁰⁶. No buttons, just gestures. Love it or hate it, it made a statement. I had never seen such an exercise in minimalism in my life.

It was a beautiful thing, and it inspired me to find ways to use gestures to improve Languages.

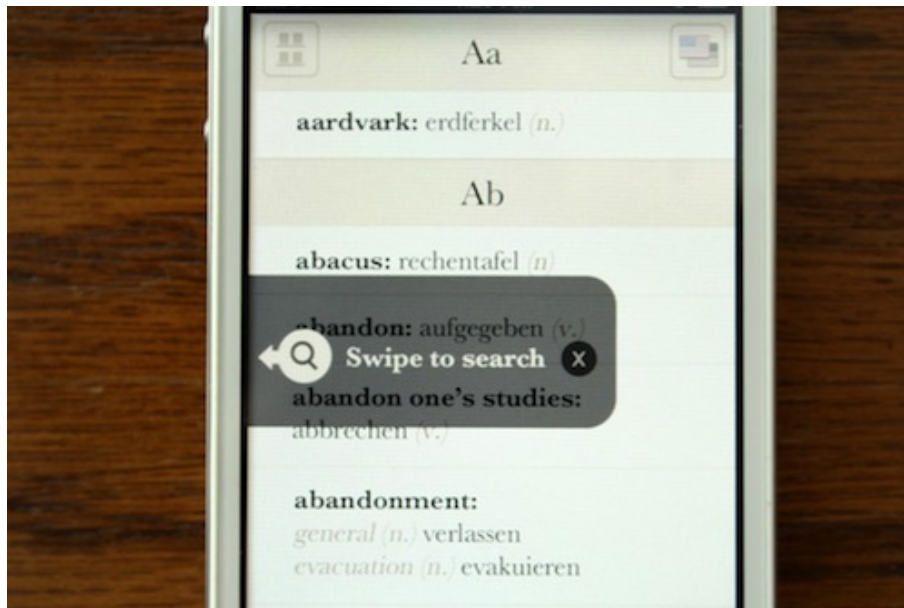
My first experiment was extreme: to create a fully gesture-based interface.



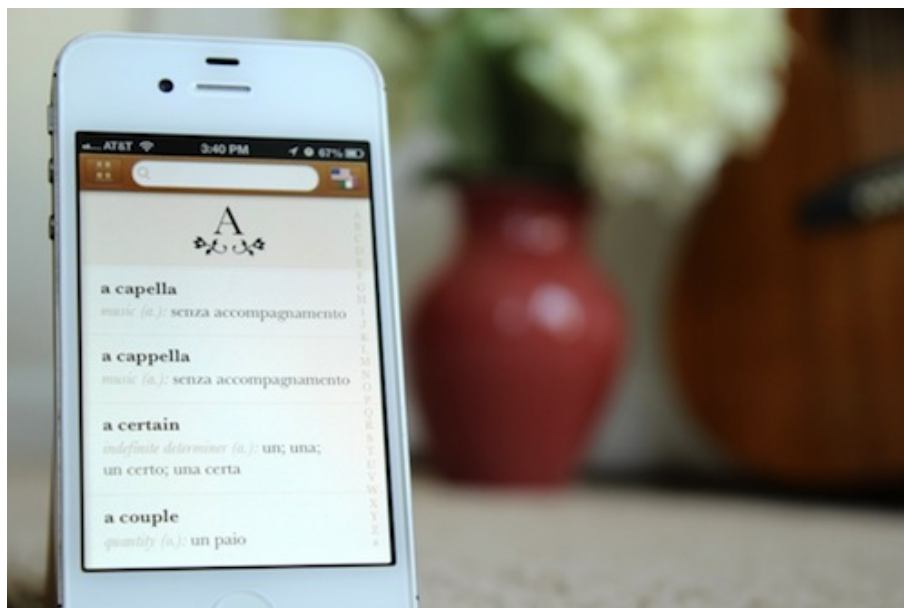
We replaced buttons with gestures. Swipe one way to search, and the other way to view the index.

As much as we loved the minimalism, we realized that we needed to teach our users about the gestures.

¹⁰⁶. <http://www.realmacsoftware.com/clear/>



We tried a number of approaches. But after doing a lot of usability testing, we realized that all of them had one major problem: Search, our most important feature, just wasn't blatantly obvious.



So, we bit the bullet and used buttons and affordances where they make sense, but we retained a lot of the gestures and minimalism that we had gained through the experiments. The result was an app that is super-intuitive and simple for beginners, but full of gestures that make life easier for power users.

TESTING

We made sure to get input on all aspects: usability, beauty, robustness. This included carrying out the following tasks:

- We observed friends, family and various strangers use the app. The key thing here is to propose tasks and ask questions about what they are thinking, but never to answer their questions. Probe into why they are confused about something, and let them figure it out for themselves. Watch for body language that indicates confusion or frustration, and note not only whether they were able to accomplish a task, but how effortless and enjoyable their experience was.
- We sought expert design reviews from top Apple designers, leading usability experts and fellow developers at conferences such as SxSW and WWDC.
- We posted screenshots to [Dribbble](http://www.dribbble.com/jerols)¹⁰⁷ to get feedback on visuals from leading designers around the world.
- We tested the app ourselves in real-world contexts using [TestFlight](http://www.testflightapp.com)¹⁰⁸.
- Finally, we thoroughly tested functionality and searched hundreds of words to catch bugs and ensure robustness and accuracy.

ICON



An app's icon means a lot. It is the first impression most users will get of an app, and we hope users will want to have it on their precious home screen.

¹⁰⁷. <http://www.dribbble.com/jerols>

¹⁰⁸. <http://www.testflightapp.com>

The first iteration of the Languages icon (the royal “L”) was simple but didn’t communicate much.

After a lot of brainstorming, we incorporated the idea of physical dictionaries on a shelf, since that was a major theme of the app. The icon was beautiful, but we couldn’t make it work well enough at small sizes. Additionally, a top designer at Apple recommended that we not use books because the app isn’t about reading.

Nuts. We really liked that icon, but we had to go back to the drawing board to find an instantly recognizable symbol that communicated the idea of translation and that wasn’t overused. A globe works pretty well, but we ultimately chose the “a” with an accent mark because it is unique and definitely communicates the idea of a foreign language. We lived with it on our home screens for a while, and it grew on us. Having aced the test of time, the icon proved to be the winner.

Next Step: Launch



We’ve come along way and learned a few things.

After a year of blood, sweat and tears, the product was finally where we wanted it to be. It didn’t have every feature that we intended to put in 1.0, but the features it did have were super-polished and ready for primetime. It was time to launch. I’ll cover our marketing and launch in the next chapter, so stay tuned. 🐼

Tale Of A Top-10 App, Part 2: Marketing And Launch

JEREMY OLSON 🍷

Our anticipation was building. Between defining our language translation app, sketching it out, obsessively designing and iterating, and juggling other projects — all covered in the first part of our case study — we had been working on Languages for close to a year. It was finally go time.

Even the coolest app in the world is doomed to swiftly descend into the abyss of obscurity if no one knows about it. So, part two of our journey is all about marketing. It turns out that you don't need a huge marketing budget to get into the top 10 in the App Store.

Seriously, Marketing?

I never thought of myself as a marketer. To me, “marketing” sounded like a dirty word. But when I was building my first app, Grades, I actively watched the indie developers who were consistently building hit apps, and sure enough, along with factors such as delightful design and amazing execution, a key factor to consistent success was great marketing.



The cool part is that, as I found out, marketing doesn't have to be about spam, meaningless buzzwords or sleazy Twitter follower generators, and it certainly doesn't have to be expensive. If your product is great,

marketing can be genuine, fun and free. Ultimately, it becomes a part of everything you do, without your even realizing it.

Marketing Starts From Day 1

We often have the misconception that marketing is what you do once you've launched a product. Too many people have come up to me after a talk or after reading one of my articles and said, "We just launched our app and got a few small blogs to write reviews, but those have generated only a few sales. What's the magic trick that gets an app on The Verge or featured by Apple?"

Sorry, no magic tricks here. Just a lot of hard work, and that takes a long time. The good news is that it's completely achievable, but you can't wait until your app is ready to launch to get started.

Do It Yourself

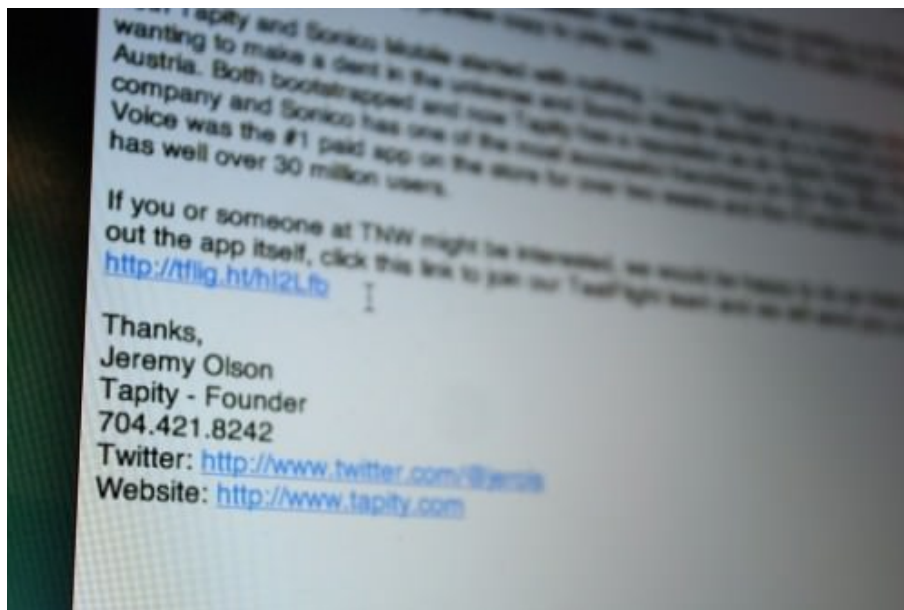
You would think that the ideal situation is to have a boatload of cash and hire a public relations (PR) firm.

Wrong.

You — the designer, the developer, the founder — are the very best person to market the app.

Why?

1. The press love to hear directly from the folks in the trenches.
2. A lot of PR agencies don't get the nuances of app marketing.
3. Relationships with journalists is the currency of PR. Even if a PR firm landed you some great articles, they would take those relationships with them, leaving you with no long-term benefits. If you do it yourself, you will be cultivating those relationships, making future launches a lot easier.



There are definitely some exceptions. I know some marketers who really understand the space and who deliver results. But in general, doing it yourself is better, especially if you are in this game for the long haul.

Make Friends

Later in this chapter, we'll talk about marketing tactics — crafting our pitch, drawing lists of blogs and so forth — but those tactics don't mean a whole lot unless you understand something very fundamental about grassroots marketing: It's all about the people.

The problem is that we view big websites like The Verge as monolithic black boxes into which we insert a pitch and hope for the best. But who runs The Verge? Real people with flesh and bones. Who runs the App Store? People. How do you get featured? By winning the hearts of those people.

These days, getting my apps featured isn't that difficult. Why? Because over the past four years, I have made friends with people at almost every major venue that features apps, including Apple. Now, whenever I release a product, I can direct message a few people on Twitter and send a few emails and, assuming my product is at the caliber that my friends have come to expect from me, be fairly confident that launch day will go pretty well. That's the power of people-focused marketing.



Remarkable iPhone Devs¹⁰⁹.

Remember that just a few years ago, I was an obscure college freshman, starting at ground zero. No apps. No friends. Not even any connections. So, if I can do it, anyone can.

How did I do it?

- **List influential people.**

It started by making a list of influential folks in the iOS industry: app designers and developers (David Barnard¹¹⁰, Phill Ryu¹¹¹, etc.), technology journalists who cover apps, folks at Apple. Limiting this list to tech journalists would have been a huge mistake. In the app world, the top players know each other, so a connection with one influential app developer could very well lead to a connection with Apple or the press — and the iOS community is chockfull of great people who love to share their experiences. I've compiled Twitter lists for top app makers¹¹², app journalists¹¹³, and Apple employees¹¹⁴.

- **Actively monitor.**

I actively watched what these people were doing, looking for opportunities to connect in tasteful ways. Suppose Phill Ryu had a question on Twitter that I could answer, or Ellis Hamburger wrote an article on The Verge that I really enjoyed — I took those opportunities to engage with

¹⁰⁹. <https://twitter.com/jerols/remarkable-iphone-devs>

¹¹⁰. <http://www.twitter.com/drarnard>

¹¹¹. <http://www.twitter.com/phillryu>

¹¹². <https://twitter.com/jerols/remarkable-iphone-devs>

¹¹³. <https://twitter.com/jerols/best-of-the-press>

¹¹⁴. <https://twitter.com/jerols/apple-employees>

folks on their turf. I never spammed them with my apps or my blog — once someone likes you, they'll naturally check out your stuff.

- **Go where they are.**

Everyone is on Twitter, so I made sure that my profile there looked interesting and I got active. I went to conferences such as the Worldwide Developers Conference (WWDC) and South by Southwest and actively got out of my shell to meet the people around me and the people on my lists. Because I was a designer, I also hung out in communities like [Dribbble](http://dribbble.com/jerols)¹¹⁵ and [Forrst](http://forrst.com/people/jerols)¹¹⁶ to connect with other designers. Finally, I occasionally emailed influential people — I didn't spam them, but I would say something nice about an article they wrote or ask a close-ended question about something they were interested in (close-ended because you are not likely to hear back on questions that require long answers).

- **Do cool stuff.**

If you aren't building cool apps that are innovative and well designed, then gaining respect in the app community will be tough, and respect is a pillar of friendship. You don't necessarily need to have apps in the App Store, though. I made a lot of my connections before my first app ever launched. A big reason for that is that I blogged and tweeted about what I was learning from app developers about making successful apps, and I posted very introspective thoughts about developing my first app. That caught people's attention, and doing that well is a great way to gain respect before you've actually "made it" in the industry. This is important, because if you've gotten someone's attention by talking to them at a conference or by engaging them on Twitter, then you need to gain their respect in order for them to take the leap of following you on Twitter (which, as weird as it sounds, can be a big deal in establishing a lasting friendship).

- **Cultivate long-term friendships.**

Once I established a connection with someone, I cultivated it by discussing issues on Twitter, in email and at conferences, giving them beta access to my apps, requesting feedback on projects (especially high-profile ones), and saying something nice about them once in a while — I don't know one human being who doesn't have at least a smidgen of ego.

This might sound a bit manipulative when broken down like that, but it's really just about being a friendly, interesting person and going out of your shell a bit to meet the great folks at the top of our industry.

¹¹⁵. <http://dribbble.com/jerols>

¹¹⁶. <http://forrst.com/people/jerols>

I like how David Barnard put it:

There are some really cool people in this industry. If all you do is beg for coverage, you're missing out on getting to know some great folks.

If you follow this advice, then the advice below will produce great results. If not, I can't guarantee a whole lot of fruit from the more tactical stuff.

Launch Plan: Concentrated Blast

The App Store is not the Web. On the Web, you can launch softly, get some articles here and there, and build your user base over time.

If you are trying to sell a mass-market app for \$0.99, then scattered marketing doesn't work. To get hundreds of thousands of downloads, you need to make an effort and get the app to "chart," to blast up the App Store's sales charts. Once your ranking is high enough, you will experience a snowball effect because the charts are where most people look for apps.

