CSS | Sass | JavaScript | RWD | WordPress

BRAND NEW CONTENT

THE ULTIMATE GUIDE TO WEB DESIGN

228 PAGES OF EXPERT ADVICE FOR DESIGNERS AND DEVELOPERS

VOLUME III

Design and build amazing sites with the latest tools, tips and techniques

Future

From the makers of
net

Future PLC, The Ambury, Bath BA1 1UA
Telephone +44 (0)1225 442244 **Website** net.creativebloq.com

Editorial
Editor Oliver Lindberg, oliver.lindberg@futurenet.com
Art editor Mike Brennan, mike.brennan@futurenet.com
Production editor Ruth Hamilton, ruth.hamilton@futurenet.com

Contributors
Rachel Andrew, Jackie Balzer, Liam Brummitt, Joy Burke, Antoine Butler, Razvan Caliman, Matt Campbell, Ian Carrico, Joe Casabona, Mike Chipperfield, Mark Dalgleish, Alex Danilo, Kevin Foley, Julian Garnier, Tom Giannattasio, Craig Grannell, Tobias Hall, Mat Hayward, Marko Heijnen, Tim Hettler, Benjamin Howarth, Samuel Hulick, Scott Kellum, Katie Kovalcin, Phil Leggetter, Ari Lerner, Joe Maddalone, Dan Mall, Eric Mann, Alex Matchneer, MaricorMaricar, Stuart Memo, Eric Morris, Dave Myburgh, Dan Neame, James OConnell, Den Odell, Luke O'Neill, Neil Renicker, Dan Rose, Lukas Ruebbelke, Tim Ruffles, Adam Simpson, Andy Smith, Dan Tocchini, Roy Tomeij, Steven Wu

Advertising
Advertising sales director James Ranson, +44 (0)20 7042 4163, james.ranson@futurenet.com
Advertising sales director Suzanne Smith, +44 (0)20 7042 4122, suzanne.smith@futurenet.com
Senior sales executive Victoria Sanders, +44 (0)20 7042 4176, victoria.sanders@futurenet.com

Marketing
Group marketing manager Philippa Newman, philippa.newman@futurenet.com

Print & Production
Production controller Keely Miller, keely.miller@futurenet.com

Licensing
Senior licensing & syndication manager Regina Erak, + 44 (0)1225 732359, regina.erak@futurenet.com

Management
Editor-in-chief Dan Oliver, dan.oliver@futurenet.com
Head of content and marketing, photography, creative and design Matthew Pierce, matthew.pierce@futurenet.com,
Director of content and marketing Nial Ferguson, nial.ferguson@futurenet.com
Creative director Robin Abbott, bob.abbott@futurenet.com
Acting group art director Simon Middleweek, simon.middleweek@futurenet.com
Editorial director Jim Douglas, jim.douglas@futurenet.com,
Chief executive Zillah Byng-Maddick, zillah.byngmaddick@futurenet.com

Subscriptions
Phone our UK hotline 0844 848 2852, **international** (+44) (0) 1604 251045
Subscribe to net magazine online at www.myfavouritemagazines.co.uk

Printed in the UK by William Gibbons.
Distributed in the UK by Seymour Distribution Ltd, 2 East Poultry Avenue, London EC1A 9PT. Tel: 0207 429 4000

Future

Future is an award-winning international media group and leading digital business. We reach more than 58 million international consumers a month and create world-class content and advertising solutions for passionate consumers online, on tablet & smartphone and in print.

Future plc is a public company quoted on the London Stock Exchange (symbol: FUTR).
www.futureplc.com

Chief executive Zillah Byng-Maddick
Non-executive chairman Peter Allen
Chief financial officer (Interim) Simon Poulton

Tel +44 (0)207 042 4000 (London)
Tel +44 (0)1225 442 244 (Bath)

FSC www.fsc.org MIX Paper from responsible sources FSC® C007184

The text paper in this magazine is totally chlorine free. The paper manufacturer and Future Publishing have been independently certified in accordance with the rules of the Forest Stewardship Council.

recycle
When you have finished with this magazine please recycle it.

INTRODUCTION

THE ULTIMATE GUIDE TO WEB DESIGN

Create incredible sites and discover cutting edge tips, tools and techniques to take your web design skills to the next level

In this all-new edition of *The Ultimate Guide to Web Design*, we bring you the essential design and development knowledge you need to create amazing websites and set yourself apart from the competition.

Comprised of the best practical articles from **net** magazine – the voice of web design – this special edition has been written by some of the biggest names in the industry and includes a wealth of tutorials, tips and techniques.

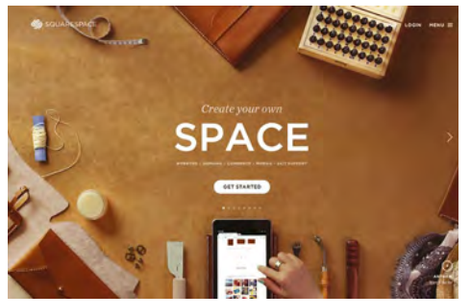
Whatever you want to do on the web, we've got something here for you. Section 1 covers web essentials to help you find the right tools, launch a product, create a killer first impression and tackle a redesign project. Section 2 dives deep into CSS and Sass, and Section 3 gets you started with responsive web design. Section 4 will see you hone your JavaScript skills, with tutorials on AngularJS, D3.js, real time and more. Finally, Section 5 covers WordPress and other excellent CMSs such as Squarespace, Ghost and Drupal.

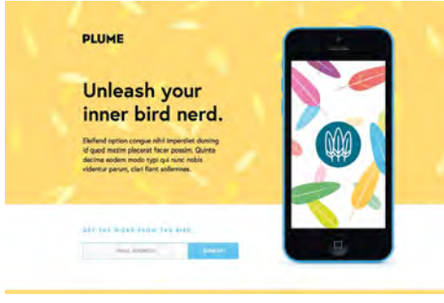
Dive in and start sharpening your web design skills now. Enjoy!

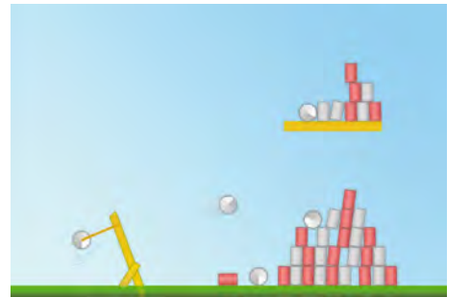
Oliver Lindberg, editor
oliver.lindberg@futurenet.com
@oliverlindberg

CONTENTS

The Ultimate Guide to Web Design: Vol III







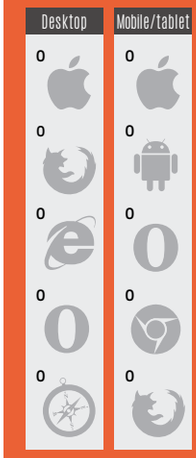
WEB ESSENTIALS

CSS & SASS

RESPONSIVE DESIGN

JAVASCRIPT

WORDPRESS & CMSs



BROWSER SUPPORT

We feel it's important to inform our readers which browsers the technologies covered in our tutorials work with. Our browser support info is inspired by @andismith's excellent When Can I Use web widget (*andismith.github.io/caniuse-widget*). It explains from which version of each browser the features discussed are supported.

net

MAGAZINE IPAD EDITION

The iPad edition of **net** has been completely rebuilt from the ground up as a tablet-optimised reading experience.

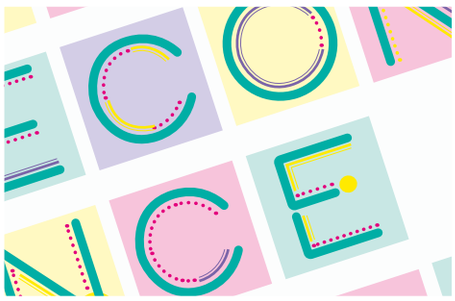
TRY IT FOR FREE TODAY!

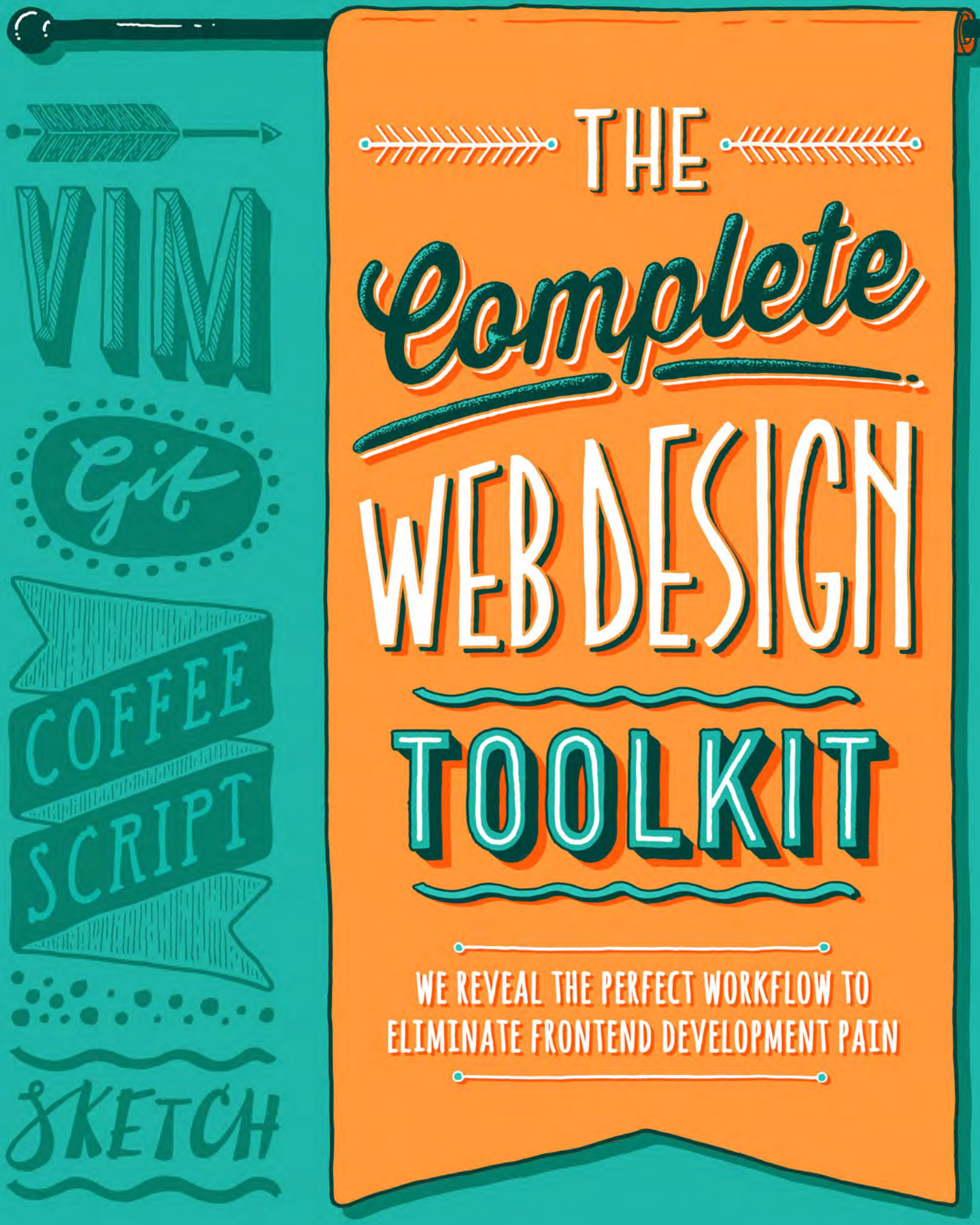
You'll find additional imagery, exclusive audio and video content in every issue, including some superb screencasts that tie in with the practical projects from the issue's authors. **Don't miss it!**

TRY IT FOR FREE TODAY WITH OUR NO-OBLIGATION 30-DAY TRIAL AT
UK: netm.ag/itunesuk-260 US: netm.ag/itunesus-260

Available on the App Store

WEB ESSENTIALS




VIM
git
COFFEE SCRIPT
SKETCH
THE
Complete
WEB DESIGN
TOOLKIT
WE REVEAL THE PERFECT WORKFLOW TO
ELIMINATE FRONTEND DEVELOPMENT PAIN

Part 1.

In the first of a two-part series, **Adam Simpson** and **Neil Renicker** tackle frontend development pain and staying ahead of the curve

It would be impossible to document every frontend tool and workflow in an article … and even if we did, it would be overwhelming and a bit discouraging. We decided to outline a comprehensive tool belt, kitted out with some of the best equipment, and share a workflow that gives the tools meaning and context. These are definitely not the only right answers, but we've found them invaluable.

Let's talk about editors

As a frontend developer, a huge percentage of your time is spent in your editor. Since we take pride in the code we author, configuring a proper editor is the first step.

The big two

There are dozens of text editors to choose from, but we want to focus in on two: Sublime Text (*sublimetext.com*) and Vim (*vim.org*). Before you protest that we left off your favourite editor, hear us out.

Our hope is that you take the time to learn the dark corners of your editor. Only then will you begin to alleviate pain points and become more productive.

Let's take a look at Sublime

Sublime Text, our first recommendation, is available on all three major operating systems, and it's blazing fast. Better yet, it has a thriving plugin ecosystem, so it's infinitely customisable. Here are a few essential Sublime plugins:

AUTHORS

ADAM SIMPSON
(@a_simpson) is a frontend developer at Sparkbox (*seesparkbox.com*); he's passionate about semantic markup, crafting quality code and creating accessible experiences. His personal site is at *adamsimpson.net*

NEIL RENICKER
Neil (*@neilrenicker*) is a converted print designer who brings that sensibility to his role as a frontend dev at Sparkbox. He writes about working on the web at *neilrenicker.com*

ILLUSTRATION

TOBIAS HALL
Tobias (@tobiashall) is an illustrator, designer, mural artist and hand-letterer extraordinaire working out of London (*tobias-hall.co.uk*)

STATIC DESIGN TOOLS

As frontend developers, we're often involved in the design of the projects we build. Many designers still love Photoshop and Illustrator for their design work, but there are are some great newcomers worth considering (see ***netm.ag/4tools-255*** for more on this). Our favourite new

Fine lines The Sketch homepage (bohemiancoding.com/sketch)

tool for web and UI design is Sketch (*bohemiancoding.com/sketch*). It's a vector-based design tool akin to Illustrator, but lighter, and intended for the web. Common UI design tasks such as rounding corners and adding drop shadows are instantly accessible, and exporting your work is effortless.

Keep your eye on these too: Froont (*froont.com*), Macaw (*macaw.co*) and Adobe Edge Reflow (*html.adobe.com/edge/reflow*). These apps all let designers work in an

New kids Fresh blood in the design tools environment

environment closer to the code. A tool that generates code is a great place to start for creating design mockups, and might give you a head start on projects. But be careful: don't let your tools generate all your code under the covers. You're responsible for what you create, so you'll feel better if you know what it's doing!

- Package Manager (*sublime.wbond.net*)
- Emmet (*emmet.io*)
- SidebarEnhancements (*github.com/titoBouzout/SideBarEnhancements*)
- GitGutter (*github.com/jisaacks/GitGutter*)

Vim, Sublime's ugly cousin

Vim is our second choice because it has a much steeper learning curve than Sublime: you might think of it as being the next level up. The power of Vim lies in its system of movements and macros. If you want to dive in and start understanding Vim's craziness, head over to your local command line and run `vimtutor`.

After you've progressed through vimtutor, you can start cultivating your own .vimrc file. The .vimrc is the configuration file for Vim, and is the place where you define all of your plugins and customisations. A couple well worth checking out are *github.com/amix/vimrc* and *github.com/mutewinter/dot_vim*, which just so happens to be from our co-worker, Jeremy Mack (@mutewinter).

Pick the right tool

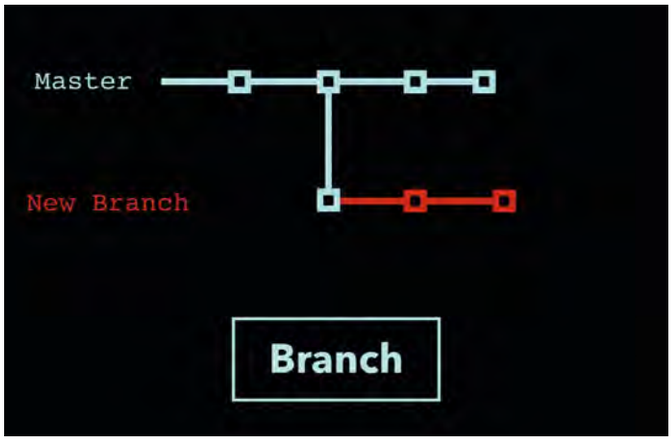
If neither Sublime nor Vim makes sense for you, here's a list of perfectly acceptable alternatives:

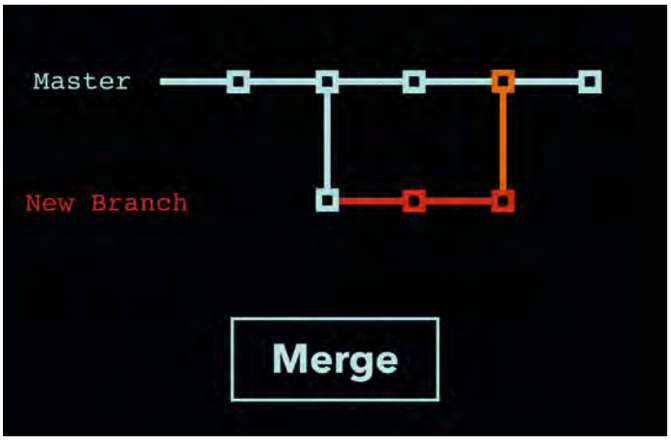
- Coda (*panic.com/coda*)
- Espresso (*macrabbit.com/espresso*)
- Atom (*atom.io*)
- TextMate 2 (*github.com/textmate/textmate*)

As always, be ready to defend your decision, but be willing to admit it if you run into a superior tool.

Protect your code with version control

Writing code is hard enough. Collaboration with other developers is harder yet. Typical collaboration tools fail when applied to code: email, USB drives, mounted network drives,





Below left Github's Atom editor (*atom.io*) shows promise as an extensible, comfortable editor

Above left Illustration of a branch in a version control system (VCS)

Above right Illustration of a merge in a VCS

and Dropbox shares are all great for sharing pictures, movies, or presentations. However, these tools offer no solution to the problem of multiple developers modifying the same codebase at the same time. There has to be a better way.

Version control to the rescue

Version control is "a system that records changes to a file or set of files over time so that you can recall specific versions later" (*git-scm.com/book/en/Getting-Started-About-Version-Control*).

WRITING CODE IS HARD ENOUGH. COLLABORATION WITH OTHER DEVELOPERS IS HARDER YET

Git (*git-scm.com*), Mercurial (*mercurial.selenic.com*) and Subversion (*subversion.apache.org*) are all version control systems (VCS). They enable a group of devs to seamlessly contribute to the same codebase.

Let's learn some terminology

'Pull' is a download from a remote version control server. 'Push' is an upload to a remote server. 'Branch' is a complete duplicate of the main code line. 'Merge' is taking a branch and merging its changes into another branch. A 'conflict' is when the software cannot automatically merge two changesets together and you have to manually complete the merge. Now that we're all speaking the same language, let's dive in.

The perfect source control workflow

We use Git and a remote Git server such as GitHub (*github.com/features*) or Bitbucket (*bitbucket.org*). We love Git's effortless branching, blazing speed and superb reliability. We use GitHub or Bitbucket because they offer incredible features such as issue tracking, a place for discussions about the code, wiki hosting, and a Git remote server.

Our workflow involves creating a branch for every change to the codebase. These are commonly called 'feature branches'. If you want to contribute code to a project, clone the Git repository and create a new branch off the master branch. This new branch is your sandbox: your changes do not have any effect on the master branch. This enables you to experiment with the code – you can break things as much as you like. This feature branch is your own little version of the project.

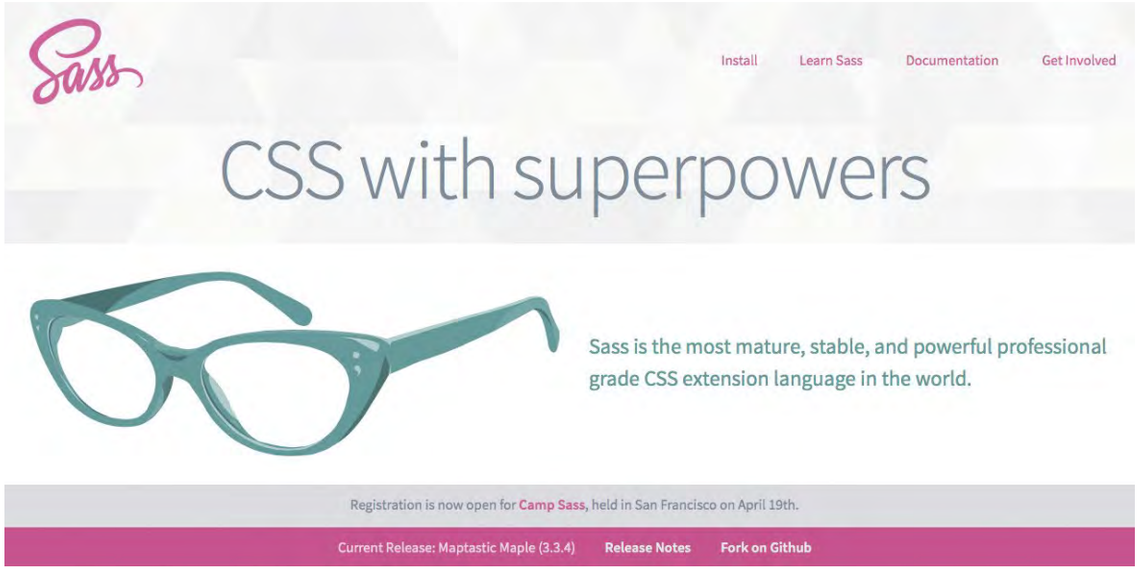
Once you are ready to share your hard work with the rest of your team, create a 'pull request' on GitHub. A pull request is a GitHub feature for merging code into master. We typically take this moment as an opportunity to peer review the work before merging in the new code. To recap, our workflow boils down to these steps:

1 Create a new branch from master
2 Make changes to that branch
3 Push to GitHub/Bitbucket and create a pull request
4 Review the pull request with a co-worker
5 Merge the branch into master

This workflow will make you more productive and safeguard your code. Go branch crazy; enjoy the flow.

Preprocessing: end manual labour

Computer languages have always evolved. This constant evolution is why we love our industry. ▶



Change is exciting, but HTML, CSS and JavaScript have remained static for years. Consider this:

- HTML is an often redundant markup language
- CSS has a 'flat' styling syntax
- JavaScript is weird

That's where preprocessing comes in – if you've been holding off, there's never been a better time to jump in. Preprocessors are high-level languages that improve upon those we already have, adding better syntax and missing features. The code then gets compiled into its native language. Preprocessors let us live in a fantasy world where we can augment these native languages as we see fit.

The important ones to focus on as a frontend developer are CSS and JavaScript preprocessors. Here's what's available to you:

CSS Preprocessors

- Less (*lesscss.org*)
- Stylus (*learnboost.github.io/stylus*)
- Sass (*sass-lang.com*)

Above Among CSS preprocessors, Sass has gained broad appeal

Right A pull request viewed on GitHub

Top right Preprocessors improve on the languages we already have, adding better syntax and features

The choice of which CSS preprocessor you use is entirely up to you. It's worth spending some time in the documentation of each of the projects and determining which one fits your requirements. You can also try all these flavours through such services as CodePen (*codepen.io*) or jsBin (*jsbin.com*).

We prefer Sass (the .scss variant), mostly because it has gained broad appeal, and it still feels a lot like CSS. Sass has always felt like CSS 2.0 to us: it's comfortable and familiar, but it has superpowers that CSS can't even fathom.

JavaScript Preprocessors

In the realm of JavaScript preprocessors, CoffeeScript (*coffeescript.org*) is the most popular. In CoffeeScript, functions are clean and beautiful, most punctuation is optional, and conditional statements can be written in plain English.

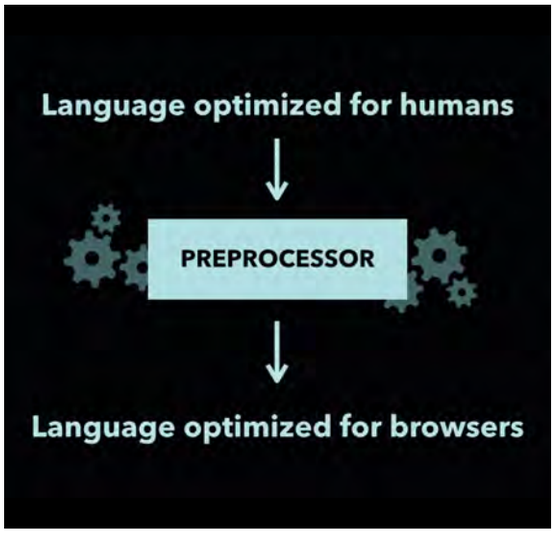
How do I get started with preprocessing?

You may want to consider applications such as CodeKit for Mac (*incident57.com/codekit/index.html*) or Prepros for Windows and Mac (*alphapixels.com/prepros*). These are third party applications aimed at making preprocessing dead simple. If you're working on your own, or on a really small team, they might be all you need.

It's challenging, however, to work on a distributed team using a third party application. Dealing with versioning and environments becomes a real pain point. That's where open source software comes in.

Open source task runners

Open source task runners can run local servers, minify/concat/lint our code, condense images, and even refresh our browser when we save new code.



Here's the best part about open source task runners: they include text-based configuration files that can be checked into source control. This ensures that every developer working on the code can work in the same environment and has access to the same third party libraries.

Getting started with task runners

There are several open source task runners that could do the job. Grunt (*gruntjs.com*) is the most popular. Gulp (*gulpjs.com*) is a promising newbie,

IF YOU'VE BEEN HOLDING OFF PREPROCESSORS, THERE'S NEVER BEEN A BETTER TIME TO JUMP IN

and some folks (*gist.github.com/toolmantim/6200029* and *algorithms.rdio.com/post/make*) are using established software such as Make for asset compilation and task running.

We recommend starting with Grunt: the getting started guide (*gruntjs.com/getting-started*) should have you up and running. For a thorough, beginner-friendly approach, try Chris Coyier's piece 'Grunt for People Who Think Things Like Grunt are Weird and Hard' (*24ways.org/2013/grunt-is-not-weird-and-hard*).

Using a tool such as Grunt will be a big boost to your workflow. There's a lot of power in it, and once ▶

THE OPEN SOURCE STACK

You're probably noticing that many of the tools we recommend are open source projects. If open source is new to you, you'll need to take a bit of time to get your machine set up to run this software. Here's what you need to get going:

GIT

- Mac: Install Git (***git-scm.com/downloads***) or download GitHub app (***mac.github.com***)
- Windows: Install Git for Windows (***msysgit.github.io***) or download the GitHub app (***windows.github.com***)

RUBY

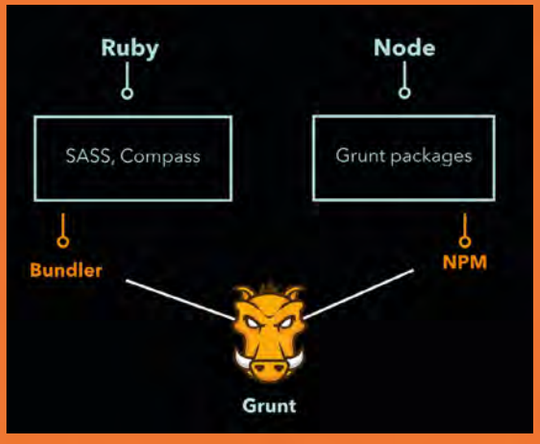
- Mac: Use the system version of Ruby in 10.9 or install Ruby using RVM (***rvm.io***)
- Windows: Install Ruby (***rubyinstaller.org***)
- Install Bundler (***bundler.io***)

NODE.JS & GRUNT

- Install Node.js (***nodejs.org***)
- Install Grunt.js (***gruntjs.com/getting-started***)

For more detailed installation guides, here are a few links to get you started:

- Mac OS X Dev Setup (***github.com/nicolashery/mac-dev-setup***)
- Love Your Frontend Tools on Windows (***seesparkbox.com/foundry/love_your_frontend_tools_windows***)



All stacked up Illustration of the open source stack

Right Grunt (*gruntjs.com*) is the most popular open source task runner

you have mastered the basics, you'll find you never want to go back.

Modular markup and styles

What's wrong with HTML?

Learning to write HTML isn't a daunting task: you only need to know a few HTML tags to get started. But as time goes on, your once-simple markup can become difficult to manage. Approaching markup from a modular perspective is a huge step forward. Fortunately, we have tooling to make this possible.

Get modular with static site generator

Writing modular markup isn't a new concept. Content management systems such as WordPress, Drupal and ExpressionEngine have this idea built in: write boilerplate code once, and then repeat it in the future as often as you need.

Static site generators (*staticsitegenerators.net*) are intended to help developers automate the management of sites without the overhead of a database. The frontend development community has co-opted them as a tool for creating static templates (HTML, CSS, and JavaScript) even if the final destination is a web app or complex CMS.

How do you choose your static site generator?

If you want minimal set-up time, and you work solo or on a small team, you might be interested in a proprietary static site generator. Hammer (*hammerformac.com*) is a Mac application that will enable you to get started quickly in a friendly native environment. Cactus (*cactusformac.com*) is a brand new Mac app promising similar benefits. These applications are essentially good-looking wrappers around a templating language.

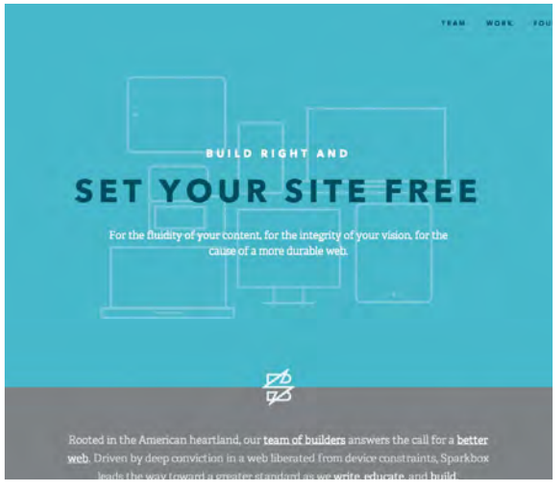
A few open source projects worth consideration are Jekyll (*jekyllrb.com*) and Middleman (*middlemanapp.com*). We're smitten with a newcomer to the party, Assemble (*assemble.io*), which is flexible enough to facilitate making decisions about the way we want to structure our project, but opinionated enough to keep us from spinning our wheels.

Understanding Git Stash

Git may feel overwhelming at first, but once you're comfortable with its basic structure you can begin to appreciate its power. One of our favourite 'power user' features is Git Stash. To understand Git Stash, you have to understand the concept of clean or dirty states in your repository. A clean state has no uncommitted changes; you have saved everything, and Git knows about all your changes. A dirty state is the opposite: there are changes you haven't committed yet. Moving from a dirty state to a clean state is the soothing rhythm of a Git repo.

In Git Stash you have an alternative that encourages experimentation and problem solving. Let's say you made some changes. You want to commit the good ones, but you don't want to lose the half-baked ones. Stash swoops in to the rescue. Run `git stash` in your project: now your repo is in a clean state. Your changes can be re-applied from the stash at any time. You can even view all your stashes by typing `git stash list`.

We hope you consider adding Stash into your workflow; it's magical! To dive deeper, head over to the documentation (***git-scm.com/book/en/Git-Tools-Stashing***).



Far left Hammer (*hammerformac.com*) is a Mac app that acts as a first step towards templating and modular markup
Left The new Sparkbox website (*seesparkbox.com*) is built entirely as a static site with Assemble, with the exception of the blog! More info at *seesparkbox.com/foundry/static_ee*

Getting Started With Assemble

We've created a demo project (*github.com/sparkbox/br-frontend-demo*) exploring how Assemble works.

Assemble enables you to separate your site into three major components: layouts, data, and partials. Layouts are like themes, or page types. Data is your content: stuff that can change regularly. Partials are markup snippets, reused across your pages.

MODULAR MARKUP ISN'T A NEW CONCEPT: WRITE BOILERPLATE ONCE, REPEAT AS OFTEN AS YOU NEED

Most static site generators have these notions, with different ways of handling and naming them. In the above demo, you can see this separation. We consign layouts, partials and data to their own special homes.

Living the Dream: Modularise All the Things

This finely tuned structure facilitates working efficiently. Instead of repeating the same code, we can iterate over each piece of data in our data file:

```
{{#each data-item }}
  {{> _partial }}
{{/each}}
```

There! We have programming power with our markup. Better yet, no content is written in the markup. It's all generated from a separate data file.

Let's make our CSS modular too! We can now give our CSS files the same name as our HTML module names. Take a look at the the `/scss` directory in the demo project, and compare it with the markup files in `templates/partials`. We've modularised our markup into bite-sized chunks, and our styles match them. You can imagine the long-term organisation and maintainability benefits this structure gives you.

What Now?

Using Assemble isn't the Holy Grail. We've found it useful, but these same principles also apply to your preferred static site generator. Remember, we're looking for these three elements of separation:

1 Layouts
2 Partials
3 Data

If you adopt this modular mindset, your projects will become more organised and scalable.

Let's Wrap This Up

That's a lot to digest: once you get serious about tooling, it can seem you've taken a step towards complexity. There's no need to bite all of this off at once. In next month's issue, we'll explore local servers, productivity, device testing, browser testing, and automated deployment. Stay tuned!

These tools aren't magical; exploring them is crucial. If one doesn't fit your team, discard it and keep looking. If you see room for a tool that doesn't exist, stand on the shoulders of the open source giants and make one to share with the community!

Thanks to Jody Thornburg (@jodytburg) for her advice and assistance in reviewing this article

THE Complete WEB DESIGN TOOLKIT

AUTHORS

ADAM SIMPSON
Adam (@a_simpson) is a frontend developer at Sparkbox (*seesparkbox.com*). His personal site is *adamsimpson.net*

NEIL RENICKER
Neil (@neilrenicker) is an ex-print designer turned frontend dev at Sparkbox. Read more at *neilrenicker.com*

ILLUSTRATION

TOBIAS HALL
Tobias (@tobiashall) is an illustrator, designer, mural artist and hand-letterer extraordinaire (*tobias-hall.co.uk*)

Part 2.

In the second of our two-part toolkit series, **Adam Simpson** and **Neil Renicker** reveal the tools they use to streamline those vital but time-consuming tasks

Last time, we covered the frontend tooling stack from top to bottom – but there's still plenty more to learn. Sometimes the biggest pain points for frontend developers are on the fringes of our workflow. It's the little things that crop up throughout the life of a project that can be the most difficult. This article will explore these pain points, and consider how modern computing and frontend tooling can be used to alleviate them.

Productivity

Have you ever thought: "All I'd need is one extra hour per day, and I could get so much more done"? It's tempting to think that adding extra hours would magically make us more productive, and free us up to do the things we love. Well, you're in luck – incorporate a few productivity hacks into your workflow as a frontend developer, and you might get that extra hour after all.

At its core, productivity is really about being lazy ... the good kind of lazy. Do you know the acronym DRY? It stands for Don't Repeat Yourself, and it's an adage you should take to heart. As you analyse your processes, you'll probably find a lot of areas where you could optimise your workflow and spend less time spinning your wheels.

Take control of the command line

We've seen a lot of tweets recently about whether or not a frontend developer should learn to use the command line. Our answer is, "Absolutely yes". Sure, you can be a talented web worker and never crack open Terminal, but we believe you're missing out on a lot of productivity gains.

Start by learning basic navigation commands: `cd`, `ls` and `pwd`. From there, you can start to learn to manipulate files, install developer packages and run build systems. If you're just getting started, give *linuxcommand.org* a try. For an even gentler start, we like John W. Long's article, The designer's guide to the OSX command prompt (*netm.ag/prompt-256*). When you're ready to go deeper, you might enjoy our article on the Sparkbox Foundry: 'Command Line Tools for Frontend Developers' (*netm.ag/commandline-256*).

Stop clicking so much

Using your mouse can be cumbersome. Sure, sometimes it's just the best way to navigate an interface, but for everyday tasks, it can really slow you down. Here are a few of our favourite apps for transitioning away from the mouse and towards the keyboard:

Alfred (*alfredapp.com*) is a free Mac application that gives you incredible power in your keyboard for actions like launching apps, finding files, and even running custom scripts or workflows. The single best piece of advice we can offer for boosting your productivity is to go and download Alfred right now. Or if you're on a Windows machine, you might give Launchy (*launchy.net*) a try.

Dash (*kapeli.com/dash*) is another of those apps we can hardly imagine working without. It gives you offline access to code documentation. While that might not sound like much, it's worth every penny.

ALFRED WORKFLOWS

Alfred (*alfredapp.com*) gives you incredible keyboard control over actions like launching apps, finding files, and even running custom scripts or workflows. Here are a few of our favourite Alfred workflows:

- **GitHub (*netm.ag/alfred-github-256*) Search and launch GitHub repos with a few keystrokes.**
- **Pinboard (*netm.ag/alfred-pinboard-256*) Search for your bookmarks saved on Pinboard (*pinboard.in*).**
- **Browser tabs (*netm.ag/alfred-tabs-256*) Search through open tabs in Chrome or Safari and jump to a particular one.**
- **Colours (*netm.ag/alfred-colours-256*) Type 'c' into Alfred, or paste in a hex colour value, then launch that colour in the system colour picker to manipulate it. You can also get alternate syntaxes of the same colour: paste in a hex value, then copy out the RGB value, for example.**
- **Harvest (*netm.ag/alfred-harvest-256*) Start and stop your timers and add notes to the Harvest time-tracking app.**
- **Can I use... (*caniuse.com*) Search from Alfred to *caniuse.com* to determine browser support for specific web APIs and CSS features.**
- **Show hidden file extensions (*netm.ag/alfred-hidden-256*) Show or hide all the invisible files in Finder: overwhelming, but sometimes necessary.**

▶ Dash is completely based on keyboard commands, and it's blazingly fast. We've never had so much fun browsing code documentation.

But here's the best part: Dash has a wonderful workflow for Alfred that enables you to instantly search code documentation from anywhere on your system. Look for the Alfred workflow in Dash's `Preferences>Integration` pane.

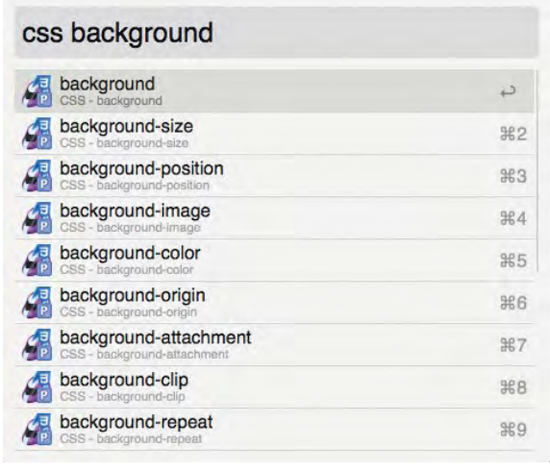
Tools for handling text If you're itching for even more productivity apps, here are a few of our favourite tools for dealing with text:

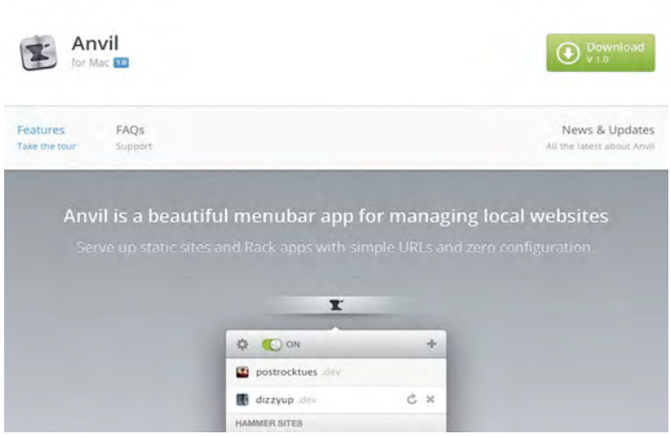
- Markdown (*daringfireball.net/projects/markdown*) enables you to quickly write semantic documents that can be compiled to HTML.
- nvAlt (*brettterpstra.com/projects/nvalt*) is a scratchpad and repository for your writing. It includes Markdown support, great keyboard shortcuts, and the ability to quickly search all your notes.
- TextExpander (*smilesoftware.com/textexpander*) creates a library of short keywords that expand into longer words or phrases. We use it for tasks like typing our email and mailing addresses, code snippets, and any phrases we use a lot.

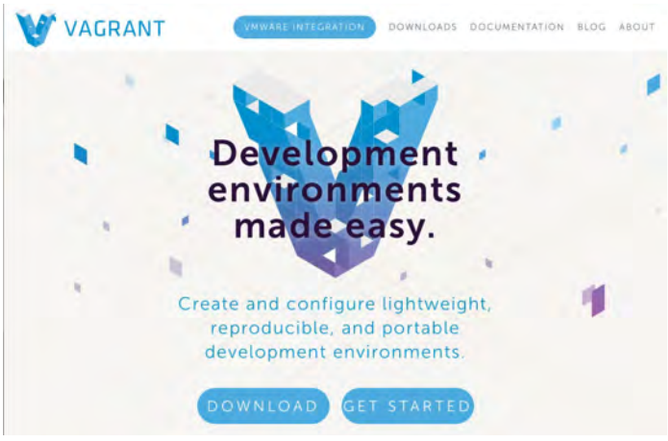
LOCAL SERVERS

Pushing changes to a remote server many times per day creates lots of pain. Working locally drastically shortens the time between making a change in your code and seeing the change reflected in your browser, thus enabling you to iterate quickly and experiment much more freely.

There are several software packages that help make setting up a local development environment much simpler. One of our personal favourites is Anvil (*anvilformac.com*), a Mac app that sits in your







Far left Dash stores snippets of code and searches offline documentation sets for over 130 APIs. It integrates with keyboard shortcut tool Alfred

Above left Vagrant lets you create a standard local development environment for you and everyone on your team

Above right Anvil makes setting up a local development environment a breeze, giving you a local server and .dev domains

menu bar. It gives you a local server and pretty .dev domains.

There are also more fully featured tools that provide server and database software to run CMSs and apps. MAMP (*mamp.info*) and WampServer (*wampserver.com*) are two such packages. Available for Mac and Windows respectively, these packages provide a complete Apache, MySQL, and PHP environment. If you like having more control over your local server configuration, you can roll your own environment by installing Apache, MySQL and

YOUR BROWSER IS A DESIGN TOOL. THAT'S NOT SOMETHING YOU HEAR EVERY DAY ...

PHP from source, which enables you to mirror a production environment on your local machine.

A fantastic plugin that we're using in our Grunt process is grunt-contrib-connect (*netm.ag/grunt-256*). This plugin spins up a basic HTTP server and enables live reloading of your pages. Sweet!

One new approach is to use a Virtual Machine (VM) to clone your production server configuration to your local machine. Services such as Docker (*docker.io*) and Vagrant (*vagrantup.com*) provide ways to accomplish this. Both packages give you text-based configuration files that reliably and predictably spin up VMS-matching criteria you specify, and enable you to check that configuration into your version-control system. These solutions require more set-up time, but may provide the ultimate local development experience.

Working locally may require an organisational change or a temporary step backwards in efficiency while you get all the pieces set up locally, but ultimately, it'll be worth it.

BROWSER AND DEVICE TESTING

Remember those days when you had to test your JavaScript by writing alerts?

```
alert("JavaScript is loaded!");
```

Hopefully, this isn't your first stop for debugging JavaScript these days, but it reminds us of a time when tooling for testing browser code was archaic and unfriendly.

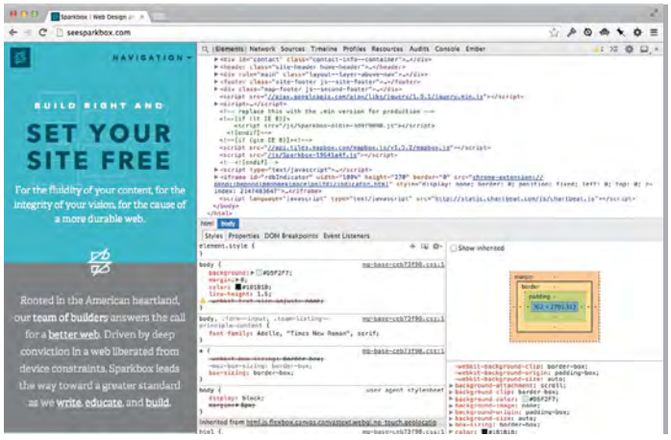
In 2006, Firebug (*getfirebug.com*) was released. Browser debugging tools existed before then, but Firebug tied the tools together and made significant improvements to them. The next year, Microsoft shipped a developer toolbar compatible with IE6 and IE7. Finally, in 2009, Microsoft shipped IE8 with developer tools built in.

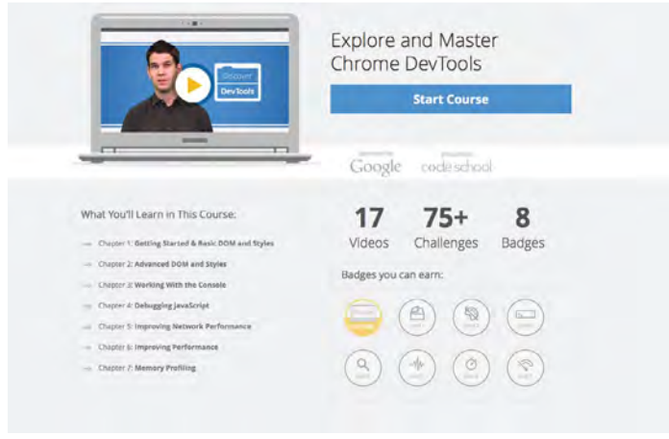
As you can see, we haven't had excellent browser debugging tools for that long, really. It's easy to forget how good we have it now.

THE BROWSER AS DESIGN TOOL

Your browser is a design tool. That's not something you hear every day, is it? We're starting to think that way, though. Our modern browsers are equipped with such sophisticated developer tools that they are no longer the last stop in our development process. Instead, they are starting to become one of the first.

Many of us began our foray into the web via traditional graphic design. Naturally, we approached the web the way we approached advertising: starting by designing a series of static, high-fidelity mockups that our clients could review. ▶





Over time, though, we've begun to realise that working in such a fashion just doesn't complement the fluid, rapidly changing nature of the web. And with advanced tooling giving us fine-grained control over our work, right there in our browsers, we have few excuses left not to start working in the new way.

Using developer tools

First things first: view the source of your web page. We love Chrome's DevTools, so we will focus on Chrome here. Most of these principles apply across browsers, although the terminology may differ slightly. On the Mac, hit `Cmd+Alt+I` to open the browser's developer tools and view the source. You can also right-click on any element in your web page, then select `Inspect Element` to open the Inspector and focus on that element.

Once you open up your inspector, you'll be greeted with the friendly Elements tab. This shows the structure of your HTML page, and enables you to explore and edit your DOM. Did you know you can drag DOM elements around, edit them by right-clicking, and delete them by pressing your `Delete` key? This is starting to feel like a design tool!

Just underneath the Elements tab, you'll find the Styles pane. The CSS that applies to your currently selected DOM node will appear in this pane. Here's the best part, though: you can edit these styles in real time – not unlike the way you would in a design tool – delete old styles or add new ones. In Chrome's developer tools, it's good to know that you can select the CSS value, then simply use your Up and Down arrow keys with variations of Shift and Alt to make incremental changes to numeric values.

We love using a layout where the Inspector sits to the right of the browser. This enables you to expand and contract the Inspector pane, which flexes your website for testing responsive designs. Most browsers offer a menu item to toggle this view.

Digging deeper

There's a lot more to explore in Chrome's developer tools. Spend time tinkering with the Resources tab, the Console tab for debugging JavaScript, and the Audits panel for getting a handle on your site's performance. We recommend reading the Chrome DevTools documentation (*netm.ag/devtools-256*)

AS YOU ENCOUNTER PAIN POINTS, TAKE A MOMENT AND TRY TO IMPLEMENT A MORE GRACEFUL SOLUTION

and watching CodeSchool's excellent free course (*discover-devtools.codeschool.com*).

Automated deployment

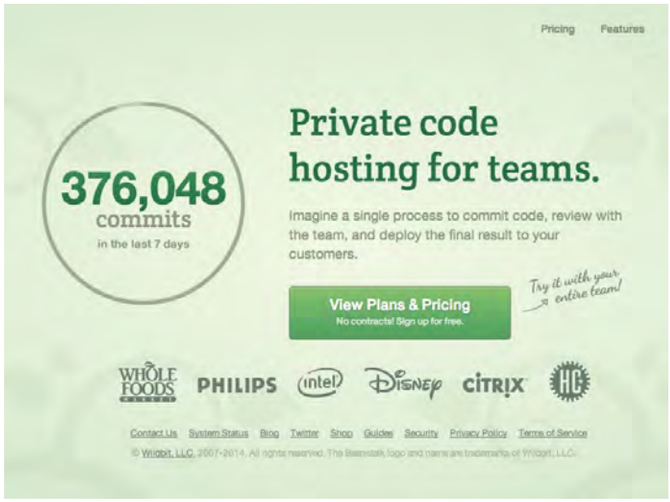
Manually dragging and dropping, copying and pasting, or selecting and moving files is not a great way to deploy your project. What if you forget one hidden file that holds your app variables? Or if someone else has to work on the deployment? Will they know what to do? The tools we have discussed so far make developer tasks repeatable, predictable, and store them in source control. We can and should approach deployments with the same ideals.

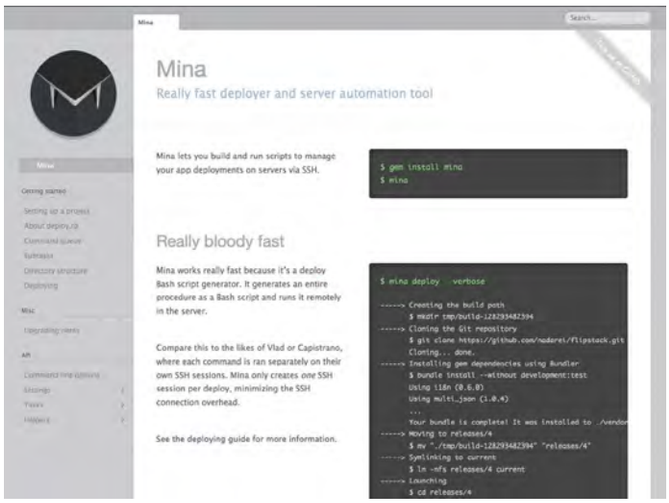
This page, left Setting your browser developer tools' Inspector as a vertical pane makes it easier to check responsive designs

Right Need help with Chrome DevTools? Code School has an excellent free course (*discover-devtools.codeschool.com*)

Facing page, left Beanstalk lets you deploy from a source-control repository to a remote host, and works well for simple deployments

Right For complex app deployments, Mina lets you build and run scripts on servers via SSH





There are several services, like Beanstalk (*beanstalkapp.com*), FTPloy (*ftploy.com*) and Deploy (*deployhq.com*) that enable you to deploy from a source-control repository to a remote host. These work great for simple deployments. However, we've found that we need more control and more functionality in our deployments.

We've used the open-source projects Capistrano (*capistranorb.com*) and Mina (*nadarei.co/mina*) to automate our deployments. These projects are similar in their functionality, with a few trivial differences. At their core, their tasks are written in Ruby, and send the server a Bash script generated from the Ruby tasks. You can think of them as pre-processors for Bash.

Deployment programs like Capistrano and Mina will Deployment programs like Capistrano and Mina will use SSH (Secure Shell, a standard secure method for remote logins) to access your server and git-clone your project. Once the program clones the project, it then proceeds to perform whatever scripted actions have been written in the deployment configuration file. Automated deployments eliminate the pain we've all felt when manually dragging and dropping files onto an FTP server – let the computer do that hard work for you.

LET'S WRAP THIS UP

The biggest lesson to take away from our exploration of frontend tooling is that you shouldn't be afraid to explore. There's no need to adopt every tool at once. As you encounter pain points, take a moment and try to implement a more graceful solution. Don't know a more graceful solution? Do some research. And as you learn, be sure to share your findings with the rest of our community.

BROWSER DEVICE TESTING

Ah, device testing. It's one of the most important parts of our job as frontend developers. So how do we make it less painful? For most of us, relying on physical devices isn't practical. That's where virtualisation comes in. Virtualisation enables us to simulate other operating systems so we can do the testing on our local machine.

Here's a rundown of our favourites:

- **Use VMWare (*vmware.com*) or VirtualBox (*virtualbox.org*) on your Mac to test in Internet Explorer. You can download free short-term Windows VM licences at *modern.ie*, or install them with one command in your terminal (learn how at *netm.ag/testing-256*)**
- **Use BrowserStack (*browserstack.com*) to do testing on virtual machines for almost every device you could imagine. It's all right there in the browser, so set-up is effortless**
- **Use Xcode's built-in iOS simulator for testing mobile versions of iOS, and Google's Android Emulator (*netm.ag/androidemulator-256*) for testing your designs on Android**

We also really love Adobe Edge Inspect (*netm.ag/edge-256*) and Ghostlab (*vanamco.com/ghostlab*) for setting up synchronised testing on your mobile devices. Device testing is a very important problem to solve. Don't be afraid to invest some time and money in a sorting out a set-up you're really happy with.

ABOUT THE AUTHOR

JOY BURKE

w: *heyjoyhey.com*

t: @*heyjoyhey*

areas of expertise:
Print and interactive design, UX, illustration

q: Which cartoon character do you most identify with?

a: Patti Mayonnaise: she gets along well with different people, plays lots of sports – and I know no other cartoon characters with a blonde fro that so resembles my own

PRODUCTIVITY

5 TOOLS FOR DESIGN EFFICIENCY: PART 1

Joy Burke shares five tools that will help you stay organised and efficient, freeing up more of your time to devote to being creative

The ever-growing array of tools and apps for managing your work can often feel daunting when trying to figure out which is best for our own personal needs. I've spent hours comparing apps through the gruelling process of reading reviews and test-driving free versions – and I'm sure I'm not the only one. It's been painful and time-consuming, but I've finally found the perfect collection of tools to keep me on track throughout the day. This is the first part of a series in which I'll be covering a few tools I use daily that keep me organised so I can work creatively with more ease.

1 BASECAMP

I've been using Basecamp (*basecamp.com*), the most popular project management tool in the world, for more than 10 years now. In fact, for the past couple of years prior to its recent relaunch celebrating Basecamp's tenth birthday, my picture was featured on its homepage!

I've tried other similar programs in the past, but none of them truly enable you to keep track of everything related to a project all in one place, and with such ease. I especially love the ability to start a new discussion thread on a particular file or text document, choose who to include in the message, and change the people included in each discussion without having to start a new one. It facilitates normal, real-life solutions for people to interact together on a project, and to do so quickly and easily. And though it's incredibly easy to get the hang of, its super friendly customer service team has put together easy walkthroughs, examples and videos to help get you acquainted with it.

2 KIPPT

Kippt (*kippt.com*) is a bookmarking web app and plugin. The premium version (just $5 a month – and worth every penny) enables you to create folders that are organised along the left-hand side of your web page, and then add as many lists as you need to within each folder. This makes both categorising and finding bookmarks (or as Kippt calls them, 'clips') a breeze, which I find is crucial when I'm trying to track down a quick reference for something.

For example, I have one folder called 'Design' in which I have about 15 lists including Showcases, Studios, Type Foundries, Typography and Writers. So if I come across, say, a type foundry online that I love and want to remember for later, all I have to do is click on the Kippt bookmarklet button in my browser window, choose the Type Foundries list I created from a drop-down menu and hit Save. I also love that when viewing a list page in the web app, the clips you've saved are displayed with an image from the web page along with its page title.

You have the ability to edit this page title, add tags and notes, share the clip elsewhere online, and

TOP TIP

Basecamp can also be used to organise purely personal projects, such as event dates, making notes about present ideas, and to-do lists so I make sure to have enough time to get it all done. Life saver!

Photo credit: Randy Michael Korwin Studios (*rmkphoto.com*)

mark it as a favourite, which adds it to the default Favourites list that comes automatically as part of Kippt. The community feed enables you to follow other people and add their clips to your own lists. You can even mark certain lists as 'Private', which makes keeping track of gift ideas for that special someone totally possible as well.

3 TODOIST

It feels like I had been searching for the right to-do list app for forever, until recently, when I came across Todoist (*todoist.com*). While the free version does provide far better options than most other apps of the same realm, I purchased the Premium account for about $30 for the year because I had finally found a tool flexible enough to do everything I need it to. Its minimal, modern aesthetics provide a stress-free experience, while being robust enough in functionality to cover all the bases.

Tools for our various needs are plentiful, but finding the right ones can be time-consuming

I've created several projects (both for work and for personal things like 'Life' and 'Presents'), which I've colour-coded to match the way I mark up the same projects in Gmail, Google Calendar and Basecamp. I can easily add, remove or change a due date or note to each task, drag and drop a task from one list to another, and even create projects within other projects to further help me stay organised.

The best part is that I can sync this all through the cloud (keeping safe backups) and access the list through the just-as-excellent Todoist mobile app on my phone – a crucial part of the process. If only I could get my Basecamp tasks to sync with my Todoist account ...

4 POCKET

I use Pocket (*getpocket.com*) the same way I use Kippt, but more specifically for storing reading materials in one place. There are plenty of tools that function essentially the same way, but Pocket does everything I need it to with ease and elegance that I've found more enjoyable than its competitors, such as the well-known Instapaper (*instapaper.com*).

The browser plugin enables one-click bookmarking so you can save things without getting too off-track from the task at hand. Then, from within your Pocket account, you can quickly filter through your saved content by jumping between lists, highlights, favourites and the archive. Top-menu toggles switch between list and tile views.

Each page thumbnail shows the main page title along with the primary image (if there is one) or the first bit of written content. The favicon and URL are displayed at the bottom of each page thumbnail for quick reference and a link to the primary source where you saved the page from. Search functionality enables you to track down a page using keywords. Via the Content Filter toggle you can filter all pages by either Articles, Videos or Images. Finally, viewing the saved page in detail view (by clicking on its thumbnail) displays the content with a hierarchy optimised for easy reading, whether viewed on the desktop or on your smartphone app.

5 DAY ONE

The Day One app (*dayoneapp.com*) is one of the best journal apps on the market. I use it to record notes from both my personal and professional life. When starting a new entry, the first line you write can be set to automatically save as the title for that entry, which makes navigating through notes in the Timeline view a breeze, and makes for a good way to keep them organised.

I record all of my meeting notes at work, all business-related phone calls I make, and even use it to save nice things I hear or witness so I can come back and reminisce about them later. Day One also gives you the ability to add a picture to an entry if you so desire. In addition to adding pictures for journal-like entries (like a picture from a ski trip that I am writing about), I'll sometimes add a sketch or something from a meeting to act as a quick visual reference that will help jog my memory when I want to refer to that again.

One of the best things about Day One is its easy access. For adding quick notes, it's as easy as clicking on the Day One icon in your menu bar, typing your text into the pop-up window that appears and hitting the Save button.

While each of these apps play a different role in my daily life, one thing they all possess is an awesome user experience. The right balance of flexible functionality and elegant aesthetics can make for a successful, enjoyable interaction for the end-user, an aim that should be at the forefront of any web app development and design team's minds.

As I mentioned earlier, the options available to us when it comes to choosing tools for our various needs are nothing short of plentiful, but finding the right ones to work for you can be tough and time-consuming. I hope you find these apps as helpful in your day-to-day workflow as I have.

VIDEO

To find out more about Day One, check out a promo video about the app at *netm.ag/dayone-254*

ABOUT THE AUTHOR

JOY BURKE

w: *heyjoyhey.com*

t: @*heyjoyhey*

areas of expertise:
Print and interactive design, UX, illustration

q: When was the last time you cried?

a: Last week. My boyfriend took me to see *Motown: The Musical* at the Oriental Theater, and I loved it so much it made me cry

PRODUCTIVITY

5 TOOLS FOR DESIGN EFFICIENCY: PART 2

In part two of her exploration of tools for productivity, **Joy Burke** shares more time-saving utilities she uses in her web design process

After joining a new design team about a year ago, it came to my attention that some of the tools I put to work when approaching web design projects aren't necessarily as widely used as I'd previously thought. Nonetheless, the utilities I'll be discussing in this article have in many ways become my 'third arm' when it comes to producing high-quality web design work, with precision and in a timely manner. Hopefully walking you through these select apps/utilities – which you may or may not have heard of before – will enlighten you to even better ways of getting a good job done.

1 CLASSIC COLOR METER

Whether I'm working on creating a colour palette, or just need to quickly know the value of a particular colour on my screen, Classic Color Meter (*netm.ag/ccm-255*) makes grabbing a hex code as easy as a simple set of quick keys.

Classic Color Meter is the enhanced version of a counterpart that comes as a default Apple computer utility, but shelling out the £1.99 it costs in the Mac App Store feels practically free considering the amount of time it saves me on such a regular basis. CCM offers a variety of ways to find a particular colour's value, but I usually just use it for grabbing a hex value.

With the application open and the part of the screen in view with the pixel of colour I want, simply hovering over that pixel (your mouse becomes an eyedropper) and hitting Shift+Cmd+C will copy the value (hex code, in my case) to the clipboard. Then it's super easy to paste that value into your design programs, create a new swatch with it, and be on your way. It's an easy way to take the guessing out of colour value generation.

2 0TO255

While we're on the subject of colour, I'd also like to highly recommend 0to255 (*0to255.com*) – the Adobe Creative Cloud programs aren't the only place I'm likely to head after copying a hex value via the Classic Color Meter.

Often times, I'll then open 0to255 in a browser window, paste the hex value I've just grabbed from CCM into the given field on the website and click Submit. The following page provides a simple interface displaying a spectrum of shades and tints based on the hex value you entered on the previous screen as the centre point for the scale: clicking on another colour will copy its hex value to your clipboard, which you can then take the same steps you had with CCM and paste it into your design programs to work with.

I often use this tool when I need to provide a bit of depth to certain design elements using flat colours in shades derived from the main colour. You can do similar things in other programs, but having the

TOP TIP

The WhatFont plug-in (*chengyinliu.com/whatfont.html*) enables you to push one button in your browser window to open the tool; then mousing over any dynamic text on a page will reveal what font is being used. Clicking on text opens another window giving more detailed information. It's super handy.

ability to get the spectrum in a single click, and copy another colour's value with another click, is a real timesaver.

3 FIREBUG

There are plenty of ways to inspect elements of code for a website, but I've found Firebug (for Firefox; *getfirebug.com*) to be the easiest for me to use.

Since being introduced to it several years ago, Firebug has become an essential part of my process for web design projects. Simply right-clicking (or hitting Ctrl+click) anywhere in a browser window and selecting Inspect Element with Firebug opens up the control panel. By default, two panes open side by side, with HTML on the left and CSS on the right, and it specifically brings you right to the spot, in both cases, of the element you right-clicked on. This makes it a cinch to 'edit' a web page on the fly without touching its actual code.

When I'm working with developers to transition a project from the design phase into development

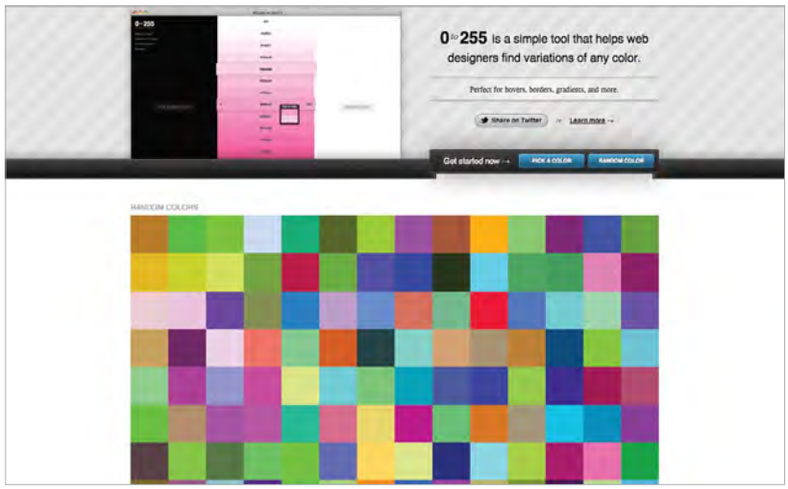
Every now and again, Hipster Ipsum makes a delightful addition to the design process

mode, I'll often use this method to review what they show me on a testing site, make specific edits with Firebug and then send those changes on to developers. It helps us fill any language gaps when trying to communicate design specs with them, and it makes it easy for them to share what they're building with designers.

4 HIPSTER IPSUM/BLOKK FONT

You may laugh at this choice, but after the first couple years I spent mocking up web designs before final content was ready (which is quite often the case), I had finally had enough with using "Lorem ipsum dolor sit amet" in my design work. On the hunt for a better solution, I came across the hilariously ironic English alternative to lorem ipsum generators: Hipster Ipsum (*hipsum.co*). All it takes is entering the number of paragraphs you need, selecting either 'Hipster neat' or 'Hipster with a shot of Latin' and, with the click of a button, you get the perfect portion of amusing filler text.

Admittedly, Hipster Ipsum isn't appropriate for every client or project. Every now and then, however, it makes for a small yet delightful addition to the design process. For mockups and wireframes that need to provide even less of a focus on typography, try Blokk font (*blokkfont.com*). It's perfect for talking through layouts, particularly during early stages of the design process.



Colour acceleration
0 to 255 (*0to255.com*) offers a quick and handy way to get your hands on the colour variations you need

5 PLACEHOLD.IT

Cats are cute creatures and everything, but I have yet to come across a project in which using various pictures of them as placeholders would help communicate a professional web design job. That's OK though – Placehold.it (*placehold.it*) is the perfect image generator for getting mockups done fast so you can more easily make smarter big-picture design decisions.

There are a couple of ways in which you can use this handy tool. Placehold.it's minimalist interface makes plugging in the pixel dimensions and outputting an image at that size a breeze. Alternatively, you can enter your desired width and height parameters into the website's URL (for example, *placehold.it/300x250* for an image that's 300 px wide and 250 px tall), and it will load a page containing nothing more than an image with those specified dimensions, which you can then save and use for your mockups. From the homepage, a number of formatting options for styling the image are also provided.

CONCLUSION TRUSTY TOOLS

The web has a way of changing even faster than the weather in the Windy City, where I'm based, and these five time-saving methods represent only a handful of the available ones for getting a good job done efficiently – it's always worth experimenting. That said, they're tools I find myself coming back to time and time again for web design projects. I hope they help you as much as they have helped me in our pursuit to create better web experiences.

VIDEO

Watch an exclusive screencast of this tutorial created by Joy Burke at: *netm.ag/tut3-255*

THE TOP 10 ONLINE LEARNING WEBSITES

The internet is brimming with learning tools. **Katie Kovalcin** takes a look at the coolest code schools on the block

Now is an amazing time to be getting started in the web industry. Not only are an increasing number of jobs becoming available, but we have many accessible educational tools popping up that make learning new skills more straightforward. The daunting task is finding the one that is right for you and your learning style. I have tested and reviewed some of the most popular schools being talked on the web today, to give you a better understanding of the pros and cons of each.

I've looked at five key elements for each tool, and scored each out of five. The things I've covered are: how well the information is presented; how easy it was to complete the tasks well; how engaging the tool was (as in, would I actually stick to it?); how much opportunity it offers for me to progress past the basics; and finally the support and help that was provided when I got stuck.

Full disclosure: I am a web designer with some prior coding knowledge, but I tried to fairly judge these as a 'beginner'. This list is not ranked in any particular order, as the services vary in range of content.

AUTHOR

KATIE KOVALCIN is a web designer currently working at Happy Cog in Austin, Texas. She has a passion for typography, branding and team collaboration
katekovalcin.com

CODE SCHOOL

codeschool.com

Of all the tools I dug into, Code School was one of my favourites. Not only does the site look great, but it offers quality content and a great student experience. You can follow a 'path', which will guide you through a succession of different courses, or you can opt for a single course on its own. There are also 'electives' on offer – additional courses you can take to supplement your path.

The videos are quirky and fun in an 'after-school special' kind of way – which is a refreshing break from typically dry video tutorials. It's like watching Bill Nye explain it – which is to say, comforting. After watching a video, you work on your own tasks – relating back to the video if required. If you get really stuck (like I did) you can 'buy' answers with points you've earned from other classes.

Code School offers some courses for free, so you can try it out for size, or you can buy a monthly subscription for $29/month. If you're looking to splash a few bucks on a learning tool, I would recommend spending your money here.

PRESENTATION ★★★★★
I loved the clever little jingles in each intro.

TASKS ★★★★★
They were an appropriate level of challenging.

ENGAGEMENT ★★★★★
I suspect I could get through a hefty amount of work seeing how far I could go down a 'path'.

SCOPE FOR GROWTH ★★★★★
Between paths and courses, you could spend quite some time furthering your skills.

SUPPORT ★★★★½
It offers a forum, and questions (mostly just brief FAQ). You can email the team, too.

CODECADEMY

codecademy.com

Codecademy is free and easy to set up. You can create 'goals' for yourself and find the skills you want to learn. These could range from applicable things like creating a portfolio site to silly, fun things like animating your name.

Once I dived into an exercise, I quickly became disappointed. There was little context or explanation of what to do. What's more, the code was right there in front of me – which, once seen, is nearly impossible to unsee. Since the tasks didn't vary at all from the examples presented, users could easily end up blindly copying and pasting the code in the tutorials, without knowing the why – which is a pretty difficult way to grasp a concept.

One nice touch offered by Codecademy is the Code Glossary – a collection of common terms with straightforward explanations - which is a good reference resource for beginners.

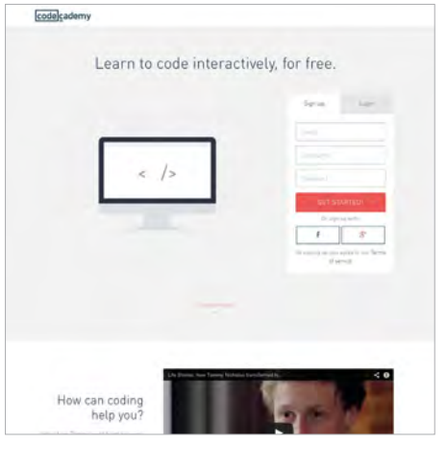
PRESENTATION ★★★☆☆
It laid everything out, but it's hard to grasp concepts with no context.

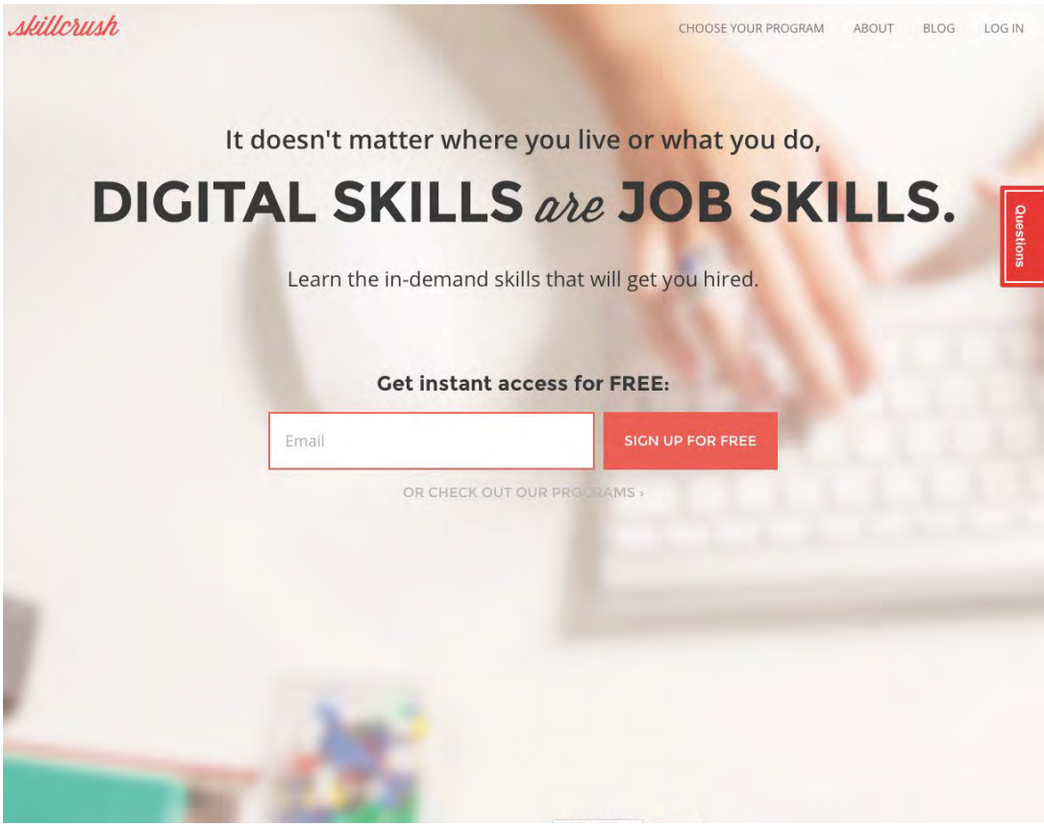
TASKS ★★★☆☆
This is difficult to gauge, as 'completing' the tasks could be as easy as just copy and pasting.

ENGAGEMENT ★★☆☆☆
I didn't feel compelled to keep going with it.

SCOPE FOR GROWTH ★★★☆☆
There's a handful of courses and extras, but definitely not as deep as other programs.

SUPPORT ★★☆☆☆
There's some FAQs, but for the most part it seems like you're on your own.





SKILLCRUSH

skillcrush.com

Skillcrush is a unique online learning resource, and the one that most closely resembles an actual curriculum in a classroom setting. Users can enrol in either a one-off three-week course, or a three-month 'blueprint' (a design- or development-focused path that takes the student through three courses that build on each other). Once enrolled, students have access to not only a classroom setting chat forum with their fellow students, but also office hours and one-on-one time with their instructor.

What I found so great about Skillcrush is the amount of attention each student receives. The instructors really are open to helping you learn for your particular needs. Instructors are willing to work with you to make sure you are learning the materials. So if learning Illustrator seems too much, submitting pencil sketches is totally okay.

The other thing that stood out was the camaraderie amongst students. Students are encouraged to publicly share their work for other students to give feedback on, which is nice for beginners.

The combination of videos, in-browser exercises and live lectures (including 'Master Classes' where industry pros guest-teach a deeper-dive class) covers a lot of different learning styles, which ensures the courses stay engaging. While the cost of Skillcrush can be a downside for some, I think it's a great place for any beginner to get a solid foundation that is actually applicable in the real world.

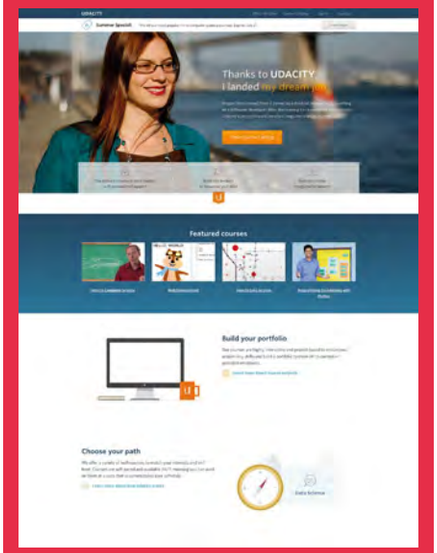
PRESENTATION ★★★★★
It covers many different formats, and tasks are clearly presented via a daily email.

TASKS ★★★★½
They were straightforward to complete.

ENGAGEMENT ★★★★★
The classroom feel made me want to stick with it.

SCOPE FOR GROWTH ★★★½☆
Limited courses offered at this time.

SUPPORT ★★★★★
The best I've seen. A dedicated instructor, fellow students and a good email response time.



UDACITY
udacity.com

Udacity offers robust guides to many different topics, with seriously advanced courses for everything from robotics to design theories in everyday life. All courses are summarised before you even begin, outlining the difficulty level, why the course is important, any prerequisites for taking it, and the project you will complete at the end. This meant I could easily scan through and choose what suited me best.

The courses walk you through a series of videos that are pretty easy to follow. You then do some mini exercises, followed by more engaging tasks like answering questions in a forum. This gave more of a 'classroom' feel than some of the other services.

Udacity's courses are very thorough, and can take anything from two weeks to six months. The downside is that the price can be steep, as the cost is on a monthly basis per course, and you move at your own pace.

PRESENTATION ★★★★☆
It's very clear.

TASKS ★★★½☆
The few courses I looked at required time-consuming equipment setup or GitHub repos.

ENGAGEMENT ★★★½☆
I liked that it felt like an online classroom setting, with the questions in-between tasks.

SCOPE FOR GROWTH ★★★★½
There is a lot of content and the price can vary based on how quickly you learn.

SUPPORT ★★★★☆
You can have a coach and ask any questions you have in a discussion.

CODE AVENGERS
codeavengers.com

Code Avengers walks you through each process in very simple, step-by-step instructions, starting with minor tasks, while you watch what happens in a mini phone simulator next to your editor. Tasks weren't so difficult that I felt the need to 'cheat', but they weren't so simple that I was just copying and pasting.

What was a little jarring was the fact you just jump right into completing the tasks without a lot of knowledge of what is happening. The real bummer, however, is that the text editor is pretty buggy. I found it very frustrating to type in.

I like that Code Avengers offers free intro courses and very fairly priced advanced courses (around $25-30). However, its offerings are a little limited right now, with only HTML, CSS and JavaScript for different levels. It does offer a Code Camp, which is a hands-on training course in selected locations.

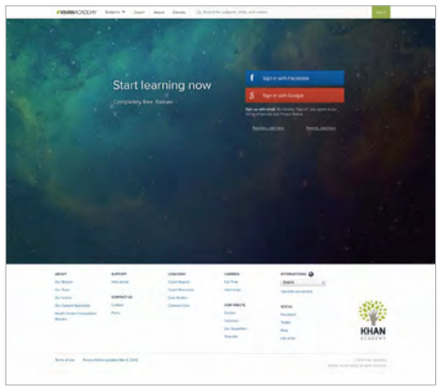
PRESENTATION ★★★★☆
It's very clear.

TASKS ★★★★½
They were straightforward to complete.

ENGAGEMENT ★★★★☆
I liked that there were set tasks with themes (like a birthday invitation).

SCOPE FOR GROWTH ★★★☆☆
There are only a limited number of courses, with three different levels to deep-dive into.

SUPPORT ★★★★☆
Help and hints are provided by a support email that promises a response within 24 hours.



KHAN ACADEMY
khanacademy.org

My first impression was that Khan Academy offers an intimidating range and volume of courses, covering everything from middle school maths to code. Since courses aren't confined to the web industry, it felt a little lacking range on web topics. There isn't a lot offered in the way of computer sciences (as they categorise it), although there are several different levels of JavaScript courses.

What's interesting with Khan Academy is that you hear someone talking while they write the code, which feels natural, like you are sitting down with someone. The courses were straightforward, easy to follow and I could then transfer the concepts to the simple exercises. Sometimes, there are mini-quizzes following a demo, which made me think about things conceptually and helped me understand what the code was doing a little better.

PRESENTATION ★★★★☆
It's very clear.

TASKS ★★★★★
It was easy to follow along.

ENGAGEMENT ★★★★☆
The exercises were engaging, but as an entire programme I don't know if I'd keep coming back.

SCOPE FOR GROWTH ★★★☆☆
Limited courses offered in the way of tech.

SUPPORT ★★★★☆
Comments and forums support the exercises.

HACK DESIGN

hackdesign.org

I'm going to be honest, I was pretty excited to try Hack Design, just to see what people are offering in the way of design courses, as opposed to just coding. It covers various facets of design, from prototyping to responsive typography and iconography – as well as more conceptual aspects like Cultivating Compassion and The Human Element.

The 'courses' are just collections of aggregated content, whether that's articles or videos from around the web. There isn't anything to 'complete' by way of tasks, but if you are after some resource recommendations to help you learn more about a specific part of design, I would turn here. There are also tool lists with recommendations from people who use them. This is an excellent catalogue of design resources for all levels. It offers great introductory content for beginners and more advanced techniques for those of us who do this full-time.

PRESENTATION ★★★★☆

It's organised very well and it's easy to find what you're looking for.

TASKS 0

None.

ENGAGEMENT ★★★★☆

I will definitely be returning to these articles for reference at a later time.

SCOPE FOR GROWTH ★★★★☆

All design levels could benefit from this content.

SUPPORT 0

There isn't a whole lot to get 'stuck' on.

UDEMY

udemy.com

Udemy is an expansive catalogue, filled with thousands of other topics besides web and technology, with prices ranging from free to hundreds of dollars. I would venture to say you could find just about anything you wanted to on here.

A nice feature is that you can see the number of students who have taken each class, as well as a star rating and reviews. The reviews help users find the real deep-dive content of the lessons (for example, one stated "a lot of this is fluff. Skip to the last video for the overview", which I found helpful).

The downside is it doesn't seem to be well curated and even links to videos from other resources. It can be pretty hard to find a really great course – unless you are willing to shell out a lot of money. The videos themselves are not of the best quality, and I found it difficult and boring to follow along with the text on the screen. There are supporting text resources, if you like that style of learning.

PRESENTATION: ★★☆☆☆

It was pretty difficult to follow.

TASKS ★★☆☆☆

It was just following along with a video.

ENGAGEMENT ★☆☆☆☆

Very dry videos.

SCOPE FOR GROWTH ★★★★☆

There are a lot of courses in the catalogue.

SUPPORT: ★★★★☆

It offers a large, searchable catalogue for support, as well as instructor support.

TUTS+

tutsplus.com

Tuts+ offers tiered subscription plans, which include access to ebooks in addition to your courses. It offers a large catalogue filled with different specifics of code like SVG, Jekyll and Sass. I take this to mean it has ongoing education courses that even professionals can take to further their skills, not just bare-bones stuff for beginners. Tutorials come in the form of write-ups about different topics, in addition to actual courses.

The courses require you to download source files and start working within those while following a video. If you prefer watching videos rather than completing tasks in an editor, this could be something to consider.

The videos are divided into mini-lessons, which makes it much easier to break things up and work on different tasks at your own pace. Tuts+ is one of the better resources for the video (non-text editor) courses. It provides relevant and current content, which is good to keep up on.

PRESENTATION ★★★★☆

Easy-to-follow videos.

TASKS ★★★★☆

They were straightforward to complete.

ENGAGEMENT ★★★½☆

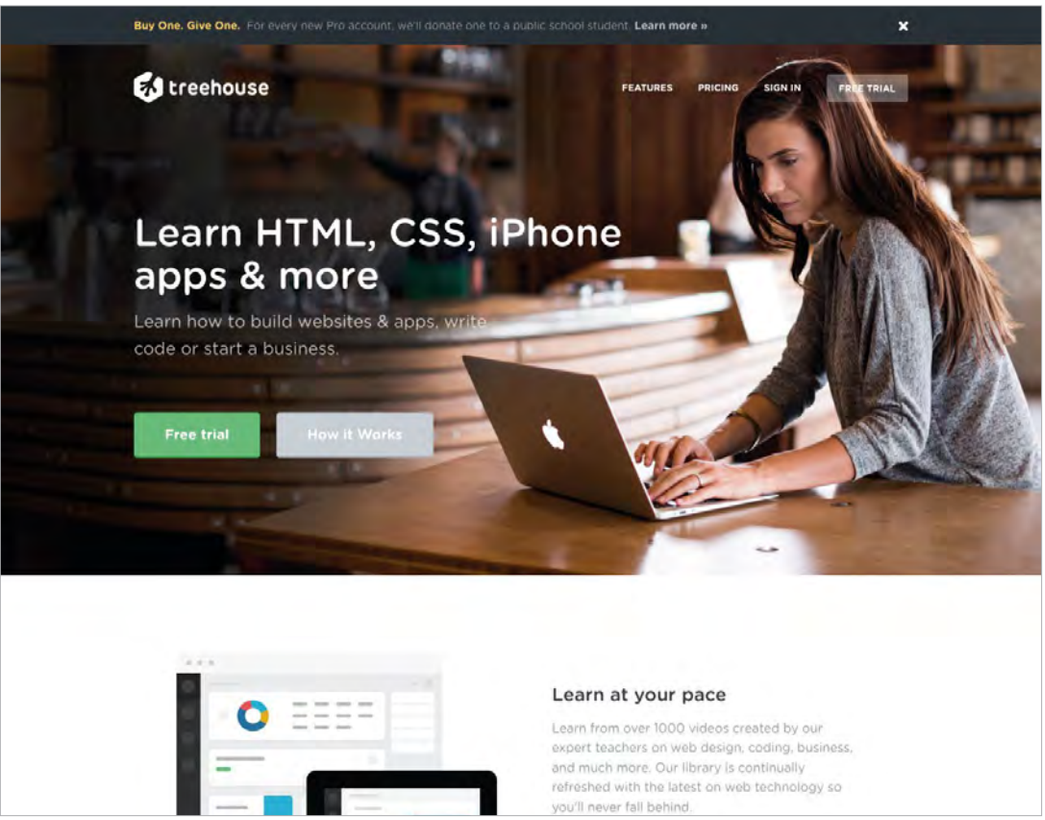
It's a personal preference, but I'm pretty unengaged when just following videos.

SCOPE FOR GROWTH ★★★★★

Offers new and relevant industry topics.

SUPPORT ★★★☆☆

You can email them (and they respond quickly!) but there isn't much else offered in way of support.



BEST OF THE REST

LYNDA
Lynda is one of the more popular video tutorial sites. It offers an abundance of content that is constantly added to with new technologies and software. Lynda is a trusted resource that many turn to when looking to pick up a new skill.
lynda.com

SKILLSHARE
Skillshare is a subscription-based service that offers hundreds of classes taught by industry professionals. Courses come complete with material that has been created by the instructors, and there are projects for you to apply your skills.
skillshare.com

MIJINGO
Mijingo is an online learning resource that features step-by-step video tutorials for both web design and development. The courses are cohesive and thorough, and some offer transcripts, starter files and additional resources.
mijingo.com

TREEHOUSE

teamtreehouse.com

Rounding out the list is one of my favourite learning tools. Treehouse has one of the most comprehensive libraries of educational materials on this list, while also being one of the most relevant. Not only does it offer a large variety of code courses, but it also offers 'workspaces' for you to try out new projects in your own time, as well as a library with videos of industry professionals' talks. It's the only one on this list to offer an iPad app to continue your learning, too.

Treehouse also offers a separate resource for careers which is extremely useful for someone trying to break into the industry. What's more, it offers courses on careers, such as how to apply for a job and how to write a business proposal. All in all, it pretty much has everything you need covered.

The projects are made up of multiple steps, made up of alternating videos and exercises. The exercises tie in nicely with what's been explained in the video, so the concepts make sense in application. If you want to go wild and start playing around with the project files outside of the course, it has them up on CodePen – a nice addition for those wanting to experiment beyond what they're tasked with.

You can try a free two-week trial on Treehouse, and from then on there are two tiers of monthly fees which give users access to all of the site's content.

PRESENTATION ★★★★★
I found this pacing to make the most sense.

TASKS ★★★★★
They were straightforward to complete.

ENGAGEMENT ★★★★★
I was compelled to actually stick to these courses and keep going with tasks.

SCOPE FOR GROWTH ★★★★½
There are a ton of topics and 'deep dives' for code, design, and your career.

SUPPORT ★★★★☆
An active forum and discussions, and additionally support email.

CONCLUSION

While this list certainly doesn't cover everything, it's a good place to start. If you have the funds to spare, I would definitely recommend Code School or Treehouse. These offered the best overall experience, content and support. That said, it's all about what's best for you and your personal learning style, so check out as much sample content as you can! It'll come down to your own preference and what content you are looking for.

BEST FOR

PRESENTATION
Code School, Skillcrush, Treehouse

TASKS
Khan Academy, Code School, Treehouse

ENGAGEMENT
Skillcrush, Code School, Treehouse

SCOPE FOR GROWTH
Code School, Tuts+

SUPPORT
Skillcrush

BEST ALL-ROUNDER
Code School

TWO SIMPLE WAYS TO SUBSCRIBE

ONLINE: www.myfavouritemagazines.co.uk/netp1y

PHONE: 0844 848 2852 AND QUOTE CODE NETP1Y

SUBSCRIBE TO net AND GET 10 ISSUES FREE*!

GREAT REASONS TO SUBSCRIBE

- Get 10 issues free*
- Get your copy before **net** hits the shops
- Delivered straight to your door
- Stay updated on the new web technologies
- Exclusive access to the world's leading experts

UK OFFERS 6 MONTH DD £25.49 (SAVE £26.89)*

1 YEAR £53.99 (SAVE £23.88)* **2 YEAR** £93.49 (SAVE £62.25)*

NOT FROM THE UK? HERE ARE YOUR EXCLUSIVE OFFERS

Europe: £33.99 (Every 6 months)

ROW: £41.49 (Every 6 months)

SPECIAL OFFER!

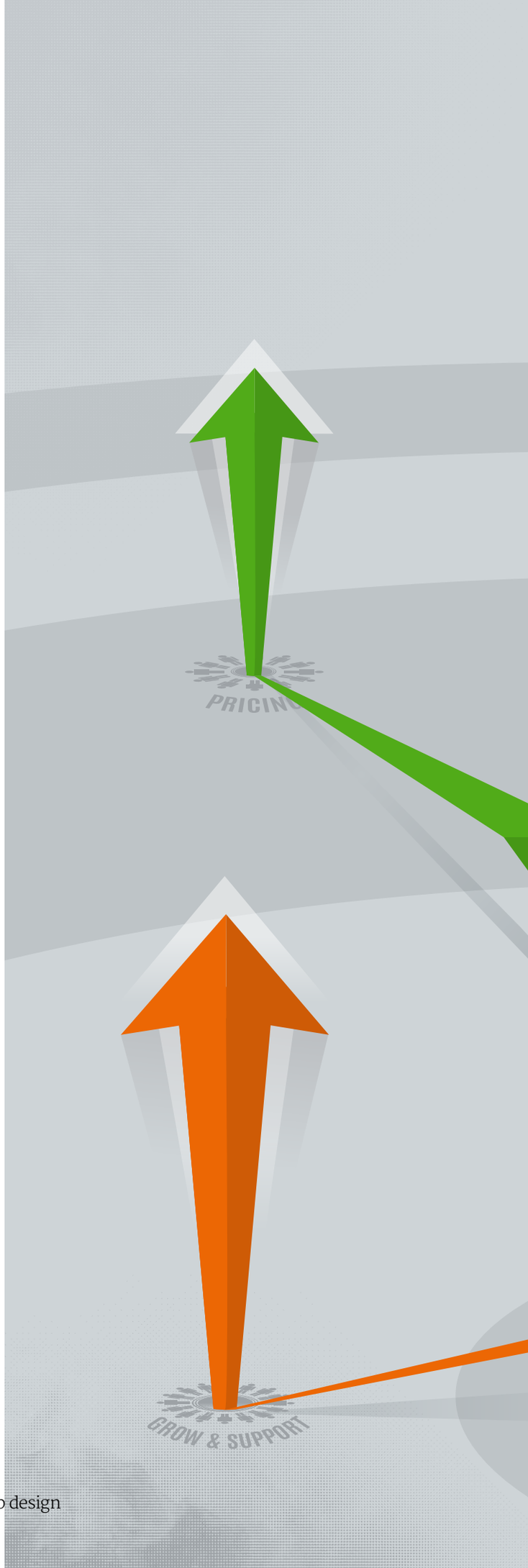
Terms & conditions: * UK savings compared with buying a two year subscription from UK newsstand. Europe and ROW have no set newsstand price and therefore we cannot advertise the specific savings you will make. You will be charged in GBP. This offer is for new print subscribers only. You will receive 13 issues in a year. Full details of the direct debit guarantee are available upon request. If you are dissatisfied in any way you can write to us or call us to cancel your subscription at any time and we will refund you for all unmailed issues. Prices correct at point of print and subject to change. Offer ends 31 January 2015. For full terms and conditions visit: www.myfavouritemagazines.co.uk/terms

Subscribe

EXPERT ADVICE

PRO WAYS TO LAUNCH A PRODUCT

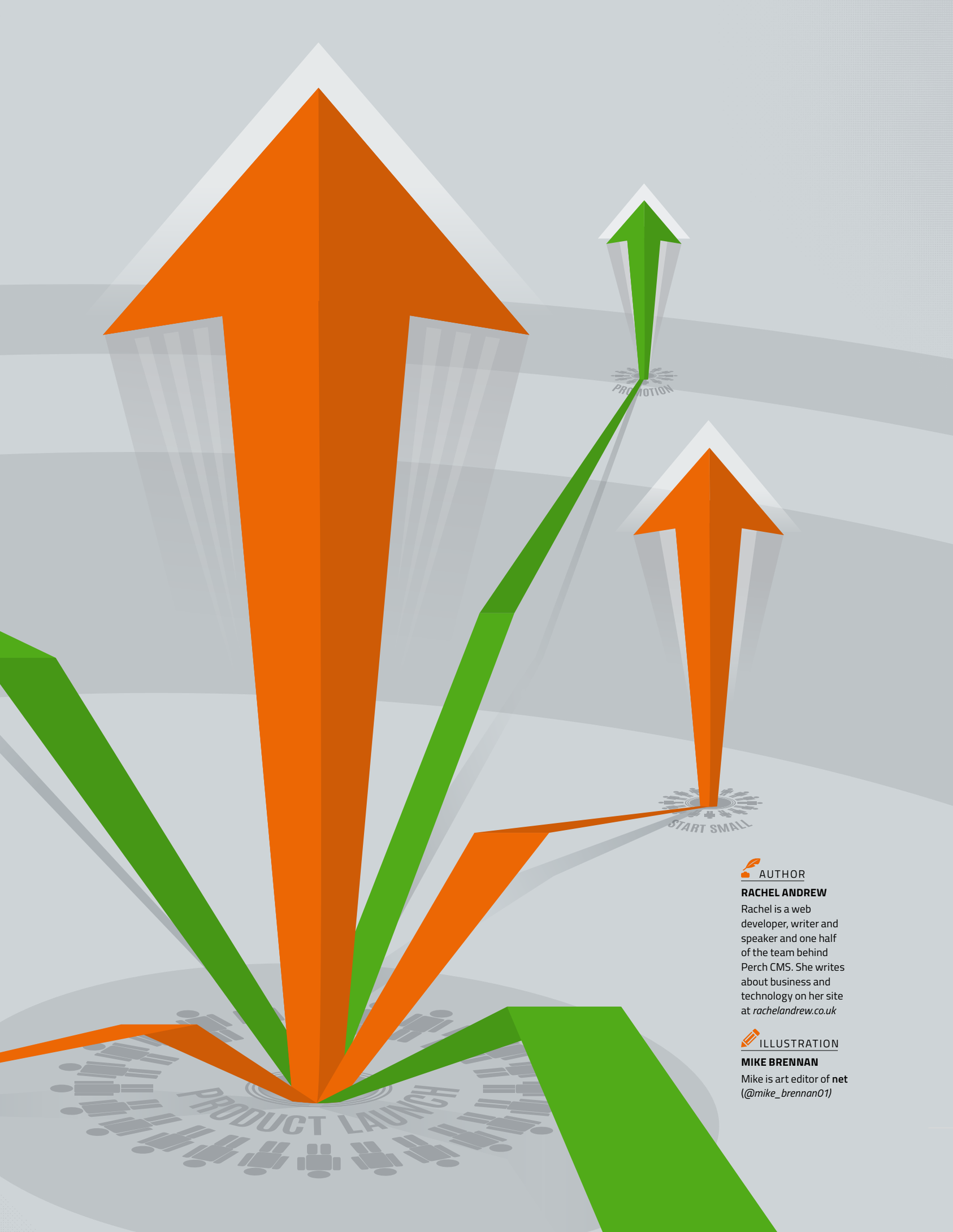
It's never been easier to bring digital products to market. **Rachel Andrew** shares her advice for launching your own products



Launching a product – either as a side project or with the aim to make it a complete business – is a goal of many web designers and developers. However, having a great idea and a well-implemented product is only part of the story. Before you can launch you'll need to have developed an entire business model around your idea; a model that will be attractive to your customer and also enable the product to grow. In addition, you'll need an infrastructure that can deliver the product and a way to provide support to you customers. My own company has transitioned over the last four years from services to products. In this article I'll describe some of the things I have learned along the way.