With that in mind, we prepared as much as we could before the launch in order to make the blast as concentrated as possible.

Building Buzz

A couple months before the launch, we started building anticipation. The goal was to collect email addresses, Twitter and Facebook accounts that we could reach come launch day.

BLOGGING

We started by writing blog posts about the making of Languages. We posted our [UX mapping process](#)¹¹⁷, [progress reports](#)¹¹⁸, [rants on gestures](#)¹¹⁹, among other things. Blogging played a large role in building our brand and credibility with previous apps, but a lesser role for this launch. It was probably still worth it, though, because continuing to build your blog's readership is always good.

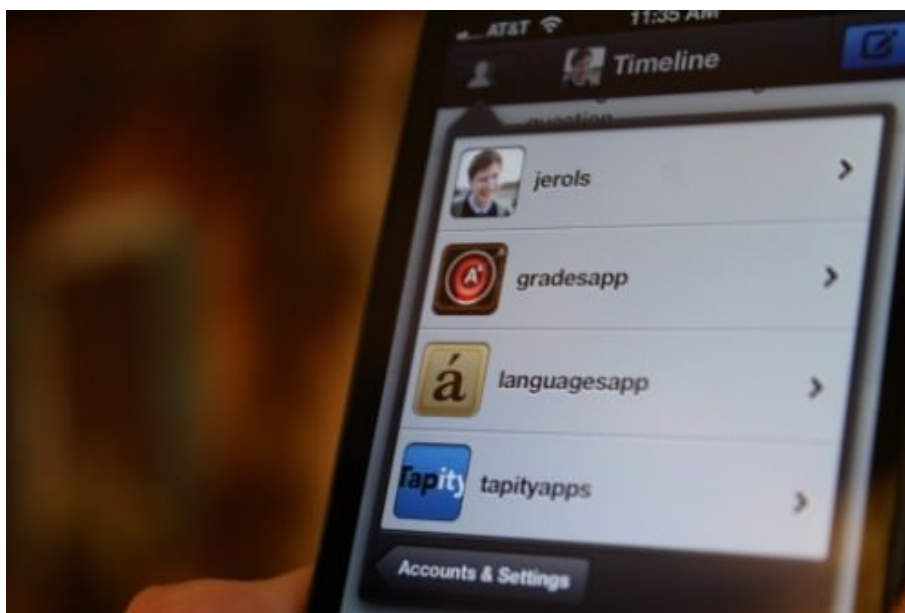
¹¹⁷. <http://tapity.com/iphone-app-design/user-experience-mapping-strategic-design-part-3/>

¹¹⁸. <http://tapity.com/iphone-app-design/where-we-are-with-languages/>

¹¹⁹. <http://tapity.com/iphone-app-design/gestures/>



TWITTER



I like to use my personal Twitter account (@jerols¹²⁰) most of the time, but we decided to create a @languagesapp¹²¹ Twitter account to make it easy for us and others to refer to the app without having to link to it (“Putting the final touches on @languagesapp”). My followers could then click the Twitter user name to find out more information about it. While it was hard to remember to post consistently to the @language-sapp account, I posted a teaser screenshot once in a while and retweet-

¹²⁰. <http://www.twitter.com/jerols>

¹²¹. <http://www.twitter.com/languagesapp>

ed it from my personal and company accounts, giving the @language-sapp account more exposure.

With fairly minimal effort, we were able to garner about 500 followers for the account.

FACEBOOK



Languages App on Facebook¹²².

I'm honestly not a Facebook marketing expert and didn't spend a lot of time experimenting with it, but I did set up a page and post screenshots and status updates once in a while. It got about 50 likes, half of which were friends. I could have experimented more and gotten a lot more likes, but it didn't seem like the most effective use of my time.

DRIBBBLE

As a designer, it was only natural to post my UI mockups to Dribbble¹²³, a design community in which you can get great feedback and sometimes get viral sharing of your designs. I'm not a Dribbble pro, and my designs have never made it to Dribbble's famous front page, but I did get some valuable feedback and a few thousands views to boot.

¹²². <https://www.facebook.com/pages/Languages-for-iOS/405844942765838?ref=ts&fref=ts>

¹²³. <http://dribbble.com/jerols>



TEASER WEBSITE



We always build a teaser website¹²⁴ a month or two before launch to collect email addresses of people who would like to know when the app launches. About 200 folks subscribed. Not something to write home about, but every little bit helps. Some of the subscribers were influencers in the industry, which is always good.

¹²⁴. <http://www.languagesapp.com/old>

Pitching The Press

Two weeks to go. Pitch time. I was fortunate enough to have been able to show an early build to some of my journalist friends at the WWDC, so I already had some really hot doors to knock on.

WARM CALLS



Most of the journalists I knew personally already followed me on Twitter, so, in most cases, the best way to initiate the pitch was to send them a direct message on Twitter, teasing them with a screenshot and asking whether they wanted to play with a prerelease build. These guys are super-busy, so I didn't hear back from all of them, but many of them were on board.

We used TestFlight¹²⁵ to send out prerelease builds to the press. Most of them had TestFlight accounts already, so getting a build to them was pretty easy. Later on, once the app was finalized and approved by Apple, we replaced TestFlight with promo codes in our pitches, which are more convenient. (Promo codes are valid after the app is approved by Apple but before it is available to the public.)

After sending out the builds, I sent the journalists personalized thank-you notes, which also communicated the basic pitch and hinted at some advanced features to check out.

Our programming partner, Sonico, also had some great connections. So, they contacted the journalists with whom they had the strongest relationships, and we contacted ours.

¹²⁵ <https://testflightapp.com>

COLD CALLS



While warm calls are ideal, I’ve had some success with cold calls — i.e. pitching journalists and individuals with whom I don’t already have a strong connection. I created a spreadsheet with all of the websites I wanted to pitch and prioritized them by how influential I perceived them to be. For the really big websites, I identified the best individual journalists to contact and sent them a personalized pitch. For the smaller websites, I generally just wrote to their **tips@** email address and only personalized the “Hey, [website’s name] team” greeting.

As expected, we got only a few responses from the cold calls, but it was still worth it.

WHEN PERSONALIZATION GOES WRONG

I generally personalize the greeting and first sentence of my emails, especially for journalists whom I know. Unfortunately, this time, I had a “brilliant” idea: to refer to the journalist’s website in the middle of the email, to make it even more personal!

Bad idea. I forgot about that reference in one of my pitches and left in the name of a competing website. Talk about embarrassing and unprofessional. From then on, I decided to keep it simple. Don’t include personalized references anywhere but at the beginning of the email, and always double-check before sending! Thankfully, the journalist was a friend and took it well.

THE PITCH

Rather than talk about what I think makes for a good pitch, I'll just show you mine.

Hey [journalist's name],

[Personalized message to the journalist and introduction to the pitch.]

Launching in a week, Languages is the fastest, most reliable and most affordable offline translation app to hit the App Store. Most translation apps break down when you need them most: traveling without an Internet connection. Languages gives you the peace of mind that you'll never be stuck without reliable translation. While other offline translators charge \$5 to \$20 for a single language, Languages packs 12 common language pairs into one app for \$0.99.

- Price: \$0.99
- Launch Date: Next Thursday, October 25
- Video Demo: <http://vimeo.com/51481324>
- Website: <http://www.languagesapp.com>
- Press Kit: <http://www.languagesapp.com/LanguagesPressKit.zip>
- Promo Code: WJ9HXJNAPL73

Please let me know if you would be interested in doing a review, and I would be happy to provide more information or answer any questions.

Thanks,
 Jeremy Olson
 Tapity – Founder
 704.421.8242
 Twitter: <http://www.twitter.com/@jerols>
 Website: <http://www.tapity.com>

It certainly wasn't perfect (if you want to learn how to make a perfect pitch, you should probably read *Pitch Perfect*¹²⁶ by Erica Sadun and Steve Sande), but it was pretty short, gave the basic value proposition pretty quickly, and provided the key information that journalists like. It also included the press kit, a folder full of screenshots, the app icon at various sizes, and a press release.

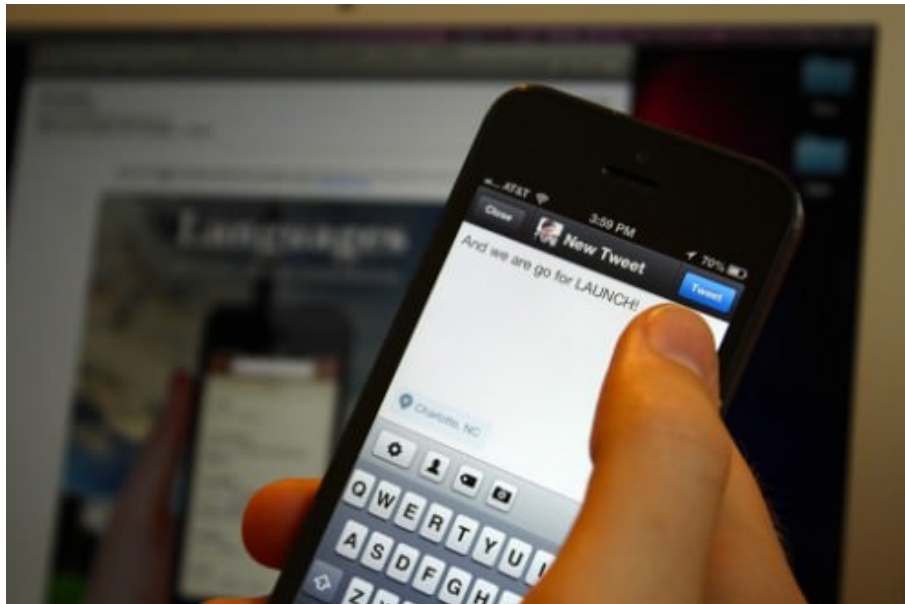
¹²⁶ <http://www.amazon.com/Pitch-Perfect-Practical-Professional-ebook/dp/Boo83U5DTC>

VIDEO OR NO?

A lot of great apps launch with an amazing launch video. It's a great idea if you can afford to hire a video producer or can shoot it yourself well... like, really well. But whether you make a video for the public or not, *definitely* make a short one for the press. Journalists would rather spend 30 seconds watching your app in action than spend a few minutes trying to “get it” for themselves on their phones.

In our case, we didn't have time to shoot a professional video, so we just made a short video¹²⁷ for the press, showing off all of the features. It was pretty rough, but it did the job.

Go Time



“Sorry, can’t talk right now. I’m super-busy launching Languages.” That was typical launch-day conversation. You would think that launching an app is a pretty good excuse to do nothing. It’s not. While I might have responded to some urgent questions from journalists or taken an emergency call with Apple here and there, this was really a day to kick back and watch all of our hard work pay off.

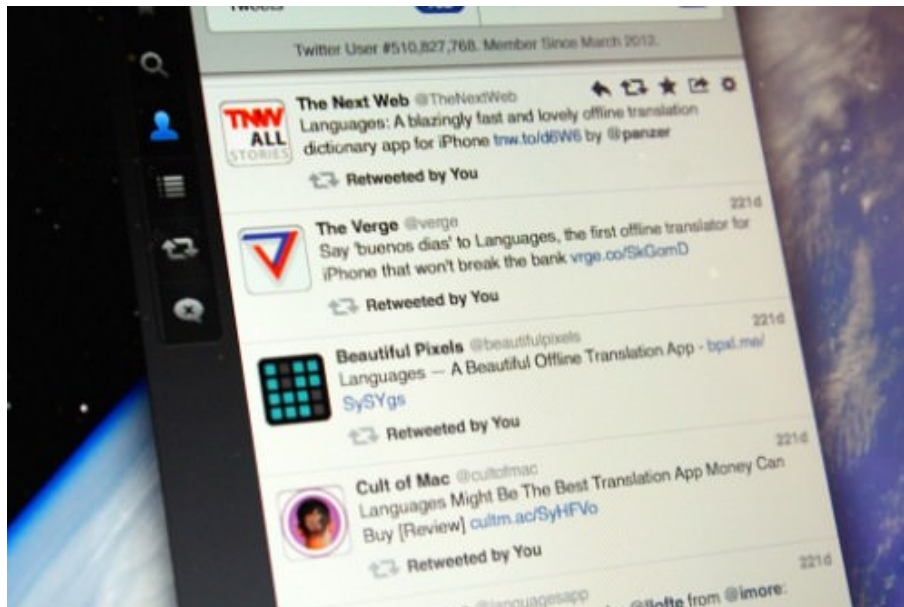
The majority of the day consisted of one activity: refreshing. Refreshing Twitter search for “Languages app” to see who was talking about it. Refreshing my “mentions” feed to see who was saying “Congrats!” Refreshing Google’s “past hour” search results for “languages” to uncover any new reviews.

¹²⁷. <http://vimeo.com/51481324>

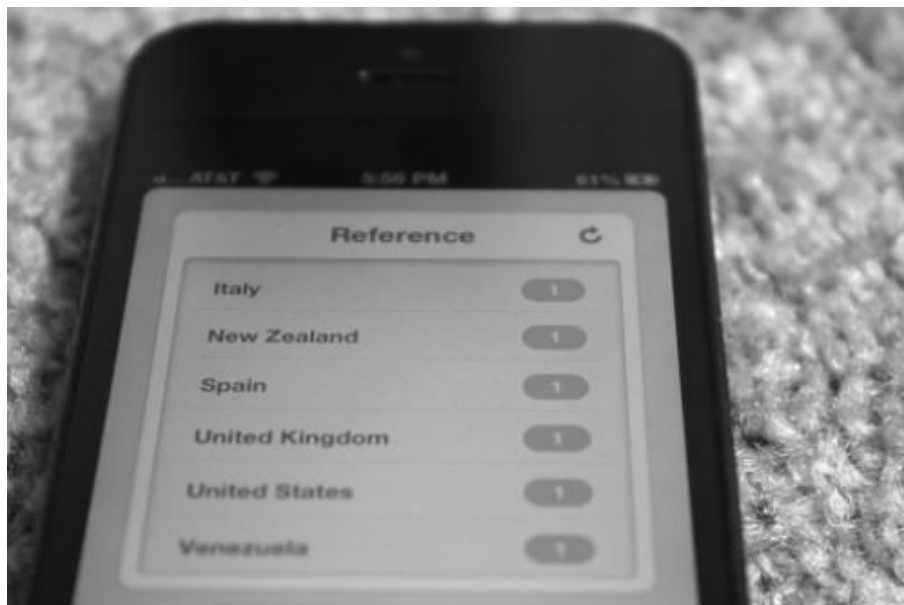
Most of all, launch day can be characterized quite accurately as one of desperate and incessant refreshing of the App Store to see where Languages stood in the infamous top charts. It is the ultimate stroke of the ego after a year of blood, sweat and tears. This, folks, is what dreams are made of.

RESULTS

Aside from a few blunders on my part from miscommunications about when journalists could start posting reviews, the launch went even better than I expected.



We got some amazing reviews from some of my favorite journalists.



We blasted the announcement to Sonico's substantial mailing list, the Languages mailing list, our blog and social media channels, resulting in a lot of buzz on Twitter and a swift climb up the App Store's charts. Before long, we reached the number one spot in the "Reference" category.