START SMALL

If I had one top tip for other people who want to launch a product it is to start small. Launch with the smallest possible thing that you can get people to pay you money for. The initial version of Perch (*grabaperch.com*) took us about four weekends to develop, and the supporting infrastructure about the same. A minimum product with a minimum amount of infrastructure means little risk and little cash spent up front. Perch was profitable within 24 hours. However, those up front costs that had to be covered before we were profitable were low. Once you get to profitability you can put money back into developing the product further.



AUTHOR

RACHEL ANDREW

Rachel is a web developer, writer and speaker and one half of the team behind Perch CMS. She writes about business and technology on her site at *rachelandrew.co.uk*

ILLUSTRATION

MIKE BRENNAN

Mike is art editor of **net** (*@mike_brennan01*)

CHOOSE A PRICING MODEL THAT WILL SUPPORT YOUR SUCCESS

There is a lot of noisy publicity around well-funded startups, that typically offer free plans to encourage mass adoption. If you're bootstrapping your product, then you need to start making money from the outset. Your pricing should not only be based on a gut feeling of what the market will accept, but also take into consideration what you will need to spend to sell and support your product. The recent example of photo site Everpix (*netm.ag/lie-251*), killed in part by an inability to pay an Amazon bill for the hosting of user photos, demonstrates the importance of ensuring that your profit margins will cover the services used to deliver your product.

For those developing products alongside freelance or client work, or a full time job, will your pricing model enable you to make your product your main source of income if it becomes successful? You should consider what happens if your product is a success. Having thousands of users who are all burning through your money isn't viable for anyone other than the funded startup.

PRODUCTS AIMED AT BUSINESS CUSTOMERS ARE OFTEN AN EASIER SELL

If you're currently at the ideas stage then looking for a product that targets the business-to-business market is generally far easier to make profitable than targeting consumers. In the consumer space, you have a lot of free products, a lot of 99p mobile apps. People are spending their own money and often not getting value from your product that they can place a cash value on. In the business space – and especially for larger businesses – people buying your product are often spending company money. Even if they're being careful with spending, if they're using your product to either make money or save time and company expense, then they can put a cash value on your offering.

COMPETING ON PRICE CAN WORK IF YOU TARGET A DIFFERENT MARKET TO MORE EXPENSIVE COMPETITORS

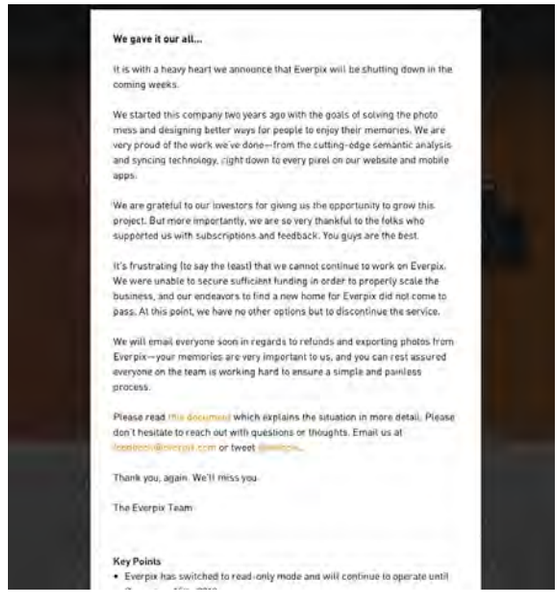
Competing on price can result in a race to the bottom, and if your competitors are well funded, they will win. There are places where charging less is appropriate. For example, in the scenario where you're offering a simpler version of a product; something designed for smaller business than your main competitor offers. In that case, you may be able to pick up some users who are priced out of the market currently.

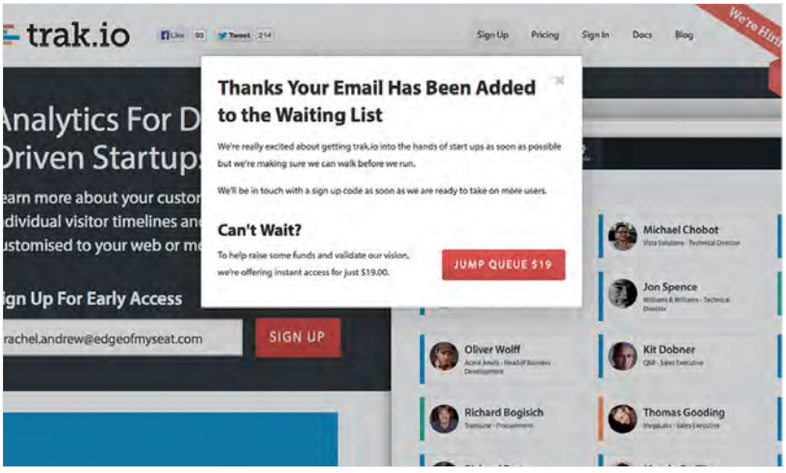
BE AWARE OF THE RESOURCE-HUNGRY FREE PLAN

If your pricing model does include a free plan, consider that your paying users need to bring in enough revenue to cover all the free users. In some situations, the free users aren't really costing you much in terms of resources and it may be worth offering a free plan to get people on board. You should limit these plans in the places where they do start to cost money, such as, in storage space or

Below left Scout provides a simpler monitoring platform than some other players and is also keenly priced to target a different market

Below right Photo startup Everpix failed to develop a model that could support their growing customer base, and had to shut down





Above New metrics startup trak.io employ a clever technique of asking for money up front to jump the beta list queue

bandwidth. The challenge here is coming up with something that offers enough value to the free user to encourage them to use the product and think about upgrading but doesn't leave you with a big overhead to support these users.

You should also consider the implications of supporting free users. If your product requires a good deal of setup, then you may find your support costs are rising faster than your revenue. This is one reason we don't offer a free trial licence for Perch. If we did, we would need to support every person who

> "If your product requires a good deal of setup then you may find your support costs are rising"

was just 'kicking the tyres'. Ultimately, this would raise the cost of the product for paying customers, and I don't think that's fair.

Unless you have a lot of money to throw at customer acquisition, take care that you're not trying to 'act like a startup'. The rules are different for the rest of us.

PAYING CUSTOMERS ARE WORTH MORE THAN JUST THE MONEY

It's important to get to people paying you as quickly as possible even if you can survive without the cash. Those real customers will give you far more useful feedback than an army of people who will

NATHAN BARRY ON THE IMPORTANCE OF BUILDING YOUR EMAIL LIST

Nathan Barry is the author of three self-published ebooks. His latest book, the second edition of *The App Design Handbook*, went on sale in November and revenue for that first 24 hours on sale was $36,297. Barry has shared detailed statistics of that 24 hour period at *nathanbarry.com/app-design-stats*, and they make interesting reading.

He writes, "Email marketing is the key to making an entire launch like this happen. It's especially important to focus on the sequence leading up to launch day, without that done right, none of this would have happened [...] it's not as easy as just sending an email saying, "go buy the book!" You have to build up anticipation, answer questions, and send reminders to get a launch of this scale. None of that would have been possible without building up an email list first."

It would be easy to think that having lots of Twitter followers would be the most important factor in a successful launch. Barry has a respectable 5,804 at the time of writing but that isn't a vast follower number for someone having such success.

He tells me, "An email list is the single most effective tool in any launch. I've found that an email subscriber is worth at least 15 times as much to your business as a Twitter follower. Email is not dead, it's the future of building profitable audiences online."

Launch success Thanks to his email list, Nathan Barry launched *The App Design Handbook* and made $36,297 in 24 hours

FINDING A LIKE-MINDED COMMUNITY

While bringing a new product to launch is a lot of fun, it can also be hard work and may require making a lot of decisions on subjects that are completely new to you. It can be really helpful to find other people who are going through the same issues, or who have already launched products and are willing to share their experiences and lend their advice.

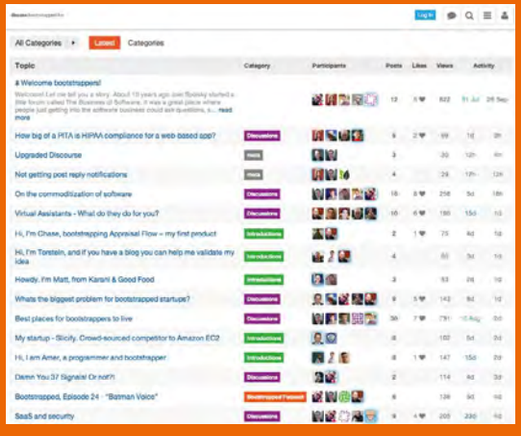
You may be able to find people in your existing network who can act as a sounding board and help you to work through difficult decisions. If not, there are also a number of online forums and networks that can help.

My personal favourite online community is the Bootstrapped Discussion Forum (*discuss.bootstrapped.fm*). This forum is a spin off from the Bootstrapped podcast hosted by Ian Landman and Andrey Butov, and is a friendly place where people who are involved in developing and launching their own products meet to ask questions and help each other out.

If you feel unsure about sharing too much business information on a public group, then many bootstrappers recommend creating 'mastermind groups'. This is simply a group of fellow entrepreneurs who meet in person or over Skype, taking turns to help each other make decisions or come up with new ideas.

Attending public forums or one of the many industry conferences can also be a great way to find people who you could approach to be part of such a group.

Ultimately, having people around you who can understand the challenges you're facing and celebrate your successes with you can make a huge difference to your ongoing motivation.



Community forums The Bootstrapped forums are a friendly place for those of us building products. Sign up, introduce yourself and join in

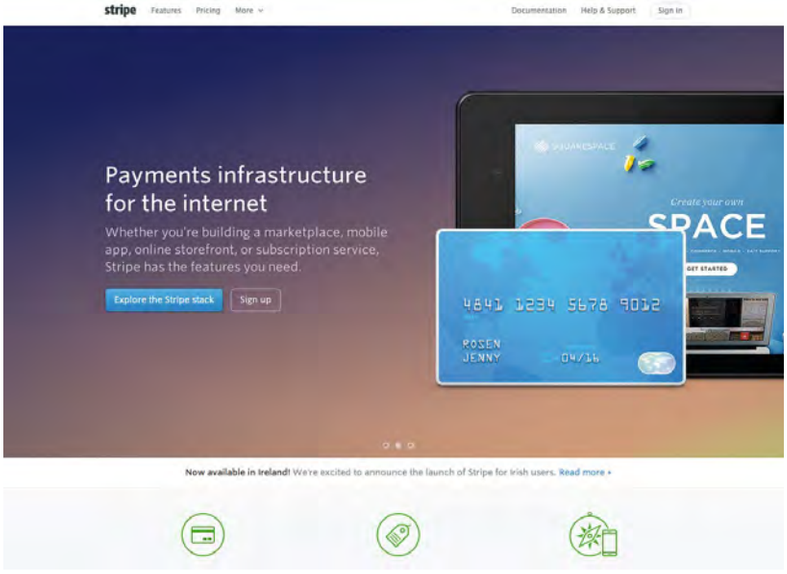
love your product as long as it is free. You want to be developing the features that are of interest to people who are willing to pay, and aligning your product more closely to that customer.

I recently signed up for the beta of a new metrics application called Trak.io (*trak.io*). After entering my email address, I got a message saying that if I wanted to bypass the queue I could do so by paying $19. This is a brilliant way to segment beta users based on who is willing to pay a small charge and who is not, and a way to find that out before the product is really mature enough to start asking for a monthly subscription from users.

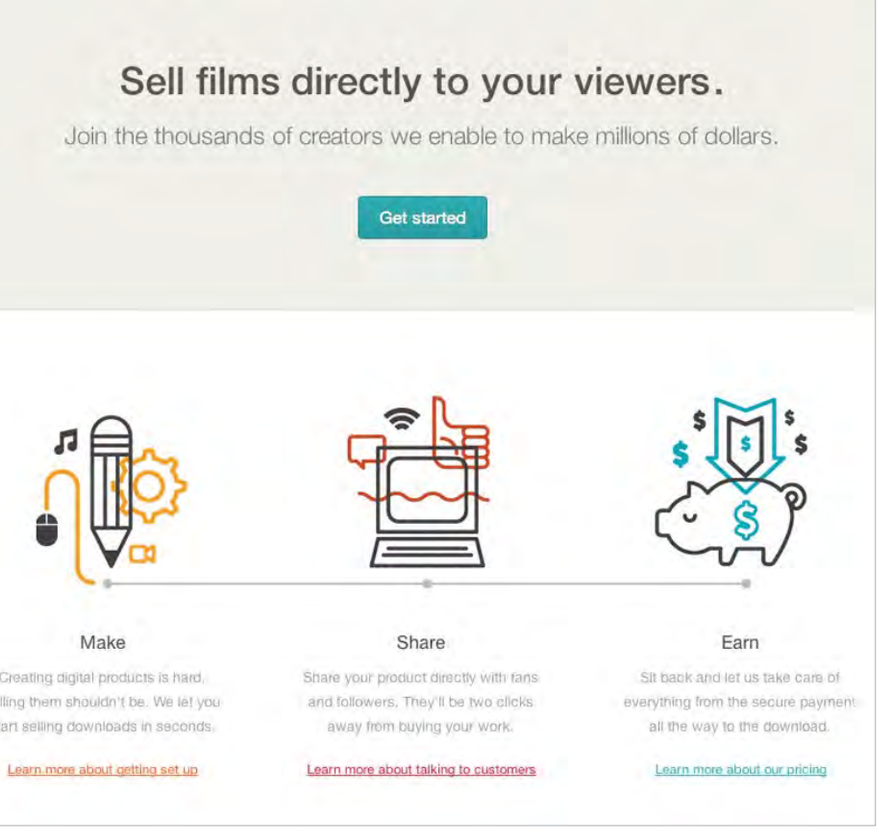
A MINIMUM VIABLE INFRASTRUCTURE

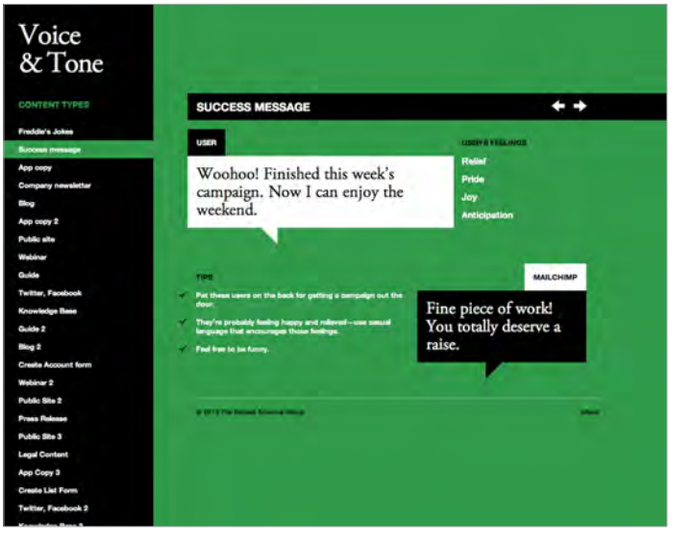
If you are at all familiar with the Lean Startup methodology, then you may have already come across the term 'Minimum Viable Product'. This is a similar concept to my advice at the beginning of this article, starting with the smallest thing that you can in order to test the market and your assumptions and then develop from there. You can do the same thing with the infrastructure around your product. Payment can be taken via PayPal (*www.paypal.com*) or a service such as Stripe (*www.stripe.com*), which is now available in the UK. Delivery of downloadable software can be managed via services such as Gumroad (*gumroad.com*). Your documentation and support site can be managed via any number of hosted applications.

It's likely you'll decide to bring some of these services in-house, allowing you to create your own delivery mechanisms, craft a more customised



Below Stripe is now available in the UK and Ireland and enables new companies to take payments without a merchant account





experience and save monthly fees. However, in order to get quickly in front of customers, these third party services can save you a lot of time.

OWN YOUR OWN CUSTOMERS AND ENSURE YOU CAN EXPORT DATA

While making full use of the services available to you, take care that you are able to keep your own list of customers and purchases or can export that data easily from the systems that you use. Try to avoid ending up in the situation where you are unable to contact your own customers, or lose purchase history or other information should you move to a different platform. This can be unavoidable if you need to sell via the Apple App Store or some other closed platform, but if you have a choice always ensure you have access to this data.

PREPARING FOR LAUNCH DAY

If you're currently working on your product, you need to also set time aside to start building an audience. A successful launch day will only happen if people are already excited about the product.

SET UP A HOLDING PAGE AND EMAIL SIGNUP

For Perch, we put a holding page online about a month prior to launch. We made a video showing the UI and basic functionality and added a signup box for people to express interest. Just over 500 people signed up and from that initial email to the list, which contained a special offer code, we had paid for everything that we had spent on Perch to date within 24 hours.

As soon as you can, get a holding page online for your product and start promoting it. Your holding page can be as simple as a form that adds people to an email list at MailChimp (*mailchimp.com*) or Campaign Monitor (*campaignmonitor.com*). Having an incentive for signing up can encourage people to give you their email address. For example, the offer of a launch day discount, or a free chapter in the case of an ebook.

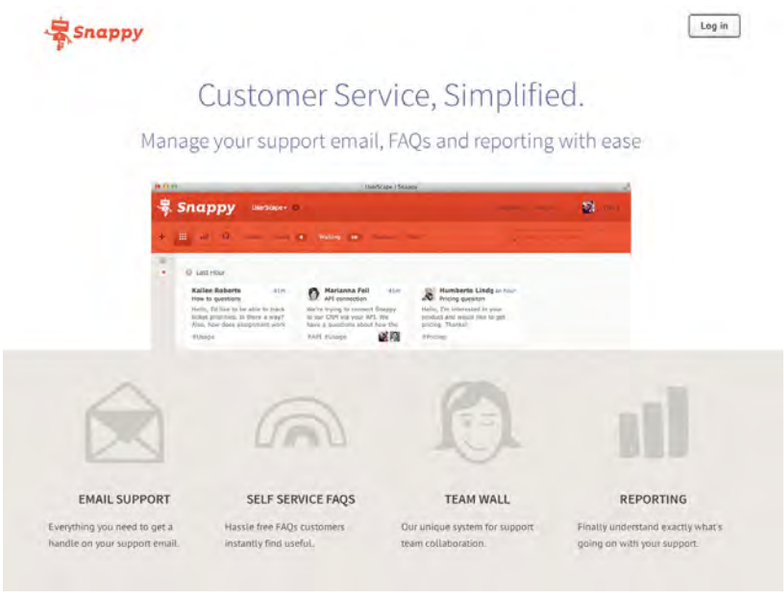
> "An incentive for signing up can encourage people to give you their email address"

CREATE CONTENT THAT'S OF INTEREST TO THE PEOPLE WHO WILL BUY YOUR PRODUCT

Before and after launch, if you can provide valuable information to the very same people who would buy your product or sign up for your service then it will help bring customers to you. If your product is aimed at web designers, write articles and blog posts that are of interest to web designers. They don't need to directly refer to the pain point your product solves, as long as they bring the ideal customers to your site ▶

Above left Gumroad is popular among ebook publishers as it provides a solution for payment and secure delivery of digital download files

Above right *voiceandtone.com* from MailChimp is a great example of how to maintain consistent messaging through your app and other materials



Above (left) Snappy is a hosted application built on top of email. It can help you manage support without a complicated helpdesk

Above (right) Sponsoring relevant podcasts has been our surprising marketing hit. Target shows that appeal to your ideal customers for best results

and causes them to be interested enough to learn about what you do.

This is where choosing to launch a product that ties in to your own industry or an interest you already have is key. It's easy for me to write articles that appeal to web designers as I've been writing for that market for over ten years, and working with designers as a developer for longer. It would be much harder had I created a product for a market I had no real knowledge of other than a problem I thought I could solve.

Content marketing is a key way to promote your products if you've got limited funds. Good content will not only index well in search engines but will also be shared by people if it really brings value in itself and isn't purely marketing. For writing ideas, look at the sort of questions people in your target market are asking on Twitter or Facebook. Simple posts answering a common problem or reviewing various solutions are a good place to start. Once you start collecting ideas, you'll find spotting potential topics becomes easier.

SUPPORTING YOUR PRODUCT

As you look towards the launch of your product, there is one area that I know from our own experience is highly important: customer support. Since the very early days of Perch, we've had as much positive feedback about the quality of our support as we do about the product itself. With a product aimed at web designers who are providing a service to their own clients, this is particularly important. Whatever your product and audience however, fast and helpful support can become your best marketing with people keen to share the excellent experience they had.

TOOLS FOR SUPPORT

Support via your own email inbox may seem like the path of least resistance at launch. However, it can soon become difficult to manage with no way to mark an issue as resolved. As soon as there is more than one person doing support a shared inbox can mean that some customers get answered

> **Content marketing is a key way to promote your products if you've got limited funds**

twice and others not at all. Therefore, I recommend implementing some kind of support system from the outset. One lightweight solution I frequently recommend is Snappy (*besnappy.com*). This hosted solution essentially creates a management layer on top of email, meaning that you and customers can still use email if that is easiest. However, Snappy gives you a way to mark issues as resolved, create standard replies and see statistics on queries incoming and answered.

Above Product businesses tend to bring in lots of small payments. *Xero.com* enables us to keep track by using the API

DEVELOPING YOUR PRODUCT BASED ON CUSTOMER FEEDBACK

Launching with the smallest possible thing means you will get a lot of feature requests after launch, and your support email or forums will be a great place to talk to customers and find out what they would most like to see. An added benefit of starting with a small product is that, instead of developing lots of features and hoping they will be useful to customers, you can develop your ongoing product roadmap, prioritising those features that will be useful to the largest group of users.

Using customer feedback can prevent you wasting your development time. At Perch, we can sometimes guess which feature requests will bubble up to the top of the list, but there have been many things that we would never have thought to develop back when we started. Had we waited to launch Perch, we could have developed a whole host of features that no one would want or use.

DON'T EXPECT IMMEDIATE RESULTS

If you're just starting out, I'm afraid I don't have a magic pill or any 'get rich quick' schemes to offer you. Launching a product takes time and hard work. While my own product has become successful enough that my company now purely focuses on that one product, it took four years of slowly moving away from client work to get here.

However, by setting yourself realistic expectations, putting in the hours, and learning as much as you possibly can from other experienced people, you can create and sell your own products.

RESOURCES

STARTUPS FOR THE REST OF US
A podcast hosted by Mike Taber and Rob Walling: *www.startupsfortherestofus.com*

SUPPORT OPS
A site and podcast with Chase Clemons: *supportops.co*

CUSTOMER SUPPORT FOR BOOTSTRAPPERS
A free ebook: *besnappy.com/books/customer-support-bootstrappers*

HOW TO LAUNCH ANYTHING
By Nathan Barry: *smashingmagazine.com/2013/06/28/how-to-launch-anything*

THE CASCADING TO-DO LIST OR HOW TO GET BIG THINGS DONE
By Brian Casel: *casjam.com/the-cascading-to-do-list-or-how-to-get-big-things-done*

PUTTING THE 'MIUM' IN FREEMIUM
By Dror Bren: *medium.com/on-startups/4656d8f5d866*

HOW WE TRIPLED REVENUE BY ADDING ONE BUTTON
By Michael Sacca: *medium.com/what-i-learned-building/a8e04b2d85fe*

7 LESSONS WE LEARNED GOING FROM ZERO TO $30K/MONTH IN UNDER A YEAR
By Alex Turnbull: *groovehq.com/blog/first-year*

PRICING UNDERPINS EVERYTHING YOU DO
By me (Rachel Andrew): *alistapart.com/column/pricing-underpins-everything-you-do*

CREATING AND SELLING YOUR OWN PRODUCTS
By Amy Hoy: *unicornfree.com/launch-roundtable*

STARTUPS, MARKETING AND GEEKERY
By Jason Cohen: *blog.asmartbear.com*

START SMALL, STAY SMALL
Rob Walling's book about developing product businesses: *www.startupbook.net*

START SMALL AND LISTEN
My article on Smashing Magazine details how we developed Perch from a tiny product: *smashingmagazine.com/2013/11/08/building-a-successful-product-start-small-and-listen*

AUTHORS

SAMUEL HULICK
Samuel (@SamuelHulick) is a UX designer and consultant, and the author of *The Elements of User Onboarding*. He critiques the onboarding experiences of popular web apps at *UserOnboard.com*

CREATE A KILLER FIRST IMPRESSION

If you've built an exceptional product, you'll want your signups to stick around to fully experience it. **Samuel Hulick** shows you how to design your product to improve your user onboarding

Now, I'm neither a psychic nor a gambling man, but without even looking at your product's user adoption metrics, I'd be willing to place a pretty significant wager that there's clear room for improvement. While product companies often invest heavily in marketing efforts that drive signups, and product efforts that drive ongoing use, it's rare to see similar investments made in ensuring the latter actually become the former.

This is a shame, as people tend to dislike change, and whenever they decide to sign up for a product, they're risking their time (and possibly money) on it being worthwhile in the long run. Being inattentive to your new signups' immediate needs right from the very beginning makes your product look like a really bad bet.

It's no wonder, then, that some products have been found to have up to 60 per cent of their new trial signups never even logging back in a second time (*netm.ag/signupstats-259*). Bleeding signups out of any part of your user adoption funnel can seriously impact your company's ability to grow, and your product's ability to flourish.

If boosting your company's growth rate, stretching your marketing dollars further and getting the most out of your product team's blood, sweat and tears are priorities for you, then exploring your options for better user onboarding is going to be very worth your while. Join me while I break down some of the top user interface patterns for turning curious visitors into highly-engaged users!

SETUP WIZARDS

When a visitor on your marketing site clicks the 'Sign up' button, they're taking a leap from the known (the site they've been on) into the unknown. What will things be like inside your product? How will they be greeted? How quickly can they get up and running?

Interestingly, many products answer those questions by not sending signups directly into the product at all, but instead opting to first drop them into a setup wizard. These are often stripped-down pages without much in the way of external navigation, or usual points of orientation like headers or footers. Much like Amazon's checkout page, they pare back everything that could distract the user from focusing on the task at hand:

CASE STUDY CONCIERGE ONBOARDING

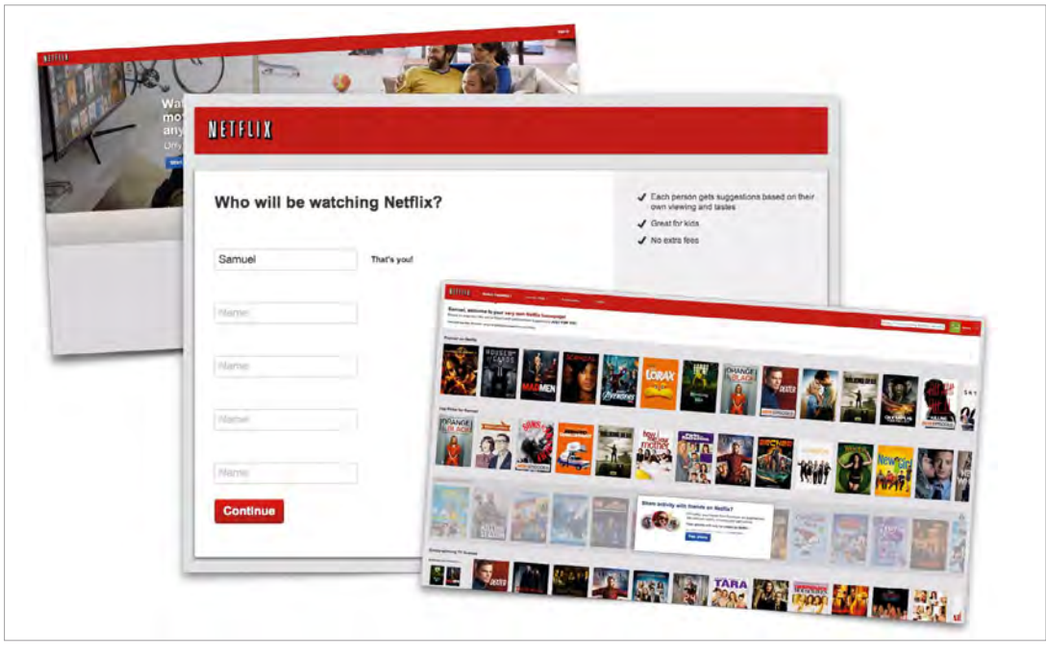
Want to significantly improve your trial-to-paid conversion rate without touching a line of code or signing up for outside services? A new trend in SaaS user adoption called concierge onboarding may be what you're looking for.

The premise is simple: instead of (or in addition to) providing a fully-automated onboarding experience within your product, you help get signups over any opening hurdles by personally assisting with their account setup.

Colin Nederkoorn, CEO of Customer.io (customer.io), ran an experimental concierge onboarding offering throughout July 2014, wherein half of his signups received a personal invitation for setup help, while the other half followed the product's normal onboarding flow. The users in the concierge batch converted at nearly twice the rate of the regular batch, and became customers more quickly overall. In fact, they signed up for more expensive plans, as well.

Nederkoorn attributes this effect to not only guiding new users to value more quickly, but establishing trust from the very beginning. This goes beyond selling products – it builds relationships. In that sense, it may even lead to longer retention rates, as well.

Taking this approach requires human effort, of course, but if your customer lifetime value (CLV) is in quadruple digits (e.g. $50 per month over 20 months), investing 15 minutes of labour in fast-tracking someone to success makes plenty of economic sense.



Disjointed setup Netflix has a setup wizard that only slightly resembles the public-facing homepage and the inner browsing experience

entering in required setup information before moving forward, often in the exact sequence they determine.

However, what these pages gain in focus, they lose in familiarity. Interstitial screens are neither the site where the user chose to take the leap, nor the product they thought they were leaping into. The longer you keep them hanging in limbo with 'in-between screens', the longer you stop the user from hitting the ground running.

One way to attain the focus without sacrificing the familiarity is to place setup wizard forms inside an overlay above the ensuing product, rather than in empty screens of their own. This way, the user can see exactly where they're headed from the very beginning, even if they have to work through some immediate business before getting there. I call it the 'Emerald City in the distance approach', after the ever-visible destination in The Wizard of Oz.

Pros

- Harnesses the user's full attention by removing all distractions
- Guarantees required information is entered, in the sequence you determine, before proceeding

Cons

- Creates a delay before landing the user into the 'real' interface
- Slows down the momentum that builds up whilst getting started
- Since actions are being taken outside the normal interface, it doesn't let the user 'map' where to go to change things in the future – making it a poor choice for kicking off high-frequency actions

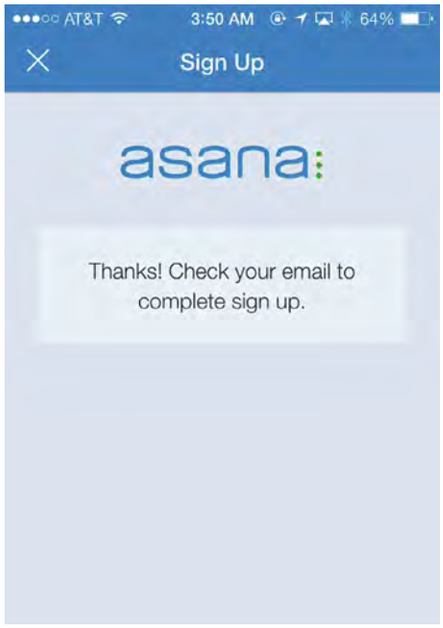
Recommendation

Use setup wizards sparingly. Focus on landing users into their new home as quickly as possible. Skew towards only asking for things you absolutely need to improve their immediate experience, and kick requests for optional information further down the line when you've earned more of their trust.

EMAIL CONFIRMATION

As soon as an account is fully set up, many products make the curious choice to throw up a wall and tell the new user to go away. Specifically, to go away and confirm their email address. This is a risky strategy, as it freezes the new user's momentum and sends them into the Pandora's box of distractions known as their inbox.

Product preview Quora's setup workflow takes place in an overlay that previews the product the user is about to enter



Address checking Asana requires new users to confirm their address before continuing in their native app experience

Is confirming their address necessary for a user to advance, or is this a pattern being followed blindly? While having a confirmed address on file may help you sleep easier at night, strongly consider at a minimum letting new users in on a probationary period, to at least get a sample of what your product provides.

Pros

- Confidence that the address on file can be used for secure actions (for example recovering a password)
- Reduced likelihood of spam or fraud accounts infiltrating your system

Cons

- Brings the momentum of the first-run experience to a screeching halt
- Increases the odds of some users leaving and never coming back

Recommendation

Defer, defer, defer. Keep the user's momentum going (and their attention intact) by letting them as far as humanly possible into your application before asking them to go away and come back later. Also, note that clicking *any* link in *any* follow-up email can confirm a user's address, so consider getting more creative than the typical 'You need to confirm your address' template.

PRODUCT TOURS

Even once new users have finally reached a product's actual interface, they still tend to be presented with another layer of intervention: tooltip tours, and their spiritual sister on mobile apps – intro screens. What these have in common is that they preempt the actual product experience with a quick showcase of what the product can offer. What they also have in common is the tendency to be skipped over.

LAND USERS IN THEIR NEW HOME AS QUICKLY AS POSSIBLE

Both kinds of tours are frequent victims of the 'next-next-next' phenomenon, in which impatient users breeze through a series of blurbs, preferring to get directly to the product itself (you've probably done this as a user – I know I have).

This becomes a big issue when the ensuing interface relies upon the user knowing the information provided in the introduction that led up to it. Ironically, when interfaces aren't reliant upon intro material, it renders that material somewhat irrelevant. Product tours can't seem to win either way.

Still, tours do have their place. They are great at pointing out interface elements that, for whatever reason, weren't designed to be prominently featured on their own. Tooltips work especially well when they're shown one at a time, to best guide the user's attention; and when they use that attention to encourage users to take a specific action – teaching by action over memorisation.

Pros

- Highlights most important elements of your interface (though your 'real' interface needs to do that, as well)
- Can serve as a stopgap for known points of confusion between redesigns

Cons

- Interrupts the current workflow to lead users by the ear through a predefined one
- Multi-step tooltip tours can often break down or encounter usability glitches

BEST IN CLASS

UserOnboard.com is a website dedicated to analysing popular web apps' signup experiences. It does this by creating detailed slideshows considering each app's process. Here's my hand-picked list of the best examples of each pattern covered here.

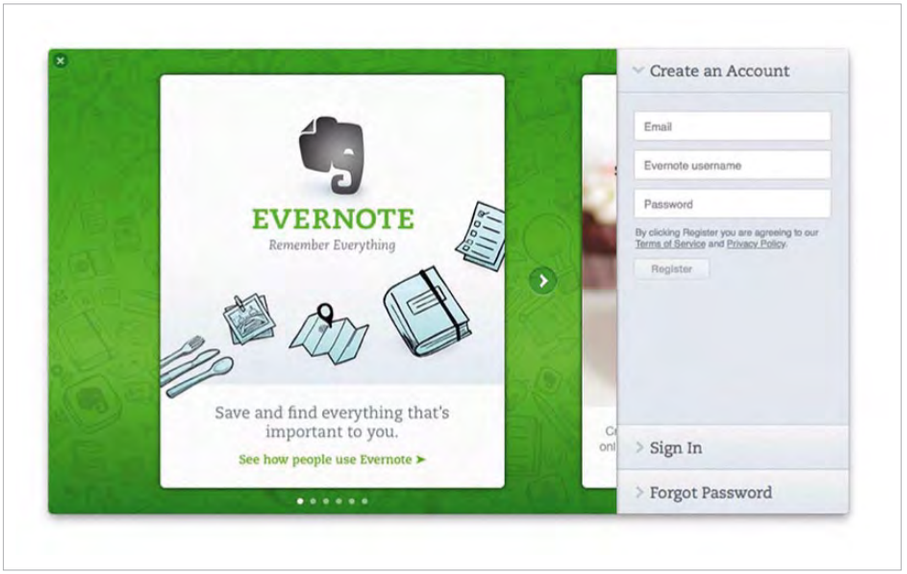
1. Setup wizard: InVision (*useronboard.com/invision*)
InVision makes great use of the 'Emerald City in the distance' technique in its setup flow. Notice how it uses this technique not just once, but twice – perhaps to ramp up the feeling on suspense.

2. Email confirmation: Tumblr (*useronboard.com/tumblr*)
Tumblr is a great example for what it *doesn't* do: namely, let something like an unconfirmed email prevent new users from exploring its product. Check out how minimal the nag banner is, when even shown at all.

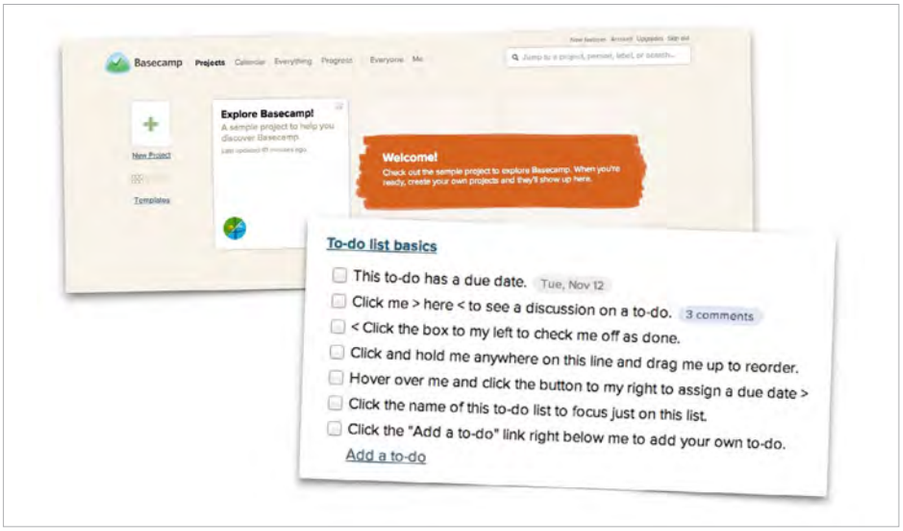
3. Product tour: Optimizely (*useronboard.com/optimizely*)
Optimizely is a website testing tool, and as you'd expect, its tooltip tour UX has been honed to perfection (it all kicks off on slide 40).

4. Blank states: Basecamp (*useronboard.com/basecamp*)
Basecamp is the gold standard in blank state design. Look at how it not only goes out of its way to welcome you, but also tees you up with a detailed example project. I particularly like the 'Do X with me' to-do list items.

5. Completion meter: Quora (*useronboard.com/quora*)
Quora nails many onboarding patterns at an eerily high level (really, it was a candidate for any of these five patterns), but its approach to quest lists is an example I bring up all the time. Pay close attention for the 'endowed progress effect' breakdown on slide 49.



Intro story Evernote's six-step intro story is unlikely to really be read by many of the signups it's shown to



Containers Basecamp has filled its empty containers with sample content, which in turn have their own sample content

Tracking progress Dropbox and LinkedIn serve as great examples of quest lists and progress trackers, respectively

- Layering one interface on top of another often leads to frontend maintenance issues

Recommendation

Use product tours as an exception, not as the rule. Propping up a poorly-performing UI with introductory crutches isn't an ideal or sustainable strategy. When you do use tours, it's recommended to still allow free exploration (i.e. don't force it to be followed), and to ensure it's easy to re-enter the tour after it has been dismissed.

BLANK STATES

With all the preceding steps out of the way, the user is finally able to explore the real-deal interface. What that interface will look like in its 'naked' state, though, will vary from one product to another.

Many product teams underestimate the importance of designing an interface for all its states: blank, filled, broken and everywhere in-between. This is especially evident when the very first screen a new user encounters is seemingly filled with nothing but empty containers. In blank states it's important to pay extra attention to copy. For example, social apps should be careful to not welcome their newest users with an off-tone 'You have no friends'.

Instead of admonishing new users for lacking content they couldn't possibly have produced yet, view this as a chance for your product to use itself to introduce itself. By filling an otherwise-empty container with supporting info or example content, blank states are a wonderful opportunity to acquaint the user with what they can expect to see once they've taken the necessary actions to fill them up. When blank states include a call-to-action, they can even single-handedly provide a boost in the right direction.

Lastly, don't forget to design your product to behave like a person would in a first-impression situation – a simple 'Welcome' can go a long way.

Pros

- Starts the first real activity within the product on a positive, helpful note
- Provides persistent, non-invasive context around which actions should be taken
- Acquaints the user with the exact process that they need to follow to repeat or change the current action (as opposed to actions taken inside a setup wizard, for example)

Cons

- Very likely to require engineering assistance if adding to an interface after it's 'fully baked' and live (which is not necessarily the case for third party tooltip software, for example)
- Can be difficult to squeeze effective blank states into very small (but important) UI elements

Recommendation

Use as fully and as comprehensively as possible. In fact, you could make the argument that designing for multiple usage states is simply a prerequisite for quality product design, full stop. I give this my highest recommendation.

IT'S VITAL TO PAY EXTRA ATTENTION TO COPY IN BLANK STATES

COMPLETION METER

Once the user is up and running inside the interface, let's make sure they actually get something significant accomplished. Quest lists and progress trackers are both excellent examples of UI patterns that serve the exact same purpose: chopping large, complex actions into smaller, more manageable tasks, and showing the user how close they are to completing them at every turn.

Quest lists provide the user with a persistent view of everything they've done and still have to do. They also create that oh-so-satisfying feeling of crossing something off a list. Progress trackers, on the other hand, typically use a percentage to represent completion and only show a single recommended next step.

Either way, once the completion meter is full, do both yourself and your new user a favour by calling out and celebrating their success. It's a great opportunity to provide them with timely encouragement and build affinity at the same time. After everything you've been through together, it makes sense!

Pros

- Makes goals more achievable by chopping them up into smaller, more consumable pieces
- Provides extra motivation by showing users their progress as they go
- Orients users around which follow-up actions are most important
- Provides ongoing guidance in a non-interruptive manner

Cons

- Requires engineering resources dedicated to something that doesn't directly facilitate action
- Will not ensure all steps get done, especially in the order outlined – as such, a bad fit for required or highly-sequential actions (see setup wizards)

Recommendation

Two big thumbs up for quest lists. They provide an excellent opportunity to get very serious about what you want people to do first, and track how well those actions correlate with a user's ultimate success and ongoing engagement. When complete, use success states to chain one major workflow to the next.

SUMMING UP

Of course, user onboarding is much more than a collection of interface patterns – it's the overall process of turning new signups into thriving, highly engaged users. In that sense, onboarding really touches on every department in the company, since it involves every point of the customer experience. That's quite an undertaking.

Still, by paying close attention to the first few interactions your signups have with your product, and crafting an experience to showcase what a wonderful fit the two of you will be, you'll both be well on your way to flourishing!

*REINVENT!

CREATE THE PERFECT REDESIGN

Dan Mall shares tips and experiences from industry pros on how to do a website redesign the right way

A redesign is a matter of course for websites. Software platform Hubspot's survey of hundreds of consumers, marketers and agencies found that 68 per cent completed a website redesign within the last 12 months. There are currently over 739 million existing websites (which increased 22 million compared to the previous month according to internet services firm Netcraft). So, if we apply Hubspot's sample data to the entire spectrum, over 500 million websites were redesigned in the last year. This leads us to a (scientific) conclusion: website redesigns are massively important.

However, we don't always treat them as such. Redesigns often suffer from poor planning and preparation, and lack clear project goals. For instance, Hubspot reveals 37 per cent of those surveyed didn't measure any metrics for their last website redesign. So how do you know whether to redesign or realign your existing site? And how do you get it right?

Author, entrepreneur and speaker Seth Godin says on his blog (*netm.ag/godin-248*), "The only reason to build a website is to change someone. If you can't tell me the change and you can't tell me the someone, then you're wasting your time." He elaborates, "The web is a direct marketing medium, something that can be measured and a tool that works best when the person who builds the page has a point of view. Instead of a committee deciding everything that ought to be on the page, and compromising at every step, an effective website is created by someone who knows what they want the user to do."

REDESIGN AND REALIGN

In 2005, designer, speaker and author Cameron Moll wrote a wonderful article for web design magazine *A List Apart* called 'Good Designers Redesign, Great Designers Realign' (*netm.ag/realign-248*). In the article, Moll presents a simple framework: "The desire to redesign is aesthetic-driven, while the desire to realign is purpose-driven."

Almost a decade later and the sands have shifted. While there are still a lot of aesthetic-driven redesigns, the definitions for 'redesign' and 'realign' are converging. More and more of us are able to champion balanced and purpose-driven evaluations as well as the aesthetics of our web presences. ▶

AUTHOR

DAN MALL

Dan is an art director, designer and advisor from Philadelphia. He's founder and design director at agency SuperFriendly (*superfriend.ly*)

ILLUSTRATION

ANDY SMITH

Andy studied illustration at the Royal College of Art, London. Based by the sea in Hastings, his clients include Nike, Sony and Mercedes (*asmithillustration.com*)

TRENT WALTON: WORKING CLOSELY WITH CLIENTS

Collaboration between all parties is a huge part of successful redesigns. I always tell my clients that we'll achieve the best results by working together because they have institutional knowledge and we have industry knowledge. It's the combination of the two that ensures a unique and appropriate site.

The best thing you can do is embed yourself with the client as quickly as possible. There are going to be a million things that work and a million things that don't, and a million things that they've already tried. We have to talk to them and listen to them because they have a wealth of knowledge that we don't have.

[For the Microsoft redesign] we focused on the things we knew and the things we believed in, especially in making sensible decisions around multiple devices, responsive layouts, what's going to add technical debt, what's going to add a lot of kilobytes to load, and all those kinds of things. We have our opinions about design and we brought those, but we never put our feet down and stomped around until they agreed with us. It was more like, 'We're bringing a recommendation; you can go with it or not.'

Whether large or small, you have to have a team that's willing to work with you on a daily basis, or something close to that. The idea of 'we're gonna go solve all your problems and come back with some comps' just doesn't work. What's probably going to happen is that you'll come back and they'll say, 'Yeah, we tried that eighteen months ago and it didn't work, so you wasted three months of our time and thousands of our dollars'. Alleviate any risk or stress by asking any question that comes to your mind.

Skeptical? I sat down with a few friends to ask them about some of their recent redesign experiences. Here are their stories.

A COMPLETE OVERHAUL

Last year, web design and branding shop Paravel (*paravelinc.com*) helped Microsoft conceptualise, create and launch a new homepage.

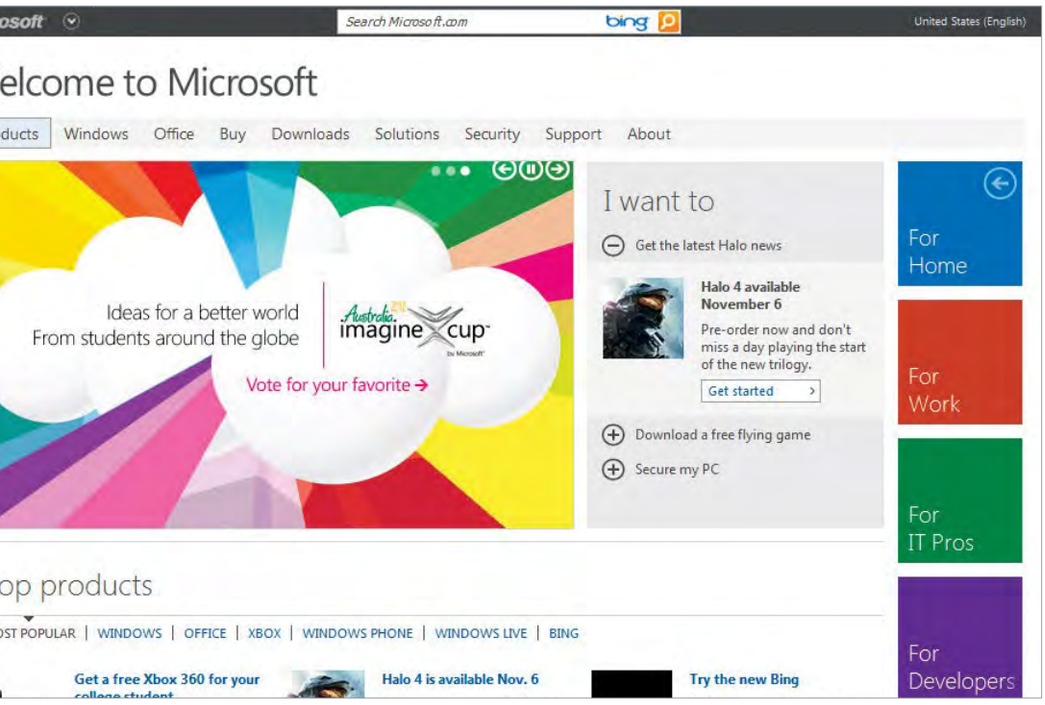
Former Microsoft evangelist and strategist Nishant Kothary had worked with Paravel on a handful of small projects in the past, and, when the opportunity came up, he knew where to turn. "In my early discussions with the team, I'd outlined a model that involved bringing in some outside talent and integrating them carefully into the Microsoft team. I was certain that they had the right skill set, but more importantly, the right attitudes," he says.

So far, Microsoft had a few great ingredients for a great redesign: the right people internally and externally, the right circumstances and the right amount of luck to bring them together. What now? How did they decide what to do?

Here's the prior version of Microsoft.com before the Paravel redesign: *netm.ag/oldms-248*. Here's the new one that exists today: *netm.ag/newms-248*. It would've been possible to take the old homepage and tweak it to achieve something similar to the new version. So why start from scratch?

"Obviously, Microsoft has a lot of things happening all at once," says Trent Walton, founder and one-third of Paravel. "They needed something that could be expanded to feature multiple pieces simultaneously. We needed to build a system that could allow them to talk about Xbox as well as new phones and computers and software at the same time. They kinda had that before, but they wanted to do a better job of it. Starting from scratch allowed us to give them a better system."

Better system The homepage that Paravel designed and built from scratch for Microsoft provided a better system



Above left The old site: Microsoft hired Paravel to redesign its homepage to provide a platform to showcase its products

Above right The pieces of the design system Paravel created for Microsoft's new site

The goal of the redesign was to improve the site's aesthetics and purpose, but it would have been incredibly difficult to realign the site because the system wasn't yet in place for that to be possible. The real magic of this project was that, together, Paravel and Microsoft created something both aspirational and practical. Perhaps Kothary puts it best: "The team built a bleeding-edge site in the real world."

> We probably could have optimised the old site but a polished turd is still a turd

Read more about the Microsoft redesign in Walton's post, 'A New Microsoft.com' (*netm.ag/mstrent-248*) and in Paravel's case study of the project (*netm.ag/params-248*).

CONSTANT IMPROVEMENT

Like Microsoft, the Four Seasons group of hotels and resorts completely redesigned its website fourseasons.com (*www.fourseasons.com*) rather than attempting to improve its existing site.

"We probably could have optimised the old site, but a polished turd is still a turd," says Chris Cocca, global ecommerce manager for Four Seasons Hotels and Resorts. "As much as it is a website, it's also a platform for us to extend beyond. It wasn't just about what you see on the frontend; we were really designing a platform underneath to influence and react to the ways people interact with travel brands. [The redesign] was bigger than just the site," he adds.

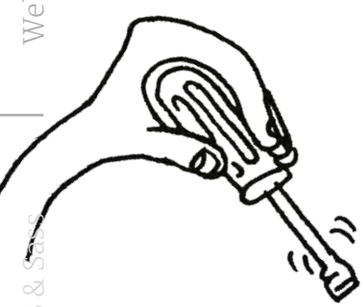
So why start from scratch with a redesign rather than realigning the previous site? "We looked at where we were when we started and realised that the systems we had in place — the design, the content, all the pieces — weren't going to be enough to even tweak," adds Cocca. "We were far enough behind that we needed to redesign first and then move into that realignment phase. We knew we weren't going to nail it in the first pass, but we knew we needed to make significant gains. We needed a new baseline."

If this baseline doesn't exist, perhaps a redesign the perfect opportunity to create it. Maybe, the point of a big redesign is to enable you to realign.

Cocca sums it up nicely: "You have to set up a budget beyond the project to maintain and improve the site. If you don't, that's how you get into situations where you need to do a massive, pain-in-the-butt redesign every five years because everything's outdated and nothing is right. My whole mission is that if we do another redesign while I'm still here, then I haven't done my job properly."

A SOLID BASELINE

Even where a project demands constant redesigning, the aim seems to be in creating a solid baseline. For the past three years, web design, development and



Unique identity The O Music Awards needed a unique visual identity every year

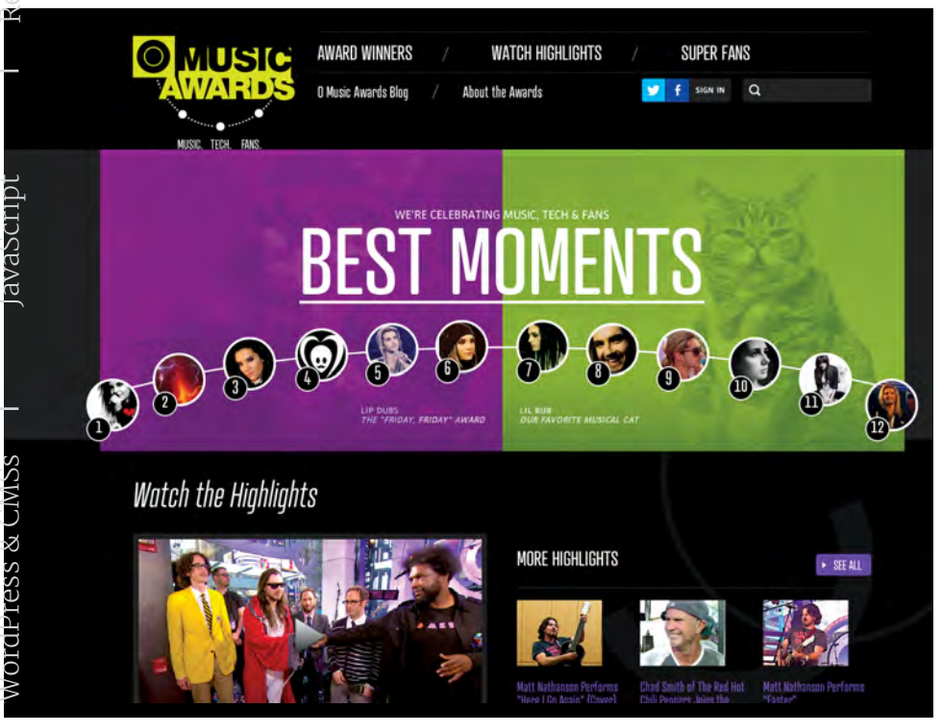
user experience consultancy Happy Cog (*happycog.com*) partnered with entertainment network MTV to create an online presence for the annual O Music Awards (OMA) (*www.omusicawards.com*), an event that celebrates the best in digital music.

When the awards show first started, each year was intentionally unique in its visually identity, forcing the Happy Cog team to frequently start from scratch. Yesenia Perez-Cruz, senior interactive designer at Happy Cog recalls Happy Cog's kickoff meeting for the most recent OMA started with questions: 'What works here and what doesn't?'

Despite the constant redesigns, the team worked towards building more consistency. The new version now contains lots of realignment in the voting and sharing experiences. "A lot of our work with the most current [version] was focused on setting up a solid foundation to build on top of. We'd be in a much better place to make subtle changes and enhancements and, honestly, just do things that we wanted to try and do," says Perez-Cruz.

WHEN PLATFORMS EXIST

It's easier to make incremental changes as you go along once you've built a suitable platform to work on. Modern companies like social network Facebook are often optimised for iteration and have existing platforms that support experimentation and provide ways to adapt quickly.



One example of this is the new Graph Search (*facebook.com/about/graphsearch*) feature of Facebook. As Fast Company Design reported (*netm.ag/fastcode-248*), the initial interface that Facebook launched was a few steps different than the one live on the site today. In fact, there were about four or five different iterations that went live before the current one stuck.

"We ended up with this design that was just white text on the Facebook blue. Very simple, very reductionist," says Russ Maschmeyer, lead designer on the product. "We were really excited to launch it; we have lots of early adopters, and not too many people had trouble finding or interacting with it. But, over time, as we released it to more and more people, we found that a lot of people were really confused by it. A lot of people complained that they couldn't find the search field. And, for those who found it, it wasn't really clear what you could do with it. We realised it was a significant problem.

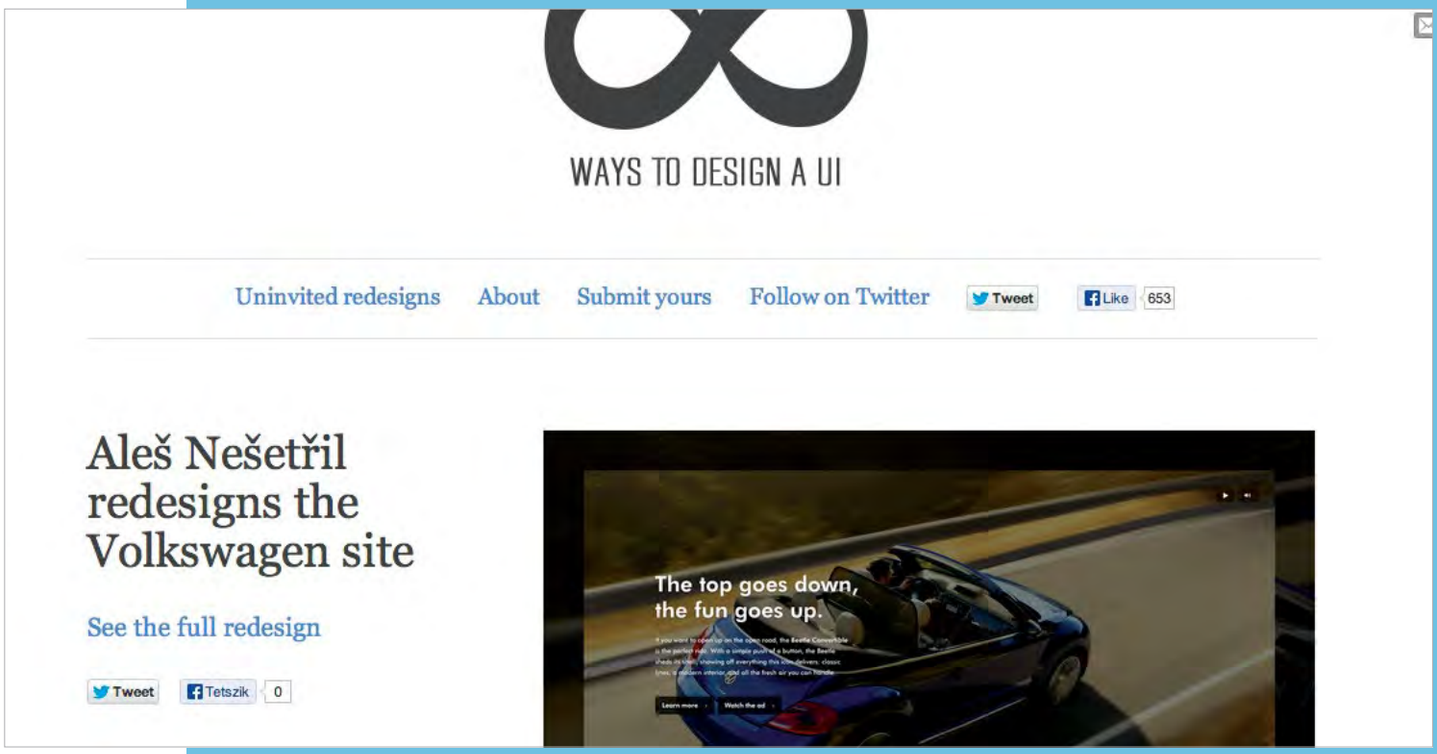
"This elegant design solution that we were really excited about was actually making it really difficult for people to use it effectively or get any value out of the new product.

"So we started with baby steps. We added a slightly darker rectangle behind the text. We tried changing the search prompt text. We didn't really want to give up on the idea because we were excited about it, but we ran all of these tests side by side. We also ran the extreme option with the drop-dead, basic, white-field search box.

"At the end of the day, the white field performed far and away better for our users than any of the incremental steps back. I was pretty crestfallen since I was so excited about the initial design direction, but in the end, I realised that changing the design of a search field actually didn't add any value to the product (aside from pure aesthetics). It didn't make the product more valuable; it certainly didn't make it easier for people to use. Our job as designers isn't only to create beautiful products, but also to make products usable and powerful. Graph search is really powerful … when you can actually find the search box," Maschmeyer laughs.

But why not completely redesign Graph Search when it wasn't performing well, as opposed to incrementally optimising?

"We actually shipped the crazy, bold, new design approach, rather than trying to move towards it. We had no problem with wanting to see how it played and then testing it against other things. That's the thing that makes me most excited to be at Facebook. With products of such size and scale, you almost never know how people are going to react to big design changes.



UNINVITED REDESIGNS

For better or worse, uncommissioned, unsolicited and uninvited redesigns have become somewhat of a fad. There's even a whole site dedicated to showcasing them, called Uninvited Designs (*uninvitedredesigns.com*).

In some cases, they can lead to great things. Designer and student Andrew Kim posted his own rebranding of Microsoft (*netm.ag/minimal-248*) – and Microsoft hired him (*netm.ag/hired-248*). In other cases, they can come off as shallow and cosmetic – and poorly reflect the designer who created them.

What role do uninvited redesigns play in our industry? "They have zero context into the business goals, user needs and strategic plan," says Josh Brewer, former principal designer at Twitter. "You're essentially looking at the surface and saying, 'I would paint this painting differently.' It's a great creative exercise and I'm not knocking anyone that has done this. There are some great third-party Twitter apps that have done good, interesting takes on the core Twitter experience, and they meet the needs of a certain, small group of people. But there's all this context that's nonexistent in the approach and solutions they're proposing."

Designer and author Andy Rutledge has been creating unsolicited redesigns – which he instead calls 'Redux' – for years, at least publicly since 2005. "It's mostly a fun exercise for me. Traditionally, I've done it because I've spent most of my design time over the last few years trying to offer some sort of instructional material.

So, they're not just, 'Hey, let's redesign XYZ.com'. It's more like, 'Something's wrong with this; let's see how it might be done better, and why'. It also helps me exercise my chops on things I don't get to design that much."

> "I'm highly critical in my exercises, but the purpose is to affect something positive"

Rather than redesigning a particular site, Rutledge prefers to address the category. "My [news redux] (*netm.ag/redux-248*) didn't take on *The New York Times* website as its subject; it took on news websites as its subject. For example, I removed many of the ads, but, in the article, I accounted for why it was necessary, beneficial and a positive step for the publication.

"I'm highly critical in my exercises, but the purpose is to affect something positive," he says. "The most common transgression is that it's just a redesign that never mentions what problems are being addressed or what issues are being tackled other than aesthetic preference. Unless there are specific constraints, design has no purpose, which is to say that redesign has no purpose. Even if it's a fun exercise, those constraints have to be acknowledged."

Above The new Four Seasons Hotel designed its new site to provide a new backend, too

Below Chris Coyier decided to redesign his site CSS-Tricks in public and raised funds via a Kickstarter campaign

"The best way to understand what's happening is to isolate the elements of a design change and understand how each of them contributes to a design direction," says Maschmeyer.

Maschmeyer outlines the value of having each version out there in the open so that they that could learn from how people used each one. But what happens when the entire process is done in the open?

REDESIGNING IN PUBLIC

Designer and writer Chris Coyier wrote about this idea of 'Working in Public' (*netm.ag/public-248*) and put it to the test by redesigning his blog CSS-Tricks (*css-tricks.com*) in this very way.

Coyier also wanted to raise some funds to be able to screencast the entire process. He launched a meager Kickstarter campaign (*netm.ag/redesignkick-248*) to raise $3,500 and ended up raising just shy of $90,000.

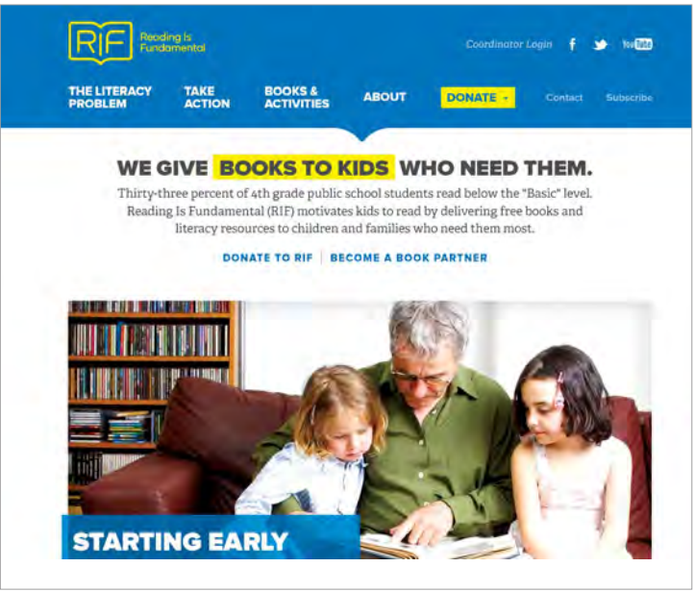
"I called it v10," he says, "which is to say that this isn't the first time I've redesigned my site. Sometimes I redesigned because I felt like it. I don't know that you always need a super good reason all the time. It's just what I do anyways. Some people collect bottle caps. That said, you usually [do need a reason] when there's money on the line.

"This was the first time CSS-Tricks was redesigned with a real business case. It happened during a time when I quit my job, so I wanted to turn it from a little side project and hobby into a real thing. So the whole redesign had that looming over it.

"I was going to do all of this work – rethinking advertising, deciding on a new aesthetics – and I realised this was the stuff I already write about. People come to the site for that stuff anyway, so why do that in a vacuum? I was gonna make all of this new content, so it was a no-brainer. I decided I might as well flip the switch on. I've done hundreds of screencasts, so it felt natural. I could just talk it out while doing it."

Certainly, working in public adds an extra layer of complexity, everything from the time it takes to document articulately to the risk of people publicly criticising your hard work. "I tried to be super objective about it," Coyier explains. "That's the key to beating haters and trolls, and even my own self-doubt. If I can be as objective as I can, then I win.

"Does the site make more money than before? If yes, then win! Do people spend more time on the site? Does the site load faster? Those are all objective, hard numbers that I can choose to make better.



RESOURCES

Everyone's redesign experiences are unique. Here are some tips:

'Stop Redesigning And Start Tuning Your Site Instead', *Smashing Magazine* (*netm.ag/tuning-248*)
'10 Terrible Reasons to Redesign Your Website', Hubspot (*netm.ag/terrible-248*)
'Clear Indications That It's Time To Redesign', *Smashing Magazine* (*netm.ag/indications-248*)
'10 points to consider when redesigning a website', *Creative Bloq* (*netm.ag/points-248*)
'How To Successfully Redesign Your Website', *Forbes* (*netm.ag/success-248*)
'10 Tips to a Successful Website Redesign', University Business (*netm.ag/tips-248*)
'How to Properly Redesign a Website', *1st Web Designer* (*netm.ag/properly-248*)

"Every objective I could measure: that was the goal. In the end, if someone said, 'I hate the new redesign', I could say, 'I don't care!'"

Frontend designer Brad Frost and designer Melissa Frost also decided to redesign in public when redesigning a website for the Greater Pittsburgh Community Food Bank (*pittsburghfoodbank.org*).

> Communicating your thoughts to a separate audience gives you a balanced perspective

In a blog post, Frost breaks down the components of what it means to design in public (*netm.ag/bradopen-248*). He concludes, "It took me a while to realise that it's not about the work that I do, but rather what that work enables others to do. By sharing your thoughts, your successes, your failures, your techniques, your process, your resources, you're able to have a much greater impact on the world than whatever happens to be in your project's scope."

A PUBLIC REDESIGN

Last year, my company SuperFriendly (*superfriend.ly*) was fortunate to do some work with non-profit organisation, Reading Is Fundamental (*rif.org*). Its mission is simple: to give books to kids who need them. We proposed that we design the site in public, and the client was probably more excited about it than we were. The team wrote posts along the way about the discovery and design process, starting with a kickoff meeting (*netm.ag/rifkick-248*), and we made our password-protected client landing page (*rif.superfriend.ly*) publicly accessible to anyone to follow along. It kept me sharp. The accountability I had created with both the client and the public made me think more deeply about the project than I probably would have otherwise. When I'm working on a project, I'm often way too close to it to be able to think objectively. Strangely and wonderfully, communicating your thoughts to a separate audience at a higher level gives you a more balanced perspective on what you're doing and how you're handling things. It's actually pretty amazing.

HOW DO YOU BEGIN?

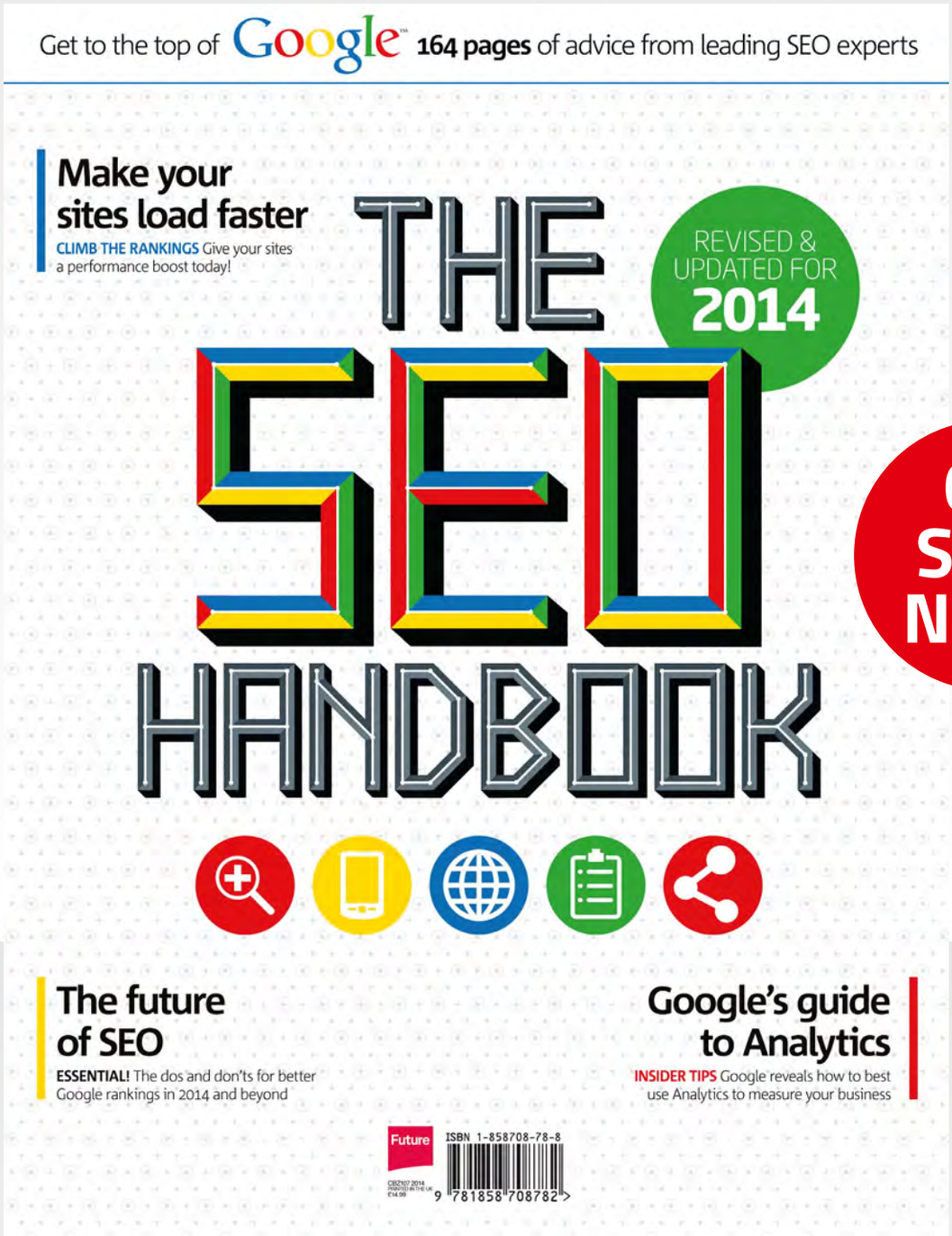
I've shared some amazing stories of how incredibly smart people approached redesigns. How do you go about redesigns? Surprisingly, everyone I talked to says just about the same thing: start small. Here's some advice: in her wonderful book, *Bird by Bird: Some Instructions about Writing and Life*, novelist Anne Lamott tells a story about her family. When her brother was ten-years-old, he procrastinated on writing a paper about ornithology until the last possible day. In desperation, he sat at the table, close to tears due to the daunting task ahead of him. His father put his arm around him and quietly said, "Bird by bird, buddy. Just take it bird by bird."

Story time Agency SuperFriendly redesigned a site in the open for Reading Is Fundamental (RIF)

SLIDEDECK

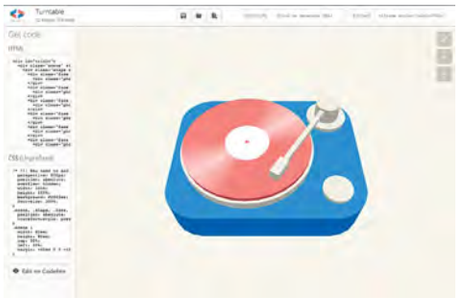
REDESIGN SCIENCE

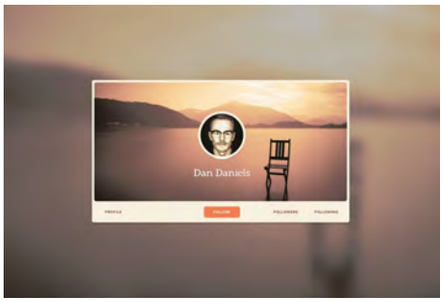
See Hubspot's presentation on 'The Science of Website Redesign': *netm.ag/1rijq1H*

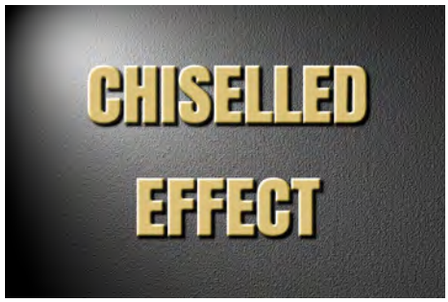


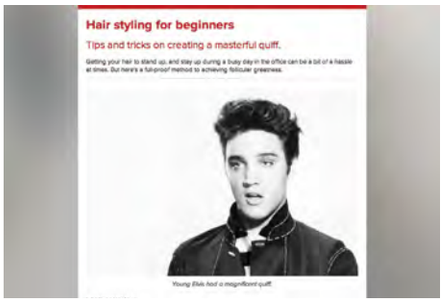
In 164 packed pages, we show you how to get to the top of Google. From making your site load faster and climbing the rankings to the latest SEO tricks and techniques, this handbook contains everything you need to know to become a master of SEO. You'll also learn all about Analytics – from a wide range of industry experts and Google itself – and key marketing techniques to drive your site or business to the top of the world's favourite search engine.

CSS & SASS










25 smart
CSS
tips
CSS TIPS
CSS TIPS

CSS tips

At some point with any technology, there's a feeling that pretty much anything that can be done has been done. When something is no longer quite as new and shiny, interest wanes and attention is drawn to the next big thing. This has often been the case in the web industry, which is prone to getting terribly excited by a certain aspect of technology before, sooner or later, relegating it to the mundane.

When CSS first appeared, it was revolutionary, and over time it has evolved to enable designers to create flexible, tightly-crafted and beautiful web page layouts. Of late, though, there's been the suggestion from various quarters that CSS is tired and perhaps its time in the sun has gone.

I want to showcase that there's still plenty of excitement and life in the world of CSS, whether that's in cutting-edge properties you may not yet have explored, or through using an aspect of CSS in a way you'd not previously considered.

Here are some tips from some of the industry's top CSS experts.

(Note: Some techniques in this feature are cutting-edge and may not be fully supported across all browsers. Thoroughly test and ensure fallbacks are in place before making any work live.)

01 MATCH IMAGES TO SITE COLOUR SCHEMES

Christopher Schmitt, conference organiser

Conferences have their own colour schemes and, with many speakers, workflow for managing portraits can be complex. Manually applying filters doesn't scale and relies on you having access to, say, a specific Photoshop action. I now use high-res greyscale PNGs and add tones using CSS filters. This enables me to match any portrait to an event's scheme, and to also reuse images across multiple themes. I just need a new CSS rule for each.

See a demo: *cdpn.io/v/pDLzl*

02 EVENLY SHARE SPACE IN A GRID'S LAST ROW

Stephen Hay, designer and author

If you've an unknown number of items to be displayed in a grid, you can use Flexbox to split the last row evenly. So if there's only one item, it will take up the entire row; if there are two items, the row will be split in half, and so on.

See a demo: *jsbin.com/EtUJUVA/1*

AUTHOR

CRAIG GRANNELL

Craig (@craiggrannell) is a writer, designer and musician. He's been writing for **net** since before 'border-radius' was new and exciting

ILLUSTRATION

MIKE CHIPPERFIELD

Mike is co-founder and creative director of Brighton-based collective Magictorch (*magictorch.com*)

SHANE MIELKE
Freelance designer
shanemielke.com

Previously of 2Advanced, freelance designer Shane Mielke jokes, "I learned in an age of web design where if you weren't animating everything and anything, you weren't doing your job, and so that's always been a part of my mentality." Once, such animated content would have been a Flash-fest, but Mielke's latest site's animation is based around CSS and JavaScript. Project links are atop rotating cuboids, while press items become magazines flicking open on a rollover; elsewhere, screengrabs subtly shift in 3D space.

Mielke explains web design has "always been about figuring out something cool you want to do and how to accomplish that," and believes CSS is well-suited to the type of visually dynamic design he creates: "I'm a big fan of things initially appearing to be very simple. Then, when you interact with them, you see a level of depth, intricacy and love that brings everything to life. CSS is perfect for this, because if something doesn't work in an older browser or on certain platforms, everything rolls back and the effects don't show."

Mielke emphasises the importance of structure. Many of his pages are unordered lists manipulated with JavaScript and CSS, so there's no complication regarding positioning and floats; he also notes how CSS can "add extra spice, such as tweens when selecting items", yet people rarely tap into this. Bloat, though, must be avoided: "It's all about the developer. You can just as easily make a huge page with web standards as with Flash, when striving for certain effects."

03 CREATE PARTICLE ANIMATIONS WITH BOX-SHADOW

Ana Tudor, coder and maths fanatic

By mixing `box-shadow` with some maths and Sass, you can graph 2D curves, emulate 3D motion and create crazy particle animations everyone's going to mistake for canvas ones!

See *a* demo*: codepen.io/thebabydino/pen/paAJw* and *codepen.io/thebabydino/pen/shtGe*

04 ANIMATE POLYHEDRA WITH TRANSFORMS

Ana Tudor, coder and maths fanatic

You've probably seen pure CSS polygons created with borders, but we've a much more powerful tool in the `transform` property. Chaining and applying transforms on nested elements allows us to create complex polygons with image backgrounds or borders and transparent interiors. Using 3D transforms, we can combine these 2D shapes into polyhedra and make the solids merge, unfold, explode or recombine in a manner easily mistaken for WebGL.

See a demo: ***codepen.io/collection/eErLu***

05 MASTER 'CALC()' FOR POSITIONING

Ana Tudor, coder and maths fanatic

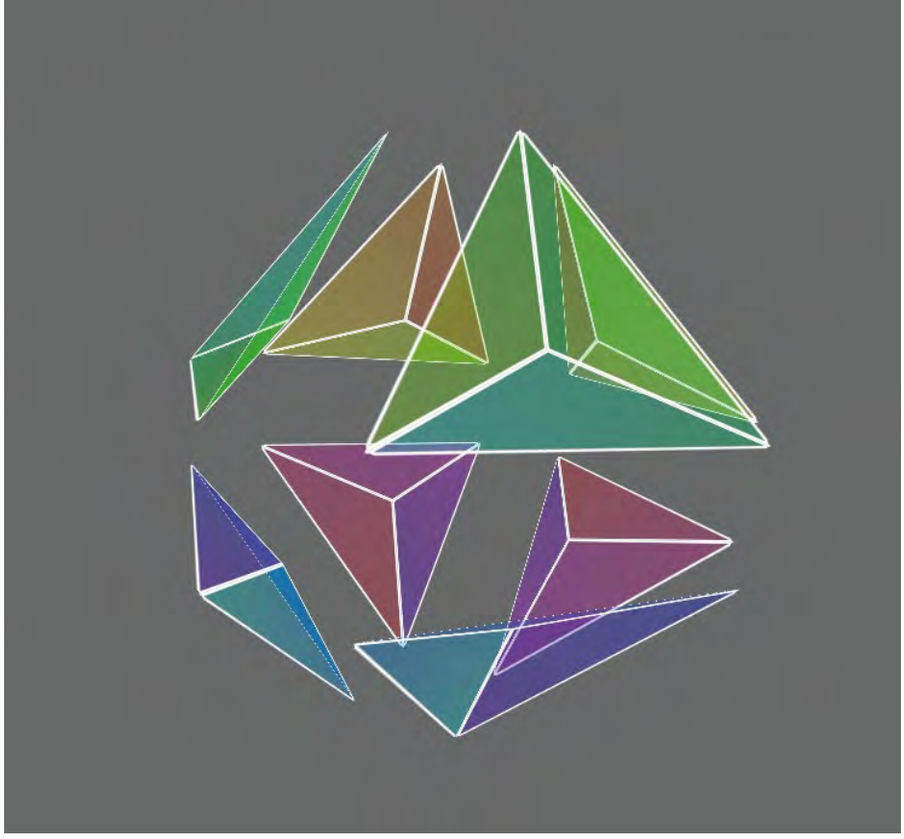
I loved `calc()` from the moment I discovered it. It's useful for taming margins, padding or dimensions, and can be a lifesaver when used for positioning or sizing backgrounds, inside gradients or transforms, and combined not just with the same old units, but also the new and cool viewport ones.

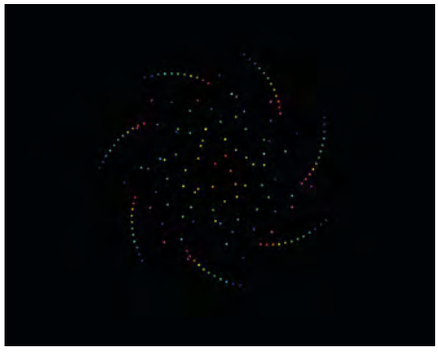
06 MAKE THE BOX MODEL SANE WITH 'BOX-SIZING'

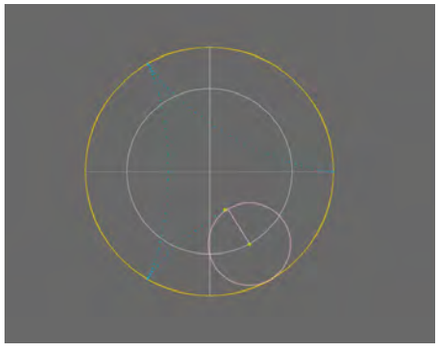
Sawyer Hollenshead, Oak Studios dev and designer

Use `box-sizing` to save your sanity. Without it, an

Colourful cats CSS can be a useful method for applying tones to greyscale images, according to Christopher Schmitt







element with a defined width of 250px and 25px of padding combines to a width of 300px, making mixing pixels and percentages tough. With `box-sizing: border-box` borders and padding are instead placed within the defined width.

07 VERTICALLY CENTRE WITH CSS

Trent Walton, Paravel founder

Historically, it's been tough to vertically centre something with CSS, such as if you've an image with adjacent text you want vertically aligned. Instead of stomping and cursing, use Flexbox to deal with alignment woes.

See a demo*: codepen.io/TrentWalton/pen/FDBbu*

08 TARGET A BANK OF RELATED OBJECTS

Jonathan Smiley, Zurb partner and design lead

Shave CSS line weight by using approximate attribute selectors on class names, to target a large bank of related objects, rather than attaching common attributes to every single class. For example ...

```
[class*="-block-grid-"] { }
```

... would target the likes of:

```
.small-block-grid-3
.large-block-grid-5
```

09 CONTROL HYPHENATION

David Storey, open web advocate

Hyphenation is taken for granted in print, and some developers use the `hyphens` property online, but few are aware of other properties that provide finer control. If you're not careful, you get hyphenation ladders where hyphens are used across multiple lines. A general rule of thumb is no more than two in a row, which you can control using `hyphenate-limit-lines`. Also, `hyphenate-limit-chars` enables you to specify the minimum length of a word that will be hyphenated, along with the minimum number of characters before and after the hyphen break.

10 TAKE ADVANTAGE OF WRITING MODES

David Storey, open web advocate

Writing modes enable you to define the direction in which text flows. Some East Asian text is written vertically, lines growing from right to left, specified with `writing-mode: vertical-rl` (`tb-rl` in IE). Vertical text isn't really used in European writing systems, but can be handy for table headings when you've limited horizontal space.

11 USE GRADIENTS IN UNUSUAL WAYS

Ruth John, designer

Background gradients can look great when used with borders and bullets. I use both on my blog (*rumyrashead.com/category/speak-write-mentor*) and with a preprocessor can call a mixin with the reused code, so as not to repeat it manually. Don't go super-crazy because gradients can be processor-heavy. SCSS mixin for list bullets:

```
@mixin gradedBullet($colour) {
background-image:
 linear-gradient(left,
```

▶

Above left Ana Tudor's demos showcase how some CSS magic can create polyhedra that can be animated and even exploded/recombined

Above top Particle animations with 'box-shadow', maths and Sass, courtesy of Ana Tudor

Above right A digital plotter in CSS, again utilising 'box-shadow' and using Ana Tudor's mastery of maths

```
lighten($colour, 15%) 10px,
$colour 11px,
$colour 20px,
darken($colour, 15%) 21px,
darken($colour, 15%) 30px,
transparent 31px
);
}
```

12 USE STRING-MATCHING ON LINKS

Ruth John, designer

On my blog, I've used CSS attribute selectors with string-matching to style social icons. These appear throughout my blog, sometimes with text and sometimes without, but always with an icon. To style the right link with the correct social icon, I use a string match on the `href` attribute of the anchor element. I use `*=` so the `href` on the anchor element only has to contain the string I specify.

```
/* for all social links */
.social a:before {display: inline-block; padding-right: 30px; font-family: 'FontAwesome';}

/*Each specific link*/
.social a[href*="twitter"]:before {content:"\f099"; color:#52ae9f;}
.social a[href*="github"]:before {content:"\f09b"; color:#5f2e44;}
.social a[href*="feed"]:before {content:"\f09e"; color:#b47742;}
```

13 MAKE FOUT WORK FOR YOU

Jason Pamental, principal at H+W Design

The web is built on the premise it should deliver content, even if the browser can't render the bling. Slow-loading web fonts can be frustrating, FOUT (Flash Of Unstyled Text) jarring as navigation and text reflows while fonts download. Google and Typekit provide an answer: the web-font loader. By injecting classes in a page, based on the loading status of fonts, you can style fallbacks with

Above right Jason Pamental says that you can make FOUT work for you through utilising a web-font loader

Right Ruth John shows how there's more to gradients than backgrounds, using them for borders and bullets on her blog

those classes to keep reflow to a minimum, also eradicating WebKit's 'invisible content' syndrome.
See a demo*: hwdesignco.com/otf/google*

14 EXPLORE SVG FOR BACKGROUNDS

Emil Björklund, inUse web developer

The only browsers now without SVG support are IE8 and below and Android 2 WebKit, and so using SVG for backgrounds in CSS is feasible, especially along with a PNG fallback solution, such as Grunticon. SVG can be styled by CSS, and there is interesting bleed-through of CSS properties (filters!) from SVG that we can play with as applied to HTML.

15 FOCUS USERS WITH 3D TRANSITIONS

Emil Björklund, inUse web developer

Using 3D transformations and making use of the `z-dimension` in user interfaces can be really useful, notably for hiding/showing or collapsing/expanding content. It's also quite easy to have a fallback to a 2D transition, or no transition at all, in those situations. It's an area where a little progressive enhancement can go a long way.

16 CREATE CIRCULAR MENUS WITH CSS AND MATHS

Sara Soueidan, frontend developer

Circular menus are popular in mobile apps, and you can use CSS transforms and transitions to create a simple circular menu. This menu can be modified and customised to create an upwards- or downwards-opening menu, too. There's no direct way in CSS to translate an item diagonally, but you can use the value of the circle's radius on which you want to position the items, and apply a simple mathematical rule to calculate the horizontal and

FOLDABLE.ME

By Mint Digital
mintdigital.com

Product development studio Mint Digital partakes in an annual hack-week. A recent brief was all about creating something to bring the online world offline and scratch a creative itch. The result was Foldable.Me (*www.foldable.me*), a tool that enables you to design, print and construct a customisable cardboard avatar. On selecting a gender, the tool enables you to add hair, facial features, clothing and colours, and the little cardboard figure can be twirled with the flick of a finger or a mouse-pointer drag.

Developer Peter Westendorp recalls that it was suggested that a 3D preview of the avatar should be built using CSS 3D transforms: "This sounded crazy and unrealistic, but we gave it a shot. After a few hours hacking on a prototype, it became apparent this really did work." The technology was especially useful in terms of the way it fared across browsers: "We could show the model in 3D on the latest browsers, whilst gracefully degrading to a 2D model on older ones. That's the great thing about CSS – it can just naturally degrade without entirely breaking the user experience."

Although Westendorp rightly states that 3D transforms shouldn't be wheeled out for every project, he believes they and other new CSS features are nonetheless something designers and developers alike should continue experimenting with: "That way, we will keep coming up with some great universal UX improvements. We have to continue innovating and shouldn't be hesitant to try out new techniques. It keeps our work interesting."

vertical translation values to pass to the `translateX()` and `translateY()` functions. That way, you end up with a `diagonal` translation to move the menu items to the correct positions on the circle. The click event that closes/opens the menu can be handled using JavaScript, or you can take it one step further and have a CSS-only menu by using a CSS checkbox hack.

In my demo, I use JavaScript and the HTML5 classList API, which isn't supported in all browsers so you'll need to view the demo in a modern browser to make it work, or uncomment the jQuery code instead of using the classList API code.

See a demo and full tutorial: ***codepen.io/SaraSoueidan/pen/wpHBt*** **CSS Checkbox hack example:** ***codepen.io/TimPietrusky/pen/fHmil***

17 ANIMATE LINKS ON HOVER

Paul Lloyd, *The Guardian* interaction designer

Hover states shouldn't be relied on to make an action work or provide important information, but you can still enhance interfaces for mouse-based users. On *24ways.org*, we reveal article titles when you hover over links in the previous/next navigation. This was achieved by creating a `::after` pseudo-element containing generated content sourced from the value of a `data-` attribute, with a CSS transition applied to have it slide into view on hover.

See a demo: ***jsbin.com/xaruj/1/edit***

18 MAKE SIMPLE KEYFRAME ANIMATIONS

Paul Lloyd, the *Guardian* interaction designer On *24ways.org*, we added animated corner flaps to summaries, which opened on hover. This was done by combining the `@keyframes` rule with the `animation` property, altering the position of a background image to achieve sprite-based animation. The trick is to declare the number of frames you have in your animation sprite with the `steps()` value.

Demo: ***jsbin.com/buwa/1/edit***

19 CREATE FLOATING 3D EFFECTS WITH SHADOWS

Catherine Farman, Happy Cog developer

A recent project required a floating product photo with a round shadow beneath, creating a 3D effect ▶

of popping off the screen. The shadow uses several CSS3 features: `border-radius` alpha transparency and `box-shadow`. It works well for product grids, showcase imagery in a homepage hero, or any whimsical design with a skeuomorphic bent.
Demo: *codepen.io/cfarm/pen/FHxnr*

Above Catherine Farman shows how CSS creates a floating shadow effect beneath a product image

Below left The new *24ways.org* reveals animated titles when mouse users hover over previous and next links in the site navigation

Below right *24ways.org* uses simple keyframe animations to add a little extra visual interest to each item when the cursor hovers over it

20 UPDATE PAGE ELEMENTS USING ':TARGET'

Simon Madine, HERE senior web developer

CSS isn't a programming language in the usual sense, but you can still do clever things without falling back to JavaScript. For example, the `:target` pseudo-class is applied to elements that are the target of a clicked link.