APPLE



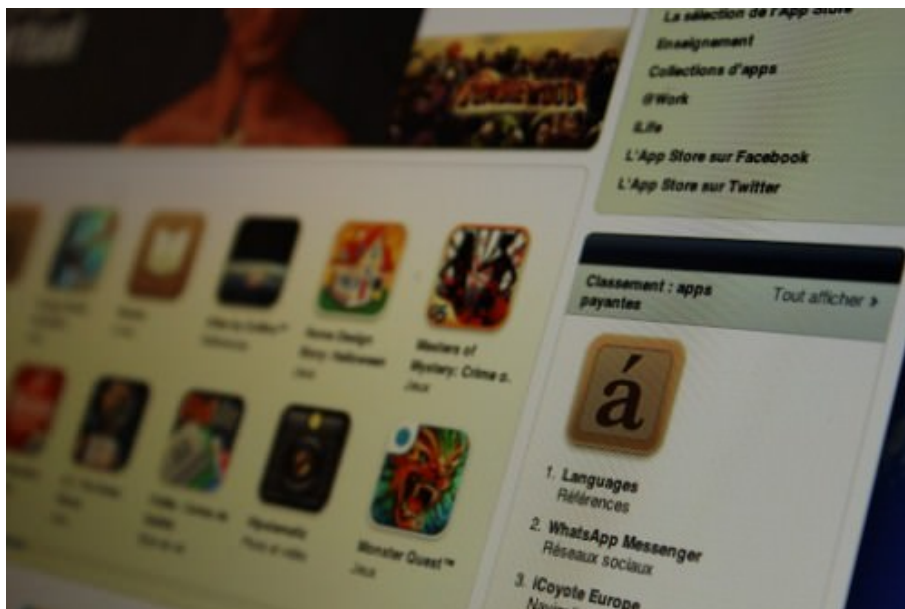
Our relationship with Apple is one of my company's greatest assets and one that took time to cultivate. Getting featured prominently by Apple is one of the best things that can happen to your app. It will usually get you far more downloads than any coverage from the press.

When people ask me how I've gotten all of my apps to be featured by Apple, I usually recommend a big marketing push during launch in order to get Apple's attention. This time, though, we were fortunate enough to have piqued Apple's interest a few months before Languages launched. We were fairly certain that Apple would feature us, but were pleasantly surprised to be featured as the "Editor's Choice" in many countries and very prominently in the "New & Noteworthy" section in others.

Blast Off!



Concentrating all of our marketing on launch day really paid off. With over a dozen press reviews, thousands of tweets and retweets, a blast to our mailing lists, and an important feature from Apple, Languages blasted into the top-25 overall paid apps on launch day. The snowball effect kicked in, and by day two, Languages peaked at number 5 in the US.



Results from the other countries were staggering, with the app hitting the number one spot in France and a few other European countries. Over 70% of our sales came from outside the US. Suffice it to say, app localization, the App Store description, and the screenshots that showed most of the major European languages paid off.

Whew, what a ride! I could finally check off one of my life goals: beating all incarnations of Angry Birds in the charts, if only for a short period of time.

Aftermath

Languages managed to stay in the top 100 for a couple weeks, and then slowly but surely made its inevitable descent down, down, down into the App Store abyss. Fortunately, Languages found a pretty comfortable resting spot around number 30 (and then later number 50) in the “Reference” category’s top-100 list.



We were pretty happy with the few hundred downloads it was getting per day. It wasn't going to make us millionaires, but most of the money you make from these kinds of hit-based apps will be from major launches — i.e. major updates, the iPad version, etc. — so, the decent consistent sales are a bonus.

SUPPORT

Support doesn't just mean updating the app with new features. One of the most time-consuming parts of making a popular app is addressing all of the bugs, feature requests and comments you receive via email and Twitter every day. During launch week, we were getting hundreds of emails a day. No joke.

Thankfully, the comments were mainly positive, asking for new languages and features.

I tried to respond to a lot of the emails, but pretty soon we realized that support was a full-time job, so we shifted responsibility to one of

Sonico's support guys. That being said, we did look through the emails to find recurring bugs reports and feature requests so that we could address them in the next version.

I expected users to ask for things like voice support and conjugation, but it turns out the most popular feature request was to be able to customize the bookshelf and to hide dictionaries not being used. That blew my mind, but we made sure to include that in the next version.

The most negative feedback was from users who said that, while the interface was great, they couldn't find certain words in the dictionary. We learned from the Apple Maps fiasco that the robustness of an app's content can easily overshadow the beauty of the interface. While our content wasn't perfect, it was pretty decent, so fortunately we didn't reenact the Apple Maps disaster, but we did need to work on it going forward.

RAISING THE PRICE

We decided to go for the mass market at launch and set the app at \$0.99. If you are trying to hit the top charts, the app needs to be an impulse buy.



Having gone with a launch-sale approach, we raised the price to \$2.99 once the app fell out of the top 100. Raising the price when the app is falling out of the charts might seem counterintuitive, but you actually make more money that way. The only reason we ever set it at \$0.99 was to get the app to the top charts and to keep it there as long as possible. Once it falls out of the charts, the people who will find it are not impulse shoppers; they will have to actively look for it and will be willing to pay a higher price for it. This has worked out well for us.

Conclusion

Wow, that was quite a journey. If you've browsed to the end for a reader's digest, you're in luck. The big lessons to take away are these:

1. Yes, marketing matters! Start way before your app launches.
2. Do it yourself by making friends in the industry.
3. Build buzz through social media.
4. Concentrate the marketing in blasts.
5. Spend time getting your pitch right.

The App Store is certainly not easy street. Building a hit app takes a ton of hard work, and there are never any guarantees. That being said, it sure is a fun ride, and there is still plenty of opportunity for those who are willing to work at it. So, get out there and build something great! 🍷

Gone In 60 Frames Per Second: A Pinterest Paint Performance Case Study

ADDY OSMANI & PAUL LEWIS 🍷

Today we'll discuss how to improve the paint performance of your websites and Web apps. This is an area that we Web developers have only recently started looking at more closely, and it's important because it could have an impact on your user engagement and user experience.

Frame Rate Applies To The Web, Too

Frame rate is the rate at which a device produces consecutive images to the screen. A low frames per second (FPS) means that individual frames can be made out by the eye. A high FPS gives users a more responsive feel. You're probably used to this concept from the world of gaming, but it applies to the Web, too.

Long image decoding, unnecessary image resizing, heavy animation and data processing can all lead to dropped frames, which reduces the frame rate, resulting in janky pages. We'll explain what exactly we mean by "jank" shortly.

Why Care About Frame Rate?

Smooth, high frame rates drive user engagement and can affect how much users interact with your website or app.

At EdgeConf earlier this year, Facebook confirmed¹²⁸ this when it mentioned that in an A/B test, it slowed down scrolling from 60 FPS to 30 FPS, causing engagement to collapse. That said, if you can't do high frame rates and 60 FPS is out of reach, then you'd at least want something smooth. If you're doing your own animation, this is one benefit of using `requestAnimationFrame`¹²⁹: the browser can dynamically adjust to keep the frame rate normal.

In cases where you're concerned about scrolling, the browser can manage the frame rate for you. But if you introduce a large amount of

¹²⁸. <http://smashed.by/edge-conf>

¹²⁹. <https://developer.mozilla.org/en-US/docs/Web/API/window.requestAnimationFrame>

jank, then it won't be able to do as good a job. So, try to avoid big hitches, such as long paints, long JavaScript execution times, long anything.

Don't Guess It, Test It!

Before getting started, we need to step back and look at our approach. We all want our websites and apps to run more quickly. In fact, we're arguably paid to write code that runs not only correctly, but quickly. As busy developers with deadlines, we find it very easy to rely on snippets of advice that we've read or heard. Problems arise when we do that, though, because the internals of browsers change very rapidly, and something that's slow today could be quick tomorrow.

Another point to remember is that your app or website is unique, and, therefore, the performance issues you face will depend heavily on what you're building. Optimizing a game is a very different beast to optimizing an app that users will have open for 200+ hours. If it's a game, then you'll likely need to focus your attention on the main loop and heavily optimize the chunk of code that is going to run every frame. With a DOM-heavy application, the memory usage might be the biggest performance bottleneck.

Your best option is to learn how to measure your application and understand what the code is doing. That way, when browsers change, you will still be clear about what matters to you and your team and will be able to make informed decisions. So, no matter what, don't guess it, test it!¹³⁰

We're going to discuss how to measure frame rate and paint performance shortly, so hold onto your seats!

Note: Some of the tools mentioned in this chapter require Chrome Canary¹³¹, with the "Developer Tools experiments" enabled in `about:flags`. (We – Addy Osmani and Paul Lewis – are engineers on the Developer Relations team at Chrome.)

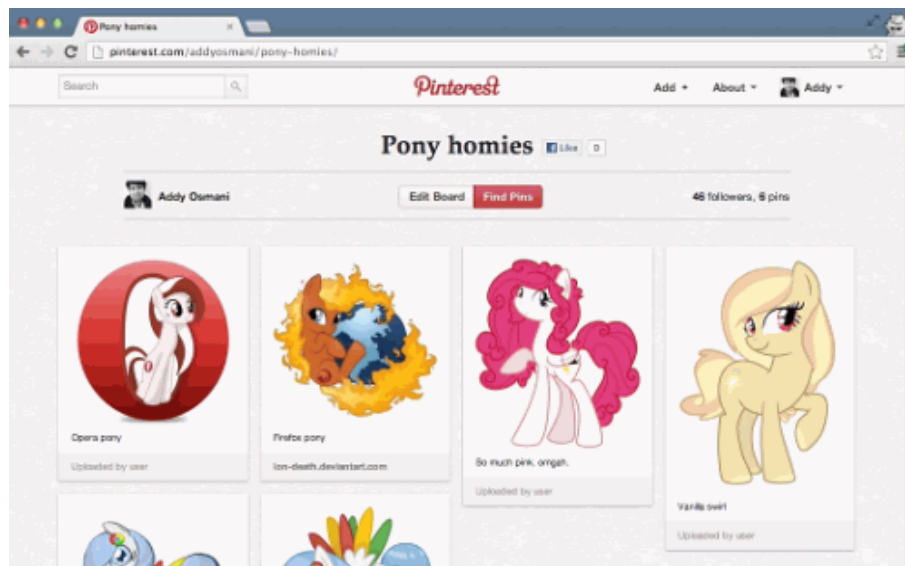
Case Study: Pinterest

The other day we were on Pinterest¹³², trying to find some ponies to add to our pony board (Addy loves ponies!). So, we went over to the Pinterest feed and started scrolling through, looking for some ponies to add.

¹³⁰. <http://aerotwist.com/blog/dont-guess-it-test-it/>

¹³¹. <https://www.google.com/intl/en/chrome/browser/canary.html>

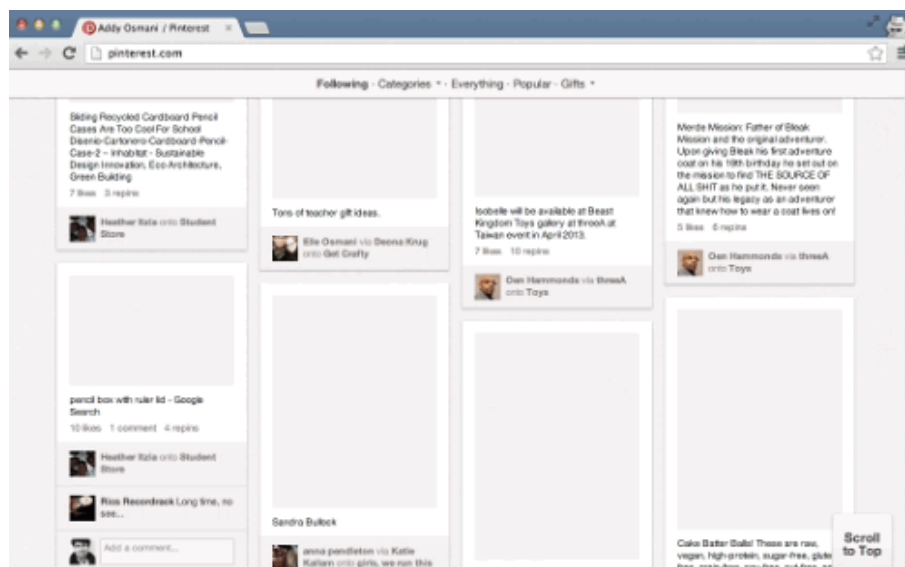
¹³². <http://pinterest.com/>



Addy adding some ponies to his Pinterest board, as one does.

JANK AFFECTS USER EXPERIENCE

The first thing we noticed as we scrolled was that scrolling on this page doesn't perform very well – scrolling up and down takes effort, and the experience just feels sluggish. When they come up against this, users get frustrated, which means they're more likely to leave. Of course, this is the last thing we want them to do!



Pinterest showing a performance bottleneck when a user scrolls.

- Animated GIF¹³³

¹³³. http://media.smashingmagazine.com/wp-content/uploads/2013/05/slow_scroll2.gif

This break in consistent frame rate is something the Chrome team calls “jank,” and we’re not sure what’s causing it here. You can actually notice some of the frames being drawn as we scroll. But let’s visualize it! We’re going to open up Frames mode and show what slow looks like there in just a moment.

Note: What we’re really looking for is a consistently high FPS, ideally matching the refresh rate of the screen. In many cases, this will be 60 FPS, but it’s not guaranteed, so check the devices you’re targeting.

Now, as JavaScript developers, our first instinct is to suspect a memory leak as being the cause. Perhaps some objects are being held around after a round of garbage collection. The reality, however, is that very often these days JavaScript is not a bottleneck. Our major performance problems come down to slow painting and rendering times. The DOM needs to be turned into pixels on the screen, and a lot of paint work when the user scrolls could result in a lot of slowing down.

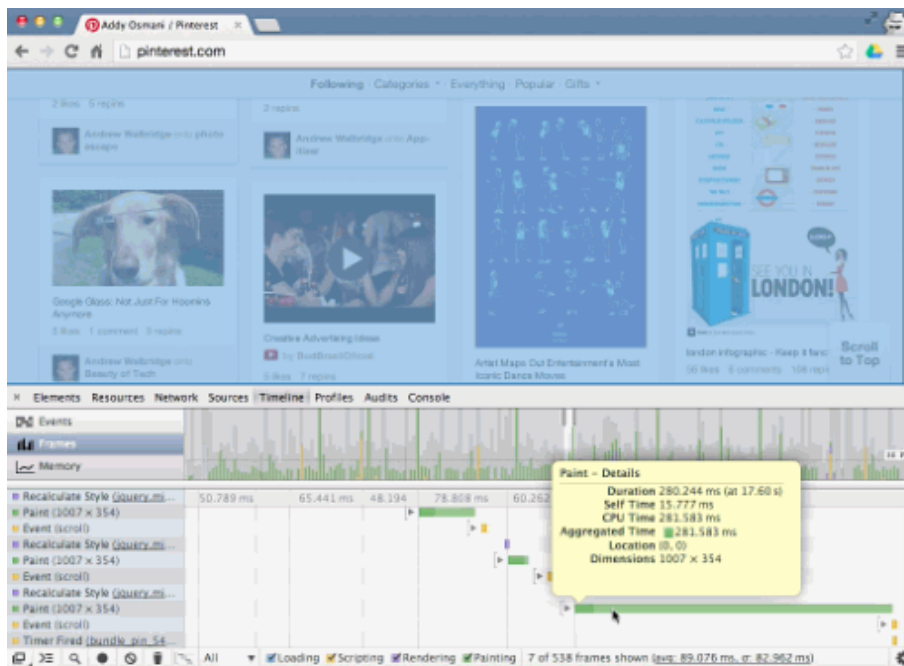
Note: [HTML5 Rocks](http://www.html5rocks.com/en/tutorials/speed/scrolling/) specifically discusses some of the [causes of slow scrolling](http://www.html5rocks.com/en/tutorials/speed/scrolling/)¹³⁴. If you think you’re running into this problem, it’s worth a read.