You can use this to define the state of a page, target a parent containing lots of elements, and your links become a means to control the look and layout of all the children with a single click.
See a demo: *jsfiddle.net/DMNSn*

21 PROVIDE FEEDBACK WITH SUBTLE ANIMATIONS

Neil Renicker, designer and developer

CSS pseudo-elements `::before` and `::after` along with CSS transitions, can enable delightful animation that provides subtle feedback to mouse users. For example, build a CSS arrow within a pseudo-element, apply a transition to the pseudo-element (`transition: all ease-in-out .15s;`), and then add a simple layout change to the `:hover` pseudo-class (such as amending `margin-top`).
See a demo: *codepen.io/neilrenicker/full/Eyhqn*

22 PREPARE FOR 'WILL-ANIMATE'

Paul Lewis, coder and Chrome developer relations team member

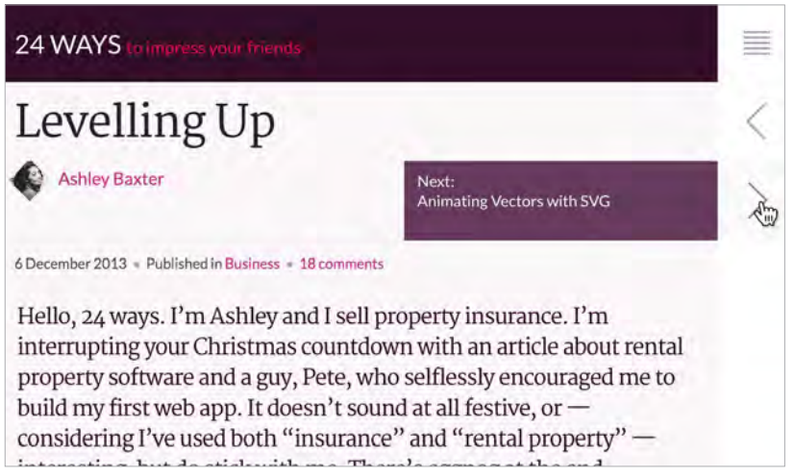
If you've used `-webkit-transform: translateZ(0)` to magically make your pages faster, the hack, which in many browsers simply creates a new compositor layer, is being replaced by `will-animate`. Soon, you'll be able to tell the browser what you plan to change about an element (its position, size, contents or scroll position) and the browser will apply the right optimisation under the hood.
From more information: *tabatkins.github.io/specs/css-will-change*

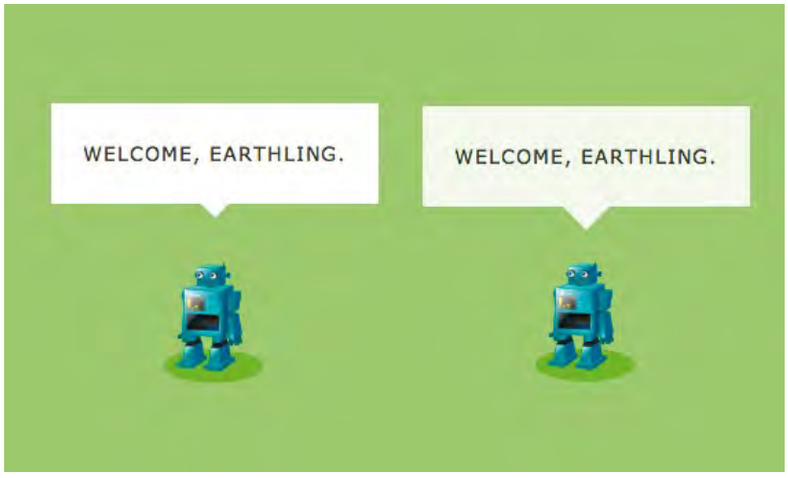
23 HUMANISE INPUT FIELDS

Yaron Schoen, Made For Humans founder

Adding quick animations to elements that users interact with makes an interface feel less computery. With input fields, try putting a `transition call` within, so whenever you focus or unfocus it, there's a smooth transition.

```
input, textarea {
-moz-transition: all 0.2s ease-out;
```





Welcome, earthling
Neil Renicker's subtle animation provides useful feedback for mouse users, and it's all done through CSS pseudo-elements

```
-o-transition: all 0.2s ease-out;
-webkit-transition: all 0.2s ease-out;
-ms-transition: all 0.2s ease-out;
transition: all 0.2s ease-out;
}
```

24 PAUSE AND PLAY CSS ANIMATIONS

Val Head, designer and consultant

You can 'pause' and 'play' CSS animation by changing its `animation-play-state` property. Setting it to 'paused' stops your animation in place, until you change `animation-play-state` to `running`, for example on hover.

```
.animating_thing {
animation: spin 10s linear infinite;
animation-play-state: paused;
}
. animating_thing:hover {
animation-play-state: running;
}
```

25 DON'T USE CSS VARIABLES

Dave Shea, designer and author

We're finally getting CSS variables, for example to write a colour's hex value once and reference it through a stylesheet. But the official spec is verbose, adds syntactic complexity, offers underwhelming functionality, and is largely unsupported by most browsers. In an era where Sass is widely popular and goes beyond variables with powerful programming logic like custom functions and if/else statements, the official spec comes up far short.

Hopefully these top tips have renewed your view of CSS and the possibilities it represents in web development and design. Don't forget to test any of these techniques thoroughly to check browser support before putting any work live.

RESOURCES

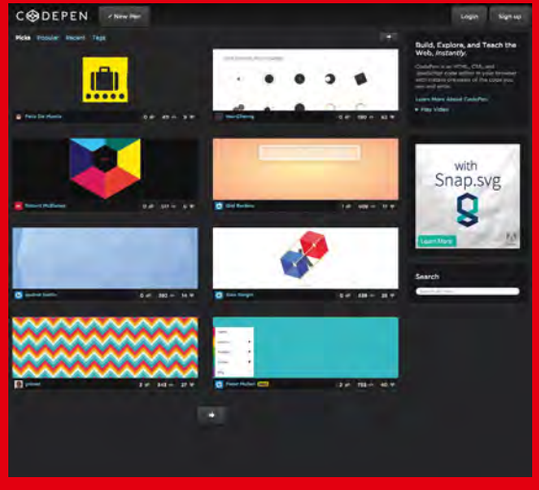
CSS-Tricks
A superb website written by Chris Coyier that provides insight into all things CSS, including browser behaviour, use-case scenarios for a range of properties and values, and critical thinking of specs.
css-tricks.com

Mozilla Developer Network
Mozilla prides itself on openness, and that's especially so when it comes to sharing knowledge. Within MDN, you'll find all manner of CSS news, demos and tutorials.
developer.mozilla.org/en-US

W3C Cascading Style Sheets
This is the home on the web for official news and information all about CSS standards. Mailing lists are available should you want to get more involved in the direction of the technology.
w3.org/Style/CSS

Codecademy
For those starting out or in need of a quick refresher course, Codecademy provides tutorials that enable you to learn the building blocks of the web.
codecademy.com/tracks/web

CodePen
An interactive playground combined with a teaching aid, CodePen enables you to create and share your demonstrations of web-page components, seeing changes live as code is amended.
codepen.io



Interactive playground You can see changes to demos live via CodePen

Download the files here!

All the files you need for this tutorial can be found at *netm.ag/tridiv-248*

ABOUT THE AUTHOR

JULIAN GARNIER

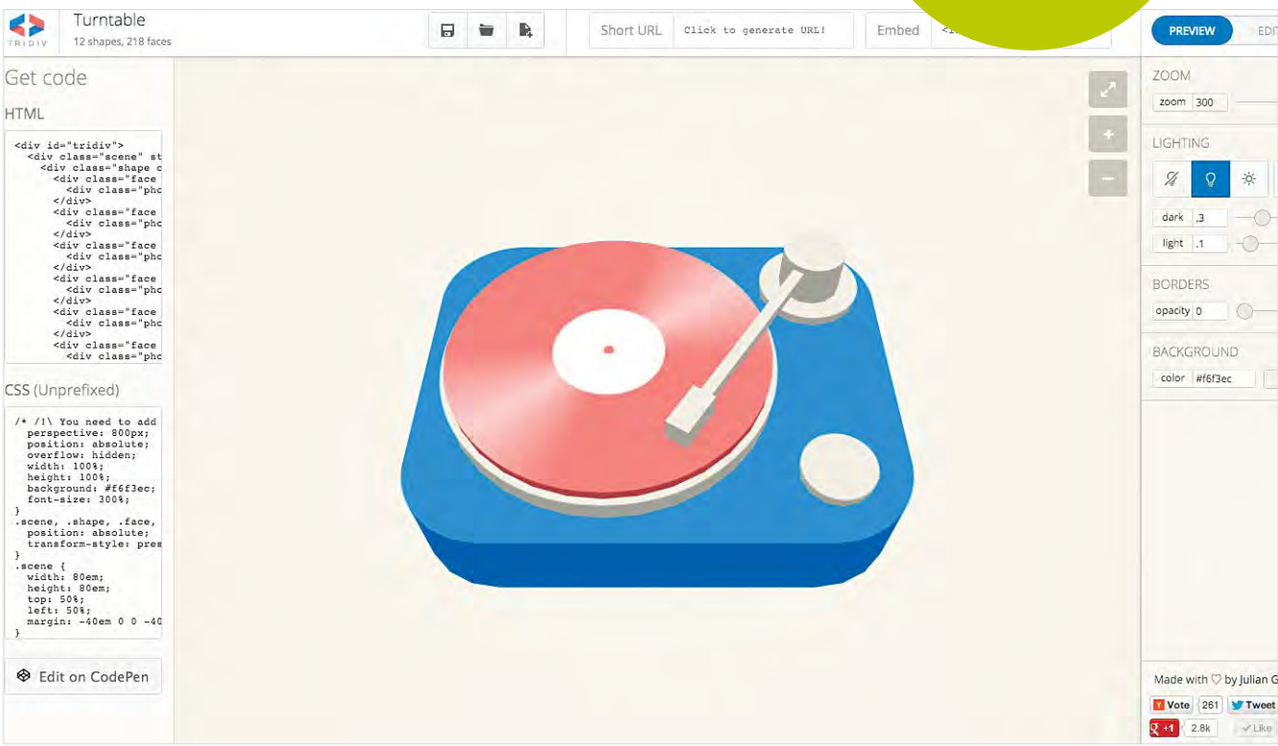
w: *juliangarnier.com*

t: @JulianGarnier

areas of expertise:
HTML, CSS, JS, design

q: what's the oddest thing you've ever seen on a designer's desk?

a: An N64 controller next to a pyrite block



CSS

CREATE AN ANIMATED 3D LOGO WITH TRIDIV

Julian Garnier explains how to produce a logo for a record label with Tridiv: his online tool for creating 3D objects without writing code

There are several ways to create 3D animations on the web, most of them requiring a good knowledge of JavaScript and WebGL, or the use of a plug-in like Flash. Thanks to CSS 3D transforms, it's possible to create 3D using only HTML and CSS, but it isn't easy to do so. Tridiv, my free online app, simplifies the process, offering a simple and intuitive WYSIWYG interface that enables users to create 3D objects without writing a single line of code.

In this tutorial, we're going to create and animate a logo for 'Tridiv Records', a fictional record label, using only HTML and CSS. The main visual for the logo is going to be created in 3D using Tridiv. Then we will add the typographic elements using regular HTML and CSS.

You can see the final animation and the code that generates it on CodePen at *netm.ag/tridiv-248*.

GETTING STARTED

We're going to begin by creating the turntable in 3D using Tridiv. Head to *tridiv.com* and launch the app. You'll need to be using either Chrome, Safari, or Opera 15 (or later).

Before starting, it's important to understand the Tridiv interface. The main section of the editor is composed of four views: on the top left is the 3D view, providing a complete view of the scene. The other three views show it from the top, side and front. Using these three views, you can create, edit and move 3D shapes.

RESOURCE

TRIDIV INSPIRATION

For inspiration for how to use Tridiv, visit the Tridiv website for some examples of its use, including an iPhone 4S 3D object. *tridiv.com/#examples*

The horizontal toolbar is divided into two parts: the left part displays information relating to your document; the right part contains tools for creating and editing shapes. The `Move` selection and `Edit` selection buttons switch between the different editing modes.

The properties pane (the sidebar) displays document settings such as zoom and snap to grid, and the properties of the shape selected (size, position, rotation, colour, and so on). The unit used for dimensions and position is ems; the rotation angles are in degrees.

To avoid any confusion later in the tutorial, we're going to use the following shorthand:

w = width
h = height
d = depth
diam = diameter
x deg = rotation in the x-axis
y deg = rotation in the y-axis
z deg = rotation in the z-axis

CREATING THE BASE OF THE TURNTABLE

Start by setting the zoom value to `200`. To help draw the shapes, activate the snap to grid setting in the `Document Settings` section of the sidebar. Set the snap value to `0.125`.

Tridiv offers a simple WYSIWYG interface that enables users to create 3D objects

The base of the turntable is composed of a simple cuboid, so click the `Add cuboid` button in the top toolbar. You should see the cuboid appear in all four views in the editor.

Rename the shape to `base` using the name field of the properties pane (under `Shape Properties`). The name of the shape must be a valid CSS class name because it will be used in the code generated by the editor. We will use these class names later when animating the logo, so make sure that you name every new shape that you create properly.

Once the cuboid is named, make sure that it is selected in the top view (it should be highlighted in blue, with a circular ring of tools around it), then click the `Edit` button at the top of the ring to show the edit handles. Drag the control handles at the sides of the cuboid, until width and depth reach `w=10` and `d=8` in the `Shape Properties`.

▶

IN-DEPTH

MORE ABOUT TRIDIV

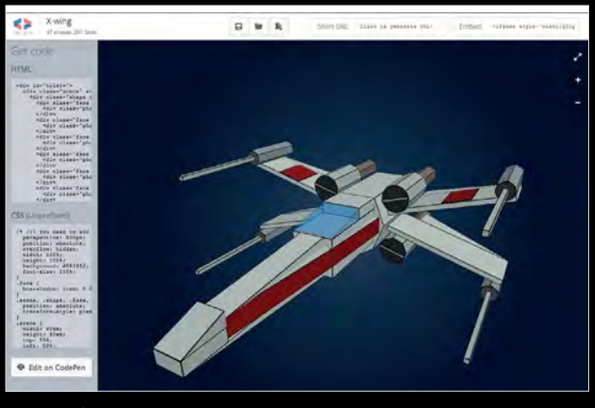
About Tridiv The app creates 3D shapes in CSS through a WYSIWYG interface

Tridiv has been developed in order to help web developers and designers create 3D shapes in CSS. The app is web-based, totally free, and no sign-up is required.

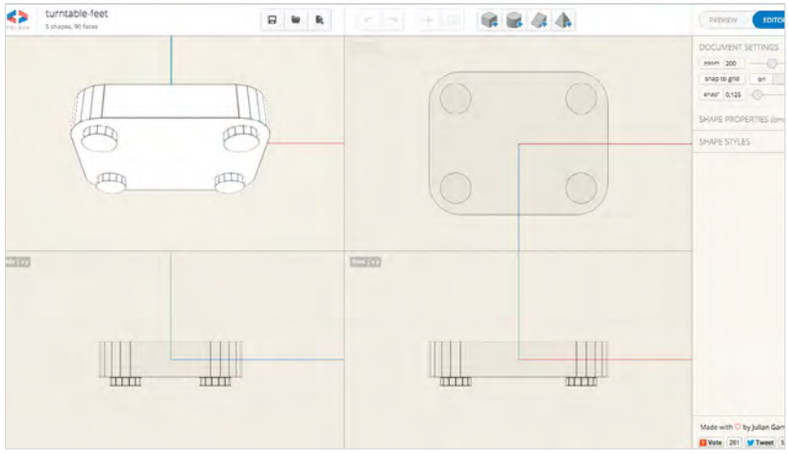
The interface is simple, but offers a complete set of features. Scaling, rotating and moving shapes works like in any other graphic design application: by selecting and dragging points inside the editor. You can assign colours, and apply textures to the 3D geometry, simply by pasting in the URL of an image. To share a Tridiv model, you just copy/paste a simple URL using the built-in shorten URL link.

But the most interesting feature of Tridiv is its ability to export clean HTML and CSS. It's even possible to specify your own HTML class name to fit your needs. It also exports directly to CodePen, which is probably the easiest way to start experimenting with the code. From there, anybody with a basic knowledge of CSS can edit and transform the model using regular CSS, and use the result in any website.

Due to performance-related problems, the app is unfortunately not compatible with Firefox at this point. But it's important to note that the exported code works on all major recent browsers, with the exception of Internet Explorer.



Complex forms Tridiv can create complex 3D models like this X-Wing, for example



Taking shape We're using cylinders to create the turntable feet. Make they're aligned correctly under the base

Click on the shape inside the side view. This will show the editing handles in this view, allowing us to change its height. Adjust the height until it reaches `h=2`. You can also type values directly into the properties pane. To round off the corners of the cuboid, change the corners values in the properties pane to `1.75`, then press the `[Enter]` key to apply the changes. You'll have something like: *tridiv.com/d/4kb.*

CREATING THE FEET

For the turntable's feet, we are going to use cylinders. Add a cylinder, then change its diameter to `diam=1.75` and its height to `h=0.5`. Click on the `Move` selection button in the top toolbar to show the draggable area on the shape. Move the cylinder under the base, placing it in one of the corners. (You may need to move it in the top, side and front views.)

Duplicate the cylinder (press the `Duplicate` button in the circular ring of tools, or press the `D` key) and to move the new cylinder to another corner of the base. Repeat the process until all four feet are correctly positioned. Don't forget to name the cylinders (for example, `feet-left-top`, `feet-right-top`, `feet-left-bottom`, `feet-left-top`). When you've done that, the result should look like this: *tridiv.com/d/4kc.*

We'll now look at creating the platter, disc, arm axis and button. The process for creating the next shapes is similar to that for the feet. Here are the dimensions used for the different cylinders:

platter: diam = 7; h = 0,5
disc: diam = 6.75; h = 0,25
button: diam = 1.5; h = 0,25
arm-axis-base: diam = 2.25; h = 0,25
arm-axis: diam = 1.375; h = 1

To refine the sides of the cylinders, you can increase the number of faces in each one, using the sides field in the properties pane. Don't add too many sides as this can negatively influence the global performance of the editor and the final animation. In this case, I'd advise you not to use more than 32 sides for the platter and the disc. You should have something like: *tridiv.com/d/4ke.*

THE ARM AND THE HEAD

For the arm and the head of the turntable, we're going to use cuboids. For the arm, create a cuboid (`w=0.25; h=0.25; d=4`), then apply a rotation of `-33°` on the `y-axis`. For the head, create a cuboid (`w=0.5; h=0.5; d=1`), then apply a rotation of `-33°` on the `y-axis`. Align both shapes with the arm-axis cylinder. The result should look like *tridiv.com/d/4kf.*

COLOURS AND TEXTURES

We're almost done with the turntable. The final step is to assign colours and apply a texture to the vinyl (an image representing the surface of the record). To assign colours, select a shape and click on the `colors` field in the properties pane. Tridiv allows you to specify individual colours for each face of a shape, but, in this example, we need to use the all field to change the colour of all the faces. To do this, just enter a hex colour code in the field, then confirm by pressing `Enter`.

Tridiv allows you to specify individual colours for each face of a shape

Here are the colours used in this example:

base: #0099FF
feet, button, axis, arm and head: #F2EEE5
disc: #fa7f7a

For the texture of the vinyl, the process is similar to assigning colours. Select the disc cylinder, then click on the `images` field in the properties pane. Paste the URL of the image you want to apply to the vinyl in the top field and confirm by pressing `Enter`. You can use an image of your own, or download the one used in this example from: *netm.ag/texture-248.*

You should now have something that looks like this: *tridiv.com/d/4kg.*

RENDERING AND EXPORTING

Now that the turntable is done, we're going to work on the way in which it's rendered before exporting it.

RESOURCE

VENDOR PREFIXES

As the CSS code generated by Tridiv doesn't use vendor prefixes, try using tools like *prefixr.com* or *leaverou.github.io/prefixfree* to ensure the code is functional in every browser.

Click the `Preview` button on the top of the properties pane. Set the zoom value to `200` to display the turntable bigger. To remove the black borders of the shapes, go to the `Borders` section of the pane and set the opacity to `0`. The result should look something like this: *tridiv.com/d/4k6*.

We want the turntable to be lit from the top. To do this, rotate the scene in such a way that the top of the turntable is facing you. The base should look perfectly rectangular. Changing the light and dark values in the `Lighting` section of the properties pane will regenerate the shadows within the scene. Change the light value to `0`.

The turntable is now ready to export!

FINISHING THE LOGO

We're ready to add the text to the logo and create the logo animation. Click the `Edit` on CodePen button on the bottom left of the `Preview` view to export the code to CodePen. It's important to note that the CSS code generated by Tridiv doesn't use vendor prefixes, so you will need to use tools like *prefixr.com* or *leaverou.github.io/prefixfree* in order to make the code functional in every browser. Start by closing the JavaScript pane, as we aren't going to use it. In the HTML pane, remove the style tag applied to the `.scene` div.

Expand the CSS pane and add the following code at the end:

```
.scene {
transform: translateY(-140px) rotateX(-55deg);
}
```

Here, the `translateY(-140px)` moves the turntable 140px upwards, leaving room for the text beneath it. Then, the `rotateX(-55deg)` sets the vertical inclination of the turntable.

To add the text, you need to add a `.title` div right after the opening `#tridiv` div in the HTML pane. Inside, add two `<spans>` (`.main-title` and `.sub-title`), separated by `<hr/>`:

```
<div id="tridiv">
<div class="title">
<span class="main-title">TRIDIV</span>
<hr/>
<span class="sub-title">RECORDS</span>
</div>
...
```

You then need to apply the correct fonts and styles. In the CSS pane, import the Open Sans font used in the logo, and add the basic styles for the text elements (shown overleaf):

▶

IN-DEPTH

ANIMATE THE LOGO

See an animation using the logo at *netm.ag/tridiv-248*. As the parts of the turntable 'fall' in, each of them share the same keyframe animation with different delays. The shapes have the top attribute set to `50%`. To create the falling effect, we animate the top attribute from `-400px` to `50%`:

```
@keyframes fall {
 0% { top: -400px; } /* We start the animation positioning the shape to a height of 400px */
 100% { top: 50%; } /* then we end it on its original position */
}
```

You can add this animation to all the shapes, as follows:

```
.shape {
 top: -400px;
 animation: fall 1s ease 0s forwards;
}
```

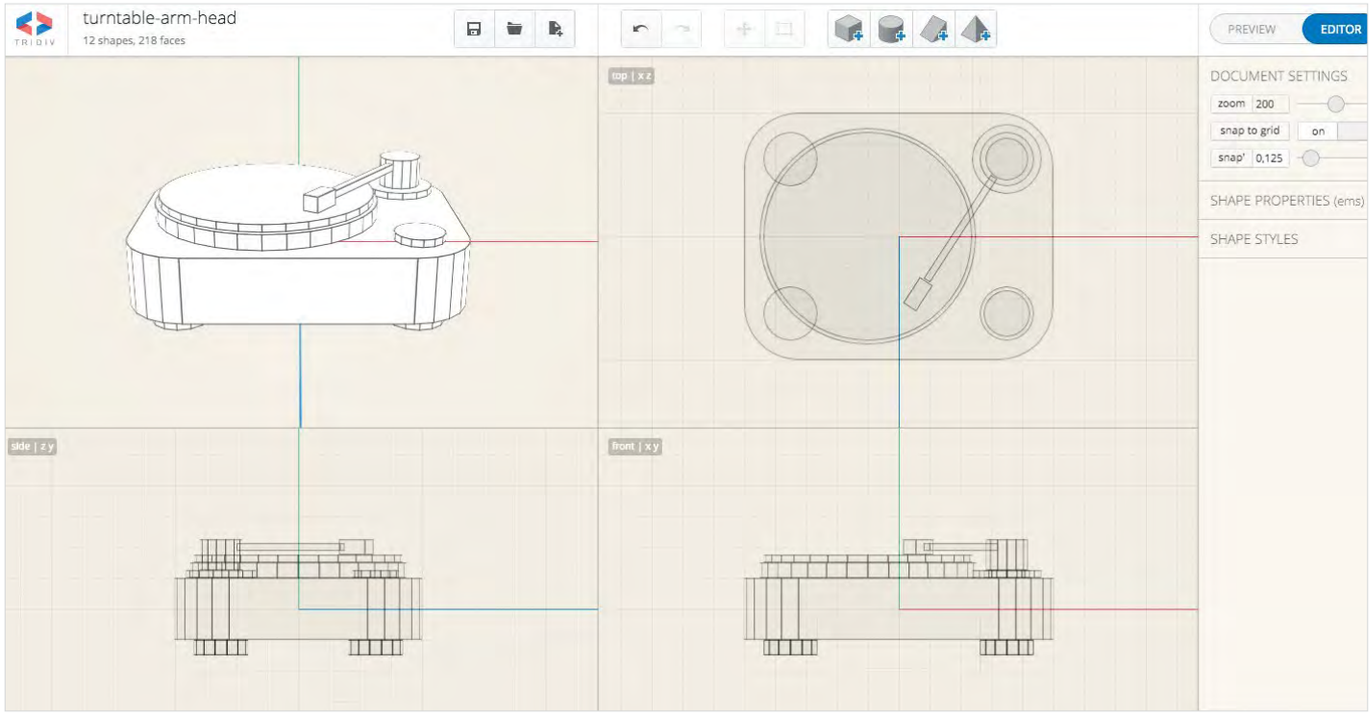
Set the top attribute to `-400px` and add a delay:

```
.platter      { animation-delay: 1.05s; }
.disc         { animation-delay: 1.35s; }
.button       { animation-delay: 1.5s; }
...
```

Create the final 'bounce' effect using the `rotateX` attribute:

```
90% { transform: translateY(-5em) rotateX(780deg) rotateY(0deg); }
95% { transform: translateY(-4em) rotateX(620deg) rotateY(0deg); }
100% { transform: translateY(-4.5em) rotateX(660deg) rotateY(0deg); }
```

Animation frames The turntable parts 'fall' in and then the entire turntable spins



Precise positioning We're using cuboids to form the turntable arm and head. Creating the turntable arm can be tricky. Therefore, it's important to ensure you check the shapes aren't overlapping

```
@import url(http://fonts.googleapis.com/
css?family=Open+Sans:300);
/* Text block centering + basic font styles */
title {
 position: absolute;
 top: 50%;
 left: 50%;
 margin: 0 0 0 -165px;
 width: 330px;
 height: 5em;
 font-family: 'Open Sans', sans-serif;
 font-weight: 300;
 font-size: 24px;
 text-align: center;
 letter-spacing: 1.5em;
 color: #0099FF;
}
title hr {
 border: 1px solid #fa7f7a;
 margin: .75em 0;
}
title span {
 display: block;
}
.main-title {
 font-size: 2.15em;
}
.sub-title {
 text-indent: .25em;
}
```

DEMO

ANIMATING THE LOGO

See an animation of the completed logo on CodePen. Turn to page 101 for instructions on how to achieve the same effect.
netm.ag/tridiv-248

Voilà! Your logo is complete. It should look something like the image below. Once your 3D model is done, you can use the power of CSS to make it even better by adding styles, animations or mouse events: just treat the 3D model like any other HTML element. In the 'Animate the logo' section on page 101, we looked at how the animation shown at *netm.ag/tridiv-248* was created using the final logo – but remember: there are no limits!

Correct lighting We've changed the light and dark values. By rotating the turntable in this way, we ensure the shadows on the sides are all uniform

CSS

HOW TO USE CSS SHAPES

Razvan Caliman introduces a new CSS feature, CSS Shapes, that allows designers to wrap content around custom shapes

Web designers have long been forced to create within the constraints of the block. Since the majority of daring ventures into non-rectangular layout using CSS and HTML end in frustration, most content on the web is still trapped in simple boxes. Creative uses of `border-radius` and CSS transform properties only create the illusion of shapes and don't actually affect the content layout as expected.

A circle carved with border-radius is still a rectangle in disguise. All this is about to change with the implementation of CSS Shapes (*html.adobe.com/webplatform/layout/shapes*).

CSS Shapes allows web designers to wrap content inside and around custom shapes. Shapes are defined in simple, practical and reusable ways, and, unlike most CSS hacks that just simulate shapes, can affect the entire layout of a web page.

Perhaps you've been as naive as me when first floating an image with some transparent parts expecting content to wrap neatly around it, filling the blanks. Instead, we saw it wrap around the image block. How disappointing! We can solve this problem using CSS Shapes.

Here's a simple circle shape example:

```
.foo { shape-outside: circle(200px at 50% 50%); float: left; }
```

The `shape-outside` property here creates a circle with a radius of `200px` starting from the centre of the element. Use of the `float` property causes the content outside of the element to wrap around the circle shape. Note that this only affects the layout of content, not the clipping of the element itself to a circle; the `clip-path` property should be used for that and you can reuse the same property value.

Content can also be wrapped inside of a custom shape by using the `shape-inside` property:

```
.bar { shape-inside: circle(50%, 50%, 200px); }
```

Explicit width and height must be set when using `shape-*` properties. For brevity, dimensions have been omitted from these examples.

Shape property values can be defined with built-in shape functions such as `circle()`, `ellipse()` and `polygon()`, or they can be derived from images which have an alpha channel by using `url(path-to-image.png)`. All types of CSS units can be used within shape definitions. This means that shapes can also be responsive. Regular margins and padding don't make sense with shapes, especially when thinking about polygons. What is a margin-top for a triangle anyway? Two new properties, `shape-margin` and `shape-padding`, are used to achieve the desired effect.

CSS Shapes lend themselves to a progressive enhancement development approach. When not supported, browsers will ignore the shape properties and fallback to plain blocks, thus keeping content visible. However, as with all new features, due diligence is necessary to make sure content is accessible for all users.

The CSS Shapes specification and implementation are still in flux with changes likely to come. Because of this, the feature is behind a flag in Google Chrome. Adobe is working on the implementation of CSS Shapes and keeps track of browser support and how to enable the feature at *html.adobe.com/webplatform/enable*.

PROFILE

Razvan (@razvancaliman) is an engineer on the web platform team at Adobe. He experiments with cutting-edge web technologies and builds things to make life easier for web developers

Download the files here!

All the files you need for this tutorial can be found at *netm.ag/cssnotjquery-254*

ABOUT THE AUTHOR

DAN NEAME

w: *dan.nea.me*

t: @cham

areas of expertise: JavaScript, CSS

q: What cartoon character do you most identify with?

a: Goofy, because pls

CSS

REPLACE JQUERY WITH HTML AND CSS

Dan Neame explains how to use CSS, not jQuery, for UI animations and widgets, leading to smoother animation and cleaner degradation

jQuery has been very successful in shaping the modern web: so much so that many of its features have fed back into the design of CSS. In this tutorial, we'll see how to reproduce common jQuery effects with CSS. You will need a basic understanding of HTML, CSS and jQuery, though even experts may learn new things to do with pseudo-classes.

There are a number of good reasons to use CSS in place of jQuery. First, CSS is often much faster than JavaScript. For example, when you use `slideUp()`, jQuery modifies the element's `height` property programmatically via the `style` attribute on the element. Modifying element properties is much slower than defining the animation via CSS: the browser can calculate what height to apply for you and doesn't need to modify the DOM, resulting in faster execution, which means smoother animation.

CSS is also both forward-compatible and degrades cleanly: if a CSS property doesn't apply then the content is still accessible; your user won't just see a particular animation or a rounded border. If there's an error in your JavaScript then everything stops.

Support for CSS properties varies between browsers. Everything here works in recent versions of Gecko/WebKit/Blink browsers and IE9+, but visit *caniuse.com* to check support for other properties.

TRANSITIONS

`slideUp()`, `slideDown()` and `slideToggle()` are some of the most-used functions in jQuery, often tied to a button press or mouseover. In order to replicate this effect, we'll need to use `transition`. This enables us to specify which styles we'd like to animate, how long we'd like the animation to take and what sort of easing we'd like to use. The following CSS will animate just the height property for two seconds, with linear easing:

```
.slider {
    transition: height 2s linear;
}
```

When the height of `slider` is changed, the browser will animate between the old and new heights. The following code is all you need to try this for yourself:

HTML

```
<button class="toggler">Click me</button>
<div class="slider"></div>
```

CSS

```
.slider {
    transition: height 2s linear;
    height: 100px;
    background: red;
}
.slider.down {
    height: 500px;
}
```

jQuery

```
$('.toggler').click(function(){
    $('.slider').toggleClass('down');
});
```

RESOURCE

JQUERY AND FRIENDS

See slides from Dan Neame's talk providing an overview of jQuery and its uses: *netm.ag/neame-jquery-254*

This creates a nice smooth animation but we're still using jQuery to apply the class. Fortunately, it is possible to modify the state of the document without using jQuery via a pseudo-class.

PSEUDO-CLASS SELECTORS

Pseudo-class selectors enable us to modify the styling of an element by matching against information outside the document; for example, if the user has hovered or clicked on an element. We'll take a look at three of these and see how they can be used to respond to user input in place of jQuery.

:focus

The `:focus` pseudo-class selector facilitates targeting elements that have been clicked on or tabbed to with the keyboard. By default, only form fields will be able to receive focus, but we can allow other elements to receive focus by giving them the `tabindex` attribute.

CSS degrades cleanly. If a property doesn't apply, your user just won't see the content

A good use for `:focus` is modal pop-ups. If the modal is placed inside the DOM tree of the element that triggers the pop-up, it will retain focus if you click inside the modal. Clicking outside the modal closes the pop-up:

HTML

```
<div class="modal-launcher" tabindex="1">
    Click here to open
    <div class="modal">
    <span class="close-button" tabindex="2">x</span>
    Modal popup!
    </div>
</div>
```

CSS

```
.modal {
    display: none;
    position: absolute;
    top: 25%;
    left: 25%;
    bottom: 25%;
    right: 25%;
    border: 1px solid #ccc;
```

▶

FOCUS ON

ANIMATING COLOUR

Animating the colour of an element is actually quite difficult to accomplish with jQuery. You'll need to use a plugin or a library such as jQuery UI (which is quite heavy). Fortunately, it's now possible to animate colour with CSS. Again, we're going to need to use `transition`:

CSS

```
.colourful {
    transition: background-color 2s linear;
}
```

This will take two seconds to run any time the `background-color` property of the element is changed. The following code changes the background colour from postbox red to blue when hovered:

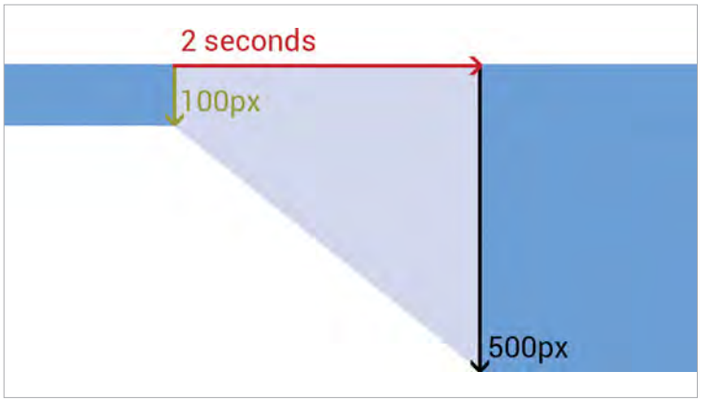
HTML

```
<div class="colourful"></div>
```

CSS

```
.colourful {
    transition: background-color 2s linear;
    height: 100px;
    background: #c00;
}
.colourful:hover {
    background: blue;
}
```

The nice thing about this approach is that it can transition between colours defined in different ways – by hex, name or RGBA value.



Height change The code on the facing page animates the height of a slider over two seconds

IN DEPTH

SCROLLING

jQuery is often used to scroll to a specific part of a page. We can emulate this with CSS with a `translate` and a `transition`. The idea is to move the page up, not move the scroll bar down: not quite the same, but suitable for widgets and self-contained areas.

HTML

```
<a href="#one" id="one">Section one</a>
<a href="#two" id="two">Section two</a>
<a href="#three" id="three">Section three</a>
<div class="infobox">
    <div class="sectionwrapper">
        <div class="section">Foo bar</div>
        <div class="section">Bish bosh</div>
        <div class="section">Fizz bang</div>
    </div>
</div>
```

CSS

```
.infobox, .section {
    height: 50px;
    overflow: hidden;
}
.sectionwrapper {
    transition: all 1s linear;
    -webkit-transform: translateY(0px);
    transform: translateY(0px);
}
#one:target ~ .infobox .sectionwrapper {
    -webkit-transform: translateY(0px);
    transform: translateY(0px);
}
#two:target ~ .infobox .sectionwrapper {
 -webkit-transform: translateY(-50px);
 transform: translateY(-50px);
}
#three:target ~ .infobox .sectionwrapper {
    -webkit-transform: translateY(-100px);
    transform: translateY(-100px);
}
```

With overflow hidden on your outer container (`infobox`), the panel wrapper is moved up inside to where you want to scroll. When you click an anchor, it `:targets` itself as its `id` is the same as the hash in its `href`. Using a sibling selector, it selects the `.sectionwrapper` in the `.infobox` and modifies its `translateY`. The panels should be of a fixed height, so you scroll to the correct position reliably.

```
    background: white;
}
.modal-launcher:focus {
    outline: none;
}
.modal-launcher:focus .modal {
    display: block;
}
.close-button {
    float: right;
    cursor: pointer;
}
```

When the 'Click here to open' text is clicked, its parent `.modal-launcher` receives focus, which causes the `.modal` inside to change from `display: none` to `block`. The `close-button` works by taking focus from the `.modal-launcher`. You could use `opacity` and `transition` in place of `display` for a modal that fades in.

:target

`:target` enables you to select elements via the hash in the URL. An element will match `:target` if the hash in the URL matches its ID. Like `:hover` and `:focus`, it provides the means to target a single element on the page, but the selection isn't lost when the user moves their mouse off the element or clicks elsewhere on the page. It also means your users can share a link that contains a state your CSS will respond to.

One use of the `:target` selector is as a router or tabs – you can use it to hide or display sections of your site depending on the URL given. The following would work well as a basic router:

HTML

```
<nav>
    <a href="#one">Page one</a>
    <a href="#two">Page two</a>
    <a href="#three">Page three</a>
</nav>
<section class="page" id="one">
    Welcome to my site!
</section>
<section class="page" id="two">
    Fascinating text here
</section>
<section class="page" id="three">
    Pictures of cats
</section>
```

CSS

```
.page {
    display: none;
```

```
}
.page:target {
    display: block;
}
```

You could combine this with the `translateY` method for scrolling to have links that `scrollTo` places in the DOM, or have things transition onto the screen.

:checked

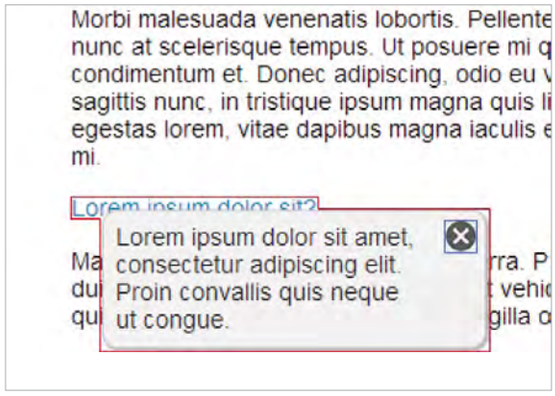
`:checked` facilitates selecting checkboxes that have been checked by the user. If you use named radio buttons, `:checked` will be exclusive; that is, selecting one of them will deselect the others. If you give them different name attributes or use checkboxes instead of radio buttons, you can target more than one element at a time.

Delegating more work to CSS instead of jQuery creates fewer bugs and improves performance

Being a form field, checkboxes can be triggered remotely using a `label` element. So, if you hide the checkbox and use a sibling selector, you can trigger any part of the DOM from anywhere else in the DOM – just put a checkbox next to the DOM tree you'd like to manipulate and you can toggle it on and off from anywhere else. The downside is that the DOM needs to be manipulated in order to satisfy CSS demands, which isn't really how the relationship is supposed to work. The following code will produce a slideshow, with no jQuery required:

HTML

```
<div class="slideshow">
    <input type="radio" name="slideshow" class="slide-
    trigger" id="one">
    <div class="slide">
        <img src="http://placekitten.com/500/200">
    </div>
    <input type="radio" name="slideshow" class="slide-
    trigger" id="two">
    <div class="slide">
        <img src="http://placekitten.com/500/200">
    </div>
    <input type="radio" name="slideshow" class="slide-
    trigger" id="three">
    <div class="slide">
        <img src="http://placekitten.com/500/200">
```



Using :focus When the blue link is clicked, the modal or tooltip is shown. The red lines show the area that has :focus

```
    </div>
</div>
<nav>
    <label for="one">1</label>
    <label for="two">2</label>
    <label for="three">3</label>
</nav>
```

CSS

```
.slideshow {
    position: relative;
    width: 500px;
    height: 200px;
    overflow: hidden;
}
.slide-trigger {
    display: none;
}
.slide {
    position: absolute;
    top: 0;
    left: -500px;
    transition: left 1s ease-in-out;
}
.slide-trigger:checked + .slide {
    left: 0;
}
```

All the slides start outside (-500px) the `overflow: hidden` container, then when a radio input next to a slide is checked, that slide is positioned inside the container. The `transition` property smoothes the animation.

CONCLUSION

In this tutorial, we've seen how you can delegate more work to CSS instead of jQuery. Keeping JavaScript light creates fewer bugs and improves performance. We've only scratched the surface here, so explore other CSS properties for yourself: just be aware of the issue of browser support.

RESOURCE

GITHUB REPOS

Check out the GitHub repositories for Dan Neame's JavaScript and CSS projects: *github.com/cham*

Download the files here!

All the files you need for this tutorial can be found at *netm.ag/filters-250*

ABOUT THE AUTHOR

ALEX DANILO

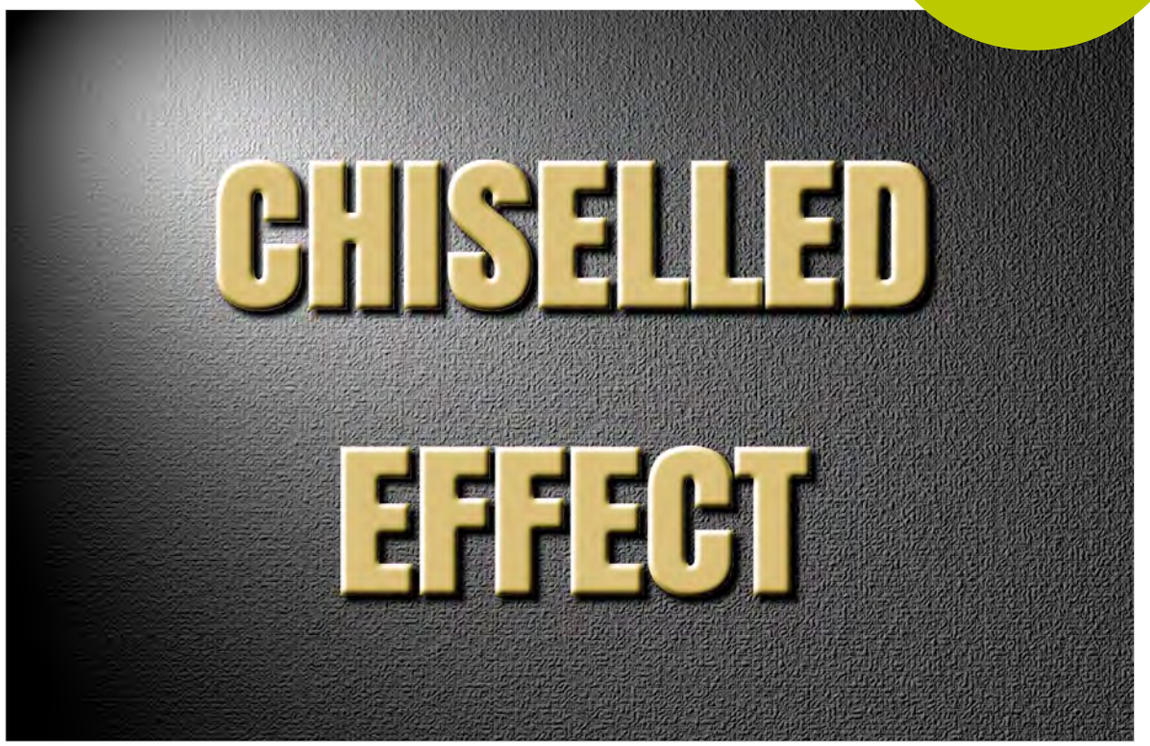
w: *alexdanilo.com*

t: @alexanderdanilo

areas of expertise:
Computer graphics, CSS, SVG, web engines

q: what's the worst present you've ever been given?

a: A green piggybank for Christmas – as an adult!



CSS

MASTER CSS FILTERS

Alex Danilo demonstrates how to use CSS filters combined with SVG filter components to help make your website stand out from the crowd

BROWSER SUPPORT CSS FILTER EFFECTS

Desktop	Mobile/tablet
18	-
3.5	4.4
-	16
15	30
6	1.0

Using CSS filters to enhance your content makes your site really sizzle. What's less well known is that you can build amazing primitives by combining SVG filter components into a fancy CSS filter that will make your site stand out from the crowd. This tutorial walks you through construction of a multi-stage filter combining lighting, blur and noise to create a stunning 3D effect on your text content. CSS filters are a great tool for changing the way your content is displayed on your site. The pre-canned effects are built up from simple building blocks and are great for blurs, colour manipulation amongst many other effects. Each of these building blocks is like a brick that, when combined together, construct a beautiful house. Even better, you can stack these bricks together yourself and make your own CSS filter effects! How? Well, we're about to dive in and build a couple of effects that you can use as inspiration for your future projects.

We'll start by looking at SVG. SVG is a vector graphics language that's built into all modern HTML5 browsers. One of its nicest features is filters. SVG includes a `<filter>` element and a bunch of different filter primitives. The SVG filter primitives are the bricks in our construction set. We're going to stack a bunch of them together and then use them on our web page content.

Once we've piled up our bricks into the filter effect we want to use, we can simply reference the markup from CSS and use CSS selectors to apply it to our content. So let's start with a fairly simple example, then later on we'll try something a bit more complex.

BUILDING A GLOW FILTER

In this first example, we're going to construct a filter that we can use to create a glow. We're going to use four different filter primitives to build our glow filter – `feGaussianBlur` (which creates a blurring effect),

feFlood (which fills an area with a solid colour), feComposite (which mixes pixels together) and feMerge (which merges the filtered content with our original text to produce the final result).

The first thing we want for our glowing text is to take a snapshot of the text and convert it to white (since later on we'll use the white text to create the glow effect). The SVG filter code to create the white text is this:

```
<filter id="white">
 <feFlood flood-color="white" />
 <feComposite in2="SourceAlpha" operator="in"
 result="white"/>
</filter>
```

This filter contains two building blocks. The first is the feFlood filter effect which fills the area affected by the filter with white (filters use a rectangular off-screen image area to create each effect in the filter chain). The second is the feComposite filter effect, which is used to combine the pixels from the original content and the output from another filter effect using some sort of combination rule.

It's important to note here that the SVG filters work in a fixed order, so they're chained together in a sequence. When we apply that filter to our content, the feFlood filter effect runs first, then the

All of the pre-canned filter effects are in fact built up from simple building blocks

feComposite effect uses the output from feFlood as its input. In the case of feComposite, it's using the 'in' operator. That's the combination rule that says 'create an image from my input using only pixels that are inside the original content'. So it's creating an intersection in maths terminology. The result of this filter is our white flooded rectangular image masked by anything it's applied to.

In this case, we get a masked result of white text. The way in which we apply this filter to our web content is to use a CSS selector to choose content that we want to be affected. An example of this could be something like:

```
.white {
        -webkit-filter: url(#white);
        filter: url(#white);
}
```

This would apply the filter to anything with a CSS class of 'white'. Of course if we tried to apply that to a blank page we'd paint white on a white background (not very useful), so let's add some background colour to see what we get:

```
.bluebg {
        background-color: lightblue;
}
<div class="bluebg">
<h1 class="white">WHITE GLOW FILTER</h1>
</div>
```

Now that we've gone through the foundation steps to build a two-stage filter and apply it to our web page text content, it's time to add some more effects.

CREATING A BLURRED EFFECT

To make our text glow effect, we need to grab the white text from our filter chain above and blur it. That'll become the basis for the glow around the text. To create the blurred version, we use:

```
<filter id="whiteblur">
 <feFlood flood-color="white" />
 <feComposite in2="SourceAlpha" operator="in" />
 <feGaussianBlur stdDeviation="4" />
</filter>
```

As you can see, we've used an additional effect – feGaussianBlur. That effect takes the white text that we created with our previous filter and makes it look fuzzy by using this markup:

```
.whiteblur {
        -webkit-filter: url(#whiteblur);
        filter: url(#whiteblur);
}
<div class="bluebg">
 <h1 class="whiteblur">WHITE GLOW FILTER</h1>
</div>
```

We need to enhance the blurred result to be a bit brighter. To do so, we'll use the feComponentTransfer effect. That's a filter effect that lets us play with the colour and alpha channels in our offscreen image in all sorts of ways.

We're going to brighten up the blurred image by modifying the alpha channel. So now our filter chain looks like:

```
<filter id="whitetransfer">
 <feFlood flood-color="white" />
 <feComposite in2="SourceAlpha" operator="in" />
 <feGaussianBlur stdDeviation="4" />
```

RESOURCE

VIDEO SESSION

Watch Alex Danilo and Alexis Deveria's video presentation, titled, 'Stunning Mobile Visualisation with CSS Filters': *developers.google.com/events/io/sessions/325944029*

Glow filter Four filter blocks combine to produce a halo-like effect

```
<feComponentTransfer>
<feFuncA type="linear" slope="3" intercept="0" />
</feComponentTransfer>
</filter>
```

You'll notice that we've used the `feFuncA` effect, which is modifying the alpha channel by multiplying its value by three (the slope argument).

We then apply that filter by using the markup as shown below:

```
whitetransfer {
        -webkit-filter: url(#whitetransfer);
        filter: url(#whitetransfer);
}
<div class="bluebg">
<h1 class="whitetransfer">WHITE GLOW FILTER</
h1>
</div>
```

All we need to do is combine the results of our filter effect with the original text we applied it to by use of the `feMerge` filter effect.

The final version of our filter chain ends up looking like:

```
<filter id="whiteglow">
 <feFlood flood-color="white" />
 <feComposite in2="SourceAlpha" operator="in" />
 <feGaussianBlur stdDeviation="4" />
 <feComponentTransfer>
   <feFuncA type="linear" slope="3" intercept="0" />
  </feComponentTransfer>
 <feMerge>
 <feMergeNode />
 <feMergeNode in="SourceGraphic" />
 </feMerge>
 </filter>
```

RESOURCE

PRESENTATION

See Alex Danilo's presentation on creating impact with CSS filters: *www.webdirections.org/resources/create-impact-with-css-filters-video-presentation-from-alex-danilo*

And then we can apply it to our markup:

```
.whiteglow {
        -webkit-filter: url(#whiteglow);
        filter: url(#whiteglow);
}
<div class="bluebg">
<h1 class="whiteglow">WHITE GLOW FILTER</h1>
</div>
```

BUILDING A CHISELLED EFFECT

This second example uses a whole slew of filter effects to create a 3D looking chiselled effect from our text content. The SVG filter effects used here include random noise, morphology, lighting and more. Don't worry, they sound daunting but are really easy to use. The way this filter works is by combining three functional steps, each of which uses a number of different filter effects. To make it work we simply do the following:

1. Convert an image into a 'bump map' (see the page opposite for more information).
2. Create an embossed look from our original text.
3. Create a drop shadow that enhances our 3D look.

CREATING THE BUMP MAP

Bump maps are used in 3D games. Think of them as a rough texture like cloth or a mountain range, where each pixel represents a different height.

To create the bump map, we put together a whole slew of filter effects. See the page opposite for the example code. Then try taking that filter and applying it to this markup:

```
.bumps {
        -webkit-filter: url(#bumps);
        filter: url(#bumps);
}
<div class="bumps">  <h1>CHISELLED EFFECT</h1>
</div>
```

Chiselled filter Lighting and bump maps combine to create a 3D-like look

CREATING A GOLD EMBOSSED LOOK

To make a nice gold coloured version of our styled text with a nice 3D look, we use some more filter effects by use of the `feComponentTransfer` effect along with some more `feComposite` and `feBlend` effects.

The `feBlend` effect applies colour mixing modes that are used in a lot of desktop applications like Photoshop and Inkscape (*inkscape.org*), which allow complex colour mixing effects to be used for all sorts of nice results. In the case of the gold filter we start by changing the fill colour of our original text to be gold using the CSS colour property. Then we apply `feBlend` using 'screen' and 'multiply', which lets us combine the colours from each layer and multiply the colour values of the pixels in each of the filter input images. The code to generate the gold embossed look is available to download at *netm.ag/gold-250*.

CREATING THE DROP SHADOW

Finally to give our styled text a really nice 3D look we can create a drop shadow using filter effects. To create the shadow we use `feMorphology` (which dilutes or erodes the input image), `feGaussianBlur` (which we used for the glow example), `feOffset` (which shifts an intermediate image around) and our trusty `feComposite` to put it all together again.

What's great about filter effects is they're only limited by your imagination

The filter effect code for the drop shadow can be downloaded from the following link: *netm.ag/drop-250*. Then we apply the result to our markup:

```
.shadowbumps {
        -webkit-filter: url(#shadowbumps);
        filter: url(#shadowbumps);
}
<div class="shadowbumps">
 <h1>CHISELLED EFFECT</h1>
</div>
```

What's great about filter effects is they're only limited by your imagination. Try all of the SVG filter effect building blocks and combine them in any way you like to create all sorts of filter chains that you can use through CSS.

Using CSS filters can make you stand out from the crowd with stunning, eye-catching visual effects to get your site noticed.

FOCUS ON

THE BUMP MAP

Bump maps are used in 3D games. We use filter effects to create the bump map:

```
<filter id="bumps">
    <feTurbulence type="turbulence" baseFrequency="0.1"
numOctaves="2" result="texture"/>
    <feMerge result="textureAndGraphic">
      <feMergeNode />
      <feMergeNode in="SourceGraphic" />
    </feMerge>
    <feColorMatrix type="luminanceToAlpha" in="texture"
                   result="textureMap"/>
    <feGaussianBlur in="SourceAlpha" stdDeviation="6"
result="blur"/>
    <feDiffuseLighting in="textureMap" surfaceScale="2"
lighting-color="white"
          diffuseConstant="1" result="backgroundDiff">
      <feSpotLight id="spotLight" x="50" y="50" z="150"
          pointsAtX="150" pointsAtY="150" pointsAtZ="0"
          specularExponent="8" />
    </feDiffuseLighting>
    <feDiffuseLighting in="blur" surfaceScale="6" lighting-
color="white"
          diffuseConstant="1.2" result="foregroundDiffAll">
      <feDistantLight id="distantLight" azimuth="-135"
elevation="40" />
    </feDiffuseLighting>
    <feComponentTransfer>
      <feFuncR type="gamma" amplitude="1" exponent="3"
offset="0"/>
      <feFuncG type="gamma" amplitude="1" exponent="3"
offset="0"/>
      <feFuncB type="gamma" amplitude="1" exponent="3"
offset="0"/>
    </feComponentTransfer>
    <feComposite operator="in" in2="SourceAlpha"/>
    <feConvolveMatrix order="3" kernelMatrix="0.0625 0.0625
0.0625
                0.0625 0.5 0.0625
                0.0625 0.0625 0.0625"
                result="foregroundDiff"/>
  </filter>
```

We've added a few new building blocks: `feTurbulence`, `feColorMatrix`, `feDiffuseLighting` and `feSpotLight`. The exact details of all the parameters to these filters are a bit beyond this tutorial, but you'll have a good idea of how you can tweak effects.

ABOUT THE AUTHOR

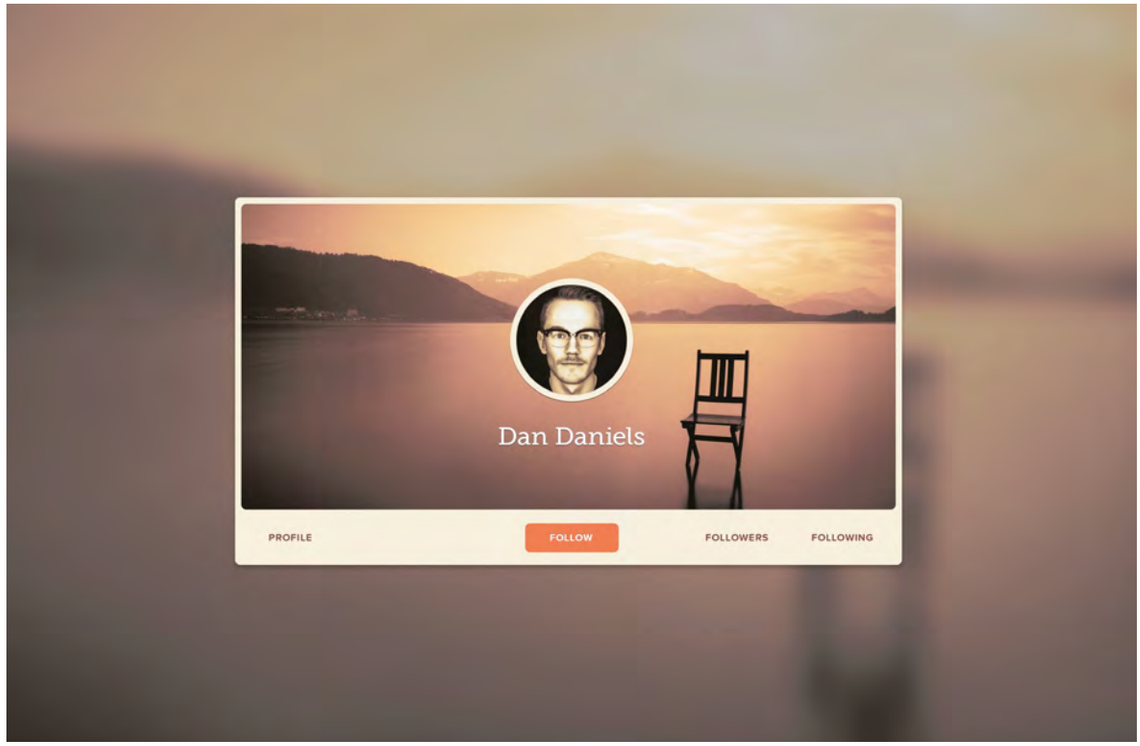
DAN TOCCHINI

w: *gridstylesheets.org*

t: @d4tocchini

areas of expertise:
All things frontend

q: Which living person do you most despise?
a: I am not sufficiently aware to say



GSS

GSS: A BETTER FUTURE FOR LAYOUT

Dan Tocchini introduces Grid Style Sheets, a groundbreaking set of polyfills that tackle CSS's longstanding layout weaknesses

CSS has made it somewhat easier for developers to work with things like typography and colours, but it never truly solved one fundamental problem: layout. CSS's dominant layout mechanism, the float, was first published over a decade ago, when most web pages were just that: static pages, not interactive, responsive web apps. It doesn't account for the layout problems today's developers face.

Unfortunately, new CSS features such as Flexbox insist on coupling layout to content, and even proposed future features won't address basic problems such as relative centre alignment. As platforms like iOS and OS X evolve to give their developers powerful new layout tools, the web platform looks increasingly unattractive to new developers. Fortunately, a better future is arriving early. It's called Grid Style Sheets, or GSS (*gridstylesheets.org*).

POLYFILLS FROM THE DISTANT FUTURE

GSS is a set of CSS polyfills for three new layout languages. In this tutorial, I'm going to introduce you to the fundamentals of the first two.

The first language is Constraint CSS (CCSS). CCSS lets developers define layouts as a set of linear arithmetic constraints and tasks an algorithm – Cassowary, the same one used in Apple's AutoLayout engine – with finding the optimal solution. The second language, adapted from AutoLayout, is Visual Format Language (VFL). VFL enables developers

VIDEO

See Grid Style Sheets in action in Dan Tocchini's exclusive screencast of this tutorial at: *netm.ag/tut1-256*

to describe one-dimensional alignments in a terse, visual style. The third language – Visual Grid Language (VGL) – is beyond the scope of this particular tutorial.

Why go through all the trouble just for better layouts? Ask yourself this: why is it so hard to centre one element inside another with CSS? There's actually a good reason: maths.

When you allow developers to describe element properties relative to each other, you end up naturally creating cyclic dependencies – logic loops – that are extremely difficult to resolve. This is where the Cassowary algorithm and constraint programming show their true power.

LAYOUTS, UNCONSTRAINED

In 1998, frustrated with traditional layout engines, former Facebook VP Greg Badros and co-author Alan Borning realised that almost any layout can be expressed as a series of linear constraints. For example, this element should be to the left of another one; this column should take up one-third of the page; this element should be centred inside this other one, and so on. To compute optimal solutions to these constraints, Badros and Borning developed the Cassowary Constraint Solving Toolkit.

Ask yourself this: why is it so hard to centre one element inside another with CSS?

Unfortunately, Cassowary remained a purely academic accomplishment for about a decade, until Apple decided that it needed a better layout paradigm for the growing number of different screen resolutions being used. The result was Cassowary-based Cocoa Auto Layout, which now powers Mac OS X and iOS.

At The Grid (*thegrid.io*) we're developing a product that requires adapting complex layouts to content and devices. CSS is simply not good enough. Instead of waiting for the W3C to adopt Cassowary, we decided that we would build our own, better, and open source future.

GSS: THE BASICS

So, how would you centre an element inside another in CSS? Forget the hacks. You can centre any element inside another with a one-liner:

```
#element1[center-y] == #element2[center-y];
```

▶

IN-DEPTH

CONSTRAINT HISTORY

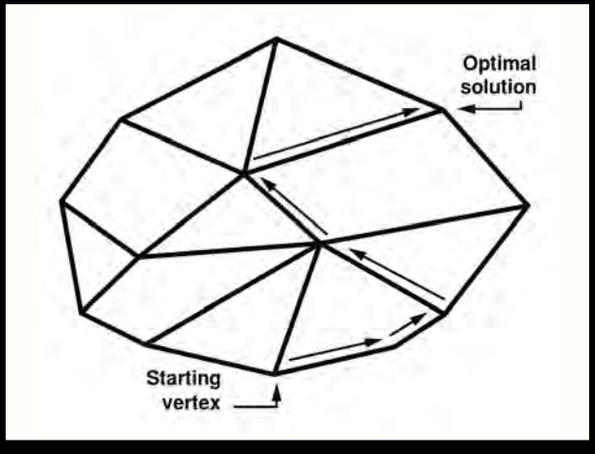
Cassowary, the algorithm powering GSS, is based on the simplex method, and has its roots in post-World War II logistics planning. In 1947, US Air Force mathematician George Dantzig realised you could more efficiently describe most logistics as a set of constraints.

For example, if a unit can deliver no more than 20 bombs, you can easily express this like so: bomb capacity <= 20. But there was still a major issue: most of the Air Force's logistics problems contained thousands of variables. Even with today's computers, it would simply take far too long to solve every possible outcome.

Dantzig's second breakthrough came when he graphed these problems as multidimensional shapes called polyhedrons. It was already known that the optimal solution to any such problem always lies at one of the corners of these shapes. But even fairly simple problems can have billions of corners.

Dantzig realised that you could find the optimal solution faster by always moving along the steeper side of any corner. Using this insight, he designed the simplex method, an algorithm that traverses polyhedron corners intelligently.

Today, nearly every corporation around the world uses a descendant of the simplex algorithm to optimise scheduling and resource distribution in everything from shipping to worker deployment, and even to stock portfolios. So why not you?



Traversing polyhedrons The power of GSS lies in a unique application of the simplex algorithm, which uses polyhedrons to compute optimal layouts

▶ Want to give all of your buttons a border-radius relative to their height?

```
button[border-radius] <= button[height] / 6;
```

With one line of GSS, you can accomplish an incredible amount of dynamism that's maddeningly impossible in CSS. Setting up GSS is simple (see *gridstylesheets.org/usage*): just include the necessary JavaScript files and use `type="text/gss"` on link tags that load .gss files or style tags with inline GSS.

TWO-WAY RELATIONSHIPS

You've probably noticed that we're using `==` instead of `=`. That's because we're writing constraints, not variable assignments. Constraints represent two-way relationships between properties. If one side changes, the other does too.

STRENGTHS

When constraints conflict, the stronger one overcomes. By default, all constraints are weak. To change this, you can add a strength, which look like CSS's `!important`. GSS comes with four strengths: `!weak`, `!medium`, `!strong` and `!require`, which makes a constraint mandatory.

So, let's refactor our border-radius example, adding a strong constraint so that the Cassowary solver knows that keeping the radius above a minimum value is more important than scaling it all the way down:

```
button {
  border-radius: <= ::this[height] / 6;
  border-radius: >= 2 !strong;
}
```

Off the grid The Grid has created a set of new CSS polyfills – including Constraint CSS – to tackle CSS's layout weaknesses

PSEUDO-SELECTORS

You might have noticed that GSS gives you a few new pseudo-selectors: `::window`, `::this` (shortened to `::`), and `::parent`.

VARIABLES

Of course, GSS supports custom constraint variables. To define one, just use it in a constraint:

```
[my-col-size] == ::window[width] / 12 !require;
.avatar[width] == [my-col-size] * 2;
```

A LITTLE TOO VERBOSE?

As powerful as one-line arithmetic constraints are, marginally complex layouts quickly become verbose. Let's say you want to build a panel with a row of four buttons and 10px of spacing. Using CCSS, it would take five lines:

```
#panel[left] + 10 == #b1[left];
#b1[right] + 10 == #b2[left];
#b2[right] + 10 == #b3[left];
#b3[right] + 10 == #b4[left];
#panel[right] - 10 == #b4[right];
```

When Apple adopted Cassowary, it developed Visual Format Language (VFL) to make it easier and more intuitive to define one-dimensional layouts. We've implemented a webified version of VFL in GSS. To accomplish that same four-button layout with VFL, all you need is one line:

```
@h |-[#b1]-[#b2]-[#b3]-[#b4]-| in(#panel) gap(10);
```

VISUAL FORMATTING

One nice thing about working with VFL is that it's visual: you can tell what's going on with just a quick glance. And, internally a VFL statement simply generates many CCSS statements – so unlike CSS with its several incompatible layout modes, GSS offers a holistic layout paradigm.

The `@h` directive defines a horizontal layout, `@v` specifies vertical ones. The pipe `(|)` defines the edge of a container element, which is specified using `in()`. Each dash `(-)` represents a gap, defined using `gap()`. Aligned elements are specified inside square brackets `([])`.

CUSHIONS

In addition to gaps, which generate equality `(==)` constraints, VFL can also generate inequality constraints `(>=, <=)` with cushions, specified using

a tilde `(~)`. A cushion tells the solver not to let the elements overlap. In our four-button example, here's how we'd express a layout where `#b3` has a weak constraint to be centre-aligned in the `#panel`, but a stronger constraint not to get closer than 10px to the other buttons:

```
#b3[center-x] == #panel[center-x];
@h |-[#b1]-[#b2]~~~[#b3]~~~[#b4]-| in(#panel) gap(10)
!strong;
```

INTRINSIC VALUES

One thing new GSS developers often forget is that constraints live in a separate world to the DOM. In our four-button example, how would you specify button width if you wanted it to be the width of its text plus padding? GSS doesn't know about those native properties unless you tell it to measure them from the DOM. To do this, you prefix properties with `intrinsic-`. In our button example:

```
button {
  width: == ::this[intrinsic-width];
}
```

With GSS conditionals, layouts can adapt not just to screen size, but to any property

As a performance best-practice, minimise the use of intrinsic properties.

@IF, OR @ELSE!

In CSS, you would use `@media` to adapt to the viewport. That's a useful tool, but we believe designers need more flexibility. That's why GSS comes with `@if` and `@else`. Think of them as a more expressive, natural replacement for `@media`. With GSS conditionals, you can design layouts that adapt not just to screen size, but to any property. Let's say you want a layout to be responsive to a panel's width:

```
@if #panel[width] >= 900 { … } @else { … }
```

PUTTING IT ALL TOGETHER

Now that you've got a feel for CCSS and VFL, let's put it together into a simple, real-world layout. Find my example at *netm.ag/gss-256*. Given the flat HTML:

```
<div id="profile-card"></div>
<div id="cover"></div>
```

▶

FOCUS ON

CENTRING

With GSS, you can centre any element relative to another using one line of code. How would you accomplish the same thing in CSS? Let's say you use absolute centring. To make it work, have to set an absolute height, use the overflow: auto hack to prevent content spillover, and use something else for Windows Phone. What about negative margins?

Unfortunately, this method isn't responsive, doesn't work with percentage-based layouts and requires you to calculate padding to compensate for the margins or use box-sizing: border-box. Negative translations? Not compatible with IE 8, forces you to use vendor prefixes, and can result in blurry rendering of text and edges. Inline block? You have to put everything in a container, recalculate left margins when font sizes change, declare the content block's width, and use a hack to support IE 9. Table cell? Non-semantic markup! Even the Flexbox solution requires a container or body styles, doesn't support IE 8 or 9, needs vendor prefixes, and has potential performance issues. Ugh.

AMBIGUITY

Constraint programming isn't without its catches. The mistake most new developers make when diving into GSS is not defining enough constraints. When your constraints are too ambiguous, you give the solver multiple ways to satisfy them, which can lead to confusion when it doesn't pick the one you had in mind. Consider the example of aligning an element to the right of the window:

```
#box[right] == ::window[right];
```

For most people, this seems straightforward: the behaviour we expect is for the box to hold its size and move to the far right of the window. In fact, this is too ambiguous. The solver could do this, but it could also decide to expand the box's width so that its right side aligns with the window's. We also don't know how it's going to treat the vertical dimension as the box stretches or moves. To solve this problem, we would define the box's size explicitly.

```
#box[right] == ::window[right];
#box[width] == 100;
#box[height] == 50;
#box[center-y] == ::window[center-y];
```

A good rule of thumb is to always constrain the horizontal and vertical size and position of your elements.

```
<div id="avatar"></div>
<h1 id="name">Dan Daniels</h1>
<button id="follow" class="primary">Follow</button>
<button id="following">Following</button>
<button id="followers">Followers</button>
<button id="message">Message</button>
```

Let's create the layout below, where the follow button has a weak constraint to stay in the middle of the panel, but stronger constraints to not get too close to the other buttons. When the panel gets too narrow, all of the buttons should vertically stack.

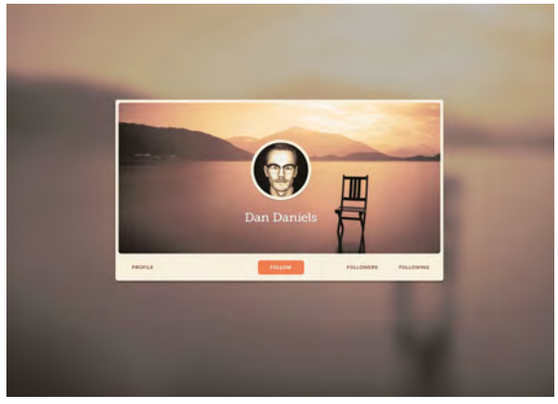
For sake of brevity, constraints will primarily use id selectors, only `structure.gss` will be covered and `texture.css` will be omitted (check out *netm.ag/demo-256* for a live demo).

First, let's define our variables and set up the card constraints:

```
[gap] == 20 !require;
[flex-gap] >= [gap] * 2 !require;
[radius] == 10 !require;
[outer-radius] == [radius] * 2 !require;

#profile-card {
  width: == ::window[width] - 480;
  height: == ::window[height] - 480;
  border-radius: == [outer-radius];
  center-x: == ::window[center-x];
  center-y: == ::window[center-y];
}
```

Let's make the avatar a 160px diameter circle:

```
#avatar {
  height: == 160 !require;
  width: == ::[height];
  border-radius: == ::[height] / 2;
}
```



Shifting buttons The orange 'Follow' button moves in response to the other buttons and to the size of the panel

PRO TIP

Use `!require` carefully and sparingly. Conflicting constraints will be impossible to satisfy and will throw an error.

Remember that GSS doesn't know anything except what you explicitly tell it. Here, we're using required intrinsic values to make sure the solver understands the size of the name element's text.

```
#name {
  height: == ::[intrinsic-height] !require;
  width: == ::[intrinsic-width] !require;
}
```

Let's add some rounded corners and constrain the buttons:

```
#cover, button {
  border-radius: == [radius];
}
button {
  width: == ::[intrinsic-width] !require;
  height: == ::[intrinsic-height] !require;
  padding: == [gap];
  padding-top: == [gap] / 2;
  padding-bottom: == [gap] / 2;
}
```

Remember that GSS doesn't know anything except what you explicitly tell it

It's time to bust out the VFL. Let's use cushions to keep the name within the horizontal bounds of the cover; we'll make the alignment `!strong` to ensure the width of the cover and panel stay large enough to satisfy this constraint.

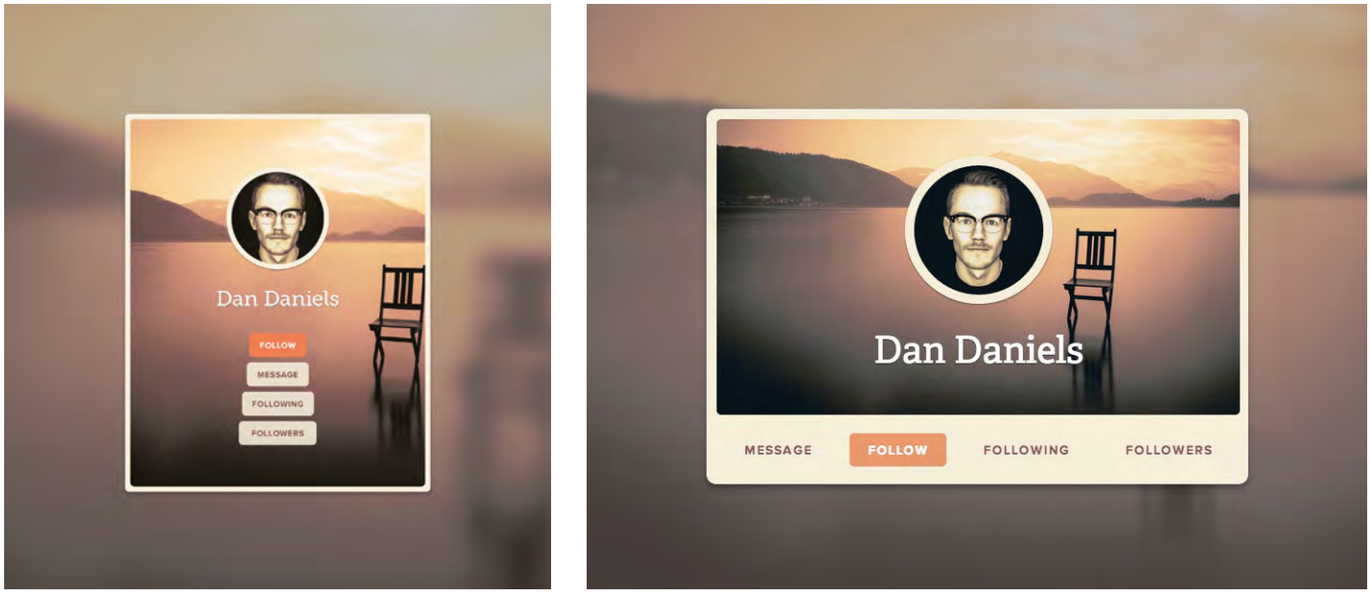
```
@h |~~~[#name]~~~| in(#cover) gap([gap]*2) !strong;
```

Let's use a GSS conditional to define two different layouts depending on the aspect ratio of the card.

```
@if #profile-card[width] >= #profile-card[height] {
```

Next, when we vertically align the avatar and name within the cover, we'll vertically centre the alignment by using `[flex-gap]` as the outer-gap. Then, we'll throw in a chain function to align the center-x of the two aligned elements with the cover's center-x.

```
  @v |-[#avatar]-[#name]-| in(#cover)
    gap([gap]) outer-gap([flex-gap])
    chain-center-x(#cover[center-x]);
```



New layouts Narrow (left) and wider versions. Notice how the different elements adapt and rearrange to fit the different screen sizes

Now, we'll stretch the cover horizontally across the profile-card with a 10px gap:

```
@h |-10-[#cover]-10-| in(#profile-card);
```

Next, we'll align the follow button below the cover image within the card. Notice how we can explicitly set a 10px gap between the cover and card, while using the larger `[gap]` variable for the other gaps on the page:

```
@v |-10-[#cover]-[#follow]-| in(#profile-card) gap([gap]);
```

Again using one line, let's centre the follow button horizontally relative to the card:

```
#follow[center-x] == #profile-card[center-x];
```

Now we can lay out our row of buttons inside the card and chain the button tops. By using cushions around the follow button, it can slide around to satisfy the centre constraint if there is space for it to do so:

```
@h |-[#message]~~~[#follow]~~~[#following]-[#followers]-|
  in(#profile-card)
  gap([gap])
  chain-top
  !strong;
}
```

Finally, let's finish our conditional. When the card is taller than it is wide, we want to simplify the layout by vertically stacking all of the buttons inside the cover photo:

```
@else {
  @h |-10-[#cover]-10-| in(#profile-card);
  @v |-10-[#cover]-10-| in(#profile-card);
  @v |-[#avatar]-[#name]-[#follow]-[#message]-
[#following]-[#followers]-|
    in(#cover)
    gap([gap]) outer-gap([flex-gap])
    chain-center-x(#profile-card[center-x]);
}
```

TRUE SOURCE-ORDER INDEPENDENCE

Do you see how the panel and buttons take on a parent-child relationship, even though there's no nesting of the HTML elements? With GSS, you can create layouts completely independently from DOM structure. This is true source-order independence. Try doing that with old-school CSS, or even Flexbox!

NOT JUST AN EXPERIMENT

GSS is still under development and there are a few missing pieces to be added, but we're so confident about it that we're using GSS to build our flagship consumer product, The Grid.

We think GSS has the potential to be the future of web layout, and we'd love to see it standardised. However, it's been almost 15 years since Badros proposed CCSS, and we need this now. The good news is that even though the web's layout mechanisms haven't evolved all that much since 1998, browsers have.

We believe that if something as groundbreaking as GSS is possible to make happen in today's browser, we ought to do it instead of waiting for standards to catch up. You shouldn't wait, either.

VIDEO

Watch Dan Tocchini's 'Grid Style Sheets: Layouts Not Possible With Tomorrow's CSS' talk at Fluent 2014: *netm.ag/fluent-256*

Sass has matured from 'just' a preprocessor to a power tool that enables programming in CSS. **Roy Tomeij** introduces Sass' brand new superpowers

ass: you have heard about it. You've read about it online, in books and in magazines, and probably even given it a try yourself. Before we get to the practical advice though, let's take a step back and see where Sass came from and where it's going. In 2006, Hampton Catlin came up with the idea for a scripting language that would add 'syntactic sugar' to work around the many limitations of CSS.

Catlin managed to rope Natalie Weizenbaum in to help him build it, and the first public version was released in 2007. After a while Chris Eppstein came on board, and Sass is currently maintained by Weizenbaum and Eppstein.

Sass first found traction within the Ruby on Rails community, and it is now the de facto standard for style sheet authoring in Rails. With the birth of various programs for compiling Sass using a graphical user interface instead of needing to use the command-line, Sass has become widespread in the frontend development community as well. While there aren't any conclusive statistics, an analysis of preprocessor use on CodePen in 2013 assigned Sass an 80 per cent share of the market (*netm.ag/preprocessor-257*).

Sass has not only influenced similar tools like LESS and Stylus, it has also sparked debate among those that define what CSS is: the CSS Working Group at the W3C. While it may have been inevitable for the working group to eventually have to start to consider features such as CSS variables, the use of preprocessors definitely helped put it on the agenda. The existence of Sass is actively shaping what CSS is going to look like.

A lot of what Sass brings are visions of the future for CSS though, as right now we need to work with browsers that only support the current CSS specification. Although a native solution would be best, Sass helps us get around that limitation and it will remain our saviour for the foreseeable future. ▶

Author
ROY TOMEIJ (@roy) Roy is a Dutch frontend architect, Sass evangelist, trainer and speaker. He's writing a course about "turning Sass up to eleven" at *advancedsass.com*

Illustration
JAMES OCONNELL James (*james-oconnell.com*) is a designer and illustrator based in Manchester

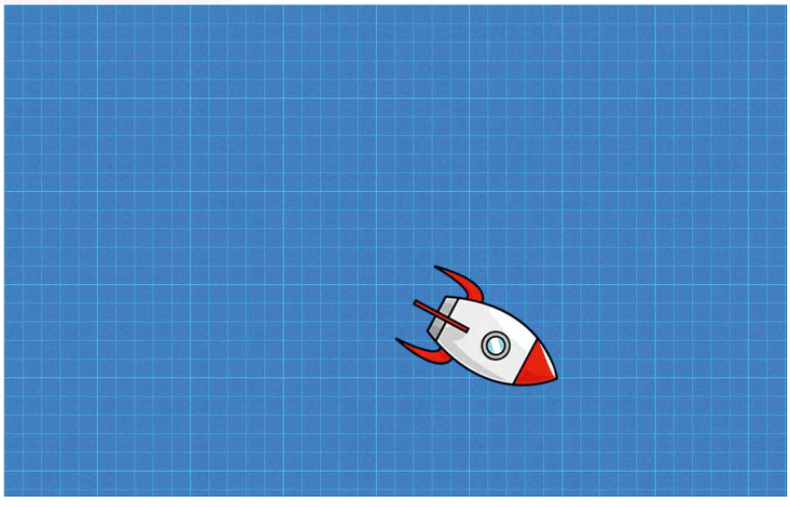
CSS with SUPER POWERS

Join the Sass revolution: we reveal how to make the most of the best CSS extension language

CSS IS A MACHINE LANGUAGE

While the above isn't actually true – CSS code uses a human readable format – it often feels like an ancient relic that lacks essential features. Preprocessors like Sass try to solve some of these issues by adding syntactic sugar, enhancing the existing CSS syntax.

All Sass will eventually be compiled (or 'converted') into CSS, and that's what gets sent to the browser. This means that with Sass we're still limited to doing what CSS can, but the way in which we do it is what warrants the existence of preprocessors. They make for a much more pleasurable coding experience, and turn CSS into a true machine language by compiling to code you're never going to read.



SASS DOESN'T CREATE BAD CODE

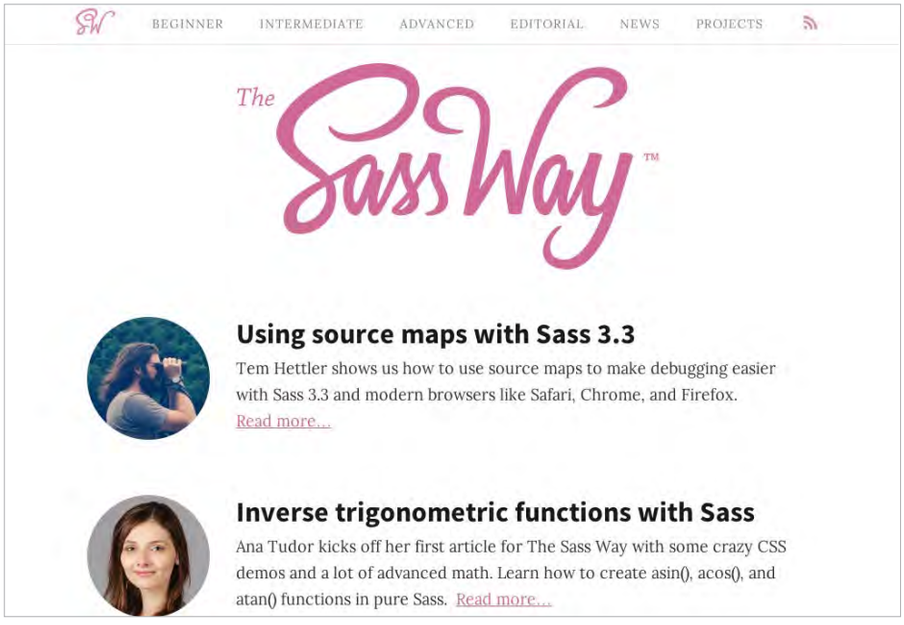
There's always a debate about quality of code when using Sass. While I'm sure there's a lot of poor CSS code out there that was compiled by Sass, it's too easy to blame the tool. Sass is powerful, and with great power comes great responsibility. Those complaining about compiled code having selectors that are too specific will probably find the author nested their Sass too deep. If you notice there's a lot of repetition in the compiled CSS, this could be due to the author using mixins where the extend directive should have been used.

Treating CSS as a machine language doesn't mean you don't need to be critical of your code. While you're not writing CSS, it pays off to take a look at the CSS that's being outputted once in a while. Make sure Sass isn't a 'magical' part of the process for you – you need to understand how the compiler works. When you know how Sass transforms what you're writing into CSS, you can predict where it's going to create code that you would have written differently by hand, and act upon that knowledge.

PROGRAMMING IN YOUR CSS

It's hard to compare Sass to what's commonly referred to as programming languages, but it is a true programming language by definition. It bases the compilation to CSS on instructions given, such as taking a value from a variable or using control structures.

The relative simplicity of Sass is what makes it easy to get started, and using the more advanced features isn't too scary either. You don't need



Above The Sass Way has articles ranging from simple tutorials and complex coding to editorials: *thesassway.com*

Facing page, top The creator and maintainers of Sass (from left): Natalie Weizenbaum, Chris Eppstein and Hampton Catlin (photo by Jed Foster)

Facing page, bottom With Sass, it's fairly easy to turn ASCII Art into a CSS animation, as explained in this article: *roy.io/crazysass*

> SASS IS POWERFUL, AND WITH GREAT POWER COMES GREAT RESPONSIBILITY

a computer science degree to program in Sass, even when going beyond the basic features of nesting and mixins. Sass will save you a bunch of time and make you a happier developer, either way.

SHOW ME THE CODE

Enough talking about Sass, let's get our hands dirty. Navigate to any of your projects and open a CSS file. Rename its extension to .scss. Congratulations, you're looking at your first Sass code! Because any valid CSS is also valid Sass, refactoring existing code to Sass is easy. You can make incremental changes without spending days of redoing your work.

When turning a CSS file into Sass, the most common features to get started with are variables (*netm.ag/variables-257*) and mixins (*netm.ag/mixins-257*). The vast majority of people leave it at that, which is a shame. Sass can do so much more, and it's worth thinking creatively about how you could solve your recurring annoyances with CSS. For example, how about a more pleasurable way of ▶

TOP 5 SASS MISTAKES

1 NESTING TOO DEEP

While it's tempting to translate your nested HTML structure to Sass, it creates overly specific selectors that will hurt you in the long run. As per the 'inception rule' (*thesassway.com/beginner/the-inception-rule*): don't nest more than four levels deep.

2 USING MIXINS INSTEAD OF EXTEND

If you find yourself using mixins that don't take arguments or a `@content` block, you're adding code bloat to your CSS. You will be repeating code in your compiled CSS. As a rule of thumb, use the `@extend` directive (*netm.ag/extend-257*) when there's nothing dynamic about your mixin.

3 ALWAYS USING VENDOR PREFIXED MIXINS

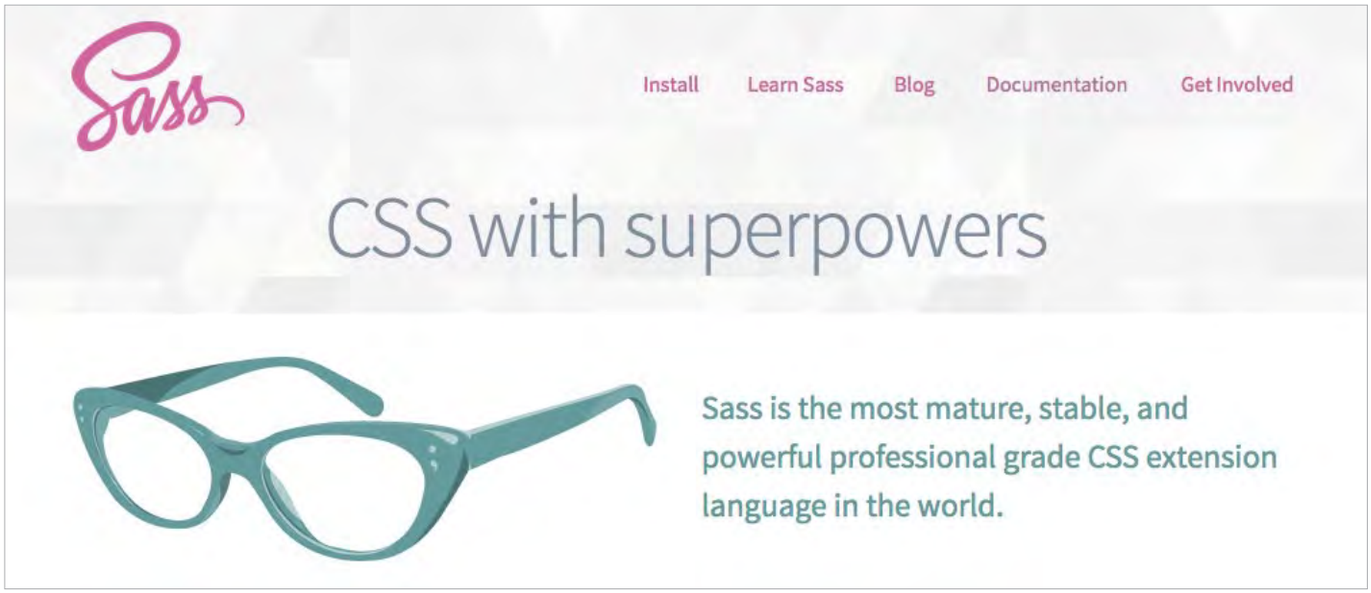
One of the great things about the Sass ecosystem is the large number of frameworks and extensions – for instance those designed to help with CSS3 vendor prefixed properties. Don't resort to these by default though: the border-radius property for instance hasn't needed prefixing for a while, so don't bloat your code by continuing to use a mixin for this.

4 COMMITTING CSS FILES TO VERSION CONTROL

Commit your source files, which in case of Sass means your .sass or .scss files, and not your compiled .css files. It will only tempt others to make changes in the wrong files, and having those changes overwritten later.

5 NOT PETTING A KITTEN

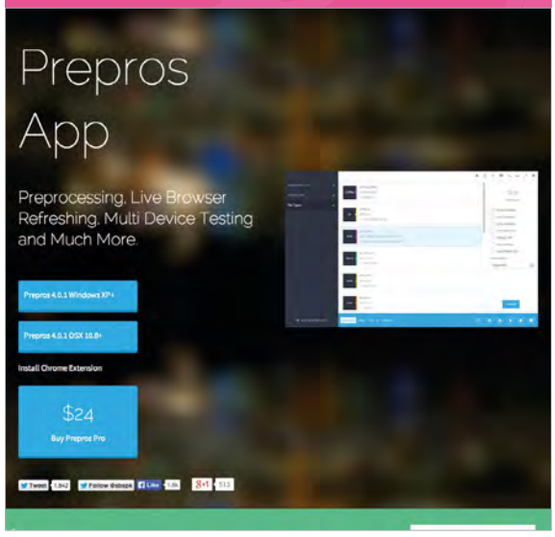
Sass is licensed under the MIT License, which makes petting a kitten more a suggestion than an actual requirement. The same goes for paying Catlin a compliment or buying Weizenbaum some jelly beans, although they would appreciate it. Seriously, it's in the author's section of the ReadMe (*netm.ag/readme-257*).



TURNING SASS INTO CSS?

There are lots of tools out there to help you get up and running with Sass. The easiest to set up are programs with graphical interfaces like Prepros (*alphapixels.com/prepros*), a free app for Windows and OS X; Compass.app (*compass.kkbox.com/Windows*) for OS X and Linux; or CodeKit (*netm.ag/codekit-257*) for OS X. If you're already using Grunt or Gulp, it should be easy to integrate Sass compiling into your workflow. If you're not afraid of the command-line, you can use the official Sass gem directly.

If you are just looking to play around with Sass for a bit, try *sassmeister.com*. It will also compile your Sass gists on the fly, making it great for sharing or showing off your code.



Top Last year the Sass website got a brand new design by Team Sass, together with a new logo

Facing page Compass was the first framework built on top of Sass, with some unique features, like automated spriting: *compass-style.org*

working with media queries, supporting high pixel density devices, grids or even right-to-left layouts? Sass has got your back.

The examples in this article probably require an intermediate knowledge of Sass, unless you're a particularly quick learner. After doing some more beginner-level tutorials on the web, you should be good to go.

MIXINS FOR RETINA DEVICES

Let's start with a simple mixin to target Retina devices. Targeting such devices takes a pretty specific media query (abbreviated for print) that's impossible to remember, and copy and pasting this would become a nightmare. Mixins to the rescue, with the added benefit that you can now use nesting to add context to your code.

```
@mixin retina {
  @media
    (min--moz-device-pixel-ratio: 1.3),
    (...), (...), (...),
    (min-resolution: 1.3dppx) {
      @content;
  }
}

.foo {
  background-image: url("file.png");

  @include retina {
    background-image: url("file@2x.png");
  }
}
```

Find the full, editable example at *netm.ag/mixinforretina-257*.

While the above works well, there definitely is room for improvement. Maybe you want to supply the filename as an argument (*netm.ag/filename-257*) instead of a `@content` block (*netm.ag/content-257*): *netm.ag/sasscode-257* or set the backgrounds all at once with images in separate folders (*netm.ag/backgrounds-267*)? Whatever output works for you, a mixin can likely be constructed for it.

FUNCTIONS FOR GRIDS

Sass comes with a whole bunch of built-in functions (*netm.ag/sass-252*), and if you use the Compass framework (*compass-style.org*), you'll have many more to work with. You can also write your own functions (*netm.ag/functions-257*), though.

For the next example, let's say we're building our own grid framework. Our grid consists of columns that are 60px wide, with with 20px wide gutters (the vertical space between columns).

> SASS COMES WITH A BUNCH OF BUILT-IN FUNCTIONS, AND YOU CAN ALSO WRITE YOUR OWN

If we have an HTML element we want to span across three columns, it needs to be 220px wide: 3 x 60px + 2 x 20px = 220px. That's three columns, plus the two gutters in-between. We won't just use this value, because we may want to change our grid dimensions later. That's why we'll make things configurable by storing this in some variables:

```
$column-width: 60px;
$gutter-width: 20px;
```

Now it's time to come up with a function to automate the above maths. I'm going to call it `span-columns`, make it accept a single argument (the number of columns) and return the output of the calculation:

```
@function span-columns($columns) {
  @return
    $columns * $column-width
    +
    ($columns - 1) * $gutter-width;
}
```

Above Looking for proof on how production-ready Sass is? Even Apple uses Sass on its website!

Facing page SassConf is the only conference in the world dedicated to Sass, and takes place in NYC in October: *sassconf.com*

The calculation is the same as the one we handcrafted, with the only difference being that we deduct 1 from the number of columns, in order to calculate the number of gutters. Now if we want to set the width of an element that needs to span three columns, we simply use our custom function like this:

```
.foo {
  width: span-columns(3);
}
```

When we change the dimensions of the columns or gutters later, we only need to change the two variables on top and all our elements will fit the new grid automatically. You can play around with it here: *netm.ag/grids-257*.

MANAGING MEDIA QUERIES

As you may have noticed in the example with the Retina media query, Sass 'bubbles up' media queries. This means you can nest them within a selector, which isn't valid CSS, and Sass will work the rest out for you. Combine that with mixins and something I call 'breakpoint configuration' and managing media queries becomes a lot easier.

Imagine we want to target phone and tablet devices in our CSS. We don't want to be bothered with pixel sizes, so we're going to use named breakpoints. In its simplest form, we can make use of the `@if` control directive in a mixin called `respond-to`, like this:

```
@mixin respond-to($device) {
  @if ($device == 'tablet')
  {
    @media (max-width: 768px) {
      @content;
    }
  }

  @else if ($device == 'phone')
  {
    @media (max-width: 320px) {
      @content;
    }
  }
}
```

Now when we want to only apply styling to a phone viewport size of 320px wide, we can call this mixin from any selector, turning the background colour from red to blue:

```
.foo {
  background: red;

  @include respond-to('phone') {
    background: blue;
  }
}
```

You can find the full, working example at *netm.ag/example-257*.

UNREADABLE CODE

This could potentially lead to a whole lot of control structures, which could mean turning unreadable CSS code into unreadable Sass code.

We can separate the breakpoint configuration from the mixin by, for instance, making use of maps (*netm.ag/maps-257*), – a new data type introduced in Sass 3.3. Or, if you're on an older version of Sass, you could mimic this behaviour by iterating over a nested list.

```
$devices: (
 tablet: 768px,
 phone: 320px
);

@mixin respond-to($device) {
 @if (map-has-key($devices, $device))
 {
  @media (max-width: map-get($devices, $device)) {
   @content;
  }
 }
}
```

> SURE, YOU COULD HAVE WRITTEN ALL OF THIS BY HAND ... BUT THE FUTURE OF CSS, FOR NOW, IS SASS

This 'configuration map' can be easily extended without ever having to touch the mixin, so you could move it to a settings.scss file if you want to centralise your configuration variables. Here's a more extensive example using maps for this: *netm.ag/maps2-257*.

THE TIP OF THE ICEBERG

Although all of the above is pretty awesome, it's just the tip of the iceberg of what Sass can do. Sure, you could have written all of this compiled CSS by hand. Developers have been doing it since 1996 and it still serves many well. The future of CSS though, for now, is Sass. Join the Sass revolution and become happier and more productive!

RESOURCES

OFFICIAL SASS WEBSITE *sass-lang.com*
Documentation, branding and more

SASSMEISTER *sassmeister.com*
Playground for trying out and sharing code

THE SASS WAY *thesassway.com*
In-depth tutorials and editorials

SASSNEWS *sassnews.com*
Weekly newsletter with the latest on everything Sass

SASS.IO *sass.io*
Irregularly updated, but a great collection of Sassy links

SACHE.IN *www.sache.in*
Discover Sass and Compass extensions

HUGO GIRAUDEL'S BLOG *hugogiraudel.com*
Giraudel, the Sass writing machine

SASS BITES PODCAST *sassbites.com*
Micah Godbolt interviews and talks code

TEAM SASS *github.com/team-sass*
Great collection of Sass related tools

SASSCONF *sassconf.com*
The only conference dedicated to Sass (check out the free videos at *vimeo.com/sassconf/videos)*

SASS ANIMATION GRID *roy.io/crazysass*
Fun experiment by the author that goes crazy with Sass and animations

Download the files here!

All the files you need for this tutorial can be found at *netm.ag/sass-249*

ABOUT THE AUTHOR

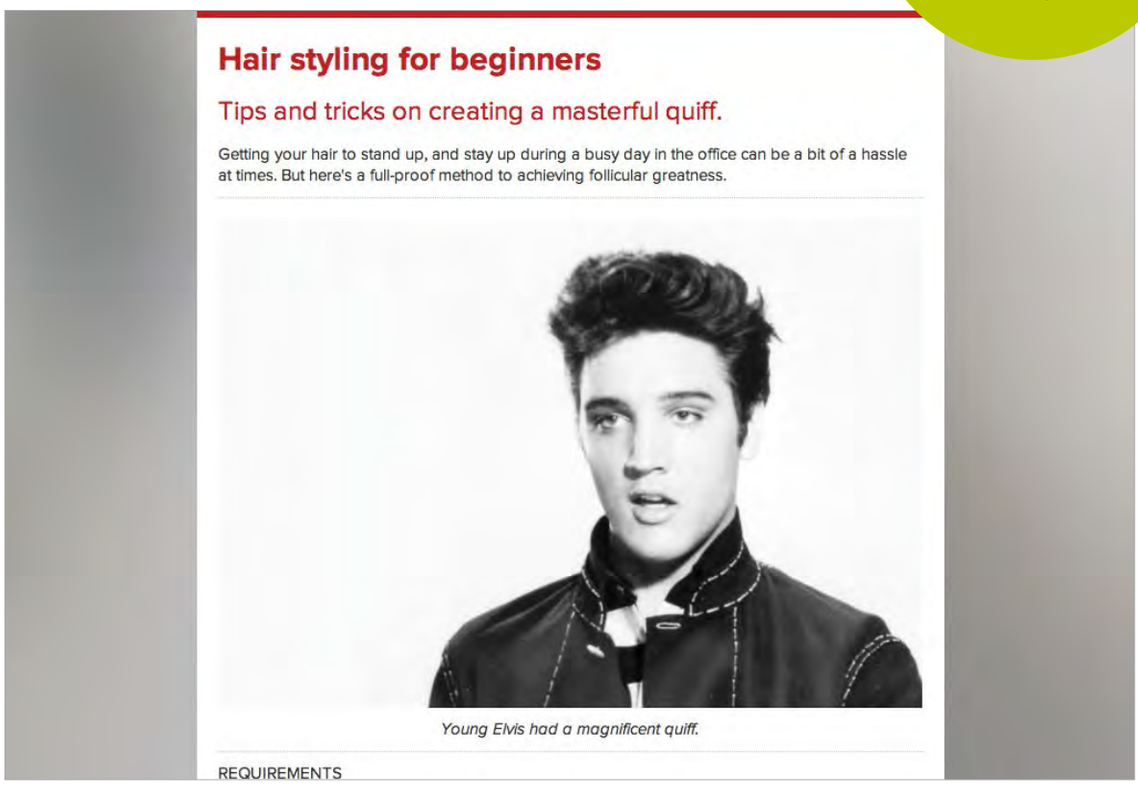
MAT HAYWARD

w: *erskinedesign.com*

t: @mathaywarduk

areas of expertise:
Frontend development

q: what's the coolest thing you do offline?
a: Hang out with rock stars (my mate is in the band The Damn Heavy)



CSS

WRITE MODULAR CSS WITH SASS AND BEM

Mat Hayward explains how to use Sass and BEM syntax to create a 'living framework' to adapt to suit any frontend development project

BROWSER SUPPORT
EM UNITS

Desktop	Mobile/tablet
1	7
1	2.1
6	8
8	1
1	1

While there are already many frontend frameworks that are available for free, it's often preferable to write the code yourself. In this tutorial, we will explore how to use the Sass CSS preprocessor and the BEM methodology to write a 'living framework' that can be adapted to suit any development project.

Sass enables us to write CSS in small, easy-to-navigate modules, meaning we can start with base styles and bolt on any components we may need for the system we're currently building.

We can also make our code easier to understand using well-constructed comments and clear naming conventions for our classes and variables. The BEM methodology gives us this clarity. BEM stands for Block-Element-Modifier and is designed to help modularise frontend development by breaking everything into blocks containing elements, then using modifiers to tweak them.

To apply the BEM methodology, we give each element within a block a class. Consider an unordered list:

```
<ul class="list">
  <li class="list__item">Item 1</li>
  <li class="list__item">Item 2</li>
  <li class="list__item list__item--end">Item 3</li>
</ul>
```

The unordered list has the block class `list` while each list item has the element class `list__item`. Only the final list item has the additional modifier class `list__item—end`. Each class follows the BEM syntax:

```
.BLOCK{__ELEMENT[--MODIFIER]}
```

GETTING STARTED

Set up the project by creating two directories: `assets` and `static`. The static directory will contain the compiled CSS and everything else that must be deployed, while the assets folder will contain any resources that don't need to be deployed (in this instance, our Sass files).

With Sass installed on an environment, it's necessary to ask it to `watch` for changes in the SCSS files and update the static CSS files accordingly. This is a pretty simple one-liner in a terminal window:

```
sass --watch assets/sass:static/css
```

Sass should have created a `screen.css` file in our `/static/css/` directory. Although at this point the file is empty, we will link to it from our HTML. (The `assets/sass` directory must exist for this to work).

Although we want our modules to be independent, there are always variables, functions and mixins that

Using BEM as a basis for variable names will reduce the number of variables used

are used by multiple modules: for example, a mixin for outputting rems with a pixel fallback, or site-wide colour variables. With this in mind, we need to create `_base.scss` in our `/assets/sass` directory and include them there.

Using BEM as a basis for variable names will reduce the number of variables used and help other developers who might pick up the framework later on. By following this convention, we can add variables for colour and text size to our `_base.scss` file. For example:

```
$color__primary--light: #DD0000 !default;
$color__primary: #C10000 !default;
$color__primary--dark: #990000 !default;
$base__font-size: 15 !default;
$base__line: 20 !default;
```

And a simple function for calculating em sizing:

CASE STUDY

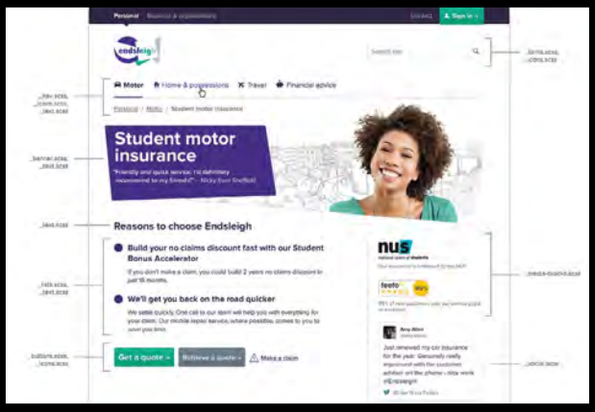
THEORY INTO PRACTICE: ENDSLEIGH.CO.UK

At Erskine, we've been putting modular frontend concepts into practice on our latest projects, including the relaunch of the website of UK insurance provider Endsleigh (*endsleigh.co.uk*).

You will be forgiven for thinking that there's nothing cutting-edge about how the website looks, but the work involved in putting it together and the attention to detail was extensive. We spent months working on content strategy and organising Endsleigh's vast array of products into something understandable and manageable before creating wireframes and a library of designed elements and modules that could be used throughout the site.

It was appropriate to continue the same depth of thought through to the build process; a little care here saved us a lot of time later on. By arranging our elements into modular groups, in much the same way that the content team organised the products, we could develop those groups into Sass modules that would later make the location of specific styles we were searching for more obvious. When the site reached its minimum viable product and was launched, the project contained some 40-odd Sass modules.

This way of working did take more development time at the outset, and getting into the mindset of writing reusable modules was difficult to begin with, but even coming back to the site to make updates weeks after its launch, we've found the organisation of the file system intuitive. That, coupled with the ability to reuse some of the more ambiguous modules on other projects, will save us time and stress in the future.



Project page Elements are divided into Sass modules using multiple module styles

FOCUS ON

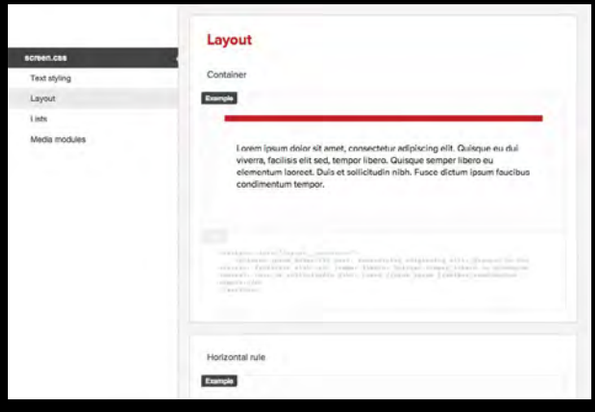
USING A LIVING STYLE GUIDE

If you've ever used Bootstrap, you will probably have seen the super-useful style guide detailing every component available through the framework, how to use them, and how they will appear (*bootstrapdocs.com*). This is a great example of a 'living style guide': a document that shows all the components of the framework created and styled by the framework itself.

A feature like this makes it possible for any frontend developer to get started on a project without needing too much face-to-face time with other developers on the team during the mandatory handover period.

It's quite simple to do something like this manually, updating it as a product grows, but the additional time will be noticeable at the business end of the build stage. Thankfully, there are a number of open-source projects that can help you to do the same with your product, including KSS (*warpspire.com/kss*), Pattern Primer (*github.com/adactio/Pattern-Primer*) and Pears (*pea.rs*). But the method I currently favour is to use the Kalei style guide (*kaleistyleguide.com*). Kalei uses JavaScript to read through your CSS and "generate Bootstrap-like documentation" on the fly. Descriptions and HTML examples are then just simply added to the CSS using markdown in the comments.

To test it out, I added it to the mini-framework I made for this tutorial. Although I encountered some issues to begin with – for instance, the clash between the styles used for the style guide and the reset used within my framework – I got it working. For smaller projects, I'd guess it wouldn't be a necessary addition, as the time it takes to implement properly will outweigh the time saved, even in the long term. However, I would definitely be happy to use this, or something similar, on larger projects with multiple developers working together.



With Kalei How to implement the layout container in HTML for the style guide

```
@function calc-em($target, $context: $base__font-size) {
  @return ($target / $context) * 1em;
}
```

Next, create _config.scss, which will later be used to override the default module variables, and screen.scss, the master Sass file. This will import the modules to be compiled into CSS for use on your website. Start by simply importing the base and config modules:

```
@import "base";
@import "config";
```

Rather than asking Sass to create multiple CSS files and linking each one into our HTML document, use the Sass @import directive to combine all the modules into one master CSS file. This reduces the amount of HTTP requests required, thereby improving performance.

ADDING MODULES

Create a directory in /assets/sass called modules. This will house all the additional modules you add

Use the Sass @import directive to combine all the modules into one master CSS file

to your site for styling different forms of content. First off, add a CSS reset module (_reset.scss). Paste in the reset code of your choice and import it into the master file in the same way as before.

When you save the changes to the master file, notice that your Terminal process has detected changes and overwritten the screen.css file used by your web page. Opening the file, you can see the contents of the reset have been ported across.

At this point, you can begin to see the workflow in action. Being able to import snippets of code into a master file means we can easily toggle individual modules on or off as needed.

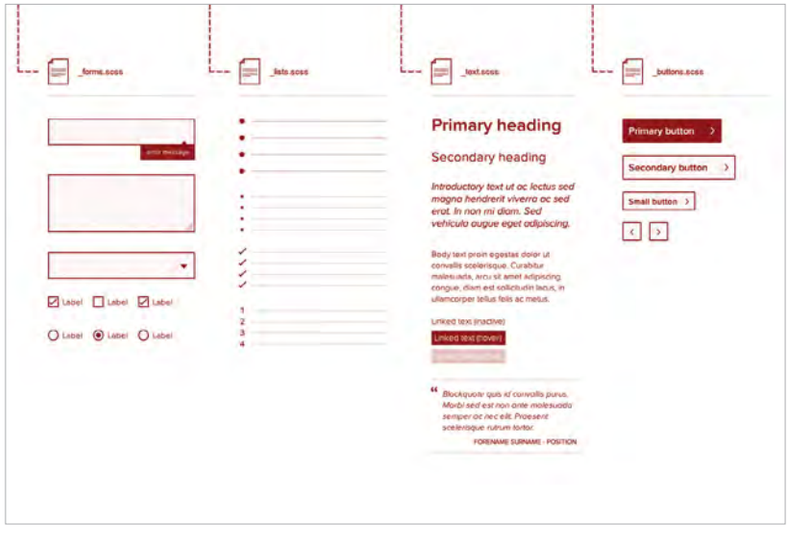
We can create a module called _text.scss and then start adding base styles for headers, paragraphs, and so on:

```
h1 {
  font-size: calc-em(36);
  line-height: (40/36);
  font-family: "Helvetica Neue", Helvetica, Arial, sans-serif;
  margin-bottom: calc-em(10, 40);
```

```
    font-weight: bold;
    color: $color__primary;
}
h2 {
    font-size: calc-em(24);
    line-height: (28/24);
    font-family: "Helvetica Neue", Helvetica, Arial, sans-serif;
    margin-bottom: calc-em(15, 40);
    color: $color__primary;
}
h3 {
    font-size: calc-em(20);
    line-height: (22/20);
    font-family: "Helvetica Neue", Helvetica, Arial, sans-serif;
    margin-bottom: calc-em(15, 40);
}
p {
    font-size: calc-em(15);
    line-height: (20/15);
    font-family: "Helvetica Neue", Helvetica, Arial, sans-serif;
    margin-bottom: calc-em(10, 15);
}
```

As there is quite a lot of repeated CSS here, we can abstract this out into variables and a mixin for text styling, reducing the amount of code used and making any future tweaks quicker.

```
$text__family-sans:  "Helvetica Neue", Helvetica, Arial, sans-serif !default;
$text__size--alpha: 36 !default;
$text__size--beta: 30 !default;
$text__size--gamma: 24 !default;
$text__size--delta: 15 !default;
$text__line--alpha: 40 !default;
$text__line--beta: 36 !default;
$text__line--gamma: 30 !default;
$text__line--delta: 20 !default;

$text__colour--base: #333333 !default;
$text__colour--alpha: #990000 !default;
$text__colour--beta: #990000 !default;
$text__colour--gamma: #333333 !default;
$text__colour--delta: #333333 !default;
@mixin text($size, $line, $margin: 0, $family: $text__family-sans) {
    font-size: calc-em($size);
    line-height: ($line/$size);
    font-family: $family;
    margin-bottom: calc-em($margin, $size);
}
html {
    font-size: $base__font-size * 1px;
}
body {
```



On demand Self-contained module files set out base styles for content types. They're used only when needed

```
    @include text($base__font-size, $base__line);
}
h1 {
    @include text($text__size--delta, $text__line--delta, 15);
    font-weight: bold;
    color: $text__colour--alpha;
}
h2 {
    @include text($text__size--beta, $text__line--beta, 15);
    color: $text__colour--beta;
}
h3 {
    @include text($text__size--gamma, $text__line--gamma, 15);
    color: $text__colour--gamma;
}
p {
    @include text($text__size--delta, $text__line--delta, 10);
    color: $text__colour--delta;
}
```

The approach we're using for naming colour variables may not be practical when trying to remember which colours are used in context to a design. To counteract this we have used module variables with more contextual names. Each variable is given a `!default` flag, which allows us to override them in our `_config.scss` file if we prefer:

```
$text__family-sans: "Proxima Nova", "Helvetica Neue", Helvetica, Arial, sans-serif;
$text__size--beta: 30;
$text__line--beta: 36;
$text__colour--base: $color__neutral--dark;
$text__colour--alpha: $color__primary;
$text__colour--beta: $color__primary;
```

RESOURCE

INTRO TO SASS

For an introduction to Compass and Sass, try Chris Coyier's video screencast for all the information you need to get started: *css-tricks.com/video-screencasts/88-intro-to-compass-sass*

```
$text__colour--gamma: $text__colour--base;
$text__colour--delta: $text__colour--base;
```

You may ask what the point of declaring then overriding these variables is, and, for one-off use, there isn't one. But what we have now is the beginnings of a modular framework that we can use as a basis for any frontend project, adding and importing more modules as we add more elements to our system. Redeclaring variables simply reduces the need for dependency between modules and gives us a cleaner starting point for each new project.

MAKING CODE EASIER TO UNDERSTAND

You could be forgiven for thinking that any developer could now pick up your work and understand it almost instantly, but that's not always the case. Fortunately, there are ways to make your code even easier to understand. Firstly, Sass comments are not compiled, so will not add any weight to our compiled CSS file, so we can write as much as we like to be extra helpful. For instance, I always like to add comment blocks to mixins and functions. Secondly, we can use the BEM methodology. We could create a Sass module for lists where inheritance is obvious:

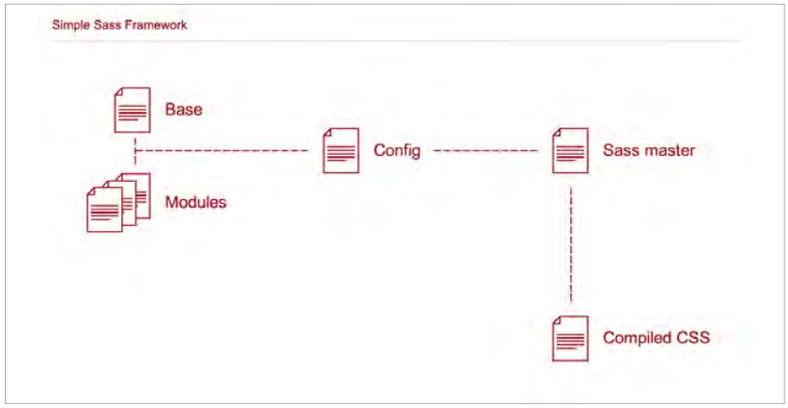
```
.list {
  padding-left: calc-em(10);
}
.list__item {
  margin-bottom: calc-em(10);
}
.list__item--end {
  margin-bottom: 0;
}
```

RESOURCE

OOCSS AND CSS

Read Chris Webb's article: 'Relationship between OOCSS and CSS preprocessors': *netm.ag/oocss-249*

To add more bullets to this list, we could then create a modifier and include that class on the unordered list element:



The framework The configuration file alters the base and modules files before being imported into the master

```
<ul class="list list--bullet">
  <li class="list__item">Bulleted list item 1</li>
  <li class="list__item">Bulleted list item 2</li>
  <li class="list__item list__item--end">Bulleted list item 3</li>
</ul>
.list--bullet {
list-style-type: disc;
padding-left: calc-em(25);
}
```

PUTTING IT ALL TOGETHER

Using the styles we have created, let's put together a very simple page based on one of my favourite subjects – the quiff:

Issues surrounding compounding font-sizes can crop up when we nest HTML elements

```
<article class="layout__container">
    <header>
      <h1>Hair styling for beginners</h1>
      <h2>Tips and tricks on creating a masterful quiff</h2>
    </header>
    <p>Getting your hair to stand up, and stay up during a busy day in the office can be a bit of a hassle at times. But here&rsquo;s a full-proof method to achieving follicular greatness.</p>
    <figure class="media media--full">
      <img class="media__item" src="/static/images/quiff.jpg" />
      <figcaption class="media__caption">Young Elvis had a magnificent quiff.</figcaption>
    </figure>
    <h3>Requirements</h3>
    <ul class="list list--bullet">
      <li class="list__item">Hair dryer</li>
      <li class="list__item">Styling putty</li>
      <li class="list__item">Finishing spray</li>
    </ul>
    <h3>Method</h3>
    <ol class="list list--number list--method">
      <li class="list__item">Wash and condition your hair as normal.</li>
      <li class="list__item">Towel dry you hair until it is only slightly damp.</li>
      <li class="list__item">Rub a small amount of styling putty between the tips of your fingers and style your quiff.</li>
      <li class="list__item">Blow dry into shape, using a
```

```
brush to pull your hair up.</li>
            <li class="list__item">Add shine and extra hold by
applying some finishing spray.</li>
        </ol>
    </article>
```

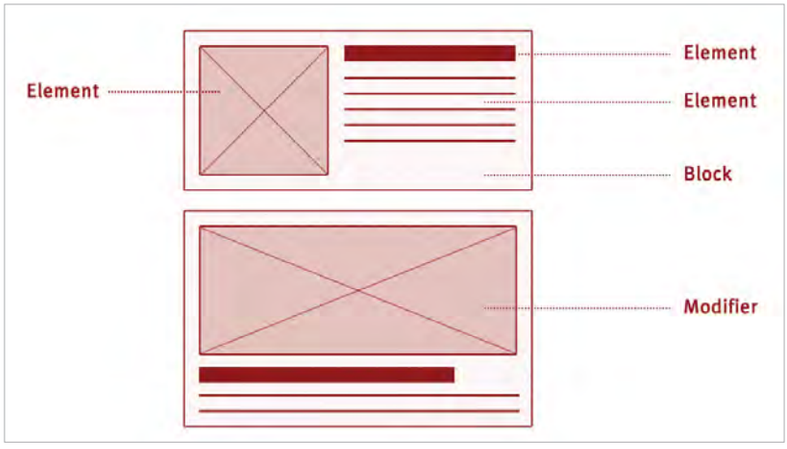
You can see the result in the image at the start of this tutorial (page 94). Our HTML page needs a bit of styling in the browser, so we can go ahead and create additional modules for layout and media, while including additional styles for the number and method list modifiers.

The following code example shows how the modifiers in `_lists.scss` may look, with module-specific variables being included at the top.

For example:

```
$list__number-color: #CC0000 !default;
.list--number {
  counter-reset: items;
  padding-top: calc-em(5);
  .list__item {
    margin-bottom: calc-em(5);
    &:before {
      counter-increment: items 1;
      content: counter(items, decimal) ".";
      color: $list__number-color;
      margin-right: calc-em(5);
    }
  }

  .list__item--end {
    margin-bottom: 0;
  }
}
.list--method {
  @extend .list--number;
  .list__item {
    margin-bottom: calc-em(15);
    &:before {
      @include text($text__size--beta, $text__line--beta);
      font-style: italic;
    }
  }
  .list__item--end {
    margin-bottom: 0;
  }
}
```

There are some examples of a layout and a media module, that complete the styling of this page in the GitHub repository that accompanies this tutorial. Visit it at the following link: *netm.ag/sass-249*. However, I'll let you be creative here and decide for yourself how to build them.



Promoting modularity BEM breaks code into blocks containing elements, then uses modifiers to tweak them

In this framework we've been using em sizing units throughout, which on smaller scale projects isn't a problem. But issues surrounding compounding font-sizes can crop up when we nest HTML elements. To combat this, we can use rem (root em) units.

To do this, we'd set a base font size on our HTML element. We already have a variable for this. For example:

```
html {
    font-size: $base_font-size
}
```

Then we would write a function to calculate rems and use that in place or our original `calc-em()` function:

```
@function calc-rem($target) {
    @return ($target / $base_font-size) * 1 rem;
}
```

Rems are supported in later versions of all major browsers (*caniuse.com/rem*), but for legacy support, we can use a pixel fallback.

GOING FURTHER

Obviously, we've only covered a very simple example in this tutorial, but the potential of using a Sass and BEM-based framework for your frontend development projects is enormous. Using the workflow outlined, you can create an essential building block that you can make use of for all your future projects, no matter how large they are, simply by adding modules as necessary.

It's important to remember that while you are building one system, you should make the base module as simple, and therefore reusable, as possible. Specificity can then be added using the `_config.scss` file and BEM modifiers.

VIDEO

Watch an exclusive screencast of this tutorial created by Tuts+ Premium: *netm.ag/tut1-249*

ABOUT THE AUTHOR

SCOTT KELLUM

w: *scottkellum.com*

t: @ScottKellum

areas of expertise: CSS, Sass, design, typography

q: what's your favourite smell?

a: Freshly baked bread

*SASS

COLOUR THEORY FOR THE WEB

Scott Kellum introduces some basic colour theory and practice to help make use of relative colour and make web designs much more flexible

There are lots of colours available to us, and for the most part, we describe colour in a way that makes it difficult to see the relationships between different colours. Names like red, green, blue, yellow, purple and maroon are great for conversations about colour but provide little insight into the structure of that colour. From the moment we can talk, colour is processed by the language part of our brains and its meaning is tied to the names we give colours.

Dimension, on the other hand, is processed differently by our brains. One foot and three feet aren't called different names. Instead, we measure dimension in a way that's proportional to a constant and well-known increment like inches or centimetres. These ideas of measurement come naturally to us and, while colour can be measured and described in a similar way, it's often very difficult for us to understand and manipulate colour as a measurement of its parts. It wasn't until Sir Isaac Newton did a series of experiments with prisms in the 1600s before we really started to understand that light was made up of different colours that could be measured independently of one another. Colour wheels were soon adopted by artists so they could better understand and mix colours.

While most artists use red, yellow and blue as primary colours, and most colour theory classes refer to these as the primary colours, science has evolved since the days of Isaac Newton and so has our understanding of how we see colour. Just about everyone's eyes contain various light receptors. Receptors called rods detect darks and lights, and cones detect three primary colours: red, green and blue. These three primary colours probably sound familiar to those of us who work on the web because this is how we write colour. Monitors have red, green and blue parts to every pixel. Each colour has 256 variations in brightness resulting in a total of 16,777,216 different colour possibilities. We can describe colour with the RGB colour function `rgb(255,115,0)` or hex values `#FF7300`, both of which describe the same colour in a slightly different way.

THINKING ABOUT RELATIVE COLOUR

To start thinking about colour and its relationships we can start with the CSS colour function `hsl()`. This function breaks colour down into hue, saturation and lightness. Hue is expressed in a unit-less degree (out of 360) while saturation and lightness are expressed as unit-less percentages (out of 100).

So hue is expressed as a value out of 360. This value is mapped to a colour wheel. As you are spinning through hue values, imagine spinning around a colour wheel. If you have red (`0`) and you want to make

yellow, you would just move hue to `60`. In colour theory there's the concept of complementary colours, which are two colours on opposite sides of the colour wheel. To find a complement of red you would just need to find the value that is `180°` around the wheel from red. In this case `0` for red plus `180` and the resulting colour is cyan.

Saturation and lightness is a bit more straightforward. Fully-saturated colour (`100`) has no grey in it while a saturation of `0` is greyscale. With lightness, `0` is black and `100` is white. Adjusting these values up and down will make your colour lighter or darker. Because our conception of colour is usually tied to the hue of a colour, it's much easier to think about saturation and lightness in terms of scale.

MANIPULATING COLOUR WITH SASS

By now you should have a solid understanding of how colour manipulation works. However, with pure CSS there's no way to adjust one colour relative to another without doing it by hand. Sass includes a huge range of colour functions (*netm.ag/sass-252*) that enable you to manipulate various aspects of a colour.

Simply adding some of this colour logic into mixins can help make them more flexible

For the most part, the most valuable functions are the HSL functions for darken, lighten, saturate, desaturate, and `adjust-hue`. Each one of these functions does simple mathematics on the original HSL values of the colour. For example, let's define a primary colour in a blank file:

```
sass
$primary: #437825;
```

Let's find another value for `$secondary` that's lighter and complementary to our primary colour.

```
sass
$primary: #437825;
$secondary: adjust-hue( lighten( $primary, 10% ), 180);
```

We could also have used the function `complement($colour)` as a shorthand for `adjust-hue($colour, 180)` as the complement of a colour is the same as 180° opposite the reference colour. These basic functions will handle just about every colour adjustment you may need, but there are some things that may be confusing if this is your first time using them. For example, `50%` `darken` or `lighten` don't take you half way to black or white. Instead, they add to the existing lightness value. So, if you darken a colour with a `L` value of `52` by `50`, the resulting colour will be black. To get around this you can use the `mix` function that simply mixes two colours together. If you mix you base colour with either black or white you may achieve a result much closer to what you expect.

```
sass
$primary: #437825;
$primary-dark: mix(black, $primary, 20%);
```

Now that we have all the tools to manipulate colours in our colour palette, we can use logic to make decisions as to how those colours are applied. Let's say we have buttons on our sites and, if the colour is dark, the text should be white but sometimes the colour is light so the text colour should be black. You can query each value of a colour with the `hue`, `saturation` and `lightness` functions.

To figure out how light our button is going to be, use the `lightness` function:

```
sass
@if lightness($color) > 50 {
  color: black;
} @else {
  color: white;
}
```

PUTTING RELATIVE COLOUR TO PRACTICAL USE

So far this has all been fairly abstract, but these techniques can be put to practical use. Themes can change colour with ease. When a client complains about the colour, it can be adjusted in one place, or you can create a single design for multiple sites that looks different by changing up the colour scheme. Simply adding some of this colour logic into mixins can help make them a bit more flexible.

At VOX Media we worked on the SB Nation (*www.sbnation.com*) redesign of over 300 websites (*www.sbnation.com/blogs*). These are sports sites so team colours are incredibly important. We worked it so every colour was relative to a distinctive team colour. There are still lots of different colour values on the sites, some darker and some lighter, but simply changing one colour can transform the site from looking like The Phin Sider (*www.thephinsider.com*) to looking like The Nunes Magician (*www.nunesmagician.com*). Hopefully the use of relative colour will be useful in your next project as well as helping to make designing much more flexible.

RESOURCE

CODEPEN DEMO

See Scott's CodePen demo of everything he talks about in this tutorial: *codepen.io/scottkellum/pen/gHByh*

ABOUT THE AUTHOR

JACKIE BALZER

w: *jackiebalzer.com*

t: @jackiebackwards

areas of expertise:
Sass, CSS, HTML, JavaScript

q: what's the most heroic thing you've ever done?
a: Defeated the entire Behance team in a day-long shuffleboard tournament

SASS

SHARPEN YOUR SASS PROGRAMMING SKILLS

Sass isn't just mixins. **Jackie Balzer** explores the computer science concepts that can supercharge your programming skills

Sass is popular for making CSS quicker and easier to write, but did you know that it is a powerful scripting language too? Because Sass itself is written in the Ruby programming language, many of the same programming concepts that can be used in Ruby can be used in Sass.

Variables, logic, control directives and functions are all at your disposal and can be used to great effect in programming your CSS. What's more, frameworks such as Compass (*compass-style.org*) offer tons of additional functionality that can be used to take your CSS programming to the next level.

VARIABLES

All popular CSS preprocessors have variables, but Sass stands apart with its multitude of data types. These determine the kind of value a variable can hold and what operations can be performed on it. Having specific data types is useful in many ways, and each data type serves a different purpose.

Sass shares several basic data types with many programming languages: numbers (integers or decimal numbers, optionally including a unit such as `px` or `deg`), booleans (true or false), strings (such as the word 'hello') and nulls (you can think of this as an empty value).

There are also some Sass-specific data types: colours (all CSS representations of a colour, such as `blue` or `#0000ff`), lists and maps. All data types support equality operators (`==`, `!=`, `>`, `>=`, `<` and `<=`) and basic operations, such as arithmetic:

```
$inner-height: 100px - 30px; // equals 70px
$mix-color: #005cff+#202020; // equals #207cff
```

You can check if values are equal:

```
$test1: 16 != blue; // true
$test2: 30% > 100%; // false
$test3: blue == #0000ff; // true
$test4: 3/4 <= .25;  // false
```

Some data types have type-specific Sass functions:

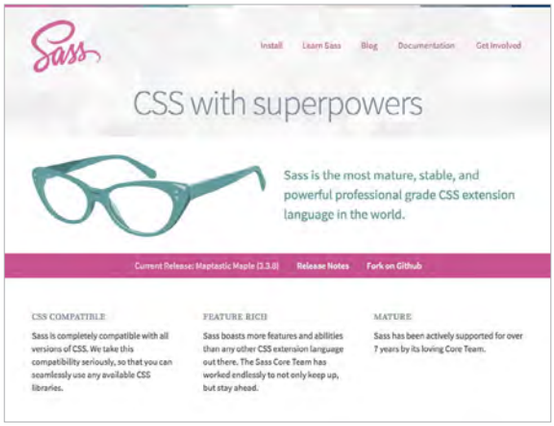
```
$str-length: str-length('hello'); // gets the number of characters in a string: 5
$length: length(5px 10px 10px 5px); // gets the number of items in a list: 4
$round: round(4/3); // rounds the value of 4/3 to the nearest whole number: 1
```

You can change values from one type to another, which is called 'typecasting'. This can sometimes have unexpected results, as demonstrated in the code below:

```
$number: 1.2; // number data type
$unit: em; // string data type, since there is no associated number
$result: $number + $unit; // adds a number type and a string type, creating the string '1.2em'
$rounded: round($result); // causes an error - $rounded: "1.2em" is not a number for `round'
```

VIDEO

Jackie Balzer has created an exclusive screencast to go alongside this tutorial. Watch along at *netm.ag/tutSass-258*



CSS with superpowers The homepage of the Sass preprocessor, good for examples, links and all your documentation needs

These basic operations are the foundation for how powerful Sass variables are.

LISTS AND MAPS

Lists and maps are other more complex data types. Lists are a series of values, like the ones we have already covered, or even other lists. Lists can be separated by spaces or commas, like so:

Because Sass is written in Ruby, many of the same programming concepts can be used

```
$margins: 10px 5px 20px 15px; // space-separated list
$font-stack: Helvetica, Arial, sans-serif; // comma-separated list
$colors: heading red, paragraph blue, figure orange, caption white; // comma-separated list of space-separated lists
```

A map is like a list on steroids: you can think of it as a list of key-value pairs. Unlike lists, maps must always be comma-separated. Like lists, you can use any data type for either the key or value, including other maps.

A simple map could look like this:

```
$map: ( heading: red,
        paragraph: blue,
        figure: orange,
        caption: white );
```

Though you can use all of the basic operations we have already covered on list and maps, their power lies within Sass functions (*netm.ag/functions-258*),

FOCUS ON

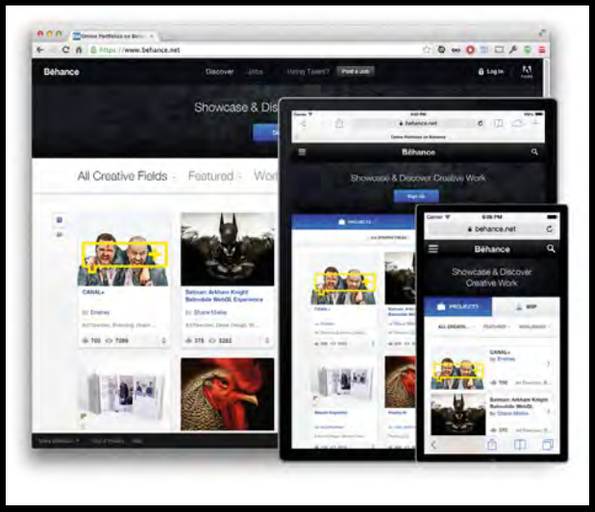
SPEAKING THE SAME LANGUAGE

Sometimes it's useful for your Sass and JavaScript to be able to talk to each other. This may be especially key if you are working on a responsive site, as you could easily find yourself with breakpoint values stored in both places. Sass is all about keeping code DRY, and that goes for more than just your Sass files.

Fortunately, Sass has the ability to let you write custom SassScript functions at the Ruby level. We'll start by finding the lowest common denominator: in this case, it's JSON. A JavaScript file can understand JSON, of course, since JSON is JavaScript Object Notation. The good news is that Ruby can also read JSON files using a JSON Ruby gem.

Now you can create a simple JSON file containing a single JSON object of your breakpoint names and values. You can write a Ruby function that gets loaded when you compile your Sass to read from that JSON file and pass the information back to your Sass files. Then, in your JavaScript, you can use something like requireJS (*requirejs.org*) – or another file loading mechanism – to load that same JSON file and access the breakpoints directly.

The end result is that you can update your breakpoint values in one place – your JSON file – and any changes will be reflected in both places. Clean and reusable, just the way we like it!



In sync Behance.net shares breakpoint values between the Sass and JavaScript to keep responsive code in sync on both sides

RESOURCES

SASSY RESOURCES

There are a wealth of resources available to help you take your Sass skills to the next level. The official Sass website itself (*sass-lang.com*) is very thorough and with tons of examples. Beyond that, here are my top sites and books to explore:

Compass *compass-style.org*
The best source for Compass documentation – an excellent, extensive framework, full of goodies.

Bourbon (*bourbon.io*)
A popular extensive library of excellent mixins.

Sassmeister (*sassmeister.com*)
A playground for Sass, this enables you to experiment, and you can easily see your CSS output.

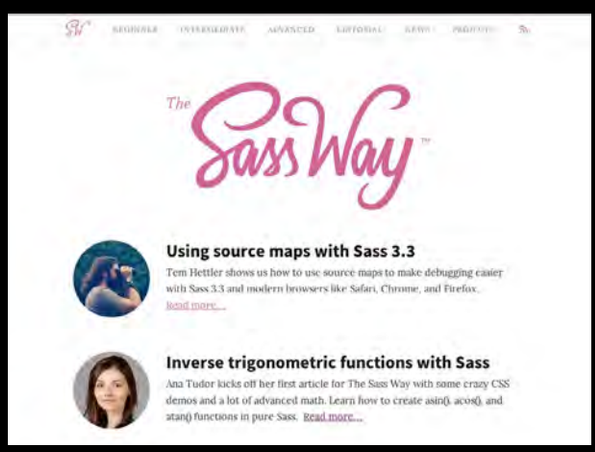
CodePen (*codepen.io*)
Practice your Sass, peruse others, share your work!

CSS Tricks (*css-tricks.com*)
Chris Coyier writes about CSS in general, and talks a lot about Sass.

The Sass Way (*thesassway.com/*)
Tons of excellent user-submitted articles covering everything you'd ever want to know about or do with Sass.

Sass for Web Designers (*netm.ag/cederholm-258*)
A book by Dan Cederholm covering everything from Sass basics to advanced techniques.

libSass (*libsass.org*)
A C/C++ port of the Sass engine.



The Sass Way A great collection of articles submitted by users

▶ which you can use to iterate over, manipulate and retrieve data from your lists and maps.

FUNCTIONS AND CONTROL STRUCTURES

Logic, control structures and functions can give you control over the flow of your code. The `@if` statement lets you control what happens if a condition is true or false. `@for`, `@each` and `@while` loops enable you to iterate over lists or maps.

Functions accept input and return a value, which are useful for repeated calculations. Sass has a ton of great functions built in, and you can write your own. Functions are awesomely flexible as they can take a specific number of arguments, named arguments or infinite arguments (*netm.ag/arguments-258*).

Text colour

Let's say we want to change our text colour based on how dark our background colour is:

```
@if(lightness($background) < 50%) {
  color: white;
} @else {
  color: black;
}
```

Logic, control structures and functions can give you control over the flow of your code

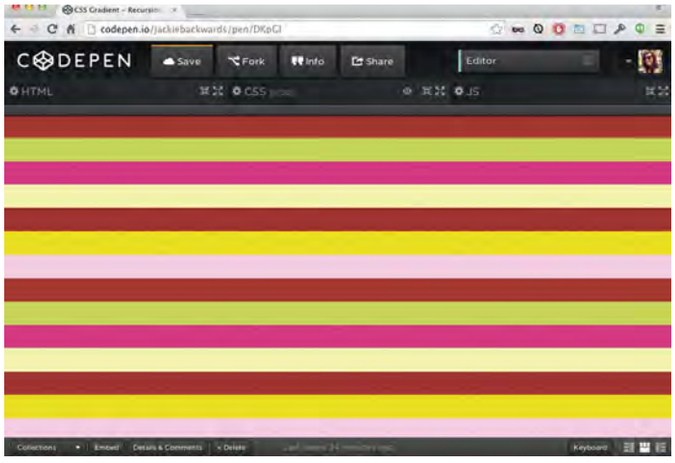
Sass' `lightness` function returns the lightness component of our background, and our equality `<` operator checks if its value is less than 50 per cent. If so, our text becomes white, otherwise, our text is black. There is an `if` function to shorten this:

```
$color: if(lightness($background) < 50%, white, black);
```

This function behaves like a a ternary operator (*netm.ag/ternary-258*), a common programming structure. It returns one of two values (the second or third argument) based on if the boolean condition (the first argument) is true or false.

How about if you wanted to create a gradient with a specific set of colours? This is easy if you have a list:

```
$colors: #b00b00 #bada55 #de1e7e #f0feaf #ac1d1c #e1e100
#facade;
html {
  @include background-image(linear-gradient(top, $colors));
}
```



Left A CodePen example, with hard stops between each colour

Right Recursion is commonly seen in nature, as in this aloe plant (Credit: brewbooks via Flickr *bit.ly/1mq8ln4*)

By default, each colour blends into the next. To make a gradient with a hard stop between the colours, each colour must have a stop at two points. This can be done with a loop:

```
$stops: ();
$list-length: length($colors);
@for $i from 1 through $list-length {
 $color: nth($colors, $i);
 $new-stops: $color percentage(($i - 1)/$list-length), $color percentage($i/$list-length);
 $stops: join($stops, $new-stops, comma);
}
html {
 @include background-image(linear-gradient(top, $stops));
}
```

First, initalise a new empty list called `$stops` which will store the colour-percentage pairs. The `@for` loop iterates over the list of colours, counting from one through the length of the list. Sass' list function, `nth`, gets the colour at index `$i` of the list. The `percentage` function calculates the percentage of the current (`$i`) and previous index's position in the list (`$i - 1`). Finally, the `join` function adds our new colour-percentage pairs to our list of `$stops`.

The resulting gradient looks like this:

```
linear-gradient(to bottom, #b00b00 0%, #b00b00 14.28571%, #bada55 14.28571%, #bada55 28.57143%, #de1e7e 28.57143%, #de1e7e 42.85714%, #f0feaf 42.85714%, #f0feaf 57.14286%, #ac1d1c 57.14286%, #ac1d1c 71.42857%, #e1e100 71.42857%, #e1e100 85.71429%, #facade 85.71429%, #facade 100%)
```

Recursion

Let's take it one step further by using a recursive function instead of a loop. In computer science, recursion is a technique where the solution to a problem depends on solutions to smaller instances of the same problem, as opposed to iteration (looping).

This function can replace our `@for` loop:

```
@function gradient($stops: ()) {
 $stops-length: round(length($stops) / 2);
 @if $stops-length == $list-length {
  @return $stops;
 }
 $i: if($stops-length == 0, 1, $stops-length+1);
 $color: nth($colors, $i);
 $new-stops: $color percentage(($i - 1)/$list-length), $color percentage($i/$list-length);
 $stops: join($stops, $new-stops, comma);
 @return gradient($stops);
}
```

The function is bigger than the loop but is more self-sufficient: it initialises its own list of empty `$stops`, adding to it and passing it to itself on every call. It uses half the length of `$stops` to determine which `$color` in the list to calculate, and ceases calling itself when `$stops` is twice as long as `$colors`. Then it returns the list of `$stops`, which gets passed to `linear-gradient()`.

This example does not necessarily save computational time, but it is an interesting look at how recursion can be used to solve a problem in a different way.

CONCLUSION

As you can see, Sass' usefulness extends far beyond the basics. It provides several different kinds of data types which you can use to manipulate values in interesting ways, use logic to output code in different ways, and control structures and functions to do advanced processing. The ideas covered in this article only just scratch the surface of the practical things you can do with Sass.

RESOURCE

SASSY BLOG

Hugo Giraudel's blog (*hugogiraudel.com*) is packed with tons of good insight, information and exercises in Sass, especially advanced concepts.

ABOUT THE AUTHOR

TIM HETTLER

w: *timhettler.com*

t: @timhettler

areas of expertise:
HTML, CSS and Sass, JavaScript, Grunt

q: what's the worst you've ever screwed up at work?

a: At my first job, I forgot a semi-colon in a line of JavaScript and completely broke one of the largest sites on the internet for about 20 minutes

SASS

TRANSLATING DESIGN DETAILS WITH SASS

Tim Hettler explains how Sass can help bridge the communication gap between designers and developers, and translate designs from Photoshop

As much as I love CSS, it has trouble keeping up with the complex sites we build today. It can't do calculations, loop through a series of items, or even store values for later use, all of which are important to have when working on large site. Add in multiple developers to a project and it becomes almost impossible to maintain the visual consistency of a site as it grows larger.

Luckily, Sass, the popular CSS extension language, provides everything we need to manage the visuals of a complex site and faithfully create subtle design details that are too difficult or time-consuming to do with CSS.

DESIGNERS ARE FROM VENUS

One reason visual inconsistencies get introduced during the development process is that designers and developers have different ways to talk about the same thing. As a developer, I would love it if a designer told me, "That headline needs to be `28px` and the colour is `#cdff03`." But that is, unfortunately, not how most humans talk.

They say things like "The font size we use for every headline" and use fancy colour terms like "electric green", and the developer translates how humans talk to what CSS requires. Every time this translation is done by hand, it introduces an opportunity for human error. A better way to handle this is to store these values as Sass variables:

```
$font-size--headline: 28px;
$color--electric-green: #cdff03;
```

By using variables, designers and developers can discuss the details of a project in equal terms.

SPEAKING THE SAME LANGUAGE

A bigger problem is that designers have the ability to do things in graphics programs like Photoshop that have no CSS equivalent. The developer has to guess how to represent it on the web. Avoiding translation entirely is the best way to reduce errors.

Sass introduces a number of functions that allow you to adjust the colour of an element in the same way it's done in Photoshop. If the hover colour of a button is achieved in Photoshop by darkening the base colour by `20%`, you can express that in Sass using the `darken` function and create a mixin that takes the `background-color` as a parameter:

```
@mixin button-color($color--bg) {
    background-color: $color--bg;
    &:hover {
        background-color: darken($color--bg, 20%);
    }
}
```

The relationship between base and hover colour is now clearly defined in the code, making Sass self-

RESOURCE

TALK SLIDES

See the slides from Tim Hettler's talk, 'Sweating the Small Stuff: Recreating Subtle Design Details Using Sass', which this tutorial is based on: *timhettler.github.io/sweating-small-stuff*

documenting in a way that isn't possible with CSS. By abstracting the code into a reusable mixin, we ensure this relationship is applied to every button we create.

AUTOMATING DESIGNER INTENT

An issue I ran into recently was determining when to use a light or dark text colour. In order to ensure enough contrast for proper readability, a light text colour should be used against dark backgrounds, and vice-versa. Typically, a designer will hand me a large matrix of every background colour and which text colour should accompany it.

I created a function that looks at the lightness value of a colour and returns the light colour if it's less than `50%`, and the dark colour otherwise. Despite having faith in my function, certain pairings weren't what I would have chosen by hand. After some research, I discovered the human eye is actually more sensitive to certain colours. Knowing that, I wrote a new function to take this bias into account (find it on GitHub at *netm.ag/compass-261*):

Sass makes it easier to create complex, detailed sites without losing any visual details

```
@function yiq-contrast-color($color, $dark, $light) {
    $red: red($color);
    $green: green($color);
    $blue: blue($color);
    $yiq: (($red*299)+($green*587)+($blue*114))/1000;
    @return if($yiq < 128, $light, $dark);
}
```

The colour is translated from RGB to the YIQ colour space, which weights the parts of a colour according to their impact on our perception. The result is a value between `0` and `255` that represents the overall brightness of the colour. Using this function, the contrast pairs that were produced matched up exactly with what the designer wanted. We may not be able to completely replace designer's intuition, but in this case we've used Sass to turn a designer's gut feeling into a repeatable pattern.

DROP SHADOW TO BOX-SHADOW

Sass can also help you translate complex layer effects. Take the drop shadow effect and its CSS counter-part `box-shadow`. In Photoshop, a drop shadow is expressed with an angle, distance, spread and size. However, a `box-shadow` in CSS is represented by an `x-` and `y-offset`, `blur-radius` and `spread-radius`. I don't mean to spring a trigonometry question on you, but imagine our `box-shadow` is a right triangle. Since we know the values of the angle and hypotenuse (the distance), how would you determine the length of the opposite and adjacent sides of the triangle (the `x-` and `y-offset`)? Stumped?

It's okay, I didn't remember how to do it either. Luckily, we can use trigonometric functions that are provided in Compass, a popular CSS authoring framework that extends Sass's functionality, to do the maths for us:

```
$x-offset: cos($angle) * $distance;
$y-offset: sin($angle) * $distance;
```

We no longer have to guess at these values since we have a formula to convert them for us. Translating the `blur` and `spread` of our `box-shadow` from their Photoshop counterparts is tricky. In CSS, the `spread` is added to the `blur` to determine the total shadow size.

However, the Photoshop size value represents the total length of the shadow, and the spread indicates how much of the shadow is a solid colour. Again, we can create a simple formula to convert from the Photoshop values to the CSS `box-shadow` syntax:

```
$css-spread: $size * $spread/100;
$blur: ($size - $css-spread);
```

Now that we've converted the Photoshop values to the four values required for a CSS `box-shadow`, we have a pixel-perfect representation of the drop shadow. Grady Kuhnline has packaged this entire process into a Compass extension (*github.com/heygrady/compass-photoshop-drop-shadow*). Using his mixin, you can convert a drop shadow into a `box-shadow` with one line of code:

```
// Sass
@include photoshop-drop-shadow(120, 5px, 0, 5px,
rgba(#000, 0.75));
// CSS
box-shadow: 3px 4px 5px 0px rgba(0, 0, 0, 0.75) ;
```

By automating this translation, we can have full confidence that what is coded in CSS is exactly what the designer intended in Photoshop, and none of the subtle design details will be lost.

MORE THAN JUST MIXINS

Sass gives us the ability to bridge the communication gap between designers and developers, making it easier to collaborate and to create complex, detailed sites without losing any visual details.

VIDEO

Watch an exclusive screencast of this tutorial created by the author: *netm.ag/tut2-253*

Download the files here!

Download all the source code you need for this tutorial at *netm.ag/sass-260*

ABOUT THE AUTHOR

IAN CARRICO

w: *iamcarrico.com*

t: @iamcarrico

areas of expertise: Drupal, Sass, Gulp, responsive web, performance

q: what was your childhood nickname?
a: 'Nutboy', because I ate Nutter Butters like it was my job

SASS

TAKE THE PAIN OUT OF SASS DEVELOPMENT

Ian Carrico walks through how to set up your Sass development environment properly, saving you from future headaches

Using CSS preprocessors has become a standard practice across web development. They have provided us with immense power in our frontend development, not just with new grid systems and colour functions, but by offering us the opportunity to keep our code truly DRY. The problem is, ensuring our development environment is ready for writing Sass can be tricky – especially to those of us who did not start as Ruby developers, and are treading unfamilar ground.

LATEST AND GREATEST

Sass is constantly evolving. It is a language that is actively being developed, offering many community-driven tools. If you want to explore the full power of Sass, this article is for you. I will explain step-by-step what tools you need to make sure your Sass development starts properly, and point you in the direction of the right documentation to help save you from future headaches.

If you are brand new to Sass and do not need the latest and greatest features, there are a number of applications to help start you off. My favourite Sass playground is Sassmeister (*sassmeister.com*), developed by Jed Foster and Dale Sande. If you are looking for an application to run locally, get your hands on a copy of Codekit (*incident57.com/codekit*). Remember, Codekit is awesome, but it does limit the versions of Sass you can use.

What makes Sass hard?

In short: Ruby. The Sass compiler is built on Ruby, and most of the community code is deployed as RubyGems. In order to ensure we keep our development environments in sync, we need to make sure our versions of Ruby and RubyGems are consistent, and documented in code. On the plus side, the tools we need to do this have been worked on for a long time, and are (mostly) well documented.

I'm not on OS X ...

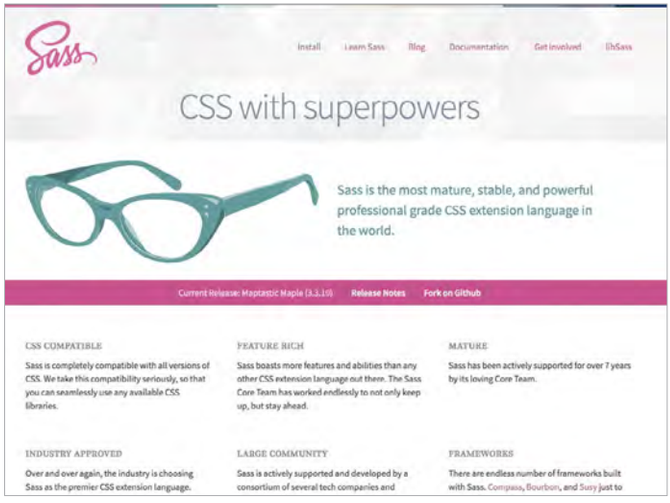
This tutorial is meant mainly for OS X, but the tools (except Homebrew) can be used on any Linux system. The only change is that instead of using Homebrew commands to install the packages, you will use the native package manager for your version of Linux. Every piece of software included has more detailed installation instructions for different systems within their documentation as well.

EASY WIN

Before we install anything, or get into the tools we need, there is one configuration file everybody should use. It stops RubyGems from installing the

VIDEO

This article is based on Ian Carrico's talk at Sass Conf. Watch his original presentation at *vimeo.com/86306777*



Left The Sass language gives us more control over our CSS than we have ever had before

Right Sassmeister is a web app that will allow you to compile Sass samples and publish them to a GitHub gist

full documentation of each gem locally, saving time during gems install.

Run this single line from the command line:

```
$ echo 'gem: --no-rdoc --no-ri' >> ~/.gemrc
```

HOMEBREW

For Mac users, the first step of this project is to install Homebrew (*brew.sh*), the unofficial package manager for OS X. Homebrew will enable us to install the rest of the tools we need in order to tackle some painless Sass development.

Preprocessors have provided us with immense power in our frontend development

For users that have not used Homebrew before, open the terminal and run this line of code to install the package manager:

```
$ ruby -e "$(curl -fsSL https://raw.github.com/Homebrew/homebrew/go/install)"
```

Once you've installed Homebrew, you will be able to run the command `brew` on the command line. To ensure you are ready to move forward, run `brew doctor`, which will check to ensure Homebrew installation is ready to use. If there are any errors, Homebrew also provides instructions on how to fix the installation.

If you already have Homebrew installed, additionally run `brew update`, which will make sure all of the package information is up to date. Generally, you should run this before any new package installations.

MANAGING YOUR RUBY VERSIONS

Ruby is constantly updated, both with major releases and patch fixes. It is important that we know which version of Ruby we are developing on, as there are differences in each version. Luckily, there are standard practices to ensure the version of Ruby you are using is correct.

There are three main tools that allow us to manage our Ruby version: RVM (*rvm.io*), rbenv (*github.com/sstephenson/rbenv*) and chruby (*github.com/postmodern/chruby*). I recommend using rbenv, and this guide will go over how to install, configure and use it.

However, if you already have RVM or chruby installed, do not also install rbenv, as the two are not compatible with each other. If you already have one of these programs installed, skip the section below, and jump to 'Ruby versions'.

INSTALL RBENV

To install rbenv, run the following on OS X:

```
$ brew install rbenv ruby-build
$ echo 'eval "$(rbenv init -)"' >> ~/.bash_profile
```

Open up a new terminal window – rbenv should now be installed to your system. Now we need to install the version of Ruby that we want to use globally, which at the time of writing is 2.0.0p451.

```
$ rbenv install 2.0.0-p451
$ rbenv rehash
$ rbenv global 2.0.0-p451
```

The first line will download and install the right version of Ruby onto your system. The second line rehashes rbenv's knowledge of what versions you have installed, and should be run after installing any version. The last line will set the global Ruby version to be the version we just installed.

By default, rbenv will download and install Ruby versions into `~/.rbenv/versions`. If you ever want to uninstall a version of Ruby, just delete the version folder from `~/.rbenv/versions`. So, the command you'd need to use to uninstall the version we just installed would be:

```
$ rm -rf ~/.rbenv/versions/2.0.0-p451
```

RUBY VERSIONS

No matter which program you are using to manage your Ruby versions, they can all respect the file named `.ruby-version` in your project's folder. This is a one-line file that will document the version of Ruby we started this project with.

With rbenv, you can create this file by:

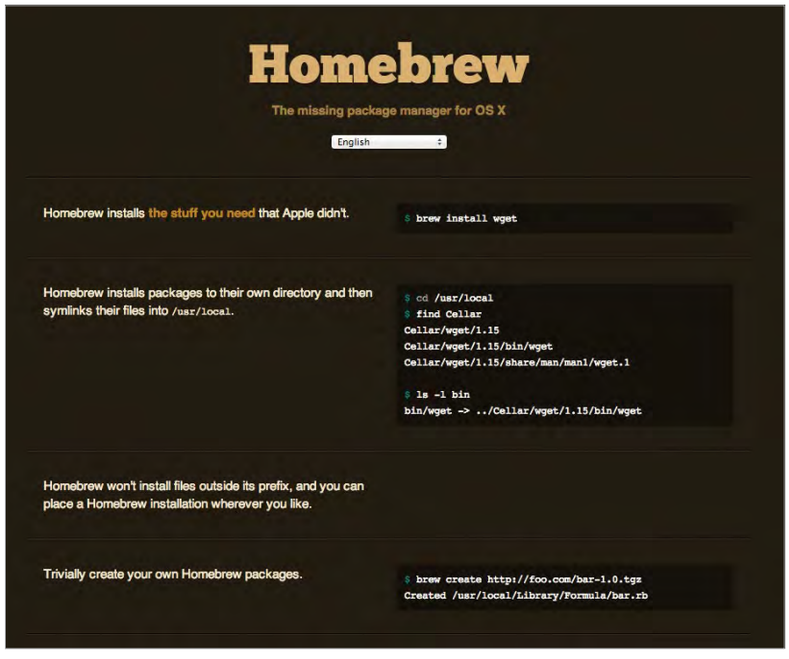
```
$ rbenv local 2.0.0-p451
```

You can also create it by running:

```
$ echo "2.0.0-p451" > .ruby-version
```

Always commit the `.ruby-version` file to your version control. Then anybody else working on the project, as long as they have RVM, rbenv or chruby installed, will always be using the same version of Ruby.

Homebrew This allows for package installation on OS X, and can be used to install a variety of tools



BUNDLER

Sass, Compass and the many Compass extensions are distributed using RubyGems. As each gem can exhibit pretty big changes in-between versions, we want to document the versions needed in code as well. We do this through Bundler (*bundler.io*), our friendly Gem version manager for Ruby.

Installation is easy, just run:

```
$ gem install bundler
```

GEMFILES

The two files that Bundler uses to make sure the versions of gems are accurate are `Gemfile` and `Gemfile.lock`. We can create the former with the versions of gems we want to use, the latter is made by Bundler with the exact versions of each gem we are using. Both must be committed to version control, so that every developer installs and uses the exact same Ruby gems.

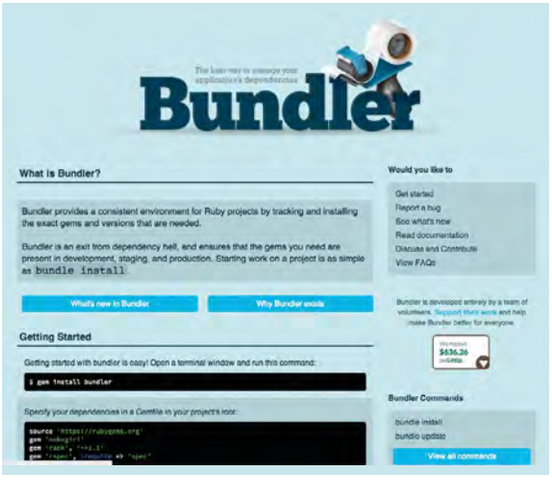
Speed up your workflow with a task runner ... Gulp.js helps automate common tasks

Each project can have different Gemfiles, as tools update and change continually. As a starting point, here is an example of a Gemfile with the most up-to-date versions of Sass and Compass available at the time of writing. You can specify either exact versions, or fuzzy versions (e.g >= 1.0) within your own Gemfile.

For more information on Gemfile's syntax, look at Bundler's documentation at *netm.ag/bundler-260*.

```
# Pull gems from RubyGems
source 'https://rubygems.org'

# Use Sass version 3.4.2 - 3.4.9.
gem 'sass', "~> 3.4.2"

# Use Compass version 1.0.1 - 1.0.9
gem 'compass', '~> 1.0.1'

# Other Compass extensions
gem 'breakpoint', '~> 2.5.0'
gem 'singularitygs', '~> 1.4.0'
gem 'toolkit', '~> 2.6.0'
```

Save the file above as `Gemfile`, then run `bundle install`. Bundler will then read the Gemfile, install



Bundler This can be used to ensure everybody is using exactly the same same versions of the gems within your project

the needed versions of each gem and install them. Finally, it will create the Gemfile.lock which will contain the exact versions Bundler installed.

USING BUNDLER

From this point forward, instead of running your usual Compass commands, put a `bundle exec` in front of Ruby commands. So, to compile or watch your Sass files, you can use:

```
$ bundle exec compass compile
$ bundle exec compass watch
```

The `bundle exec` will ensure you are running these commands through Bundler, thus always using the proper versions of each gem.

If you need to update the gem to a newer release, you have two options, depending on the version you want to install. If you want to update to a newer version that is still valid to the Gemfile, you would run `bundle update gemname`. If the version you want to use is not valid, then the Gemfile needs to be updated, followed by `bundle install`.

WRAPPING IT IN MAGIC

For bonus points, we can speed up our workflow with a task runner. Gulp.js (*gulpjs.com*) can be used to automate common frontend tasks, from compiling Sass to minifying images. It runs each task concurrently, ensuring the least amount of time waiting at our command line.

Setting up Gulp.js starts with ensuring you have Node.js installed (*nodejs.org/download*). Once Node.js is installed successfully, then Gulp.js can be installed globally with:

```
$ npm install -g gulp
```

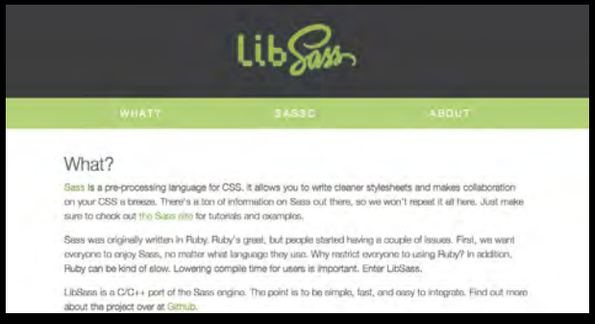
▶

FOCUS ON

WHY IS SASS SO GREAT?

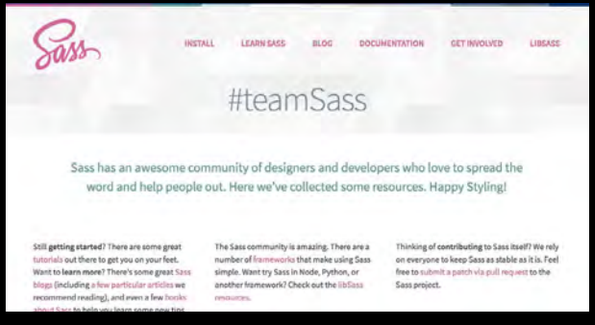
It may seem like this tutorial is long. Since we are playing with Ruby, there are lots of dependencies and requirements. Is all of this worth it just for the power of a CSS preprocessor? Why not use something a little simpler, keep everything in Node.js? Better yet, let's just use Bootstrap – that gives me everything I need, right?

The reason I love Sass so much is not just because of the power it has as a CSS preprocessor, it is because of the community that has grown around it. This has not just made Sass more powerful, but there are so many tools that have been built in Sass that I couldn't possibly imagine building a site without it any more.



LibSass This is a C implementation of Sass, built to make compiling Sass even easier and faster. Although it does not currently have feature parity with Sass, there is constant development

The beauty of this tutorial is that it only needs to be run once, and your environment is pretty much set up forever. Afterwards, you will only need to update gem and npm package versions occasionally. Moreover, the Sass community has been hard at work trying to simplify this process even further, primarily through building LibSass (*libsass.org*).



Team spirit The Sass community is constantly making new tools, blog posts, tutorials and podcasts

In short, I am proud to be apart of the #teamSass community, and will continue my work to make Sass development easier for everyone. I am amazed at the creativity the community has used to create the tools that power the web.

Gulp.js This helps you automate into a simple and fast workflow

Next, within your project directory, create a package.json file and install the required npm packages by running:

```
$ npm init
$ npm install --save-dev gulp compass-options gulp-compass
```

Gulp uses the file gulpfile.js within your project directory to create and manage tasks. Below is an simple gulpfile that will compile your Sass for you, using Compass.

To run the command, just copy the code into your gulpfile.js, and run `gulp sass`. At *github.com/iamcarrico/netmag-code* there is a more complex example gulpfile with many more tasks you can run.

```
var gulp = require('gulp');
var compass = require('gulp-compass'),
  paths = require('compass-options').dirs();

gulp.task('sass', function() {
 return gulp.src(paths.sass + '/**/*.scss')
  .pipe(compass({
    config_file: './config.rb',
    css: paths.css,
    sass: paths.sass
  }))
  .pipe(gulp.dest(paths.css));
});
```

THAT'S IT!

That wasn't so hard was it? A little bit of environment setting-up, and now your Sass development can focus on the Sass, not the underlying Ruby. From here, the sky is the limit. Check out the tools the community has been creating to extend your Sass development even further.

IN-DEPTH

WHERE TO GO NEXT?

Sass has been made so powerful because of the community that has been formed around it. There are innumerable tools that have been created to make frontend development easier, faster and more DRY.

Twitter accounts to follow

- The Official Sass Twitter: @SassCSS
- Hampton Catlin, creator of Sass: @hcatlin
- Natalie Weizenbaum, lead developer of Sass: @nex3
- Chris Eppstein, creator of Compass: @chriseppstein

According to the Sass' official documentation, to use Sass you must pay Hampton Catlin a compliment and get Natalie Weizenbaum some candy. Technically not required, but at least pay them a follow on Twitter.

Awesome Compass extensions

I use each of these tools personally on almost every project I do.

- **Breakpoint** (*breakpoint-sass.com*)
  Creating simple media queries
- **Singularity** (*singularity.gs*)
  Grids without limits
- **Toolkit** (*github.com/Team-Sass/toolkit*)
  The Swiss Army Knife of responsive web design
- **Color Schemer** (*github.com/Team-Sass/color-schemer*)
  A robust colour toolset
- **Navigator** (*github.com/Team-Sass/navigator*)
  A Ruby testing framework for Sass

Where else can I find information?

There are many folks writing excellent tutorials and blog posts on Sass every day. Check out the Sass community page (*sass-lang.com/community*) for the latest in Sass writings. Also, I cannot recommend enough Dale Sande's book *Sass in the Real World,* available on GitBook (*netm.ag/realworld-260*).

Get involved

There are many Sass meetups across the world. Find the one that is closest to you and meet other Sassy people in your area. Follow the general @SassMeetup Twitter account for information of all meetups as well. Moreover, if you get the chance, go to SassConf (*sassconf.com*) or Camp Sass (*campsass.com*) to really dive deep within the community.

ABOUT THE AUTHOR

MARK DALGLEISH

w: *markdalgleish.com*

t: @markdalgleish

areas of expertise:
JavaScript, frontend development and Node.js

HEAD TO HEAD

GRUNT VS GULP

Mark Dalgleish takes a closer look at the pros and cons of Grunt and Gulp, two competing JavaScript build tools

GRUNTJS.COM	GULPJS.COM
Grunt is a Node.js-based task runner, most commonly used for frontend projects.	Gulp is a lightweight build tool, using Node's stream API to speed up and simplify complex build processes.
PHILOSOPHY	
Grunt builds are mostly made up of JSON-like configuration rather than code. Instead of writing a programmatic build script, you supply options to a series of pre-packaged build tasks inside a Gruntfile.	Gulp takes the opposite approach, favouring code over configuration. There are only a few simple methods you need to create a working Gulpfile. Due to its streaming API, builds are extremely fast.
PLUGINS	
Grunt has a large, growing community surrounding it. There are currently almost 2,300 plugins available, supporting common tasks like static asset compilation and deployment.	Gulp still has a relatively young ecosystem, but there are already around 300 plugins that have been published to npm. A core benefit of Gulp's streaming API is that plugins are small and easily connectable.
SPEED	
Tasks run synchronously by default. Each task is individually configured to interact with the file system. This I/O-heavy approach can cause performance degradation if your project grows too large. Grunt plugins try to improve the situation, but it's not ideal.	Gulp plugins usually just transform virtual files in memory, making them much faster than their Grunt counterparts. The larger and more complicated your build process becomes, the more you'll appreciate Gulp's superior performance.
SCAFFOLDING	
Yeoman, a very popular JavaScript scaffolding tool, has first-class support for Grunt. The vast majority of Yeoman generators create a project that includes a Gruntfile. There is even a Yeoman generator the can scaffold Grunt plugins.	The Yeoman team are prototyping the use of Gulp in their official generators. Yeoman was instrumental in the creation of many Grunt plugins, so this process is likely to be repeated with Gulp. Just like Grunt, there's a generator that can scaffold boilerplate Gulp plugins.

VERDICT

The Grunt team is planning to implement similar virtual file transformations to Gulp, so the choice between the two will increasingly depend on whether you prefer configuration or code for your build process. Unless your build is modified by frontend designers with little JavaScript experience, Gulp is likely to be a much more popular choice in the long run.

FACT FILE

ALTERNATIVE

Many small JavaScript projects leverage npm's "scripts" functionality to create an extremely lean build process. This approach is particularly popular in the Node community where the more minimal solutions are often the most popular.

DEPENDENCIES

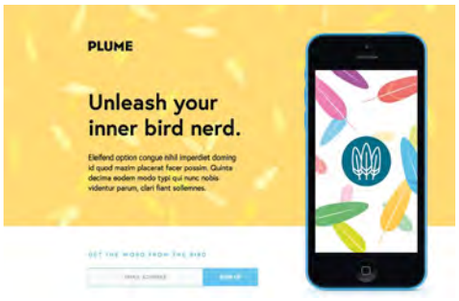
Grunt and Gulp require Node.js (*nodejs.org*).

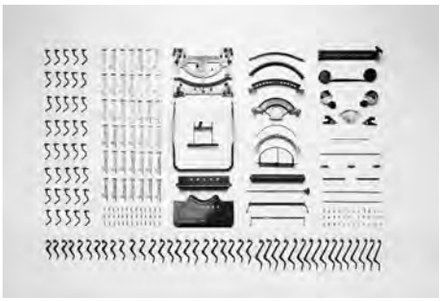
INSTALLATION

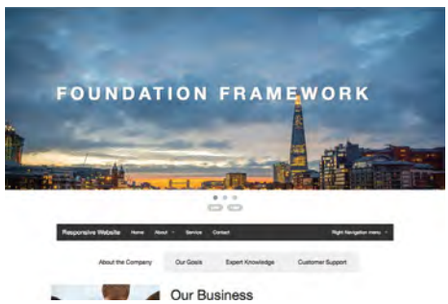
Grunt
npm install -g grunt-cli

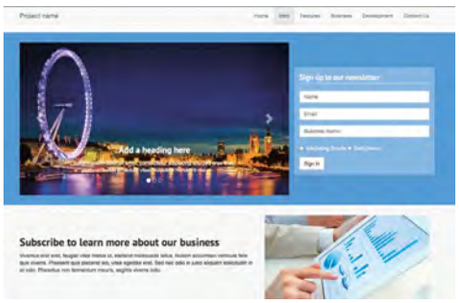
Gulp
npm install -g gulp

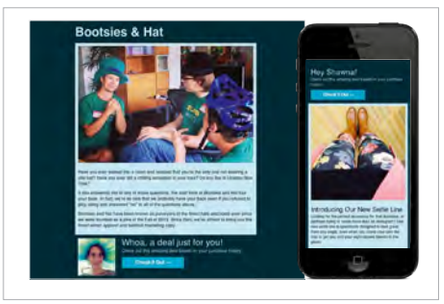
RESPONSIVE DESIGN











AUTHOR

DAN ROSE

Dan (*danrose.me*) is the creator of Photoshop Etiquette (*photoshopetiquette.com*). He's an interface designer at WSOL, conference speaker and pioneer of Syracuse Sync (*syracusesync.com*)

ILLUSTRATION

LUKE O'NEILL

Luke (*lukeoneill.co.uk*) is the art editor of *T3* magazine and also works on freelance projects in the fields of graphic design, art direction and illustration

NEW TOOLS IN RESPONSIVE WEB DESIGN

Chances are your design tool needs have changed drastically in just two years. Are the tools on the market up to the task? **Dan Rose** looks at four apps aimed at aiding responsive design

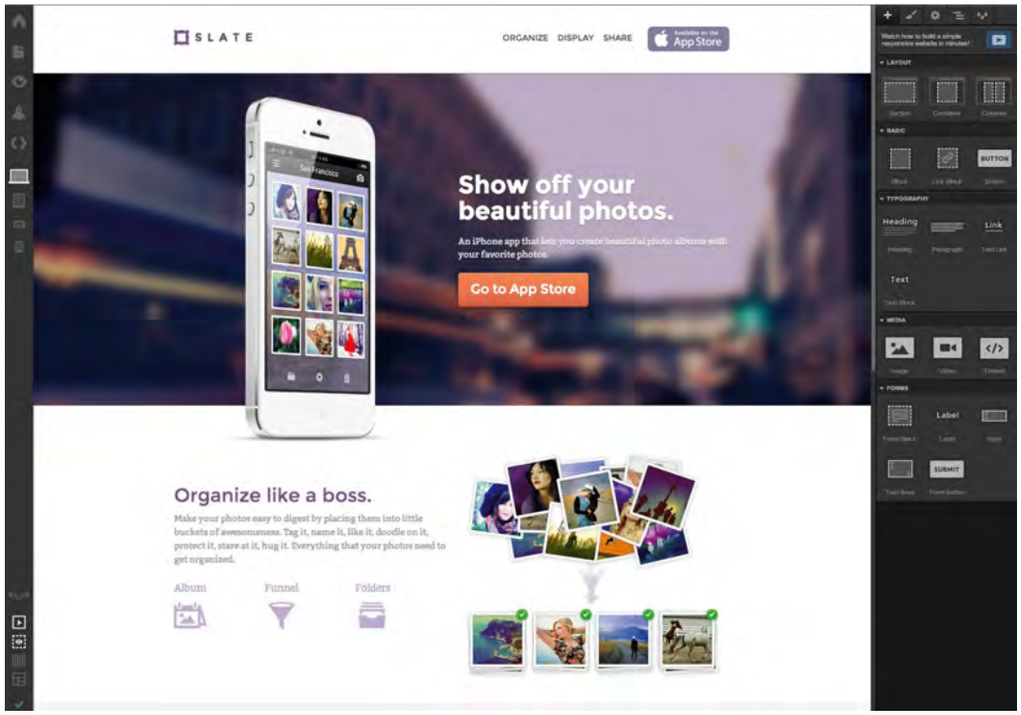
Perhaps it's taken you each and every day since Ethan Marcotte dropped the responsive web design (RWD) bomb. Maybe it happened instantly. Most likely, it was somewhere in between. Inevitably, you realised it. RWD is hard. Part of the difficulty is that it's tricky to communicate visually. How do you show a client or a developer your intention for a single element at multiple breakpoints, outside of actually building it?

Certainly HTML and CSS are great for implementing RWD and, for some, designing solely with code is a viable option. Others struggle when design is abstracted through the layer of code. However, an environment for vetting out creativity through direct manipulation is necessary.

Until fairly recently, such an environment was typically an image editor like Photoshop. Though it provided a means to express our creativity, it also handcuffed us to static deliverables.

You could use an image editor to make multiple comps, but that workflow is potentially tedious, and if we're being honest, our breakpoints should be dictated by content stressing rather than convenience (only changing an element when all the others change). Our previous set of tools simply can't accommodate the complexities of relaying responsive design, visually.

Web design demands that our tools have fluid canvases, integrate web fonts and speak in the language of CSS. We have a few apps that accommodate all of that to choose from. So, how have some talented folks addressed our need for a responsive design tool? Let's take a closer look at four apps that aim to take the pain out of responsive web design.



WEBFLOW

For Sergie and Vlad Magdalin, the need for a better web design tool came out of some brotherly frustration. Project work had been going swimmingly, with Sergie handling design and Vlad handling development. However, repetition and time exhaustion started to drive them nuts. Instead of allowing their workflow shortcomings to persist, they decided to do something about it.

"Initially, like nearly every design agency on the planet, we explored creating our own custom content management system," explains Vlad Magdalin.

"Eventually, that effort led us to think about a purely visual web design approach, which got both of us really excited, and we started working on a prototype," he adds.

That prototype was the first version of Webflow, which is now a complete web app. "From the very early days of this project, we wanted Webflow as an application to be as close to the medium as possible. Being a web application allows us to more accurately state that 'what you see is what you get', because it's the same exact code and view that will be shipped to production. For example, there's a lot more to web design than just graphic design (interaction with elements, form submission, resizing behaviour, zoom/accessibility behaviour), so it really helps to be working as close to the final product as possible. Being an in-browser application helps us achieve that," says Vlad.

Where Webflow really excels is in its versatility. Certainly there's the layout tools and if you only use it for interface design, you'll be quite satisfied by the app's intuitive controls. However, there's another layer to Webflow that may appeal to some: optimised and accelerated hosting with site backup and versioning. Theoretically, you could start building a prototype, refine it and have production code ready to be hosted, all within Webflow.

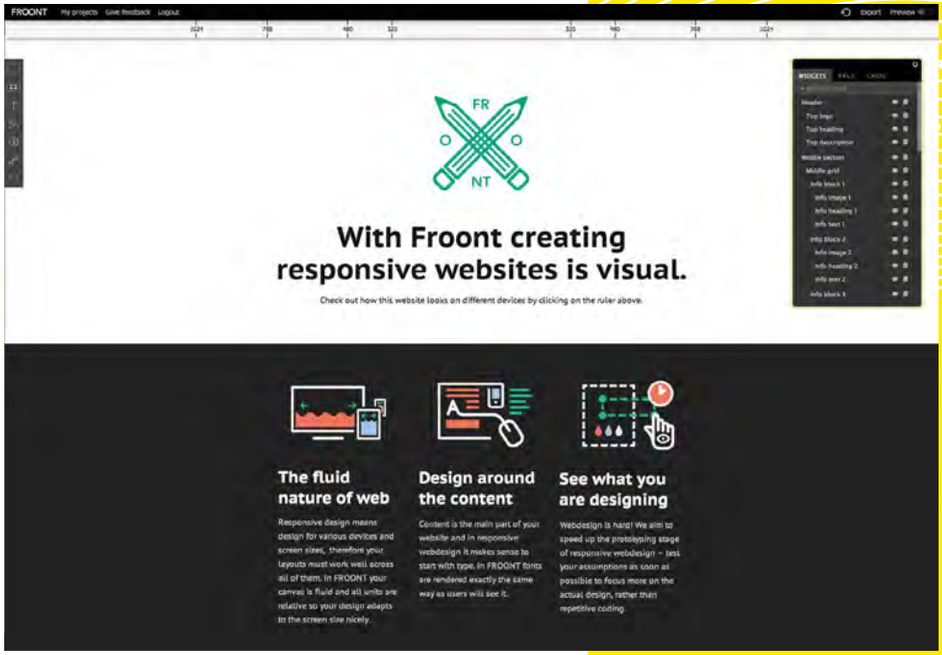
However, Webflow isn't without its constraints, and that's a good thing. For example, as liberating as it is to draw a box and drag it wherever you'd like, sometimes it's to the detriment of web standards. "We deliberately enforce the constraints of the CSS box model (margin, padding), because that enables a lot of downstream benefits (such as automatically responsive layouts and fluid designs)." You may sacrifice complete autonomy, but it affords you a design that becomes responsive by default.

"It's the beginning of a new web design platform," declares Vlad. "We'll soon unveil other features that will help web designers work much more efficiently and avoid doing a lot of repetitive work."

So who should use Webflow? Seems like if you're looking for a turnkey app, from prototyping to deployment, this could be the tool for you. That doesn't exclude those looking to use it for one specific task, but Webflow is certainly more than capable of being your workhorse.

BUY

w: *webflow.com*
t: @webflowapp
Platform: Browser
Pricing: $16 a month for personal (20 projects). $35 a month for professional (50 projects)



BUY

w: *froont.com*
t: @froontapp
Platform: Browser
Pricing: $9 a month for Scout (100 public projects). $19 a month for Freelancer (500 public projects)

FROONT

Sometimes the best ideas come from previous ones. As was the case for San Francisco designer Sandijs Ruluks who, just a few years back, launched a simple, WYSIWYG content management system, Berta.me (*berta.me*). The original intention was to just use it for his own site, but, after sharing it with friends, the user base grew to the point where he could charge for hosting. There was still one problem: the inability to design for small screens.

Ruluks proceeded to introduce a breakpoint ruler, grid controls and a handful of other responsive-focused features. Out of this, FROONT was born. It's a seemingly logical sequence of events, yet it validates the importance of iteration and the benefits of addressing your users' problems.

"Our focus is on things that are painful or impossible to make with the current set of tools. For example, we aren't planning to add ability to draw custom shapes, as Photoshop and Sketch are good enough for that. Instead, we focus on best practices in responsive web design, web fonts and interactivity. We aren't going to the direction of a CMS and we are totally against full design automation; that's not where the innovation lives," says Ruluks.

Probably the most notable difference between FROONT and the other web design tools has to do with your wallet. It is currently free to publish

ADJUSTING THE WEBFLOW WORKSPACE

Sometimes the default workspace setup doesn't give us enough breathing room, or perhaps it emphasises panels we won't use frequently. Even though it's a web app, you can adjust the Webflow workspace quite easily:

1. **Auto-Hide the Left Rail:** click on the double-headed arrow at the very bottom. The effect is much like auto-hiding your dock on Mac OS X, providing you with just a bit more elbow room.
2. **Collapse Panels:** in the right rail, simply toggle the arrows to collapse/expand panels. It may make sense to have only those panels related to a certain task open at a time (for example, creating the entire layout, refining multiple pieces of text).
3. **Go Fullscreen:** it's easy to give this web app a desktop feel. If you're on a Mac, pull it out in its own window and make it fullscreen (`Shift+Cmd+F`). Since Webflow gives you device view modes, it negates needing to have your browser window re-sizeable.

up to 10 public sites using FROONT, and it gives you HTML and CSS, should you want to use that elsewhere. Ruluks promises that FROONT "will be free as long as projects are public." Private projects are also available on paid plans.

Why's that important? Perhaps paying for a next-gen design tool is a worthy investment for someone using it from prototype-to-frontend development, but what about a designer just looking to experiment? An argument can always be made for the worth of a tool, but it's refreshing to have options when there's so many workflows out there.

TESTING MADE EASY

One of the benefits to using a browser-based design tool is the accessibility of the browser itself. To preview your work from a native app on a mobile device, some intermediary app needs to exist. Being able to visit a URL on your phone to preview your FROONT project makes the process extremely simple. "Everything you make is accessible via any internet-ready device simply by typing in the URL, so that designers can test their designs in a 'real life' environment," adds Ruluks.

> "One of the benefits to using a browser-based design tool is the accessibility"

Sharing is just as easy. FROONT doesn't require any uploading or extra steps outside of the app. Simply supply the project URL and discuss with your client.

"Having projects in the cloud makes it easier to collaborate and iterate," Ruluks points out.

RESPONSIBLE BREAKPOINTS

Every app that wants to play in the next-gen sandbox needs to tackle how we visualise breakpoints. FROONT's approach is fairly intuitive: you're given the 'famous four' of 320, 480, 768, and 1024px along the top of the app. You can edit any of those by simply 'scrubbing' to another value.

It's perhaps not as autonomous as Reflow, but still allows you to shape your breakpoints to your content, as opposed to specific devices. Any way you look at it, the ability to adjust breakpoints lends itself to being content-centric.

REFLOW

When discussing design tools, Adobe is naturally part of the conversation. It's earned that right with its contributions of Photoshop, Illustrator, Fireworks (RIP 2013) and the like. But the keen eye has been focused on its Edge tools, a lineup of apps geared towards addressing the difficulties web creators encounter. Reflow is the responsive design tool within that said lineup.

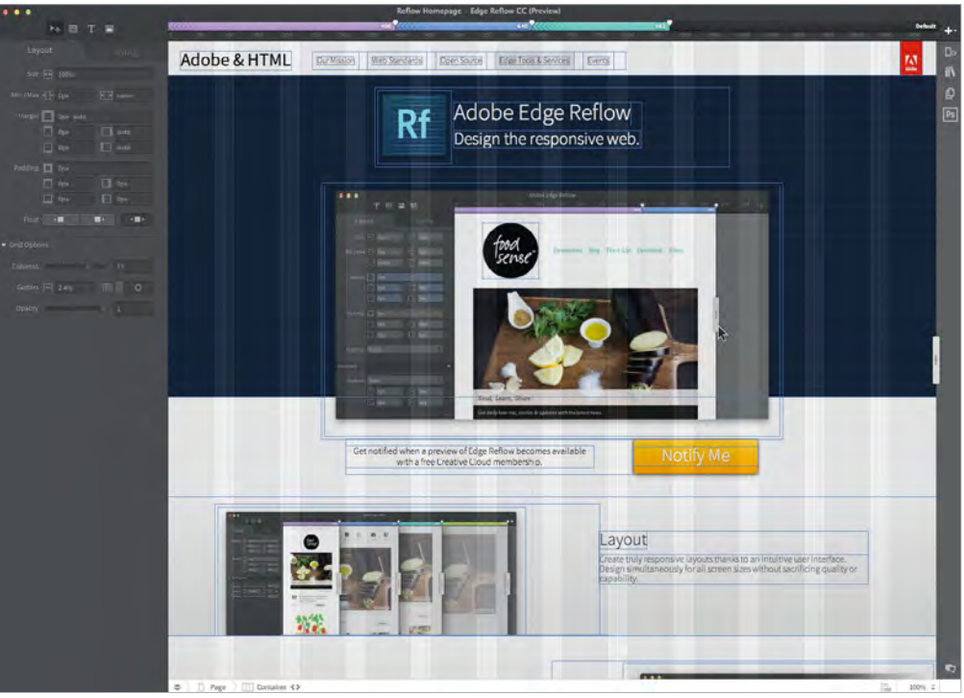
The premise is straightforward: "Reflow focuses on helping designers communicate their vision and responsive intent. Designers today don't have the luxury of throwing static comps over the wall and hoping for the best," explains Jacob Surber, Reflow senior product manager.

With Photoshop, there was a responsive communication problem. How do you show design intent with static comps of absolutely sized and fixed positioned elements? With Reflow, the canvas is fluid and the elements are relative, allowing you to better communicate how elements adapt at different screen sizes to both clients and developers.

The visualisation of the breakpoint ruler is impressive. Zones are created along the ruler, sitting above the app to indicate where media queries occur. Each is assigned a different colour. This proves useful when navigating around Reflow. While other apps may dial back the approach to visualising breakpoints, Reflow's 'at-a-glance' functionality puts breakpoints centre stage. Feel free to add as many breakpoints as you deem necessary. A conscious decision by the Reflow team allows the content you create to be the catalyst for media queries rather than popular device widths, which are subject to change any time a new Apple device launches. "If you're going to design your responsive site for more than iPhone, iPad and desktop (which you should be), Reflow lets you resize the design surface, see where your content breaks down, then add a media query," explains Surber. "Instead of devices dictating breakpoints, your content leads the way, ensuring designs look great no matter what screen they're viewed on."

PHOTOSHOP INTEGRATION

When's a good time in your workflow to use Reflow? For some, right from the beginning makes sense. Others still prefer to start exploring design in Photoshop, whose static exports can be a dead end in terms of responsive. Fortunately, there's the option to import your Photoshop document in to Reflow.



Get started in the former, and fine tune atop a fluid grid in the latter. As Surber details, "Once the assets, text and static layouts are extracted from your PSD, Reflow converts them to per cent based HTML and CSS then relatively positions all of the items. From here, you can add structure and hierarchy."

A NUDGE OF CODE

One of the most important questions next-gen design tool teams need to ask themselves is: will we promise production-ready HTML and CSS? We designer and developer types tend to be finicky about the purity of exported code. Reflow presents a nice compromise: CSS snippets are available for those who wish to use them (through a pop-up panel), but it's not a key cog of the interface or intended workflow. HTML is exported in the assets folder alongside your Reflow project, but again, it's only there should you choose to use it.

Reflow isn't intended to be a publishing tool. It's a design tool that helps bridge the gap between design and responsive development. If you're looking for a way to get cleaner, more optimal code (it tends to be heavily reliant on `<div>`s and `<p>`s) out of the app, try Reflow Cleaner by Terrence Ryan *(blog.terrenceryan.com/reflow-cleaner)*.

DEVELOPED IN THE OPEN

From the beginning, the Reflow team has taken the approach of "developing in the open", which is quite admirable. Customer feedback has influenced a great deal of the current features, and the discussion occurs everywhere from Twitter to forums, to a pre-release program. As a result, Reflow is unconfined to the 30-day download limit associated with a free Creative Cloud account. Perhaps that will change down the road, but it's a great way to experiment without commitment.

BUY

w: *html.adobe.com/edge/reflow*

t: @Reflo

Platform: Native app

Pricing: Included with Adobe Creative Cloud subscription

CONNECTING PHOTOSHOP TO REFLOW

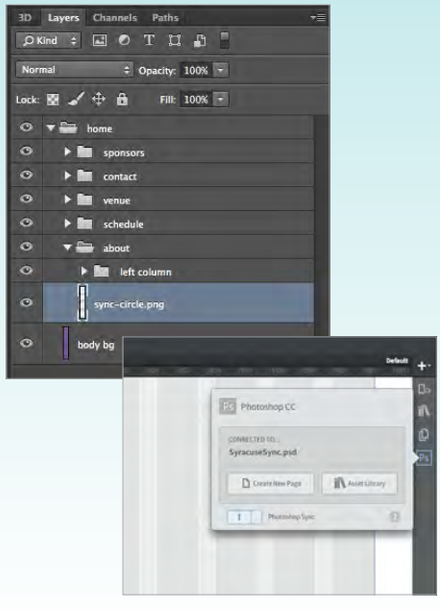
1. In Photoshop, open a PSD and add suffixes like `.jpg` or `.png` to the names of layers, groups or Smart Objects. Keep this PSD as the "active tab" if you have multiple PSDs open.