Measuring Paint Performance

FRAME RATE

We suspect that something on this page is affecting the frame rate. So, let’s go open up Chrome’s Developer Tools and head to the “Timeline” and “Frames” mode to record a new session. We’ll click the record button and start scrolling the page the way a normal user would. Now, to simulate a few minutes of usage, we’re going to scroll just a little faster.

¹³⁴. <http://www.html5rocks.com/en/tutorials/speed/scrolling/>



Using Chrome's Developer Tools to profile scrolling interactions. [Larger view](#)¹³⁵.

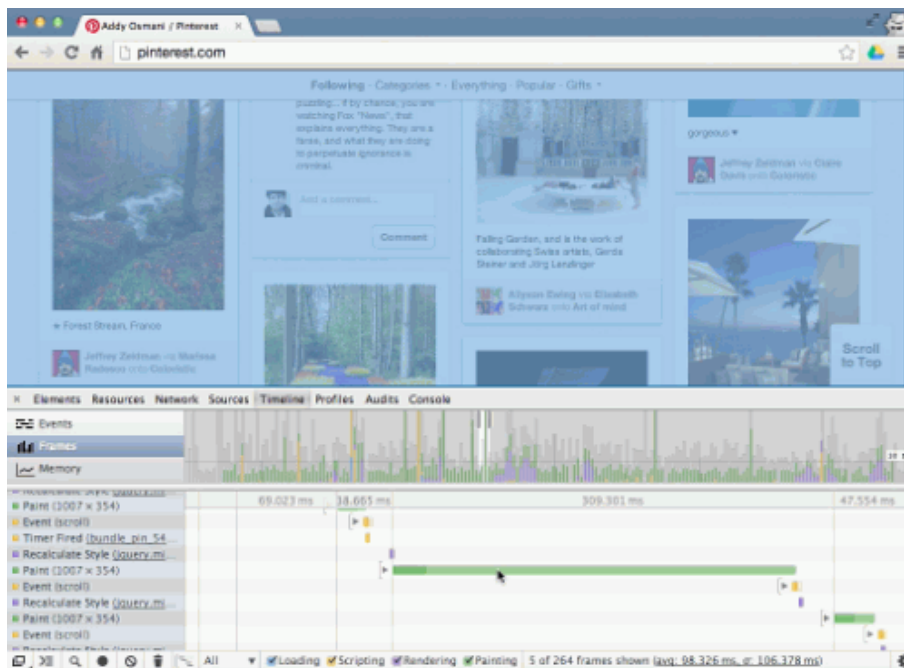
- [Animated GIF](#)¹³⁶

Up, down, up, down. What you'll notice now in the summary view up at the top is a lot of purple and green, corresponding to painting and rendering times. Let's stop recording for now. As we flip through these various frames, we see some pretty hefty "Recalculate Styles" and a lot of "Layout."

If you look at the legend to the very right, you'll see that we've actually blown our budget of 60 FPS, and we're not even hitting 30 FPS either in many cases. It's just performing quite poorly. Now, each of these bars in the summary view correspond to one frame — i.e. all of the work that Chrome has to do in order to be able to draw an app to the screen.

¹³⁵. <http://media.smashingmagazine.com/wp-content/uploads/2013/05/Screenshot-2013-05-15-at-17-57-48.png>

¹³⁶. <http://media.smashingmagazine.com/wp-content/uploads/2013/05/performance.gif>



Chrome's Developer Tools showing a long paint time. [Larger view](#)¹³⁷.

- [Animated GIF](#)¹³⁸

FRAME BUDGET

If you're targeting 60 FPS, which is generally the optimal number of frames to target these days, then to match the refresh rate of the devices we commonly use, you'll have a 16.7-millisecond budget in which to complete everything – JavaScript, layout, image decoding and resizing, painting, compositing – everything.

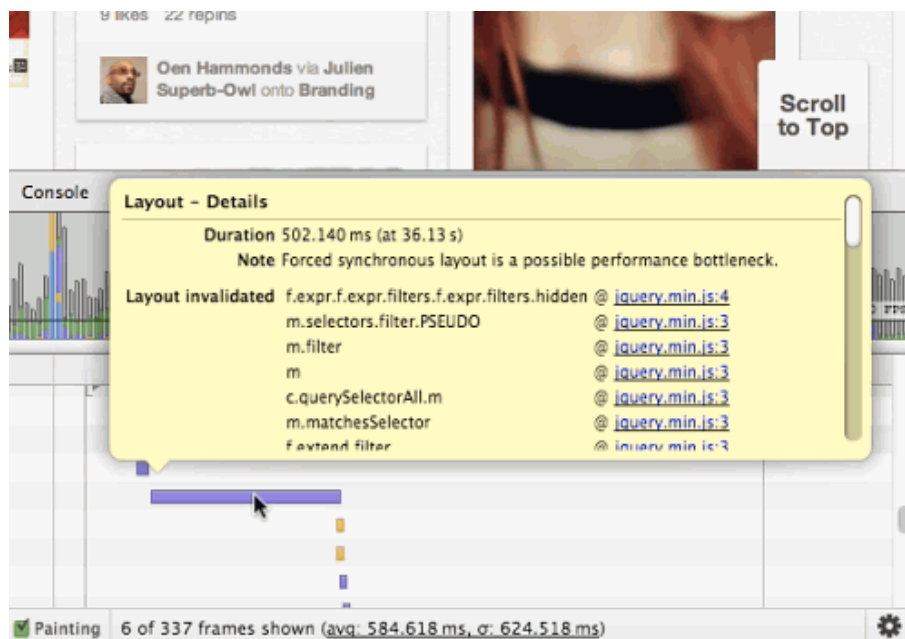
Note: A constant frame rate is our ideal here. If you can't hit 60 FPS for whatever reason, then you're likely better off targeting 30 FPS, rather than allowing a variable frame rate between 30 and 60 FPS. In practice, this can be challenging to code because when the JavaScript finishes executing, all of the layout, paint and compositing work still has to be done, and predicting that ahead of time is very difficult. In any case, whatever your frame rate, ensure that it is consistent and doesn't fluctuate (which would appear as stuttering).

If you're aiming for low-end devices, such as mobile phones, then that frame budget of 16 milliseconds is really more like 8 to 10 milliseconds. This could be true on desktop as well, where your frame budget might be lowered as a result of miscellaneous browser processes. If you blow this budget, you will miss frames and see jank on the page. So, you likely have somewhere nearer 8 to 10 milliseconds, but be sure to

¹³⁷. <http://media.smashingmagazine.com/wp-content/uploads/2013/05/screen4343431.png>

¹³⁸. <http://media.smashingmagazine.com/wp-content/uploads/2013/05/selection1.gif>

test the devices you're supporting to get a realistic idea of your budget.



An extremely costly layout operation of over 500 milliseconds. [Larger view](#)¹³⁹.

- [Animated GIF](#)¹⁴⁰

Note: We've also got an article on how to use the Chrome Developer Tools to find and fix rendering performance issues¹⁴¹ that focuses more on the timeline.

Going back to scrolling, we have a sneaking suspicion that a number of unnecessary repaints are occurring on this page with `onscroll`.

One common mistake is to stuff just way too much JavaScript into the `onscroll` handlers of a page — making it difficult to meet the frame budget at all. Aligning the work to the rendering pipeline (for example, by placing it in `requestAnimationFrame`) gives you a little more head-room, but you still have only those few milliseconds in which to get everything done.

The best thing you can do is just capture values such as `scrollTop` in your scroll handlers, and then use the most recent value inside a `requestAnimationFrame` callback.

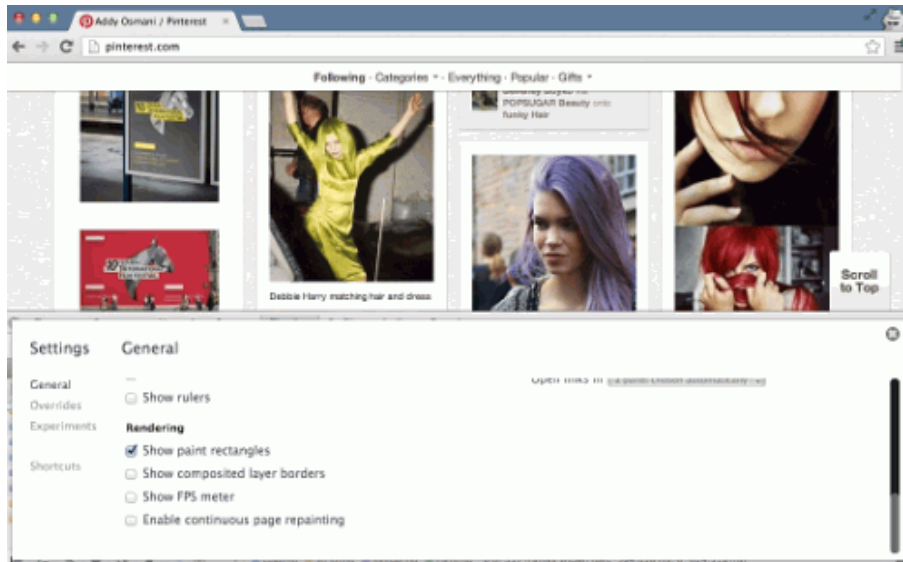
¹³⁹. <http://media.smashingmagazine.com/wp-content/uploads/2013/05/Screenshot-2013-03-25-at-14.34.26.png>

¹⁴⁰. <http://media.smashingmagazine.com/wp-content/uploads/2013/05/highlight.gif>

¹⁴¹. <http://addyosmani.com/blog/performance-optimisation-with-timeline-profiles/>

PAINT RECTANGLES

Let's go back to **Developer Tools** → **Settings** and enable “Show paint rectangles.” This visualizes the areas of the screen that are being painted with a nice red highlight. Now look at what happens as we scroll through Pinterest.



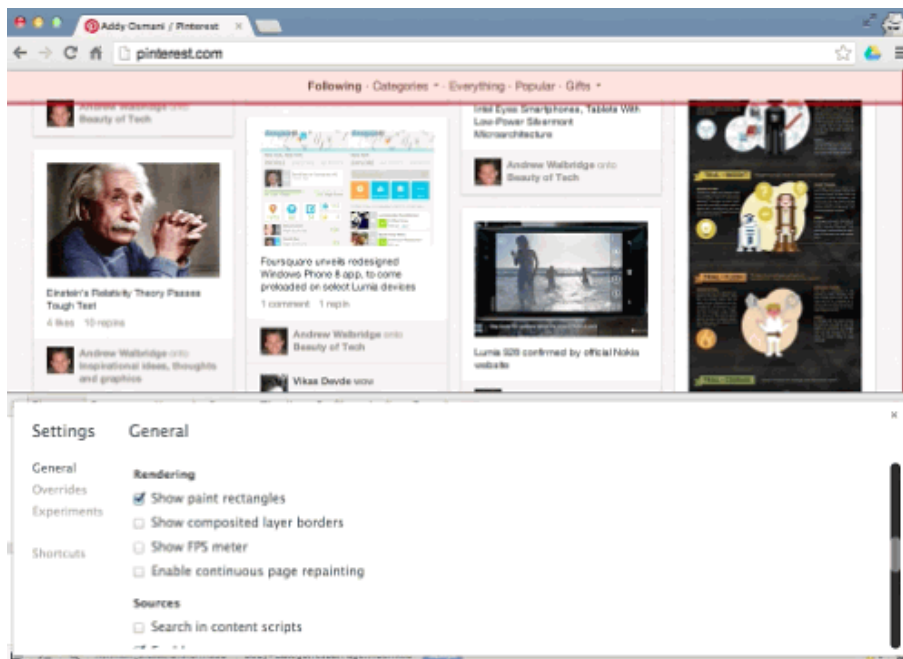
Enabling Chrome Developer Tools’ “Paint Rectangles” feature. [Larger view](#)¹⁴².

- Animated GIF¹⁴³

Every few milliseconds, we experience a big bright flash of red across the entire screen. There seems to be a paint of the whole screen every time we scroll, which is potentially very expensive. What we want to see is the browser just painting what is new to the page — so, typically just the bottom or top of the page as it gets scrolled into view. The cause of this issue seems to be the little “scroll to top” button in the lower-right corner. As the user scrolls, the fixed header at the top needs to be repainted, but so does the button. The way that Chrome deals with this is to create a union of the two areas that need to be repainted.

¹⁴². <http://media.smashingmagazine.com/wp-content/uploads/2013/05/Screen-Shot-2013-03-25-at-14.35.17.png>

¹⁴³. <http://media.smashingmagazine.com/wp-content/uploads/2013/05/rects.gif>



Chrome shows freshly painted areas with a red box. [Larger view](#)¹⁴⁴.

In this case, there is a rectangle from the top left to top right, but not very tall, plus a rectangle in the lower-right corner. This leaves us with a rectangle from the top left to bottom right, which is essentially the whole screen! If you inspect the button element in Developer Tools and either hide it (using the **H** key) or delete it and then scroll again, you will see that only the header area is repainted. The way to solve this particular problem is to move the scroll button to its own layer so that it doesn't get unioned with the header. This essentially isolates the button so that it can be composited on top of the rest of the page. But we'll talk about layers and compositing in more detail in a little bit.

The next thing we notice has to do with hovering. When we hover over a pin, Pinterest paints an action bar containing “Repin, comment and like” buttons — let's call this the action bar. When we hover over a single pin, it paints not just the bar but also the elements underlying it. Painting should happen only on those elements that you expect to change visually.

¹⁴⁴. <http://media.smashingmagazine.com/wp-content/uploads/2013/05/Screenshot-2013-05-15-at-19.00.12.png>



A cause for concern: full-screen flashes of red indicate a lot of painting. [Larger view](#)¹⁴⁵.

- [Animated GIF](#)¹⁴⁶

There's another interesting thing about scrolling here. Let's keep our cursor hovered over this pin and start scrolling the page again.

Every time we scroll through a new row of images, this action bar gets painted on yet another pin, even though we don't mean to hover over it. This comes down more to UX than anything else, but scrolling performance in this case might be more important than the hover effect during scrolling. Hovering amplifies jank during scrolling because the browser essentially pauses to go off and paint the effect (the same is true when we roll out of the element!). One option here is to use a set-Timeout with a delay to ensure that the bar is painted only when the user really intends to use it, an approach we covered in "[Avoiding Unnecessary Paints](#)"¹⁴⁷. A more aggressive approach would be to measure the **mouseenter** or the mouse's trajectory before enabling hover behaviors. While this measure might seem rather extreme, remember that we are trying to avoid unnecessary paints at all costs, especially when the user is scrolling.