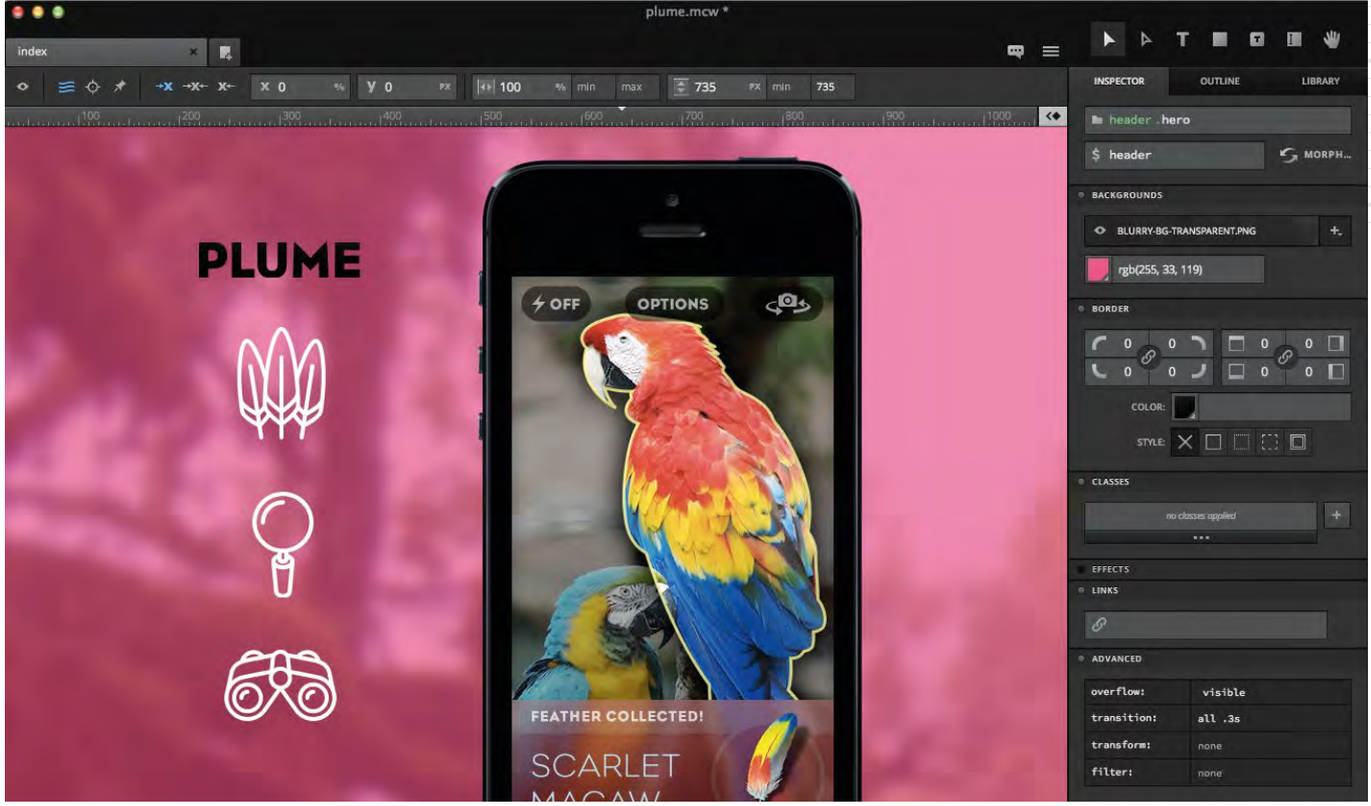
2. Switch over to Reflow and locate the Photoshop Connect panel to the right of the app (it's a 'Ps' icon).

3. Choose `Create New Page` and watch your PSD import to a new Reflow project.

Better still, the two apps continue talking to one another after you've imported a PSD. By turning on `Photoshop Sync` within the Photoshop Connect panel in Reflow, any style change made on the Photoshop side will automatically update the Reflow project. All in all, this workflow provides flexibility and connectivity between an app most are familiar with (Photoshop) and one you're learning (Reflow).

(Note: you must be using the latest versions of Photoshop CC and Reflow CC.)





MACAW

BUY

w: *macaw.co*
t: @macawco
Platform: Native app
Pricing: $179 for professionals

Traditionally, having a design app produce your code is like having a toddler file your taxes. You may get it to work, but it'll be a mess of unintuitive spans, IDs, and non-breaking spaces. Macaw not only seeks to change that, but suggests that the CSS it produces could be more succinct than what you'd write.

Longtime Photoshop proponent Tom Giannattasio realised a need for something better suited for web design. "It wasn't until 2012 that I devoted myself to building something," says Giannattasio. "I started with simple experiments and eventually left my job at MIT to do it full time. I eventually convinced my partner (Adam Christ) to come on full time."

The team of two have since drawn the curiosity of the web design community with their "pet" project, Macaw. Some screenshots here, a sneak peek video there (see the resources links opposite) and a wildly successful Kickstarter project that was fully funded within the first 24 hours, the Macaw guys clearly have the attention of an industry.

BEING CODE-SAVVY

Besides having the requisite user interface with CSS units and responsive breakpoints on a fluid canvas, Macaw claims to be "the code-savvy design tool". Production-quality code isn't just a feature of the app, it's arguably what sets it apart from the others.

How does it manage to produce clean HTML and CSS? In a word: "Alchemy". That's Macaw's under-the-hood code engine that produces code to rival a top-flight developer. "On the HTML side, it gives the designer full control over the semantics of the document by simply naming each element using a simple syntax," Giannattasio explains. It's as simple as naming a layer to produce the correlation.

Most impressive is how Alchemy handles CSS. When asked about the magic behind the scenes, Giannattasio replies, "The tags and classes set for the HTML are then used to consolidate the CSS in the most succinct and practical way possible – taking into account concepts like specificity, applicability, cascading, advanced selectors and media queries." That's no easy task, given the innumerable ways we could approach CSS. However lame the pun, Macaw truly leverages its bird's-eye view: "Because it has oversight of an entire project, it often writes styles more succinctly than I would think to."

SOMETHING FOR YOU AND ME

When designing your code, it's natural that there's benefit for interface designers as well as frontend developers in using Macaw. Designers should find the HTML and CSS assignment of elements and classes intuitive, while developers will appreciate the clarity of the code that's output. Both effectively 'touch' the code, and a convergence occurs.

"Being a native app provides an overall better experience. Many people expect the next-gen design tools to live within in the browser; we believe the next-gen design tool will be powered by a browser. It's a subtle but important distinction. Typical consumer-facing browsers are incredible tools, but hinder the user experience of a professional tool because they're designed for browsing. We want to provide an optimal design experience and avoid fighting things like keyboard shortcuts that can't be (easily) overridden."

One of the best things that RWD has taught us is that prescriptions for the only way to do something are prone to fail. Likewise, any of the tools mentioned could be the one you've been waiting for, or the one you'll never use. It depends on a number of things: team structure, project scope, budget and a whole lot more. These tools are fantastic steps forward from the traditional ones we had, but none can truly be effective if they're viewed as 'the one way to do RWD'. There's simply more to it than that.

How interesting it could be if you use one tool for a small site and another for a larger project? Or one tool for communicating design intent and another for publishing? The good thing about each of these next-gen tools is versatility. Each is capable of being a workhorse if you take the time to utilise features like pages, custom breakpoints and code export. Likewise, each is capable of making small tasks easy. Just remember that the 'right' tool is often the one that allows you to complete the 'right' task, proficiently and efficiently.

ON IMAGE EDITORS

Part of the hype surrounding these tools is their potential to be the 'Photoshop killer'. While it's tempting to place such a distinction on any of these, image editors aren't going anywhere in our design workflows. CSS certainly gives us the means to implement style, and all the tools mentioned here provide a means of direct manipulation on a canvas. However, none are environments suited for creating raster or vector assets, nor do they intend to be. As Giannattasio declares, "We're not trying to replace Photoshop … Macaw works best alongside Photoshop and Illustrator to help designers get their assets into an interactive medium."

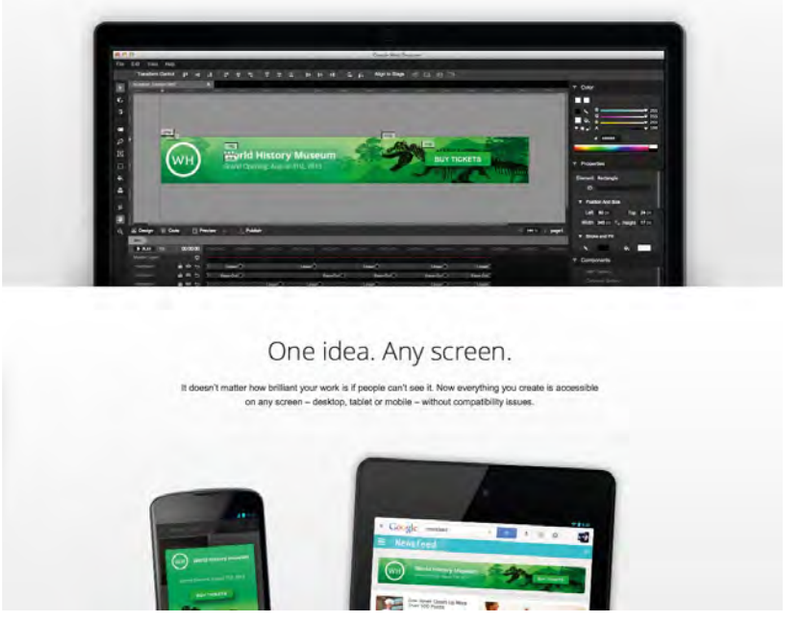
By repurposing your image editor as a high-fidelity sketchpad of sorts, you may set your self up for success before, or even while, using a next-gen design tool. Reflow's Photoshop integration is an example of a workflow that takes what you 'sketch' in an image editor and prepares it for interactivity.

The idea isn't necessarily to cut down on the amount of tools, but use them for better, more efficient purposes.

SUPPORT AND REAP THE BENEFITS

As a community, it's important that we support and contribute to the development of our tools. The silo mentality that's perceived by users and placed upon software teams doesn't exist with the products highlighted in this article. Each app has a small team of talented people genuinely reaching out for input from designers and developers. It's easy to complain about what a tool is missing, or how it should intuitively know your particular workflow, but it's our duty to communicate with the folks behind these apps and let them know how they can be improved.

Webflow, Reflow, FROONT and Macaw each have the potential to impact your responsive design workflow. Even beyond these four talented teams are designers and developers racing to offer tools to address a once barren landscape. Two years in and these tools speak of how bright the future is for addressing this whole 'responsive' thing. Experiment and see what works for you.



Good option Google Web Designer is useful if you're looking to start toying with animation. The feature set caters well to ad production

RESOURCES

REFLOW
Introduction tutorial (*gotoandlearn.com/play.php?id=178*)
Reflow Cleaner (*blog.terrenceryan.com/reflow-cleaner/*)
Adobe TV Reflow Channel (*netm.ag/tv-249*)
WEBFLOW
Webflow University (*tutorials.webflow.com*)
FROONT
Introduction video (*vimeo.com/63723477*)
MACAW
Sneak peek video (*macaw.co/peek*)
Interaction design video (*macaw.co/interact*)
MORE RWD TOOLS
Google Web Designer (*google.com/webdesigner*)
Easel (*easel.io*)
Divshot (*divshot.com*)

Download the files here!

Download all the files you need for this tutorial at *netm.ag/macawfiles-256*

ABOUT THE AUTHOR

TOM GIANNATTASIO

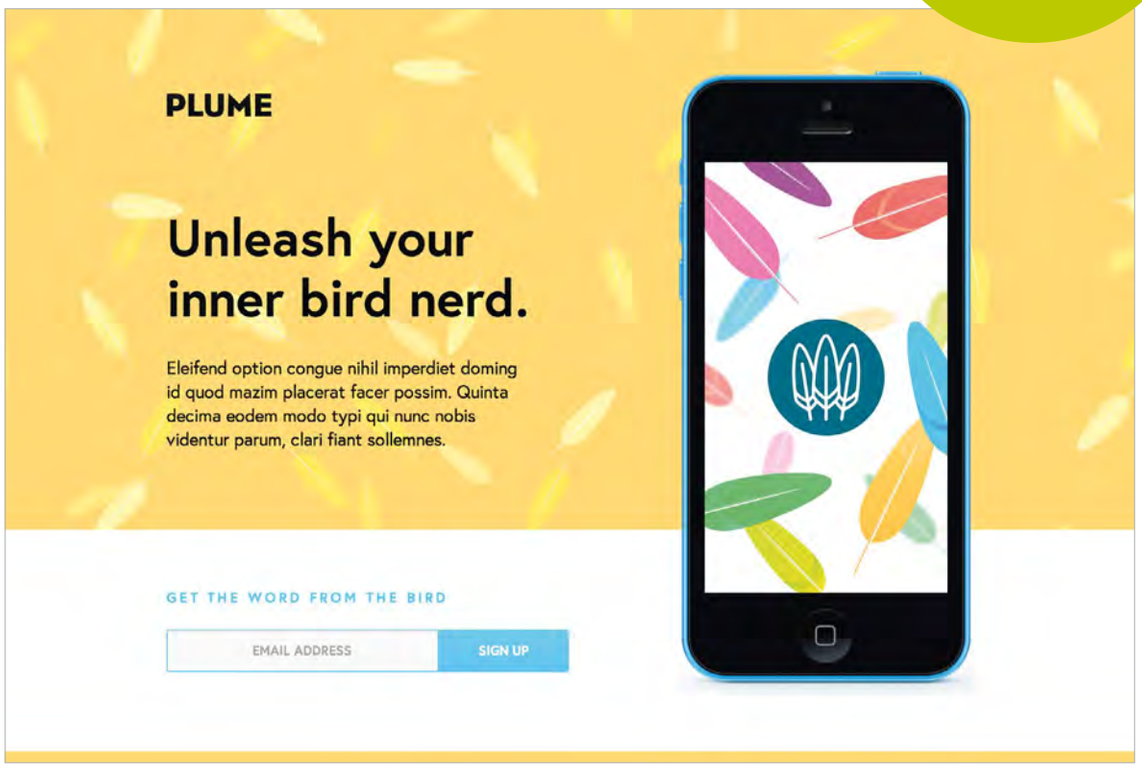
w: *macaw.co*

t: @attasi

areas of expertise: Interaction design, CSS3, JavaScript

q: which living person do you most despise?

a: Justin Bieber and King Joffrey



*RWD

CODE-FREE RESPONSIVE DESIGN WITH MACAW

Macaw is a new breed of web design tool – **Tom Giannattasio** explains how it can boost your responsive design workflow

Responsive design is time-consuming and difficult, but Macaw can help alleviate some of those woes. It was built specifically for the modern web, with speed and ease of use in mind. Designers familiar with Photoshop or Fireworks should feel right at home within its interface.

Macaw enbles you to draw and manipulate elements on a canvas. The usual suspects are all present: transforming, drag and drop, nudging, and a new friend called pudge. Elements can be grouped and depths can be managed as if they were layers.

The difference is that all of these familiar manipulations are being translated into HTML and CSS on the fly. When you move an element around the canvas, Macaw automatically calculates the margins, floats, clears and other properties that are necessary to place that element in a static document flow.

Macaw is built on the same framework as Google's Chrome browser and therefore inherits its rendering capabilities. This opens a wealth of opportunities, especially when it comes to responsive design. You can simply resize the canvas, insert a breakpoint and optimise the layout for different screen sizes, all within a single file.

In this tutorial, I'll show you how to use Macaw to create a responsive coming soon page without touching Photoshop or a single line of code.

RESOURCE

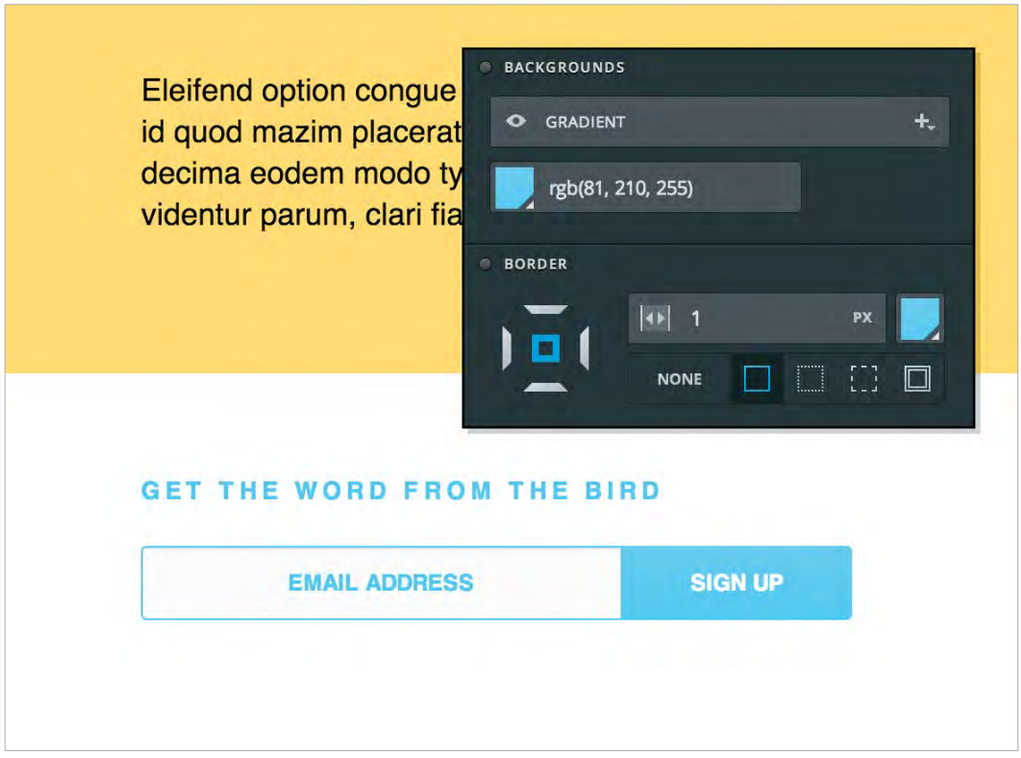
MACAW DOC

This provides an in-depth description of Macaw's features, and is scattered with expert tips on improving your speed and workflow: *docs.macaw.co*

01 The canvas in Macaw can be resized at any time to see how your design flows in different viewports. It also has a fluid, column-based grid, which can be modified to your liking. The grid provides useful aids during the design process. For example, click Cmd+right or left to nudge an element perfectly to a grid column.

For this project, we'll use a 12-column setup, with a 100% width grid and 3% gutters. You can set these properties in the Grid palette, which is visible when no elements are selected.

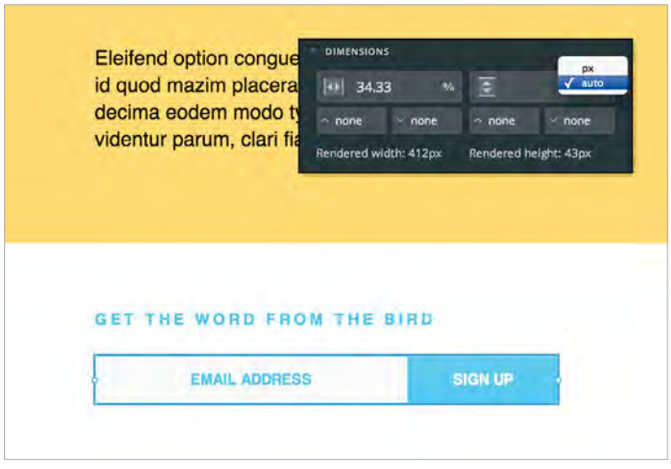
02 Adding elements is as simple as drawing shapes in Photoshop. Select the Input tool (N) and draw a shape on the canvas below the orange header. You'll notice a blinking cursor inside the shape, which indicates that this is a text editable element. Type the words 'Email address' and press Cmd+Return to commit. This will be used as the placeholder text when published.



Styling up It's simple to edit colours and border properties, and add gradients or shadows in the Backgrounds palette

Drag the input into place below the blue header. If your grid is visible, you should be able to snap the input into place. Note the blue guides that appear when moving an element. These are positioning guides, used to help visualise an element's margins and coordinates.

Draw a button next to the email input using the Button tool (B). Again, you can directly edit the text used in the button. Type 'Sign up' and press Cmd+Return to commit.



Creating containers Containers are similar to Photoshop groups, but can receive styling

03 With our elements in place, let's add some styling and advanced options. Select the input using the Select tool (V). The Inspector will update with applicable options. In the section labelled Input Options is a drop-down that lets you set the input's type. Change it to Email .

Now select the button. In the Backgrounds palette, click the colour icon and select a bright blue. Click OK and continue adding styles to the input and button. It's easy to experiment with gradients, shadows and border properties.

04 Select both the input and the button (hold Shift while using the Select tool). Press Cmd+G to group the elements together in a container. Containers are much like Photoshop groups, but they have actual dimensions and can receive styling. They also gain abilities like

EXPERT TIP

NUDGE AND PUDGE

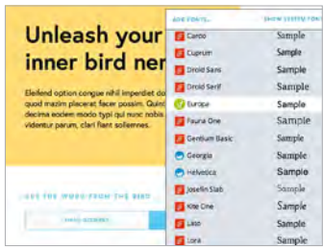
Macaw has two extremely useful commands that make working with a grid a breeze. You can nudge elements around using the arrow keys. However, you can also pudge an element's dimensions by holding Alt while pressing an arrow key. Even more useful, if you hold Cmd+arrow, this will nudge an element left or right by a grid unit, and Cmd+Alt+arrow will resize an element by a grid unit.

auto height, which allows it to shrink or grow based on the rendered height of its children. Change the container's height to auto by clicking the `px` suffix in the `Height` field in the `Dimensions` palette and select `Auto`.

05 Let's add some beautiful fonts using Typekit. First, you will need to create a kit on the Typekit website (*typekit.com*). Be sure to add 'localhost' to the kit's allowed domains.

Back in Macaw, dive into the container you created in step 4 by double-clicking it using the Select tool. This lets you work inside the container without affecting elements outside of it. Select the input and button. Click the `Font Name` field in the Typography palette to open the Font Picker. Click the `Add Fonts...` button and add your Typekit ID in the dialog. Now the Font Picker will include the fonts from your Typekit account.

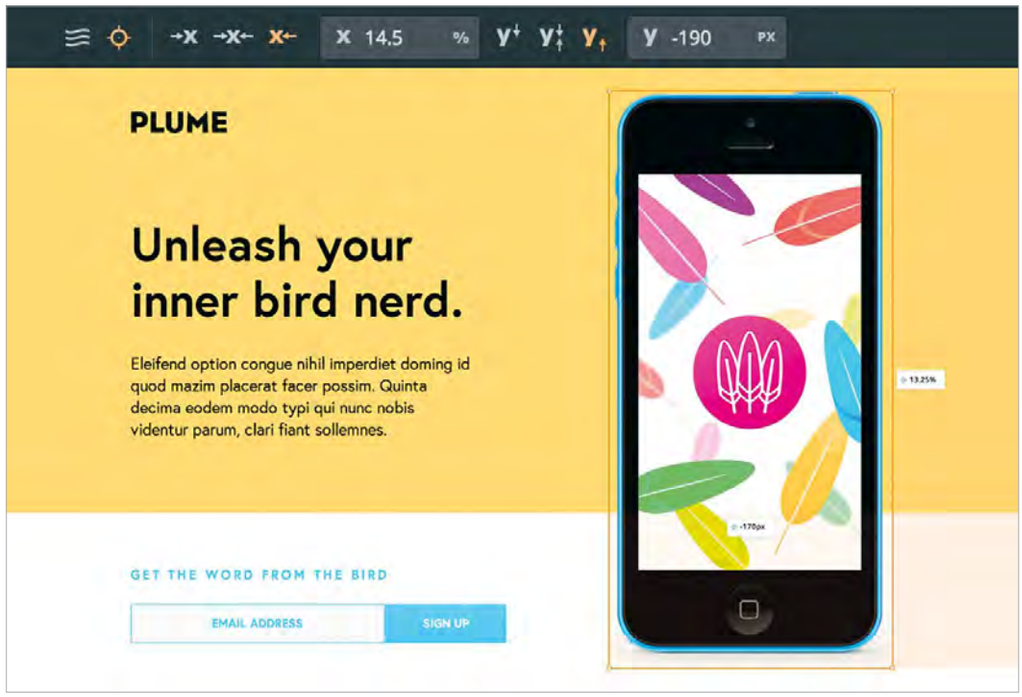
Change the fonts on your form and the text elements in the header. You may need to dive in and out of containers to access these



Type time Add fonts from a Typekit account

elements. Double-click a container to dive in and double-click outside of it to dive out. You can also use the breadcrumb bar at the bottom to jump to specific locations.

06 Let's add some images to that boring header. Click the Library tab. This is where you manage a project's assets. You'll see that there are a few images already in the library. Macaw stores assets in the mcw file so you can pass it along to others without losing references.



Adding images Macaw will automatically convert retina-ready images to make them suitable for high-resolution displays

We'll add the phone to the page. Double-click on the header, then click and drag phone.png from the library to the canvas. This is a retina-ready image and, if you're using a high-resolution display, Macaw will automatically convert it. You can check by navigating to the Image palette in the Inspector. This palette lets you examine the current size of an image in relation to its original size. Make sure that the `@2x` icon is selected.

We want to position the phone so it's always extending below the bottom of the orange section. To do so, we'll use absolute positioning. Select the phone and click the crosshair in the Property bar. Set the x origin to `right` and the y origin to `bottom` using the x and y icons in the Property bar. Now move the phone so that it's positioned to the right of the text and extending below the orange. Even though the header has an auto height, the phone will now extend down as we'd like it to.

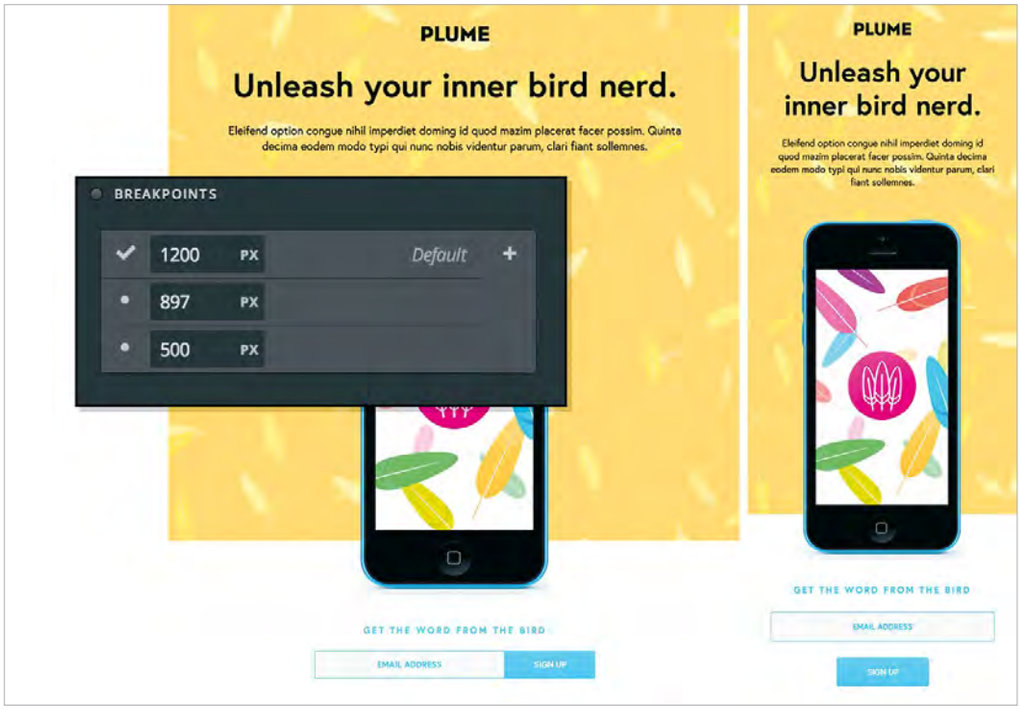
07 We'll now add some pizzazz to the header by adding a background image. Click `body` on the breadcrumb bar. Select the header and click the `+` button in the Backgrounds palette. Choose `Image...` from the drop-down. This opens the background image dialog. Click `Choose from library ...` and select `feather.png`. Set it to cover, using the size drop-down to ensure it always fills the orange header. There are plenty of other properties in here, so feel free to experiment.

08 To make sure the design holds up in different viewports, scroll to the right of the canvas and move the drag handle to the left to resize the canvas and watch your elements flow down the page. Macaw has automatically calculated the document flow, so you can see exactly how your design will behave in the browser.

EXPERT TIP

TEXT SHORTCUTS

Macaw has filler text built in. When editing a text element, type the shortcut 'loremXY' and press `Tab` to quickly insert filler text. Replace the X with a number and the Y with a unit (w for word, s for sentence and p for paragraph). For example, 'lorem2p' will insert two paragraphs. Typesetting is also simple, as calculations such as characters-per-line are visible in the bottom-right of the Typography palette.



Breaking point Breakpoints can be added wherever you like – alter your layout, and toggle between breakpoints on the Breakpoint palette

EXPERT TIP

INTERACTIVITY

For interactions, add a variable name to an element in the Inspector palette and Macaw will populate a scripting variable for it. In the Scripts palette you'll see a list of variables. Write in any JavaScript or jQuery, and Macaw will wrap your scripts in an anonymous function, populate the necessary scripting classes and generate selectors to make your scripts work.

Let's add a breakpoint. While resizing the canvas, you'll see a tooltip appear, indicating the width. Drag the handle down to a width of 900px and press `Cmd`. Click `Yes` in the alert to insert a breakpoint. You can now modify your design to cater to this specific width. Layout can be changed and styles can be reworked. You can easily toggle between breakpoints using the Breakpoints palette or by hovering breakpoints on the ruler.

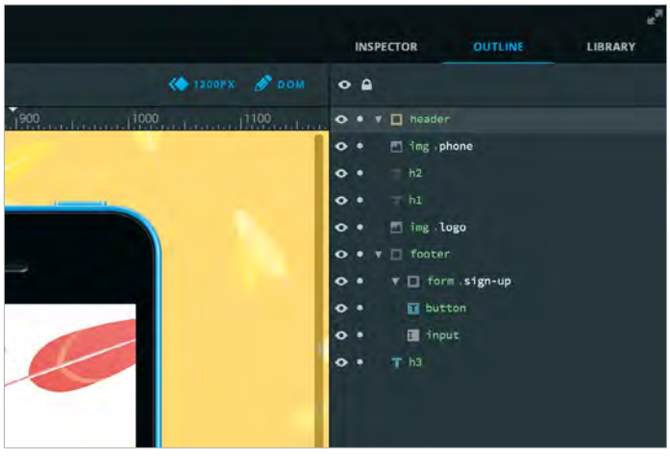
When working with breakpoints it's important to understand that properties trickle. When you change a property, it's applied to the current breakpoint. That value will trickle to the breakpoints below it, but it does not travel upwards. If you have different values set for a property across breakpoints, Macaw will outline the field in blue. When you hover over one of these fields, a property table will display showing the values on each breakpoint. You can quickly grab a value and apply it by clicking on one of the values in the table. You can also distribute a value to all breakpoints by `Cmd+clicking` it. Go ahead and add breakpoints wherever you think the design starts to break down. I added them at 900px and 500px.

09 Macaw uses a powerful design-to-code engine to convert your document into clean, succinct HTML and CSS. Macaw will handle the heavy lifting for you, but it needs some help first.

Click the `Outline` tab. Here you can rename and organise elements in a manner similar to Photoshop's Layers palette. Renaming elements plays a key part in publishing, because it defines the semantics and class names to use.

Macaw uses a simple dot-syntax naming scheme. The first word used in the name is parsed as a tag name. If that tag exists in the HTML spec, it will be highlighted in green and used when publishing. Macaw then looks for a full stop followed by a class name. This is a simple and way to define the semantics inside your document. Select the container element you created earlier, double-click its name in the outline and rename it 'form.sign-up'. You'll see that `form` is highlighted in green because it's a valid HTML tag.

10 Our design is optimised for all breakpoints and elements are all named semantically. You can publish your document at any time by pressing `Cmd-P`. This will generate all the HTML and CSS for your project and open the preview browser. From the preview window you can ensure proper rendering, view other pages and inspect the generated code. Macaw also has a built-in feature called Remote Preview, which enables you to broadcast your design to other devices on the same wireless network. Simply navigate to the IP address shown in blue in the preview window's address bar and Macaw will automatically reload updates made to the published project to that device.



Transformation A powerful design-to-code engine will turn your designs into HTML and CSS

Download the files here!

All the files you need for this tutorial can be found at *netm.ag/frameworks-255*

ABOUT THE AUTHORS

PETER COLES

w: *fffunction.co*

t: @fatelvis

areas of expertise:
Design, frontend, tea and biscuits

q: When was the last time you cried?
a: I wasn't crying. I'd just been cutting onions

DAN GOODWIN

w: *fffunction.co*

t: @bouncingdan

areas of expertise:
User research, user interface design, interactive prototyping

q: When was the last time you cried?
a: Watching Airplane for the 273rd time

DAN REEVES

w: *fffunction.co*

t: @heydanreeves

areas of expertise:
Front- and backend development

q: When was the last time you cried?
a: I stub my little toe all the time

FRAMEWORKS

ROLLING OUT WITH FRAMEWORKS

fffunction's **Peter Coles**, **Dan Goodwin** and **Dan Reeves** put the Sassaparilla, Foundation and Gulp.js frameworks to work

Frameworks, frameworks, frameworks: if there's a direction frontend and prototyping are heading in, this is it. At their best, they are quick, iterative beasts that enable us to create interactive pieces of design to share with clients in no time. At their worst, they represent a bloated, cookie-cutter approach to designing on the web. To separate wood from trees, we thought we'd share how we use three different frameworks in our process at fffunction and how you can set them up.

SASSAPARILLA

The set-up

As the name suggests, Sassaparilla relies on Sass and Compass, so you'll need to have them installed. Installation and set-up details are at *sass.fffunction.co*. We'll assume you've followed these steps, or are using a preprocessor such as Mixture (*mixture.io*).

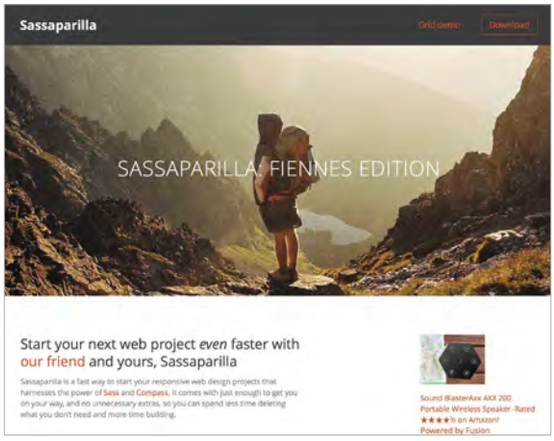
Using a simple version of Sassaparilla (*github.com/fffunction/netmag-sassaparilla*) we'll look at creating a simple page layout using Sassaparilla's built in grid.

Setting your grid divisions

Open the `_mixins.scss` file in the `libs` folder: you'll find two mixins, one for the logic generating our grid divisions, and one to define the grid HTML syntax.

@mixin grid_columns

We're taking 100% and dividing it by a number that we'll specify when we call the mixin. The



Sassaparilla Fiennes Edition Grab your copy free from *sass.fffunction.co*

function runs and outputs the syntax for each grid division. We apply a similar logic to push, pull, hide and show classes.

@mixin grid

This works with the syntax from the 'grid_columns' mixin to add style via three specific CSS hooks:

```
*[class*="colspan"]
```

Frameworks, at their worst, represent a bloated, cookie-cutter approach to web design

This defines the default values for any HTML element containing the class of `colspan`. As this is a wildcard selector, it will pick up any class with `colspan` in it. Later we'll use syntax (such as `colspan2-1`) to override this default.

```
*[class*="as-grid"]
```

Used in addition to the previous class, this tells any column to float into grid formation.

Putting it to work

In order to get this set-up running, we'll include the mixin for the grid, and then use the syntax generated by the `grid_columns` mixin to start outputting the grids.

Open up `_grid.scss` from the addons folder. At the top you'll see the include for the grid:

```
@include grid
```

▶

FOCUS ON

SASSAPARILLA FIENNES EDITION

In case you missed it first time around, we at fffunction released Sassaparilla last year and we didn't really consider it to be a framework at the time – and, in part, we still don't. However, following the addition of a super-flexible grid, along with improved typographic niceties, we concede the fact that it very well *might* be one now.

The aims of Sassaparilla however, still remain unchanged.

- To provide a foundation by which to write customisable and scalable code
- To make producing beautiful typography an easier, more flexible and more accessible process
- Include the bare minimum needed for project set up and no more. It's better to add on than take things away

fffunction use it ...

For in-browser design and for starting frontend builds (because it's extremely flexible, and super lightweight). In addition to this, it comes in useful during the process of setting up frontend style guides, where you might need code to be shared between design and production.

Although Sassaparilla can be used for prototyping, one aspect of its purpose – to remove things that you don't need – is also part of its downfall (because you need a big toolkit of elements during prototyping).

Sassaparilla Fiennes Edition was released in February 2014, a year after version one was released. In keeping with its mountaineering theme, it's named after Sir Ranulph Fiennes (tip of the hat to the great man).

This updated version features the Sassafras grid (as covered in the main section of this tutorial) in addition to an overhauled typography module and further goodies. You can grab your copy free from *sass.fffunction.co*.

FOCUS ON

THE PHILOSOPHY BEHIND SASSAPARILLA'S GRID

The grid in Sassaparilla Fiennes edition was developed with the following in mind:

● Multiple, percentage grid divisions

The grid shouldn't be confined so equal divisions of one number. If you want to use divisible by different numbers all in one design, then you should be able to.

● An inherent independence from media queries, but with the power to work within them

A grid in its simplest format is a sub-division of a parent width. In need of a four-column grid? Well that's just multiples of 25 per cent at whatever the parent width may be.

By adhering to this simple principle and combining it with a mobile-first mentality, we can harness multiple grids that become available as we go.

● Set your syntax

Your framework shouldn't set your syntax. If you'd rather write things a different way then you can. All of Sassaparilla's grid set-up is in the open, for you to alter as you please!

Box fresh How grids remain available up the stack in a responsive design

This includes the grid mixin which provides the CSS hooks for our HTML.

Now we need to set up our first grid using the `grid_columns` mixin:

```
@include grid_columns(# Number of columns);
```

This mixin must be included as a child element: we suggest nesting in a class of `.row`. So to output a two-column grid, do the following:

```
.row { @include grid_columns(2); }
```

...which would output:

```
.row .colspan2-1 { width: 50%; }
.row .colspan2-2 { width: 100%; }
```

The first number indicates how many columns exist in total (two); the second indicates how many of those columns you wish to span (one or two).

So to reference this in HTML:

```
<div class="row">
    <div class="colspan2-1 as-grid">Col A</div>
    <div class="colspan2-1 as-grid">Col B</div>
</div>
```

This gives us two 50% columns with the additional as-grid class floating them next to each other.

From here everything is possible: you can generate as many grid divisions as you need (be that six, 12 or even 20 column grids).

For more on using the grid and implementing guttered grids, read the Sassaparilla documentation.

Embedding within media queries

The real power of setting up this way is that you can add grid changes or further build upon them where you need more flexibility or change in layout.

Say you've set up a two-column grid for mobile. Then, at a certain width, you'd like an additional five-column grid:

```
.row {
    @include grid_columns(2);
    @media screen and (min-width: 600px) {
        @include grid_columns(5);
    }
}
```

By adopting a mobile-first approach via `min-width`, you now have a five-column grid available from 600px and up. You wouldn't need to add syntax to elements that needn't change at this breakpoint.

For example:

```
<header class="row">
    <div class="colspan2-1 as-grid">My Logo</div>
    <div class="colspan2-1 as-grid">My strap line</div>
</header>
<div class="row">
    <div class="colspan2-1 colspan5-3">I get wider</div>
    <div class="colspan2-1 colspan5-2">I get smaller</
div>
</div>
```

Here the header colspans would maintain proportional width (50%) while the second set of colspans would change width above 600px (50% to 60%, and 50% to 40% respectively).

This is just a basic example of the level of control possible by combining the grid and media queries.

In summary

We've barely touched the surface of what's possible with Sassaparilla, but hopefully we've shed some light on how you can be thinking about how grid systems can be build with flexibility in mind, and how frameworks don't always need to be chock-full of features to bring something useful to the table.

Frameworks needn't always be feature-rich to bring something useful to the table

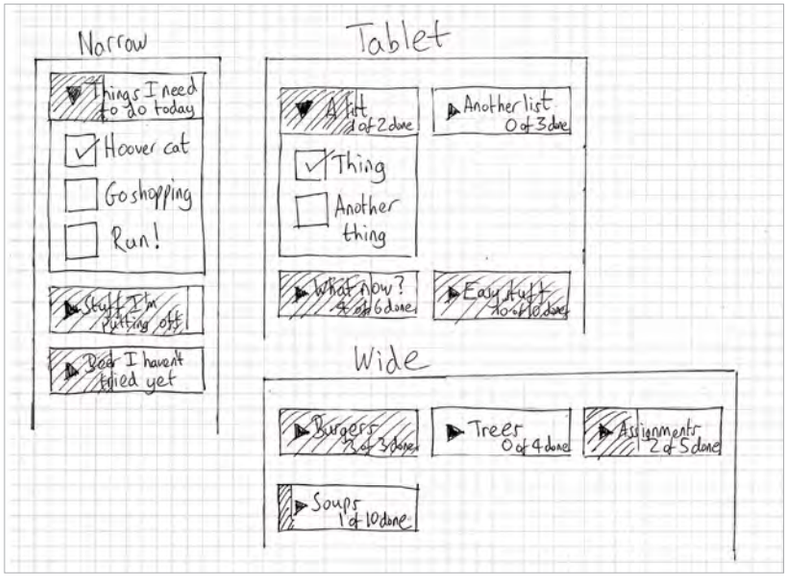
FOUNDATION

We often use Foundation (*foundation.zurb.com*) to build responsive interactive prototypes: as it comes with such a huge range of UI elements, we can quickly put these together. Because Foundation doesn't have the easy typographical finesse of Sassaparilla and because it can be hard work to tame its code, we tend not to use it in production code.

What we're going to build

We need more to-do list apps in our lives, right? I'm going to prototype a simple idea I've come up with, where to-do lists are presented as cards with a shaded progress bar to show at a glance how much has been done. Tapping/clicking a card opens it up to reveal the list with options to add items and mark them complete.

I've sketched out how this could be presented on narrow, medium and wide viewports.



Squeezy does it Wireframe sketches of our to-do app at different viewport widths

The repo at *github.com/fffunction/netmag-foundation* has the source for the various steps we work through.

To get going, I downloaded Foundation 5 in full and moved the CSS, image and JavaScript folders from the root into a folder called `assets` for neatness.

VIDEO

Watch an exclusive screencast of this tutorial created by the authors at: *netm.ag/tut1-255*

STEP 1 Use Foundation accordions to present our collapsible to-do lists

For speed, we're going to use a Foundation `accordion` element to create each card, with a to-do list being the content. The Foundation documentation (*foundation.zurb.com/docs/components/accordion.html*) shows us the markup:

```
<dl class="accordion" data-accordion>
  <dd>
    <a href="#panel1">[accordion item heading]</a>
    <div id="panel1" class="content">[accordion item
content]</div>
  </dd>
</dl>
```

In the repo, `todoo_step_1.html` presents four lists each in an accordion.

STEP 2 Use Foundation Block Grid to put our lists into responsive columns

We want our collapsed lists to be presented as cards in columns, with a single column for our narrow viewport, two columns for our medium viewport and three for our wide viewport.

The Foundation Grid can't help here, because we need to vary the number of columns (in other words, items) in each row between viewports. But ▶

the Foundation Block Grid (*foundation.zurb.com/docs/components/block_grid.html*) can.

The markup is an unordered list with classes to define the number of items in a row at the small, medium and large viewports:

```
<ul class="small-block-grid-1 medium-block-grid-2 large-block-grid-3">
    <li><!-- [item content] --></li>
    <li><!-- [item content] --></li>
    <li><!-- [item content] --></li>
    <li><!-- [item content] --></li>
</ul>
```

In the repo, `todoo_step_2.html` presents our lists in the block grid.

STEP 3 Using Foundation Icon Fonts for basic icons

To make our collapsed cards look more clickable, we'll add a simple arrow icon. We'll also add a plus icon onto the 'Add item' link in each list.

Foundation's Playground (a labs section with experimental features you can use in Foundation or in other projects) includes a simple, multipurpose icon font, which is easy to add to a project and provides icons for a multitude of tasks.

We downloaded and added Foundation Icon Fonts 3 (*zurb.com/playground/foundation-icon-fonts-3*) and added icons to our accordion heading links and to the Add item link:

```
<i class="fi-play"></i>
<i class="fi-plus"></i>
```

This can be seen in `todoo_step_3.html` in the repo.

STEP 4 CSS to present progress bar and to tidy up

Finally, we're going to add our progress bar and a textual label to each list to show how many items are in it and how many are completed. We also tidy up our prototype:

- add a heading
- add body padding
- remove bullets
- increase font sizes to make things easier to read and click / tap
- add padding on our icons
- rotate the arrow icon on an open list to reinforce toggle open/closed

In `todoo.html` in the repo, you'll see some simple BEM-style CSS classes added to the markup and an additional style sheet `screen.css`.

Where next?

We've quickly created a basic interactive prototype. We can use this o present and test our ideas with users and devices, and to iterate changes and explore new ideas.

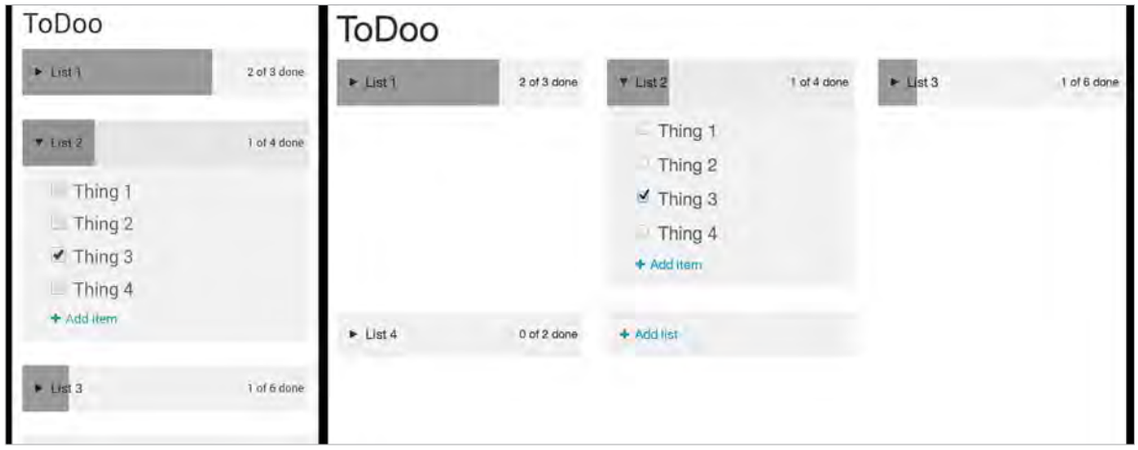
With a little extra work we could add the JavaScript/jQuery to enable lists and list items to be created, edited, reordered and deleted.

We used Foundation to get our prototype built quickly using a flexible, well documented framework. To build our app, we'd develop on a suitable implementation stack which may or may not include a frontend framework.

GULP.JS

The next piece of the fffunction framework puzzle is one around which we can write our build scripts and task automation: Gulp (*gulpjs.com*).

If you're aware of Grunt you'll know what this is about, but Gulp differs from Grunt in two key ways: the use of streams, and a code-over-configuration approach. A stream is a method of plugging the output of one tool into the input of another, enabling you to



Opening out Three-column desktop and single-column mobile prototype versions of the accordion menu

compose large systems out of small tools that do one thing well. Gulp is a framework for plugging in your favourite tools to form a build script.

Getting started

To begin we'll need Node and npm. If you don't have these, you can download installers from the website (*nodejs.org*) or, if you're on OS X, by using Homebrew (*brew.sh*).

All the files from this part of the project are in the repo at *github.com/fffunction/netmag-gulp*.

First install the gulp CLI helper, which enables you to run the build with the `gulp` command:

```
npm install -g gulp
```

Next you'll need the dependencies for this build script. We're going simple and only compiling the Sass from the Sassaparilla project. Here's the command to get everything needed:

```
npm install --save-dev gulp gulp-compass gulp-minify-css
```

Gulp.js differs from Grunt in its use of streams and code-over-configuration approach

While there are many purpose-written Gulp plugins (gulp-compass, for instance), the majority of things can be accomplished easily with existing Node modules.

The `--save-dev` option saves these packages to your `devDependencies` list in `package.json`. If you don't already have this file, I recommend creating one according to the docs (*npmjs.org/doc/json.html*) or using `npm init`. Then anyone using this project will know exactly what packages they need and npm can automatically install them.

Writing the script

We're now ready to write the script. Gulp only has four methods, so it's easy to learn, then gets out of your way, enabling you to write a build script how you like.

Create yourself a `gulpfile.js` with these lines:

```
var gulp = require('gulp');
gulp.task('default', function() {
    gulp.src('./index.html')
    .pipe(gulp.dest('./'));
});
```

```
danr@Dans-MacBook-Air ~/Desktop/netmag-gulp (master*)
→ gulp
[gulp] Using gulpfile /Users/danr/Desktop/netmag-gulp/gulpfile.js
[gulp] Starting 'default'...
[gulp] Finished 'default' after 19 ms
```

Big gulps Running our gulp script in Terminal

This is the minimum viable script. It doesn't do much, but it introduces you to three of the four Gulp methods: `gulp.task`, `gulp.src` and `gulp.dest`. Respectively they define a task, get a stream from a source file, and output it to a destination folder.

Run `gulp` in the terminal to test it's all working.

Compiling

Now we can start compiling our Sassaparilla. Import the relevant modules and write a task:

```
var compass = require('gulp-compass');
var minifyCSS = require('gulp-minify-css');
gulp.task('compass', function() {
  gulp.src('./assets/css/*.scss')
    .pipe(compass({
      config_file: './compass/config.rb'
    }))
    .pipe(minifyCSS())
    .pipe(gulp.dest('./assets/css/'));
});
gulp.task('default', ['compass']);
```

Note the default task now takes an array of tasks to call when you run `gulp` from the terminal.

After running `gulp` this time you should have a compiled and minified CSS file in your `css` directory.

Watching

The addition of a watch task will introduce us to the fourth and final gulp method (`gulp.watch`) and enable us to run gulp once and compile any changes to our styles:

```
gulp.task('watch', function () {
  gulp.watch(['./assets/css/**/*.scss'], ['compass']);
});
gulp.task('default', ['compass', 'watch']);
```

Where next?

We've got Gulp watching and compiling our Sass. If we wished to go further, we could, for example, look at linting, concatenating and minifying our JavaScript, live reloading in browsers across devices, automated build tests and other deployment steps.

RESOURCE

READING LIST

Two great resources for prototyping in Foundation are at *foundation.zurb.com/prototyping.html* and *netm.ag/dive-261*. To read more about working with Gulp, try *laracasts.com/lessons/gulp-this* and *netm.ag/gulpstart-255*.

Download the files here!

Download the source code at *netm.ag/tut1-258* or *netm.ag/tut1Git-258*

ABOUT THE AUTHOR

STEVEN WU

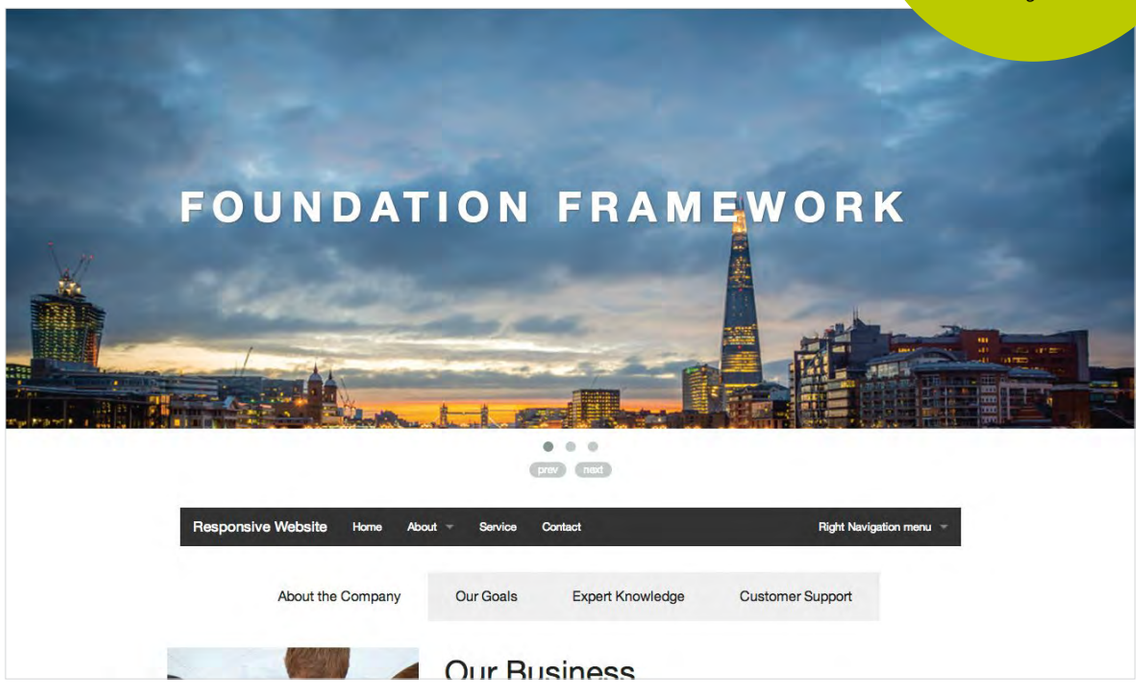
w: *designtodevelop.com*

t: @designtodevelop

areas of expertise: Magento development, WordPress theme development, frontend development

q: what's the most heroic thing you've ever done?

a: I saved Azeroth from the evil clutches of Lich King in World of Warcraft. Next stop, Deathwing!



*RWD

BUILD RESPONSIVE SITES WITH FOUNDATION

Steven Wu on how to take advantage of the latest feature-rich version of Foundation 5.0 to develop fast-loading, responsive sites

As its name suggests, Foundation is primarily used as a foundation for any responsive website project. It is a lightweight framework packed with rich features designed to quickly develop fast-loading, responsive websites. When designing responsive sites, the mobile-first approach springs to mind as the smartest way to build – and this is something Foundation has taken into account in its latest iteration.

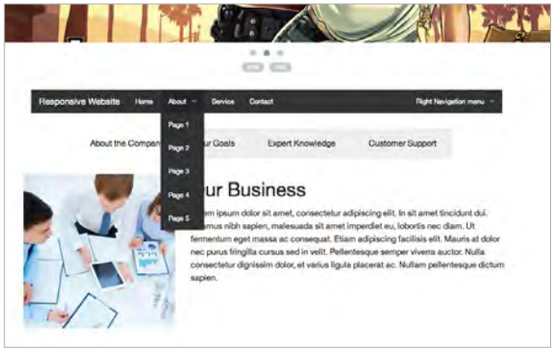
When ZURB was building the new Foundation 5.0, the main priority it had in mind was to redevelop the framework for optimum speed. Interchange – the responsive image solution – is now built directly into the framework. You can also now use any kind of content type, including images, CSS and video, and it will be downloaded directly in the appropriate format for the device type. On top of this, it is now possible to create different HTML partials, and Interchange will swap out the correct content type for the appropriate device.

jQuery 2.0 has replaced the Zepto.js library for better performance. The new off-canvas navigation is now part of the core 5.0, with CSS animations. There are also a ton of hardware accelerations, making a Foundation 5.0 lightning-fast compared to its predecessors.

Other noticeable updates to this new release include a medium grid size, the inclusion of fastclick.js to provide mobile users with a snappier experience, Bower upgrade manager, Sublime Text package to automatically scaffold your new components in your project, and much more. Using the new Foundation 5.0, we can learn to quickly build a responsive website.

VIDEO

See the tutorial in action in Steven Wu's exclusive accompanying video at *netm.ag/foundationvideo-258*



Navigation bar Foundation's versatile top bar contains complex code and provides many presentational classes to define the look of your site

DOWNLOAD AND SETUP

First, head over to ZURB's website and download the latest version of Foundation (*netm.ag/FoundationDownload-258*). At the time of writing, the newest release was version 5.3, which is the version we'll be using in this tutorial. Foundation gives you the option of writing your CSS in pure CSS or Sass. To allow beginners to follow this tutorial we'll be using traditional CSS.

Once you have downloaded Foundation, extract it and open up the index.html. This HTML document

With Foundation 5.0, the main priority was to develop the framework for optimum speed

already has most of the structure set up, including the Foundation stylesheets and other necessary libraries. Let's go ahead and delete everything after the opening `<body>` tag all the way down to just above the jQuery library at the footer of the document. We'll add in our own custom style sheet in the `<head>`:

```
<link rel="stylesheet" href="css/style.css" />
```

Now we're ready to get started.

RESPONSIVE CAROUSEL

Let's start off with giving our website immediate impact by adding in a JavaScript-based carousel. Foundation has its own built-in responsive image slider called Orbit, which has been deprecated since the new version 5.3. Instead, there are a few third-party image sliders that work extremely well with Foundation, including Owl Carousel (*netm.ag/owl258*) and Slick (*kenwheeler.github.io/slick*). I'll use Owl for ▶

FOCUS ON

GETTING STARTED WITH FOUNDATION

There are many methods for downloading and running Foundation for your latest project. Some have certain advantages compared to others.

The simplest method is to head straight to ZURB's Foundation website (*netm.ag/ZurbDownload-258*) and download the complete source. Here, you will be given a range of different options. You can download the complete version, just the bare-bones essentials in a much lighter-weight version, or customise the download by selecting which components and features you require for your particular project.

For Git lovers and those who would like the full revisions and updates on the framework, you can clone the full repository over on GitHub: *github.com/zurb/foundation*.

Alternatively, you can use Bower (*bower.io*) to manage this framework. Since the new release on version 5.0, ZURB has utilised Bower to handle the updating process of its framework.

Before proceeding, make sure you have the following installed on your system:

- Git (*git-scm.com*)
- Ruby 1.9+ (*rubyinstaller.org*)
- NodeJS (*nodejs.org*)

In the command line, run (you may need to use sudo):

```
$ npm install -g bower
```

Followed by installing Foundations CLI with:

```
$ gem install foundation
```

To set up a brand new project with Foundation simply run:

```
$ foundation new PROJECT_LOCATION
```

When working with Sass, the simplest way to compile your Sass is to use Compass. Make sure you set up your config.rb with your workflow and set Compass to watch your project:

```
$ compass watch
```

When ZURB releases a new version of Foundation, you can easily run an update by going to the root of your project and running:

```
$ foundation update
```

IN-DEPTH

FURTHER CUSTOMISATION TO THE TOP BAR

Foundation's Top Bar is an extremely complex navigation component, but wonderfully simple to implement. It's suitable for use with small, medium or large screen sizes. In this tutorial we've only touched the basics on implementing a navigation. We can further customise this by adding in neat little extras. Let's take a look at some of these.

Sticky top bar

By applying the class sticky to the containing `div` of your top bar navigation, this will force your navigation to stick to the top of the web browser when users scroll down the page. You also have the option of using the `data-options` attribute. If you only want the sticky menu to appear on large monitors you just add in `sticky_on:large`.

```
<div class="sticky">
<nav class="top-bar" data-topbar data-options="sticky_on:
large">
```

Fixed navigation

Similar to the sticky function, the fixed function will cause the top bar to remain fixed to the top of web browser, even before user scrolls through the page. Simply add a wrapping `div` of `fixed`.

```
<div class="fixed">
<nav class="top-bar" data-topbar>
```

Input fields

There are several common supported components you can add in the navigation. To apply an input field such as a search box, use the `has-form` class:

```
<li class="has-form">
  <div class="row collapse">
    <div class="large-6 columns">
      <input type="text" placeholder="Type here">
    </div>
    <div class="large-4 columns">
      <a href="#" class="alert button expand">Search</a>
    </div>
  </div>
 </li>
```

▶ this. Download the plugin, and once extracted, we want to load the plugin files, starting with the CSS:

```
<link rel="stylesheet" href="css/owl.carousel.css">
<link rel="stylesheet" href="css/owl.theme.css">
```

Owl Carousel requires jQuery 1.7, but Foundation already has this library supplied for us, so we will only need to include the JS plugin at the bottom of our HTML document just before the `</body>` tag.

Now let's build the actual responsive slider:

```
<div id="owl" class="owl-carousel owl-theme">
    <div class="item">
      <h1>Foundation Framework</h1>
      <img src="img/slider-1.jpg" alt="Cityscape>
    </div>
    <div class="item"><img src="img/slider-2.jpg" alt=""></
div>
    <div class="item"><img src="img/slider-3.jpg" alt=""></
div>
  </div>
```

Then we just need to call the Owl initialiser function:

```
$(document).ready(function() {
    $("#owl").owlCarousel({
        navigation : true,
        slideSpeed : 300,
        paginationSpeed : 400,
        singleItem:true
    });
  });
```

Let's add some styling to our carousel:

```
#owl .item img{
  position: relative;
  display: block;
  width: 100%;
```

Clearing Lightbox This component helps create a responsive lightbox gallery

Just a note Users can supply captions to each image by using the data attribute data-caption

```
  height: auto;
  z-index:10;
}
```

We've given this a relative position because we want to place a H1 on top of the carousel image. Let's position our H1 and then add in some typography styling:

```
#owl .item h1 {
  position: absolute;
  top: 40%;
  left: 27%;
  z-index: 20;
  margin:0;
  color: #ffffff;
  text-align: center;
  font-weight: 900;
  font-size: 450%;
  text-transform: uppercase;
  letter-spacing: 10px;
}
```

NAVIGATION AND GRIDS

One of the great benefits of using a framework such as Foundation is that it comes bundled with its own grid system. This makes laying out and structuring content a breeze.

The best part of the grid system is that it comes fully responsive and optimised for multiple device types. To use the grid system, we just need to wrap our content in a containing `div` called `row`. This creates our horizontal block. Then, to create our vertical columns we use the `div` class called `columns`. We can then specify the grid column we want – so, if we're after a large column, we just add in the class `large-12`.

```
<div class="row">
  <div class="large-12 columns">
```

Foundation's grid system makes laying out and structuring content a breeze

```
  </div>
</div>
```

Within our grid system, start by opening a `<nav>` and give this a data attribute of `data-topbar`. Within our `<nav>` tag open up a `<ul>`. The first `<li>` item will be used as our burger menu on a mobile device, and this will not be seen by a desktop browser.

```
<ul class="title-area">
  <li class="toggle-topbar menu-icon"><a
href="#"><span>Menu</span></a></li>
</ul>
```

RESOURCE

EXTRA DOCS

To learn more about this responsive framework, check out ZURB's Foundation website for the full documentation on how to use all the components it has to offer: *foundation.zurb.com/docs*

Directly below this we can open up a `<section>` with a `top-bar-section` class. Make sure you have the correct classes applied. The top bar component has a lot of magic happening in the JS, and these classes help to provide presentational styles to define the look and feel of the site.

Within our `<section>` we can create a new `<ul>` with a class of `left` in order to position this to the left-hand side of our navigation. To implement a drop-down menu, we can just add the class `has-dropdown` to our `<li>` item and Foundation will do the rest of the hard work for us.

```
<section class="top-bar-section">
<ul class="left">
 <li><a href="#">Home</a></li>
 <li class="has-dropdown"><a href="">About</a>
   <ul class="dropdown">
    <li><a href="#">Page 1</a></li>
   </ul>
 </li>
</ul>
</section>
```

TABS

To neatly organise lots of different types of content into a singular container we can use tabs. Foundation has a Tabs component to enable us to easily switch between items within the container.

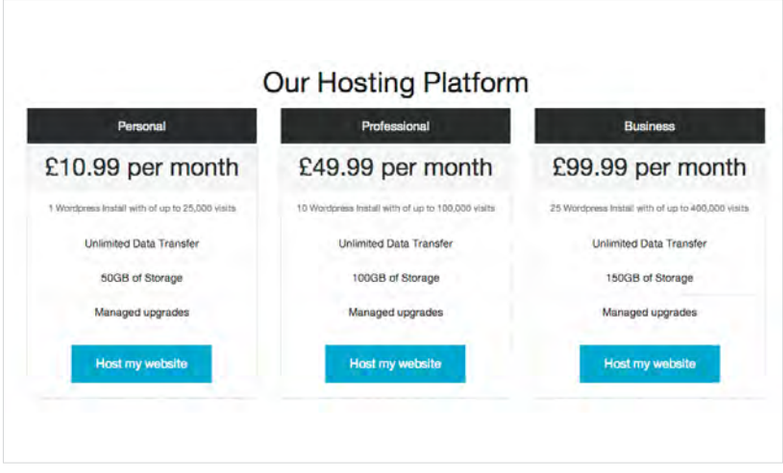
To begin with, we'll wrap our tabs within a `row` container, followed by a `large-12` and `columns` class to set up our grid layout. Using a `<ul>`, we give this a class of `tabs`, followed by a data attribute of `data-tab`.

```
<ul class="tabs" data-tab>
    <li class="tab-title active"><a href="#panel1-1">About the Company</a></li>
    <li class="tab-title"><a href="#panel1-2">Our Goals</a></li>
    <li class="tab-title"><a href="#panel1-3">Expert Knowledge</a></li>
    <li class="tab-title"><a href="#panel1-4">Customer Support</a></li>
  </ul>
```

Each of our `<li>` items will have a class of `tab-title`, and each one will act as our title for our tab navigation. The first tab will have an `active` class to be set as our default tab on page load. In order to create our content for each tab item, we simple create a `div` with `tabs-content` and within this give it an ID of `panel1-1` to reference to the associated tab.

```
<div class="tabs-content">
 <div class="content active" id="panel1-1">
   <!— Content —to go here —>
 </div>
</div>
```

Sizing up Creating a pricing table is fairly simple, with Foundation using a mobile-first approach



We can build images of variable heights into a responsive lightbox gallery

LIGHTBOX GALLERY

One of the fancy features in the Foundation framework is that we can easily build images of variable heights images into a responsive lightbox gallery. Simply create an unordered list with the predefined class `clearing-thumbs` and the `data-attribute`, then we just need to specify the large image and the thumbnail image within a `<li>` item.

```
<ul class="clearing-thumbs" data-clearing>
    <li><a href="img/gallery-large-1.png"><img src="img/gallery-thumb-1.png"></a></li>
    <li><a href="img/gallery-large-2.png"><img src="img/gallery-thumb-2.png"></a></li>
 </ul>
```

Now, when you click on the thumbnail it will expand into a photo gallery in a lightbox.

PRICING TABLE

For any awesome site that includes a subscription-based product, you'll want to add a pricing table. Foundation's pricing table is simple to implement and comes fully responsive. Within a row container, create a `<ul>` with a class of `pricing-table`, and in our `<li>` item we can give specific class names to apply special styles to certain content.

```
<div class="large-4 columns">
    <ul class="pricing-table">
      <li class="title">Personal</li>
      <li class="price">&pound;10.99 per month</li>
      <li class="bullet-item">Managed upgrades</li>
      <li class="cta-button"><a class="button" href="#">Host my website</a></li>
    </ul>
</div>
```

If we duplicate the above three times, the content will be neatly aligned, and as we reduce the window screen size it will automatically scale to fit the monitor resolution.

EMAIL SUBSCRIPTION FORM

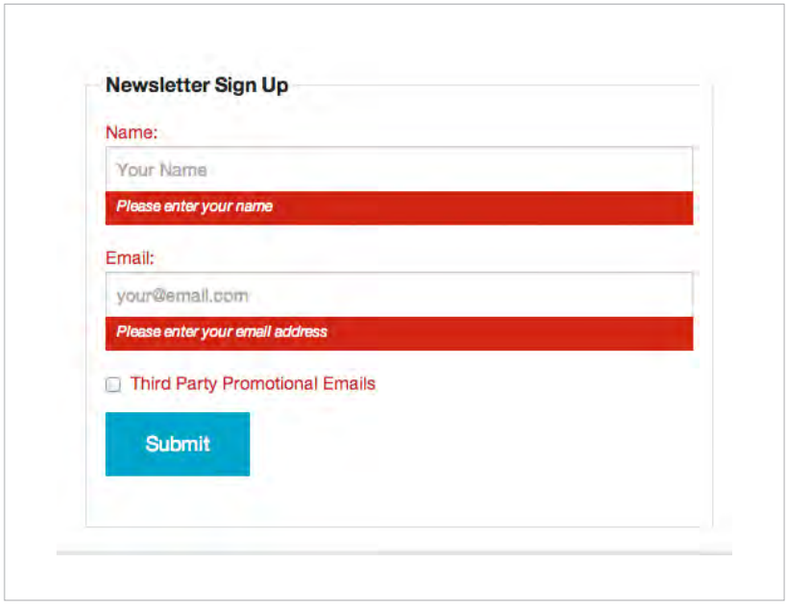
Writing your own script to validate your form can be time-consuming and tricky at times, but luckily Foundation provides us with a powerful and versatile form layout system, with Abide – an HTML5 form validation library.

To use Abide, add the attribute `data-abide` to your form element. For any mandatory input fields, add the required attribute to it. Below each label we can associate a small tag that contains our chosen error message.

```
<form data-abide>
 <fieldset class="large-6 large-offset-3 columns">
  <legend>Newsletter Sign Up</legend>
   <label>Name:
     <input type="text" placeholder="Your Name" required />
   </label>
   <small class="error">Please enter your name</small>
  <label>Email:
    <input type="email" placeholder="your@email.com" required />
  </label>
  <small class="error">Please enter your email address</small>
  <input id="checkbox1" type="checkbox"><label for="checkbox1">Third Party Promotional Emails</label>
  <button type="submit">Submit</button>
 </fieldset>
</form>
```

Button styles are also predefined in Foundation – the framework really does all the heavy lifting, while we can just use the components available to lay out our page. We can finish off our website by adding a much-needed footer.

```
<footer id="footer">
   <div class="row">
     <div class="large-6 columns">
```



Error styles Foundation offers a versatile form component, including features to quickly add validation and error styles

```
      <nav>
       <ul>
        <li><a href="#">Home</a></li>
       </ul>
      </nav>
     </div>
     <div class="large-6 columns copyright>
      <p>Copyright info</p>
     </div>
   </div>
  </footer>
```

Some simple styling for our footer:

```
#footer {
 background-color: #e3e7e9;
}
#footer ul {
 padding-top: 10px;
}
#footer ul li {
 display: inline-block;
 margin:0 15px 0 0;
}
#footer .copyright {
 padding-top: 10px;
 text-align: right;
}
```

With minimal work we were able to quickly put together a responsive, grid-based website. ZURB's Foundation framework is easy to use, powerful and provides the flexibility to prototype and build production-ready websites for any device.

REVIEW

FOUNDATION TEAM

With thanks to Jonathan Smiley, who is a partner at ZURB and works on the Foundation team, and Brandon Arnold, the design lead on the Foundation team, for their review of this tutorial.

Download the files here!

Download the source code at *netm.ag/wu-259* or at *netm.ag/bootstrapgit-259*

ABOUT THE AUTHOR

STEVEN WU

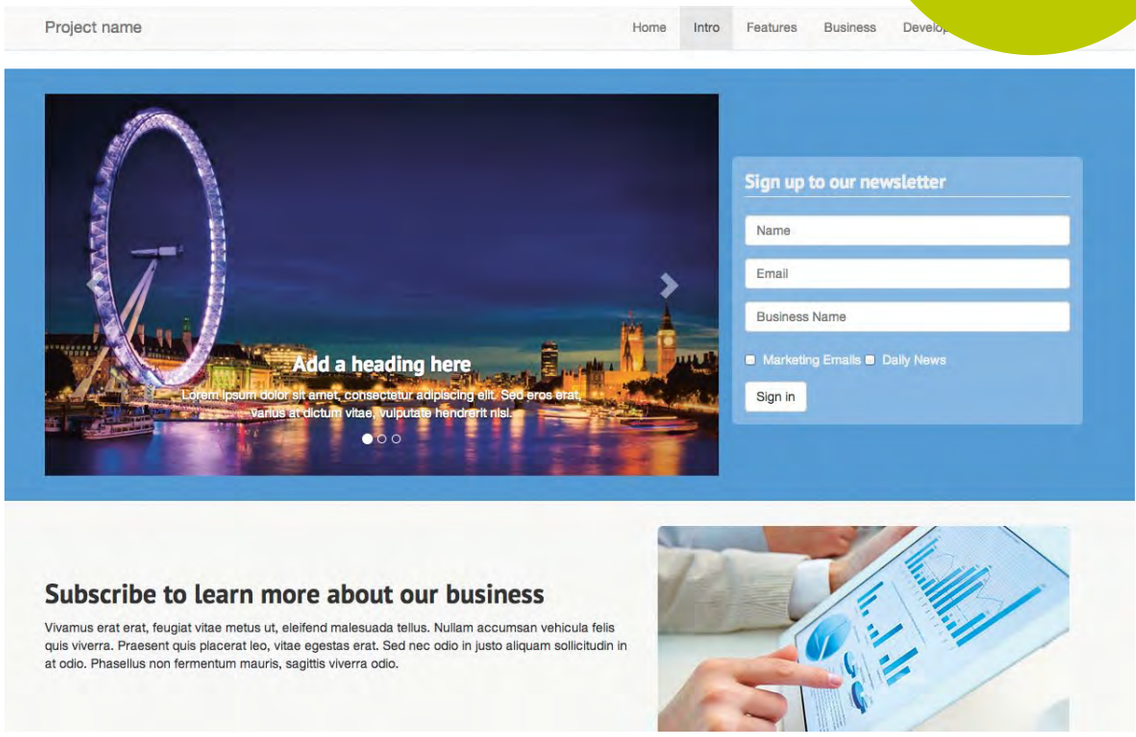
w: *designtodevelop.com*

t: @designtodevelop

areas of expertise:
Magento, WordPress, frontend web development

q: what's your opinion on Marmite?

a: It should be banned and replaced with a sweet and sour spread



*RWD

BUILD A SIGN-UP PAGE WITH BOOTSTRAP 3

Steven Wu shows you how to use the latest, mobile-friendly version of this framework to build a custom sign-up page

With the recent release of Bootstrap version 3.0, big changes have been made. Most notably, the responsive frontend framework has been rebuilt from the ground up to be mobile-first. This means whenever you start using this framework, your website will be responsive and mobile-friendly from the get-go.

The overall aesthetic has also changed. Most of the original Bootstrap styles have been removed, and in their place is a modern, flat approach. Even the icon library has transformed from being image-based to using glyph icons, making it much easier to add icons into your project and improving overall performance. The codebase has been rewritten, except for the class names in the base CSS – even the variable names are all different. Bootstrap previously used CamelCasing but this has been updated to use hyphens instead.

There are now four grid systems, separated by the screen width you wish to work with. These are broken down to extra-small devices (phone), small devices (tablets), medium devices (desktops) and finally large devices (large desktop screens). Other particularly noteworthy additions to version 3 include improvements to the navbar, modals, a better customiser, list groups and panels.

In this tutorial I will take a closer look at these changes and improvements, and demonstrate how the new Bootstrap can be used to build a custom sign-up landing page.

VIDEO

Steven Wu has created an exclusive screencast to accompany this tutorial. Watch along at *netm.ag/BootstrapVid-259*

DOWNLOAD BOOTSTRAP

First off, you need to head over to Bootstrap's website (*getbootstrap.com)* and download the latest version. At the time of writing, the newest release is version 3.2.0. A luxury for those who are familiar with CSS preprocessors such as Sass or Less is that you can write your CSS with either of these preprocessors. To allow beginners to follow with along this tutorial, we'll be using traditional CSS.

Once you've extracted this zip file, you will discover that it only contains the CSS, Fonts and JavaScript we require to adopt this framework. If you open up the `index.html` in the project files, you will see the `<head>` has all the basic CSS framework and external links which we need to get started with this project. You will also notice that just above the closing `</body>` tag is the minified latest version of jQuery plugin.

NAVIGATION

Bootstrap's navbar component is responsive, and has been designed to be mobile-first. This navigation will be automatically collapsed and is toggleable on mobile devices.

This responsive framework has been rebuilt from the ground up to be mobile-first

Simply set up a `div` with the appropriate default navbar classes to ensure our navigation stays static at the top of our page. You have the options to stick the navigation to the top or even bottom.

```
<div class="navbar navbar-default navbar-static-top"
role="navigation">
</div>
```

Within this we'll need a `div` class name of `container`. The container `div` is used to set a fixed width and the correct padding for our navigation. Inside our container we can add in our navigational structure.

```
    <div class="navbar-header">
      <button type="button" class="navbar-toggle" data-
toggle="collapse" data-target=".navbar-collapse">
        <span class="sr-only">Toggle navigation</span>
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
      </button>
```

▶

FOCUS ON

WORKING THE CAROUSEL

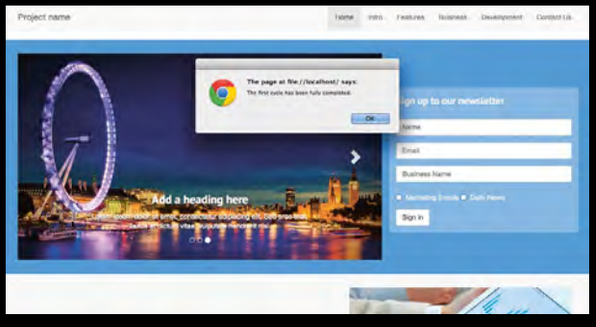
To initiate our carousel, we've used the data attribute `data-ride="carousel"` to mark the starting animation on page load. If data attributes are not your cup of tea, you can use JavaScript to manually initiate the carousel. Wrap the JavaScript in a `script` tag. The carousel plugin is identified by our unique ID carousel-slider, using the `carousel()` method, which will execute once the HTML document is ready.

```
<script>
 $(document).ready(function(){
   $('#carousel-slider').carousel({
            interval: 3000,
            wrap: true,
            pause: false
   })
 });
</script>
```

There are a few options that you may parse to the `carousel()` method. `Interval`, set in milliseconds, is the amount of time in which to cycle to the next slider. `Wrap` determines whether the carousel rotates continuously or if there's a hard stop. `Pause` will pause the cycling on mouse-over and resume on mouse-off.

As previously mentioned, you can set these using data attributes, append these to the option name (such as `data-interval="3000"`, or `data-wrap="true"`). You can even include events to hook into the carousel. We can fire an alert message once the first slide transition has been completed:

```
<script>
$(document).ready(function(){
   $('#carousel-slider').on('slid.bs.carousel', function () {
      alert("The first cycle has been fully completed.");
   });
});
</script>
```



Carousel It's easy to apply options and events in the carousel component

New look Bootstrap has been rebuilt to be mobile-first – the form elements and grid system are fully responsive and function on all devices

```
      <a class="navbar-brand" href="#">Project name</a>
    </div>
    <div class="navbar-collapse collapse">
      <ul class="nav navbar-nav navbar-right"
role="tablist">
        <li class="active"><a href="#">Home</a></li>
      </ul>
    </div>
```

The `button` and icon-bar are used as the burger menu on a mobile device, which won't be seen on the desktop. The class `navbar-brand` is used to identify our website logo. Following this is a `<ul>` which will contain our actual website navigation links.

CAROUSEL AND GRID SYSTEM

Bootstrap comes with a fluid grid system, which scales up to 12 columns as the device or viewport size increases. To incorporate the grid system into our layout, add a parent `div` called `container` (this sets a fixed width) followed by a `div` called `row`. For medium-sized devices and desktops of up to 992px, we can use the class prefix `col-md-8` which will span eight columns, within which our carousel will sit.

We'll start off by creating a containing `div` with a unique ID for our rotating carousel plus the `carousel` and `slide` classes which generate the rotating effect. The `data-ride` attribute is especially important as it's used to mark an animation starting point on page load.

```
<div id="carousel-slider" class="carousel slide" data-
ride="carousel">
  <ol class="carousel-indicators">
    <li data-target="#carousel-slider" data-slide-to="0"
class="active"></li>
    <li data-target="#carousel-slider" data-slide-to="1"></
li>
    <li data-target="#carousel-slider" data-slide-
to="2"></li>
  </ol>
```

The `<ol>` we just declared are the indicators or small circles placed at the bottom of the slider to show the current active slide and number of remaining slides. Directly beneath this is the slider content.

This space is defined using the class `carousel-inner`. Within this we can have unlimited 'item' `divs` to house each element of our slider content. The first 'item' `div` must have a class of `active` defined.

Bootstrap comes with a fluid grid system, which scales up to 12 columns as device size increases

```
<div class="carousel-inner">
  <div class="item active">
    <img src="img/banner-1.jpg" alt="City">
    <div class="carousel-caption">
      <h3>...</h3>
      <p>...</p>
    </div>
  </div>
</div>
```

A carousel slider would not be complete without specific controls with a left and right arrow to manually change the slider. Each arrow is set up as an anchor tag, with a span class of `glyphicon glyphicon-chevron-left` and respectively `-right`. Since Bootstrap

uses fonts instead of images to show icons, we use glyphicon classes to present icons.

```
  <a class="left carousel-control" href="#carousel-slider" role="button" data-slide="prev">
    <span class="glyphicon glyphicon-chevron-left"></span>
  </a>
  <a class="right carousel-control" href="#carousel-slider" role="button" data-slide="next">
    <span class="glyphicon glyphicon-chevron-right"></span>
  </a>
</div>
```

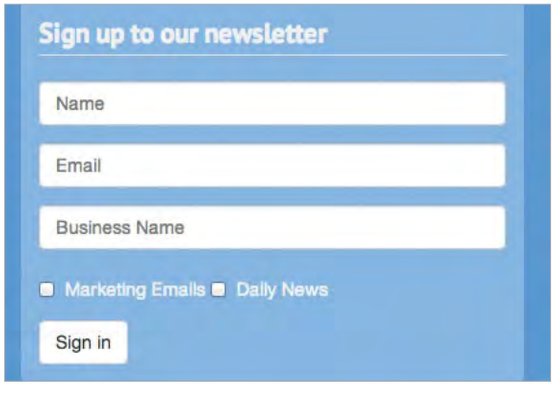
This wraps up the carousel, and without a single line of JavaScript. It all feels like magic.

NEWSLETTER SIGN-UP FORM

Next to our carousel we'll place our sign-up form. Creating responsive forms is simple with Bootstrap, as it automatically adds global styles for you. Start by wrapping your form in a grid column of `col-md-4` and a class name of `sign-up-form` for styling purposes.

```
<form class="form-horizontal" role="form">
 <fieldset>
  <legend>Sign up to our newsletter</legend>
  <input class="form-control" placeholder="Name" name="name" type="text" required>
  <input class="form-control" placeholder="Email" name="email" type="email" required>
  <input class="form-control" placeholder="Business Name" name="business" type="text" required>
```

The class `form-horizontal` is a predefined grid that aligns labels and groups of form controls together in a horizontal layout. Here, the `checkbox` class will provide the appropriate padding and margin for our checkboxes.



Forms All inputs are now displayed as a block with 100 per cent width. Users can change the size attribute to modify the padding and font-size

IN-DEPTH

USE GRUNT AND BOWER WITH BOOTSTRAP

Since the release of version 3.1 onwards, Bootstrap hasn't just made improvements to its boilerplate, but also designed a whole new development workflow. You can now use Bower and Grunt to formally carry out all the manual labour, including compiling your Less or Sass, running tests and more.

Use Bower to fetch the latest version of your frontend packages – this really helps to save a heap of time and headaches. Firstly make sure you have Git (*git-scm.com/downloads*) and Node.js (*nodejs.org*) installed, and in the command line install Bower (you may need to use sudo (*sudo.ws*) for the below command):

```
$ npm install -g bower
```

Now let's install Bootstrap:

```
$ cd /path/to/directory/
$ bower install bootstrap
```

Once installed, you will notice Bootstrap is now set up in a web app folder structure – an approach that makes it far easier to integrate this framework into existing application solutions. This feature is a great benefit of the new version of Bootstrap, as it forces developers into good habits from the offset.

We'll now install Grunt, to deal with all the nitty-gritty tasks that make frontend development rather tedious:

```
$ npm install -g grunt-cli
$ cd /path/to/bootstrap/
$ npm install
$ grunt default
```

Finally you can run any of the handy commands below:

```
$ grunt watch
```

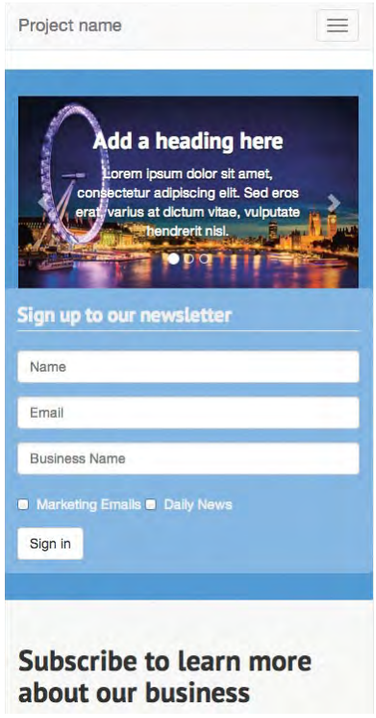
Watches your Less/Sass source files and compiles them upon save.

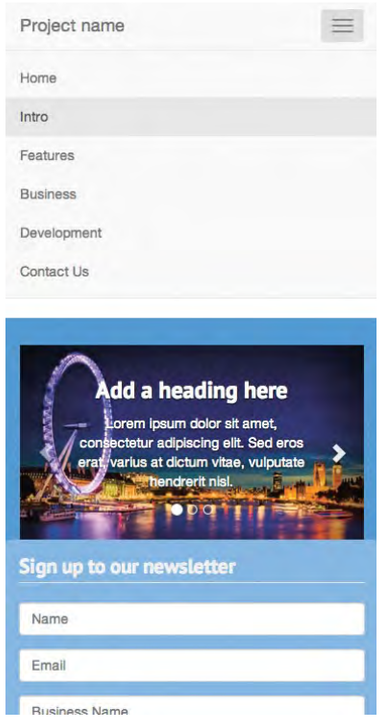
```
$ grund dist
```

Regenerates the `/dist/` directory and minifies CSS and JavaScript source files.

```
$ grunt test
```

Runs JSHunt and Qunit.





Above left The carousel component is now fully responsive, relieving the pain of having to write multiple media queries

Above right The Navbar is made up of horizontal navigations on wide viewports, which collapse on mobile devices

```
  <div class="checkbox">
    <label>
      <input type="checkbox"> Daily News
    </label>
  </div>
  <button type="submit" class="btn btn-default">Sign in</button>
 </fieldset>
</form>
```

Within the style.css some basic styling has been applied to our sign-up form.

```
.sign-up-form {
        padding: 1em;
        background:rgba(255,255,255,0.3);
        border-radius:5px
}
.sign-up-form legend {
        color: #efefef;
}
.sign-up-form input {
        margin-bottom: 1em;
}
```

RESOURCE

DOCUMENTS

To view the full list of components, CSS classes and JavaScript components, check out Bootstrap's documentation at *getbootstrap.com/getting-started*

CONTACT US FORM

Now let's set up a contact form. Start by creating a `<section>`. Within this we will again need our container `div` followed by a row `div`. Below this, we want to add a heading.

Using a `span` tag we can now incorporate a `comment-icon` using the glyphicon, simply by referencing the class name.

```
<h2><span class="glyphicon glyphicon-comment"></span> Get in touch with us</h2>
```

Following our heading, open a `form` tag and set up a grid column of `col-md-6`. To create optimum spacing between form controls we can use a predefined class called `form-group`.

```
<div class="form-group">
  <label for="contact-name">Name</label>
  <input type="text" class="form-control" id="contact-name" placeholder="Enter Name" required>
</div>
<div class="form-group">
  <label for="contact-company">Company</label>
  <input type="text" class="form-control" id="contact-company" placeholder="Enter your company name">
</div>
```

Repeat this process again so that we'll have our form controls sitting side by side.

Beneath this we can add in a `<textarea>` wrapped in a full-width row of `col-md-12`, followed by a submit button.

By utilising Bootstrap's framework we can build a production-ready site for any device

```
<label for="contact-message">Message</label>
<textarea class="form-control" rows="3" id="contact-message"></textarea>
<button type="submit" class="btn btn-default">Submit</button>
```

STYLING

Open up the style.css and you will find styles applied to the contact form, with a photographic background image. Further down this stylesheet is a media query that ensures our navigation has some top padding, but only for larger screen sizes, and for our sign-up form. You can finish off this landing page by adding in your own footer.

There you have it: a simple sign-up landing page. With minimal work, by utilising Bootstrap's responsive framework, we can quickly build a production-ready website for any device.

CSS

VERTICAL MEDIA QUERIES

Antoine Butler explores the possibilities of height-based media queries

There's this not-so-new thing called a media query, and we've all been under-using it – perhaps because too few of us know exactly what it is. A media query is a logical expression, that when `true`, enables you to tailor a layout based on a number of conditions against an optional media type, such as a `screen`. For devs, a typical use looks like this:

```
@media screen and (max-width: 40em) { ... }
```

That translates to: `if on a screen with a maximum browser/screen width of 40 ems, do this`.

Other conditions we can test against include `color`, `orientation` and `resolution`. The most consistently supported is `size`. Despite this, almost all media queries target only one half of the size spectrum.

Consider:

```
@media  screen and (max-height: 40em) { ... }
```

A fixed nav at the top of your page on a mobile phone in landscape leaves little room for content. A stacked fixed vertical nav that needs 700 pixels of vertical space no longer fits on most mobile devices. A large portrait image may not fit in the average landscape viewport of a tablet. Vertical media queries can solve all of these problems and then some.

AN ADAPTIVE SOLUTION

The height of layouts aren't easily made squishy, making height-based media queries an adaptive solution. When using them, avoid setting `min-height` and `max-height` queries based on device size. Users can browse websites from within many applications, and these variances will skew your available vertical space. Instead of letting the device dictate your breakpoints, let the content do that for you.

Wikipedia.com, for example, has subtle typographic changes on its mobile site. These changes can be easily implemented with vertical media queries. Rather than using device detection or some other method, I'll pick 650 pixels as our breakpoint for mobile adjustments.

```
@media screen and (max-height: 40em) {
figure { width: 100%; }
article { font-family: serif; }
article section { display: none; }
}
```

Here, I tell all `figure` elements to span their parent's width, set the `article` font to `serif` and hide all `sections`. With that, I've converted the layout to an accordion, as seen on *m.wikipedia.org*.

Other common adjustments you can make to most sites using vertical media queries include: tightening up `line-height`, decreasing header `font-size`, decreasing the height of sticky navigation or header bars, limiting white space and adding or removing `columns`.

SHIFTING FOCUS

Vertical media queries can also help support responsive images. While mobile devices have gotten smaller, people are also increasingly accessing the web on larger-screened devices and monitors. By giving more focused attention to the height of our layouts, we can shift content focus and improve the user's scrolling experience. Vertical media queries let us do that – and we can really think beyond the fold.

PROFILE

Antoine (*aeb.sr*) works as a developer at HZDG and spends his days reimagining the responsive web. When he's not crafting interactive experiences, he tries to promote diversity in the tech community

ABOUT THE AUTHOR

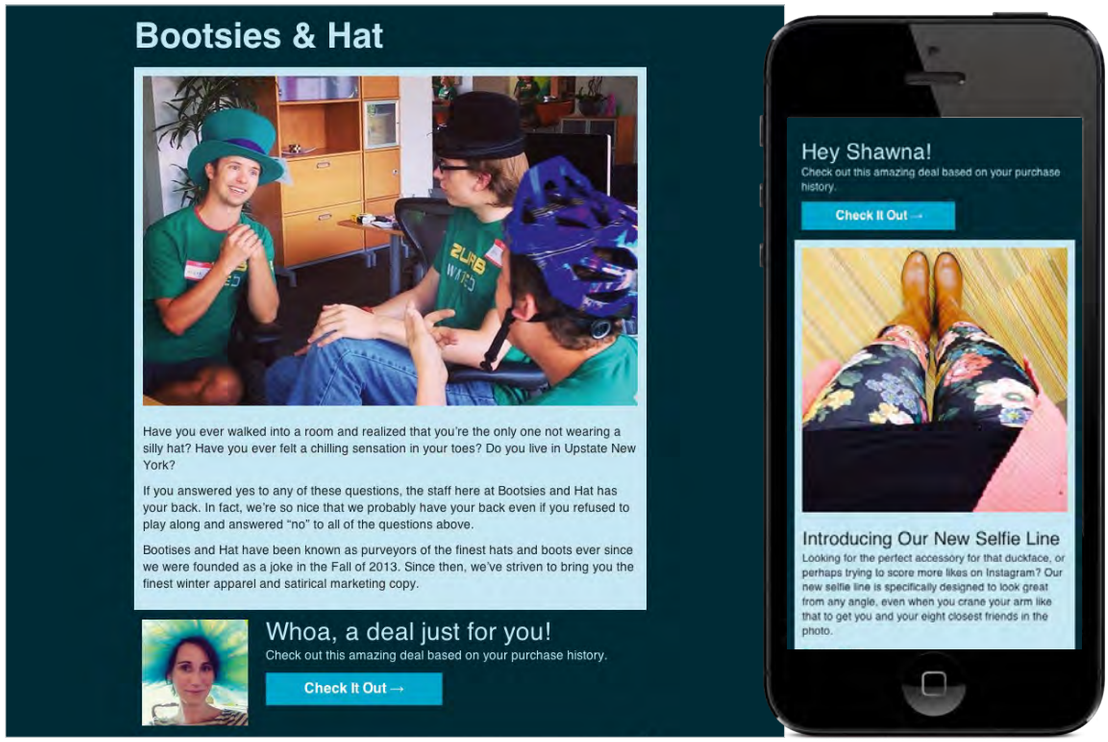
ERIC MORRIS

w: *zurb.com*

t: @thedeerchild

areas of expertise:
Responsive email frameworks

q: what's your favourite smell?
a: Mixed appleberry



RWD

RESPONSIVE HTML EMAILS

Eric Morris demonstrates how to make your HTML emails look great on any device or client, from the newest iPhone to Microsoft Outlook

As social media platforms and communication channels continue to roll out, you may start to wonder if HTML email still has a place in this world. But, marketing data firm Custoria reports that, while customer acquisition rates via Facebook have remained relatively stable, acquisition rates via email have quadrupled over the past four years (*netm.ag/email-261*). The trick to successful email, it seems, is to make newsletters look good.

With research showing that 47 per cent of emails are opened on mobile devices (*zurb.com/quips/1226*), there's a lot riding on your email to make a good impression on mobile. However, with existing solutions, we found ourselves straining to modify purpose-built templates, or reinventing responsive techniques specifically for this medium. On top of of that, any solution we developed had to support Microsoft Outlook, a client infamous for seemingly disregarding them at will.

We realised that templates weren't the answer, so we decided to build Ink (*zurb.com/ink*), a responsive email framework. Similar to Foundation (*foundation.zurb.com*), our responsive web framework, Ink gives us a solid, tested base upon which we can build responsive email, in addition to providing built-in UI elements for rapid prototyping.

We start with a visual design whenever we develop a new Ink-based email. Often this is just a simple Sharpie or whiteboard sketch, but it's important to have an idea of what you're building before you start.

For this tutorial, we're going to create a marketing email for a fictitious ecommerce startup, Bootsies & Hat (B&H). The marketing email should have: the brand's colours and logo, a few social media links for good measure and physical address and 'unsubscribe' link to comply with the CAN-SPAM act.

We want to organise the marketing blast into three content sections. The email will start with

RESOURCE

DEMO

See a demo of the Bootsies and Hat email by visiting: *netm.ag/ink-252*

an announcement and a juicy hero image to draw the reader in. The second section will contain a personalised deal. The third section will feature the pictures and descriptions of our newest products.

DIVIDE AND CONQUER

Next, we break up the design into individual code components. Ink has three main structures for that:

- **Container:** holds the email in a 580px-wide content area and centres it in the body.
- **Row:** divides the content vertically and into content sections.
- **Column:** divides up rows horizontally. On the desktop, each row is divided into 12 columns, with a section spanning n columns. On mobile devices, column sections each become their own row and expand to `100%` width.

We often print the visual design and draw grid sections with Sharpies. An annotated structure map to follow as we code is surprisingly useful.

Ink gives us a solid, tested base upon which we can build a responsive email

SETTING UP INK

Now that we've got an idea of what we're going to code, it's time to break out Ink. Ink has pretty frequent releases, so we advise you to grab the most recent version (*zurb.com/ink/download.php*) whenever you start a new project. After downloading Ink, pop open the ZIP file and copy the contents of `boilerplate.html` into your text editor.

The boilerplate file already links to the Ink CSS. It also provides a basic foundation for your message, such as meta tags and some wrapper tables to help make your email responsive.

First we set the background colour for the email. Since some mail clients strip away the body tag, you should set this on the `.body` `<table>` as well as on the actual body:

```
body, table.body {
 background: #002b36;
}
```

We start the structure with a `<table>` container with a class of `container` to hold the header. For an example (see the code snippet overleaf):

▶

RESOURCES

TOOLS WE USE

We use a number of software tools to help ensure that we're productive and thorough, as well as to catch issues with our code before they become problems. Here are a few we recommend:

Sublime Text 2
When working with the complex code that emails require, simple features like syntax highlighting, code folding and tag completion can go a long way towards ensuring valid markup.

Emmet.io
Emmet is a plugin for text editors that lets you write large amounts of markup using small, CSS selector-inspired snippets. Since Ink has lots of nested tables and very dense syntax, Emmet lets us write large blocks of code quickly and without errors. For example:

```
#page>div.logo+ul#navigation>li*5>a{Item $}
```

... becomes

```
<div id="page">
 <div class="logo"></div>
 <ul id="navigation">
  <li><a href="">Item 1</a></li>
  <li><a href="">Item 2</a></li>
  <li><a href="">Item 3</a></li>
  <li><a href="">Item 4</a></li>
  <li><a href="">Item 5</a></li>
 </ul>
</div>
```

Litmus
Testing is hugely important, and Litmus lets us test on all our target platforms (plus a number of others) all at the same time, without needing a huge library of hardware devices.

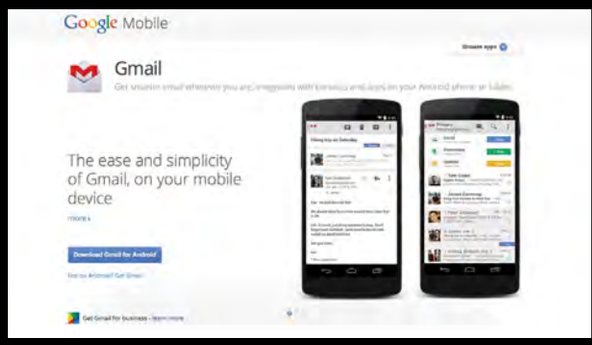
Google Chrome
While doing device tests and Litmus tests is helpful for getting full coverage, the process can be slow. It's important to test along the way so that you can spot layout problems early.

Ink Inliner
Lots of clients strip out style tags, so it's important to inline all your styles before sending. We rolled out our own inliner so that we could have more control over the output.

Campaign Monitor
Delivering email reliably and staying off of spam blacklists is surprisingly complex. Campaign Monitor has our back, so we can just worry about creating great content.

FOCUS ON

THE GMAIL APP



Ink relies on media queries, which aren't supported by the Gmail mobile apps for iOS and Android. This isn't a problem for most email lists, since Gmail only had about a three per cent market share in November 2013. Depending on your readers, this can be an issue. Ink includes an optional feature called the block-grid, which works by forcing a number of items of even width into a vertical stack as the viewport window becomes too small for blocks to fit side-by-side. While you can't create overly complex layouts using the block-grid, it does allow for some multi-column content that's responsive in Gmail. To create a two-column layout:

```
<table class="container">
 <tr>
  <td>
   <table class="block-grid two-up">
    <tr>
     <td>
      Main Content
     </td><td>
      Right Sidebar
     </td>
    </tr>
   </table>
  </td>
 </tr>
</table>
```

While the columns will wrap if the viewport is too small for them to both fit, they won't expand to the full width of the viewport like the standard Ink grid. To add this functionality on clients that do support media queries, add the progressive enhancement:

```
@media only screen and (max-width: 600px) {
 table[class="container"] .block-grid td {
  width: 100% !important;
 }
}
```

▶

```
<table class="container newsletter">
 <tr>
  <td>
```

Inside this container we make a `<table>` with a class of `row` and a `<td>` with both `wrapper` and `last` classes. There's typically a `<td>` for every column section in the row, with the last class applied to the last `<td>` in the row. Since there is only one column section (12 columns wide), there's only one wrapper, which must receive the last class:

```
<table class="row">
 <tr>
  <td class="wrapper last">
```

Inside the wrapper `<td>`, we create a column with a new `<table>`, this time with the classes `twelve` and `columns`. However, in addition to the one content `<td>`, we need a second, empty `<td>` with a class of `expander`. The expander `<td>` forces the column section to become full-width on small screens:

```
<table class="twelve columns">
 <tr>
  <td>
   <h1>Bootsies & Hat</h1>
  </td><td class="expander"></td>
```

To fill in the content in the newsletter section, put an `<img>` tag inside the regular `<td>` of the column section in the first row and some text separated by `<br>` or `<p>` tags in the second. With the `<img>`, be sure to specify an exact height and width using HTML attributes. Also provide a valid `alt` attribute:

```
<img src="images/hero.jpg" width="560" height="373"
alt="Amazing deals on bootsies and hats for the whole
family!">
```

All the `height`, `width` and `alt` attributes are necessary because some clients block images by default or

Products The new products portion of the newsletter as seen on a desktop client. The images collapse on top of the text when viewed on a mobile device

else refuse to resize them based on CSS. Setting containers apart with `background-color` is simple: Add one class and a single `background-color` CSS rule. Appropriate padding is trickier, however. Ink has a built-in helper class, `text-pad`, that can be applied to the content `<td>`s of the columns to take care of this.

NEW PRODUCTS SECTION

Like the newsletter section, we create a container `<table>` and three row `<table>`s for the 'new products' section. The table's rows each have two `wrapper` `<td>`s. Only the second has a class of `last`:

```
<table class="container products">
 <tr>
  <td>
   <table class="row">
    <tr>
     <td class="wrapper">
      <!-- Picture column goes here -->
     </td><td class="wrapper last">
      <!-- Text column goes here -->
```

Insert a new `columns` table with the class `four` into the first wrapper. Then place the product image inside the content `<td>`. Inside the second wrapper, insert a `columns` table of class `eight` and fill the content `<td>` with description text, and perhaps a link through to the site. Instead of using the `text-pad` class this time, we'll use `left-text-pad` on the image `<td>`s and `right-text-pad` on the text `<td>`s. The directional text-padding classes pad only the outside edge on large screens, but pad both sides when the columns become full-width on mobile devices. The breakout section between 'newsletter' and 'new product' sections starts with the same `container > row > wrapper > columns` structure. This section, however, will look like an individual row in the product section, except with three and nine column split.

Ink's visibility classes allow you to selectively show or hide elements on different platforms. In this case, we only want to show the image (our imaginary customer's profile picture) on desktop, and we want to swap in a headline with the customer's name on mobile devices. To hide the image:

```
<div class="hide-for-small">
 <img src="images/you.jpg" width="120" height="120"
alt="Our loyal customer.">
</div>
```

To swap out the headers, put two `<h4>` tags in the content `<td>`, one with the `hide-for-small` class and one with the `show-for-small` class:

Single column The entire layout of the newsletter collapses down into a single column on a mobile device

```
<h4 class="hide-for-small">Whoa, it's a deal just for you!</
h4>
<h4 class="show-for-small"">Hey Eric!</h4>
```

The footer row's markup requires a few modifications to fill the full browser width. First, set up the section like normal, with the container outside of the row. Next switch the `row` and `container` classes, leaving the wrapper `<td>` on the inner table. Next, on the `<td>` for the outer table, add a class of `center` and an `align center` attribute. Finally, add a `center` tag between the two tables.

For example:

```
<table class="row">
 <tr>
  <td class="center">
   <center>
    <table class="container">
     <tr>
      <td class="wrapper">
```

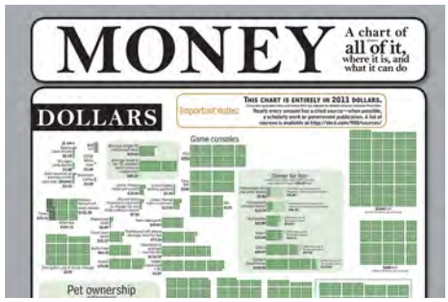
Now the row extends across the entire width of the body, which means that you can style it by adding a background colour. The content is still restricted to the `580px` container. After you've inlined your code, it's important to test it thoroughly.

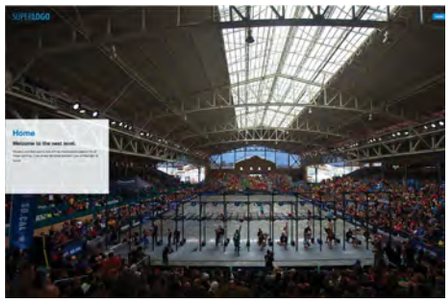
You now have the tools to create responsive emails that'll look great almost anywhere.

VIDEO

Watch an exclusive screencast of this tutorial created by the author: *netm.ag/tut2-252*

JAVASCRIPT





ABOUT THE AUTHOR

DEN ODELL

w: *akqa.com*

t: @denodell

areas of expertise: JavaScript, CSS3, semantic markup, code management

q: what's the coolest thing you do offline?
a: I DJ in clubs around Europe. I once played the warm-up set for Soulwax and 2ManyDJs in front of 20,000 festival goers

*JAVASCRIPT

SEVEN STEPS TO BETTER JAVASCRIPT

Den Odell presents his seven-step plan for writing flawless code, and rounds up the most useful tools for streamlining the process

With browser performance improving, along with the steady adoption of new HTML5 APIs, the volume of JavaScript on the web is growing. Yet a single poorly written line of code has the potential to break an entire website, frustrating users and driving away potential customers.

Developers must use all the tools and techniques at their disposal to improve the quality of their code to be confident that it can be trusted to execute predictably every time. This is a topic close to my heart and I've been working over many years to find a set of steps to follow during development to ensure only the highest-quality code gets released.

Follow these seven steps to dramatically improve the quality of your JavaScript projects. With this workflow, fewer errors will occur and any that do will be handled gracefully, leaving users to browse without frustration.

1 CODE

Start by invoking ECMAScript 5's strict mode (*netm.ag/strictmode-249*) in your functions with a `"use strict"` statement, and use the module design pattern (*netm.ag/modulepattern-249*), minimising the use of global variables by sandboxing separate code modules within self-executing function closures, passing in any external dependencies to keep modules clear and concise. Only use established and well-tested third-party libraries and frameworks, and keep your functions small, separating any business logic or data from your DOM manipulation and other view-layer code.

Larger projects with multiple developers should follow an established set of coding guidelines, such as Google's JavaScript Style Guide (*netm.ag/jsguide-249*), and will need stronger code management rules, including stricter dependency management using the Asynchronous Module Definition (AMD) through a library such as RequireJS (*requirejs.org*), package management using Bower (*github.com/bower/bower*) or Jam (*jamjs.org*) to reference specific versions of your dependency files, and the use of structural design patterns, such as the Observer pattern (*netm.ag/observer-249*), to facilitate loosely-coupled communication between your different code modules. It's also a wise idea to use a code storage repository system such as Git or Subversion via services such as GitHub (*github.com*) or Beanstalk (*beanstalkapp.com*) to keep your code backed up in the cloud, provide the ability to revert to previous versions, and, for more advanced projects, to create branches of code for implementing different features before merging them back together once complete.

SLIDEDECK

MORE DEPTH

The slides for Den Odell's conference presentation 'High-quality JavaScript Code' expand on the ideas set out in this article: *netm.ag/quality-249*

2 DOCUMENT

Use a structured block comment format such as YUIDoc (*yui.github.io/yuidoc*) or JSDoc (*usejsdoc.org*) to document functions so any developer can understand its purpose without needing to study its code, reducing misunderstanding. Use Markdown syntax (*netm.ag/markdown-249*) for richer, long-form comments and descriptions. Use associated command-line tools to auto-generate a documentation website based on these structured comments that keeps up to date with any changes made in your code.

3 ANALYSE

Run a static code-analysis tool, such as JSHint (*jshint.com*) or JSLint (*jslint.com*) against your code regularly. These check for known coding pitfalls and potential errors, such as forgetting to use strict mode or referencing undeclared variables, and spot missing braces or semicolons. Correct any issues the tool flags up to improve your code quality. Try setting default options for your project team to enforce coding standards, such as the number of spaces by which to indent each line, where to place curly braces, and the use of single or double quotes throughout your code files.

With these seven steps, fewer errors will occur and those that do will be handled gracefully

4 TEST

A unit test is a small standalone function that executes one of the functions from your main codebase with specific inputs to confirm it outputs an expected value. To improve your confidence that code will behave as expected, write unit tests using a framework such as Jasmine (*pivotal.github.io/jasmine*) or QUnit (*qunitjs.com*) for each of your functions, using both expected and unexpected input parameters. And don't forget those edge cases!

Run these tests in multiple browsers across multiple operating systems by taking advantage of services such as BrowserStack (*browserstack.com*) or Sauce Labs (*saucelabs.com*) which allow you to spin up virtual machines in the cloud on demand for testing. Both services provide an API allowing your unit tests to be run automatically across a number of browsers simultaneously, with the results fed back to you as they complete. As a bonus, if your code is stored in GitHub, you can take advantage of BrowserSwarm (*browserswarm.com*), a tool that automatically runs your unit tests when you commit your code.

5 MEASURE

Code-coverage tools such as Istanbul (*gotwarlost.github.io/istanbul*) measure which lines of code are executed when your unit tests run against your functions, reporting this as a percentage of the total number of lines of code. Run a code-coverage tool against your unit tests, and add extra tests to increase your coverage score to 100 per cent, giving you greater confidence in your code.

Function complexity can be measured using Halstead complexity measures (*netm.ag/halstead-249*): equations devised by computer scientist Maurice Halstead in the 1970s that quantify the complexity of a function according to the number of loops, branches and function calls it contains. As this complexity score decreases, the easier it becomes to understand and maintain the function, reducing the likelihood of errors. The command-line tool Plato (*github.com/es-analysis/plato*) measures and generates visualisations of JavaScript code complexity, helping identify functions that could be improved, while storing previous results, allowing quality progress to be tracked over time.

6 AUTOMATE

Use a task runner such as Grunt (*gruntjs.com*) to automate the processes of documentation, analysis, testing, coverage and complexity-report generation, saving yourself time and effort, and increasing the chance of addressing any quality issues that arise. Most of the tools and testing frameworks highlighted in this article have associated Grunt tasks available to help you improve your workflow and your code quality without you having to lift a finger.

7 HANDLE EXCEPTIONS

Invariably, at some point, your code will throw an error when it's run. Use `try...catch` statements (*netm.ag/trycatch-249*) to handle runtime errors gracefully and limit impact on your site's behaviour. Use a web service to log runtime errors thrown. Use this information to add new unit tests so as to improve your code and eradicate these errors one by one.

STEPS TO SUCCESS

These seven steps have helped me produce some of the code I'm most proud of in my career so far. They're a great foundation for the future, too. Commit to using these steps in your own projects to produce high-quality JavaScript code, and let's work together to improve the web, step by step.

RESOURCE

USEFUL BOOK

Published by Apress Media, Den Odell's 2009 book, *Pro JavaScript RIA Techniques*, outlines best practices for writing code: *netm.ag/projavascript-249*

VIDEO

Watch an exclusive screencast of this tutorial created by Tuts+ Premium: *netm.ag/tut2-249*

Download the files here!

All the files you need for this tutorial can be found at *netm.ag/d3-257*

ABOUT THE AUTHOR

TIM RUFFLES

w: *truffles.me.uk*

t: @timruffles

areas of expertise: JavaScript, training

q: what makes you squeamish?

a: Eyeballs. Biology class, I say no more

D3

CREATE CLEAR DATA VISUALISATIONS WITH D3

Tim Ruffles shows you how you can use D3 to wrangle complex sets of data into beautiful, clean visualisations

D3 is a toolkit for building visualisations from scratch. It's a thin wrapper around the DOM, so with HTML, CSS and JavaScript you're already halfway there. It's tremendously powerful, and more than anything it's tremendous fun. Never again will you disappoint your designer by stuffing their expertly tuned Dribbble-fodder into a boring chart.

D3 is best taught by example, so we'll recreate xkcd's 'Money' visualisation (*xkcd.com/980/huge*). We'll take take data on the income of people, firms and countries, and visualise them in broad groups of magnitude. D3 enables us to include very different incomes in the same visualisation.

D3 enables us to include very different income values in the same visualisation

GIVING UGLY CODE A MAKEOVER

Our task is split between data and presentation. A common source of ugly D3 code is not getting your data in the right shape. If you try to visualise your data before you've wrangled it into the right shape you'll end up with a mess.

We'll wrap up the data part in what D3 calls a 'layout'. Layouts sound visual, but are actually the data-wrangling side of a given visualisation. They're standard JavaScript functions called with input data and returning new data formatted for display. A histogram layout might be passed an array of 1,500 objects, and return 10 arrays, each bucketing 150 objects. Displaying its output is entirely up to you. This splits the reusable data-processing algorithm from a specific presentation.

We need to break the incomes down into magnitude groups, and then visualise them as blocks. Our income data is in the format `{ value: 2500000, title: "Jimmy Carr" }` our target format is below. It reflects the visualisation: the previous group is included in the next for comparison, and we've calculated the number of unit blocks to display.

```
[
 {
  key: "1000000",
  total: 2124000000000,
  values: [
   { title: "Jimmy Carr",
     value: 2500000,
     units: 25 }, /* ... more salaries */
   { fromLast: true,
     value:  105000,
     units: 1 }, /* total of previous group */
  ]
 },  /* ... more groups of salaries */
];
```

IMPLEMENTATION

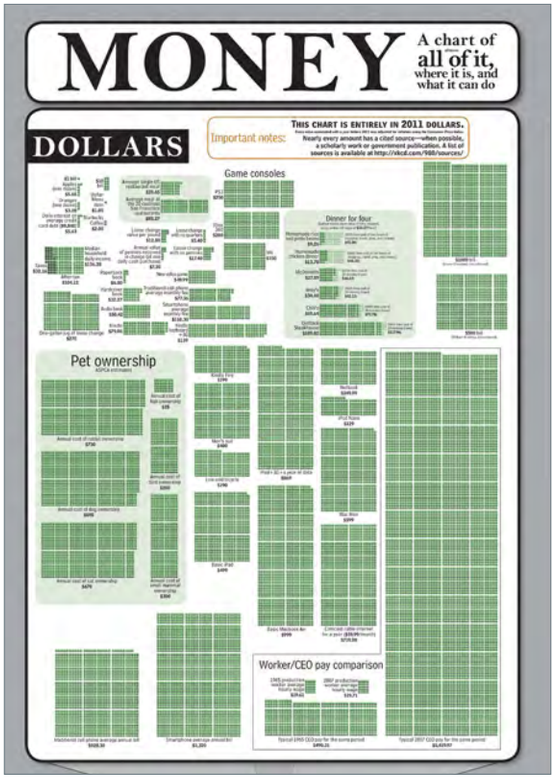
Let's implement! Our data is hierarchical so `d3.nest()` is the right tool: it takes an array and groups it by a key function.

```
function blockLayout() {

 var grouper = Math.log;
```

RESOURCE

EXAMPLES

D3's original author, Mark Bostock, has a huge selection of visualisations on his site, complete with full source code: *bl.ocks.org/mbostock*



xkcd's Money visualisation This is the inspiration behind the visualisation we'll be creating. See the full version at *xkcd.com/980/huge*

```
layout.group = function(x) { grouper = x; return this; };
return layout;

function layout(data) {

  data.sort(function(a,b) { return b.value – a.value; });

  var nested = d3.nest()
    .key(function(d) { return grouper(d.value) })
    .entries(data)
    .map(function(group) {
      group.values.forEach(function(v) {
        v.units = getUnits(v.value,group);
      });
      group.total = group.values.reduce(sumValues,0);
      return group;
    });

  d3.pairs(nested).forEach(function(pair) {
    var group = pair[0], total = pair[1].total;
    group.values.push({
      value: total,
      group: group,
      fromLast: true,
      units: getUnits(total,group)
    });
  });
```

▶

*IN-DEPTH

D3 IN A NUTSHELL

If you have used jQuery, D3 is comfortingly familiar. We work on selections of elements via a chaining API.

```
d3.select(vizEl).selectAll(".group")
 .property("draggable",true)
 .style("background","blue")
```

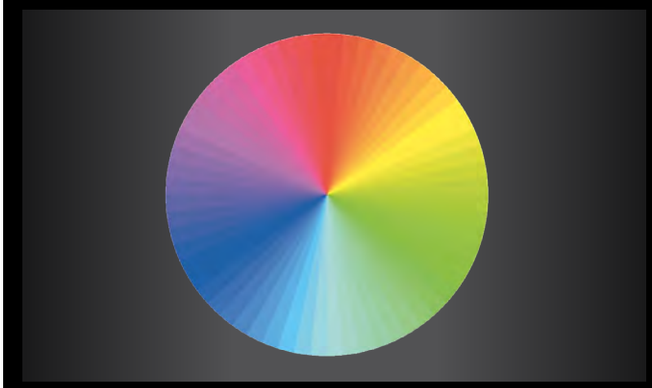
This code selects all `group` elements within `vizEl`, makes them `draggable` and sets their background colour to blue.

But where jQuery's API loves the DOM, D3's loves data. We call `.data()` on a selection to provide it with a data set. Callbacks passed to selection methods are called for every element with its corresponding datum, and the datum's position in the data set:

```
d3.select(vizEl).selectAll(".group")
 .data([
   { title: "one", value: 10 },
   { title: "two", value: 20 }
 ])
 .style("background","blue")
 .style("width",function(data,index) {
   return data.value + "px";
 })
 .text(function(data,index) {
   return data.title;
 })
```

Here the `width` and text content of each element is derived from its corresponding datum. We'll normally want to keep the number of elements in sync with the number of datums. D3's three contexts – `enter()`, `update` and `exit()` – allow us to control this. `enter()` paired with `append()` adds elements for datums without one, `update` handles cases where we have an element datum pair, and `exit()` affects elements left alone after we've paired the data.

Starting simple It's not much, but this is the first step to D3



FOCUS ON

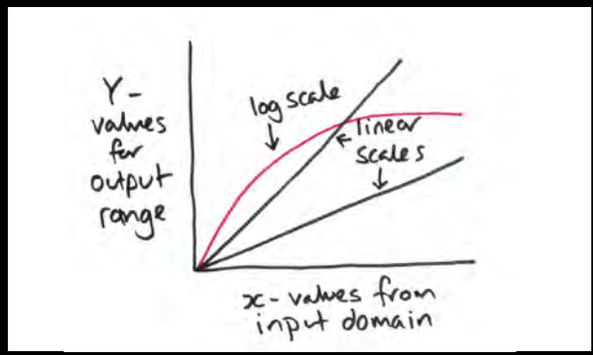
SCALES AND COLOURS

Dataviz coding will always involve transforming values in our data into attributes for display. D3's scales make this quicker and cleaner. Scales return a function that transforms values from our data – the *domain* – into values for our visualisation – the *range*. The simplest to understand is `d3.linear()`, which is a line on the XY axis. Linear scales are useful, but there are many other types. If you have data with a huge range you can use `d3.scale.log()` to crunch down the variance into something graphable. `d3.scale.ordinal()` breaks data into categories.

Scales are packed with tools. If you want to draw an axis with 10 evenly spaced marks, call `ticks(10)` and you'll receive an array of 10 values along your range. To handle input above or below the intended domain, `clamp()` the scale.

One final feature of scales is interpolation. Frequently we'll be transforming a number into a visual unit, like pixels or colours. To output pixels, we simply set the range to `["10px","1000px"]` and it'll output pixel values ready to use in `style()` callbacks. It can even handle colour transitions between RGB, HSL or named CSS colours.

As a demonstration, I've created a colour wheel using two scales sharing the same domain – a hue scale and an angle scale. See it live at *netm.ag/wheel-257*, and the code is at *netm.ag/wheelcode-257*.



Converting values Scales take values from our input domain and transform them into values in our output range for presentation

```
  return nested;
}

function getUnits(value,group) {
  return Math.ceil(value/parseInt(group.key));
}
function sumValues(a,s) { return a + s.value; }
}
```

As layouts are just JavaScript functions, and functions are objects, we expose our configuration function as the layout's `group` property. To configure our layout, we call `.group(keyFunction)` on the layout in

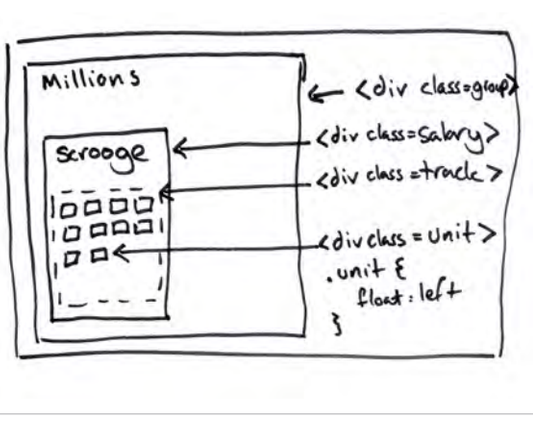
SVG is great when you're after the kind of shapes that HTML doesn't support

a chaining style. To use it, we call it on our original dataset to create `salaryGroups` ready for binding.

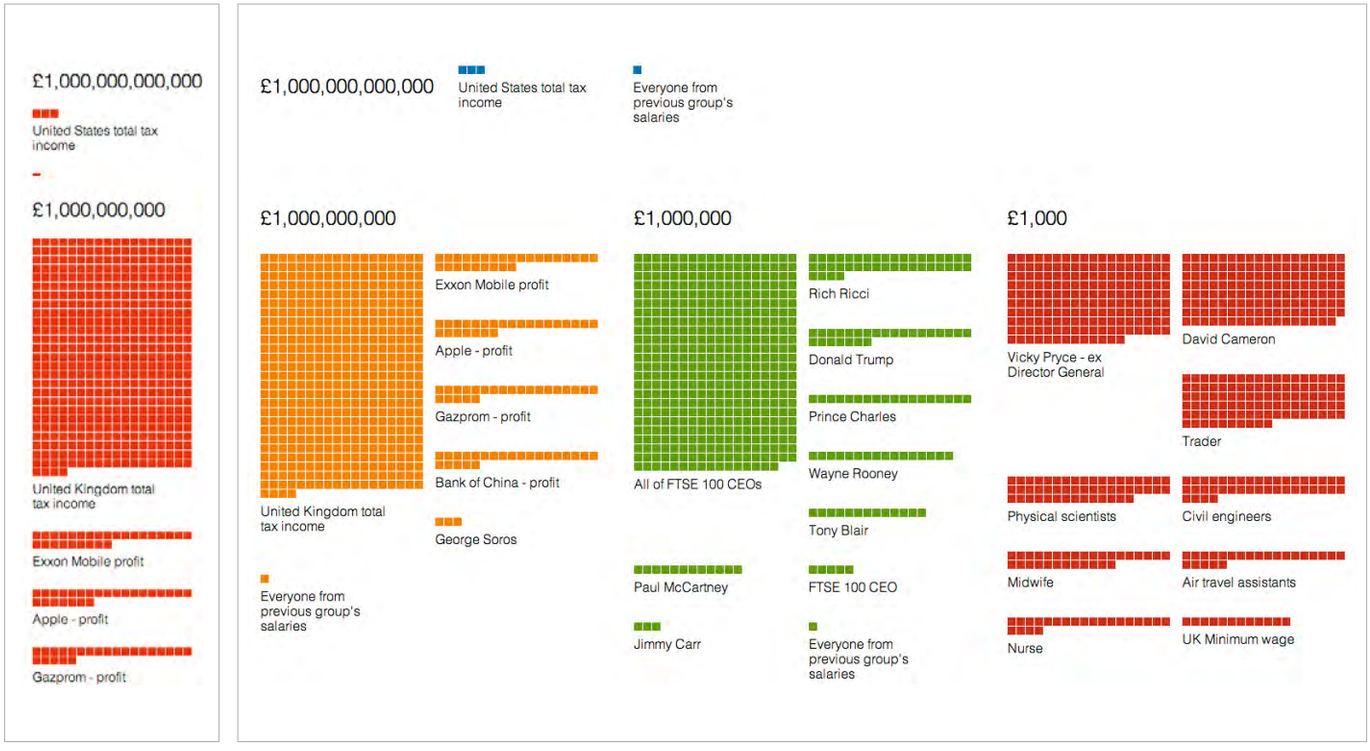
```
var layout = blockLayout()
 .group(function(value) {
   if(value < 1e6)  return 1000;
   if(value < 1e9)  return 1e6;
   if(value < 1e12) return 1e9;
   return 1e12;
 });

var salaryGroups = layout(listOfSalaries);
```

VISUALISING

At this point we've not actually displayed anything on screen. This division of labour allows our layout



Rough code Sketch of the HTML/CSS implementation of the visualisation



Left The visualisation's output with basic styling

Right The complete visualisation. It's testament to D3's design that it comes in at under 150 lines of JS, HTML and CSS

to be reused with different data sets, key functions and final visualisations. Now we've cleaned up our data, we can visualise.

Here's a sketch of the HTML/CSS implementation – we append the top level groups to represent the `salaryGroups` from our layout:

```
var groups = d3.select("#viz").selectAll(".group")
 .data(salaryGroups)

var groupsEntering = groups.enter()
 .append("div").attr("class","group")
```

Next we present the individual salary elements. Our data is nested: we've appended elements for the top level group data which has the shape `{ key: "1000", values: [/* salaries */] }`. We therefore make a selection of `.salary` elements inside each group, and pass a function pulling out `.values` to `data()` to dig down the hierarchy for the salaries:

```
var salaries = groups.selectAll(".salary")
 .data(function(x) { return x.values; });

var salariesEntering = salaries.enter()
 .append("div").attr("class","salary");
```

Finally we need to consider our units. We create track elements per salary, then use our nested `data()` trick again to make many units from each salary.

```
var units = salariesEntering
 .append("div").attr("class","track")
 .selectAll(".unit").data(createBlocks)
 .enter()
 .append("div").attr("class","unit")
```

`createBlocks()` is simple: if a salary needs 10 unit blocks, return an array of length 10:

```
[//]: # article-code.js:7
function createBlocks(salary) {
  var blockCount = salary.units, blocks = [];
  while(blockCount--) blocks.push(salary);
  return blocks;
}
```

It's easier to define all static styling via CSS. We want the units to flow inside a constrained track. The rest of the styling I'll leave to you.

```
.unit {
 float: left;
 width: 10px; height: 10px;
 background: red; }
.track {
 overflow: auto;
 width: 200px; }
```

With a little more styling and titles added we're done. Now get out there and visualise!

RESOURCE

INSPIRATION

The Visual Complexity blog full of inspiring visualisations. Its author has also written *The Book of Trees* – an exploration of data visualisation: *visualcomplexity.com/vc*

Download the files here!

All the files you need for this tutorial can be found at *netm.ag/matterjs-255*

ABOUT THE AUTHOR

LIAM BRUMMITT

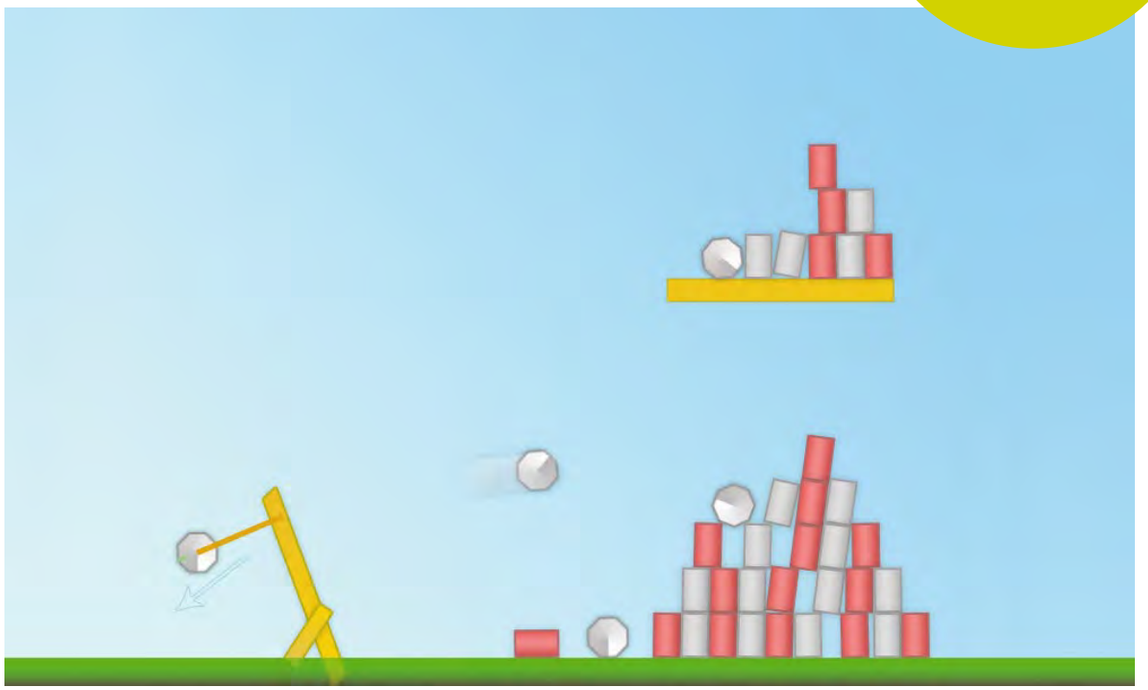
w: *brm.io*

t: @liabru

areas of expertise:
HTML5, CSS3, JavaScript, app and game development

q: When was the last time you cried?

a: The last time I heard "IE6 support required"!



HTML5

BUILD AN HTML5 GAME WITH MATTER.JS

Liam Brummitt explains how to use his JavaScript physics engine, Matter.js, to create a simple web game built on HTML5 technology

Matter.js is a flexible rigid-body 2D engine with a focus on high performance, stability, ease of use and cross-platform compatibility. Although it isn't intended to be a complete game engine – more a component of one – it includes most of the things you need to get started, including a game loop, mouse controller, collision system and a renderer.

The project is currently in alpha and comes with all the usual caveats – like not using it on critical productions – but it is still a good fit for indie-style 2D games that don't require a heavyweight engine.

"But I'm already using the WhizzBang2000 engine," I hear you say! No problem. Since Matter.js includes an event callback system and configurable controllers, it should be possible to integrate the physics engine into whatever JavaScript game engine or rendering library you wish to use.

The engine includes an example canvas-based renderer, with basic support for both primitive (vector) rendering and sprite-based (texture) rendering of the physical bodies in your game world. If you need to accomplish anything further, it's easy to use this as the basis for your own renderer.

Or if canvas is a bit too pedestrian for your tastes, and you'd prefer the bleeding-edge power of WebGL, you're in luck: swap in the included example WebGL renderer (which makes use of the superb Pixi.js library) and you're good to go. It's possible to extend this scene-graph-based renderer to use something like three.js too.

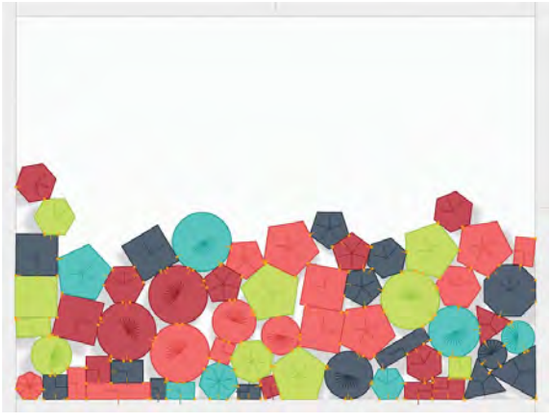
BUILDING A SIMPLE GAME

It's about time I showed you how to get up and running with Matter.js. We'll set up a simple scene:

RESOURCE

MORE INFO

You can find out more about Matter.js on the engine's project page at *brm.io/matter-js*



Scalable engine Matter.js can handle plenty of simulated rigid bodies

a pyramid of blocks, a slingshot and a boulder we'll be able to lob and smash it all down with.

If at any point you get stuck, or something doesn't make sense, you can always check the engine examples and API docs over at the project page (*brm.io/matter-js*).

First things first: grab a copy of the engine from the project page, and create a blank HTML page set

You can integrate Matter.js into any other JavaScript game engine or rendering library

up with a `script` reference to Matter.js. Open another `script` block ready to enter the code below.

When you include Matter.js, the `Matter` object becomes globally available, which is where all the Matter functions live. To keep things short, I like to start by aliasing the modules I need to use at the top of the script, like so:

```
var Engine = Matter.Engine,
    World = Matter.World,
    Bodies = Matter.Bodies,
    Constraint = Matter.Constraint,
    Composites = Matter.Composites,
    MouseConstraint = Matter.MouseConstraint,
    Events = Matter.Events;
```

Create an empty function called `init`. You will put most of the code here. It is very important to make sure that after the `init` function is declared, you bind it to the window load event – otherwise you won't see anything!

```
window.addEventListener('load', init);
```

FOCUS ON

WHY HTML5 PHYSICS GAMES?

As we've seen from the success of physics-based games on mobile app stores, physics is often a major component of modern games, with many using it as their core mechanic. Why? There just seems to be something so satisfying about building and destroying things!

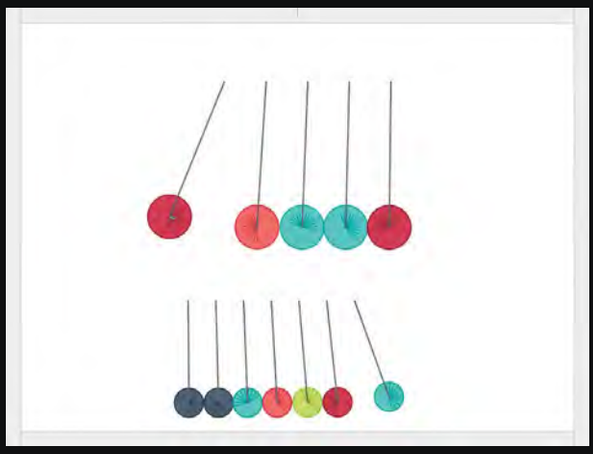
A few years ago, the idea of creating real games on web-based technology would have seemed crazy. Fortunately, web technology is rapidly becoming a viable way to get your content onto a number of platforms, without breaking (too much of) a sweat.

In recent years there have been massive advances in HTML5 technologies, from the performance of core JavaScript engines (V8, SpiderMonkey, and so on) to APIs such as canvas and WebGL.

And it's only going to keep getting better. Recently, Mozilla has been working with both Unity (*netm.ag/unitywebgl-255*) and Unreal Engine (*netm.ag/unrealwebgl-255*) to get them working at near-native speed in the browser, with the help of asm.js: a highly optimised version of JavaScript. Now that's impressive.

Matter.js may not be quite in this league – it isn't a fully featured game engine like Unity or Unreal – but it may still be a good fit for indie-style 2D games that don't require a heavyweight engine.

Online play The popular Unity game engine is adding WebGL export in version 5



RESOURCES

DEMO SCENES

You can currently see 20 different examples and demos on the Matter.js project page (*brm.io/matter-js*), including the classic Newton's cradle above, which – as well as being slightly hypnotising – demonstrates the momentum-conservation properties of the engine.

There's also a mobile-specific demo, which includes code for shifting gravity based on the device's tilt sensors. While this currently only works smoothly on higher-end devices, it shows the potential for mobile use, even at this early stage.

The demos on the project page use simple white-on-black outline graphics, but you can see more colourful versions on CodePen (*codepen.io/liabru*) or visit the full online demo (*brm.io/matter-js-demo*) and play with the renderer settings for yourself.

Explore parameters Test the effect of Matter.js settings in the full online demo

▶ First, let's create an engine, passing the element where the canvas is to be inserted:

```
var engine = Engine.create(document.body);
```

We want to control the scene with the mouse. The engine includes a helpful constraint called `MouseConstraint`, so we'll add it now:

```
var mouse = MouseConstraint.create(engine, {
    constraint: { stiffness: 1 }
});
```

Here I've passed the `engine`, which is required for the `MouseConstraint` to work. I've also passed an options object, which changes the constraint's `stiffness` to make it a little less slack than the default setting.

Let's add some bodies to the scene. First of all, we need a ground object, otherwise objects will fall out of the world!

```
var ground = Bodies.rectangle(395, 600, 815, 50, {
isStatic: true });
```

Here the `Bodies` module is a factory that can create common types of geometric object: squares, rectangles, circles, triangles and other polygons. I've passed the position and dimensions we require.

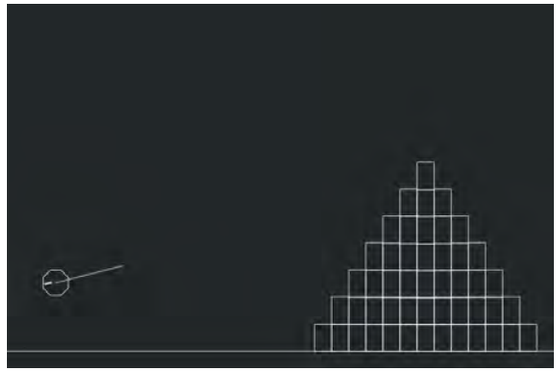
I've also passed an options object again. This time we flag the ground as `isStatic` since the ground doesn't normally move.

CREATING PHYSICS OBJECTS

Now let's create a rock. We'll be adventurous and use a hexagon, with radius 20px:

```
var rock = Bodies.polygon(170, 450, 8, 20);
```

To fling our rock, we need some elastic. We'll use a `Constraint` for this:



Simple demo The slingshot game we will be creating in this tutorial

```
    var anchor = { x: 170, y: 450 },
    elastic = Constraint.create({ pointA: anchor, bodyB: rock,
    stiffness: 0.1 });
```

As you can see, one end of the `Constraint` will be fixed to `pointA`, an anchor at a fixed position in the scene. The other end is fixed to the centre of the rock we just created. I've set the `stiffness` pretty low to make the `Constraint` act more like elastic, to give us some firepower for our slingshot.

SETTING UP A TARGET

Next we need a target to smash. The engine supports `Composite` bodies, which are collections of `Body`, `Constraint` and even other child `Composite` objects that can be manipulated as a group.

We want a pyramid of blocks, which normally takes a bit of code to set up. Since this is a common requirement, Matter.js includes a `Composites` factory that can create stacks and pyramids easily, like so:

```
    var pyramid = Composites.pyramid(450, 300, 13, 10, 0, 0,
    function(x, y, column, row) {
        return Bodies.rectangle(x, y, 25, 40);
    });
```

Matter.js supports Composite bodies: collections of Body and Constraint objects

I won't go through all the arguments here, but you can find them in the API docs via the project page. (Essentially, the pyramid factory uses a callback in which you create your bodies and the factory will stack them correctly for you.)

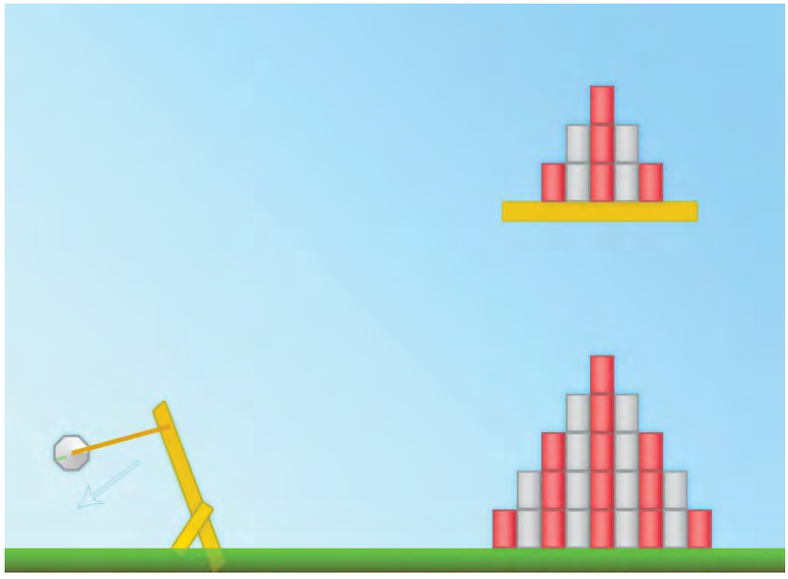
We now have all the bodies and constraints we need, so let's add them all to the world:

```
    World.add(engine.world, [mouse, ground, pyramid, rock,
    elastic]);
```

SETTING UP GAME LOGIC

The final part is to add a bit of game logic. We need the rock object to disconnect from the elastic when the player has let go of the slingshot, then reload with another rock.

For this, we'll bind an event to the engine. The `tick` event is called on every update of the engine. This normally means there are 60 ticks a second (at 60fps):



Graphical polish The Matter.js slingshot game we have created during this tutorial, after a little extra work

```
    Events.on(engine, 'tick', function(event) {
        if (engine.input.mouse.button === -1 && rock.
        position.x > 190) {
            rock = Bodies.polygon(170, 450, 7, 20);
            World.add(engine.world, rock);
            elastic.bodyB = rock;
        }
    });
```

The code above does several things. First, it checks that no mouse button is currently being pressed; and given that, it checks that the rock has moved horizontally past its initial resting position while on the elastic.

If this is true, the rock should be released. The code creates a new `rock` body and attaches it to the `elastic` constraint. This means that the previous rock is no longer attached and is released, hopefully now flying towards our pyramid of blocks. What's more, we now have a brand new rock on the slingshot that is good to go.

Now we have everything set up correctly, the last thing to do is kick off the simulation:

```
    Engine.run(engine);
```

Again, make sure you have wrapped all of the above code in a function and bound it to the window load event, as discussed at the start of the tutorial.

Hopefully when you open your page in the browser, the game should be ready to try out. If it doesn't work, check the console, then check the link to the tutorial code and compare the two.

RESOURCE

SOURCE CODE

Want to go further, or contribute to Matter.js yourself? Find the code on GitHub at: *netm.ag/github-255*

Download the files here!

Download the GitHub repository for this tutorial at *netm.ag/js-256*

ABOUT THE AUTHOR

MATT CAMPBELL

w: *github.com/codingcampbell*

t: @codingcampbell

areas of expertise: HTML5, JavaScript, LAMP

q: Which living person do you most despise?

a: Dong Nguyen, creator of my favourite love-to-hate game, Flappy Bird

JAVASCRIPT

CREATE INTERACTIVE JS VIDEO EFFECTS

Matt Campbell shows how JavaScript and a few image layers can create video-quality animations you can enhance with interactive elements

Modern web users have embraced video, but developers often find video integration problematic owing to cross-device/platform limitations (most mobile browsers do not support autoplay for inline video) and fewer opportunities to infuse playback with user interactivity. With some fairly simple methods, you can replicate video-like effects without needing to transcode large files or fight autoplay restrictions on mobile devices.

This guide will step through some simple techniques to create dynamic lighting effects by progressively adjusting the opacity of three image layers. The resulting effect will look as smooth as video – check out the demo at *growcode.github.io/tutorial-js-lighting/demos*. We'll also show some ways to enhance these animations with a few fun opportunities for user interactions.

IMAGE SET-UP

First you'll need to create three images of the same object/scene with different lighting configurations. The first image should show the initial state of the effect, the second will be the middle transition, and the third one will be the final state.

STACKING THE IMAGES

1 Set up a `<div>` container element that has `position: relative` style.

2 Add each `<img>` inside the container with `position: absolute`.

The `absolute` positioning will enable the images to stack on top of each other while the container's `relative` style prevents the images from escaping to the top of the page.

The HTML and CSS code should look something like this:

```
<!doctype html>
<html>
  <head>
  <style>
```

RESOURCE

IN PRACTICE

My colleagues at Grow are no strangers to pushing JavaScript to the limit. Check out how Box2D is used to visualise bandwidth issues on YouTube: *thisisgrow.com/work/how-video-gets-to-you*

```
    #container { position: relative; }
    #container img { position: absolute; }
  </style>
  <body>
    <div id="container">
      <img src="path/to/image1.jpg"/>
      <img src="path/to/image2.jpg"/>
      <img src="path/to/image3.jpg"/>
    </div>
    <script>
      // we will be putting more code here
    </script>
  </body>
</html>
```

TRACKING THE IMAGES IN CODE

We need a very basic structure in the code to track and fade the images. This structure can also keep track of the animation's progress. It looks like this:

```
function Shader(images) {
  this.images = images;
}
```

The constructor here just accepts a `NodeList` of images we will be tracking. Using the handy `querySelectorAll` function (*developer.mozilla.org/en-US/docs/Web/API/Document.querySelectorAll*) to get the images from the DOM, we can construct our object like so:

```
var container = document.getElementById('container');
var shader = new Shader(container.
querySelectorAll('img'));
```

OPACITY CHANGES

The lighting effect is achieved by showing the image on the bottom of the stack first, unmodified. In fact the bottom image is never faded at all – it is just what the user sees first.

As the animation progresses, we gradually fade in the images from bottom to top, so the user will see the middle image fade in first, followed by the top image a few moments later.

We can do this by tracking the progress of the animation from 0 to 1 (0% to 100%, if you like). Based ▶

Using some fairly simple methods, you are able to replicate video-like effects

FOCUS ON

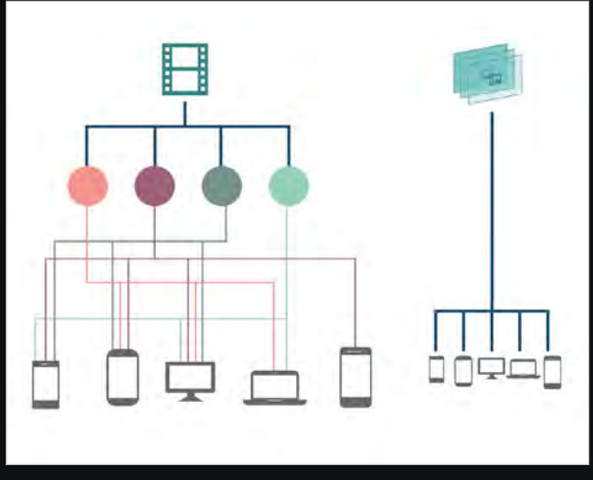
WHY NOT VIDEO?

Video offers functionality similar to the features described in this article. Specifically, videos are animations that are normalised to a 0-100% progress. Modern HTML5 browsers even expose an API to update a video's progress with JavaScript.

But the power of HTML5 video comes with its own caveats. First and foremost is the ever-changing land of codecs. Mobile devices are close to ubiquitously supporting H.264, but the desktop platforms still have configurations that don't support it. Firefox in particular relies completely on the OS to provide support for that codec. These situations call for transcoding your video to formats such as Theora or the newer WebM codec. This is quite a difference from the dependability of JPEG and PNG image support.

HTML5 video on mobile also suffers from playback limitations. At Grow just a few months ago, we had a client who wanted video-quality effects in an HTML5 ad that was targeted to tablets. But on both Android and iOS, HTML5 video (and audio) cannot autoplay at all. In fact they can't play unless the user initiates them with a touch action.

The technique described in this article was used to work around this limitation to provide an effect that animates automatically and can also react to non-touch interactions such as tilting (accelerometer input).



Cut through the noise A multi-device/platform video experience can be a complicated task. JS, HTML5 and CSS offer a standardised, direct path to success

RESOURCES

REFERENCES

A quick recap of some of the utlities employed in this tutorial, and where they can be found

QUERYSELECTORALL

`querySelectorAll` provides the ability to find elements in the DOM using CSS selectors, similar to how jQuery works.

developer.mozilla.org/en-US/docs/Web/API/Document.querySelectorAll

REQUESTANIMATIONFRAME

You can hook into the browser's rendering loop by passing a callback function to `requestAnimationFrame`. The browser will invoke the callback after each screen paint. Typically this is vsynced to the monitor and also intelligently stops updating if the tab becomes inactive.

developer.mozilla.org/en-US/docs/Web/API/window.requestAnimationFrame

HEADTRACKR

This library makes it easy to integrate head tracking into your web page. This works on any WebRTC-compliant browsers by applying computer vision techniques using a webcam as the video source.

github.com/auduno/headtrackr

H.264 CODEC SUPPORT

caniuse.com/mpeg4

WEBM CODEC SUPPORT

caniuse.com/webm

▶ on that `progress` variable we can fade the top images appropriately so that the middle image finishes its fade-in before the final image:

```
function clamp(value, min, max) {
  return Math.max(min, Math.min(max, value));
}

Shader.prototype.update = function (progress) {
  this.images[1].style.opacity = clamp(progress - 0.33, 0, 0.33) / 0.33;
  this.images[2].style.opacity = clamp(progress - 0.66, 0, 0.33) / 0.33;
};
```

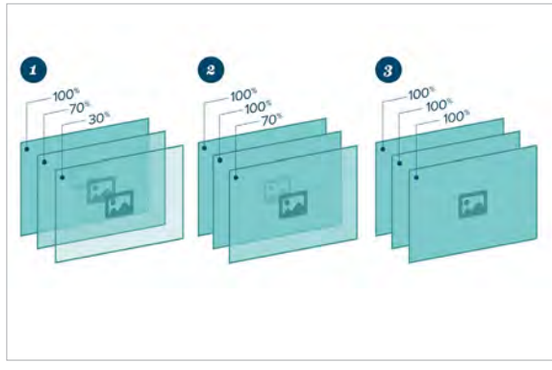
The `clamp` function here simply keeps `progress` within the bounds of `0` and `0.33` (the max it will be divided by). This is necessary because before the animation is at 66% progress, the final image would calculate `progress - 0.66` as a negative number (for instance, `-0.66`) – it's better to just keep it at `0` in that case, since that image is not supposed to be fading yet. This will delay the fading of that image until the animation progress is beyond 66% (the only time `progress - 0.66` would be positive).

In the `update` function, the first opacity line instructs the middle image to fade in between 33% and 66% of the animation. The final image starts fading at 66% and finishes at 99% (basically the end).

TWEENING

Now that our animation is able to react to a `progress` variable, it's time to animate that variable. Applying the current time to a sine wave is an easy way to do this, providing a natural bounce from 0 to 1.

```
/* Define the animation loop */
function animate() {
  /* Get sine for current time */
  var angle = Math.sin(Date.now() * 0.002);
```



Flurry of fades As our animation progresses, the second and third images move to full opacity, gradually obscuring the layers behind them

```
    /* Normalize sine to 0 through 1 */
    var progress = (1 + angle) * 0.5;

    /* Ask animation to update the fades */
    shader.update(progress);

    /* Repeat! */
    requestAnimationFrame(animate);
}

/* Begin animation */
animate();
```

First we get the current time in milliseconds using `Date.now()`. Multiplying it by `0.002` will slow down the 'bounce' of the sine wave.

Our animation expects a range of `0 to 1`. Since sine waves are in the `-1 to 1` range, we have to normalise the range using `(1 + angle) * 0.5`.

Next, we pass the resulting `progress` variable to our animation to update the fades of the images.

Finally we use `requestAnimationFrame` (*netm.ag/requestanimation-256*) to call a function the next time the browser paints the window – this is usually synced to your monitor's refresh rate). If your browser doesn't support this, it is possible to use `setTimeout` instead:

Animating the opacity is interesting, but you can animate any CSS property in this way

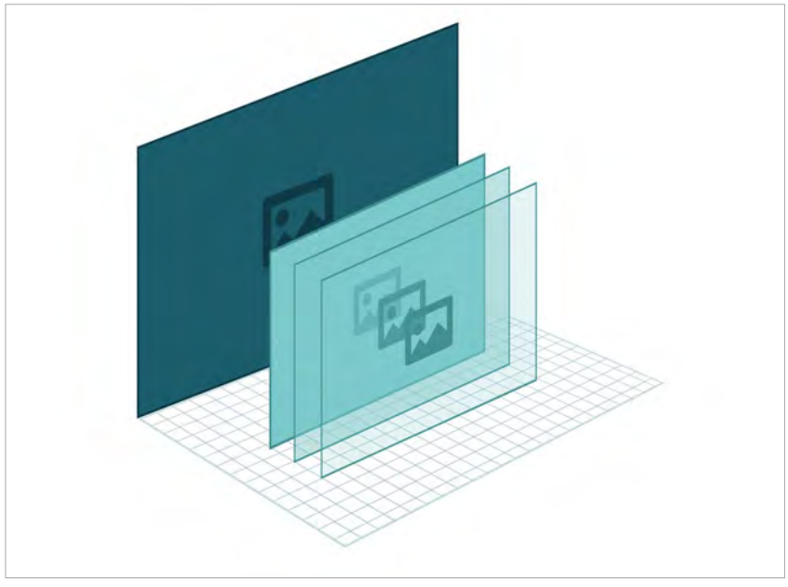
```
setTimeout(animate, 16); // 16ms = ~60fps
```

ANIMATING OTHER PROPERTIES

Animating the opacity is already an interesting effect, but you can animate any CSS property in the same way. One cool addition is to add a background image that moves along with the `progress` variable. This would create a 'parallax' effect, adding even more depth to the scene – you can check out an example at *growcode.github.io/tutorial-js-lighting/demos/parallax.html*.

First we need to update the CSS to put a background image in the container:

```
#container {
    position: relative;
    width:800px;
    height:800px;
    overflow:hidden;
    background: url(images/background.png);
}
```



Line 'em up By placing opacity-based animation layers in front of a background plane, a parallax effect helps 'sell' our illusion of lighting. Tying background image movement to the lighting animation makes the user feel like the camera is swinging around our object

Since the background image will be scrolling, it must be wider than the image container. The CSS code forces the container to stay 800x800 in size and `overflow:hidden` hides the part of the background image that falls outside that container.

In the code, we're tracking the `container` DOM object already. We will now be animating the `backgroundPosition` style in order to move the background image.

In the previous `animate()` function, we can update this property directly after the `shader.update(progress)` call:

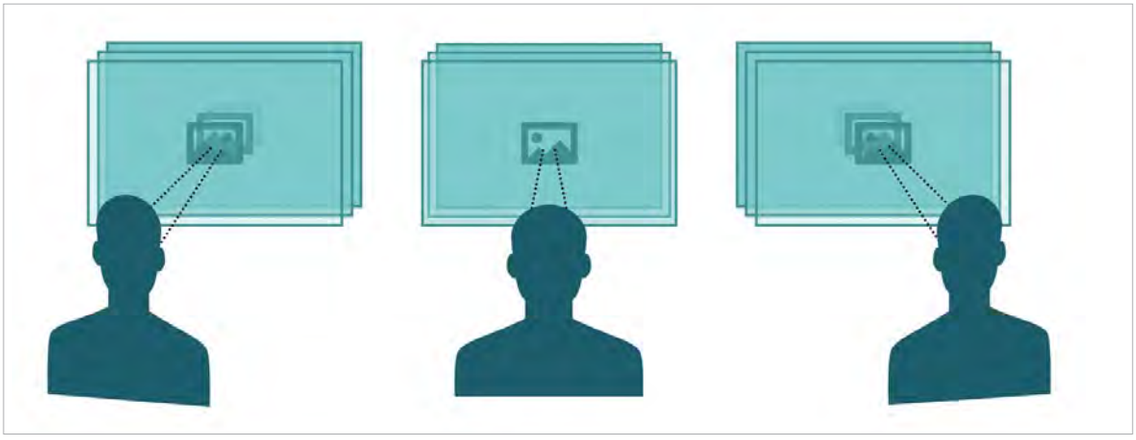
```
function animate() {
    var angle = Math.sin(Date.now() * 0.002);
    var progress = (1 + angle) * 0.5;
    shader.update(progress);

    /* Move the background image */
    container.style.backgroundPosition = -50 + progress *
50 + 'px 0';

    requestAnimationFrame(animate);
}
```

As the animation progresses, this will move the background 50 pixels.

ADDING INTERACTIVITY

Since the animation reacts to the value of the `progress` variable, we can choose to adjust that

All about perspective
Once we know the position of a user's head in relation to their webcam, we can turn off auto-animation of our lighting effect and tie it to user position. When the user moves left, we can make it feel like our light source is shifting to the right, and vice versa



variable based on any input instead of just bouncing it with a sine formula. One cool and easy input you should try using is the headtrackr library (*github.com/auduno/headtrackr*), which seamlessly adds headtracking to a web page.

By tying the animation progress to the position of the user's face, we can create the illusion that the user is 'looking around' the object. We'll capture the value of the user's face from the webcam and adjust the light shading to mimic a real-world interaction. Check out a demonstration at *growcode.github.io/tutorial-js-lighting/demos/headtracking.html*.

First, download the js file (*github.com/auduno/headtrackr/blob/master/headtrackr.min.js*) and reference it in your `<head>` tag:

```
<script src="include/headtrackr.js"></script>
```

This will expose a global `htracker` object, which we'll use later.

The headtrackr library needs both a `<video>` and a `<canvas>` element to work its magic. These are actually invisible, and we can just add them beneath the `container` div:

```
<canvas id="inputCanvas" width="320" height="240"
style="display:none"></canvas>
<video id="inputVideo" autoplay loop
style="display:none"></video>
```

Now we'll be able to listen for a `headtrackingEvent` event. The headtrackr library fires this event any time it has an updated position for the user's head. That event will tell us the `x` (horizontal) position of the user's head, which we will simplify to the `0..1` range and pass on to the animation. This would replace the `animate()` function we were using before.

```
/* Initialize headtrackr with the input video/canvas */
var videoInput = document.getElementById('inputVideo');
var canvasInput = document.
getElementById('inputCanvas');
var htracker = new headtrackr.Tracker();
htracker.init(videoInput, canvasInput);

/* Listen for when the user's head moves */
document.addEventListener('headtrackingEvent',
function (e) {
    /* Restrict value range to -10 .. 10 and divide to -1 .. 1
*/
    var headX = Math.max(-10, Math.min(10, e.x)) * 0.1;

    /* Convert range from -1 .. 1 to 0 .. 1 */
    var progress = (1 + headX) * 0.5;

    /* Update the animation */
    shader.update(progress);
});

/* Tell headtrackr we're ready for the events */
htracker.start();
```

Now after headtrackr finds your head, you should be able to move around in order to change the light shading animation.

Laptop users can actually move the laptop instead of their head, making it appear more like a gyroscope-powered effect.

CONCLUSION

They say necessity is the mother of invention. But in this case we didn't need a new shiny library or browser update to solve a problem, just a few pre-existing, dependable techniques.

The code involved is neither new nor complex, although hopefully you will agree that the end result is pretty neat. It's always fun to apply things you already know to newer challenges, and it's especially cool to mix newer web technology (headtracking) with old.

RESOURCE

TOP TIP

`math.sin(x)` is a great way to animate things. Every 6.28 units (that's 2PI!) it will oscillate from -1 to 1 with a natural ease. To find out more check *netm.ag/math1-256*, *netm.ag/math2-256* and *netm.ag/math3-256*

WordPress

THE COMPLETE GUIDE

148 pages of web-building advice

Make your own website today – it's easy!

✓ Personal blogs ✓ E-commerce sites ✓ Design tips

ABOUT THE AUTHOR

ARI LERNER

w: *fullstackedu.com*

areas of expertise:
Distributed computing and advanced frontend development

q: what makes you squeamish?
a: Code without tests, blood and egregiously impersonal emails from recruiters

*JAVASCRIPT

ADD INTERACTIVITY INTO HTML WITH ANGULARJS

By building interactivity into the structure of a site, AngularJS could change the face of web development. **Ari Lerner** explains more

With the new emphasis on web technologies from companies such as Google and Facebook, and the maturing Node.js framework, web technologies have taken centre stage and are maturing rapidly. Web development is on the verge of a huge shift. When we build a web application today, we are forced to separate the structure from interactivity, and build the application in two stages.

AngularJS enables developers to combine these two equally important components, taking advantage of the fact that web browsers are becoming more intelligent and widely distributed, and we no longer need to be as sensitive to less advanced browsers. In this short introduction to AngularJS, I'll examine how the framework is altering the face of web development today.

TRADITIONAL WEB APPLICATION DEVELOPMENT

Fundamentally, browsers are event-driven applications. We write our interactivity by targeting DOM elements in our HTML using JavaScript selector functions, and adding event listeners to the element.

For instance, when we want to run a function on an element with the `id` of `buybutton`, we add a `click` event listener with a handler function to an event. Our HTML might look like this:

```
<button id="buybutton">Buy now</button>
```

To add event handling, our JavaScript might look something like so:

```
document.findElementById('buybutton')
.addEventListener('click', function(event) {
  // The button with the id of buybutton
  // has been clicked
});
```

While effective for simple applications, this method can prove troublesome in more complex applications. Even with this simple example, there is a tight coupling between the document structure and the JavaScript. That is, the element with the `buybutton` `id` tag must not change.

When design changes, or the name or DOM structure of the button changes, our brittle JavaScript is likely to stop working. Unless we have effective frontend integration tests, broken interactivity can easily be missed.

ENTER ANGULAR

Angular, on the other hand, declaratively integrates interactivity alongside the element description. We can rewrite the above example within the HTML file:

RESOURCE

INTRODUCTION
Read AngularJS the awesome parts, Ari Lerner's introduction to AngularJS: *netm.ag/awesome-261*

```
<button ng-click="buyFn()">Buy now</button>
```

Rather than concerning ourselves with how the element is found in the page, we can let Angular worry about attaching events to the elements while we figure out the business logic of our application. It frees us developers from worrying about the work it takes to apply the JavaScript, and build our application. The confusion of how elements work is shrunk significantly as the interactivity is written into the web application itself.

As the nature of embedding interactivity into our HTML declaratively defines the purpose of the DOM element, Angular changes how we teach web development. Rather than building upon the DOM, we build upon our application's data. Instead of studying on how the browser works and what CSS selectors are, we focus upon expanding knowledge of the functionality of the application.

Rather than building upon the DOM, we can build upon our application's data

GETTING STARTED

It is extremely easy to get started with Angular. There are two components required:

1 Integrating the angular.js library file in our HTML
2 Bootstrapping our application

We can integrate the `angular.js` library in the same way we can integrate any JavaScript library with a `<script>` tag:

```
<html>
 <head><title>Our first angular application</title></head>
 <body>
   <script src="scripts/angular.js"></script>
 </body>
</html>
```

To bootstrap our Angular application, we'll need to tell Angular where to set the root of our application. We can add an attribute to an element on the page called `ng-app`. This enables us to embed our application anywhere, and not take over the entire application. This allows Angular to play nicely with other web frameworks.

When we want our Angular application to control the entire page, we can place it on the root element:

```
<html ng-app>
 <!-- ... -->
```

With that, our Angular application is ready to go. When the browser loads the library, it will hook into the `onready` event fired by the browser, start parsing the DOM tree and automatically start our app. Once it is running, we have a fully functional Angular environment to interact with.

We can bind both data and functions to our HTML document by using simple directives. A directive is a function that is run on an element to give it superpowers – for example, augmenting that element's functionality. `ng-click`, for instance, adds the ability to run a function when the element is clicked.

An adding calculator might be written like so:

```
<button ng-click='total = total + 1'>+1</button>
Total: {{ total }}
<button ng-click='total = total - 1'>-1</button>
```

Building bi-directional data bindings in Angular is as easy and natural as it should be. We simply describe the data and functional interactivity as we want it to appear and Angular takes care of the rest.

Not yet convinced? Angular enables us to create custom elements easily, as well as modularising our functionality. We can write a complex web application like so (this is a simplified snippet of one of our production Angular applications):

```
<header user="currentUser"></header>
<product-list ng-model="products"></product-list>
<checkout ng-model="shoppingCart"></checkout>
```

THE COST OF DEVELOPMENT

With the growth of Angular around the web development community, it is becoming easier to find and hire Angular developers to build complex web applications. Angular strips away the work required to build boilerplate functionality, and makes it easy to build engaging web experiences in a fraction of the time it might usually take.

The community is growing larger every day and open source components are appearing in the community faster than ever. In fact, the growth of Angular is showing similarities with Ruby on Rails between 2004 through 2006.

ANGULAR IS THE WAY OF THE FUTURE

It's easy to see how Angular itself could change the face of web design. Dedicate a part of your team to building a part or the whole of your application in Angular – I promise, you'll never look back.

RESOURCE

NG-BOOK

The canonical book on Angular, highlighting everything the Angular professional needs. It's constantly being updated with new releases: *ng-book.com*

Download the files here!

All the files you need for this tutorial can be found at *netm.ag/angularjs-249*

ABOUT THE AUTHOR

LUKAS RUEBBELKE

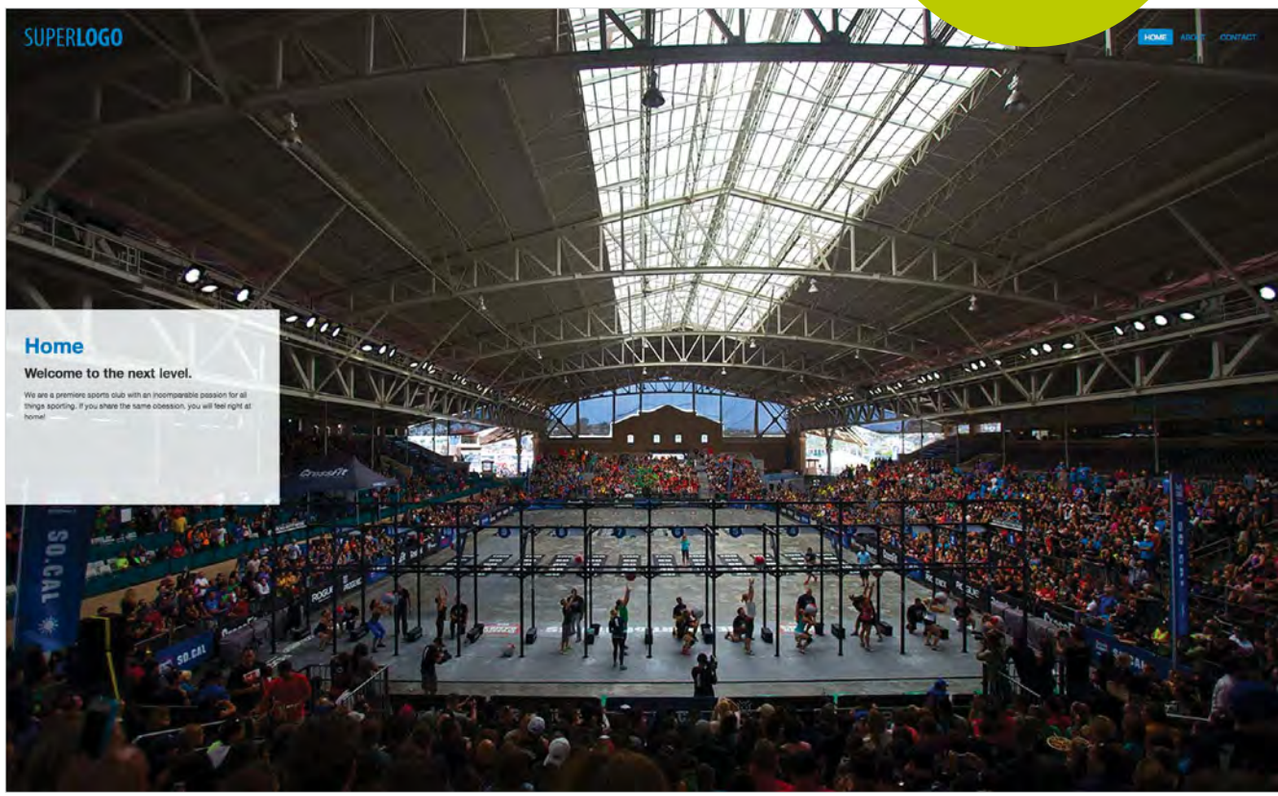
w: *onehungrymind.com*

t: @simpulton

areas of expertise: JavaScript, AngularJS, Node

q: what's the coolest thing you do offline?

a: Troubleshoot my internet connection



JAVASCRIPT

BUILD AN ANIMATED ANGULARJS WEBSITE

Lukas Ruebbelke explains how to create a simple AngularJS website and add in animations using the animation API

AngularJS was born out of the need to find a better way to create enterprise web applications and, given the meteoric rise of the framework in the last year, it's safe to say that AngularJS has succeeded in grand fashion. Not only do you have incredible tools to create large-scale applications, but the code naturally lends itself to being testable. And while I appreciate these capabilities, I've always felt like there was a vast expanse of possibilities that AngularJS had not ventured into. That all changed when the AngularJS team introduced the new animations API in AngularJS. Now we can create an engaging user experience through animations.

For the sake of time and space, I'm going to focus on the AngularJS parts specifically and not dig into the HTML and CSS structure. With that said, I encourage everyone to download the respository and explore the parts I don't get into in this article.

We're are going to start with a simple, static AngularJS website and animate it using AngularJS and TweenMax from the Greensock Animation Platform. The site has a full-sized background image for each page that slides from right to left when a new page is selected and a content panel that slides from left to right to show the content for that page. Visit the GitHub repository here: *netm.ag/git-261*.

See a demo here: *netm.ag/demo-261*. The main two files for this project are `index.html` and `js/app.js`, which serve as the starting point. Starting with these files, we'll add in the code necessary to animate the changing background and content panel on the left.

For reference, `index.finish.html` and `js/app.finish.js` contain the completed code for the animations. I'm assuming a basic knowledge of AngularJS but please refer to the documentation at *docs.angularjs.org/api* if you have any questions about a particular piece of AngularJS code.

ENABLE ANIMATIONS

Rather than in the core, animations in AngularJS are included as a separate JavaScript file called `angular-animate.min.js`, so that the `ngAnimate` module is available to our application. Keep in mind that we're using AngularJS 1.2-RC.3 (or the official 1.2 version). Now that we've added the source file for `ngAnimate`, we need to inject it into our website module by changing `angular.module('website', [''])` to `angular.module('website', ['ngAnimate'])`.

ADDING ANIMATIONS

Technically, we're starting with a fully functioning website but things are a bit underwhelming at the moment. Now that we've enabled animations, it's time to turn the tide against the boring and add in some animations. We're going to animate the background images first and then animate the content panel after.

JavaScript animations in AngularJS are created by calling the `module.animation()` factory method with the name of the animation you wish to build, and a function that will define the animation's behaviour.

```
myModule.animation('.bg-animation', function ($window) {
    return {
        enter: function (element, done) {
            someAnimation(element,
            done);
            return function(cancelled) {
                //this function is called when
                the animation is done
            }
        },
        leave: function (element, done) { }
    };
})
```

The AngularJS animation naming convention is CSS class-based. That's why we've named our animation `.bg-animation` and not `bg-animation`. AngularJS animations are actually a set of event hooks used to delegate to whatever you are actually using to do your animations whether it be CSS transitions, CSS keyframe animations or JavaScript animations. By delegating the actual animation portion, AngularJS gives you endless opportunities to handle the animations as you see fit whether it be by hand or using a third party library.

In our case, we're listening for the `enter` and `leave` event which is triggered when `ng-if` adds or removes an element to the DOM. Both event handlers take an `element` as well a `done` parameter. The `element` parameter is the element that the animation event was triggered on and the `done` parameter is a callback that needs to be called when the animation is complete so AngularJS knows that it can safely move on.

If you return a function within the animation, then that function will be fired when the animation completes or when it is cancelled. This is optional. However, it proves useful if you need to clean up any animation-related properties on the element after the animation has closed. Now that we have the basic structure in place for both animation events, it's time to add the actual animations.

A done parameter is a callback that needs to be called when the animation is complete

```
.animation('.bg-animation', function ($window) {
    return {
        enter: function (element, done) {
            TweenMax.fromTo(element,
            0.5,
                { left: $window.innerWidth},
                {left: 0, onComplete: done});
        },
        leave: function (element, done) {
            TweenMax.to(element, 0.5,
                {left: -$window.innerWidth, onComplete:
                done});
        }
    };
})
```

When a background image is added to the DOM, we want it to start at the far right of the window and move to far left.

```
TweenMax.fromTo(element, 0.5, { left: $window.innerWidth},
{left: 0, onComplete: done});
```

▶

RESOURCE

YEAR OF MOO

Year of Moo is the blog of Matias Niemelä and the defacto resource for all things relating to AngularJS animations: *yearofmoo.com*

FOCUS ON

THE FUNCTIONALITY

The underlying data structure for the website is going to be pages, so here's how to define those in our JavaScript:

```
.controller('MainCtrl', function ($scope) {
    $scope.pages = {
        'home': { label: 'Home', sublabel: 'Sublabel',
        content: 'This is page content.' },
        'about': { label: 'About', sublabel: 'Sublabel',
        content: 'This is page content.' },
        'contact': { label: 'Contact', sublabel: 'Sublabel',
        content: 'This is page content.' }
    };
    $scope.currentPage = 'home';
    $scope.page = $scope.pages['home'];

    $scope.isCurrentPage = function (page) {
        return $scope.currentPage === page;
    };
})
```

In our controller, we're defining a `pages` object on `$scope`, which is going to define our content for the site. This is essentially a key-value map that we'll use to get and set the current page as well as display the content in the HTML.

We're keeping track of the current page we're on by defining `$scope.currentPage` and initially setting it to `home`. We're also setting a `page` property on `$scope` to hold the actual content of the page we are on. We're also defining a convenience function `$scope.isCurrentPage` that returns `true` or `false` based on the value of the page parameter and the `page` we're currently on.

AngularJS binding We display the properties of our page's objects in our HTML

We accomplish this with `TweenMax.fromTo` and by telling it to animate the `element` for 0.5 seconds. At the start of animation the `left` style property is set to `$window.innerWidth` pixels and the animation itself will animate that `left` property to `0` pixels. Notice in the `to` animation object we define an `onComplete` event handler and set it to `done`.

It's also worth mentioning before we go any further that we are using the `$window` service, which is basically an AngularJS wrapper around the native window object. This service isn't extended or wrapped in any way so the `$window` service acts just like the regular window element.

The animation for when a background image is being removed from the DOM is slightly simpler.

```
TweenMax.to(element, 0.5, {left: -$window.innerWidth,
onComplete: done});
```

Because the `element` is already in place, we can get away by just using `TweenMax.to` and setting the `left` property to a negative `$window.innerWidth`. This will cause the image to slide off the screen to the left. And now that we have our `.bg-animation` defined, how do we actually hook it up so that it works? Remember

Remember that animations follow a CSS class-based naming convention

that animations follow a CSS class-based naming convention. Here are the background images without the animation.

```
<img bg class="fullBg" ng-if="isCurrentPage('home')"
src="images/bg00.jpg">
<img bg class="fullBg" ng-if="isCurrentPage('about')"
src="images/bg01.jpg">
<img bg class="fullBg" ng-if="isCurrentPage('contact')"
src="images/bg02.jpg">
```

And here are the background images with the animation enabled.

```
<img bg class="fullBg bg-animation" ng-
if="isCurrentPage('home')" src="images/bg00.jpg">
<img bg class="fullBg bg-animation" ng-
if="isCurrentPage('about')" src="images/bg01.jpg">
<img bg class="fullBg bg-animation" ng-
if="isCurrentPage('contact')" src="images/bg02.jpg">
```

ya

And that is it! You simply have to add the animation to your element as if it were a CSS class. This is starting to look a lot like a class-based directive isn't it? Really powerful stuff!

BG DIRECTIVE

You may have noticed the `bg` attribute on our background images. This attribute actually represents a directive that I wrote to make the images correctly size to the full width and height of the screen. I did this by converting the awesome jQuery.fullBG plugin from @bavotasan. You can check out the directive in the source files and read about the original plugin here: *netm.ag/redux-261*.

ANIMATING THE CONTENT PANEL

We're almost done with our website. The only piece missing is animating the content panel. We're going to handle this a little differently by toggling its visibility binding `ng-hide` to a property called `isInTransit` on `$scope`.

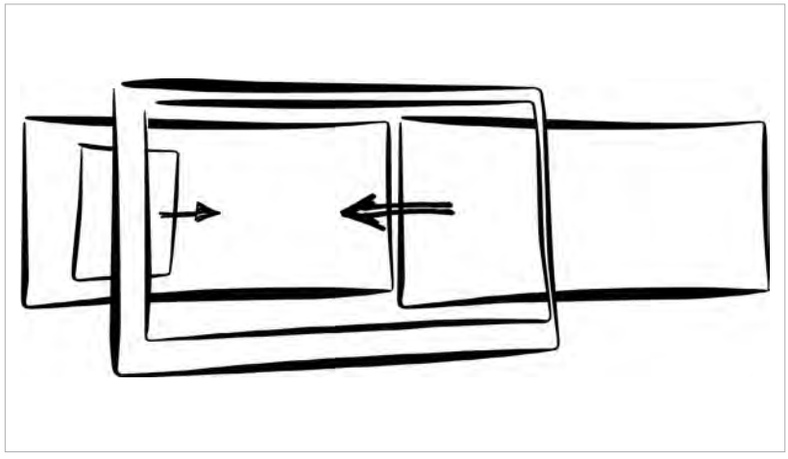
We're defining `isInTransit` on `$scope` and then setting it to `false` since we want the content panel to be visible initially.

```
.controller('MainCtrl', function ($scope) {
    // Code omitted
    $scope.isInTransit = false;

    $scope.setCurrentPage = function (page) {
        if ($scope.currentPage !== page) {
            $scope.page = $scope.pages[page];
            $scope.currentPage = page;
            $scope.isInTransit = true;
        }
    };

    $scope.$on('bgTransitionComplete', function(){
        $scope.isInTransit = false;
    });
})
```

We're setting `isInTransit` to `true` when a new page is set which will cause the content panel to hide itself. We're also setting `isInTransit` to `false` when the `bgTransitionComplete` event is triggered. Make a mental note of that event because we will get to where it gets fired in just a moment.

Now that `isInTransit` is defined and we have a way to set it to `true` or `false`, it's time to wire it up to the HTML.

```
<div class="panel panel-animation" ng-hide="isInTransit" >
<!-- Code omitted -->
</div>
```



Content panel The content panel slides in from the left while the background image slides in from the right

In the previous code, we are toggling visibility of the content panel based on the value of `isInTransit` with the code `ng-hide="isInTransit"`.

We have also added the class `panel-animation`, which we're going to define as an AngularJS animation next.

```
.animation('.panel-animation', function () {
    return {
        addClass: function (element, className,
        done) {
            if (className == 'ng-hide') { }
            else { done(); }
        },
        removeClass: function (element, className, done) {
        if (className == 'ng-hide') { }
            else { done(); }
        }
    };
});
```

The underlying functionality of the `ng-hide` directive is accomplished by adding and removing a class called `ng-hide`, which explains why the events for this animation are `addClass` and `removeClass`. In this case we only want to perform an animation if the class being added or removed is `ng-hide`, which is why we are checking the `className` parameter.

If the `className` is `ng-hide`, then we will perform the animation and, if not, then we simply call the `done` callback (which basically skips the animation). And now that the structure is in place, it is time to add in the TweenMax animations.

```
.animation('.panel-animation', function () {
    return {
        addClass: function (element, className, done) {
            if (className == 'ng-hide') {
```

▶

RESOURCE

GREENSOCK

To learn more about the Greensock Animation Platform, check out the Greensock site. We've barely touched the surface of what the library is capable of: *greensock.com*

```
                    TweenMax.to(element, 0.2, { opacity:
                    0, onComplete: done });
            }
            else {
                    done();
                }
            },
            removeClass: function (element, className, done) {
                if (className == 'ng-hide') {
                    element.removeClass('ng-hide');
                    TweenMax.fromTo(element, 0.5,
                        { opacity: 0, left: -element.width() },
                        { opacity: 0.8, left: 0, onComplete:
                        done });
                }
                else {
                    done();
                }
            }
        };
})
```

When `ng-hide` is added, we want to animate the content panel off the stage.

```
TweenMax.to(element, 0.2, { opacity: 0, onComplete: done });
```

We're going to do this by setting `opacity` to `0` over the course of 0.2 seconds. And when `ng-hide` is removed, we are going to slide the content panel in from the left and fade it back in.

```
element.removeClass('ng-hide');
TweenMax.fromTo(element, 0.5,
    { opacity: 0, left: -element.width() },
    { opacity: 0.8, left: 0, onComplete: done });
```

We're going to use `TweenMax.fromTo` to start with (the panel to the left of the screen) and an `opacity` of `0` with the `from` object `{ opacity: 0, left: -element. width() }`. We're then going to fade it in by setting opacity to `0.8` or `80%` and `left` to `0`. Because we cannot set `!important` programmatically with JavaScript, we are removing the `ng-hide` class manually with `element. removeClass('ng- hide')`. This is the unfortunate reality of doing animations in this manner, but it's only a small hoop to jump through.

Remember I said to note the `bgTransitionComplete` event we were listening for in the controller? Well, as it stands, we have a way to set `isInTransit` to `true` that will hide the content panel, but we aren't setting it to `false` to bring the content panel back. We want the content panel to slide in after the background image has finished animating so that's where we're going to fire that event. For example, see following code:

```
.animation('.bg-animation', function ($window, $rootScope) {
    return {
        enter: function (element, done) {
            TweenMax.fromTo(element, 0.5,
                { left: $window.innerWidth},
                {left: 0, onComplete: function () {
                $rootScope.$apply(function(){
                    $rootScope.$broadcast('bgTransitio
                    nComplete');
                });
                done();
            }});
        },
        // Code omitted
    };
})
```

The first thing we need to do is inject `$rootScope` into `.bg-animation`, so that we can use it to broadcast events on. Then we're going to define an actual function to be called when `onComplete` fires. In that function handler we're going to put this bit of code.

We're going to define an actual function to be called with onComplete fires

```
$rootScope.$apply(function(){
        $rootScope.$broadcast('bgTransitionComplete');
});
done();
```

Because the `onComplete` callback is happening as a part of a TweenMax operation, we need to inform AngularJS that something has happened that it needs to know about. That is why we are calling `$rootScope.$apply` and calling `$rootScope.$broadcast('b gTransitionComplete');` in the closure. And last but not least, we are calling `done()` to finish off the animation and give control back to AngularJS.

CONCLUSION

We have a completely functional website built on AngularJS and Greensock! The most impressive part for me is that once the functionality is in place, setting up the animations is fairly easy.

I recommend checking out the Year of Moo (*yearofmoo.com*) blog by Matias Niemelä. He wrote `ngAnimate`, so is very qualified to talk about it, and has provided some resources on learning how to use the new API.

REVIEW

MATIAS NIEMELÄ

With thanks to Matias Niemelä for his peer review of this tutorial. Matias is a core developer on the AngularJS project.

ABOUT THE AUTHOR

ALEX MATCHNEER

w: *machty.com*

t: @machty

areas of expertise:
JavaScript engineer, member of the Ember.js core team

HEAD TO HEAD

ANGULARJS VS EMBER.JS

Alex Matchneer compares the AngularJS toolset and the Ember.js framework for building client-side web applications

ANGULARJS.ORG	EMBERJS.COM
AngularJS is a lightweight, Google-sponsored toolset for building client-side web applications. It takes an HTML-first approach to web app development while providing a foundation for data-bound templates, testability and dependency injection.	Ember.js is an open source, heavyweight framework for building client-side web applications. It provides universal data-binding and a URL-driven approach to structuring applications with a focus on scalability. It was forked from SproutCore in 2011.

STRENGTHS

Angular's beauty lies in its simplicity. Relative to Ember, it has a much smaller API surface and it's very easy for beginners to get up and running. It also features extremely friendly, built-in capabilities for testing and dependency injection. Angular's HTML-first approach to application development via directives is very powerful, and is arguably the most elegant, fine-grained API solution for manipulating or decorating the behaviour and structure of HTML elements.

Ember's strength lies in the conventions it provides (and somewhat enforces) to allow you to build scalable apps via community-driven, battle-tested patterns. The Ember Router enables you to very easily describe complex nestings of your app's state in a URL-accessible manner. It provides a proprietary object model, which enables classic object-oriented patterns of class extension and mixin inheritance in your apps, as well as universal data-binding between all objects.

CHALLENGES

There's very little guidance for structuring applications, and there is much community fragmentation, particular concerning medium to large apps as to best practices for scaling Angular apps. The few devs who know how to scale take vastly different approaches, so there's very little battle-testing for any particular approach to scalability. Angular lacks universal data-binding, which necessitates a separate, often slow and error-prone API for propagating change events between objects outside the view/template layer.

Ember's API surface is huge and its learning curve is quite steep. Its object model requires the use of getter and setter methods rather than simple JavaScript property assignment, which some developers find jarring. As a fork of SproutCore, Ember wasn't built from the ground up with testability in mind, and has only recently begun to support decent patterns for testability, but it's not as straightforward as Angular's approach and often requires an understanding of Ember internals to avoid certain gotchas.

VERDICT

If you're planning on building anything large, the Ember.js framework is almost certainly the right choice for you and your project. Otherwise, you should give both AngularJS and Ember.js an honest try to see what suits you and your application's needs the best.

FACT FILE

FUTUREPROOF

Both frameworks feature APIs that target and converge nicely with the Web Components specifications, and both stand to benefit from JavaScript features made available in the ES6 specifications.

DEPENDENCIES

Angular (35kb min+gz) has no strict dependencies, but will use jQuery if it is present. Ember (95kb min+gz, including dependencies) depends on jQuery as well as the Handlebars.js templating library.

Download the files here!

All the files you need for this tutorial can be found at *netm.ag/howarth-257*

ABOUT THE AUTHOR

BENJAMIN HOWARTH

w:*about.me/benjaminhowarth*

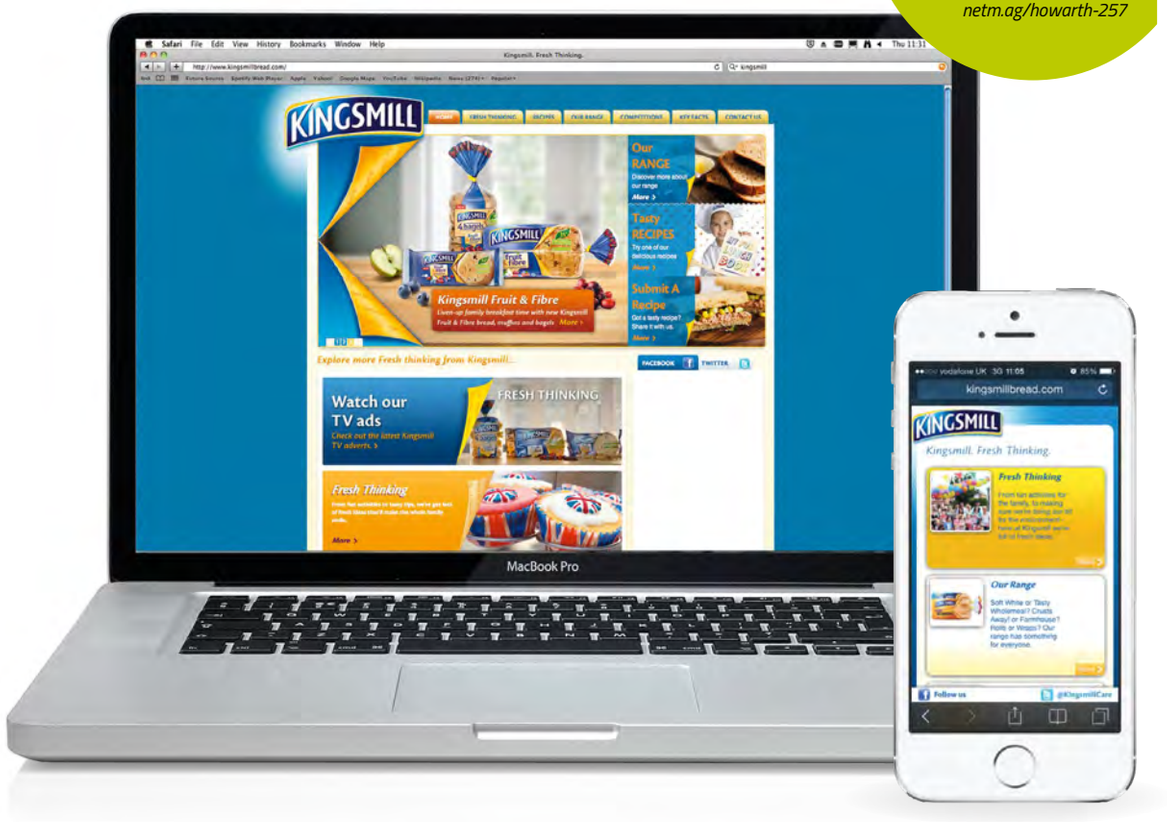
t: @benjaminhowarth

areas of expertise:

a: Open source frameworks on the Microsoft .NET stack

q: What makes you squeamish?

a: Moths. Especially when they fly round your head – fuzzy things near my ears make me want to run away very fast



SPA

TRANSFORM YOUR WEBSITE INTO AN SPA

Benjamin Howarth explains how to take an accessible website and turn it into a scalable, manageable single-page app

As the plethora of devices, platforms and browsers grows larger every day, single-page applications (SPAs) are becoming more and more prevalent. HTML5 and JavaScript are now first-class citizens in application development, as well as revolutionising how we design and create great website experiences with CSS3.