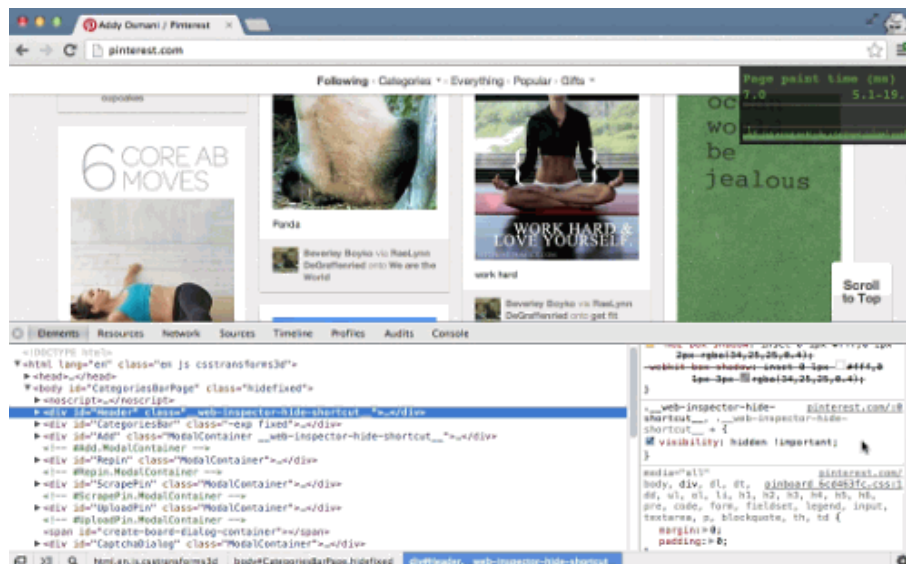
OVERALL PAINT COST

We now have a really great workflow for looking at the overall cost of painting on a page; go back into Developer Tools and "Enable continuous page repainting." This feature will constantly paint to your screen so that you can find out what elements have costly paint times. You'll get this really nice black box in the top corner that summarizes paint times, with the minimum and maximum also displayed.

¹⁴⁵. <http://media.smashingmagazine.com/wp-content/uploads/2013/05/Screenshot-2013-03-25-at-14.35.46.png>

¹⁴⁶. <http://media.smashingmagazine.com/wp-content/uploads/2013/05/scroll.gif>

¹⁴⁷. <http://www.html5rocks.com/en/tutorials/speed/unnecessary-paints/>



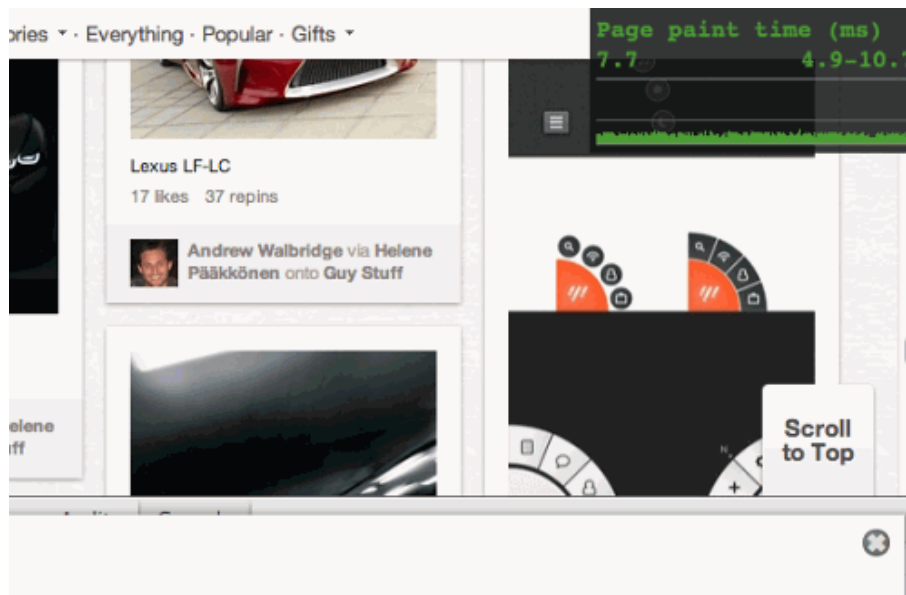
Chrome's "Continuous Page Repainting" mode helps you to assess the overall cost of a page. Larger view¹⁴⁸.

- Animated GIF¹⁴⁹

Let's head back to the "Elements" panel. Here, we can select a node and just use the keyboard to walk the DOM tree. If we suspect that an element has an expensive paint, we can use the **H** shortcut key (something recently added to Chrome) to toggle visibility on that element. Using the continuous paint box, we can instantly see whether this has a positive effect on our pages' paint times. We should expect it to in many cases, because if we hide an element, we should expect a corresponding reduction in paint times. But by doing this, we might see one element that is especially expensive, which would bear further scrutiny!

¹⁴⁸. <http://media.smashingmagazine.com/wp-content/uploads/2013/05/screen-shot43234242.png>

¹⁴⁹. <http://media.smashingmagazine.com/wp-content/uploads/2013/05/painthud.gif>



The “Continuous Page Repainting” chart showing the time taken to paint the page.

- [Animated GIF](#)¹⁵⁰

For Pinterest’s website, we can do it to the categories bar or to the header, and, as you’d expect, because we don’t have to paint these elements at all, we see a drop in the time it takes to paint to the screen. If we want even more detailed insight, we can go right back to the timeline and record a new session to measure the impact. Isn’t that great? Now, while this workflow should work great for most pages, there might be times when it isn’t as useful. In Pinterest’s case, the pins are actually quite deeply nested in the page, making it hard for us to measure paint times in this workflow.

Luckily, we can still get some good mileage by selecting an element (such as a pin here), going to the “Styles” panel and looking at what CSS styles are being used. We can toggle properties on and off to see how they effect the paint times. This gives us much finer-grained insight into the paint profile of the page.

Here, we see that Pinterest is using [box-shadow](#)¹⁵¹ on these pins. We’ve optimized the performance of [box-shadow](#) in Chrome over the past two years, but in combination with other styles and when heavily used, it could cause a bottleneck, so it’s worth looking at.

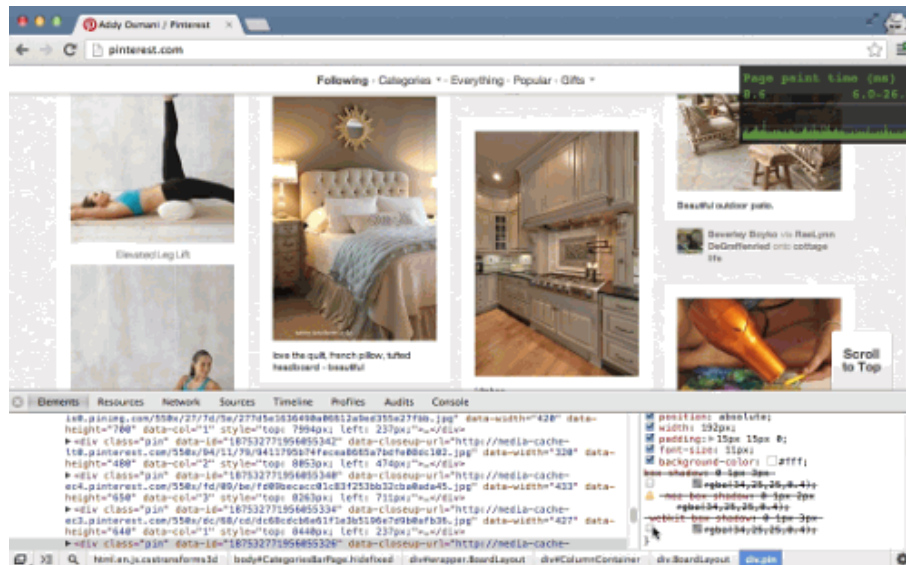
Pinterest has reduced continuous paint mode times by 40% by moving [box-shadow](#) to a separate element that doesn’t have [border-radius](#). The side effect is slightly fuzzy-looking corners; however, it is

150. <http://media.smashingmagazine.com/wp-content/uploads/2013/05/cont.gif>

151. <http://www.html5rocks.com/en/tutorials/speed/css-paint-times/>

barely noticeable due to the color scheme and the low **border-radius** values.

Note: You can read more about this topic in “CSS Paint Times and Page Render Weight¹⁵².”



Toggleing styles to measure their effect on page-rendering weight. Larger view¹⁵³.

- Animated GIF¹⁵⁴

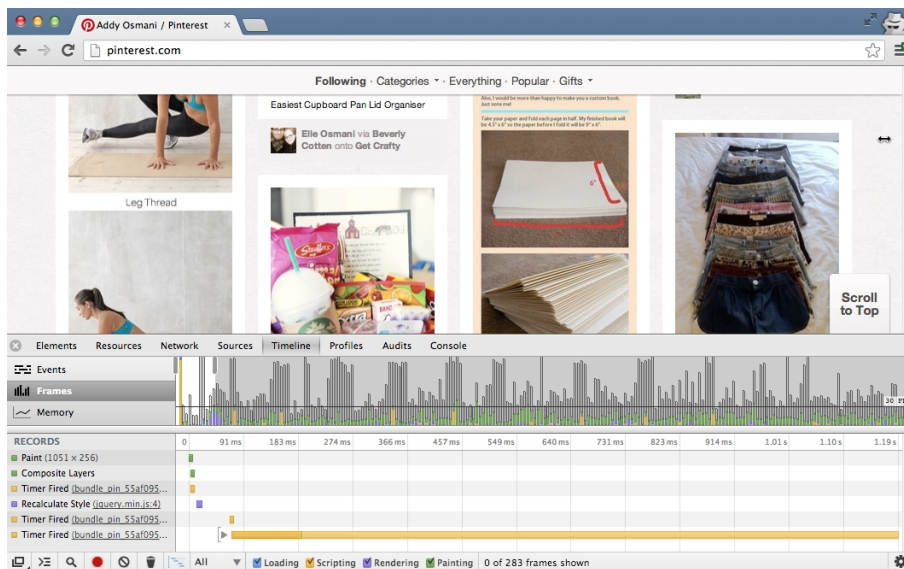
Let’s disable **box-shadow** to see whether it makes a difference. As you can see, it’s no longer visible on any of the pins. So, let’s go back to the timeline and record a new session in which we scroll the same way as we did before (up and down, up and down, up and down). We’re getting closer to 60 FPS now, and that’s just from one change.

Public service announcement: We’re absolutely not saying don’t use **box-shadow** — by all means, do! Just make sure that if you have a performance problem, measure correctly to find out what your own bottlenecks are. Always measure! Your website or application is unique, as will any performance bottleneck be. Browser internals change almost daily, so measuring is the smartest way to stay up to date on the changes, and Chrome’s Developer Tools makes this really easy to do.

¹⁵². <http://www.html5rocks.com/en/tutorials/speed/css-paint-times/>

¹⁵³. <http://media.smashingmagazine.com/wp-content/uploads/2013/05/Screenshot-2013-03-25-at-15.47.40.png>

¹⁵⁴. <http://media.smashingmagazine.com/wp-content/uploads/2013/05/box.gif>



Using Chrome Developer Tools to profile is the best way to track browser performance changes. [Larger view](#)¹⁵⁵.

Note: Eberhard Grather recently wrote a detailed post on “Profiling Long Paint Times With DevTools’ Continuous Painting Mode¹⁵⁶,” which you should spend some quality time with.

Another thing we noticed is that if you click on the “Repin” button, do you see the animated effect and the lightbox being painted? There’s a big red flash of repaint in the background. It’s not clear from the tooling if the paint is the white cover or some other affected being area. Be sure to double check that the paint rectangles correspond to the element or elements that you think are being repainted, and not just what it looks like. In this case, it looks like the whole screen is being repainted, but it could well be just the white cover, which might not be all that expensive. It’s nuanced; the important thing is to understand what you’re seeing and why.

HARDWARE COMPOSITING (GPU ACCELERATION)

The last thing we’re going to look at on Pinterest is GPU acceleration. In the past, Web browsers have relied pretty heavily on the CPU to render pages. This involved two things: firstly, painting elements into a bunch of textures, called layers; and secondly, compositing all of those layers together to the final picture seen on screen.

Over the past few years, however, we’ve found that getting the GPU involved in the compositing process can lead to some significant speed-

¹⁵⁵. [images/Screen-Shot-2013-03-25-at-14.39.25.png](#)

¹⁵⁶. <http://updates.html5rocks.com/2013/02/Profiling-Long-Paint-Times-with-DevTools-Continuous-Painting-Mode>

ing up. The premise is that, while the textures are still painted on the CPU, they can be uploaded to the GPU for compositing. Assuming that all we do on future frames is move elements around (using CSS transitions or animations) or change their opacity, we simply provide these changes to the GPU and it takes care of the rest. We essentially avoid having to give the GPU any new graphics; rather, we just ask it to move existing ones around. This is something that the GPU is exceptionally quick at doing, thus improving performance overall.

There is no guarantee that this hardware compositing will be available and enabled on a given platform, but if it is available the first time you use, say, a 3D transform on an element, then it will be enabled in Chrome. Many developers use the [translateZ](#) hack to do just that. The other side effect of using this hack is that the element in question will get its own layer, which may or may not be what you want. It can be very useful to effectively isolate an element so that it doesn't affect others as and when it gets repainted. It's worth remembering that the uploading of these textures from system memory to the video memory is not necessarily very quick. The more layers you have, the more textures need to be uploaded and the more layers that will need to be managed, so it's best not to overdo it¹⁵⁷.

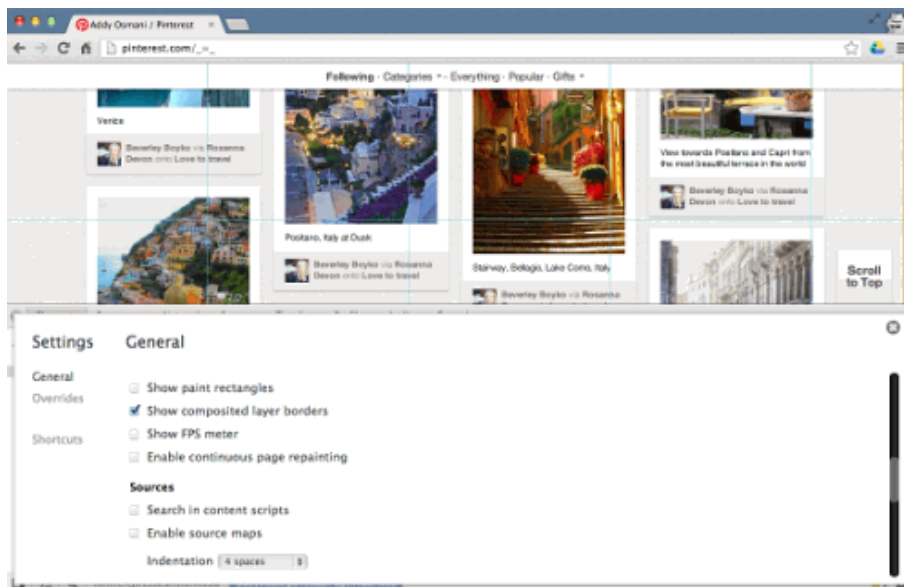
Note: Tom Wiltzius has written about the layer model in Chrome¹⁵⁸, which is a relevant read if you are interested in understanding how compositing works behind the scenes. Paul has also written a [post](#) about the [translateZ](#) hack¹⁵⁹ and how to make sure you're using it in the right ways.

Another great setting in Developer Tools that can help here is "Show composited layer borders." This feature will give you insight into those DOM elements that are being manipulated at the GPU level.