As a result, the traditional pattern for developing a comprehensive 'digital strategy' is changing fast, and becoming a lot more streamlined. Platforms like PhoneGap enable us to create apps in HTML5 and JavaScript, which can then be adapted and cross-compiled for various mobile platforms (iOS, Android, Blackberry and Windows Phone). However, gaps still persist between languages and frameworks – especially between building 'apps' for mobile and tablet platforms, and building traditional, accessible and progressive websites.

As a freelancer, I've experienced clients asking for a website, then a mobile website, then an app (I built Kingsmill Bread's mobile and desktop/tablet websites on the MIT-licensed, .NET-based Umbraco CMS). The 'traditional' route of building this would be to do the site first, with a responsive (or specially-designed) CSS layout, then build a separate app that plugs into an underlying content store.

This model works great if you have specialists in iOS, Java and Silverlight (yes, Silverlight – the

VIDEO

See this SPA tutorial in action in Benjamin Howarth's exclusive screencast at: *netm.ag/tut1-257*

Sliced bread The Kingsmill Bread desktop and mobile websites share content using the Umbraco CMS, with user-agent specific templates

joys), but falls flat when it comes to scalability and manageability. Factor in the average cost of bespoke app development (approximately £22,000) and average return (£400 a year, if you're lucky – see *netm.ag/economics-257*), and suddenly you're pouring money into an awfully large technological black hole.

WHAT ABOUT JAVASCRIPT?

"JavaScript saves the day!" I hear the frontend developers cry. Not so fast. JavaScript solves the majority of the cross-platform issues, but then you're not building a website any more. Google doesn't index content loaded with AJAX, so you've lost accessibility and progressive enhancement: the key tenets of good web architecture.

The traditional pattern for developing a digital strategy is becoming a lot more streamlined

What's that I hear? "NodeJS saves the day! JavaScript everywhere!" Hold your horses there, hipster. NodeJS doesn't yet have a JavaScript-based CMS. "What about all those lovely, stable, mature content management systems that make websites awesome for their content managers?" Are you seriously suggesting direct-porting WordPress, Drupal, Umbraco et al to JavaScript just to solve this problem? (I had someone suggest this solution to me at the Scotch on the Rocks conference this year, and my reply was pretty much exactly that).

So how do we design a website that's accessible, and then 'upgrade' it to look, feel and behave like ▶

IN-DEPTH

WHAT'S A PROMISE?

In functional programming languages, a promise is effectively a proxy for lazy loading, thus preventing any operations from blocking your execution (and subsequent debugging). A perfect example from their documentation is replacing this:

```
step1(function (value1) {
  step2(value1, function(value2) {
    step3(value2, function(value3) {
      step4(value3, function(value4) {
        // Do something with value4
      });
    });
  });
});
```

With this:

```
Q.fcall(promisedStep1)
.then(promisedStep2)
.then(promisedStep3)
.then(promisedStep4)
.then(function (value4) {
  // Do something with value4
})
.catch(function (error) {
  // Handle any error from all above steps
})
.done();
```

This enables you to manage your functions asynchronously, allowing each to run in its own thread, and making debugging more manageable. We can also manage UI updates whilst we're doing background work, such as retrieving JSON data from any APIs we're working with. BreezeJS uses Q to allow us to write the following:

```
var client = new BreezeManager("baseURI")
.query("RestfulURIToQueryMyData")
.success(function(data) {
  // do something with data
}).fail(function(data) {
  // throw an error
}).complete();
```

It gives us separation in an inversion-of-control process in a functional language. This is very useful for complex UIs, that have rich interaction with backend JSON services. Here we're using it to show you how to interact with database data on the server, served up by Microsoft's Web API stack.

an SPA? I'm glad you asked that, because that's what we're going to build in this tutorial.

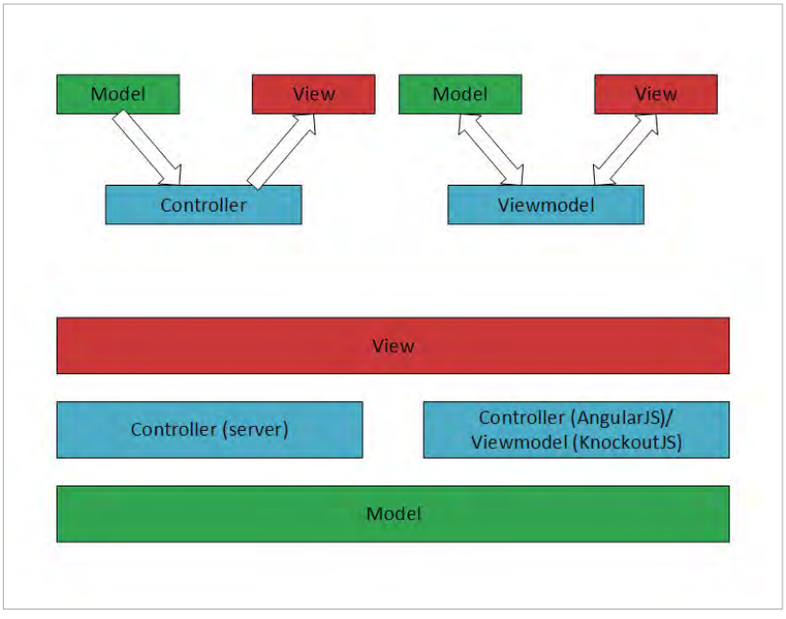
Architecture is everything, I will cover a simple blog site, which you can then go on to adjust according to your project. I'm a .NET bod by trade, and it's worth noting this tutorial is partially written in .NET for the server-side code. If you don't like that, the design patterns will easily translate into an MVC framework of your choice – Zend, CodeIgniter, Ruby, Django and so on.

BRIDGING THE GAP

Most websites nowadays are built with a model-view-controller (MVC) architecture, whereas apps, which rely on data loaded on-demand, are built with a model-view-viewmodel (MVVM) architecture. The two paradigms are related, but differ slightly in event handling. Whereas a controller would have an action to handle a particular event (submission of data), a viewmodel would have a specific event defined for handling UI updates and any model binding modifications (people familiar with C# will know this as the INotifyPropertyChanged pattern).

The key tenet in this architecture is working out how to share the model and the view between the two patterns. When designing your site in this fashion, 99.9 per cent of the time your views will be identical both on the server and client sides. The only difference is when and how they are populated with model data – be it by controller on the server, or controller on the client.

We'll start off with creating a simple website (which can be downloaded at *netm.ag/demo-257*) using ASP.NET MVC 4. It's got a couple of controllers in it, a model created with Entity Framework to show an example of server-side model population, and some views with a responsive CSS3 template built with Bootstrap. It contains everything on the left-hand side of the diagram below – a basic model, a controller and a couple of views, just to get everything started.



Both sides Architecture of MVC and MVVM design patterns, and how we're going to try and bring them together across the client and the server

IT'S A BREEZE

We're now going to install AngularJS and BreezeJS into the project. NuGet (the package manager for Visual Studio) makes this simple to add into our project. BreezeJS depends on Q, a tool for making and managing asynchronous promises in JavaScript.

To create a very simple SPA experience from our existing site foundation, we need to share the routing table in MVC on the server with AngularJS.

Sites are built with MVC architecture, whereas apps are built with MVVM architecture

We'll do that using a controller that serialises and spits out the routing table in a format that Angular will understand.

Angular expects routing data to be configured as follows:

```
when('/blog', {
    templateUrl: 'partials/blog.html',
    controller: 'BlogListCtrl'
  })
.when('/blog/post/:blogId', {
    templateUrl: 'partials/blog-post.html',
    controller: 'BlogPostCtrl
  })
.otherwise({
    redirectTo: '/index'
  });
```

As we can see from this code, Angular's routing provider expects to know both the controller and the corresponding view. We'll be feeding AngularJS a generic controller to catch most of our requirements, and then extending it for particular views (i.e. blog list and blog post, as these contain more than just vanilla data).

To do this, we need to identify which controllers (and corresponding routes) are to be fed from the server into Angular. This is where we take advantage of .NET's reflection capabilities – code that reads

code, or 'code-ception' as I like to think of it. Warning: here be dragons! We'll be marking our controllers on the server. This is to be shared with the JS side of the site, with metadata, using a .NET feature called Reflection to gather these controllers up. On these controllers, we'll be marking out our actions with metadata to identify the corresponding JS view, model and controller, as shown in the diagram on the right.

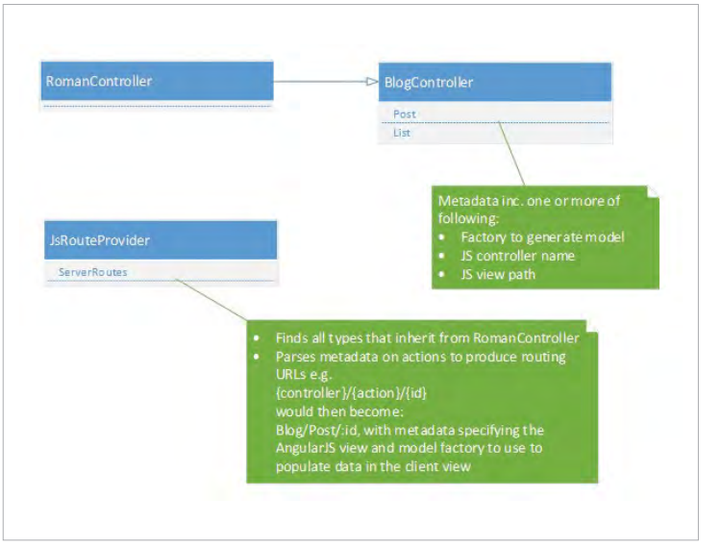
This approach is not foolproof. If you have filters or constraints on the routes or controller actions, whatever is provided by the JS routeProvider as Angular routes may not actually end up being rendered by the server. For more information on advanced MVC routing, route constraints and filtering, I highly recommend checking out Scott Allen's courses on Pluralsight on the ASP.NET MVC stack (*netm.ag/allen-257*).

PIECE BY PIECE

Now we've provided Angular with our routing table, we need to start gluing it together. Firstly, we update our controllers to inherit from our generic RomanController. We'll be overriding some parts of this later, but for now it's a generic marker for us to identify the routes and URLs we want Angular to have access to.

Next, we need to mark the actions we want to be available in AngularJS, with a custom attribute: `RomanActionAttribute`.

```
public class RomanDemoController : RomanController {
    private RomanSPA.Demo.Models.
RomanSPAStarterKitEntities _context;
    public RomanDemoController() : base() {
      // Yes, I'm not using dependency injection for my DB
context, cause this is a demo ;-)
      if (_context == null) _context = new Models.
RomanSPAStarterKitEntities();
    }
    [RomanAction]
    public ActionResult Index() { return View(new
IndexModel()); }
    [RomanAction(Factory=typeof(BlogListFactory), Controlle
rName="BlogController", ViewPath="/assets/blog-list.html")]
    public ActionResult Blog() { return View(_context.
BlogPosts); }
    [RomanAction(ControllerName="BlogPostController")]
    public ActionResult BlogPost(string slug) {
      if (_context.BlogPosts.Any(p => MakeTitleUrlFriendly(p.
Title) == slug)) {
        return View(_context.BlogPosts.First(p =>
MakeTitleUrlFriendly(p.Title) == slug));
      } else {
        return HttpNotFound();
```



Populating routes A UML diagram of how our routes are populated from the metadata on each action, which we specify with RomanActionAttribute

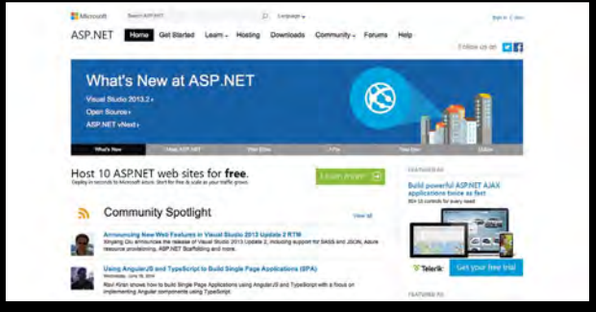
FOCUS ON

ACTION FILTER?

Microsoft's ASP.NET MVC platform is a framework that demonstrates an aspect-oriented programming (AOP) pattern. AOP frameworks allow you to decorate your code with metadata attributes that then point to methods which fundamentally alter the behaviour of the code being executed at runtime.

Examples of AOP frameworks include Spring for Java (and Spring.NET for us MS bods, which is a direct port), FLOW3 and PECL support for AOP in PHP, and a commercial tool I use called PostSharp that's a .NET 'inter-weaver' for adding AOP to non-MVC projects, by wrapping around code as you're compiling it.

An action filter is an example of a cross-cutting concern in AOP. In the ASP.NET MVC stack, it can be used to modify the result of an action on a controller. In this tutorial, I use it to potentially serve up different data, depending on the parameters supplied.



ASP In the ASP.NET stack, action filters can change the result of an action

```
        }
    }
    [RomanAction]
    public ActionResult About() { return View(); }
    private string MakeTitleUrlFriendly(string title) {
        return title.ToLower().Replace(" ", "-");
    }
}
```

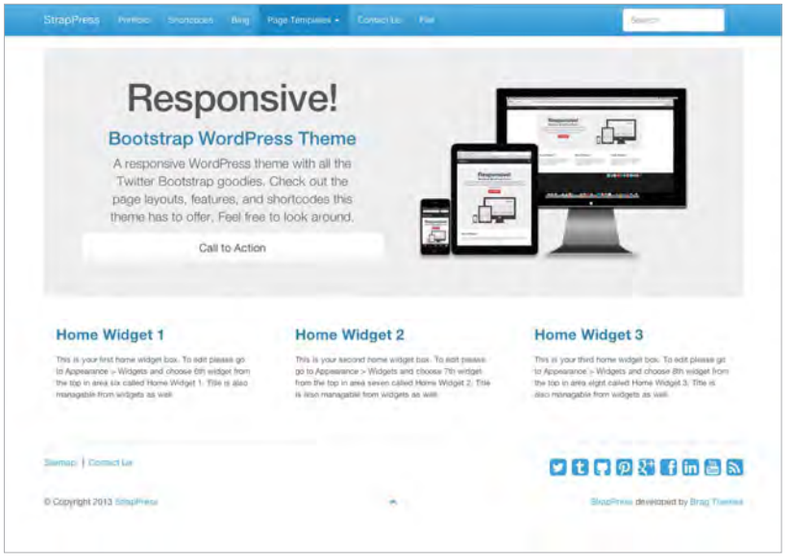
This attribute is an action filter, and has three parameters, all optional: model factory, controller name and view name. This gives our routing table the option to specify any explicit overrides we want, or to go with a generic, default behaviour (i.e. load a controller with no JSON model data, which then loads a partial view from the server). Note how `[RomanActionAttribute]` can be automatically shortened to `[RomanAction]` by the C# compiler.

The fundamental part of this exercise is that we should be able to share views and expose server data for complex transformations in our app. This means we can take advantage of HTML5 offline caching to create a great user experience on the app side.

We'll leave the `Index` and `About` views blank. For the `BlogList` action, we're going to specify a factory, a custom controller and a custom view location for the client-side view. For the `BlogPost` action, we'll specify a custom controller to load the individual post from the JSON we get from our BlogList factory.

Now, we set up a `GenericController` in AngularJS, in `/App/` controllers, and we set up auto-routing in AppJS so that all the routes are provided from the server to the client. You can provide extra routes manually by overloading the `RoutesApiController` and overriding the `ExtraRoutes` property with your own custom list of routes to pass to AngularJS.

Templates Bootstrap is the go-to CSS framework for a responsive site these days, spawning dozens of online template markets




```
angular.module('RomanSPA', ['ngRoute'])
    .config(['$routeProvider', function ($routeProvider,
$locationProvider) {
        $.ajax({
            url: '/api/RouteApi/AllRoutes',
            success: function (data) {
                for (var i = 0; i < data.length; i++) {
                    $routeProvider.when(data[i].RoutePattern, {
                        controller: data[i].controller,
                        templateUrl: data[i].templateUrl
                    });
                }
                $routeProvider.otherwise({ redirectTo: '/' });
            },
            fail: function (data) {
            }
        });
    }])
    .value('breeze', window.breeze);
```

CORE FUNCTIONS

Finally, in our GenericController, we put in two core functions – one to manually apply our specified template, and one to retrieve the model using the factory on the action attribute.

```
angular.module('RomanSPA')
    .controller('GenericController', ['$scope', '$route', function
($scope, $route){
        $scope.pageUrl = '/';
        if ($route.current !== undefined) { $scope.pageUrl =
$route.current.templateUrl; }
        // Retrieve our view - be it from server side, or custom
template location
        $.get({
            url: $scope.pageUrl.toString(),
            beforeSend: function(xhr) { xhr.setRequestHeader('X-
RomanViewRequest', 'true'); },
            success: applyView
        });
        // Retrieve our model using the modelfactory for our
current URL path
        $.get({
            url: $scope.pageUrl.toString(),
            beforeSend: function (xhr) { xhr.
setRequestHeader('X-RomanModelRequest', 'true'); },
            success: applyModel
        });
        function applyView(data) {
            $scope.$apply(function () { angular.element('body').
html($compile(data)($scope)); });
        }
        function applyModel(data) { $scope.model = data; }
}
    }]);
```

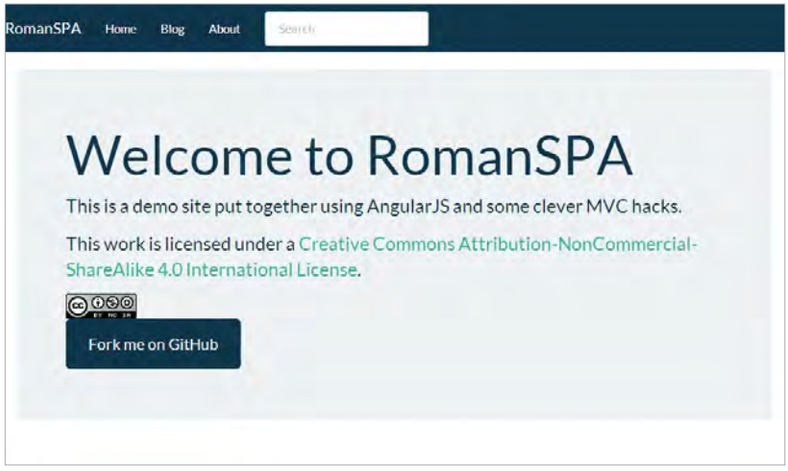
Finally, we're at a point where:

- We have gathered up all the routes on the server that we want AngularJS to take advantage of. These will be exported to Angular's routing library when our app boots up.
- We have specified a generic controller, from which our specific controllers can inherit. However, this means we have 'generic' SPA functionality, with pages partially loading as we navigate through the site.
- Child actions that we may want AngularJS to have access to, but shouldn't be in the routing table (such as navigation, footer and other partial views), can be marked out with `[RomanPartial]`.
- A request to `/RomanDemo/Index` will give us a full page, but when AngularJS requests it, it'll either provide a partial view or a JSON object, depending on the metadata we have supplied.
- Actions we want to specify metadata for (or export to Angular routing) – i.e. custom JSON model, custom template URL or custom AngularJS controller – are marked with `[RomanAction]`.

We can take advantage of HTML5 offline caching to create a great app user experience

We're almost set. From here, to guide you in the kind of direction you can go with your hybrid site app, we'll create a BlogController that inherits from `GenericController`, and use this to gather up all the blog posts we want. We'll then use this as a way to enable users to navigate a blog, whilst most of the data is held in HTML5 offline storage for us.

```
angular.module('RomanSPA')
    .controller('BlogController', ['$scope', function ($scope) {
        $controller('GenericController');
        function storePostsOffline(data) {
            if (!supportsLocalStorage()) { return false; }
            localStorage["RomanSPA.data.blog.posts"] = data;
            return true;
        }
        var manager = new breeze.EntityManager('/breeze/
BlogApi');
        var query = new breeze.EntityQuery('BlogPost');
        manager.executeQuery(query)
            .then(function(data) {
                storePostsOffline(data);
                $scope.
```



In action The initial site template booted up, with a quick Bootstrap theme from *bootswatch.com*

```
posts = data;
            })
            .fail(function(data) {
              // silently fail for now
            });
    }]);
```

HTML5 offline storage is the biggest reason you'd want to do this. You can also extend the view engine to automatically store templates and views offline. By doing this, you're allowing your website visitor to treat your website more like an app – which they can use even while they're without WiFi or phone signal.

WRAPPING UP

We now have:

- A basic MVC site, with routes for basic pages and a blog.
- An MVC app that sits on top, powered by AngularJS, with a matching route table.
- AngularJS making AJAX requests using jQuery, which server-side MVC then interprets to return JSON model or HTML view appropriately.

The result of this mashup? The ASP.NET MVC RomanSPA framework! (vomitorium, sickle-scrapers and sulphur-tasting water all optional.) You can use this foundation to build out many more complex parts to your site.

The core framework, and extensions for both AngularJS and KnockoutJS or BreezeJS, have been 'forked' into separate projects. You can either install the AngularJS or KnockoutJS editions, which will get you kick-started straight away with all the necessary prerequisite libraries installed, or you can simply install the core and write your own extensions for EmberJS or any other MVx JavaScript framework that takes your fancy.

RESOURCE

NUGET

NuGet deals with dependency management and downloading to keep your development simple. The packages for this tutorial can be found here: *netm.ag/NuGetAngular-257* *netm.ag/NugetKnockout-257*

Download the files here!

All the files you need for this tutorial can be found at *netm.ag/leggetter-258*

ABOUT THE AUTHOR

PHIL LEGGETTER

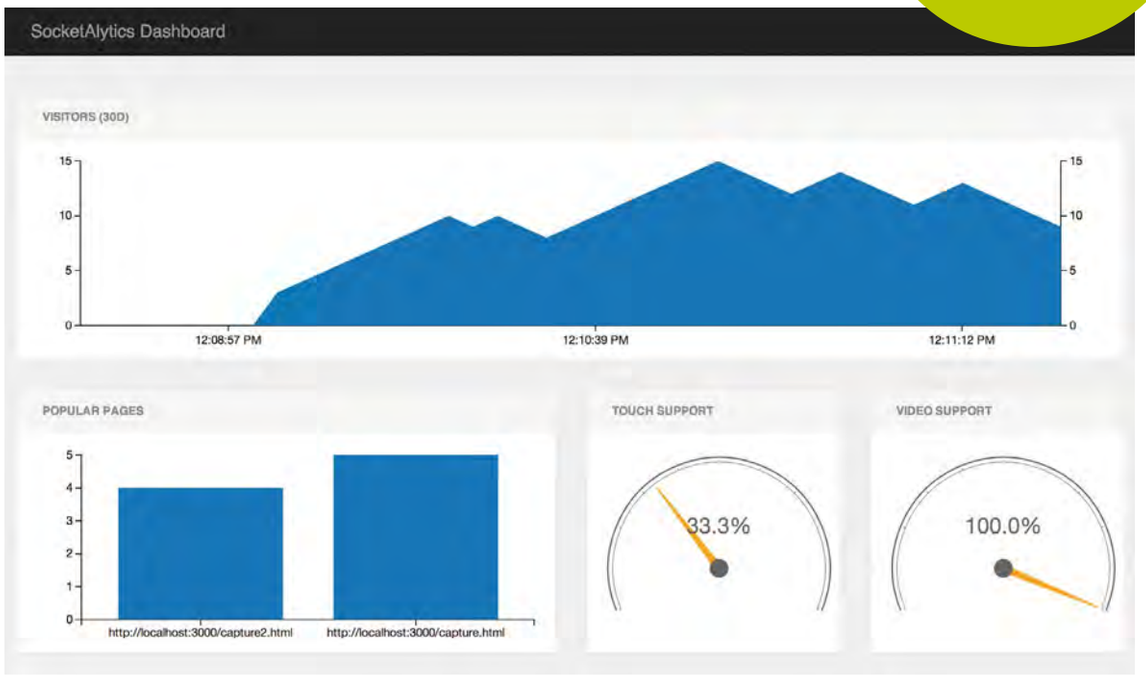
w: *leggetter.co.uk*

t: @leggetter

areas of expertise: Real-time web technologies, JavaScript, HTML5

q: what's the most heroic thing you've ever done?

a: Helping a boy who'd just been hit by a car



REAL TIME

BUILD A REAL-TIME APP WITH SOCKET.IO

Phil Leggetter on how to build an analytics dashboard with the world's most popular real-time framework, Socket.IO

Socket.IO is probably the best known of all the real-time web frameworks. In combination with Node.js, it's responsible for a significant increase in awareness of the benefits of 'the evented web', real-time data and real-time interactive user experiences. In this tutorial, I'll take a look at what's in the new Socket.IO 1.0, and show how it can be used to build an app for real-time analytics.

To keep things simple we'll capture the page URL and a couple of browser capabilities using Modernizr (*modernizr.com*). That information will be sent to the server using a Socket.IO connection. The server will do some basic analysis on that data, store it and publish the data to a dashboard. The dashboard will receive the information and display it using the Epoch charting library (*fastly.github.io/epoch*).

GETTING STARTED

For this app you're going to need at least Node.js v0.10.25 installed (*nodejs.org/download*). You also need to download the 'getting started' zip from *netm.ag/gettingstarted-258*. Unzip that archive and you'll see a `node_modules` directory with all the Node.js modules we need for this tutorial.

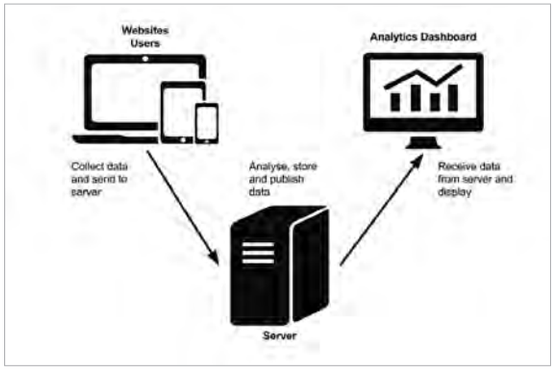
We'll put all our client-side code in the `public` directory – there's a `bower_components` directory in there, containing all the JavaScript libraries our app will depend on. There is also a dashboard directory where we'll build the `dashboard` frontend.

Let's get Socket.IO set up and create a test page that connects to the running server. First create an `index.js` file in the root of your working directory. Since we want to serve a dashboard, we'll start by

RESOURCE

TWITTER

Socket.IO has a Twitter presence (@SocketIO) and frequently shares examples of how others have used Socket.IO. It's a great resource for best practice and as inspiration.



Starting up The real-time analytics dashboard setup

creating a web server using Express (*expressjs.com*), one of the most popular Node.js web server packages:

```
var express = require( 'express' );
var app = express();
app.use( express.static( __dirname + '/public') );
var server = require( 'http' ).Server( app );
server.listen( 3000, function(){
  console.log( 'listening on *:3000' );
} );
```

Socket.IO is responsible for increased awareness of the benefits of 'the evented web'

In `index.js` we're using Express' `express.static` function to declare that all files in the `public` folder can be served directly. Let's create a `capture.html` in the `public` directory. Run node index.js in your working directory from a command prompt and navigate to `http://localhost:3000/capture.html` in your browser. Make sure you see that file being served. Now we can add Socket.IO to `index.js`.

```
var server = require( 'http' ).Server( app );
var io = require( 'socket.io' )( server );
io.on( 'connection', function( socket ) {
  console.log( 'We have a connection!' );
} );
```

A reference to the running server is gained via `require( 'http' ).Server( app )` and Socket.IO can then attach to that server using `var io = require( 'socket.io' )( server )`. We also bind to the `connection` event using `io.on( 'connection, function() {} )` and log to the connection when the event is triggered. This event signifies that a connection from a client has been detected. ▶

FOCUS ON

DEBUGGING SOCKET.IO APPS

It's highly unlikely that we're going to be able to build an application without making a single mistake, so at some point during the development of every application we need to debug. When using a framework – where the code is third-party – digging into unfamiliar code can represent a significant barrier. So, the logging, debugging and tooling support offered by a framework is very important.

One of the highlighted features in the Socket.IO 1.0 release was better debugging (*socket.io/docs/logging-and-debugging*). So what are the options to help debug a Socket.IO application?

You can get fine-grained logging from the Socket.IO server by running the application and using a `DEBUG` environment variable. For our application, the following would list all debug output.

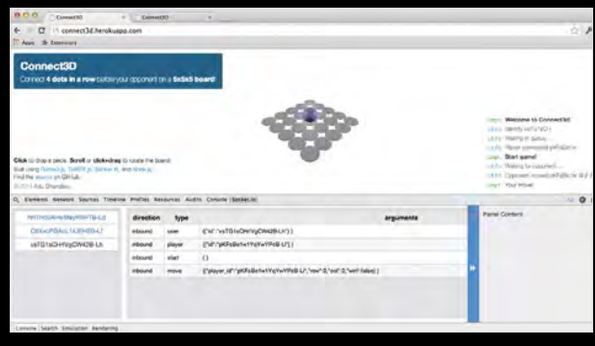
```
DEBUG=* node index.js
```

The most useful initial level is probably `socket.io:socket`. On the client the debug level is set via `localStorage.debug`.

```
localStorage.debug = '*';
```

As per the server, the most useful initial level is probably `socket.io-client:socket`. Both of these options use the debug instrumentation library (*github.com/visionmedia/debug*).

On the client there's also the option to use the Socket.IO Chrome extension (*netm.ag/extension-258*) and there are plans afoot to create a cross-browser extension or addon.



Debugging The Socket.IO Chrome extension helps track sessions and event data

FOCUS ON

CLIENT-SIDE JS PACKAGE MANAGERS

Managing your application's dependencies can be painful, so it's no surprise that almost all languages have package management solutions. Most people will have heard of NPM when it was associated with Node.js, but there are also options available to client-side JavaScript and other client-side technologies.

- Bower (*bower.io*) is probably the most popular choice. This was used in the real-time analytics application, but the dependencies were checked in to simplify the getting started process. `bower.json` is the manifest file used to reference dependencies. One restriction is that Bower supports a flat dependency tree, so multiple versions of the same package aren't supported.
- NPM (*npmjs.org*) is generally referenced in assocation with Node.js packages, but the contents of a package doesn't have to be a Node.js module. NPM is seeing increased usage for client-side JavaScript when applications are built with developer tools such as Browserify (*browserify.org*). The manifest file is `package.json` and NPM (along with Browserify) provides support for deep-dependency tree. This means multiple versions of the same package are supported.
- Component.io (*netm.ag/component-258*) takes a slightly different approach, based on thinking about packages as Web Components (*webcomponents.org*) that may contain JS, HTML, CSS and so on. JavaScript should be defined and included using a CommonJS syntax. The manifest used is `component.json` and dependencies can be managed using the `component-set` command (*github.com/jkroso/component-set*).
- JSPM (*jspm.io*) is an entirely browser-focused solution tied to the System.JS module loading system (*github.com/systemjs/systemjs*), which is a Universal Module Loader that means any module type can be loaded. It uses `package.json` and dependencies can be referenced and retrieved from JSPM or NPM.

Packages Above are some of the most popular client-side package managers

▶ Finally, we need to add some JavaScript to `capture.html` to connect to Socket.IO.

```
<script src="/bower_components/socket.io-client/socket.io.js"></script>
<script>
  var socket = io( 'localhost:3000' );
  socket.on( 'connect', function() {
    console.log( 'connected' );
  } );
</script>
```

To achieve this we've included the Socket.IO client library, created a new socket connection using `io( 'localhost:3000' )` and bound to the `connect` event so we know when the server connection has been established. Reload `capture.html` and open up the JavaScript console to check for the `connected` log message. Look at the prompt you used to execute `node index.js` and you'll see `We have a connection!`.

CAPTURING DATA

With the basics in place we can now start capturing data from the client. We'll use `capture.html` as a way of developing and testing this functionality.

The information we're going to capture, and ultimately display, is as follows: that a connection has been established, what page URL the user is viewing and a couple of the user's browser capabilities such as Touch and HTML5 Video support.

```
<script src="/bower_components/socket.io-client/socket.io.js"></script>
<script src="/bower_components/modernizer/modernizr.js"></script>
<script>
  var socket = io( 'localhost:3000/capture' );
  socket.on( 'connect', function() {
    var data = {
      url: window.location.href,
      touch: Modernizr.touch,
      video: Modernizr.video
    };
    socket.emit( 'client-data', data );
```

```
2.0.0-p353 in socketalytics/ on master
› node index.js
listening on *:3000
{ url: 'http://localhost:3000/capture.html',
  webgl: true,
  touch: false,
  video: true,
  websocket: true }
```

Analytics Data captured on the client, sent to server and logged to console

```
  } );
</script>
```

The URL we're connecting to is `localhost:3000/capture` where `/capture` is an example of a Socket.IO namespace (*netm.ag/names-258*). Namespaces are handy for partitioning data – here, they identify a communication channel for capturing data.

In the `connect` event callback we create a `data` object upon which we store all the information our app wants; the page URL is stored on `data.url` and Modernizr is used when capturing values for `data.touch` and `data.video`. Finally, the information is sent to the server using `socket.emit( 'client-data', data )`.

We're now sending all the data we want to capture to the server. Let's update the server code in `index.js` to prove it's getting that data:

```
var io = require( 'socket.io' )( server );
var capture = io.of( '/capture' );
capture.on( 'connection', function( socket ) {
 socket.on( 'client-data', function( data ) {
  console.log( data );
 } );
} );
```

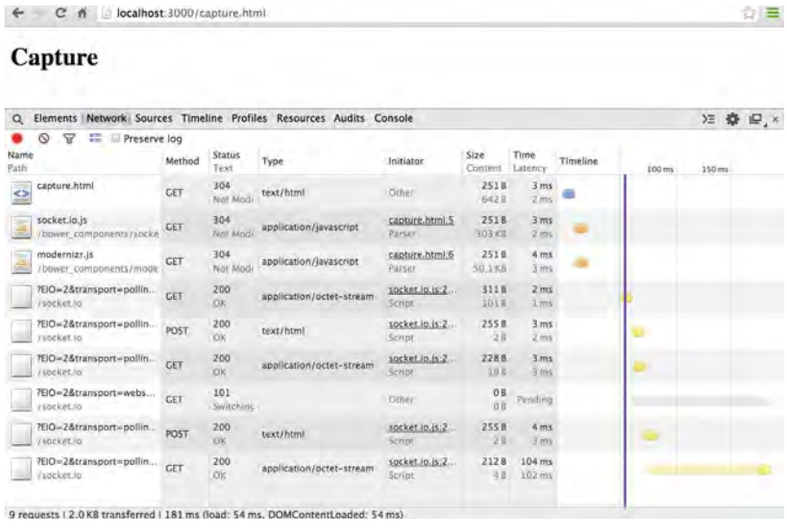
With the basics in place we can now start capturing data from the client

You'll notice that the server code has also been updated to now reference the `/capture` namespace. It's important that we use a namespace or we'll also count connections to our dashboard in the stats when we create it. We bind to the `client-data` event, which mirrors the event we emit on the client. If you restart the node process and refresh `capture.html` you'll see the information that's been captured is logged to the command prompt.

STORING STATS DATA

On the server we want to keep track of connections and associated data so we can calculate stats.

```
var socketData = {};
var stats = { connections: 0, touch: 0, video: 0, pages: {} };
var capture = io.of( '/capture' );
capture.on( 'connection', function( socket ) {
 ++stats.connections;
 socket.on( 'client-data', function( data ) {
  socketData[ socket.id ] = data;
  stats.touch += ( data.touch? 1 : 0 );
  stats.video += ( data.video? 1 : 0 );
  var pageCount = stats.pages[ data.url ] || 0;
  stats.pages[ data.url ] = ++pageCount;
 } );
} );
```



Inside action Opening up the Network Tab in Google Chrome Developer Tools provides interesting information about the connectivity work Socket.IO is doing under the hood

First, the connection count is incremented when a user connects to the `/capture` namespace. The `socketData` variable is used to map a socket to its associated stats data using a unique `socket.id`. The storage takes place in the `client-data` event callback. `touch` and `video` values are incremented if supported by the client. The number of connections on each page is stored by URL in the `stats.pages` property.

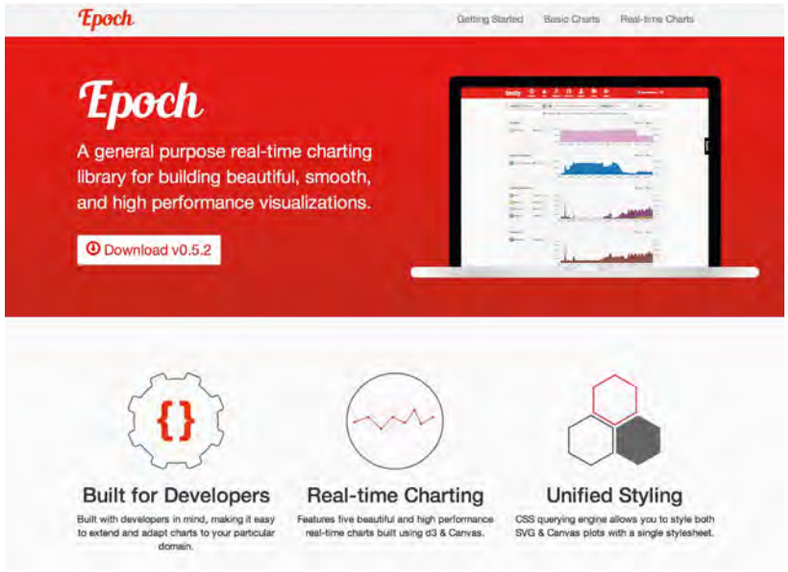
We also need to update the stats as connections are lost. Within the `connection` callback bind to the `disconnect` event on the `socket` object:

```
  stats.pages[ data.url ] = ++pageCount;
 } );
 socket.on( 'disconnect', function() {
  --stats.connections;
  stats.touch -= ( socketData[ socket.id ].touch? 1 : 0 );
  stats.video -= ( socketData[ socket.id ].video? 1 : 0 );
  --stats.pages[ socketData[ socket.id ].url ];
  delete socketData[ socket.id ];
 } );
```

In the same way that we incremented values in the `connection` callback we decrement values in the `disconnect` callback.

PUBLISHING THE DATA

Now that all the data is stored, we want to publish it so that a dashboard can consume and display the stats. Since all the data is already stored in the `stats` ▶



Epoch A general purpose real-time charting library for building smooth, high performance visualisations

object, we just need to publish it. For this, we'll use a `/dashboard` namespace.

```
var stats = { connections: 0, touch: 0, video: 0, pages: {} };

var dashboard = io.of( '/dashboard' );
dashboard.on( 'connection', function( socket ) {
 socket.emit( 'stats-updated', stats );
} );
```

The `/dashboard` namespace is referenced and when a new connection is detected the current stats are emitted to the newly connected socket only.

We also need to trigger an event to all dashboard clients whenever the stats update – for example, after a `client-data` event is received and processed, or when a `/capture` connection triggers the `disconnect` event. To do that, simply emit a `stats-updated` event on the dashboard namespace at the end of the appropriate callbacks:

```
dashboard.emit( 'stats-updated', stats );
```

REAL-TIME ANALYTICS DASHBOARD

The only thing that's left now is to build the dashboard. If you open `public/dashboard/index.html` you'll see it includes the Epoch charts library, its stylesheet and dependencies.

Within that file you'll also find various element placeholders with clearly named `id` attributes for our real-time analytics charts: `visitors` area chart for connection count, `pages` bar chart for popular pages, touch and video gauges for `touch` support and `video` element support respectively.

A file `js/dashboard.js` is also referenced from `index.html`. Open this up and initialise the Epoch charts.

```
var visitors = $('#visitors').epoch( {
  type: 'time.area', axes: ['left', 'bottom', 'right'],
  data: [ { values: [ { time: Date.now()/1000, y: 0 } ] } ],
} );
var pages = $( '#pages' ).epoch( { type: 'bar' } );
var touch = $( '#touch' ).epoch( { type: 'time.gauge' } );
var video = $( '#video' ).epoch( { type: 'time.gauge' } );
```

The server is already publishing the stats as soon as it has them, so in order to receive that data we need to subscribe to the `/dashboard` namespace and bind to the `stats-updated` event.

FINISHING OFF

Finally, we need to extract the data we want from the `update` and tell our charts to update.

```
var video = $( '#video' ).epoch( { type: 'time.gauge' } );
var dashboard = io( 'localhost:3000/dashboard' );
dashboard.on( 'stats-updated', function( update ) {
  visitors.push( [ { time: Date.now()/1000, y: update.
connections } ] );
  var pagesData = [];
  for( var url in update.pages ) {
   pagesData.push( { x: url, y: update.pages[ url ] } );
  }
  pages.update( [ { values: pagesData } ] );
  touch.update( update.touch / update.connections || 0 );
  video.update( update.video / update.connections || 0 );
} );
```

To update the `visitors` chart we reference `update.connections` and update the visitors chart. We loop over `update.pages` and format the data where `x` is the page URL and `y` is the number of connections on that page. The `touch` and `video` gauges are updated with a value between 0 and 1 based on the ratio of stat versus connections.

With everything in place you can now restart the node process and load `http://localhost:3000/dashboard/index.html` to see the dashboard. Open `http://localhost:3000/capture.html` in one or more tabs to start capturing data and watch the captured data instantly displayed in the dashboard.

CONCLUSION

We've only scratched the surface of what can be achieved with this application and with the new tools and features in Socket.IO 1.0. But hopefully this tutorial has given you a taster of just how powerful and easy it can be to use real-time technologies in your web apps.

VIDEO

Phil Leggetter has put together an exclusive screencast to go alongside this tutorial. Take a look at *netm.ag/realtime-258*

2ND EDITION
Updated for 2014

iPad for Business

Ditch the laptop, the iPad is *all* you need!

97 great apps to help you work smarter

Available at all good newsagents or visit

www.**myfavouritemagazines**.co.uk/computer

Download the files here!

All the files you need for this tutorial can be found at *netm.ag/audio-250*

ABOUT THE AUTHOR

STUART MEMO

w: *stuartmemo.com*

t: @stuartmemo

areas of expertise: Web audio, JavaScript, open source pun-based projects: Qwerty Hancock, Wavey Jones, Abbey Load

q: what's the worst present you've ever been given?

a: My brother gave me a pair of personalised socks with the name 'Ian' sewn into them

WEB AUDIO

RETHINK SOUND ON THE WEB

Stuart Memo invites us to reapproach the use of audio on the web and presents five considerations when introducing audio to a web application

Sound on the web. I know. Historically, these two things haven't gone together very well. Most of us have been burned when visiting a site whose creator has deemed, most probably as an afterthought, that a background bed of awful electro-jazz would be an excellent way to enhance the mood. Fortunately, it doesn't have to be this way. We can do things differently.

When people talk about native versus web apps, audio never gets a mention. Yet it is this element that helps the UI of many native apps excel over their web counterparts. Take the camera app on a phone, for example. How do we know when a photo has been taken? You'll most likely see some sort of flash on screen to indicate that an image has been captured, but it's usually backed-up with the sound of an emulated mechanical click of a camera's shutter. What about Twitter's famous "pull to refresh" control? Would it be as satisfying if it didn't have that little 'pop' when you let go?

So why think about this now? What's changed? Well for one thing, we don't have to rely on Flash anymore. With the arrival of the `<audio>` tag and with more and more browsers implementing the Web Audio API (*w3.org/TR/webaudio*), we can produce sound natively in the browser with zero lag and with more control than we've ever had before. We don't even have to use audio files if we don't want to. The Web Audio API allows us the create sound on the client, much like we do when creating images from scratch by painting to the `canvas` element, but more on that later.

Say we want to notify a user when the upload of their photo to our website is complete. We may want to provide some audio feedback by playing a sound. While this is technically fairly straightforward to implement, it could be detrimental to the user experience if it's not properly thought through. In order to provide some guidance, I've created a list of five important things to consider when bringing audio to your web app.

1 IS IT NECESSARY?

There's no need to be popping and bleeping for every hovered link or with every CSS transition. An action or result should only be accompanied by a sound if it greatly reinforces or clarifies an important message to the user. Notifying the user of something that requires their attention? Probably. Clicking links? Definitely not.

2 ENVIRONMENT

Perhaps one of the reasons that native mobile apps tend to embrace sound much more is that it becomes necessary to provide additional feedback in an environment that's more likely to be noisy and filled with visual distractions. Is your web app more likely to be used on the move, or in a quiet office

BROWSER SUPPORT WEB AUDIO API

Desktop	Mobile/tablet
31	No
25	No
No	No
17	30
7	25

where such sounds could be distracting for others? If you're considering adding audio to enhance a mobile experience, consider the pitch of your audio. Bassy sounds, for example, won't cut through background noise, making it difficult to hear.

3 IS IT ANNOYING?

Audio comes with the added burden of being potentially extremely irritating. Sometimes it's the sound itself, other times it's down to it being repeated. Sometimes it's both (listen to Samsung's recent notification whistle: *netm.ag/whistle-250*) Luckily, we can reduce the chance of this happening by doing a few things:

- Make the sound short. Less than a second is preferable, half a second is even better.
- Make sure the notes that make up your sound are harmonic, and that the timbre is pleasant.
- Only repeat the sound if absolutely necessary.

Regarding repetition, chat applications such as those that are part of Facebook and Google+ use sound for notifying the user that someone has sent them a message. This makes sense if a user has

Audio comes with the added burden of being potentially extremely irritating

Facebook open in a different tab than the one they're currently working in, as it is a clear indicator that something requiring attention is waiting. However, if they're currently replying to said chat, what benefit does the notification bring?

4 PERFORMANCE

Depending on how long it is, a snippet of audio could have a file size larger than most images on your page. Luckily, this won't affect anything visually, or block any functionality. So, instead of downloading audio files alongside your HTML and images, consider loading it in the background after the page has loaded, while the user is happily interacting with your page.

Better still, consider using the Web Audio API where you can. The Web Audio API is a high-level way of manipulating sound in the browser using JavaScript. While it's not currently implemented in all major browsers, it's definitely worth thinking about using it along with the audio element as a fallback. The API allows you to create sound in the browser from scratch without using pre-existing audio files. While its ins and outs are outside the scope of this article, you can see how I created the `success-demo.html` notification sound (available to download in the tutorial files) in JavaScript if you're using Chrome, Safari or Firefox.

5 CONTEXT AND IDENTITY

Audio created to enhance the user experience of a fun site aimed at children won't be appropriate for a finance web app. Imagine the mood of the person who is using your app and ensure you choose your tone appropriately.

Sound can also become as integral a part of a brand as its visual identity. The Mac startup sound (*netm.ag/startup-250*), the Nokia ringtone (*netm.ag/nokiaring-250*), and Intel's Inside chime (*netm.ag/chime-250*) are examples where if we were to hear these sounds out of context, we'd still know the companies involved. While it's not necessary to shove your unique motif down the user's ear canals at every opportunity, consider creating a suite of sounds that reflect the visual identity of your site. Facebook recently took this idea a step further; its incoming call sound is made up of four musical notes: F, A, C, and E (*netm.ag/ping-250*).

CODA

If done well, audio can enhance and enrich even the dullest of web apps. Forget the autoplay background music, focus on how it can improve the overall experience and enjoyment of your site. Admittedly, it's not always going to be appropriate. As well as annoying your users, you have to watch out for annoying those physically near your user. The iPhone user who hasn't turned off the sound of the on-screen clacking keyboard comes to mind. Adding audio should be done carefully with a light touch.

Now that the Web Audio API is supported in Chrome, Safari, Firefox and Opera, we now have complete control of sound without the downloading of weighty assets. The power of the API shouldn't be ignored either. Dynamically applying audio effects such as reverb, or cross-fading two separate pieces of music transforms the current capabilities of HTML5 games (*netm.ag/games-250*), making the entire experience much more immersive. You can even create musical instruments in browsers that challenge their native rivals. But, I would say that. See my drum machine made with the Web Audio API: *beatpetite.com*.

Sounds can be as effective at providing feedback to your users as any visual cue. It's time we reapproach it with the wariness experience has instilled in us.

RESOURCE

WEB AUDIO WEEKLY

Due to there being a lot of excitement online regarding web audio, there's a lot of interesting projects and demos regularly being published. Chris Lowis highlights the best in his regular newsletter: *tinyletter.com/webaudioweekly*

VIDEO

Watch an exclusive screencast of this tutorial created by Tuts+ Premium: *netm.ag/tut2-250*

Download the files here!

All the files you need for this tutorial can be found at *netm.ag/lazy-258*

ABOUT THE AUTHOR

ERIC MANN

w: *10up.com*

t: @EricMann

areas of expertise:
HTML, CSS, PHP, JavaScript, WordPress

q: what's the most heroic thing you've ever done?

a: On a backpacking expedition as a Scoutmaster, I tracked down a lost boy and his mother. They were trying to cross a canyon in the dark, with no equipment

RESOURCE

AUTHOR'S BLOG

Follow Eric Mann's blog (*eamann.com*) for daily tips on site optimisation, tutorials and discussions of new web design tools.

WEB PERFORMANCE

OPTIMISE PERFORMANCE WITH LAZY LOADING

Eric Mann explains how lazy loading enables you to include rich graphical media in a website, without sacrificing page performance

The continued growth of the web has led developers and content producers to cram as much into web pages as they can. High-resolution video, interactive advertisements, high-density graphics, rich visitor analytics – these can all be found on a regular web page. As consumer bandwidth and desktop capabilities increase, so does the footprint of any particular website.

At the same time, visitors are beginning to browse the web over mobile connections. This migration of site traffic from high to lower bandwidth is in direct opposition to the producers' goals of including more in their sites. New solutions are needed to combat and resolve this problem.

Lazy loading – only loading embedded assets such as high-resolution graphics and video when absolutely necessary – is one of the most straightforward solutions available to tackle this issue. It takes a little planning and finesse on the part of the development team, but can make a whole world of difference to a site's load time and overall performance.

Anything that loads outside of the main browser viewport can be loaded lazily. However, instead of optimising all the things, let's focus on two pieces of low-hanging fruit: images and video.

LOADING IMAGES

A highly engaging site is rich with graphical content. News articles and blog posts in particular are easier to consume when paired with illustrative images. Unfortunately, embedded graphics must be processed in line with the rest of the page. Higher-resolution images will load slowly and detract from the visitor experience.

Instead, we can selectively load only the images visible in the device viewport. To do so, we change our markup to replace the `src` attributes of our standard `<img />` tags with a `data-lazy` attribute. The image's source reference then points instead to a small placeholder image.

```
<img src="img/placeholder.png" data-lazy="img/horses.hidpi.png" />
```

When the page loads, the browser will only download the placeholder image. This should be a very simple image of the fewest colours possible.

Hidden values A given website's markup might only be 20KB, but the page's images can easily total 1MB or more

The placeholder will be cached by the browser and reused, resulting in a single HTTP request to build out imagery throughout the page.

JavaScript routine

We will build a JavaScript routine into the bottom of our page to process all of the lazy-loaded images and only present the ones we want to see.

The first part of our JavaScript-based lazy loading routine is a function that can determine whether or not a given image element is in the viewport. We

Higher-resolution images will load slowly and detract from the visitor experience

will include the 300 pixels immediately above and below the viewport as a loading threshold so any user scroll events will reveal already-loaded images rather than our placeholder.

```
function inView( image ) {
  var $image = $( image ),
    view_top = $window.scrollTop() - 300,
    view_bottom = view_top + $window.height() + 600,
    height = $image.height(),
    _top = $image.offset().top,
    _bottom = _top + height;
  return height > 0 && _top <= view_bottom && _bottom
>= view_top;
}
```

If the page loads from the top, any images within 300 pixels of the bottom of the viewport will be loaded automatically. If the page loads somewhere in the middle – like when a visitor visits a URL with ▶

FOCUS ON

TECHCRUNCH

When TechCrunch began investigating a redesign in early 2013, two objectives were clear: increasing the visual engagement of the site with a deeper use of images, and making the site mobile-friendly.

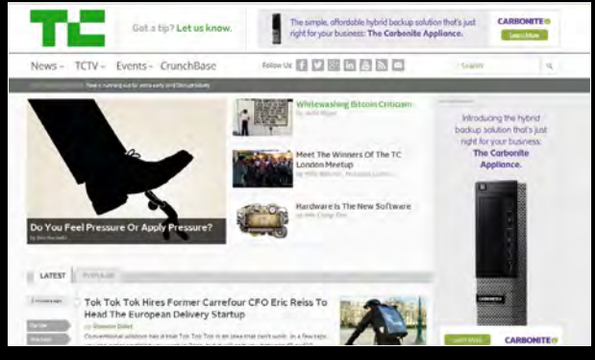
The new site design incorporates a responsive layout that is optimised both for tablet and mobile experiences. Additionally, a new editorial push requires images for every article. Feature-length articles also include a full-width, high-resolution image both on individual article pages and in the excerpt that is presented on the homepage.

It soon became apparent, however, that including more images on the page competed with the other major objective during the redesign: speed. Every one of those new article images added a significant amount of data for the browser to download. The full-width images on feature length articles were initially so large they left gaping holes in the still-rendering page.

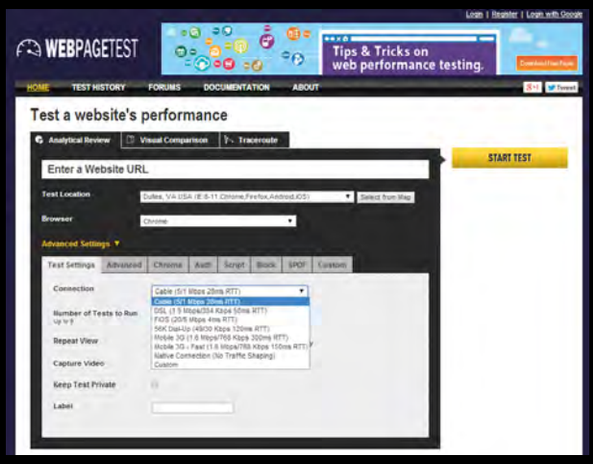
Considering our secondary goal of enhanced page speed – the development team was in a bind. Without any optimisations, the homepage took more than 18 seconds to load in a browser with a clear cache. Our launch goal, on the other hand, was just five seconds to load from a clear cache (the goal for repeat visits was a three-second load time).

We began optimising the site by rewriting image includes to leverage the same lazy loading techniques covered by this tutorial. Further optimisations included lazy-loaded video and even on-site advertising. Social sharing icons, which can be heavy since they include scripts from third-party servers, are only loaded when the visitor actually attempts to share an article.

Lazily loading as many assets as possible helped TechCrunch achieve its targeted load times from day one.



Speeding up The new TechCrunch.com features an image-rich, responsive layout that makes clever use of lazy loading



RESOURCES

TOOLS AND TRICKS

While the easiest way to test the impact of lazy loading on low-speed, low-power devices is to actually use such a device, not everyone has access to a complete device lab to do so. Instead, there are a few electronic tools you can use to grab performance data and simulate devices and connection speeds without spending a fortune on hardware.

Pingdom Network Tools

Pingdom (*tools.pingdom.com*) does more than ping sites and test them for availability. Its Website Speed Test will load your site in an emulator from any of three regions and track how long every resource takes to load.

Pingdom also provides rich analysis tools for finding any apparent bottlenecks in your site. A unified performance rating (scored out of a possible 100) is drilled down into performance recommendations.

WebPagetest

Testing from a variety of browsers in different regions over a range of connection types becomes hugely important when digging into page performance. The WebPagetest tools (*webpagetest.org*) allow you to simulate everything from the newest Chrome on a high-speed connection in Virginia, to IE8 on a 56K dial-up connection in São Paulo, Brazil.

You can see just how long it takes both new and repeat visitors to load and navigate your site. Seeing just how degraded a site can appear on a low-capability system drives home the importance of page optimisation. WebPagetest can help identify quick candidates for speeding up any site.

▶ a hash – images within 300 pixels of either the top or bottom of the viewport will load.

Visible images

Next, we will define the function that actually loads visible images. This function will grab the `data-lazy` attribute and use that to replace each `<img/>` tag's `src` attribute. The browser automatically detects the changes to the DOM, downloads the new assets, and replaces our placeholders in the page.

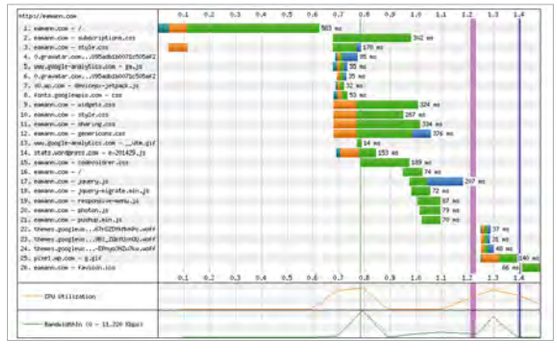
```
function maybeLoad( images ) {
  images.each( function( i, image ) {
    if ( image.hasAttribute( 'data-lazy' ) && inView( image
) ) {
      image.src = image.getAttribute( 'data-lazy' );
      image.removeAttribute( 'data-lazy' );
      image.trigger( 'lazy-load' );
    }
  } );
}
```

To improve the `maybeLoad()` function, we can automatically update the `images` collection to remove newly loaded images from the array. This reduces both the application's memory footprint and the number of elements through which we need to iterate each time.

Loading threshold

As visitors scroll the page, we should also re-scan for images that have moved within our loading threshold. As the window's scroll event fires several times while the page is still scrolling, the event handler is throttled to prevent overflowing simpler devices' memories or overtaxing their processors.

```
$( window ).on( 'scroll', function() {
 if ( undefined === throttle_id ) {
  return;
 }
```



Waiting game Lazily loading assets moves the purple 'DOM Loaded' bar to the left and makes the site responsive in significantly less time

```
 throttle_id = window.setTimeout(
   function() {
     maybeLoad( images );
     throttle_id = undefined;
   },
   250
 );
} );
```

The event above will only fire once every 250 milliseconds, allowing us to continuously scan the page as the browser scrolls. Reducing the throttle speed will ensure even fast scrolling will never reveal an unloaded image. However, it might negatively impact slow scrolling by demanding more system resources to manage the event.

LOADING VIDEO

Video assets can also drag down a page's performance significantly. When a video is embedded in the page using a script tag, the browser processes the script tag immediately, usually loads an iFrame in its place, and then begins downloading a large Flash (or Silverlight or MP4)

We can set it up so the site waits for the visitor to actually click on the video before it loads

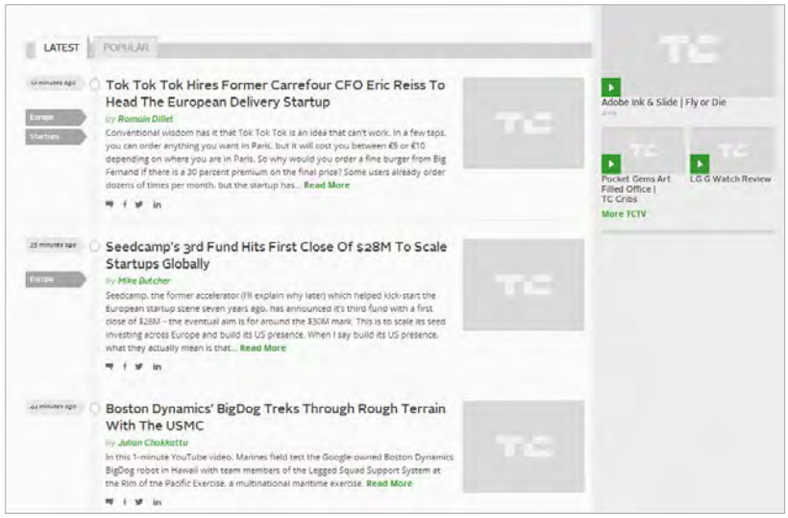
resource. This can lock the browser's interface and result in a visitor bouncing off your site to another. One technique to avoid this scenario is to force videos to display in a modal overlay – a lightbox – instead of in-line with the rest of your page content.

Two elements

Instead of a script embed from, say, YouTube, your page's markup will consist of two elements: a span used to present a play button, and the video's thumbnail (optimised using the lazy loading technique for images covered above):

```
<span class="iframe_play" data-embed="https://www.youtube.com/watch?v=Xz91KWAe6t0&autoplay=1"></span>
<img class="iframe_thumb" src="http://img.youtube.com/vi/Xz91KWAe6t0/hqdefault.jpg" />
```

Then, we can set it up so that the site waits until the visitor actually clicks on the video before loading the video itself. We do this by listening for mouse events on the video play button (the `iframe_play` element) and invoking a custom event handler.



Quick fix With the exception of the homepage feature island, all images on TechCrunch.com are lazy-loaded, including video thumbnails in the sidebar

```
$( document ).on( 'click', '.iframe_play', play_video );
```

On this user action, you have two options: Swap the placeholder thumbnail for the video player itself, or create a lightbox overlay to display the video atop the page. Our `play_video()` function will employ the lightbox technique, automatically generating and launching a modal window that plays our video.

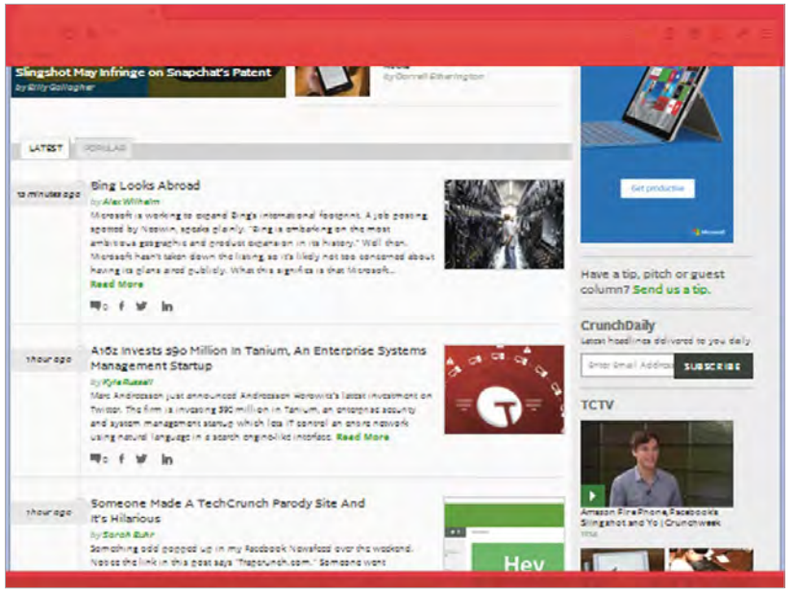
When the lightbox closes, it's removed from the page entirely, keeping the page's markup clean and lightweight.

```
function close_video() {
 $modal.remove();
 $overlay.remove();
};
```

Markup

Our tool dynamically generates the markup for both a modal overlay and a container for the video. In the case of YouTube, it's using an iFrame embed to build out the video element. Since the autoplay flag is set in the URL, the video will start immediately in the modal. Clicking on the close link will remove the generated markup, immediately stopping playback when the video is removed from the DOM.

```
function play_video( e ) {
  var $this = $( this ),
      video = this.getAttribute( 'data-embed' ),
      content = document.createElement( 'iframe' ),
      overlay = document.createElement( 'div' ),
      modal = document.createElement( 'div' ),
      closer = document.createElement( 'a' ),
```



Left A loading threshold will force our application to load only images currently in the viewport or the red highlighted areas

Below Displaying only a thumbnail and play control on videos embedded in HTML5 presentations allows the entire presentation to load quickly


```
    $overlay = $( overlay ), $modal = $( modal ), $closer = $( closer );
    content.src = video;
    overlay.style.cssText = 'position: fixed; top: 0; left: 0; width: 100%; height: 100%; background-color: #000;z-index: 159900;';
    $overlay.on( 'click', close_video );
    modal.style.cssText = 'position: fixed; top: 30px; left: 30px; right: 30px; bottom: 30px; background-color: #fff; z-index: 160000;';
    closer.innerText = 'close';
    $closer.on( 'click', function( e ) { e.preventDefault(); close_video(); } );
    modal.appendChild( closer );
    modal.appendChild( content );
    document.body.appendChild( overlay );
    document.body.appendChild( modal );
}
```

Loading a video on-demand in a modal window provides all of the interaction of a standard embedded video. It just foregoes the overhead of loading the video's media stream when the page first displays.

FURTHER APPLICATIONS

Lazy loading web content allows a site to straddle the middle ground between providing a rich user experience and providing a speedy one. It's a clever use of engineering savvy to cut corners where possible without actually cutting down on the content displayed on the site. Thus far we've covered images and video, but just about any other content can be pulled in after the initial page load.

Sharing buttons and analytics

Sites like TechCrunch defer loading social media interactions (like Facebook, Twitter and LinkedIn buttons) until after the visitor hovers their mouse over the sharing region. With 20 articles on the homepage, this would usually mean loading 60 somewhat heavy remote includes. Waiting to load the social sharing icons until absolutely necessary is a considerable boost to site performance.

Lazy loading allows sites to balance a rich user experience and a speedy one

Other sites load analytics scripts, markup for standalone modules and even advertising asynchronously. Virtually any content that can be loaded separately from the page's main markup and assets, should be.

THE BEST OF BOTH WORLDS

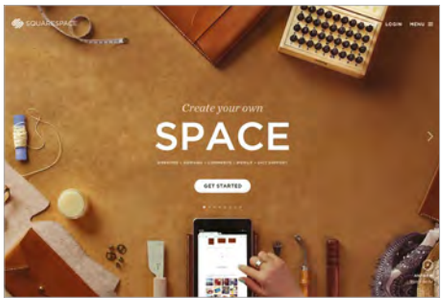
Websites will continue to grow in scope and scale in order to present well-rounded experiences to those with capable hardware and seemingly-unlimited bandwidth. Presenting a similarly rich experience to those with limited access and less high-performing devices, though, doesn't need to be a daunting task. Tricks like lazily loaded media help combine the best of both worlds.

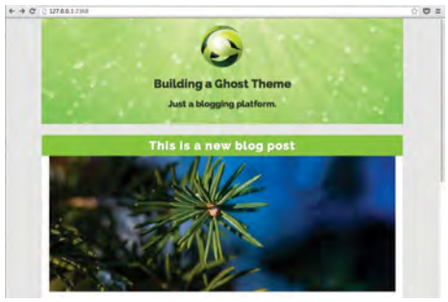
TOP TIP

GET TESTING

Make sure your testing is as thorough as your audience is diverse. Test from multiple devices, geographic locations and network types. What works for one visitor might fail for another.

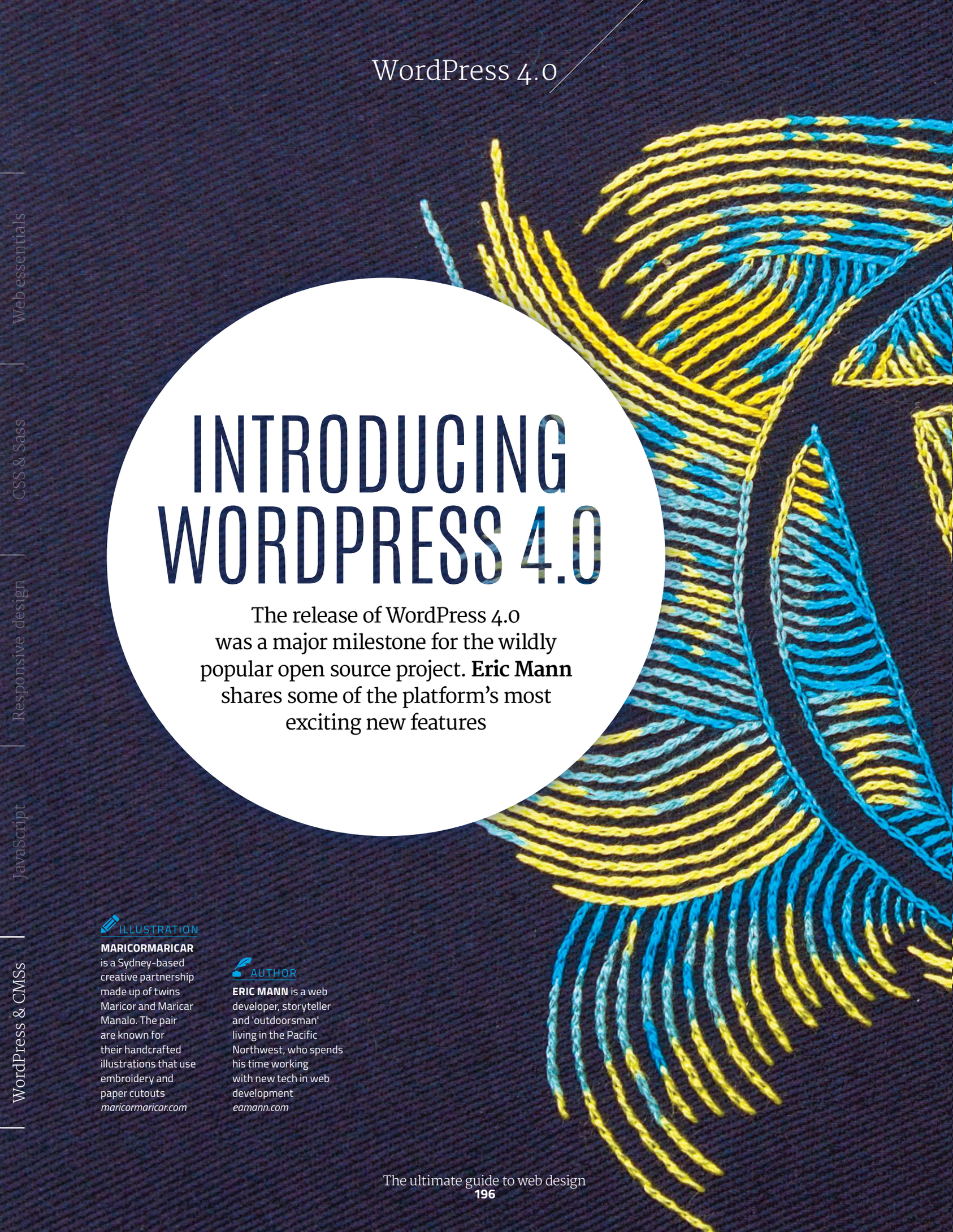
WORDPRESS & CMSs





INTRODUCING WORDPRESS 4.0

The release of WordPress 4.0 was a major milestone for the wildly popular open source project. **Eric Mann** shares some of the platform's most exciting new features



ILLUSTRATION

MARICORMARICAR is a Sydney-based creative partnership made up of twins Maricor and Maricar Manalo. The pair are known for their handcrafted illustrations that use embroidery and paper cutouts
maricormaricar.com

AUTHOR

ERIC MANN is a web developer, storyteller and 'outdoorsman' living in the Pacific Northwest, who spends his time working with new tech in web development
eamann.com

Back in 2007 I started writing a blog on WordPress. I've been working professionally with the platform, and contributing code, since 2010. In this time, I've seen 17 major releases come and go. Still, nothing excited me more, as a writer and as a developer, than the new feature set for WordPress 4.0.

The core team announced the start of WordPress 4.0's development at the end of April 2014. The 23rd release in its history, WordPress 4.0 was heralded as a landmark '.0' release. It was led by a talented, UX-focused engineer and backed by hundreds of savvy developers across the globe. The initial slate of feature proposals was enough to excite every WordPress fan, and watching the update being narrowed down to a set of well-coded, well-tested, well-vetted features was incredible.

As with any new release, there are features all users love and those we're less than excited about. For the first time I can remember, every one of version 4.0's features is something to talk about. I have, though, whittled down my list of "I can't believe we finally have this" to what I believe are the most important new features.

1 INSTALL IN YOUR LANGUAGE

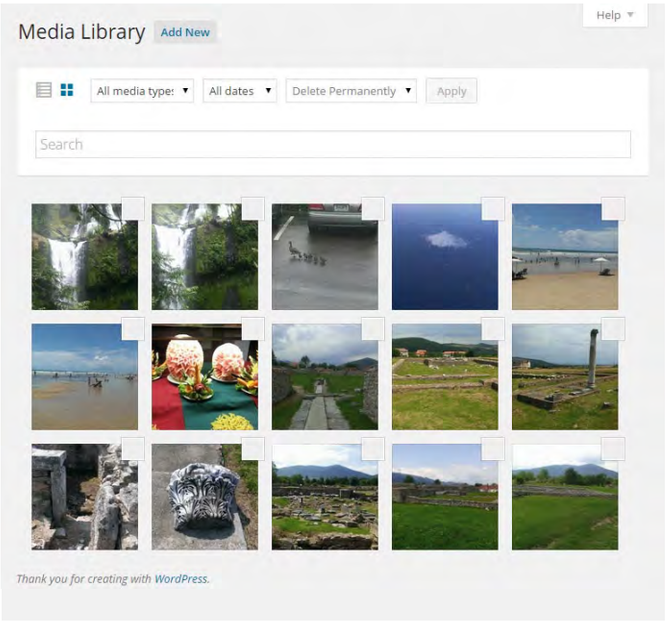
One of the overarching goals of the WordPress platform is to "democratise publishing". Features ship in every version to make it easier for writers to put their voice online under their own terms. As the majority of the world communicates in languages other than English, this means the WordPress admin needs to present content in a non-English format.

There is a large contingent of volunteers that works tirelessly to translate text in WordPress to various languages. Many themes and plugins also tie in to WordPress' translation features, making it possible for non-English speakers to use the entire platform, including optional extensions.

Up until 4.0, though, the installation and setup process for WordPress was English-only. New users either had to use English throughout the installation, or find someone to walk them through the process. As you can imagine, this was a major barrier to entry for non-native English speakers.

> Nothing excited me more than the new feature set for WordPress 4.0

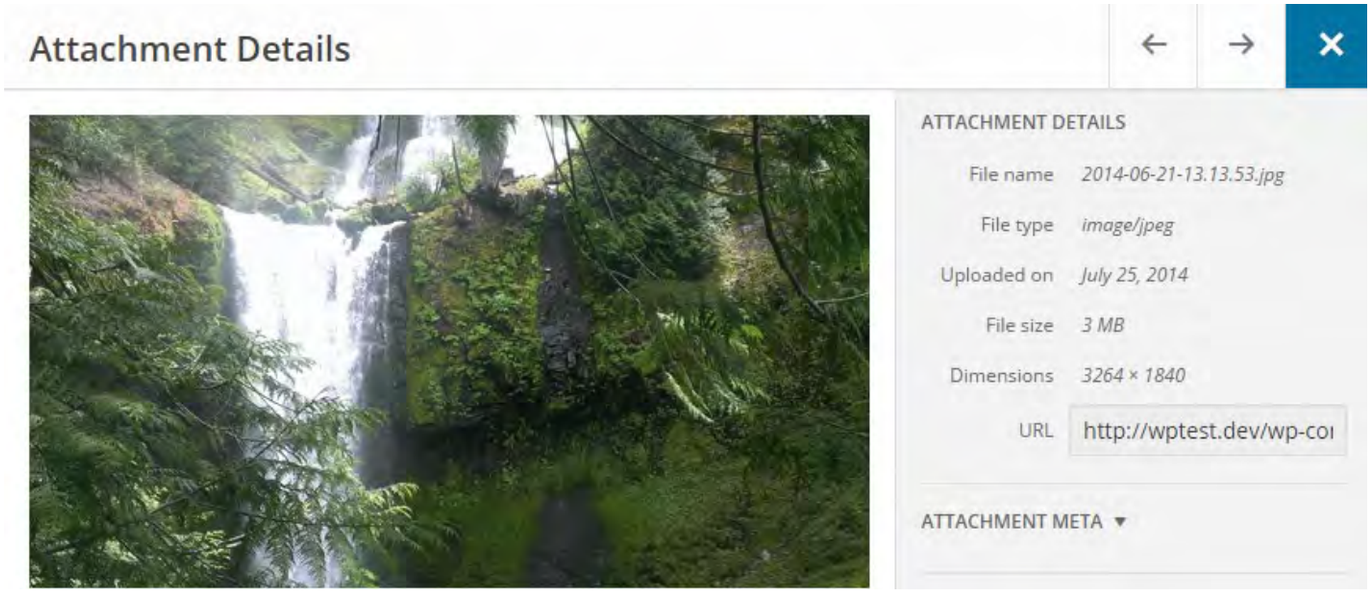
WordPress 4.0 changed this. During initial setup and installation, the platform now allows users to specify their primary language. Selecting a language other than English triggers the download of a language pack, automatically converting instructions throughout the rest of the installation screens – and the WordPress admin itself. The core WordPress community recognises the value of our growing international audience. For the first time, non-native English speakers are able to install and configure WordPress without needing to hire an expensive contractor to set things up for them.



Below left The first screen presented by WordPress 4.0's installer allows for language selection, automatically translating the rest of the installer

Below right The new Media Library grid view includes larger thumbnails, making it easier to manage assets in the admin

Far right The Media Library's grid view displays media details in a convenient modal window



These changes have made WordPress more accessible to more users than ever before. As May of this year marked the first time non-English downloads of the platform exceeded English downloads, we can only expect further international adoption for WordPress from this point forward.

2 MEDIA MANAGEMENT

WordPress 3.5 introduced a completely reimagined media gallery that made it far easier to both import and work with images inside of posts and in the admin itself. WordPress continues to optimise the media experience in version 4.0 by introducing a new media grid to the gallery.

In the past, editors only had one option for viewing items in the Media Library – the list view. This presented a paginated list of items, each with relatively small thumbnails. Managing media on a large site meant navigating a somewhat boring list of images spanning several pages. If you've been using WordPress for a while, you'll know the pre-4.0 media list view was a less than ideal experience.

The updated Media Library enables the new grid view by default. This view presents a rich grid of larger image thumbnails that pulls in subsequent pages of content using the same approaches we employ for infinite scroll. Clicking any particular thumbnail loads the attachment's meta information in a modal window. This further empowers rapid editorial changes without the need to navigate within WordPress to different pages.

Without this meta information modal, managing individual assets in bulk was tedious and time-consuming. Even viewing asset captions and

INSIDER'S VIEW: Helen Hou-Sandi

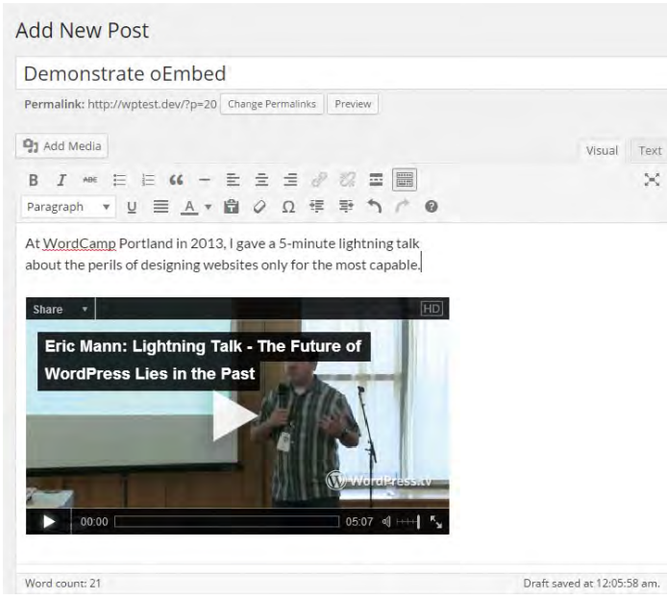
After the second beta release, I had the opportunity to sit down with Helen Hou-Sandi, the release lead for WordPress 4.0, to learn a little how she'd seen the project evolve over the past few months. Past versions of WordPress often had themes for the release to guide the overall development of the project. Version 4.0, however, had less of a central theme.

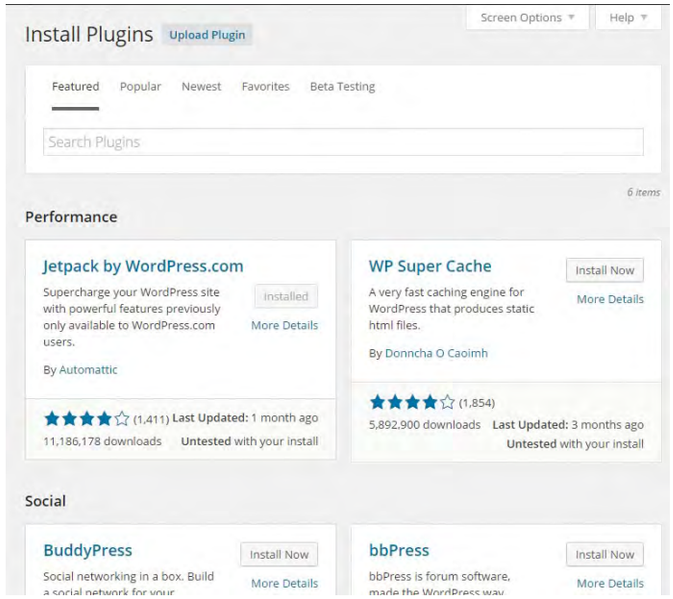
"We've been better at iterating on what we've already got," Hou-Sandi explains. As a result, most features scheduled for the release build on past successes, or focus on refreshing elements.

An example of the latter is WordPress' plugin interface. According to Hou-Sandi, plugins are "central to what WordPress can do", but the interface has remained relatively untouched since 2009. She explains that one of the biggest focuses of 4.0 has been to improve the experience for new users, and that revamping the way plugins are both discovered and managed within WordPress has been high on the development team's list.

Ultimately, Hou-Sandi says, the objective of 4.0 has been to "do small things continually on all fronts" rather than focusing exclusively on any one area. She hopes the final product will benefit multiple groups – making WordPress simpler for newer users while also extending powerful new APIs for developers.

WordPress is very much a living project, making continued progress on various fronts with each new release. The goal for 4.0 is the same as every version: to make WordPress better.





HOW TO GET INVOLVED

WordPress has a gruelling release cycle, targeting a new version every four months. Getting stellar new features into WordPress core on such a limited timetable takes a lot of work, and requires a lot of community involvement.

Want to get involved in the development of code for WordPress? The best way to get started is by following the core development blogs:

- WordPress Core (*make.wordpress.org/core*)
- WordPress UI Group (*make.wordpress.org/ui*)
- WordPress Accessibility Group (*make.wordpress.org/accessibility*)

Even non-developers can contribute to ongoing development. Features need to be tested, bugs need to be reported and systems need to be documented. You can help build the next version of WordPress, even if you never write a single line of code.

Want to help keep WordPress rolling? Start by contributing in some of the following places:

- WordPress Support Forum (*wordpress.org/support*)
- WordPress Code Reference (*developer.wordpress.org/reference*)
- Get involved with your local Meetup group (*wordpress.meetup.com*)
- Attend or volunteer at a WordCamp near you (*central.wordcamp.org*)

descriptions forced a separate page load. The refined experience presents all attachment details in an easy-to-access location that saves time and effort.

Even non-image attachments like documents and Zip archives benefit from the new grid treatment. While they lack individual thumbnails, presenting MIME-type icons in the same format as the larger image thumbnails definitely makes the Media Library as a whole feel more unified.

3 EDITOR UPGRADES

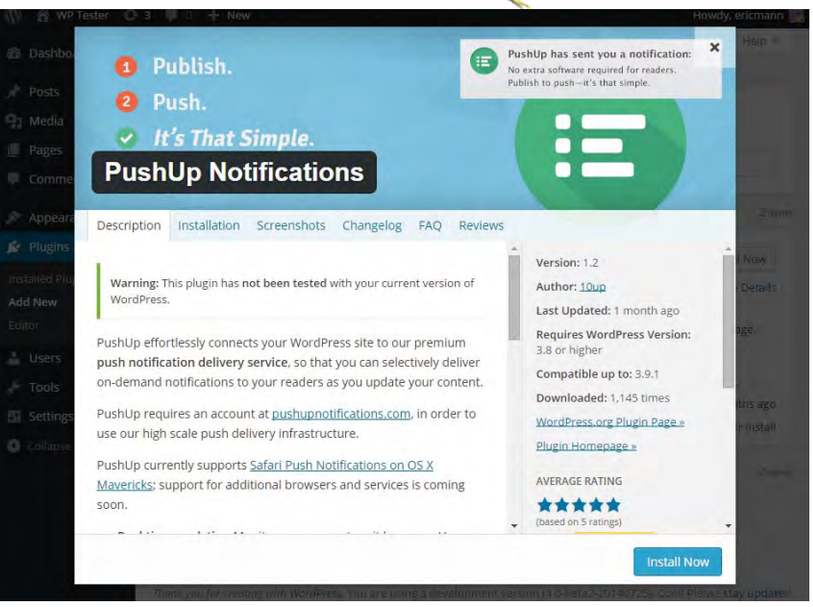
The most-used feature of WordPress by far is the WYSIWYG content editor. Whether you use WordPress for casual blogging, professional news media, or merely to curate a static body of content, you'll at some point have used the editor itself.

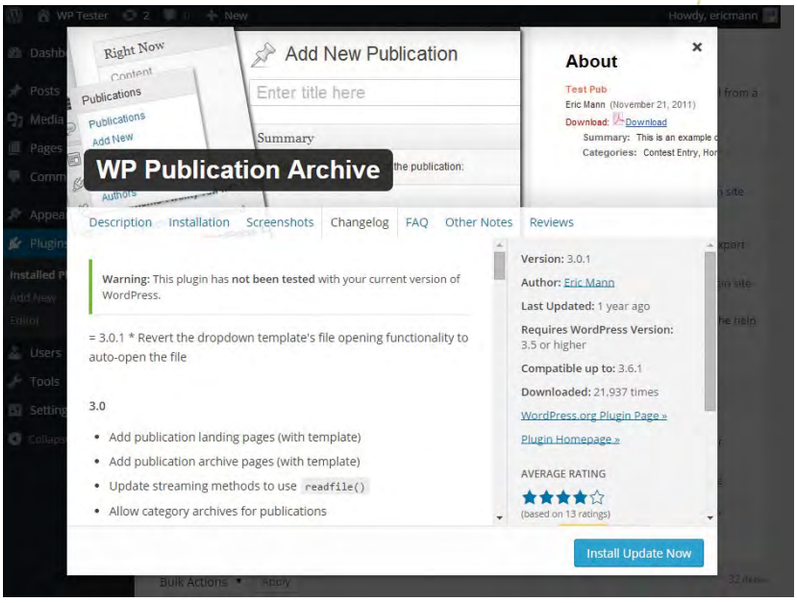
As the most recognisable element of the platform, you'd expect the editor to receive a fair amount of polish. WordPress 4.0 brings several updates to the editorial experience and the editor itself. There are two exciting changes to keep an eye on.

oEmbed previews

WordPress has long been a champion of using oEmbed to easily share content from one site or platform to another. The 4.0 release iterated on existing support to add new oEmbed providers for sites like *ted.com* and *collegehumor.com*. It also drastically evolved the way we experience oEmbed within the content editor.

In the past, editors could merely copy and paste the URL of an oEmbed-read resource (like a YouTube video or Twitter message) directly into the post editor. When viewed on the frontend, this raw





Opposite left When oEmbed-capable links are present within post content, WordPress will generate a preview of the linked resource

Opposite right The new plugin installation page presents more streamlined discovery tools – starting with better categorisation of available plugins

Above left The details modals for plugins display richer information than in previously – including the plugins' graphical banners

Above right When plugins are ready for upgrading, a More Details link exposes the plugin's changelog, ratings and compatibility information

URL would be dynamically converted to a richer representation of the linked material. The feature was a huge value-add for editors linking to external content, but also made it a bit tricky to optimise body content around an embedded resource.

WordPress 4.0 changed this feature up by enabling live previews of the embedded content. Once the link is pasted into the editor, WordPress automatically scans the link, fetches the associated resource through oEmbed, creates a representative preview and displays that preview in the editor.

> The top toolbar will pin itself to the top of the page even as your content flows on

It's an actual implementation of WYSIWYG for the WordPress editor that eliminates the pain of using oEmbed to extend post content. The original URL for the embedded asset will always be available on the Text view; but once you've seen the inline preview, you'll never want to leave the Visual editor.

Sticky toolbar

My favourite new feature in WordPress isn't so much an addition as a refinement of existing behaviour. In WordPress 4.0, the toolbars for the visual editor – both at the top and the bottom of the content editor – stick to the browser window when you scroll the page. As you write and your article grows longer, the content area automatically expands to contain every word. The top toolbar – with rich text formatting options and media embed features – pins itself to the top of the page as your content flows on. No more scrolling in the window to change paragraph formatting or insert links.

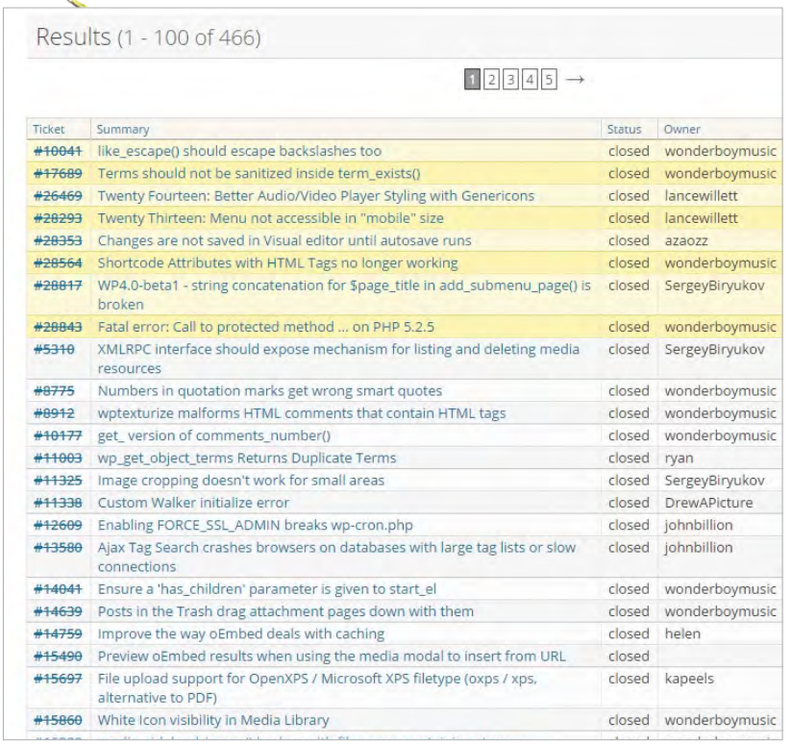
While scrolling back to the top of the article, the footer of the content editor pins itself to the bottom of the browser window. Whether you're editing content in the beginning of the post, the end of the post or somewhere in-between, you retain access to both your formatting tools and after-the-post meta information like word counts.

This is a somewhat minor change in the editorial experience that has major benefits for the writers of long-form content. Once you use the new 'sticky' editor, you'll question why it took this long for the feature to evolve.

4 ENHANCED PLUGIN EXPERIENCE

I started my career with WordPress as a plugin developer, so the end-user plugin experience has always been a key point of interest for me. Plugins are central to making WordPress function as a customisable platform, and as a contractor I spent a large number of hours teaching clients how to work with plugins. Unfortunately, the plugin screens in WordPress have not been the most user-friendly in the past. Worse still, they haven't received much more than a new paint job in the past five years.

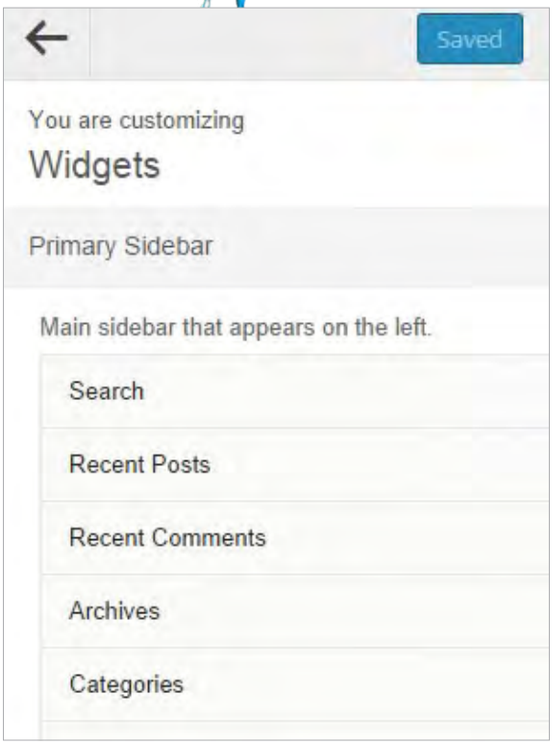
WordPress 4.0 introduced a number of changes to the way new users experience plugins, both in terms of discovery and management. The plugin installer used to attempt to aid feature discovery with a basic ▶



Results (1 - 100 of 466)

1 2 3 4 5 →

Ticket	Summary	Status	Owner
~~#10041~~	like_escape() should escape backslashes too	closed	wonderboymusic
~~#17689~~	Terms should not be sanitized inside term_exists()	closed	wonderboymusic
~~#26469~~	Twenty Fourteen: Better Audio/Video Player Styling with Genericons	closed	lancewillett
~~#28293~~	Twenty Thirteen: Menu not accessible in "mobile" size	closed	lancewillett
~~#28353~~	Changes are not saved in Visual editor until autosave runs	closed	azaozz
~~#28564~~	Shortcode Attributes with HTML Tags no longer working	closed	wonderboymusic
~~#28817~~	WP4.0-beta1 - string concatenation for $page_title in add_submenu_page() is broken	closed	SergeyBiryukov
~~#28843~~	Fatal error: Call to protected method ... on PHP 5.2.5	closed	wonderboymusic
~~#5310~~	XMLRPC interface should expose mechanism for listing and deleting media resources	closed	SergeyBiryukov
~~#8775~~	Numbers in quotation marks get wrong smart quotes	closed	wonderboymusic
~~#8912~~	wptexturize malforms HTML comments that contain HTML tags	closed	wonderboymusic
~~#10177~~	get_ version of comments_number()	closed	wonderboymusic
~~#11003~~	wp_get_object_terms Returns Duplicate Terms	closed	ryan
~~#11325~~	Image cropping doesn't work for small areas	closed	SergeyBiryukov
~~#11338~~	Custom Walker initialize error	closed	DrewAPicture
~~#12609~~	Enabling FORCE_SSL_ADMIN breaks wp-cron.php	closed	johnbillion
~~#13580~~	Ajax Tag Search crashes browsers on databases with large tag lists or slow connections	closed	johnbillion
~~#14041~~	Ensure a 'has_children' parameter is given to start_el	closed	wonderboymusic
~~#14639~~	Posts in the Trash drag attachment pages down with them	closed	wonderboymusic
~~#14759~~	Improve the way oEmbed deals with caching	closed	helen
~~#15490~~	Preview oEmbed results when using the media modal to insert from URL	closed	
~~#15697~~	File upload support for OpenXPS / Microsoft XPS filetype (oxps / xps, alternative to PDF)	closed	kapeels
~~#15860~~	White Icon visibility in Media Library	closed	wonderboymusic



search field and essentially meaningless tag cloud. Unless you already knew what to look for, the chances of finding something new for your site were limited. The new installer highlights 'featured', 'popular' and 'new' plugins, with the featured plugins further broken down into categories like 'performance' and 'social'.

On the installation pages, plugins are now displayed in detailed tiles presenting their title, a brief description and the plugin author, as before. These tiles also highlight the number of downloads and average plugin rating. A clever More Details link launches an overlay displaying the plugin's full information as pulled from *WordPress.org* – including the plugin's graphical banner.

This details screen is also presented when plugin updates are available, making the plugin's changelog, ratings and compatibility information readily accessible to administrators.

5 CUSTOMIZER IMPROVEMENTS

The Theme Customizer underwent a strong set of changes with 4.0, and has been rebranded as just the 'Customizer'. According to the WordPress 4.0 Customizer Improvements infomation (*netm.ag/improvements-259*):

"'Customize' could refer to anything. That's the point ... the Customizer can be used for anything, and we'd like to encourage more experimentation with different uses of the Customizer."

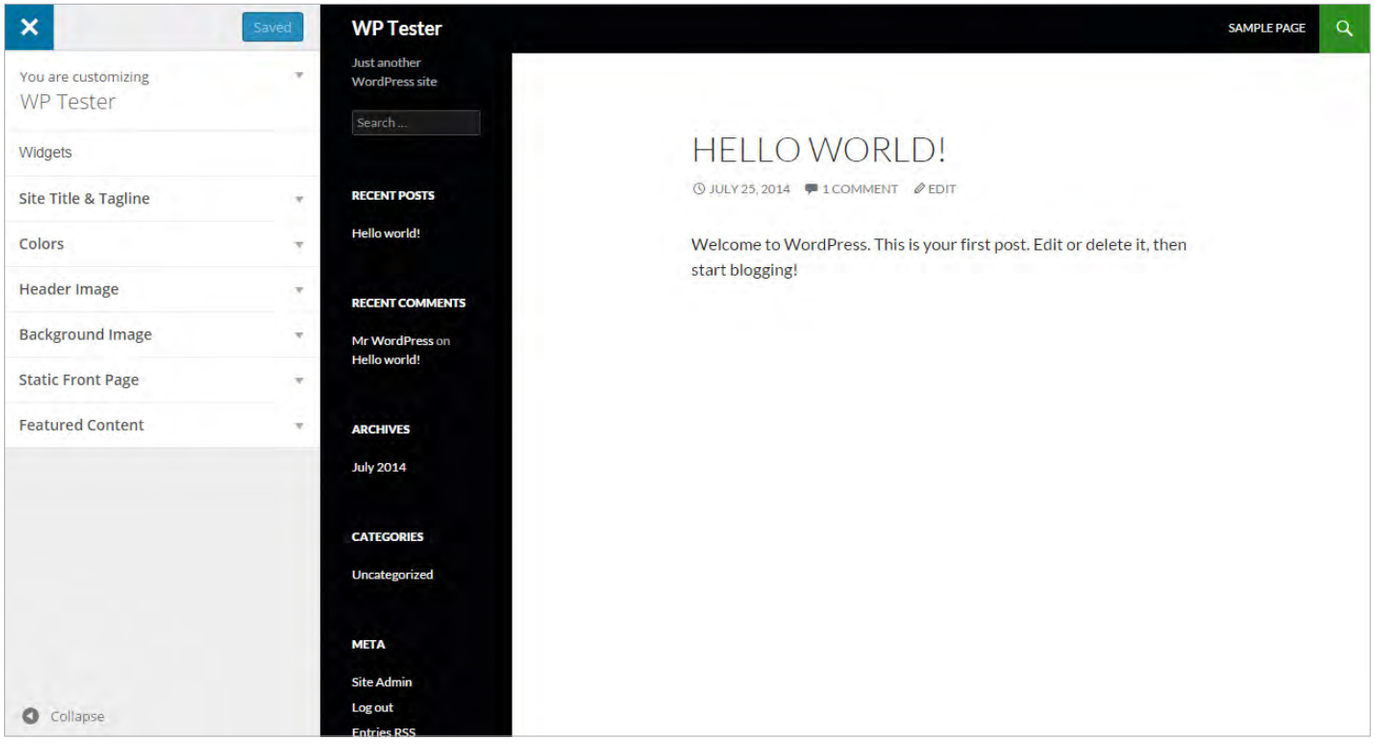
Widgets panel

Widget support was added to the Customizer in the previous version of WordPress, allowing site owners to see live previews of changes to sidebar widgets as they were made. Expanding the scope of the Customizer means we'll be adding more to the tool in coming releases, so widgets have earned their own Panel within the Customizer UI.

Panels are a way to group Customizer sections, allowing both cognitive and visual separation between different elements that are meant to be managed separately. Site administrators can dig into their theme to customise just the colours, banner imagery and content layout on one panel. Then they can delve deeper to manage the content and arrangements of widgets on another. Widgets are just the first of a promising series of changes to the Customizer system.

Developer API

While it seems like a minor refinement to the Customizer tool itself, the addition of a Panels API opens the door for developers to make future themes even more flexible. Every site owner wants a unique design, but not everyone has the time to build a custom theme – or the budget to hire a developer.



Far left As with all new versions, WordPress 4.0 improves upon the past by fixing hundreds of minor user-reported issues

Middle The Customizer's new Widgets panel allows for quick and easy management of sidebars and widgets – complete with live previews

Left The rebranded and redesigned Customizer allows theme developers to control every aspect of the frontend of their site

Flexible themes that support the Customizer are the easiest way for the majority of non-technical users to reach into their WordPress site and truly make it their own. Editors – and writers – can modify the design of their homepage, About page, content archives, and even more though the Customizer. The Customizer API adds visual finesse to the existing theme options API, empowering developers to truly enable customers to lift the hood on their site.

> Every new feature speaks to the power and ingenuity of the WordPress community

Being able to nest certain features under one or more Panels makes it easy to separate one type of customisation from others. For example, global theme features can live on one screen, contextual features that only present data to logged-in viewers can live on another.

The changes introduced by WordPress 4.0 mean the only limitation on what the Customizer can do is the imagination of the theme developer. Thankfully, theme designers tend to be on the creative side.

BOTTOM LINE

Most development communities place a lot of weight on '.0' releases. WordPress 2.0 saw a completely redesigned backend that introduced Ajax for a snappier administrative interface, and the WYSIWYG post editor we still use today. WordPress 3.0 saw the merger of the WordPress and WordPress MU (multi-user) code bases to present an improved platform capable of hosting multiple sites on one installation.

Some have argued that WordPress 4.0 isn't as significant a change as its '.0' predecessors. In truth, the core WordPress community never intended for any of its '.0' releases to be landslide events in the first place. Instead, WordPress focuses on small, rapid iterations of a large collection of features, releasing a new version every four months to maintain forward momentum on development.

This release happened to coincide with the number 4 – but that version number doesn't require any special requirements on the feature set. However, given the features outlined in this article, and the countless minor fixes and optimisations under the hood, WordPress 4.0 proved to be no smaller a milestone than any '.0' that's come before.

Every feature in this release speaks to the power and ingenuity of the community that maintains WordPress. These features also demonstrate just how much momentum WordPress will continue to carry into future versions to come.

Download the files here!

All the files you need for this tutorial can be found at *netm.ag/wordpress-254*

ABOUT THE AUTHOR

JOE CASABONA

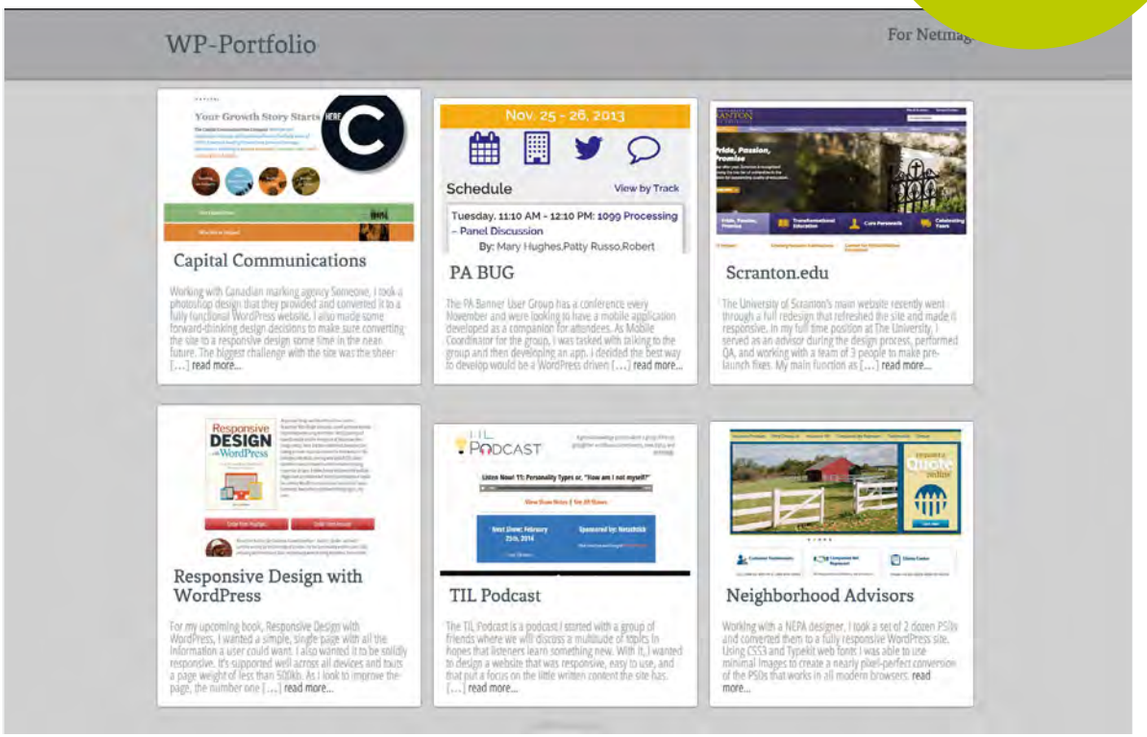
w: *casabona.org*

t: @jcasabona

areas of expertise:
HTML, CSS, PHP, MySQL, WordPress

q: Which cartoon character do you most identify with?

a: Leo from *TMNT* – I used to be him when my three brothers and I played together



RWD

BUILD A RESPONSIVE WORDPRESS PORTFOLIO

Joe Casabona explains how to build a responsive portfolio, by creating a WordPress plugin for a Custom Post Type, and a new theme template

Web development may change rapidly, but two things that are here to stay are WordPress and responsive design. Knowing how to build responsive WordPress themes and plugins is a must. In this tutorial, we will look at building a WordPress plugin and theme template for a responsive portfolio.

We will work with two mocked-up templates: an archive page, which will list all of the recent projects, and a single page, which will show a specific project. The archive page for the portfolio is a pretty simple one with a header and three columns of projects at full width. This will shrink to two columns, then one column, as the screen gets smaller. The HTML and CSS is available at GitHub: *netm.ag/wordpress-254*. Each project on the page will have this structure.

This is the HTML that will be generated by the WordPress Loop:

```
<div class="card">
    <img src="project.png">
        <h3>Name of Site</h3>
        <p>Short description and a link <a href="#">read more...</a></p>
</div>
```

The single page is going to have a similar layout, wrapping all of the text in a container called `.project` instead of `.card`. The CSS is also going to be fairly lightweight and scaled. You may also notice in the site files that there is a `style.scss` file. I've

RESOURCE

READ THE BOOK

The issues raised in this tutorial are discussed at greater length in Joe Casabona's new book, *Responsive Design with WordPress*: *rwdwp.com*

developed all of the CSS using Sass, but fear not: the generated `style.css` is compiled in a readable way.

CREATING A NEW CUSTOM POST TYPE

A Custom Post Type is an object in WordPress that allows us to add any types of content we want to the WordPress editor, treating them the same way as posts. In a fresh WordPress install, there are menu options for Posts and Pages. These are both considered post types that handle content differently. With Custom Post Types, we can add options for creating new types of content to the WordPress admin menu. We'll create a Custom Post Type called Portfolio.

We're going to develop this Custom Post Type as part of a bigger WordPress plugin for portfolio projects. While we could add the functionality to the theme, this is bad practice because then our content is tied to our design: if we change the theme, we lose the portfolio. We will handle display through two

RWD is here to stay. Knowing how to build responsive WordPress plugins is a must

methods: templates/template tags, and shortcodes that can be used through the editor.

The first step is to define the plugin. Create a folder in `/wp-content/plugins/` and name it whatever you like. I've named mine `/jlc-projects/`. Inside that folder, create a file of the same name (for example, `jlc-projects.php`) and add this code:

```
<?php
/*
Plugin Name: Joe's Portfolio Plugin
Plugin URI: https://github.com/jcasabona/wp-portfolio
Description: A simple plugin that creates and display a
projects portfolio with WordPress using custom post types!
Author: Joe Casabona
Version: 1.0
Author URI: http://www.casabona.org
*/
define('JLC_PATH', WP_PLUGIN_URL . '/' . plugin_basename(
dirname(__FILE__) ) . '/' );
define('JLC_NAME', "Joe's Portfolio Plugin");
require_once('jlc-project-cpt.php');
?>
```

There are a few things going on here. The first is the standard plugin definition for a WordPress plugin; the next few lines create constants and then include the ▶

FOCUS ON

LEARNING SASS

I just recently got into Sass (*sass-lang.com*) and I'm a little upset I didn't get into it earlier. Sass, which stands for Syntactically Awesome Style Sheets, is a CSS pre-processor that lets you write CSS in a way similar to programming languages. You can define variables, make comments hidden from the output file, write functions and extensible CSS classes, perform mathematical operations, and more.

As someone who primarily considers himself a programmer, I find this method of writing very appealing. I can abstract away a lot of what I would normally repeat. Plus, as far as RWD goes, Sass makes it much easier to keep all of the media queries for one class together. For example:

```
.card{
    /* Default Styles Here */
    @media screen and (min-width: $bp-medium){
        display: inline-block;
        width: 40%;
        }
    @media screen and (min-width: $bp-large){
        width: 44%;
        }
    }
```

No matter what size your project is, I would recommend using Sass for the CSS. It will make managing your style sheets a lot easier, and make responsive sites better, through improved organisation and by compiling multiple resources into one.

If you want to learn Sass, the best resource out there is *Sass for Web Designers* by Dan Cederholm (*netm.ag/Sass-250*).

Book apart Dan Cederholm's *Sass for Web Designers* is an excellent Sass primer

FOCUS ON

RWD AND WORDPRESS

This article only scratches the surface of responsive design using WordPress. There are a lot of other factors to keep in mind. Remember that developing responsive websites is about more than just the site shrinking on small screens. You should also ensure that you're using best practices by:

- **Using ems for breakpoints**

These are much more accessible for users who may have different browser settings and handle the wide variety of screen resolutions much better.

- **Using content-based breakpoints**

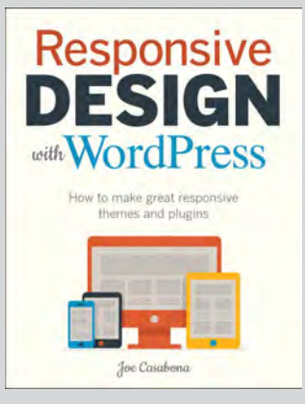
Make sure you're developing in such a way that your breakpoints are determined by the content of your website and not the device's resolution. That way you know your content looks good universally, not just on three specific devices.

- **Keeping speed and size in mind**

In a perfect world, everyone would be on Wi-fi or 4G and not have to worry about data caps. Unfortunately, that isn't the case. As we develop websites, we need to remember to load only what's needed, reduce page weight and remove extra HTTP requests. This will make things much easier on the user.

Tools like Sass and WordPress can help us achieve these goals, especially the final one. In my recent book, *Responsive Design with WordPress* (*rwdwp.com*) I go over best practices for responsive design, why they are important and how to implement them in WordPress, as well as pitfalls to avoid.

I also include several tutorials like this one in the last chapter, taking you through developing forms, photo galleries, and even a products page responsively in WordPress.



Further reading Joe Casabona's book complements and extends this tutorial

files the rest of the plugin needs. At this point, there is only one other file: `jlc-project-cpt.php`.

You will also notice that I'm using the prefix JLC_ (or jlc-) for everything. You should choose your own prefix to use. Prefixing variables and function names will decrease the chance of your plugin conflicting with other themes or plugins.

Before we jump into `jlc-project-cpt.php`, I want to add one more bit of code to `jlc-projects.php`. The code below will create a new image size, which we will use with our Custom Post Type:

```
if ( function_exists( 'add_theme_support' ) ) {
    add_theme_support( 'post-thumbnails' );
    add_image_size('jlc_project', 1100, 640, true);
}
```

Now it's time to create `jlc-project-cpt.php`. I'll only be discussing the important code here, but you can find the complete code on the GitHub repo. First (after the opening `<?php` tag) we define the Custom Post Type:

```
add_action('init', 'jlc_projects_register');
function jlc_projects_register() {
    $args = array(
        'label' => __('Portfolio'),
        'singular_label' => __('Project'),
        'public' => true,
        'show_ui' => true,
        'capability_type' => 'post',
        'hierarchical' => true,
        'has_archive' => true,
        'supports' => array('title', 'editor', 'thumbnail') ,
        'rewrite' => array('slug' => 'portfolio', 'with_front'
        => false)
        );
    register_post_type( 'portfolio' , $args );
    register_taxonomy("jlc-project-type", array("portfolio"),
    array("hierarchical" => true, "label" => "Project Type",
    "singular_label" => "Project Type", "rewrite" => true));
}
```

This is your standard Custom Post Types definition function. We add an action to call it on `init`, then

New UI The admin page to add a new project. Notice the Project Link section

send our list of arguments to `register_post_type()`, along with the type's slug, which will be 'portfolio'. After registering the post type, we register the custom taxonomy to go along with the post type. It's important to keep these two functions together. If you don't, and the taxonomy somehow gets registered first, WordPress will throw an error.

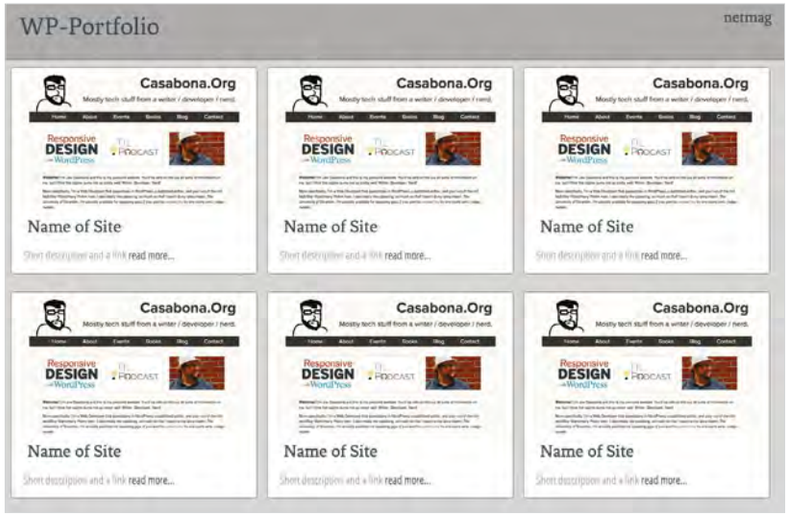
After the Custom Post Type is defined, it's time to add the custom metadata we want to use. Our Custom Post Type supports a title, the editor (which will serve as the body text), and a thumbnail, which is where the featured image will go. There is one more thing I like to add to my portfolio pieces: a URL to the website I'm showcasing. First, we'll create the function that will add this box in the admin:

```
add_action("admin_init", "jlc_projects_admin_init");
function jlc_projects_admin_init(){
    add_meta_box("jlc-projects-meta", __("Project Link"),
    "jlc_projects_options", "portfolio", "side", "low");
}
function jlc_projects_options(){
        global $post;
        if ( defined('DOING_AUTOSAVE') && DOING_
        AUTOSAVE ) return $post_id;
        $custom = get_post_custom($post->ID);
        $link = $custom["jlc_projects_link"][0];
?>
  <input name="jlc_projects_link" placeholder="http://"
  value="<?php echo $link; ?>" />
<?php
    }
```

These functions are fairly straightforward. When the admin is initiated (that is, loaded), we'll call a function called `jlc_projects_admin_init()` that will create a new meta box for portfolio items. In order to generate that box, a new function, `jlc_projects_options()`, is called.

Once inside the `options` function, we simply grab the `link` value, which we've called `jlc_projects_link`, and print it out. But first, we want to make sure an autosave isn't being performed. If it is, we will probably lose data. After that, we need to actually save the metadata when the post is saved:

```
add_action('save_post', 'jlc_projects_save');
function jlc_projects_save(){
    global $post;
    if ( defined('DOING_AUTOSAVE') && DOING_AUTOSAVE ){
        return $post_id;
    }else{
        update_post_meta($post->ID, "jlc_projects_link",
        $_POST["jlc_projects_link"]);
    }
}
```



Basic grid The HTML template used here, at full width. It's a simple header with three columns of projects

With the admin section for our Custom Post Types created, it's time to add some frontend functionality to help display our projects to visitors. This consists of an archive template, a single page template and a shortcode (not covered in this tutorial). But before we do that, there is one other function we're going to create for displaying images: picturefill.js .

This piece of JavaScript (you can find the GitHub repo at *rwdwp.com/23*) allows you to define a set of media queries to switch an image to a version friendlier to the size of the screen it is being viewed on. This also has implications for load time, since you can probably assume that a smaller screen means a mobile device using 4G, 3G, or even EDGE. I know that isn't always the case, but it's better than nothing.

You can see the markup pattern for a standard picturefill element on the GitHub repo. We can have an unlimited number of `<span>` elements for each size of the image we have. There is also a fallback for users without JavaScript. As you can imagine, since WordPress creates multiple versions of every image we upload using the Media Uploader, it lends itself nicely to picturefill.js. The first thing we should do is load the script, which is located in the `/js/` folder in our plugin's directory. We add the following code to `jlc-projects.php`:

```
function jlc_projects_scripts() {
    wp_enqueue_script( 'picturefill', JLCP_PATH.'js/
    picturefill.js', array());
}
add_action( 'wp_enqueue_scripts', 'jlc_projects_scripts' );
```

This will load our JavaScript with the scripts being loaded by other plugins. It will also ensure that we aren't loading picturefill.js more than once. ▶

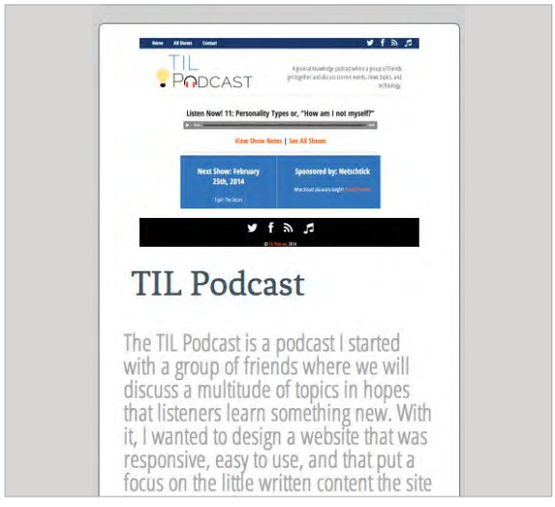
RESOURCE

MORE ABOUT PICTUREFILL

Scott Jehl's original blog post about picturefill.js discusses the need for the polyfill and includes a link to a demo site: *netm.ag/picturefill-254*

Since our projects will be using the featured image section to display the screenshot, we can replace the default featured image markup by using the `post_thumbnail_html` filter. Note that this function will actually replace all featured image sections on the site. If this could cause a conflict (it probably will), you should add some conditionals to your plugin to make sure this filter is only being used on portfolio pages.

```
add_filter( 'post_thumbnail_html', 'jlc_projects_get_
featured_image');
function jlc_projects_get_featured_image($html,
$aid=false){
    $sizes= array(''thumbnail', 'medium', 'large', 'jlc_project',
    'full');
        $img= '<span data-picture data-alt="'.get_the_
        title().'">';
        $ct= 0;
        $aid= (!$aid) ? get_post_thumbnail_id() : $aid;
        foreach($sizes as $size){
            $url= wp_get_attachment_image_src($aid,
            $size);
            $width= ($ct < sizeof($sizes)-1) ? ($url[1]*0.66) :
            ($width/0.66)+25;
            $img.= '
                <span data-src="'. $url[0] .'"';
            $img.= ($ct > 0) ? ' data-media="(min-width: '.
            $width .'px)"></span>' :'></span>';
            $ct++;
        }
        $url= wp_get_attachment_image_src( $aid,
        $sizes[1]);
    $img.=  '<noscript>
        <img src="'.$url[0] .'" alt="'.get_the_title().'">
                </noscript>
            </span>';
        return $img;
}
```



Mobile view A portion of the archive template displayed on a mobile-sized screen. The cards shrink to single column with centered images

There are a few things going on here. The first is that the function has an array of all the image sizes in WordPress that we want to use. If you have your own sizes defined, you will have to add them here. This is so the picturefill element is accurately populated. After some set up (defining the image sizes, opening the picturefill element, initialising a counter), it moves through the `$sizes`, printing an image entry for each.

For each entry, `wp_get_attachment_image_src()` is called to grab the URL of the image based on the image's ID (which `get_post_thumbnail_id()` returns based on the post ID) and the size. `wp_get_attachment_image_src()` returns an array that includes the image, the width, the height, and whether or not it's cropped. There is also a bit of maths going on here to calculate when to determine the breakpoints, as well as how to handle the thumbnail image. I'm not going to discuss this in detail here, but it's worth noting that this is an important problem to solve, and one for which solutions are evolving rapidly.

Now any time we get the post's thumbnail, the HTML returned will be from our function.

If your plugin includes CSS, keep it minimal so it doesn't butt heads with the main theme

CREATING THE ARCHIVE PAGE

Next, we will create the archive template for the new Custom Post Type. This is what will be displayed by default and will serve as our main portfolio page.

(Note: we will not be creating the site's homepage in this tutorial, but doing so would require either a template tag or shortcode that will execute a custom Loop using `WP_Query`.)

Create a new file in whatever theme directory you are using and call it `archive-portfolio.php`. WordPress's template hierarchy is smart enough to know that, when a user is at the portfolio page, it should display the content using this template. My recommendation at this point is to copy the `page.php` template for this template. We will simply replace the Loop portion.

I recommend that you use a template without a sidebar, or a single-column template. The CSS referenced here will work a bit more nicely. Here's what our Loop looks like:

```
<?php while (have_posts()) : the_post(); ?>
    <div class="card">
        <?php the_post_thumbnail('jlc_project'); ?>
        <h3><?php the_title(); ?></h3>
```

```
        <p><?php echo get_the_excerpt(); ?> <a href="<?php
        the_permalink(); ?>">read more...</a></p>
    </div>
<?php endwhile; ?>
```

This should be pretty straightforward. Because we are replacing the default HTML for `the_post_thumbnail()`, the argument of which image to use doesn't matter because all sizes will be returned using picturefill.js markup. I opted to use `get_the_excerpt()` in order to exclude any markup included by `the_excerpt()`.

When designing a plugin that includes some CSS, it's important to make it as minimal as possible so that it doesn't butt heads with the theme's CSS or give the user the ability to exclude your CSS completely. Since we're creating templates within the theme, we have a little more wiggle room. Here's a portion of the (Sass-generated) CSS that I've added to each project on the archive page:

```
.card img {
    display: block;
    margin: 0 auto; }
@media screen and (min-width: 45.88em) {
    .card {
        display: inline-block;
        width: 40%; } }
@media screen and (min-width: 54.62em) {
    .card {
        width: 44%; } }
@media screen and (min-width: 76.38em) {
    .card {
        width: 29%; } }
@media screen and (min-width: 99.4em) {
    .card {
        width: 30%; }
```

I've determined which breakpoints were best for placing the project cards side by side. I've also made the featured images centre automatically.

CREATING THE SINGLE PAGE

Now we'll create the single template for portfolio projects. Whenever a user visits a single project's page, this is what will display. Create a new file in your theme, call it `single-portfolio.php` and copy another template to paste in there (I'd recommend whatever you used for `archive-portfolio.php`). This time we will be replacing the Loop with this code:

```
<?php while (have_posts()) : the_post(); ?>
    <h2><?php the_title(); ?></h2>
        <?php the_post_thumbnail('jlc_project'); ?>
        <?php the_content('Read the rest of this entry'); ?>
        <?php
```

Single project The single project view. The project reduces in width as the page grows to keep the text easy to read

```
            if(function_exists('jlc_projects_get_link')){
                $jlc_link= jlc_projects_get_link($post->ID);
                if($jlc_link){
        ?>
                <a href="<?php print $jlc_link; ?>"
                class="button">Visit the Site</a>
        <?php
                }
            } ?>
<?php endwhile; ?>
```

This looks similar to the archive template, but we do call an extra function here: `jlc_projects_get_link()`. We will add this to our plugin, and it will return the URL for the current project. In the event there is no URL, false should be returned and no button is displayed. Here's what the function (located in `jlc-projects.php`) looks like:

```
function jlc_projects_get_link($id){
    $url= get_post_custom_values('jlc_projects_link', $pid);
    return ($url[0] != '') ? $url[0] : false;
}
```

The CSS for this page will depend largely on the theme: I've used some CSS to generate a nice button. Make sure that whatever CSS you create yourself does not interfere with the main theme.

IN CONCLUSION

By now, we've created a plugin to add a new Custom Post Type for portfolios, integrated picturefill.js to handle images better, and created two theme templates to display the information.

The GitHub repo for the tutorial contains all of the code shown here, plus the theme I used, a shortcode and a template tag.

VIDEO

Watch an exclusive screencast of this tutorial created by the author: *netm.ag/video-rwd-254*

ABOUT THE AUTHOR

KEVIN FOLEY

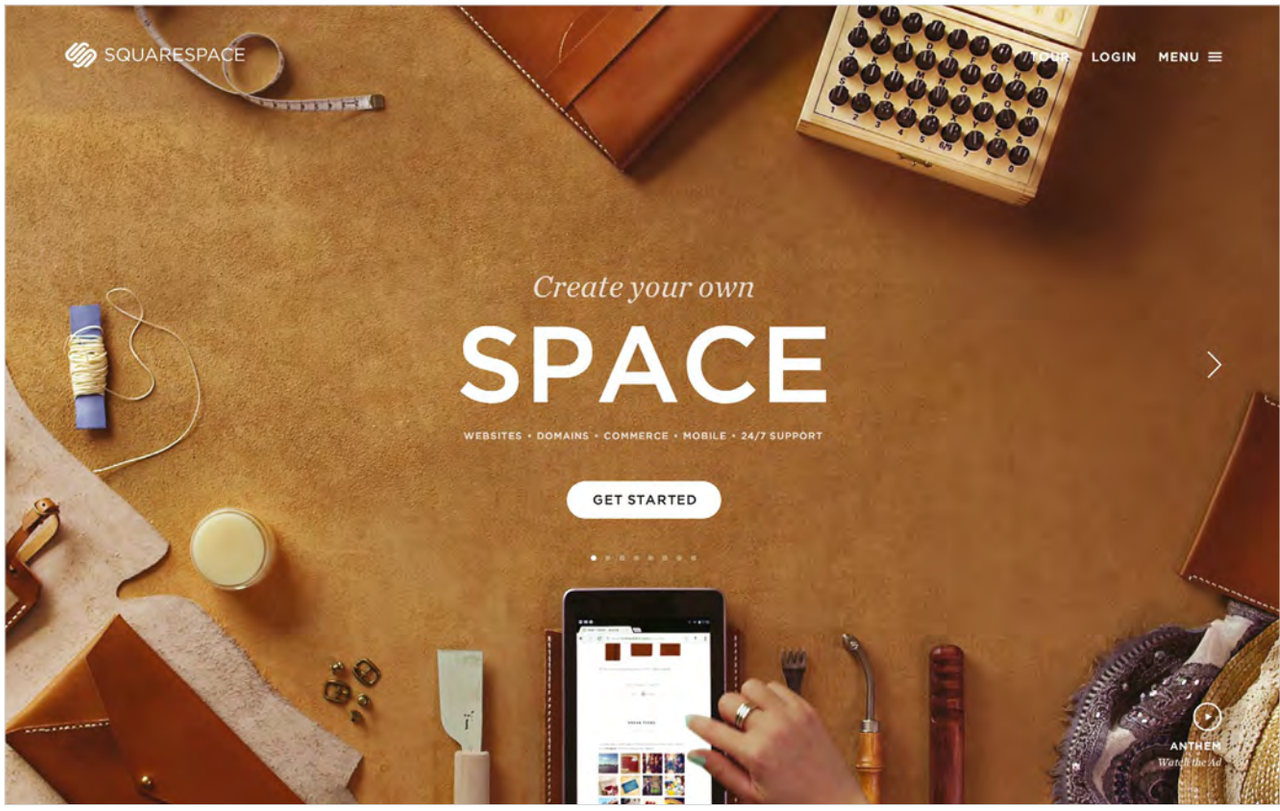
w: *foleyatwork.com*

t: @foleyatwork

areas of expertise:
frontend design and development

q: what's the oddest thing you've ever seen on a designer's desk?

a: A complete set of Gwar action figures



SQUARESPACE

USE SQUARESPACE'S DEVELOPER PLATFORM

Kevin Foley, Squarespace's developer evangelist, introduces the Squarespace Developer Platform to help create dynamic websites

If you've heard of Squarespace (*squarespace.com*), you probably know it as a simple website builder for non-technical folks, and that's true. But there's another side to Squarespace. About a year ago the company launched its Developer Platform (*developers.squarespace.com*), which allows frontend developers and designers to take control over the frontend code that's behind their Squarespace websites.

This article will outline some of the basics of starting a Squarespace site and creating templates. In the process, we'll cover what the platform does, why someone would want to use it, the basics of the template language and some of the advanced features included in the platform.

WHAT IS IT?

The Squarespace Developer Platform is a hosted environment to create dynamic websites. The infrastructure, including hosting, caching, version control and performance enhancements, is managed by Squarespace. This allows website creators to focus more on functionality and design rather than maintainability, uptime and performance. While it may be a deal-breaker for open-source zealots, Squarespace isn't open source or self-hosted like

RESOURCE

SQUARESPACE ANSWERS

If you hit a development roadblock, visit Squarespace Answers. Filter the questions by the developer tag to view more relevant info: *answers.squarespace.com/tags/developers*

many other content management systems (CMSs). Controlling the entire stack allows the company to tightly integrate the backend and frontend. This is evident in the responsive and Retina-ready image loader, solid cost-effective hosting and public data API, just to name a few features.

GETTING STARTED

The platform's ease of use is evident when starting a site. Go to *developers.squarespace.com*, fill out a simple short form found in the footer and you're ready to start building. The signup form will create a Squarespace site and, if you don't already have a Squarespace account, it will create an account for you. Developer sites are free as long as they're in development, and you don't have to use a credit card to signup.

DOWNLOADING TEMPLATE FILES

Once the signup process is complete you'll find yourself in the Squarespace CMS. Navigate to the `Developer` section of the CMS and you'll find information for connecting to Squarespace's servers

The Squarespace Developer Platform is a hosted environment to create dynamic sites

via SFTP or Git. To connect with SFTP, match up the information from Squarespace with the fields in the FTP editor of your choice. Each FTP client is a little different but they all require four essential fields: hostname, port, username and password.

Each template is a Git repository as well. By using version control system Git, downloading your files is a little faster and also can be done with one line in your console:

```
git clone: your-site.dev.squarespace.com/template. git your-site
```

THE DEVELOPMENT ENVIRONMENT

Now that you've downloaded your files it's time to take a look under the hood. A Squarespace workflow is similar to most other platforms: open up files in any text editor, edit, save and refresh. There are some plugins that make this process a little faster and easier.

Overleaf is the toolkit Squarespace's internal developers typically use when developing their website templates:

▶

IN-DEPTH

OUR INTERNAL WORKFLOW

When you're working with multiple developers, version control is a must. Every Squarespace template is also a Git repository, so you have version control capabilities from the very beginning. Of course, making a Git push for every change can be inconvenient. And a purely Git workflow leaves you with no reliable way to preview changes before they go out. Luckily this is an easy problem to fix on the Squarespace Developer Platform.

Here's a short guide to setting up a workflow on Squarespace:

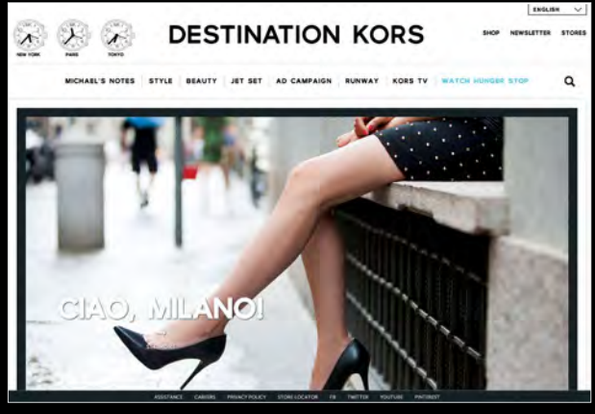
1. Start a production (prod) site and a development (dev) site (one for each developer) on Squarespace. All Developer Platform sites are free while they're in development. The dev site will be your test environment and the prod site will be your live site.

2. Clone your prod Git repo. We'll be working from a single code base so this will be the only code you need to download.

3. Push changes to dev via SFTP. You can do this either with a FTP Client or text editor. Internally, we use Sublime Text 2 with the Sublime SFTP plugin. You could also use Coda or Coda 2 to make SFTP changes.

4. Git pushes your changes to prod.

Each developer has their own fully-featured dev environment, and everyone is pushing changes to the same codebase. This workflow makes rebasing and sorting out merge conflicts very simple and will work well for any small to medium-sized team.



Destination Kors This was one of the first sites built using the Developer Platform

FOCUS ON

VIEW SOURCE FOR YOUR DATA

You should be able to use your data in any way you'd like. Your data on the Squarespace Developer Platform is available in many formats to make developing easier. For instance, when making an Ajax request you can add `?format=json` to any URL and get a minified JSON object with all the data available on that page. Or, if you'd like to make things even easier just add `?format=main-content` and you'll get a HTML return of the main content on any page. Exposing the API publicly makes it simple and easy to use. Here is a list of some of the data formats you can use on any Squarespace site:

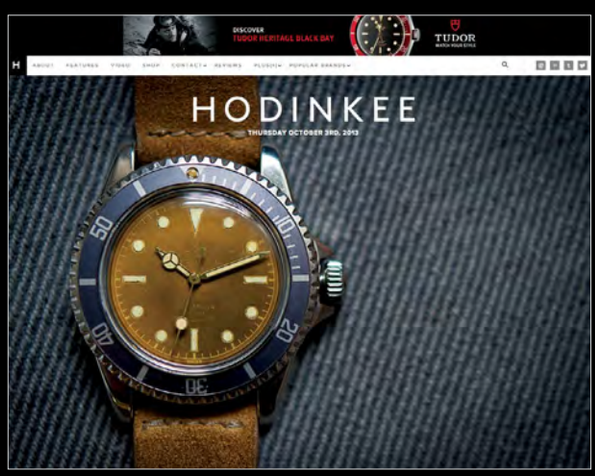
`?format=json` (minified) – This format is useful when making Ajax requests in JavaScript. It's a minified object that you can request and manipulate any way you'd like.

`?format=json-pretty` (human readable) – This is useful for when you're developing your site. JSON is a very human readable format, so you can reference what data is available on any given page.

`?format=rss` – An RSS feed on every page of your site.

`?format=main-content` – Very useful for Ajax requests. Rather than requesting JSON data and writing a HTML string in JavaScript, this returns parsed HTML ready to be inserted onto any page.

To see how each action works, just go to any Squarespace site – like *www.squarespace.com*, for example – and you can see what each one does.



Time piece Online wristwatch magazine HODINKEE uses Squarespace for its blog

A useful toolkit:
Sublime Text 2
Sublime SFTP A plugin that autosaves your changes to the server.
Gitbox A simple GUI for working with Git.

A combination of SFTP for quick changes and Git to manage your production codebase is usually best, especially for large projects.

RIGID STRUCTURE

Each template consists of a rigid structure of template and configuration files that make up your template, along with a folder structure to organise the files. As follows:

- **/assets/** Contains design assets: images, fonts and icons
- **/blocks/** Contains block files
- **/collections/** Contains collection files
- **/pages/** Contains static page files
- **/scripts/** Contains JavaScript
- **/styles/** Contains stylesheets
- **/** Sitewide templates and configuration

Some file types (like `.css` or `.js`) are self-explanatory, while others may not look familiar. For example:

This template code interfaces closely with Squarespace's public JSON API

conf – Configuration files define the various configuration options for different collections.
region – Found in the root folder, these are the scaffolding for your site, like WordPress's `index.php`.
.block – Block files are found in the blocks folder and are reusable bits of code that can be injected into any part of a template. For instance, a navigation is a block file.
.list – List files found in the collections define the layout for the list view of collections, like an array of blog posts, for example.
.item – Item files found in the collections folder define the layout for a single content item.
.page – Page files are found in the pages folder and are static pages, uneditable in the CMS.
.less – Found in the styles folder and preprocessed on the server for better CSS authoring workflow, you can include `.less` files in your template.

When building a Squarespace site you'll run into languages that are familiar to frontend developers, such as HTML, CSS and JavaScript. The template language, JSON Template (find more info here: *jsont.squarespace.com*), is a bit more obscure, but it's simple and easy to pick up.

TEMPLATING

Take, for instance, a simple code snippet that would parse on the server and output the title and tagline of a website:

```
{.section website}
        <h1>
                <a href="/">
                        {siteTitle}
                </a>
        </h1>
        {.section siteTagLine}<p>{@}</p>{.end}
{.end}
```

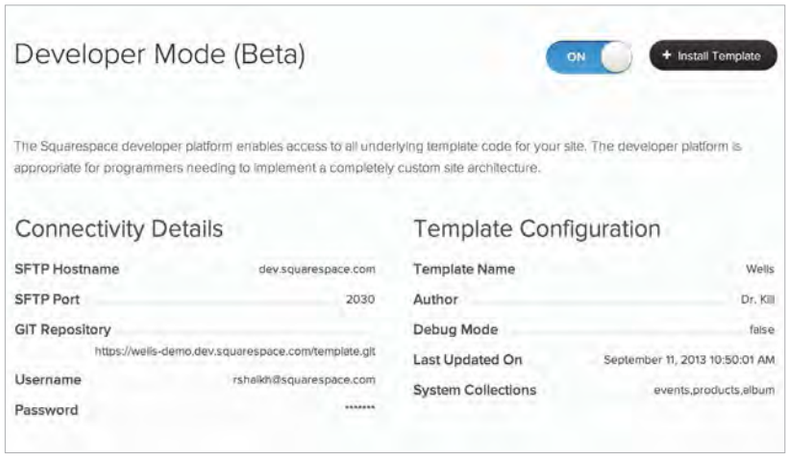
This template code interfaces closely with Squarespace's public JSON API. Append `?format=json` (`?format=json-pretty` for the human readable version) to any page on any Squarespace site and you'll see a large JSON object with all the data available to the template language. As web designer Jeffrey Zeldman said on his Big Web Show podcast, "It's like view source for your data".

A typical workflow would include keeping the JSON data for a page open in your browser and writing your template code according the structure and hierarchy of the JSON data. Let's look at a simplified version of what a JSON object may look like and compare how it interfaces with the template language.

JSON data:

```
website: {
        siteTitle: "Squarespace",
        siteTagLine: "Everything you need to create an
        exceptional website"
}
```

Welding power This photography studio uses Squarespace to power its site



Developer mode The SFTP and Git credentials are easy to locate in the developer tab in the Squarespace CMS

Template:

```
{.section website}
        <h1><a href="/">{siteTitle}</a></h1>
        {.section siteTagLine}<p>{@}</p>{.end}
{.end}
```

The first line in the template code defines the scope inside the website object. The template language has access to all the key and value pairs inside that object, so when we write `{siteTitle}`, the template language outputs `"Squarespace"`.

The next line is structured a little differently. At first this seems odd, but it's for a good reason. A website always has a title, but it may not have a tagline. The `{.section siteTagLine}{.end}` portion of the code is like a `truthy` statement in JavaScript; it checks if the key and value pair exists and, if so, it outputs that value.

In this case, our JSON data does include a value, so the output would be:

```
<p>Everything you need to create an exceptional
website</p>
```

If the website didn't have a tagline, there would be no `siteTagLine` key and value pair and the template code wouldn't return anything.

THE CONTROLLER

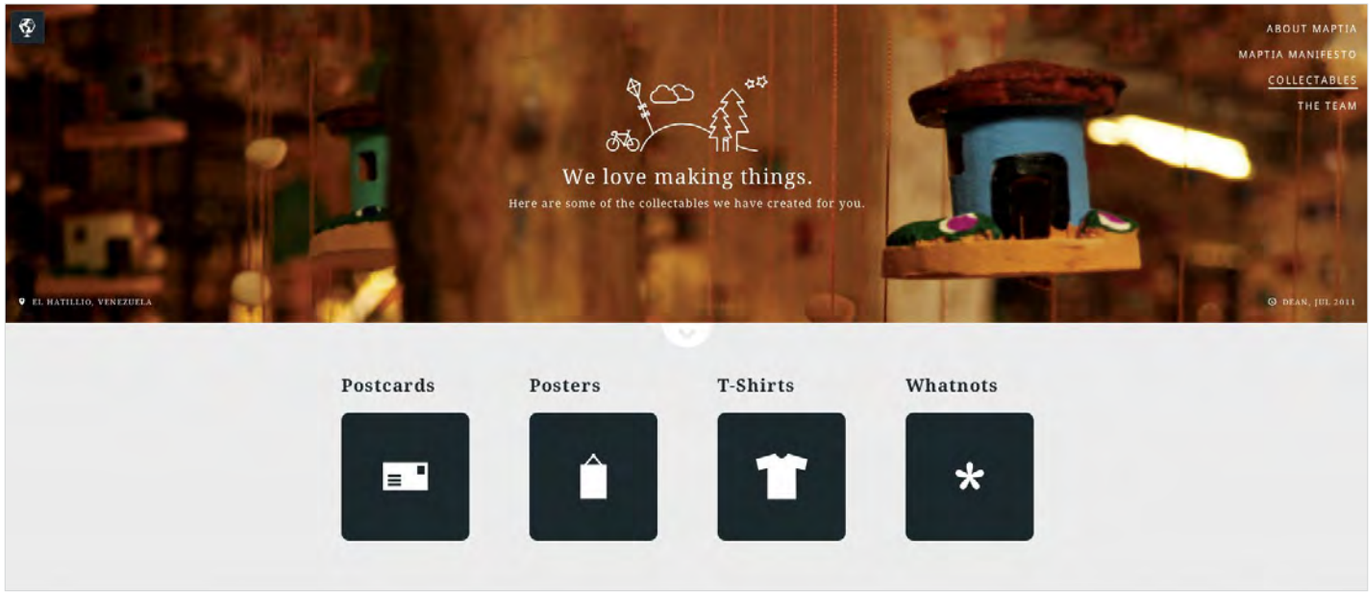
If Squarespace were a model-view-controller (MVC) framework, the `template.conf` file would be the controller. This file is important because it helps you extract bits of code (mainly `.region` and `.css` and `.less` files) from throughout your template to put together coherent layouts and stylesheets that can be used by the person or the team who will be managing content on the website. Turn overleaf for a sample `template.conf` file.

▶

RESOURCE

JSON TEMPLATE

Visit Squarespace's site for more information about using JSON Template, from data tags to thumbnail images: *jsont.squarespace.com*



Starting up Startup company Maptia (*maptia.com*) has a mission to build the most inspirational map in the world. Using Squarespace for its blog, its platform helps users tell stories about places

Here is a sample `template.conf` file:

```
{
"name" : "NET Example",
"author" : "Kevin Foley",
"layouts" : {
        "default" : {
          "name" : "Default",
          "regions" : [ "header", "main", "footer" ]
        }
},
"navigations" : [ {
        "title" : "Main Navigation",
        "name" : "mainNav"
} ],
"stylesheets" : [ "theme.less", "layout.less", "module.less",
"typography.less" ]
}
```

CHECKS AND BALANCES

There are a few additional important things to note about this file:

1. To check for mistakes in the syntax, just run it through a JSON validator.

2. The CSS is loaded here rather than in the template. We do this so we can compile all your stylesheets into one file (and therefore one HTTP request) for better performance.

Let's look at the process of adding a layout a little more closely:

```
"layouts" : {
        "default" : {
                "name" : "Default",
                "regions" : [ "header", "main", "footer" ]
        }
}
```

This code is telling the CMS to create a layout option called `"Default"` that consists of `header.region`, `main.region` and `footer.region`. Those region files will be compiled in the order they appear in that array. You can add multiple layout options by adding another object to that code. Let's say we made a region file with a sidebar and we'd like to give content editors the option to add a sidebar to any page, like so:

```
"layouts" : {
        "default" : {
                "name" : "Default",
                "regions" : [ "header", "main", "footer" ]
        },
        "sidebar" : {
                "Name" : "Sidebar",
                "regions" : [ "header", "main-sidebar",
                "footer" ]
        }
}
```

BUT WAIT, THERE'S MORE!

Squarespace is a powerful platform for creating beautiful websites. Visit *developers.squarespace.com* for more information and to sign up for a free developer sandbox site.

SIGN UP

FREE SANDBOX DEVELOPER SITE

To get your free developer sandbox site, and for more information, visit *developers.squarespace.com*

*WORDPRESS

WORDPRESS EVOLUTION

Marko Heijnen reflects on the 4.0 update and considers the future for WordPress

WordPress recently launched a new version, 4.0, amid much excitement around the release's updates and how they would benefit end-users and developers. The release boasts over 1,200 updates since 3.9. Although no groundbreaking new additions were included, 4.0 leads the way for future releases, with a number of underlying changes and clean-ups making WordPress 4.0 a more stable CMS.

NEW ADDITIONS

My personal contributions to the release of WordPress 4.0 included one patch. I initially co-created the `WP_Image_Editor` class with Mike Schroder in WordPress 3.5. With ongoing improvements, the update for WordPress 4.0 adds the method `get_quality()`. An important class that handles all image manipulation within the platform, the new method will help developers retrieve the current quality of an image when using the class directly in their own projects.

Whilst a number of incremental changes are present within the frontend and backend, the WordPress 4.0 release additionally included code cleanups and an application of standards for increased code quality. For example, almost all uses of the PHP function `extract()` were removed. The function previously exploded an array into variables, which although handy to use, accidentally overwrote variables without users knowing. Users were previously unsure as to what was being parsed by `extract()`. The removal of the function increases the stability of WordPress as a package, as no variables have the possibility of being overwritten.

> 4.0 leads the way for future releases, with underlying changes and clean-ups making WordPress 4.0 a more stable CMS

Additionally, Scrutinizer, a code inspection platform, was used to detect dead code and unused variables by searching through the codebase to find possible issues. Although not necessarily fixing every issue, the access modifiers gave a good indication as to the current status of the code, increasing the overall quality.

The code cleanups and removal of dead code within update 4.0 are a huge benefit to developers, making it easier to contribute to the ongoing improvement of WordPress as a CMS platform.

With `textarea` support for Customizer, less code is required for developers to test themes. When installing plugins on a dev version of WordPress, a new tab is visible. This saves developers time when completing updates as it displays all features being worked on as plugins.

THE FUTURE

The most notable change in this release is the creation of updates we can build on. But what do I think is the next big step for WordPress? The incorporation of a RESTful API.

More complex websites and mobile apps are continually being built with WordPress, so it is important for WordPress to have a RESTful API. Such an inclusion could support current WordPress apps as well as mobile apps that use WordPress as a backend. As a feature available as a plugin now, users are currently able to test this service.

Going forward, the platform will continue to update its legacy code and add the APIs that developers need to better accommodate these use-cases as their popularity increases. Although a lot of work is still needed around this, I am hopeful that we will see this update in WordPress 4.1 or 4.2.

Secondly, I expect to see the rebuild of the image editor in WordPress. A new project in which I am involved, the current version displays a number of issues including extendibility. Although the status of the build and release date are currently undetermined, I see a huge potential here for a more extendable UI for plugin developers to create new features for users.

There are also plans to extend the class `WP_Image_Editor` with new methods. Hopefully these extensions add the default inclusion of image filters (such as greyscale or sepia). As a personal goal for this project, I will begin working on new ways to show images on mobile.

*PROFILE

Marko (*markoheijnen.com*) is an avid WordPress enthusiast, and has been making contributions since 2010. As well as working full time for 1&1, he is the main core developer for GlotPress

Download the files here!

All the files you need for this tutorial can be found at *netm.ag/ghost-252*

ABOUT THE AUTHOR

STEVEN WU

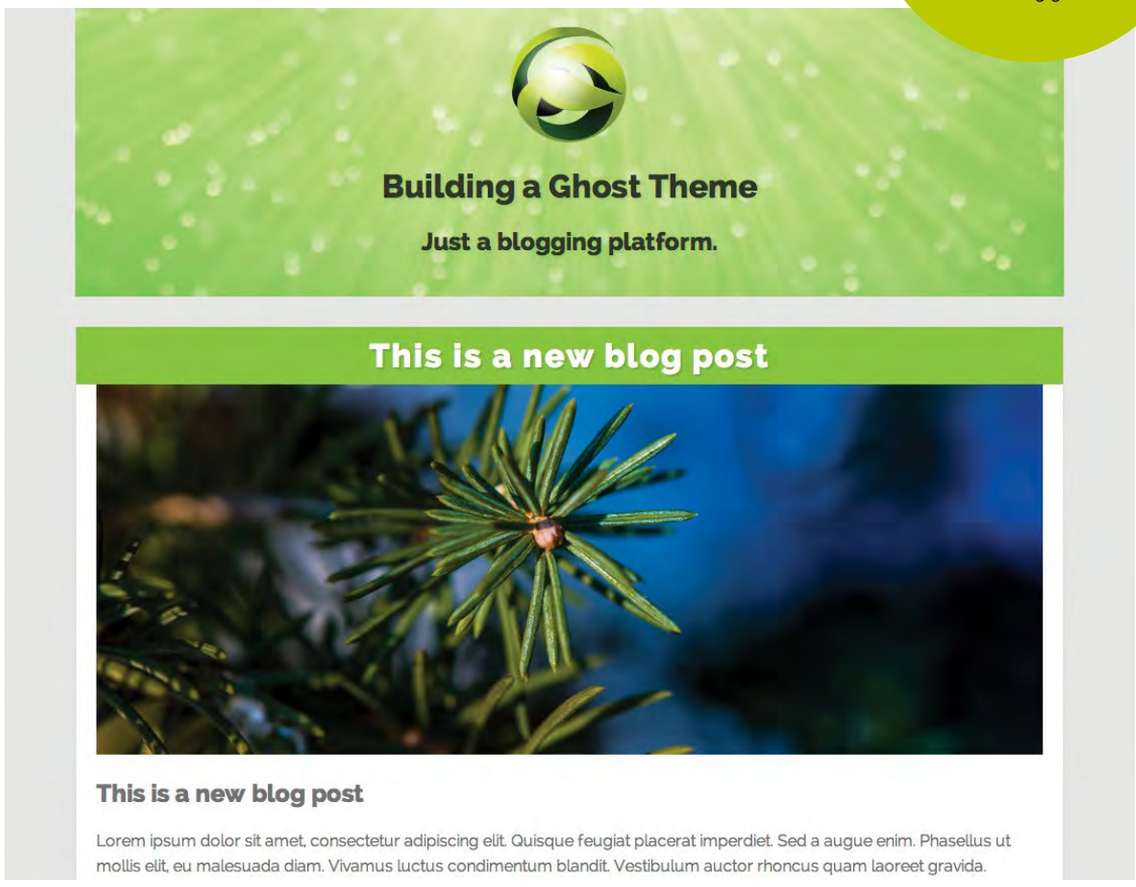
w: *designtodevelop.com*

t: @designtodevelop

areas of expertise: Magento and WordPress, HTML, CSS, JavaScript, PHP

q: what's your favourite smell?

a: The smell of success (technically odourless)



CMS

BUILD A GHOST THEME

Steven Wu explains how to get started and build your own theme for the popular new open source blogging platform Ghost

Ghost is a new and free open source blogging platform. Successfully funded through Kickstarter, it surpassed its original request of only £25,000 achieving over £196,000 in funding from the community in May 2013. Started by John O'Nolan, Ghost has a unique purpose in providing bloggers with a simple interface that allows them to write and publish their content without the hassle or distraction by the sheer complexity of development from traditional platforms.

Ghost has been beautifully designed from the ground up. Its clean and simplified UI allows you to quickly browse through the archive so you may spend less time managing your blog and more time blogging. It has a smart writing screen using Markdown with a real-time preview on the right-hand screen and simple drag and drop functionality to add images into place.

Ghost has three main core principles: it's developed for users rather than developers like many blogging and CMS platforms out there; the platform has a MIT licence so you can do what you like with this platform with few limitations; finally it's made for love. Ghost is a non-profit organisation; its motivations are to support bloggers rather than satisfying investors. In this tutorial I will guide you how to install and set up Ghost locally and build your first Ghost theme.

To begin building our Ghost theme, start within the Ghost installation folder (see opposite for the

REVIEW

JOHN O'NOLAN

With thanks to John O'Nolan for his peer review of this tutorial. John is the creator of Ghost.

installation guide). Under `Content > Themes`, create a new theme called `mytheme`. Make sure this is in lowercase without any spaces (hyphens are acceptable). This will be the directory that houses our theme codebase. Within this directory, create the following files and folders:

```
- /assets/
  — /css/
        — normalize.css
        — screen.css
  — /images/
  — /js/
  — /fonts/
- /partials/
        — header.hbs
- default.hbs
- index.hbs
- post.hbs
```

Both `index.hbs` and `post.hbs` are the only files required to be a valid theme. Without any of these you will receive an error.

ACTIVATE THE NEW THEME

Now in the Ghost dashboard, navigate to `Settings > General`. Under `Theme`, select the new theme you just created called `mytheme`. If it's missing, you'll need to go to the terminal and restart Ghost. Click `Save` to activate this theme. You won't see anything in the frontend yet. This is because we have yet to add any markup in our theme.

USING HANDLEBARS

Ghost makes use of a templating language called Handlebars.js (*handlebarsjs.com*). Predefined Handlebars expressions make it simple to build and maintain Ghost themes.

Handlebars separates the templates from the raw HTML for you. Bear in mind that, with Handlebars, you can't write functions or hold variables. Handlebars is developed simply to display content where the expressions are outputted.

Handlebars expressions are wrapped with curly brackets and look like this: `{{author.name}}`. This

Predefined Handlebars expressions make it simple to build and maintain Ghost themes

FOCUS ON

INSTALLING GHOST

Ghost is a lightweight web application. It only takes a matter of minutes to install this locally and get it up and running. Ghost is a JavaScript application built upon Node.js. It's required to have Node.js installed in order to run Ghost. If you haven't already, head over to *nodejs.org* and download it. Once you've got this installed, go to *ghost.org/download* to download the latest Ghost software and uncompress it.

To install Ghost in the terminal, run:

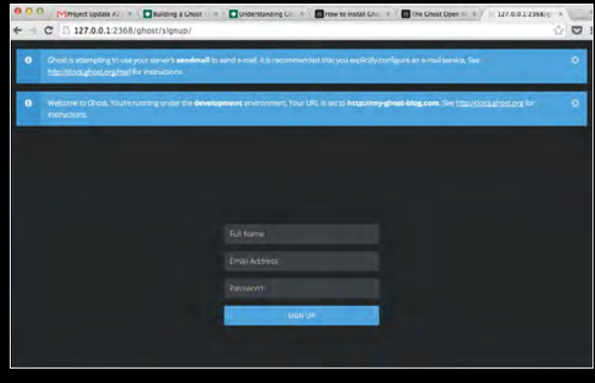
```
$ cd /path/to/downloads/ghost
$ npm install —production
```

Npm will now install all the necessary dependencies to the production and development environment. Once this is completed you can now start Ghost in development mode with:

```
$ npm start
```

In your web browser, navigate over to your newly installed Ghost blog at *http://127.0.0.1:2368*. You can register your administration login by going to *http://127.0.0.1:2368/ghost*. When you visit this URL, you'll notice Ghost notifies you of the send email and Ghost URL being used. Configure this by changing the base URL. Open up the `config.js` in your text editor and under the development URL (line 11) and production URL (line 47) change this with *http://127.0.0.1:2368*.

Once you've completed development, you can shut down Node.js, by pressing `Ctrl + C` in the terminal.



Dashboard access Two notifications require you to change the base URL

basically looks up the `author.name` property and outputs it.

DEFAULT.HBS

Let's get our hands dirty and start creating our theme. Open up the `default.hbs` in your favourite text editor. This is the base template and includes all the basic `<html>`, `<head>`, `<body>` tags used throughout your website.

In this template we input our HTML doctype, basic meta tags and head and body tags. (See `default.hbs` in tutorial files.) You'll notice expression tags: `{{! Responsive Meta Tags }}`. Any expressions proceeded with an exclamation point within the curly brackets are comments and will not be printed in the final source code.

The `{{ghost_head}}` helper is used to output any system scripts, styles and meta tags. The `{{ghost_foot}}` helper is used to output scripts at the bottom of the document. The Handlebars expression `{{{body}}}` is an important one. This is where all your content will be displayed, which extends the default template. The following `{{body_class}}` is used to automatically generate CSS class names for targeting specific pages:

```
<body class="{{body_class}}">
    <div class="mytheme_page">
        {{{body}}}
    </div>
{{ghost_foot}}
</body>
```

Index.hbs

Below We separate our header as a partial template to better manage and reuse code throughout our theme

Bottom You can use the {{date}} helper to output the blog published date and use the format option to control the date format

Now in our `index.hbs` (see this source code in the project files), we use the Handlebars expression `{{!< default}}` at the very head of this document to reference to our previous base template. This template will be used for our homepage. We will want to style each blog post within the foreach helper. By using the opening `{{#foreach posts}}` and closing `{{/foreach}}` loop, anything inside this will display each post with the markup.

To display the content of each blog post we use a Handlebars expressions `{{content}}`. We can also limit the word count by using the parameter `words="100"`.

CLASS AND ID NAMING CONVENTIONS

You'll notice all of the class names are proceeded with `mytheme_`. This is a recommended practice when building a Ghost theme. Ghost will automatically assign particular class names and more specifically IDs to certain elements in your theme. You'll want to prevent clashes and consider your scope of class names.

PARTIALS

Typically we can insert our header markup just below the `{{!< default}}`, but the advantage of Handlebars templates are hierarchical support, whereby one template can extend another. This includes the use of partials. This helps to eliminate repetition of code and encourage reusability. We can separate our header into a partial template.

Ghost will automatically assign particular class names to certain elements in your theme

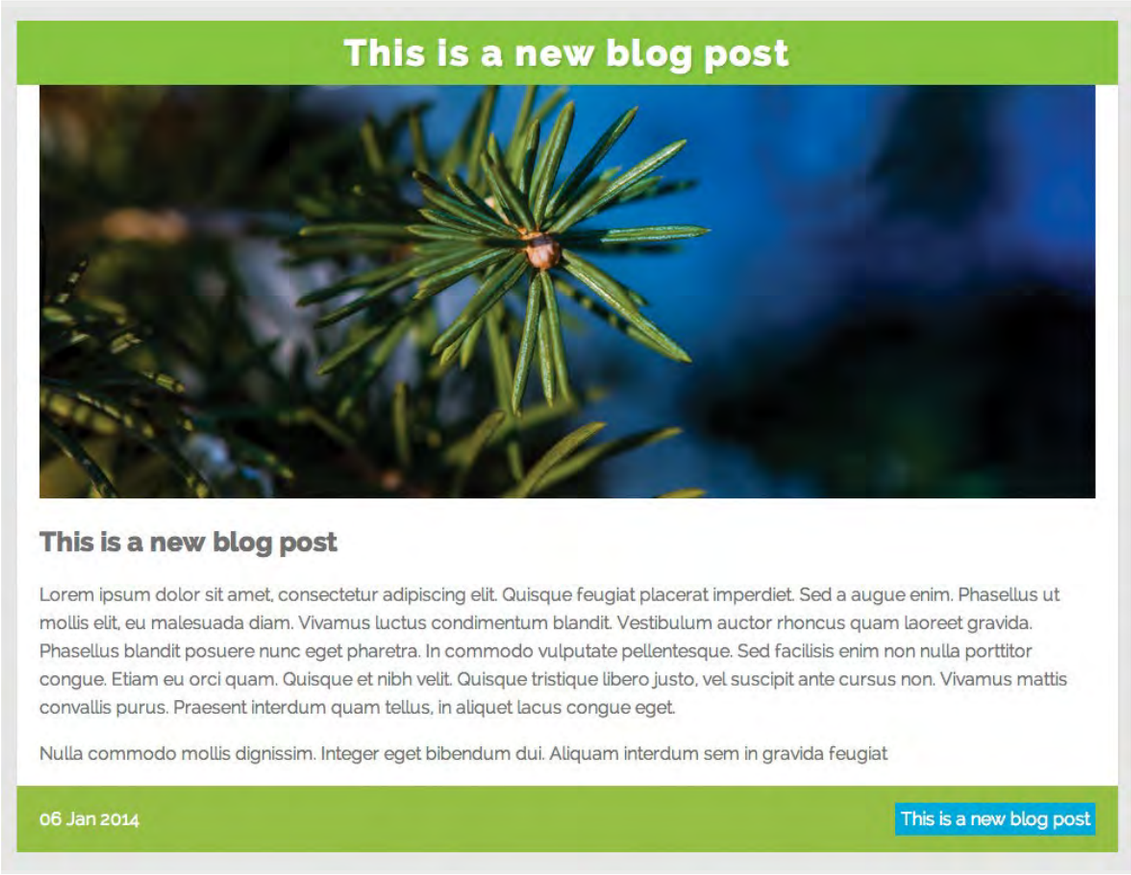
Within the partials directory open up the `header.hbs`. In Ghost dashboard `Settings`, you can upload your own blog logo and blog cover image. We'll use an `if` statement to check whether a blog cover image exists. If so, we'll output it as a background image:

```
{{#if @blog.cover}}style="background-image: url({{@blog.cover}})"{{/if}}
```

This time, we'll check if a blog logo is available.

```
{{#if @blog.logo}}
        <a class="blog-logo" href="{{@blog.url}}">
            <img src="{{@blog.logo}}" alt="Blog Logo" />
        </a>
{{/if}}
```

The `@blog` global data accessor has access to global settings in Ghost we can output in our theme. Now let's dive into the fun part of styling our theme.



Homepage help Using the foreach helper, index.hbs handles the post's object to output each blog post on the homepage

In our theme, we've linked to `normalize.css` for our HTML5-ready CSS reset. Within the `screen.css` is where we'll input all our custom theme styles. We'll then add some global styles, followed by styling our header and set a `max-width` to prevent our layout from expanding over certain pixel size:

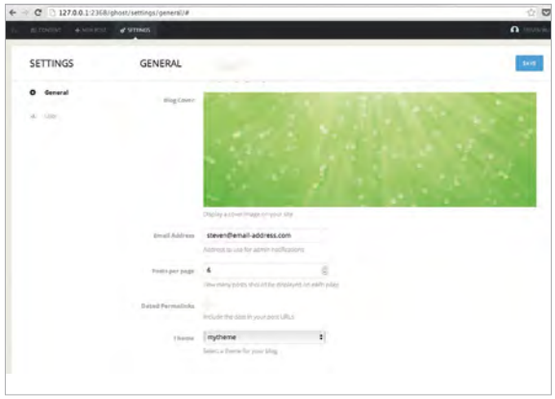
```
.mytheme_page {
    max-width: 980px;
    margin: 0 auto;
}
.mytheme_header {
    padding:20px 0;
    text-shadow: 2px 2px 2px rgba(26, 26, 26, 0.1);
    text-align: center;
    color: #2f3727;
}
```

Now style each blog post within its article container:

```
main article {
    margin: 30px 0;
    borderz-left:1px solid #DBDBDB;
    border-right:1px solid #DBDBDB;
    background-color: #FFFFFF;
}
.mytheme_post_content {
    padding: 0 20px;
}
.mytheme_post_title {
    margin: 0 0 20px 0;
    padding: 10px;
    font-size: 2em;
    letter-spacing: 2px;
    text-align: center;
    text-shadow: 2px 2px 2px rgba(26, 26, 26, 0.2);
    color: #FFFFFF;
    background-color: #8ACD36;
}
.mytheme_post_title a {
    text-decoration: none;
    color: #FFFFFF;
}
.mytheme_main_img img {
    width: 100%;
    max-width: 100%;
    border: 0;
}
```

VIDEO

Watch an exclusive screencast of this tutorial created by the author: *netm.ag/tut4-252*



Theme activation Select the newly-created theme in the dashboard to activate it. You may need to restart Ghost in the terminal to pick up the theme

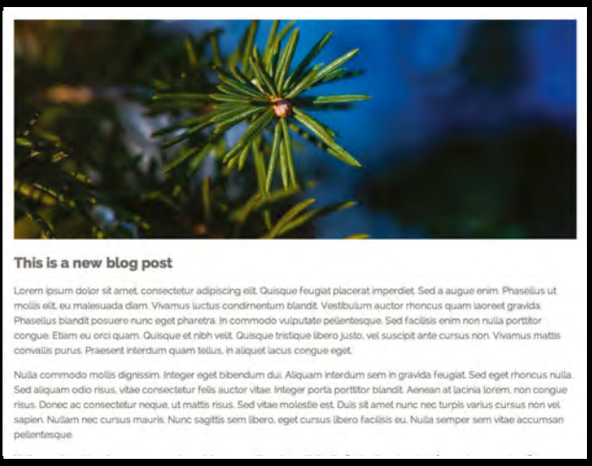
▶ Place the date on the left-hand side and the `Read more` button opposite. Give this link the presentation of a button:

```
.mytheme_post_info {
    overflow: auto;
    padding: 0 20px;
    background-color: #98C148;
}
.mytheme_date {
    float: left;
    padding-top: 20px;
    color: #FFFFFF;
}
.button {
    float: right;
    padding: 20px 0;
}
.button a {
    padding: 5px;
    transition: ease .3s;
    text-decoration: none;
    color: #FFFFFF;
    background-color: #39A9DA;
}
.button a:hover {
    background-color: #199ED9;
}
```

We touched upon the key aspects of developing a Ghost theme. Hopefully this insight has opened up possibilities to creating your own personalised theme. You'll see that building a Ghost theme is very simple to pick up with the pre-defined Handlebars expressions. Don't forget to download the tutorial's accompanying files: *netm.ag/ghost-252*. Also check out the Ghost documentation for more information to help you build and customise your own theme by visiting: *docs.ghost.org/themes*.

IN-DEPTH

PREPARING POST.HBS



Simple expressions We simply use `{{#post}}` and `{{/post}}` expressions to output the content on the single post template

Having only set up the homepage template, the `post.hbs` controls the display of each single blog post page. Again on this template (see `post.hbs` in the project files. Download them: *netm.ag/ghost-252*) we use the same `{{!< default}}` and `{{> header}}` Handlebars expressions. This time we use the opening `{{#post}}` and `{{/post}}` expressions to display a single blog post. While in the single post context we have access to the author data including the author name and bio.

We can display the author details by simply adding the code displayed below:

```
<section class="author">
    <h3>Written By:</h3>
    <h4>{{author.name}}</h4>
    <p>{{author.bio}}</p>
</section>
```

We can now apply some CSS styling to our single blog post. It's recommended that you actually place all your single post styles in a separate CSS file.

It's important to do this is because, in the near future, Ghost will be unavailing a new split screen feature, which will load the custom theme styles in the admin.

Post display Post has access to author details, outputted on the blog post page

Q&A

HANNAH WOLFE

Ghost's lead developer chats to **Martin Cooper** about Node.js, embracing new languages and using her skills to terrorise her teachers

INFO

job: Chief technology officer

company: Ghost Foundation

w: *hannah.wf ghost.org*

t: @erisDS

net: Do you want to introduce yourself?
HW: I'm the co-founder and CTO of the Ghost Foundation (the non-profit company that is the caretaker of the Ghost software), and lead developer on the Ghost project itself. I spend my days on IRC and GitHub managing the creation of Ghost and the supporting infrastructure.

net: When did technology first appear in your life?
HW: It was the morning of my sixth birthday. I bounded down the stairs to find, on the dining room table, an Intel 486, with a 5 ½" floppy disc drive and a red ribbon tied around it. On the screen, in bright colours, rotated the words 'Happy Birthday Hannah'. I was spellbound. How did it know it was my birthday? Wait! How did it know my NAME?! I wanted to know how it worked, inside and out. Three years later, I built my grandparents their first computer.

Then, aged 11, I picked up a library book: *HTML in Easy Steps*. I read the book cover to cover, including the appendix on a new thing called CSS, went to my computer and coded my first web page. The next year I terrorised my teachers at school by handing in all my homework in the form of a website on floppy disk.

net: You describe yourself as a polyglot developer. Do you collect languages or learn them as required?
HW: It's about using the right tool for the job. Picking up new gadgets for my toolbox is fun, but I don't tend to have the time or motivation to really learn a new language until I have a reason to use it. I keep up-to-date on the latest tools and frameworks, but I'm not one of those developers who'll grab something new and code a random side project just to test it out – although I often wish I was.

net: The use of libraries always causes heated debate on @netmag. Some developers assert you should always learn the language first. What's your take?
HW: Learning is a very personal thing. I like jumping in at the deep end with a project: grabbing a new language along with a new framework isn't uncommon in that scenario. I picked up both Flask and Django whilst learning Python, and I'm not sure anyone really learns Ruby without also learning Rails. When it comes to JavaScript it varies based on what you're trying to achieve. I really don't think there's a wrong or a right way to learn.

net: Node.js is a comparatively young platform, yet you chose to build Ghost around it. Why?
HW: There are a number of technical reasons why Node.js was the right choice, not least its incredible speed and the benefit of being full-stack JavaScript. Specifically, we opted for something new and shiny because we felt it necessary in order to 'reboot' something as overdone as blogging. Node.js is aimed at the future of the web, and is particularly good for creating APIs for consumption by a rich client. With Node.js, both of these pieces are written in JavaScript, which is a big plus when you're open source.

net: What can we expect next from Ghost?
HW: We're wrapping up the conversion of our admin panel from Backbone.js to Ember.js. Our internal JSON Data API has been cleaned up and OAuth is being added so we can open it for external use. Later in the summer we'll be working on getting the first proper examples of Ghost Apps (plugins) out into the public.

net: You're a big open source fan. What's the best way to start contributing to the community?
HW: There are two key things to get to grips with: the code base you're contributing to, and the way that particular project works in terms of contributions. For the latter, the best thing is to find out where the majority of discussions happen – usually IRC or a mailing list. With Ghost, developers hang out in #ghost on freenode. For the former, nothing beats grabbing a small bug and fixing it.

ABOUT THE AUTHOR

DAVE MYBURGH

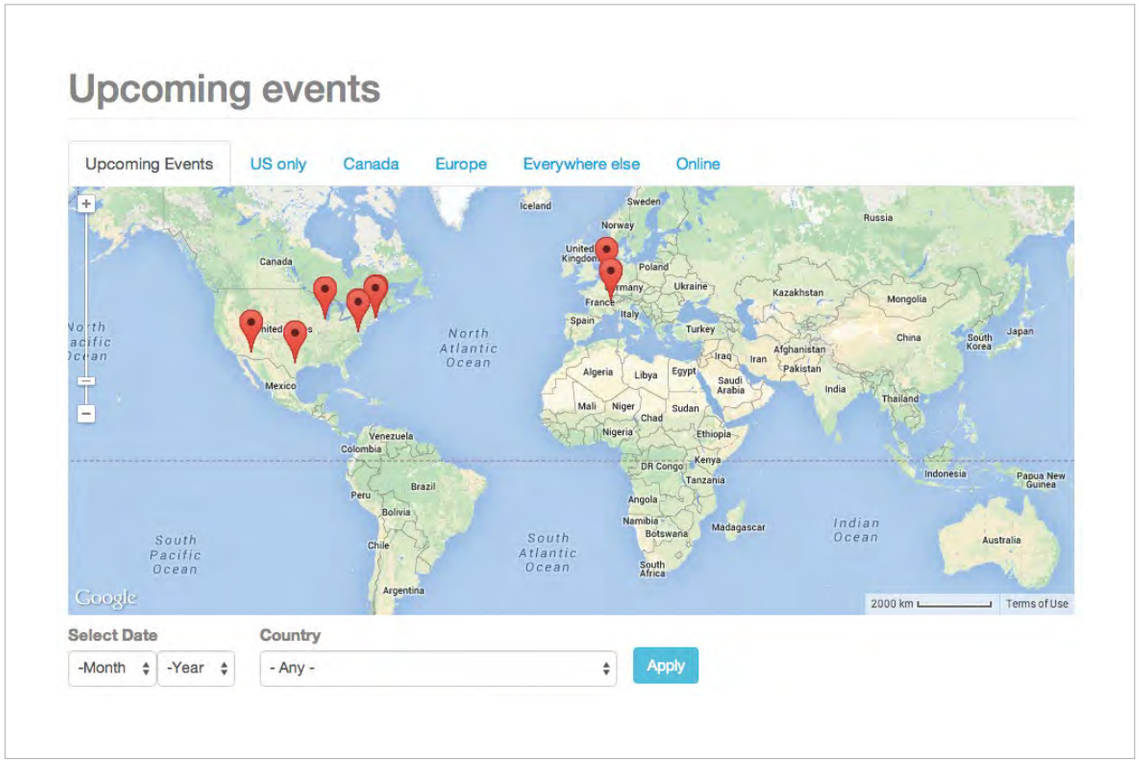
w: *acquia.com*

t: @davemybes

areas of expertise:
HTML, CSS, PHP, Drupal

q: what's your opinion on Marmite?

a: I love it! Although I do seem to be in the minority on this side of the pond. I'm just sad it's hard to get here in Montréal



DRUPAL

CREATE CUSTOM GOOGLE MAPS USING DRUPAL

Dave Myburgh shows you how to create a custom global event map from location data stored in Drupal

When we decided to update Acquia's training website to Drupal 7, we wanted to have a map on the event page showing the locations of all our upcoming courses, as well as a map for each course, so users can quickly get a feel of where events will be taking place. Visitors can also open up the maps in Google Maps to obtain directions.

In the past, it's been difficult to create maps from location data stored in Drupal. There are numerous modules and methods available to deal with this, but finding the right combination can be tricky. For the training site, we figured out how to make it all happen using the following three modules (along with any other modules they require):

- Location 7.x-3.2 (*drupal.org/project/location*)
- GMap 7.x-2.9 (*drupal.org/project/gmap*)
- IP Geolocation Views & Maps (IPGV&M) 7.x-1.25 (*drupal.org/project/ ip_geoloc*)

In this walkthrough, I'll show you how we used these modules to generate custom event maps. I'll begin by configuring each module, using IP Geolocation Views & Maps to collect location data from the site's visitors, then creating a global map that shows the location of upcoming events along with a pin indicating the location of the viewer.

You can take a look at the results by visiting *training.acquia.com/events*.

RESOURCE

Visit the Event category of the Download & Extend section of the Drupal site for more useful modules: *drupal.org/download*

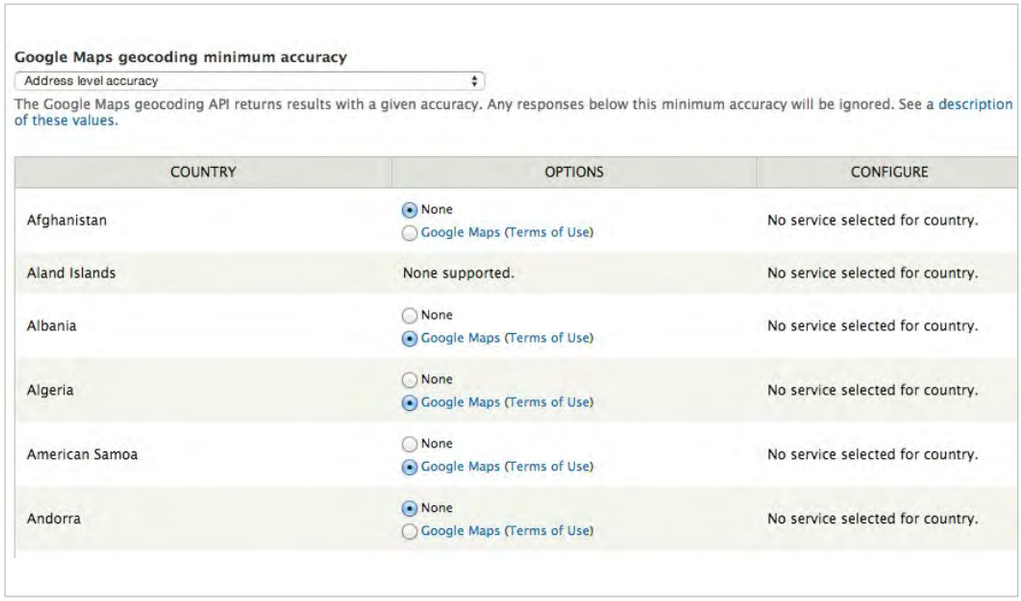
01 First, you need to enable the following modules: Location, Location CCK, GMap, GMap Location and IP Geolocation Views & Maps (IPGV&M). There will be some dependencies that are required – for example, Views – so go ahead and enable those too.

02 Go to `Admin > Config > Content > Location` and configure the Location module. Under `Main Settings`, we enabled the `Open map link in new window` option and left the `Map Links` tab alone. On the `Geocoding Options` tab, we selected `Address level accuracy` from the `Google Maps geocoding minimum accuracy` dropdown. We also ticked the `Google Maps` button for each country we run courses in. If you add an event in a new country, you must come back here and enable geocoding for that country.

Step 3 Working on the GMap module

03 To configure the GMap module, go to `Admin > Config > Services > Gmap`, where you'll enter your Google Maps API key. Enter the map defaults you want to use. We used a default width of 350px, a default height of 160px and no default centre, to fit in with our design for the event pages. The `Marker action` option can be set to `Open info window`, so that users get a pop-up when they click the marker.

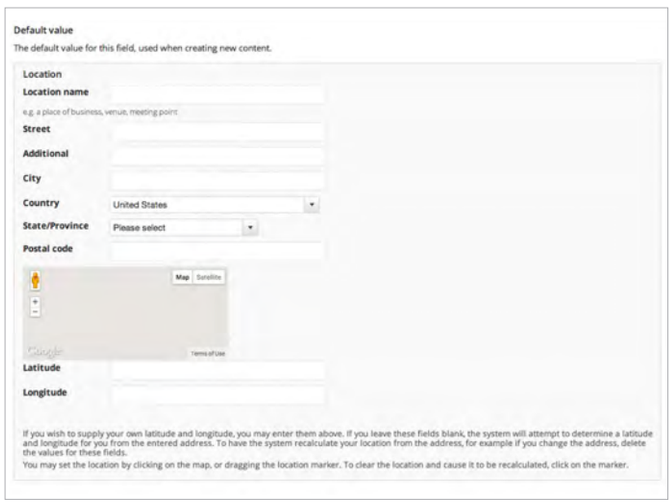
04 The IPGV&M module enables you to collect location information from your users. When they visit a page with a map on it, their location can be shown on that map and used to give directions to an event. Go to `Admin > Config > System > ip_geoloc` to configure it. We checked `Employ the Google Maps API...`, along with `Employ Smart IP...` as a backup. We also entered the URLs where we wanted to collect user location data: all the event nodes and the main events landing page. We left the other options on their defaults.



Step 2 When configuring the Location module, tick the Googe Maps button for each of the countries you're holding events in

05 We wanted to show a Google map of the event location on each event node. The Location module can be used to create a location field that takes an address and geocodes it to a latitude and longitude for display on the map. We didn't set any defaults in the `Default value` settings, to prevent empty maps appearing if no location data is entered – for example, for an online event.

06 In the `Collection settings`, we chose to `Allow` collection of the following elements for a location: `Location name`, `Street location`, `Additional`, `Country`, `City`, `State/Province` (choosing



Step 5 Leaving default values blank means a map won't appear if location data isn't entered

EXPERT TIP

OTHER MAPPING OPTIONS

The OpenLayers module can also be used for mapping in Drupal. It supports many different map types, including OpenStreetMap, Google and Bing. Along with the Address Field and Geocoder modules, OpenLayers is an alternative to using Location, GMap and IP Geolocation Views & Maps. More information can be found at *drupal.org/project/openlayers*.

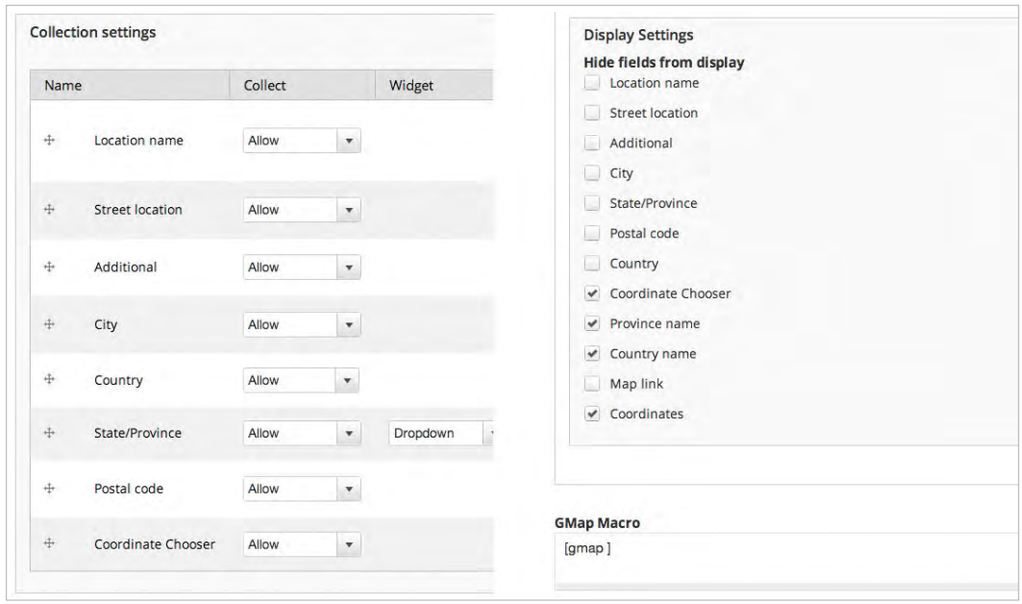
▶ Dropdown from the second menu), Postal code, and Coordinate Chooser (this allows the editor to enter a latitude and longitude if geocoding fails for some reason). In Display Settings, we ticked: Coordinate Chooser, Province name, Country name and Coordinates. The GMap Macro is simply set to [gmap] and our GMap marker is Blue.

07 On the Manage Display tab for the event content type, select Address with map as the Format for the Location field. Now when viewing an event with an address, you will see the actual address as well as a map of the location. A link to Google Maps is provided below the map so users can interact with it – for example, to get directions from their location.

08 If a map does not appear after saving an event, go back and edit the node to ensure that a latitude and longitude are now visible below the location data. If not, you might need to enable geocoding for that country, or manually enter the latitude and longitude. If that doesn't work, double-check all the settings above. As a last resort, check the issues list for that particular module on the Drupal site (*drupal.org/project/issues/gmap*) to see if you can get some help there.

09 The last thing we need to do is create a display that shows all our events across a world map.

Step 9 The Views module creates a display



Step 6 Under Collection settings, select Allow for the elements of information you wish to be collected for a location

We can do this using the Views module, in conjunction with the IPGV&M module. Views is one of those ubiquitous modules that tends to be used on all but the very simplest websites. It is required by IPGV&M, so you should already have it enabled at this point.

10 Create a new View (Admin > Structure > Views > Add) and check Create a block with the Display format set to Map (Google API, via IPGV&M). In the settings for this option, you'll see dropdown selects for the latitude and longitude fields. If you've already added those fields to your field list in the View, select them; otherwise leave them blank for now – you can add them after you've added the fields. Leave the Default location marker set to default.

11 Map options is where the tricky stuff comes in. You really have to play around with the options in order to get your desired outcome. We used the following:

```
{"mapTypeId":"roadmap",
"disableDefaultUI":true, "zoom":2,
"zoomControl":true, "scaleC
ontrol":true,"centerLat":20,
"centerLng":-30}
```

In Map style (CSS attributes) add CSS styles for the map itself: here, we're using height: 400px;width: 100%. Select Center the map on the visitor's current location from Map centering options and enter a hex colour code (without the #) for the user's icon colour.

12 Add Location: Latitude and Location: Longitude fields to the View. Be sure to set these fields to Exclude from display. Then add whatever other fields you want to show in the pop-up when someone clicks on the marker on the map: for example, Content: Title, Content: Date, Location: City and so on. We added a filter to the View to show only event nodes that had a start date in the future (Content: Date (start date) > now).

13 Save the View and add the block where you want it to show on your site. You can see our map at *training.acquia.com/events*.

RESOURCE

TRY THE GOOGLE MAPS JAVASCRIPT API

The Google Maps JavaScript API is an extremely powerful tool for creating interactive maps for your website. Luckily, Google provides excellent documentation to help you navigate its complexities. In many cases, the various Drupal modules that deal with the API allow you to use custom options – and the documentation is the best place to find and understand the numerous possibilities available. The IP Geolocation Views & Maps Map options, for example, can be found at *netm.ag/259-mapoptions*.

ABOUT THE AUTHOR

JOE MADDALONE

w: *joemaddalone.com*

t: @joemaddalone

areas of expertise:
JavaScript developer, instructor at egghead.io

HEAD TO HEAD

LEAFLET VS GOOGLE MAPS

Joe Maddalone pits Google's multimillion dollar-backed mapping service against its rather smaller, open source rival

LEAFLET	GOOGLE MAPS
Leaflet is an open source JavaScript library that provides mapping capabilities. It was developed by Vladimir Agafonkin and launched in 2010.	Google Maps is a web mapping service and technology. It was initially released in 2005 following Google's acquisition of Where 2 Technologies.
OVERVIEW	
Additional features can be added in by utilising plugins. Leaflet is usually used in conjunction with OpenStreetMaps, but can utilise other map data services, including Google Maps.	Google Maps requires very little knowledge to implement even some of its most complex features. However, end users have very little control over how the map data is presented.
THE REGULAR PERSON CASE	
Adding a map to a simple website is definitely a lot trickier for the average person using Leaflet. Many features will prove too much effort for those looking for a plug-and-play solution.	There is simply no easier way to add a fully featured map to a website, with directions, street view, distance data and more. There are some limitations in resolution and requests per day.
THE DEVELOPER CASE	
The level of control over Leaflet is unparalleled, with an ever-growing collection of plugins and constantly updated, developer-driven API. The ability to design every aspect of your map implementation without fear of licensing or unexpected data changes makes Leaflet the clear development choice.	Developing a consumer-facing application based on Google Maps would require a huge leap of faith that Google won't taint the experience with its own agenda. A dependency on Google's backing service may present itself one day as unwanted ads, reviews, +1s and other distracting data.
EVER ONWARDS	
Leaflet's development velocity is only matched by the pace of the open source projects it is partnered with and the number of third-party projects being built around the simple mapping library.	Street View and the acquisition of Skybox's real-time micro-satellite service imply a clear path to a staggering amount of data unlikely to be seen in any open source counterpart.

VERDICT

Need directions? Use Google Maps. Need to develop an application? Use Leaflet. The mere need to compare an application backed by billions of dollars to one where 90 per cent of the code is still contributed by a single developer from Kiev exposes the new ecosystem of web applications we have entered into recently. Removed from the OS and low-level languages like C/C++, open source applications like Leaflet are the proverbial 'barbarians' once at the gate of the commercial incumbents, now building superior castles right next door.

FACT FILE

STATS

As of June 2014:

- 2,037,496 sites using Google Maps
- 20,757 sites using Leaflet

CONTRIBUTE

Leaflet: *github.com/Leaflet/Leaflet*

Google Maps: You shall not pass!

INDEX

THE ULTIMATE GUIDE TO WEB DESIGN

VOLUME III

Design and build amazing sites with the latest tools, tips and techniques

This all-new guide includes everything you need to start designing and building amazing websites people will love. You'll find tutorials covering the very latest CSS, Sass and JavaScript techniques, articles on best practice responsive web design, tips on how to get the most out of WordPress and other CMSs, and much more!

It's the complete guide to becoming a better web designer and developer. Take your skills to the next level now – what are you waiting for?

Like this? Then you'll love...

Visit www.myfavouritemagazines.co.uk/design for more information