¹⁵⁷. <https://plus.google.com/115133653231679625609/posts/gv92WXBBkgU>

¹⁵⁸. <http://www.html5rocks.com/en/tutorials/speed/layers/>

¹⁵⁹. <http://aerotwist.com/blog/on-translate3d-and-layer-creation-hacks/>



Switching on composited layer borders will indicate Chrome's rendering layers. Larger view¹⁶⁰.

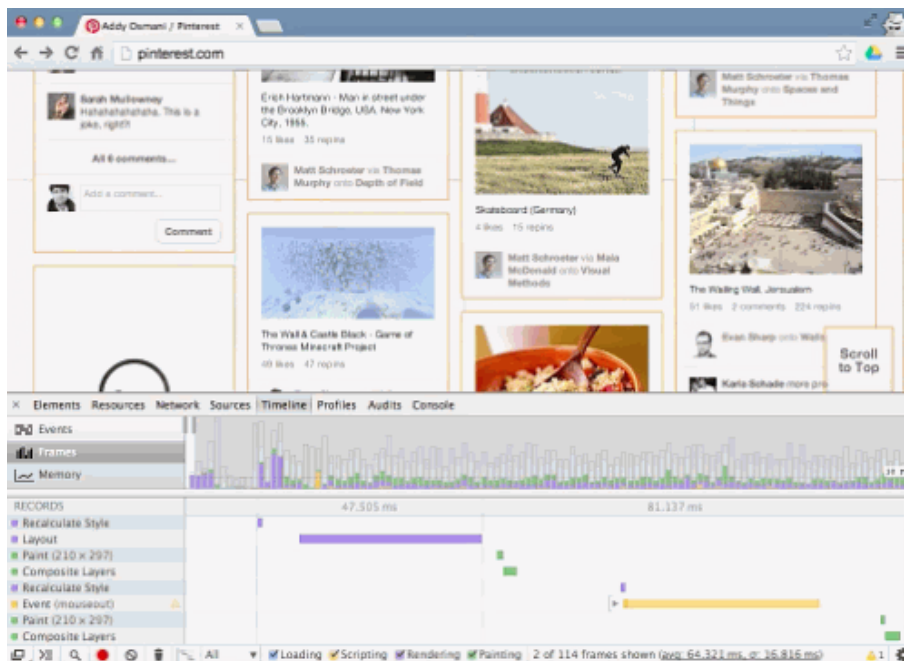
- Animated GIF¹⁶¹

If an element is taking advantage of the GPU acceleration, you'll see an orange border around it with this on. Now as we scroll through, we don't really see any use of composited layers on this page – not when we click “Scroll to top” or otherwise.

Chrome is getting better at automatically handling layer promotion in the background; but, as mentioned, developers sometimes use the `translateZ` hack to create a composited layer. Below is Pinterest's feed with `translateZ(0)` applied to all pins. It's not hitting 60 FPS, but it is getting closer to a consistent 30 FPS on desktop, which is actually not bad.

¹⁶⁰. http://media.smashingmagazine.com/wp-content/uploads/2013/05/layer_folders_addy_mini.png

¹⁶¹. <http://media.smashingmagazine.com/wp-content/uploads/2013/05/nolayers.gif>



Using the `translateZ(0)` hack on all Pinterest pins. Note the orange borders. Larger view¹⁶².

• Animated GIF¹⁶³

Remember to test on both desktop and mobile, though; their performance characteristics vary wildly. Use the timeline in both, and watch your paint time chart in Continuous Paint mode to evaluate how fast you're busting your budget.

Again, don't use this hack on every element on the page – it might pass muster on desktop, but it won't on mobile. The reason is that there is increased video memory usage and an increased layer management cost, both of which could have a negative impact on performance. Instead, use hardware compositing only to isolate elements where the paint cost is measurably high.

Note: In the WebKit nightlies¹⁶⁴, the Web Inspector now also gives you the reasons¹⁶⁵ for layers being composited. To enable this, switch off the "Use WebKit Web Inspector" option and you'll get the front end with this feature in there. Switch it on using the "Layers" button.

¹⁶². <http://media.smashingmagazine.com/wp-content/uploads/2013/05/Screenshot-2013-05-15-at-19.03.13.png>

¹⁶³. <http://media.smashingmagazine.com/wp-content/uploads/2013/05/transformpost.gif>

¹⁶⁴. <http://nightly.webkit.org/>

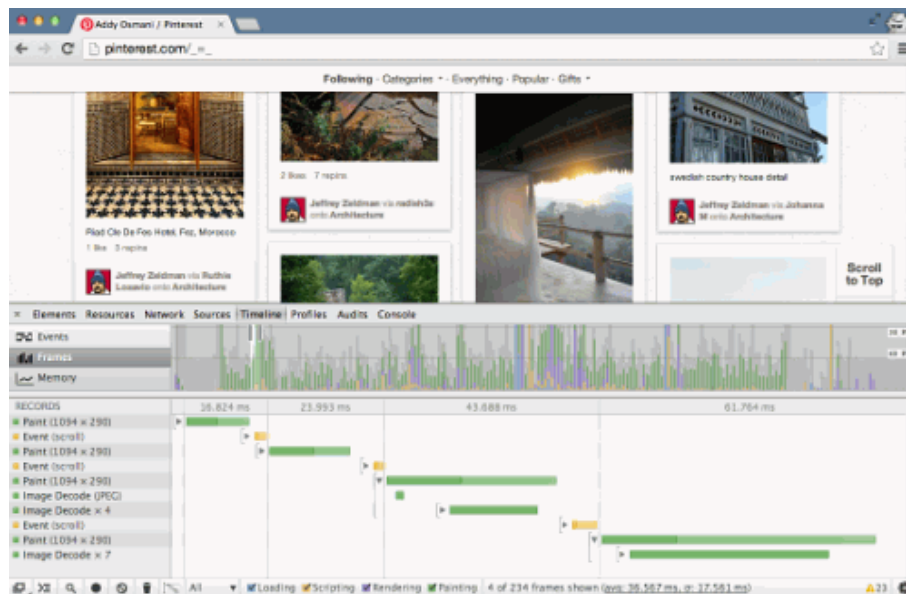
¹⁶⁵. <https://twitter.com/addyosmani/status/313978378220879872/photo/1>

A Find-and-Fix Workflow

Now that we’ve concluded our Pinterest case study, what about the workflow for diagnosing and addressing your own paint problems?

FINDING THE PROBLEM

- Make sure you’re in “Incognito” mode. Extensions and apps can skew the figures that are reported when profiling performance.
- Open the page and the Developer Tools.
- In the timeline, record and interact with your page.
- Check for frames that go over budget (i.e. over 60 FPS).
- If you’re close to budget, then you’re likely way over the budget on mobile.
- Check the cause of the jank. Long paint? CSS layout? JavaScript?



Spend some quality time with Frame mode in Chrome Developer Tools to understand your website’s runtime profile. [Larger view](#)¹⁶⁶.

FIXING THE PROBLEM

- Go to “Settings” and enable “Continuous Page Repainting.”

¹⁶⁶. <http://media.smashingmagazine.com/wp-content/uploads/2013/05/Screenshot-2013-05-15-at-19.36.22.png>

- In the “Elements” panel, hide anything non-essential using the hide (H) shortcut.
- Walk through the DOM tree, hiding elements and checking the FPS in the timeline.
- See which element(s) are causing long paints.
- Uncheck styles that could affect paint time, and track the FPS.
- Continue until you’ve located the elements and styles responsible for the slow-down.



Switch on extra Developer Tools features for more insight. [Larger view](#)¹⁶⁷.

What About Other Browsers?

Although at the time of writing, Chrome has the best tools to profile paint performance, we strongly recommend testing and measuring your pages in other browsers to get a feel for what your own users might experience (where feasible). Performance can vary massively between them, and a performance smell in one browser might not be present in another.

As we said earlier, don’t guess it, test it! Measure for yourself, understand the abstractions, know your browser’s internals. In time, we hope that the cross- browser tooling for this area improves so that developers can get an accurate picture of rendering performance, regardless of the browser being used.

Conclusion

Performance is important. Not all machines are created equal, and the fast machines that developers work on might not have the performance problems encountered on the devices of real users. Frame rate in partic-

¹⁶⁷. http://media.smashingmagazine.com/wp-content/uploads/2013/05/fixing_mini.jpg

ular can have a big impact on engagement and, consequently, on a project's success. Luckily, a lot of great tools out there can help with that.

Be sure to measure paint performance on both desktop and mobile. If all goes well, your users will end up with snappier, more silky-smooth experiences, regardless of the device they're using. 🐼

FURTHER READING

- [“Performance Profiling With the Timeline”¹⁶⁸](#), Chrome DevTools, Google Developers
- [Let's Make the Web Jank-Free¹⁶⁹](#) (resources)
- [“Don't Guess It, Test It!”¹⁷⁰](#) (article and video), Paul Lewis
- [“CSS Paint Times and Page Render Weight”¹⁷¹](#), Colt McAnlis, HTML5 Rocks
- [“Accelerated Rendering in Chrome”¹⁷²](#), Tom Wiltzius, HTML5 Rocks
- [“Avoiding Unnecessary Paints”¹⁷³](#), Paul Lewis, HTML5 Rocks
- [“Fluid User Interface With Hardware Acceleration”¹⁷⁴](#) (slidedeck) Ariya Hidayat, W3Conf 2013

¹⁶⁸. <https://developers.google.com/chrome-developer-tools/docs/using-timeline>

¹⁶⁹. <http://jankfree.org>

¹⁷⁰. <http://aerotwist.com/blog/dont-guess-it-test-it/>

¹⁷¹. <http://www.html5rocks.com/en/tutorials/speed/css-paint-times/>

¹⁷². <http://www.html5rocks.com/en/tutorials/speed/layers/>

¹⁷³. <http://www.html5rocks.com/en/tutorials/speed/unnecessary-paints/>

¹⁷⁴. <https://speakerdeck.com/ariya/fluid-user-interface-with-hardware-acceleration>

Inside Google's User Experience Lab: An Interview With Google's Marcin Wichary

BY DAN REDDING 🍷

Editor's note: Marcin Wichary worked as a Senior User Experience Designer at Google until the end of 2012.

Marcin Wichary's fascination with the relationship between humans and machines began at an early age. As a boy in Poland, he was mesmerized by the interaction between arcade patrons and the video games they played. Years later, Marcin would help shape the way that millions of computer users interact with some of the world's most popular websites. He would even recreate one of those arcade games for the Web.

Marcin is Senior User Experience Designer at Google, but his numerous roles and broad influence at the company are not conveniently definable. His fingerprints are on the code of Google products ranging from Search to Chrome. He gained publicity for his work on the Google Pac-Man Doodle¹⁷⁵, which he co-created with fellow Googler Ryan Germick¹⁷⁶. According to Ryan, "Marcin is a genius. He's a UX designer but he's also maybe one of the best front-end programmers on the planet."

Marcin joined Smashing Magazine author Dan Redding for a conversation regarding his professional career, his interest in photography and a curious creation known as the Crushinator.

¹⁷⁵. <http://www.google.com/pacman/>

¹⁷⁶. <http://www.magneticstate.com/blogdept/2009/interview-google-designer-ryan-germick/>

The Interview



Q: Hi, Marcin! I understand that you’re involved in a variety of departments and projects at Google. Can you summarize your professional roles at the company?

Marcin: Sure. So, I’ve been at Google for the past five and a half years, and I’ve always been kind of a weird mix of a user experience designer and a developer. I guess I kinda grew more comfortable with it as time went by. I started working on the Internal Tools team, working on our Internet – or the experience of our Internet – and some of the tools that Googlers use that you can’t really show to anybody outside, programs that you can’t tell your Mom, “Hey, this is what I did this week!” I’d been doing that for about two and a half years, and then I moved to kind of the proper User Experience team, the global team that works on products that actually people can use. I’ve been on Search for, well, two and a half years as well, I think.

Q: And you also work on the Doodles, for example, right?

Marcin: Yeah! So, that has always been kind of on the side. And then I just joined the Chrome team this year. So, that is kind of my second or third big change within Google. Google is great because it allows you to do a number of things, kind of as your twenty percent project, and one of mine has always been this ongoing collaboration with the Doodle team, as a... I don’t even know what to call it... a “technology consultant” collaborator! [Laughs]



Video of Google's Martha Graham doodle in action. Art by Ryan Woodward. *A fan recreated¹⁷⁷ the original doodle.*

Q: What was your involvement in the Martha Graham doodle?

Marcin: I developed a technique to animate it. The proper visuals had been done elsewhere. There's a number of technologies that you could think of if you wanted to put something on the Google home page that's animated, right? There's an animated GIF. It could maybe be a YouTube video, and we've done that. It could be a number of other techniques. But all of them have pros and cons.

For this particular doodle, none of the ones we knew of were good enough. Especially on the home page, we have a number of constraints. You know, we need to serve it to the user as fast as possible. We really care a lot about speed. And we also wanna make sure that it's gonna work on every possible platform. So, for example, that would exclude Flash animation, just because a number of touch-based devices wouldn't support that.

When I look at Web development, a lot of what's necessary is weird ingenuity, where you look at the limited number of tools at your disposal and try to put them together in a funny way. Eventually, I developed this technique to create this big sprite. Instead of just having hundreds of frames and sending them down the wire and possibly costing a lot of bandwidth, you would just construct a sprite with the smallest possible rectangle that contains the difference between two frames. It would be kind of like you had a first frame and then every other frame would be the delta. And there was this little mechanism to construct it and then play it back. It worked pretty well, so we were happy. I'm not an illustrator: I'm a designer, but I cannot draw. I'm the worst Pictionary player ever. So, I won't ever be allowed to draw anything on the home page, for the good of humanity.

As a designer, in many cases, I'm invisible. I can help with technology, but the technology is ultimately invisible. Many people really did

¹⁷⁷. <http://www.acumenholdings.com/blog/how-to-animate-the-martha-graham-google-logo-for-yourself-tutorial-2/>

enjoy the Martha Graham doodle, and I've got the quiet satisfaction that I helped make it possible.



Marcin says, "This is from the San Diego Zoo, 2009. I don't photograph living creatures often, but when I do, it's usually a lot of fun." (Image: Marcin Wichary)

Q: What is the Crushinator?

Marcin: [Laughs] Okay, so that's actually the [name of the animation] technique that I just described. I'm trying to convince the Doodle team to set up a proper blog where they can talk about stuff like this, both as artists and technologists. The great thing about the Web is that you can always right-click, "View source" or "Inspect element" and try to figure it out. And, as a matter of fact, I do recall there being a [blog post](#)¹⁷⁸ somewhere on the day that the Martha Graham doodle came out, where somebody reverse-engineered the whole thing to figure out how it works. They didn't know that the technique was called Crushinator because that's my stupid *Futurama* reference.

Q: I think that the Crushinator, as you call it, and this style of JavaScript-powered sprite animation are actually a very innovative animation style that is lightweight and available cross-browser, as you've said. Do you see the potential for that technique to become a widely adopted tool for animation on the Web?

Marcin: I don't think about it that way, because it's kind of what I do. I've been doing Web development for many, many years. A lot of Web development is, as I mentioned, putting [technologies] together

¹⁷⁸. <http://www.acumenholdings.com/blog/how-to-animate-the-martha-graham-google-logo-for-yourself-tutorial-2/>

and seeing what happens. After a while, I developed an attitude that pretty much nothing's impossible. You know that with HTML5 around and all of these cool things happening in the browser, something's probably gonna help you. So, I do a lot of putting things together, and they actually don't seem like that big a deal to me because that's what I do. And sometimes you just throw [your experiment] at random people inside Google, and some of them become more popular, and others nobody cares about. So, I don't think I'm the right person to talk about whether anything that I hack together has more utility than anything else. A lot of what I'm hoping to do with this kind of random hackery is figuring out, how can we push HTML5 further this week? You know, just hoping to inspire some people.



"This is a photo a friend and I took of me against a Cylon replica. This is a dramatic recreation of my typical day." (Image: Marcin Wichary)

Q: Well, I think you should give yourself some credit. Some of the work you guys are doing is very innovative. And it's also very visible; it gets a lot of attention. There's potential there for you to be a real influencer of these kinds of technologies.

Marcin: I'm kind of recognizing that, for better or worse, people sometimes come to me and treat me as if I know something about what

I'm doing. [Laughter] And maybe I do. Sometimes I feel like it's my first day with a new video game, and I'm just randomly pressing buttons and things happen on the screen and I don't know exactly why, but it's cool, right? Because HTML5 is cool.

Q: What is the most useful usability or user experience finding you have learned recently?

Marcin: There are a number of those, but unfortunately, I can't really talk about them because they are internal.

But... this is going to sound funny. A while back, there was this report that came out — from Princeton University¹⁷⁹, I think — that said using Comic Sans will make your material easier to remember. The harder your typeface is to read, the better the chance that people will remember what you wrote. And everybody I know perversely sends me links to this study, even literally last week, because they know how much I hate Comic Sans... [Laughter] ... with passion. And people are sending this study to me, saying "You were wrong, and we were right." You know, tongue in cheek.

But of course, that really isn't what the study is about. First of all, you don't have to use Comic Sans. Second of all, it's just one factor, the same way there's typically always a battle between usability and security, right? In order to make things more secure, sometimes it means you have to make them less usable. Like the simple fact that whenever you type a password, you can't see it. It's asterisks or dots or whatever, because it makes it harder for other people to eavesdrop on you. But it also makes it easier for you to mistype. So there's a ton of these kind of choices that you have to make. You have to find a nice medium between security and usability. Those decisions are really, really hard and require a lot of thought. So, the same thing with Comic Sans and that study. You could use another font. Or use Comic Sans — it will make the learning easier. That's cool. But if your students develop an intense hatred for typography in general or design or, I don't know, the universe, [Laughter] I don't think you've won much. I'm kind of tongue in cheek now, too. But not really.

Q: When did your interest in code and computer programming begin?

Marcin: It actually all kind of ties together in a number of funny ways. When I was very little, my dad had the best job ever... for a nerd like me. His job was fixing arcade games and pinball machines. So, I

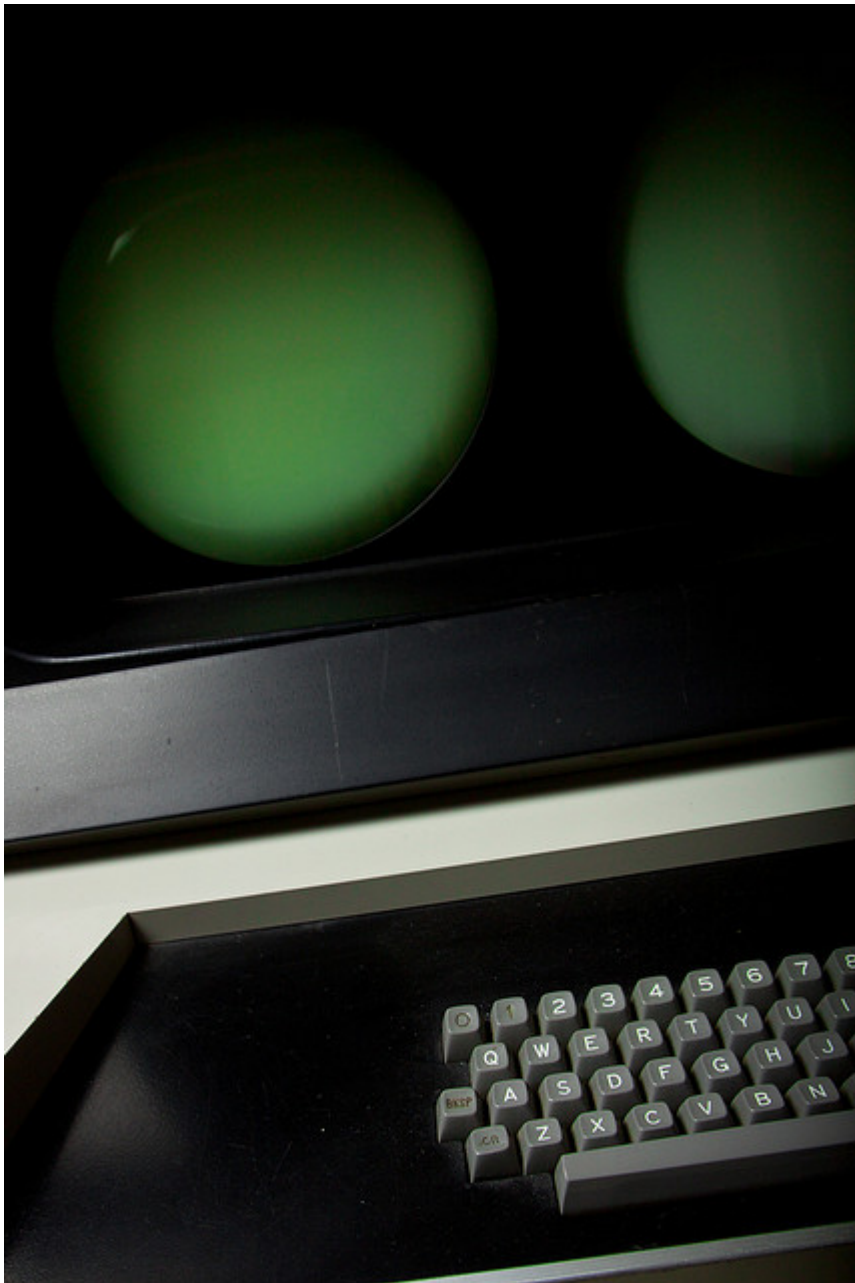
¹⁷⁹. <http://www.princeton.edu/main/news/archive/S28/82/93O80/index.xml?section=research>

could follow him to all of these little arcade parlours in Poland, where I'm from. I could play for free, first of all, which was awesome. But actually, I much preferred watching other people play games. Of course, I would play games – I loved games and still do. But I could also get a little sneak peek – if you opened the arcade game, what happens there? If you went into debug mode or testing mode, which many of the arcade games had, there were a number of tools, if you had access to them, to see the graphic elements in the video game, the sound effects and all of the settings. And I was fascinated! It was like looking under the hood of a car – except, you know, a much cooler car. Maybe that's what got me into computers.

As I was finishing my computer science degree in Poland, working on my Masters, I discovered this whole area of human-computer interaction – which people work on for a living, and it's important, and it matters, and you can make a career out of it. That was a revelation to me. Suddenly, the past God-knows-how-many years that I was doing [amateur computer programming] were cast in a very different light.

Q: Do you think that when you were in that arcade in Poland, when you were a kid, and you were watching a patron play Space Invaders, that you were monitoring the user experience by watching that person play?

Marcin: Maybe! It's an interesting thing. I used to be fascinated by books by the Polish author Stanisław Lem. Probably my favorite book of his is called *Tales of Pirx the Pilot*. I was fascinated by it because it's about space and aliens and the future – it was awesome for a kid. But reading this book today, the whole theme is Pirx interacting with machines. The machines are robots, or spaceship A.I., but it really is about human-computer interaction. And the question now is whether I was already interested in that when I was very little. Or maybe the books actually shaped me. Or maybe it was both. And the same maybe with the arcades. Maybe I was watching people play and kind of thinking about what it meant. Or maybe I was so inspired by how much fun you could have with technology and how much delight that I started trying to do it myself in whatever way I could – and today it happens to be HTML5. And who knows what's gonna happen ten years from now.



*Marcin says that photos like this one are an attempt to “anthropomorphize machines.”
(Image: Marcin Wichary)*

Q: What do you think the Internet might look like at that time?

Marcin: That’s always pretty tricky! I actually make a point of not trying to predict the future because I’m really terrible at that. But I can tell you one thing I’m excited about. A number of things are happening right now that get me really excited about the idea of computer-user interfaces that for the first time don’t feel like they’re coming out of a computer. One idea is that the retina display and other technologies that will probably arrive make it very easy to put something on the screen that mimics the imperfections of the real world. Maybe the typography doesn’t have to be so perfect, because it’s never so perfect in books. Maybe there’s gonna be something with interfaces being actual-

ly broken in some way — broken to mimic real life, not broken because we're bad at developing things.

I'm especially excited about the implication of this in relation to typography. Over the course of hundreds of years, we built up [a typographic practice] with total nuance, with a lot of history, and we arrived at this very rich era where typography is amazing, right? It can do so much. There's so much history there and so much beauty. And then the computer came around in the nineteen eighties and took all that away! [Laughter] You got monospace fonts and "underline"... Now we've got Unicode, and HTML5/CSS3 allow you to use different typefaces, and maybe, slowly, we're gonna get all of this back again — and maybe even more. Maybe we'll get all of the things we love in books and all of the things we love in computers combined.

Q: Can you give me an example of a day in the life of Google's UX design team?

Marcin: There's really no template for a day. Probably the most important part of my day, and the one I treasure most, is just talking to other designers. Either ranting like we do about everything that has to do with design, [Laughter] or showing them something I'm working on and getting feedback. We have so many amazing kinds of designers here.

For me, a lot of the day is spent staring at a computer. In my case, it's a lot of HTML5, so some kind of text editor is in front of me — or, in some cases, Photoshop. I've spent so much time in HTML5 that it's actually easy for me to just treat it as a design tool.

Q: What else can you tell me about your set-up and workflow at Google?

Marcin: I've always had a kind of low-level, old-school approach. I coded one site in FrontPage many, many years ago. [Laughter] But ever since then, I've pretty much coded everything by hand, from scratch, in a very simple text editor. These days it happens to be TextMate, although I don't actually use any of the advanced TextMate features. I feel that gives me more control.

I actually make a point of starting things from scratch, even if they have been written elsewhere, or even by me before, because it allows me — you know, in a fast-moving environment where HTML5 and CSS3 change a lot — it allows me to learn all of those new things, to try a different approach, to think about how I did it the last time and try to do it differently, just for the sake of doing it differently. And that always feels really great. That approach probably only works for, like, little projects, little goofy experiments that I do. I don't know... you can't really

rewrite Gmail every single month from scratch. [Laughter] But it's kind of been working for me.

Other than that, it's really very simple. Browser in one window, TextMate in another window, and a lot of Command-Tab'ing and a lot of Command-R'ing. You know, simple old habits. Muscle memory.

I have a couple of little tools that I made for myself and others at Google that I should open source, and I feel really bad for not having done that. One of them has another funny name. It's called Transmogri-fier, which is a nod to *Calvin and Hobbes*, which I love a lot. It's a little tool that allows me to very easily test different types, or different parameters of my prototypes. You know, JavaScript is not very good at CGI parameters, so Transmogri-fier gives me this nice UI, and I don't have to worry about the innards and switching and hiding the options panel. I just plug in one line of JavaScript, and the tool takes care of most of that for me.

A number of other tools are floating around at Google that are pretty great, and I helped with some of those. And writing tools is actually a cool experience, too. You can learn a lot. Like, for example, writing a tool that you can embed in another page and it doesn't ever break that page. This should be super-easy, but it's not, with cascading styles just waiting for you to trip up. So, you can learn a lot of different skills.

Q: Google has a reputation for reliance on data and intensive testing of products and design. How does this affect the user's experience of Google websites?

Marcin: We've historically been very focused on the efficiency of user experiences. For a lot of the products we have, including Google Search, our main consideration was the efficiency of getting to the proper response as fast as possible, and the answer being as precise as possible. We've gotten a lot of different comments over the years, with people suggesting that we don't design our interfaces, that they kind of just *are* — they just exist. In most cases, the most deliberate design decision on our part is for our interfaces not to feel like they're designed. They will seem more objective. Less editorialized. Relying on data is a very important part of this.

We've been looking at this in more detail recently. We've been thinking that, maybe for some of our projects that we're working on currently, we should approach design a little bit differently. It's going to be very interesting to see what happens.

Q: How did your interest in photography come about?

Marcin: I've always dabbled in photography. Back when I was in Poland, we had one of the first digital cameras — you know, a very primitive one with half a megapixel or whatever. But I never really consid-

ered myself a photographer. Really, it was kind of an impulse buy in late 2007, as I was heading out on a trip to New York. I started chatting with a friend of mine who is a photographer, and I ended up overnighting my first DSLR with my first lens, and I kind of fell in love with it.



"I am sure no one played [this piano] seriously for a long, long time, so I did. I just sat down and started playing — the sonatas I've learned, the things I've put together myself. I had to be a bit creative, since some of the keys no longer worked." (Image: Marcin Wichary)

I play piano occasionally, and I'm terrible at it. I think I'm good at being terrible at it, trying to figure it out on my own. When playing piano, the most interesting part to me is emotion. Can I learn to play things that mean a lot to me? Or can I compose something in which I evoke a certain emotion in other people — or even in myself? It's less to do with technical proficiency.

In photography, what I realized is that the technology is the driver for me. A big part of me is fascinated with the technology of photography. I know that's not true for other people. Maybe other people are interested in emotion or some other things.

Photography has been around for much, much longer [than computers], and it's great to realize that there's so much to photography that you could pretty much learn something new forever. Not only is there already so much that's happened, but it's also still happening right now. We're at this really, really amazing time when digital photography is becoming commonplace — it's already commonplace. What that means is that first of all, we have photographers everywhere, which is great. It's gonna be exciting to see what happens in the next couple years, or ten years, or twenty years. And that's another thing that gets me excited

about photography in the same way I'm excited about HTML5 — because the best is yet to come. ☛

More Information

- [Marcin's website](#)¹⁸⁰
- [Marcin and Ryan's Google Pac-Man I/O presentation](#)¹⁸¹
- [Ryan Germick explains the Crushinator](#)¹⁸²
- [Marcin's Pac-Man bug story](#)¹⁸³
- [HTML5 Presentation created by Marcin and Google Chrome team](#)¹⁸⁴
- [Marcin talks Pac-Man, HTML5 and more](#)¹⁸⁵

¹⁸⁰. <http://www.aresluna.org/>

¹⁸¹. <http://www.youtube.com/watch?v=ttavBa4giPc>

¹⁸². <http://www.magneticstate.com/blogdept/2011/crushinator>

¹⁸³. <http://www.aresluna.org/stories/blocked/>

¹⁸⁴. <http://slides.html5rocks.com/>

¹⁸⁵. <http://ajaxian.com/archives/web-ninja-interview-marcin-wichary-creator-of-google-pacman-logo-html5-slide-deck-and-more>

Mistakes I've Made (And Lessons Learned Along The Way)

JEREMY GIRARD 🍷

We all make mistakes. Whether in our design and development work or just in life in general, we all do it. Thankfully, even the biggest mistakes carry valuable lessons.

As a contrast to the many Web design articles that focus on successes and what we can learn from those triumphs, this article looks to the other end of the spectrum to explore what failures teach us.

Along the way, I will share stories of some of the missteps I have made in the course of my career and the lessons I've learned in the process — being ever mindful of composer John Powell's words:

The only real mistake is the one from which we learn nothing.

Mistake #1: Putting Process Over Projects (And People)

Anyone who has been designing and developing websites for any amount of time has come up with a process for working. Having a process is good, but be careful that it does not overshadow the project itself or the people involved.

I was reminded of this a few years ago in a project that was going badly. The simple reality was that I was not getting along with the project manager who was appointed by the client. Our personalities clashed almost from the start, as I found her feedback and requests to be misguided and her personality abrasive. At the same time, I am sure she found me unhelpful and combative because I was unwilling to honor all of her requests.

As frustration grew, I tried to fall back on our process as a way of adding structure to the relationship and trying to get it back on track. If she made a request that took us outside of our normal process, I explained how we could not do it without setting the project back in both time and budget. The worse the project got, the more I deferred to our process, until the client, exasperated to the limit, told me that I seemed to care more about our process than the project.

My plan had backfired. I had tried to lean on our process in order to fix the problems, instead of having a difficult confrontation and dealing with the real issue – the fact that personality clashes were becoming strained to the point that nothing was being accomplished.

Eventually, we reset the project by calling for a meeting to clear the air and address the problems honestly so that we could move forward. While I continued as the project lead on our side, I brought in another team member, someone who did not have a rocky history with the client's project manager, to handle the day-to-day communications. Even though she acted as little more than an interpreter for me in many cases, the fresh voice and personality from our side did wonders for the relationship, and the project manager responded to our new team member much better than she had to me.

Additionally, we looked at the client's requests a little more deeply and, rather than dismissing them outright because they deviated from our normal process, tried to identify the reasoning behind each request so that we could honor them in the spirit in which they were made (which we normally do anyway). We realized that those requests didn't really affect our normal process in a big way. Any deviation was minor, and the relationship and the project were much better off with the flexibility in our process.



People > Project > Process

People are more important than the project, which is more important than the process.

Of course, you need to strike a balance. A process exists for a reason, and if you abandon it whenever anyone shows resistance, then there is little point in having a process at all. That being said, any good process has some flexibility to accommodate the different needs of clients and projects.

Lesson learned: Followed blindly, no process will save you from having to deal with difficult personalities or bumps in the road. A process is

meant to help a project along, not to be hidden behind when the going gets tough. For additional reading on client communications, see my previous articles, “[Keys to Better Communication With Clients](#)¹⁸⁶” and “[How to Deliver Exceptional Client Service](#)¹⁸⁷.”

Mistake #2: Telling Instead Of Showing

I frequently speak with clients about their website needs. I listen to their concerns and the issues they’re having with their current website, and I tell them how we can meet their needs. Note that I said I “tell” them how we can help, when I should usually be *showing* what we can do for them.

This might not seem like a big difference, but it could mean the difference between winning a new project or losing it to someone else — which is exactly what happened to me recently.



Certain things are more effective when shown instead of told. (Image: [Joe Penniston](#)¹⁸⁸)

A few weeks ago, I was informed by a prospective client that they had decided to work with another provider. Whenever this happens, I am gracious and thank the client for considering us in the first place. I also ask them what the deciding factor was. In this case, they loved our proposal and solutions, but another company had given a detailed demonstration of their preferred CMS and showed how they would use it to keep the website up to date. That company *showed* them instead of *told* them.

¹⁸⁶. <http://www.smashingmagazine.com/2012/07/16/keys-better-client-communication/>

¹⁸⁷. <http://www.smashingmagazine.com/2012/01/25/how-to-deliver-exceptional-client-service/>

¹⁸⁸. <http://www.flickr.com/photos/23322134@N02/4665323296/>

I'd be lying if I said I didn't kick myself upon hearing this feedback. I would have been happy to give this client a CMS demonstration, but they didn't ask, so I didn't offer. Instead, I answered their questions — all the while thinking I was giving them what they wanted.

The other provider's CMS is not necessarily easier to use than the one I was offering, but I never even made that case because I *told* the client how easy our solution was to use, instead of *showing* them.

Lesson learned: Talk is cheap. Regardless of whether the client specifically asks for a demonstration in your proposal, showing them goes a long way by backing up your words.

Mistake #3: Not Informing Clients Of Staffing Changes

Staffing changes are a reality in this industry. Team members move on to other positions and opportunities, but business must go on. Projects need to be finished, and websites and clients need to be supported. As one team member departs and another joins, you will establish a plan for existing projects and clients, assigning responsibilities and tasks as needed. Still, however solid and measured your plans may be, don't neglect to inform your clients of these staffing changes.

I learned this lesson when a longtime colleague recently left for another position. We had a plan in place for the transition, a plan that involved him working with us part time to continue handling certain clients and services. The impact on our clients would be minimal, and I decided that we didn't need to inform them of the changes because the services we provided would not suffer and the change in our staff would likely go unfelt. I was wrong.

It didn't take long for one of our clients to reach out to my departing colleague. My colleague's work emails were now being forwarded to me, so I received the client's request. We made the changes requested, and when I emailed the client to notify them that the work was done, I also explained the change in staffing to account for why the response was coming from me. As you can probably guess, they were surprised by this news, and what should have been a non-issue suddenly became an issue, simply because the client hadn't learned of this sooner and was taken by surprise.

While some staffing changes are certainly not appropriate to discuss with clients, others really do affect clients in a pretty big way. A person may be the client of a company as a whole, but if their day-to-day interaction is with a particular team member, then that team member "becomes" the company in their eyes. If that team member ever de-

cides to leave, the client could feel as though they are switching providers, even though the company is still more or less the same.

So, be proactive in informing clients of staffing changes. By explaining your plan for the transition of responsibilities with their account and reassuring them of your continued support of their company, you show them that, despite the change in staffing, you are still thinking about them and their needs.

Mistake #4: Focusing On Money At A Time Of Transition

Speaking of transitions, another reality in this industry is that clients sometimes decide to move onto another provider. When this happens, there is a period of transition away from your services, and you will likely need to be involved in that transition. This can be a strange and uncomfortable time, in part because you're concerned about money.

Ongoing clients have an incentive to pay their invoices because they want to continue working with you. Clients who switch providers are worrying because of the possibility that they won't honor any outstanding invoices – including time spent helping them transition away from your services.

This situation is delicate and needs to be handled case by case. Their reason for leaving, their overall payment history, how much they currently owe you (if anything), and how involved you will need to be during the transition are all factors that will determine how you handle the situation. The big lesson I have learned, however, is that dwelling on exactly when you will get paid during this time of transition, which is often a time of uncertainty and even fear for the client, is rarely wise.

When I've focused on payment and gotten aggressive in making sure the client understands their financial obligation to us, those clients have actually turned out to be *less likely* to settle their accounts in good time than clients whom I approach more softly.

Providing outstanding service to a client during a time of transition is the best way to end a relationship. If the relationship ends on a positive note, then the client will be more likely to pay what they owe and to say nice things about your company, because the last impression you've left them with was helpful and positive.

Again, how you handle such situations will vary. If the breakup is messy, or you are owed a substantial amount of money or lawyers have to get involved, then you would handle that transition differently than if you had a good client who was leaving simply because you were no longer a good fit.

For more tips on handling client payment issues, see “[Dealing With Clients Who Refuse to Pay](#)¹⁸⁹.”

Mistake #5: Looking To The Past, Instead Of The Future

When we make decisions on a project, we often look for relevant data to justify our decisions. Referring to website analytics and usage data can help us make informed decisions, but remember that all of this data refers to the past, not the future.

The Web industry is constantly moving forward, and if we make our decisions based solely on data gleaned from past usage, then the solutions we develop will be perfectly suited to those past situations, not necessarily future ones. This happened to me about a year ago when we were working with a client to come up with a mobile strategy for their website. While we absolutely wanted to make the website responsive, the scope of the project and the budget simply did not allow it. Plans were made and a budget allocated to redesign the website the following year, and a fully responsive design would certainly be part of that project, but for now, a separate mobile-only website would be our short-term solution.

As with many mobile websites, our plan was to include only a small, targeted subset of the enormous content archive found on the current website. Looking back now, it was a mistake. Unfortunately, [content parity](#)¹⁹⁰ wasn’t an option, so to determine what content to include, we looked to the analytics to see which pages mobile users were accessing. Office locations and directions, leadership team biographies, and contact details were the most popular pages being requested by mobile users, so that was what we included on the mobile website. There was, however, a problem with this logic of including only currently popular content on the mobile website: It did not account for future needs.

As we were working on the mobile website, the client began to focus on their blog. They formed a team of authors among the subject matter experts in their organization and began publishing a lot of quality content – content that quickly became popular with their audience. This new blog content was often promoted and shared via social media, and many visitors accessed those links via mobile devices.

You can probably see where this is heading. Because we had no data to show that the blog would be popular on mobile devices, we left the

¹⁸⁹. <http://www.smashingmagazine.com/2010/04/09/dealing-with-clients-who-refuse-to-pay/>

¹⁹⁰. <http://bradfrostweb.com/blog/mobile/content-parity/>

blog off of the mobile website. When the blog picked up steam and attracted interest from users on social networks and mobile devices, the website we had developed became a major problem. The experience would be as follows:

1. A mobile user would see a comment about or link to an article in social media and, being curious about the article, click the link.
2. The mobile device would navigate to the blog article on the full website, but then quickly redirect to the mobile website's home page.
3. Because the blog was not accessible from the mobile website's menu, the visitor had to tap the "View full website" link and, on their small phone, try to find the blog on the full website. If that's not frustrating, what is?



Making new mistakes helps you learn new lessons. (Image: [Elyce Feliz](#)¹⁹¹)

Obviously, this experience was exceedingly poor, and very few visitors went through the entire process just to read the article. Most just left

¹⁹¹. <http://www.flickr.com/photos/99175982@N00/4448688868/>

when presented with the mobile home page, instead of the article they were hoping to see. Even though we knew from the start that this mobile-only website was temporary, had we more effectively planned ahead and not based our decisions solely on analytics from the past, we may have been able to avoid this problem and develop a better solution.

In this case, the answer was to kick off the responsive redesign project sooner and do away with this separate mobile-only website and its subset of content. The lesson we learned is that we have to look to both the past and the future when making decisions on a project.

This is why clients hire us in the first place – not only for our execution, but for our expertise. This expertise includes knowing where the industry is headed, what principles have to become an integral part of the experience (content parity) and what new technologies or approaches we can bring to a website today to ensure that it works well tomorrow.

The Value Of Mistakes

All of the blunders covered in this chapter are ones I've made that either took a project off track or strained a relationship or made a product far less successful than it could have been. As soon as I realized each mistake, I wished I could jump back in time and have a do-over.

Well, I've yet to find that elusive time machine, but I do get do-overs of sorts. Every time I encounter a similar situation, I am able to make a better decision as a result of having learned the lesson from the previous mistake. That is my do-over, and that is the value of learning from one's mistakes.



We all make mistakes. (Image: opensource.com¹⁹²)

WHAT ABOUT YOU?

What mistakes have you made, and what lessons have you learned from them? Not many folks like to talk about their mistakes, disappointments and things that just didn't work out, but quite often they're just as useful as all those amazing success stories you can read about in hundreds of books and articles. What tools worked for you and which didn't, and why? Please share your stories and your thoughts with us by using the hashtag [#smworkflow](#)¹⁹³! 🐼

¹⁹². <http://www.flickr.com/photos/47691521@N07/5496629643/>

¹⁹³. <https://twitter.com/search?q=%23smworkflow&src=typd&mode=realtime>

About The Authors

Addy Osmani & Paul Lewis

Addy Osmani¹⁹⁴ and Paul Lewis¹⁹⁵ are engineers on the Developer Relations team at Chrome, with a focus on tooling and rendering performance, respectively. When they're not causing trouble, they have a passion for helping developers build snappy, fluid experiences on the Web.

Dan Redding

Dan Redding designs brand identity and websites at his studio, Magnetic State¹⁹⁶. He also hosts a podcast called The Magnetic Eye¹⁹⁷, featuring interviews with artists and designers. You can follow Dan on Twitter at @danredding¹⁹⁸.

Jeremy Girard

Jeremy was born with six toes on each foot. The extra toes were removed before he was a year old, robbing him of any super-powers and ending his crime-fighting career before it even began. Unable to battle the forces of evil, he instead works as the Director of Marketing and Head of Web Design/Development for the Providence, Rhode Island based Envision Technology Advisors. He also teaches website design and front-end development at the University of Rhode Island. His portfolio and blog, at Pumpkin-King.com¹⁹⁹, is where he writes about all things Web design. Twitter: @jeremymgirard²⁰⁰.

Jeremy Olson

Jeremy Olson (@jerols²⁰¹) is founder and lead designer at Tapity²⁰², an Apple Design Award winner, and author of *The App Design Handbook*²⁰³.

¹⁹⁴. <http://twitter.com/addyosmani>

¹⁹⁵. <http://twitter.com/aerotwist>

¹⁹⁶. <http://www.magneticstate.com/>

¹⁹⁷. <http://www.themagneticseye.com/>

¹⁹⁸. <http://www.twitter.com/danredding>

¹⁹⁹. <http://www.pumpkin-king.com/>

²⁰⁰. <http://www.twitter.com/jeremymgirard>

²⁰¹. <http://www.twitter.com/jerols>

²⁰². <http://www.tapity.com>

²⁰³. <http://nathanbarry.com/app-design-handbook/m>

Richard Shepherd

Richard ([@richardshepherd](https://twitter.com/richardshepherd)²⁰⁴) is a UK based Web designer and front-end developer. He loves to play with HTML5, CSS3, jQuery and WordPress, and currently works full-time bringing [VoucherCodes.co.uk](http://www.vouchercodes.co.uk)²⁰⁵ to life. He has an awesomeness factor of 8, and you can also find him at richardshepherd.com²⁰⁶.

Wilson Page

Wilson is a front-end developer at FT Labs²⁰⁷ in London. He enjoys pushing the limits of responsive design, and dreams in JavaScript. You can [follow him on Twitter](https://twitter.com/wilsonpage)²⁰⁸ for frequent front-end goodness.

²⁰⁴. <https://twitter.com/richardshepherd>

²⁰⁵. <http://www.vouchercodes.co.uk/>

²⁰⁶. <http://richardshepherd.com/>

²⁰⁷. <http://labs.ft.com/>

²⁰⁸. <https://twitter.com/wilsonpage>

About Smashing Magazine

Smashing Magazine²⁰⁹ is an online magazine dedicated to Web designers and developers worldwide. Its rigorous quality control and thorough editorial work has gathered a devoted community exceeding half a million subscribers, followers and fans. Each and every published article is carefully prepared, edited, reviewed and curated according to the high quality standards set in Smashing Magazine's own publishing policy²¹⁰.

Smashing Magazine publishes articles on a daily basis with topics ranging from business, visual design, typography, front-end as well as back-end development, all the way to usability and user experience design. The magazine is — and always has been — a professional and independent online publication neither controlled nor influenced by any third parties, delivering content in the best interest of its readers. These guidelines are continually revised and updated to assure that the quality of the published content is never compromised. Since its emergence back in 2006 Smashing Magazine has proven to be a trustworthy online source.

²⁰⁹. <http://www.smashingmagazine.com>

²¹⁰. <http://www.smashingmagazine.com/publishing-policy/>